13th International Erlang/OTP User Conference

Stockholm, November 8, 2007

Proceedings

http://www.erlang.se/euc/07/

Erlang Training and Consulting

KREDITOR

tail-f

corelatus

QuviQ

synapse mobile networks s.a.

Sjöland&Thyselius

ERLANG

ERICSSON

Mobile Arts

Marcus Arendt Aktiebolag

Erlang/OTP User Conference 2007

Programme

08.30 Registration

Session I Chairman Ulf Wiger

09.00 Transport for London Journey Angel
Marcus Taylor and Vincenzo Nicosia

09.30 YXA Developments
Fredrik Thulin

10.00 Erlang Developments in LambdaStream
Samuel Rivas

10.30 Coffee

Session II Chairman Francesco Cesarini

11.00 Erlware for Managing Distribution and Build
Eric Merritt and Martin Logan

11.30 Quality Cruising -- Making Java Work for Erlang
Erik Stenman

12.00 Generic Syntactic Analyser: ParsErl
Anikó Nagyné Víg and Tamás Nagy

12.30 Lunch

Session III Chairman Mickaël Rémond

14.00 Developing RESTful Platforms With Erlang And Adobe Flex
Gordon Guthrie

14.30 Integrating OTP with Enterprise Service Bus
Leslaw Lopacki

15.00 Towards Hard Real-Time Erlang
Vincenzo Nicosia

15.30 Coffee

Session IV Chairman Claes Wikström

16.00 ProTest - An EU STREP project
John Hughes

16.15 Information about current Erlang/OTP Releases
Kenneth Lundin

16.45 Non-Destructive Arrays
Richard Carlsson and Dan Gudmundsson

17.00 How to program efficiently with Binaries and Bit Strings
Per Gustafsson

17.30 Bus transport to the **ErLounge**

TfL Journey Angel

Context Aware Decision Support

Erlang Training and Consulting Ltd
www.erlang-consulting.com

Marcus Taylor
marcus@erlang-consulting.com

The Transport for London Brief

Demonstrator Name Mobile avatar solution – 'Journey Angel'

Description **This demonstrator will deliver a mobile avatar system prototype that will assist the passenger throughout his time in London: pre-journey, in-journey and post-journey.**
The software uses a smart mobile client to produce a mobile avatar with speech capability.

The Avatar will support advisory/decision support actions including:

- **Installation, configuration and personalisation of Avatar**
- **Incident alerting**
- **Delay alerting**
- **Planning support**
- **The client software will communicate to the server system for updated content.**
- **The Avatar will do pseudo realistic lip synching.**
- **Loquendo will be used for real time text to speech**

Technology

- **Client**
  - *N73*
  - *Symbian*
  - *Flash*
  - *Loquendo*
  - *MRIX*
- **Server**
  - *EjabberD (XMPP compliant jabber IM server)*
  - *Erlang/OTP*

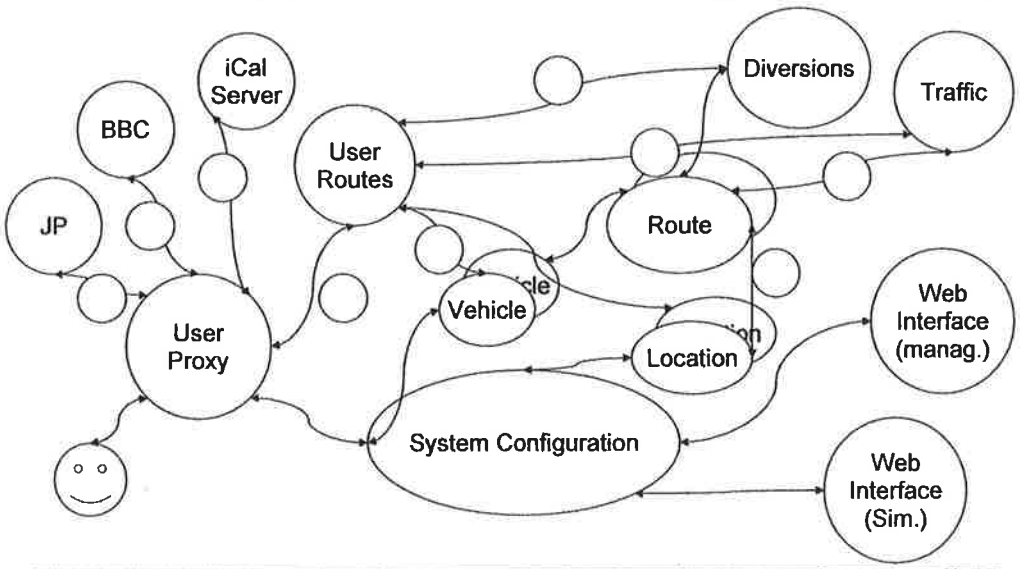
What behind Journey Angel ?

- **TheJourney Angel is a powerful platform for distributed context-aware information spreading, using**
  - *Standard technologies (Erlang, Jabber, Flash)*
  - *Decentralized management*
  - *Intrinsically robust architecture*
  - *Scalability*
- **Those characteristics are critical for TFL: they need a solution to manage customized messaging to millions of travellers everyday....**
- **...and Journey Angel gives such a solution**

The "target" (from TFL perspective)

- **The system allows users to:**
  - *Get travel information*
  - *Plan trips according to their future activities*
  - *Get context-aware information about delays, diversions, alerts and similar*
  - *Get additional (mostly unspecified) customised information, e.g. advertisement, weather, commercial....*
- **It is likely that TFL would like to develop the system into to a real product...**
- **...and there are many other use-cases that can be addressed by context aware system....**

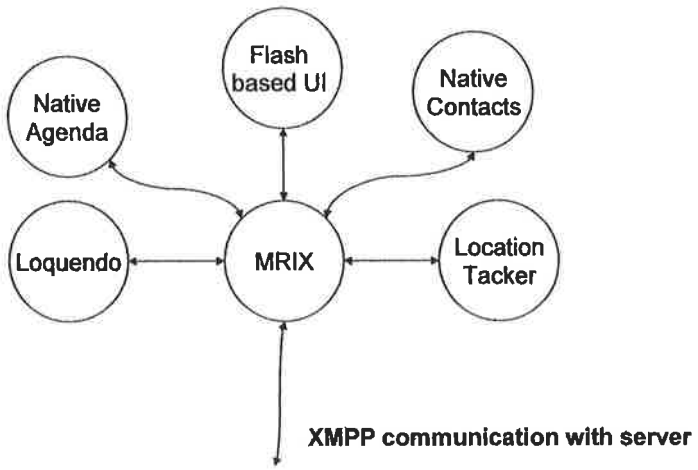
The final system: what we have now



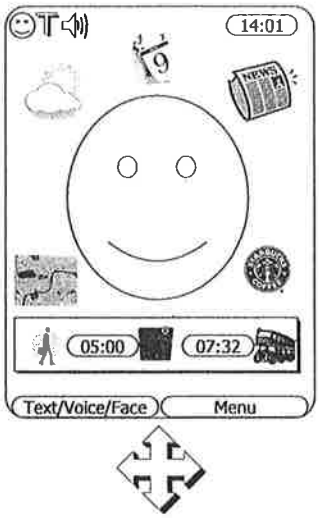
Journey Angel: Device

- **Journey Angel on the device uses the following:**
  - *MRIX to provide integration between the UI, TTS, server-side and native device applications (calendar and contacts)*
  - *XMPP to communicate with the server*
  - *Flash for the User Interface*
  - *Loquendo for Text To Speech*

Device side architecture



TfL Start Menu (TVF)



Menu

- Take me Home
- Take me to...
- Meet with...
- Where am I?
- What next?
- Settings
- Exit

Traversing Horizontally
Weather, Calendar, Alerts
Map/Itinerary and Poll

MRIX: Integrating server and Flash UI

- An MRIX script provides communication between the server and Flash UI
- The script runs in the background communicating with the server
- Messages received can be personalised and then converted to speech
- Once in audio form the Flash UI can be told to 'speak the message'

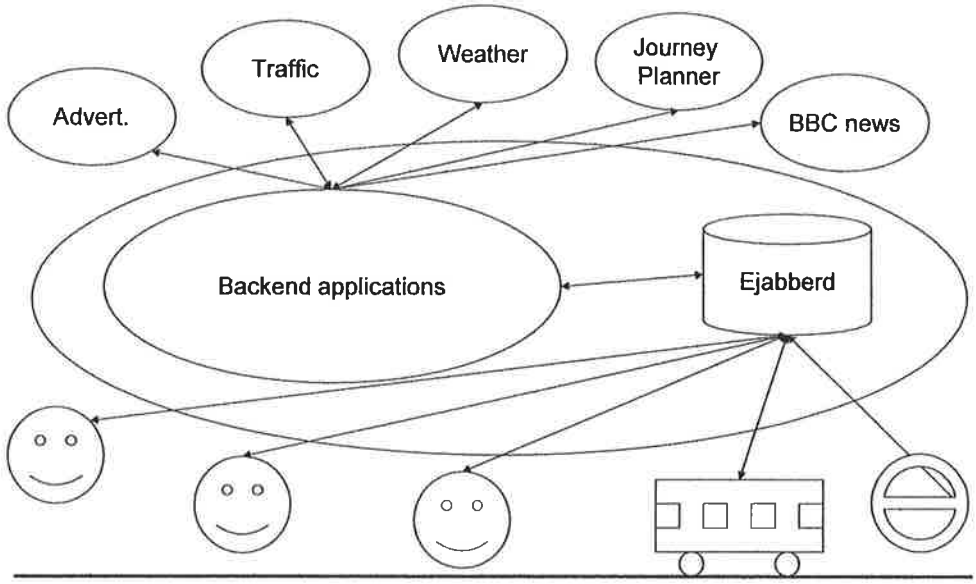
MRIX: Integrating Flash UI and native apps

- MRIX enables the built in Contacts and Agenda to be used
- Routes can be automatically retrieved based on an appointment in the agenda
- Contacts can be 'tagged' with network information, such as Cell ID and Bluetooth Access Point to help locate a person

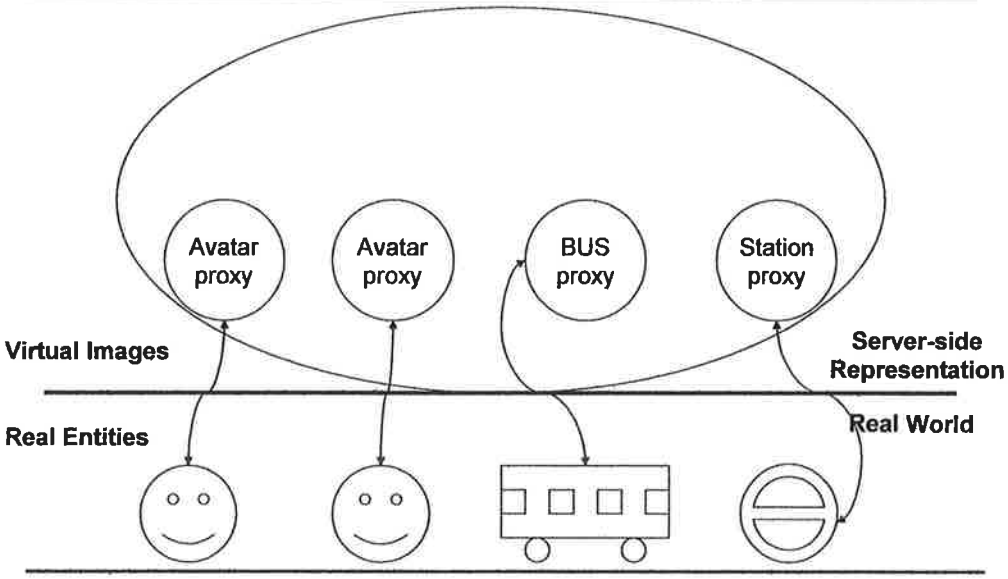
Journey Angel: Backend

- Journey Angel relies on a backend infrastructure based on
  - *XMPP (Jabber) communication protocol (Ejabberd)*
  - *Erlang server applications*
  - *Interfaces to information feeds*

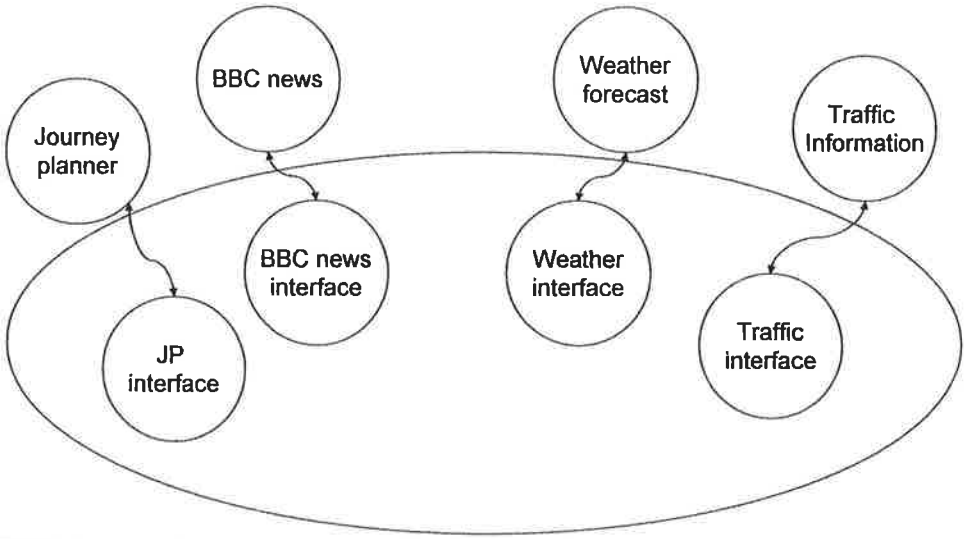
Behind the scenes...



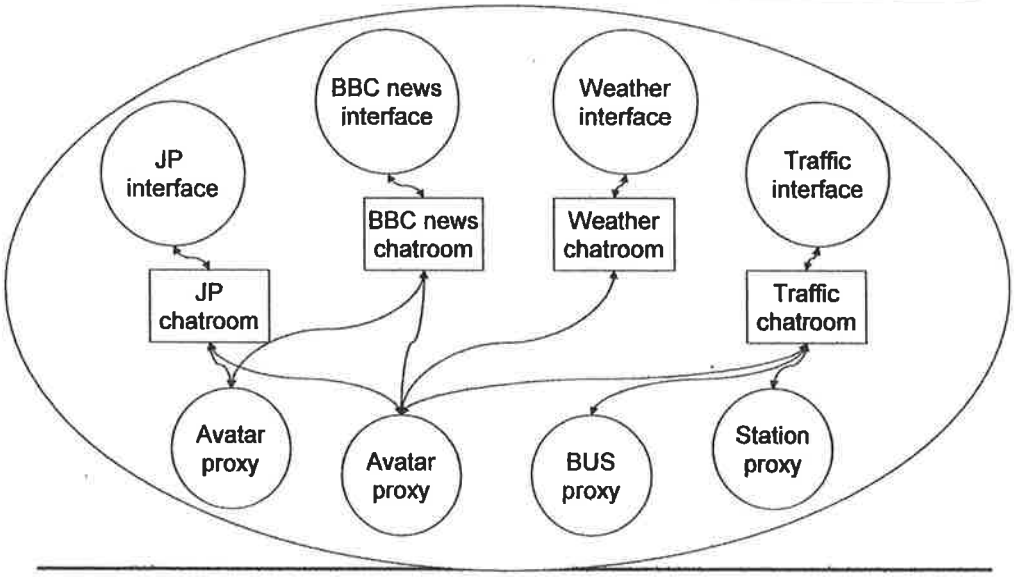
Backend Applications: proxies



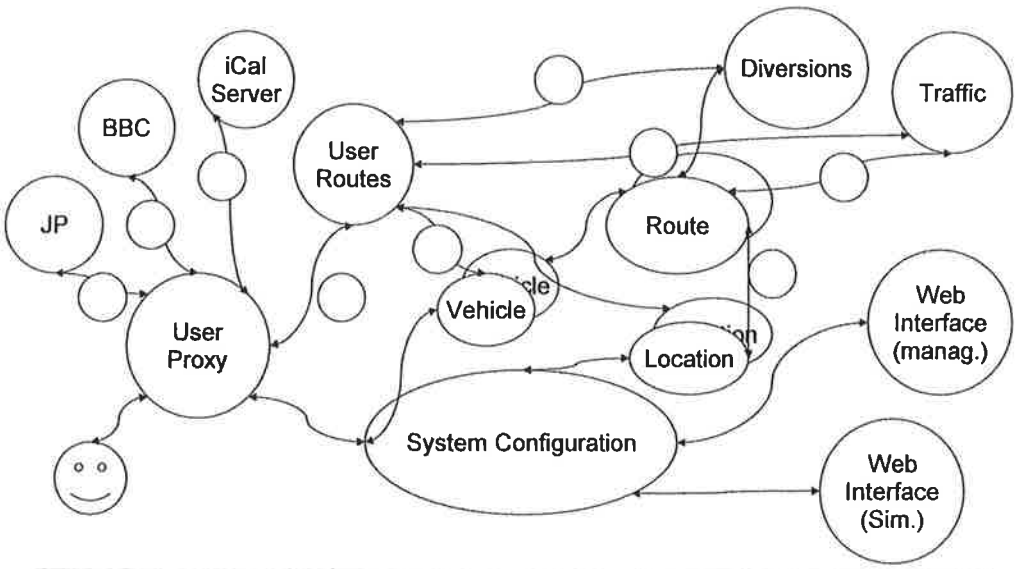
Backend Applications: interfaces



Internal communication



The System



Conclusions

- The system demonstrates how it is possible to deliver real time decision support systems to mobile users by combining Jabber, XMPP, 3G and Smart Phones using Symbian
- The use of Erlang/OTP has enabled us to deliver a feature rich, robust and functioning system in short timescales
- The system has the potential to provide personal angels for horizontal and vertical applications such as finance, health, logistics where the context and timeliness of information constitute value.

1 (5)

YXA PROJECT

Fredrik Thulin <sip:ft@it.su.se>
Sektionen för IT och media
Stockholms universitet

EUC 2007

(draft version)

YXA AT EUC 2004

- SU telephony systems
- SIP background
- Me and Erlang
- YXA at the time

- Plans :
  - Distributed services
  - Policy control
  - Event logging
  - RFC compliance
  - Perimeter defense

YXA 1.0

should be released by EUC'07

Project goals

- Robust SIP server for 10,000's of users
- Scalable by distributing servers
- Short time-to-market
- Interoperability

YXA characteristics

- Specific version requires specific Erlang/OTP version
- Adopts new stuff
  - try/catch
  - orelse in guards
  - EDoc
- Easy to extend/modify

Frameworks

- I Like frameworks :
  - Configuration subsystem
  - User database backends
  - Transports
  - Events
  - local.erl with 87 hooks

  - SIP Event server (RFC3265) framework for packages

Robustness

- 2867 test cases
- Test integrated in release process
- Snapshots and release candidates
- Dialyzer
- Pay close attention to compiler warnings
- SIPit's

Speed

- Who needs speed?
  - Presence
  - Four servers better than 16
  - Fast initial parsing

- Profiling
- Logging
- 70 CPS on old laptop (70 * (INVITE + BYE))

Value for others?

- Lots of documented code
  - OTP supervisors, gen_event and gen_servers
  - Binary and string parsing
  - Network code (TCP, UDP, TLS, IPv6 (!))
  - SSL stuff
  - Mnesia
- Well written code I hope
- ./configure && make && make install

Project info

http://www.stacken.kth.se/project/yxa/

svn://anonsvn.it.su.se/yxa/trunk/

BSD license

Erlang Developments in LambdaStream

Samuel Rivas **samuel.rivas@lambdastream.com**

1 - LambdaStream History

LambdaStream is a technology-based startup company that develops software products for streaming and on-demand media delivery. It was born as a spin-off from the MADS research group from the University of a Coruña.

Its main product, VoDKA, is a distributed video-on-demand (VoD) server developed with Erlang/OTP technology. Through its four years of existence, LambdaStream has developed a set of products around VoDKA to cover markets such as mobile TV, Internet TV, broadcast TV, IPTV, and billboard TV.

2 - Overview of Relevant Erlang-based Projects

Many of the LambdaStream's products are based on Erlang/OTP technology. This section will show a short overview of those projects.

3 - More Detailed Review of Antares

Apart from VoDKA, Antares is one of the relevant LambdaStream products. Antares is an Electronic Service Guide for DVB-H Mobile TV systems. It can integrate information coming from heterogeneous sources, store it in a DB with a unified representation of those data, and broadcast it according to different standards.

4 - Conclusion

A review of some successful commercial deployments and conclusions about Erlang/OTP.

1 (14)

erlware

Erlware For Managing Distribution and Build

Erlang User Conference 2007

Erlware Goals

- Allow the Erlang Community to drive itself
- Allow the community to evolve the language/platform
- Make the language/platform accessible to reasonably competent programmers

2

Steps to Reach Erlware's Goals

- Define a Common Repository for OTP Artifacts
- Make it easy for anyone to host their own version of the Repository
- Build tools that leverage the features of the Repository

Repository Design Goals

- Be Lazy
- Don't add more meta data then you need
- Where ever possible leverage existing OTP metadata

Repository and OTP

- OTP already provides most of the metadata that we need.
- OTP Applications make good artifacts.
- OTP *.app files make for good metadata.
- OTP *.boot, *.script, *.rel files make really good release packages.

Anatomy of a package

Application

- Contains a src directory
- Contains an ebin directory
- Contains a *.app file
- Contains source code

Release

- Contains an *.rel file
- Contains a sys.config file
- Contains command templates

4

Repository Implementation

- Dirt Simple
- Http Server with WEBDAV Enabled
- Specific, well documented directory structure

Repository Layout

The structure of the repo is

<erts-vsn>/<arch>/<side>/<packagename>/
<vsn>/<package>

and

<erts-vsn>/Meta/<appname>/<vsn>/<.app file>

Example: The Sinan-1.0.1 release package for mac sits in:

http://repo.erlware.org/pub/5.5.5/
i386-apple-darwin8.9/releases/sinan/

Leveraging the Repository

Sinan is a build system that 'understands' erlang and OTP. It makes it very easy to handle complex OTP build logic

Faxien is a distribution system built on the same backend as Sinan. It provides functionality similar to rubygems or CPAN for Erlang.

- OTP Centric Build System
- Expects OTP Applications, Understands OTP Applications
- Written and extensible in Erlang

Getting Started

Generate an OTP Project

sinan gen

After asking a few questions generates a useful, sinan compliant, compilable OTP application with a complete skeleton

Dependency Detection

Just run

sinan depends

Evaluates the OTP application dependencies and restrictions in the project. Hits the repository to gather information. Comes back with a list of hard dependencies (app, version) for each of the dependencies and transitive dependencies in the project. Many other tasks in Sinan depend on this one.

Building

Just run

sinan

or

sinan build

Builds all of your artifacts into the _build directory. Will only rebuild when a file needs to be rebuilt. Understands includes, parse transforms etc.

Testing

We decided to leverage eunit in Sinan. Want to run all of the eunit tests in your project.

sinan test

Sinan prints successes and failures to the shell. It also generates html code coverage reports to _build/reports.

The Good Stuff - Releases

Want to generate *.rel, *.boot, *.script files for your project? How about taring it up and getting it ready for a push to an Erlang node?

Just do

sinan release

or

sinan tar

Best of all, Faxien knows how to push these releases out to a repository. That makes distribution a no brainer.

Extra's

How about running dialyzer on your project?

sinan analyze

The first time its run it generates a plt file for dependencies so you only get the output relevant for your project.

Want an erlang shell with all of your paths set so you can noodle with your project?

sinan shell

The Future

Anything you want. Sinan is an extensible system so anything useful can be quickly incorporated. Of course, there is still a lot of functionality we want to expose in Sinan. No doubt the community will come up with a bunch of things that we have never thought of.

- Easy to find and install otp packages
  - list out packages available in remote repositories
  - simply type faxien install <package name> to install anyone of them
  - Install applications and releases
- Easy to publish OTP packages so that community can access them
  - faxien publish <package directory>
  - auto discovers package type, app, release, or erts

How does installation work?

- Configured to know about a list of repositories
- Pulls down packages both releases and applications then installs the now local tars creating scripts and boot files the fit the local environment among other things.
- Pulls down erts (you need not install Erlang from source anymore)
- If a package is not found for the erts vsn you specify Faxien automatically selects the next lowest compatible erts vsn.
- By default all packages are installed in /usr/local/erlware or on windows c:\erlware though this is fully configurable. (windows code not complete as of this writing)

How does publishing work?

- Publishes into a repo location specific to the local architecture, if project is pure erlang then publishes to a platform neutral location
- Auto discovers the package type by looking for things like a .app file or a .rel file, or the fact the the directory looks something like "erts-<vsn>".
  - faxien publish sasl-1.3.2.1 works
  - faxien publish my_release-2.3.2 works
  - faxien publish erts-5.5.5 works
- if a package does not have the appropriate OTP structure it will not be published.

Install the tools

- http://code.google.com/p/faxien (links also found at www.erlware.org)
- Execute faxien_launcher -b to install faxien
- now install Sinan with faxien install sinan

Installation

The past -

- Google for an application that fits your needs
- Find lots of stuff written in perl but no erlang
- Some obscure website listed on the 13th page has a half baked erlang app
- Hack around with it for a bit
- Give up and write your own

With Faxien -

- faxien list
- pick an app
- faxien install_app <appname | app.tar.gz> [vsn]
- faxien install <release | release.tar.gz> [vsn]

Publishing

The past -

- build it
- write some convoluted make file that would break for the next guy,
- put it up on your website and hope people find it
- Be disappointed when no one does and watch the community grow slowly without your code

With faxien

- "faxien publish my_app"
- "faxien publish my_rel"

Now everyone can get at it

Extra Commands

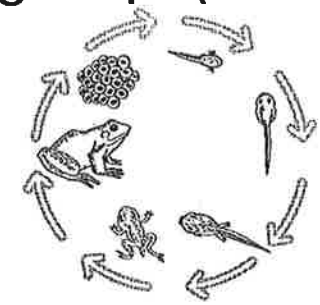
Faxien allows you to do quite a bit more than this short presentation has time to address. Two commands I want to note are:

faxien list - list all available apps that match the supplied pattern across many repos

faxien upgrade - upgrade one or all apps on the local file system to their latest vsn

And of course lets not forget...
faxien help – to print out a list of what faxien can do

Summing it up. (Erlware cycle)



1. Install Faxien (Erlang Package Management)
2. Install Sinan (Erlang Build System)
3. Use Sinan projgen to create a project
4. Build, document, and test with Sinan
5. Publish an app or release with Faxien
6. Other folks download and use your package

Erlware Goals Revisited

- Allow the Erlang Community to drive itself
  - Create otp packages with sinan publish and install them with faxien. Now we can easily leverage eachothers work.
- Allow the community to evolve the language/platform
  - No need to install erlang from source, you can create your own release and people can simply install that – if you want to bundle your alternative to stdlib instead of the original go ahead
- Make the language/platform accessible to reasonably competent programmers
  - OTP apps and releases are easy to install and build install with Faxien and Sinan.

Steps to Reach Erlware's Goals

- Define a Common Repository for OTP Artifacts
  - The repo = **done**
- Make it easy for anyone to host their own version of the Repository
  - Just a webserver = **done**
- Build tools that leverage the features of the Repository and help build our community
  - Faxien & Sinan = **done**

erlware®

Quality Cruising
Making Java Work for Erlang

Erik Stenman

This is a preliminary version for the EUC proceedings the final version with more detailed examples will be made available on the EUC web-page.

Introduction

I will talk about automated testing, and how to make Java do that work for you.

But first a short recap for those of you who weren't here last year or don't remember my talk about Kreditor.

Kreditor Europe AB

- Founded in December 2004.
- Bring trust to Internet shopping, by providing old style billing through hi-tech solutions.
- More than 1600 Internet shops connected.
- The company vision:
  - "Be the coolest company in Sweden."

Some implementation details

- The system is built from scratch using LYME (Linux, Yaws, Mnesia, and Erlang).
- We have a distributed system with multiple servers to provide a fault tolerant, high availability solution.
- We aim for 5 nines availability, in a setting where we introduce new features in the system every week (often every day).
- The problem fits Erlang really well.

Process

- We have have an informal agile process.
- We have very short time to market, for simple changes the time from idea to use in production can often be less than one hour.
- It is crucial for us to have an automated comprehensive test suite.
  - With a framework that works.
  - Which is **used**. **Always**.
- Enters Yatsy and Cruise Control.

Yatsy

- Yet Another Test Server – Yaws compatible
  (Yatsy is Swedish for Yatzee – testing is a bit like a dice game.)
- Why (yet) another test server?
  - The "*released*" OTP-test server is from 2004.
  - It isn't really open source.
  - We just couldn't get it to work.
- First version hacked together over a weekend by Tobbe.
  - Released as open source:
    `http://code.google.com/p/yatsy/`

Yatsy - example

```
-module(example_SUITE).
-export([all/1, init_per_suite/1, fin_per_suite/1, init_per_testcase/2,
        fin_per_testcase/2, simple/1]).
-include("yatsy.hrl").

all(doc) -> ["Test cases for example."];
all(suite) -> [simple].

init_per_suite(Config) when list(Config) -> Config.
fin_per_suite(_Config) -> ok.
init_per_testcase(_TestCase, Config) when atom(_TestCase),list(Config) -> Config.
fin_per_testcase(_TestCase, _Config) -> ok.

simple(doc) ->
    ["Check that we can get an new example."];
simple(Conf) when is_list(Conf) ->
    [] = example:new(),
    ok.
```


KREDITOR

Cruise Control (CC)

- Cruise Control is a framework for continuous integration.
- It is open source.

  It is written in Java.
- It automatically checks out the latest version from a repository, does a build, and runs all tests – as soon as anyone checks anything in.

KREDITOR

CC – Components

- A set of plugins
  - Version control pollers (eg. wrapped svn st -u)
  - Compile and test systems (eg. ant)
  - Publishers
    - web site, email, rss, irc, etc...
- A build queue
- And some other things (admin gui, etc ...)

The CC loop

- Poll for event
  - usually version control update or time based
- Compile and run tests
- Gather results
  - return value from script and xml report files
- Publish results

Using CC

- A set of projects
  - commit-[branch], nightly-[branch], etc ...
- One config file
  - Which plugin to use
  - Parameters for each plugin

KREDITOR

Integrating CC with Yatsy

- CC starts a script
  - svn update
  - make
  - run yatsy
- Reporting
  - Yatsy reports test results trough an xml file
    - Reverse engineered from JUnit

KREDITOR

Conclusion

7

- CC gives you immediate feedback on the test status of all your branches all the time.
- Our contribution:
  - Open sourced test suite – Yatsy.

    Nice simple integration between CC and Erlang – we let Java do some work for the Erlang developer.
- CC is just as valuable as a VCS.
- If you are not using CC for your projects, start using it **NOW**.

KREDITOR

1(9)

Generic syntactic analyser: ParsErl *

Róbert Kitlei, László Lövei, Tamás Nagy, Anikó Nagyné Vig,
Zoltán Horváth, Zoltán Csörnyei
Department of Programming Languages and Compilers,
Eötvös Loránd University, Budapest, Hungary
{kitlei,lovei,n_tamas,viganiko,hz,csz}@inf.elte.hu

Abstract

The increasing demand in automatic code transformation tools – which can preserve the layout, and can handle the whole macro syntax – led us to develop our scanner and parser tool. ParsErl is a generic syntactic analyser for Erlang. The scanner and the parser are generated from an XML definition of the grammar. The result of the scanning process is a graph, which can be optimised or balanced for applications. The tool can preserve the original layout of the source code, including the original macro definitions. Our preprocessor creates connection between the original source code's tokens and syntax tree's nodes. We can provide the substituted and parsed code for the applications and we can generate the original source code back, when it is needed.

1 Introduction

The increasing amount of codebase which has to be maintained resulted in an increasing demand in automatic code transformation tools. For example, refactoring tools which can change (usually applied in order to improve) the structure of the code without changing its behaviour [2, 3, 4, 6, 5].

These tools work on a higher abstraction layer than textual format. The usual approach is to apply syntax analysis that produces an abstract syntax tree (AST) of the source code. The standard Erlang parser with the `syntax_tools` application provides an interface to produce and work with such an AST [1, 7, 8].

The problem with this approach is that this parser was designed for code generation. It provides an interface which can generate text from the AST, but this result will be pretty printed, because the parser discards the layout, whitespace, and punctuation while building the syntax tree. These information are irrelevant for code generation but highly valuable for the code transformation

*Supported by GVOP-3.2.2-2004-07-0005/3.0 ELTE IKKK and Ericsson Hungary.
A full technical paper about the design of the internal structure was submitted to CC2008.

2

tools. Preserving this information we can preserve the original layout in contrast to pretty printing.

The other problem with the standard tools arise when the language supports macros. Macros are usually substituted with their definition by a preprocessor before parsing. The Erlang tools can support macros without substituting them if they "behave" well. If a macro cuts syntactic entities in half, the tools cannot parse it. This means that some code can be compiled, but cannot be parsed by the standard tools before prepocessing.

In this paper we will show that these problems can be solved with a new parser, if the design aim is the layout preserving, and the support of the full macro syntax. Furthermore we will give an API which makes it possible to apply the framework in different projects.

2 Motivation

The prototype version of our refactoring tool, RefactorErl, suffered from the above problems. Because in refactoring tools it is a crucial point to be able to support the whole syntax and to keep the layout as it was as much as possible. This is very important, because the pretty printing makes it hard to follow the changes made in the code, let alone carry out further changes even with the refactoring tool not to mention by hand.

3 Structure of the tool

In Figure 1 we show the layers and parts of the tool. We generate the scanner and the parser based on an extensible XML description of the Erlang grammar. After the scanning and parsing process is done, an application can work on the graph representation. Our layout preserving printer can restore the original source code in textual format from the graph representation.

3.1 XML

Both the lexical elements and the syntactic rules, and the resulting structure reside in the same XML file. This makes the definition easily adoptable, customisable to language changes, and to different application needs. Obviously the former happens really rare, because that would need changes in a lot of applications, and could cause issues with backward compatibility.

3.1.1 Lexical elements

Lexical elements are described by the element *lexical*. Patterns (elements *pattern*) are quite similar, only they don't constitute an element themselves. Important patterns are the whitespace-and-comments before and after the tokens (named PRE and POST), which are included in the token text themselves. Patterns can be incorporated using a *match* element. Elements may further consist

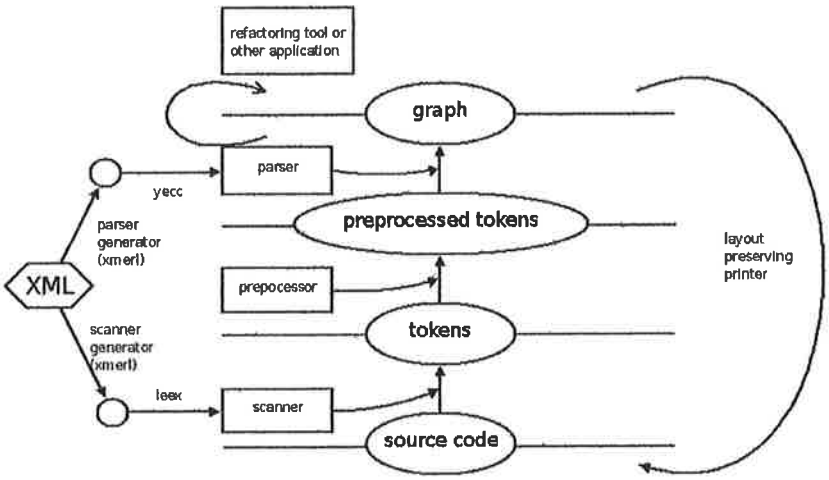


Figure 1: The structure of ParsErl

of plain text, branches, optional and repeated parts, tagged by *text*, *branches*, *opt* and *rep* respectively. There are some additional facilities for easier character inclusion: *chars-of* and *chars-but*, which permit all characters included in (or excluded of) a given set.

The following example shows the description of integers.

```
<lexical name="integer">
  <match name="PRE"/>
  <opt>
    <branches>
      <br>
        <text>1</text>
        <chars-of><range><from>0</from><to>6</to></range>
        </chars-of>
      </br>
      <br>
        <chars-of><range><from>2</from><to>9</to></range>
        </chars-of>
      </br>
    </branches>
    <text>#</text>
  </opt>
  <match name="Digit"/>
  <rep>
    <match name="Digit"/>
  </rep>
  <match name="POST"/>
</lexical>
```

The following regular expression is generated from this description:

4

```
{PRE}(1[0-6]|[2-9]#)?{Digit}{Digit}*{POST}|
```

3.1.2 Syntax elements

The syntax elements describe the context free grammar rules. All rules with the same head symbol are organised under a *ruleset* element. They may contain rules that are not represented in the graph themselves: these are called *copy-rule*. All other rules have to specify in which class do they belong. This way we can simplify the syntax graph by storing only so much information as necessary. For example, all different kinds of expressions are in class *expr*, and are not distinguished from each other further on.

Rule elements (the right hand sides of rules) consist of *tokens* and *symbols*. For the sake of brevity, elements *optional* and *repeat* are also available.

With the *symbols* we have to define how we want it to be connected to the head symbol. For example the function clause's pattern elements are connected to it with a link tagged with *pattern*, the guards with *guard* and the body's elements with *body*. These tags will be used for information retrieval.

Rules may also contain attributes that are stored as additional information in the graph during parsing. For example, guard sequences may contain *conjunctions* and *disjunctions*; both are represented as an *expr* node with the appropriate kind as attribute.

The following example and Figure 2 show the rules for function clauses.

```
<ruleset head="FunClause">
  <rule class="clause">
    <attrib name="type">funcl</attrib>
    <symbol name="Atom" link="name"/>
    <token type="op_paren"/>
    <optional>
      <symbol name="Expr" link="pattern"/>
      <repeat>
        <token type="comma"/>
        <symbol name="Expr" link="pattern"/>
      </repeat>
    </optional>
    <token type="cl_paren"/>
    <optional>
      <token type="when"/>
      <symbol name="Guard_seq" link="guard"/>
    </optional>
    <token type="arrow"/>
    <symbol name="Expr" link="body"/>
    <repeat>
      <token type="comma"/>
      <symbol name="Expr" link="body"/>
    </repeat>
  </rule>
</ruleset>
```

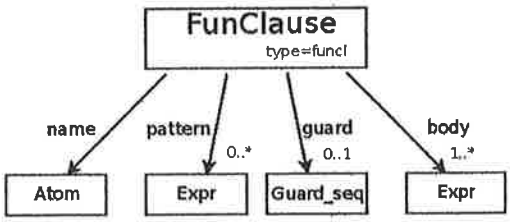


Figure 2: The function clause rule without the tokens

3.2 Scanner

The scanner is automatically generated from the XML definition with an XSLT. The XSLT is written with the Erlang's `xmerl` application [13, 14, 15]. The XSLT transformation's result is the input of the `leex` application [12].

The definition can be easily adjusted to keep the comments and the whitespace information or discard them. In our case we chose to attach the whitespace information and comments to the lexical categories of the language, therefore there is no whitespace token. The interface of this module is the standard interface provided by the `leex` application.

3.3 Preprocessor

A middle layer has been introduced between the scanner and parser to be able to support any kind of macros which are allowed in the languages definition. This layer defines connection between the original source code's tokens and syntax tree's nodes. This relation is not trivial because the syntax tree can only be built when the macros had been substituted. Therefore our preprocessor has to be aware of the structure being built by the parser. As a result of this the preprocessor does not provide a standard interface for invocations. It is embedded into the parser.

3.4 Parser

The parser is also generated from the XML definition. The XSLT transformation's result is the input of the `yecc` application [9, 10, 11]. The built structure can be adjusted just by adjusting the definition in the XML. The API is extended compared to the `yecc`'s default interface in order to get the correct result structure. Parsing one form at a time is supported, and additionally the extended API provides means to parse a whole file as well.

3.5 Graph

A graph is the result structure of the parsing. The shape of the graph only depends on the definition of the syntax in the XML file. Therefore based on our

6

preferences/needs the shape can be adjusted to a certain extent. Because the `yecc` uses LALR-1 analysation method the structure has to be resemble a tree.

For example in our refactor tool we decided not distinguish between the different expressions. The expressions' type is always expression. The standard parser's expression types are just attributes.

4 Information retrieval

High level information retrieval is supported by a query language that makes it easier to traverse graph structures with fixed depth. This query language consists of *path expressions*. To evaluate it a start node and the list of links we want to follow from the start node is required. The direction and filters of the links can be given. Direction can be forward or backward. The possible filters are:

```
Filter = {Filter, 'and', Filter} | {Filter, 'or', Filter} |
         {'not', Filter} | {Attrib, Op, term()}
Attrib = atom()
Op = '==' | '/=' | '=<' | '>=' | '<' | '>'
```

The links also have indices which start at 1. Therefore it is possible to choose one link or an interval of links.

```
Index = integer() | {integer(), integer()} | {integer(), last}
```

`Start, End` means the indices larger or equal than `Start` and smaller than `End`.

For example (using our structure) if we want to retrieve the module name of the source file we can write a *path expression* like this. Suppose we have the root of the file in the `Root` variable:

```
path(Root, [{form, {kind, '==', module}}, {attr}])
```

The result would be a list containing one element. The module name is the result node's attribute which can be obtained by another function call.

5 Linking with other applications

The API demonstrated in Section 4 provides an interface to other applications which can be easily used. The built graph structure can be fine tuned to specific applications. The information retrieval mechanism - the functions, parameters - do not change when the defined structure changes. These altogether yield a highly adoptable/optimisable structure.

7

6 Conclusion and Future work

In this paper we have shown that it is possible to support the whole syntax of the Erlang language with a parser which can retain the original layout of the code. Furthermore, having the definition of the whole language and the result structure defined in one XML file makes the language definition easily adjustable to changes in the language. The resulting structure can be easily adapted to any specific problem. For example balancing the resulting graph's height and width for optimising to the application's algorithm. The framework even makes it possible to add extra information to the graph which cannot be derived directly from the syntactic rules.

The `ifdef`, `ifndef` macros introduce further difficulty. For example consider the following code:

```
-ifdef(debug).
-define(LOG(X), io:format("{~p,~p}: ~p~n",
                          [?MODULE,?LINE,X])).
-else.
-define(LOG(X), true).
-endif.
```

The macro's body is different depending on the value of the condition. Even if we work on the unsubstitued version of the source, we have to consider what the substituted code would be.

A further development would be to develop the printing mechanism to be able to parameterise it with design rules to enforce the same layout of different developers.

References

[1] J. Barklund and R. Virding.
Erlang Reference Manual, 1999.
Available from http://www.erlang.org/download/erl_spec47.ps.gz.

[2] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts.
Refactoring: Improving the Design of Existing Code.
Addison-Wesley, 1999.

[3] H. Li, C. Reinke, and S. Thompson.
Tool support for refactoring functional programs.
Haskell Workshop: Proceedings of the ACM SIGPLAN workshop on Haskell, Uppsala, Sweden, p. 27–38, 2003.

[4] H. Li, S. Thompson, L. Lövei, Z. Horváth, T. Kozsik, A. Víg, and T. Nagy.
Refactoring Erlang Programs.

In Proceedings of the 12th International Erlang/OTP User Conference, November 2006.

[5] R. Szabó-Nacsa, P. Diviánszky, and Z. Horváth.
Prototype environment for refactoring Clean programs.
In The Fourth Conference of PhD Students in Computer Science (CSCS 2004), Szeged, Hungary, July 1–4, 2004.

[6] Lövei, L., Horváth, Z., Kozsik, T., Király, R., Víg, A., and Nagy T..
Refactoring in Erlang, a Dynamic Functional Language.
In Proceedings of the 1st Workshop on Refactoring Tools, pages 45-46, Berlin, Germany, July 2007.

[7] Erlang 4.7.3 reference manual.
http://www.erlang.org/download/erl_spec47.ps.gz

[8] Erlang 5.5.5 reference manual.
http://www.erlang.org/doc/reference_manual/part_frame.html

[9] Torbjörn Törnkvist
How to improve the performance of YECC-generated Erlang (JAM) parsers
Published in the Software Engineering Research Center of the RMIT University(SERC), Melbourne, Australia. December 12, 1997.
http://www.erlang-projects.org/Members/mremond/serc/how_to_improve_the_p/block_10914819836344/file

[10] Magnus Fröberg
Automatic Code Generation from SDL to a Declarative Programming Language
In Proceedings of the Sixth SDL Forum, Darmstadt, Germany, October 1993.
www.erlang.se/publications/sdl2erlang.ps

[11] yecc documentation.
www.erlang.org/doc/man/yecc.html

[12] Leex beta version download page
http://tinyurl.com/yvl6tp

[13] Ulf Wiger
XMErl - Interfacing XML and Erlang.
In the Sixth International Erlang/OTP User Conference (EUC 2000), Stockholm, Sweden, October 3, 2000.
http://www.erlang.se/euc/00/xmerl.ppt

[14] Mickael Rémond
XML and Erlang: Building a Powerful Data Management Tool.
In the Sixth International Erlang/OTP User Conference (EUC 2000),

Stockholm, Sweden, October 3, 2000.
http://www.erlang.se/euc/00/remond/mgp00001.html

[15] xmerl documentation.
http://www.erlang.org/doc/apps/xmerl/

Building A REST-based Platform With Erlang And Adobe Flex

hypernumbers

Gordon Guthrie
Dale Harvey
Hasan Veldstra

This presentation the details of the platform, in particular how Erlang and Flex are a match made in heaven

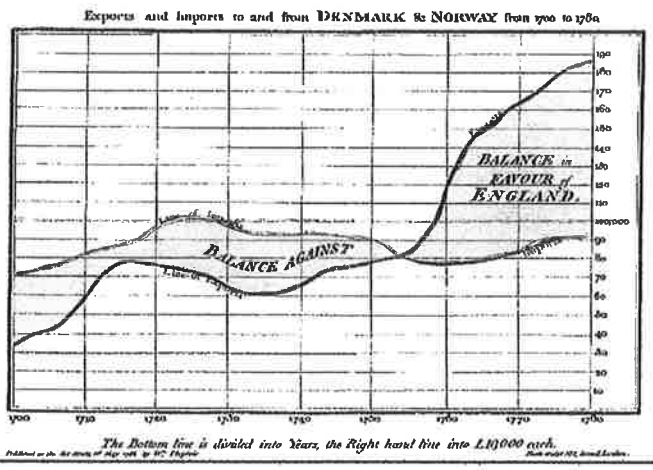
- The Product
  - the hypernumbers Matrix Server
- API Design
- Adobe Flex
- Next Steps

2

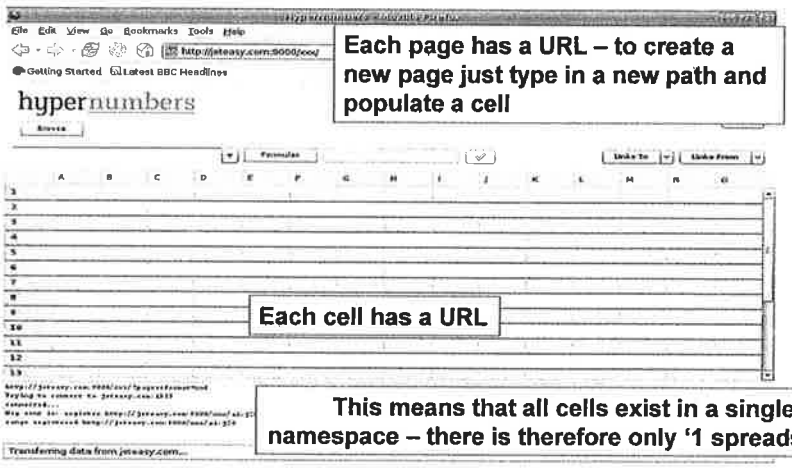
what we have been working on

THE PRODUCT

The product is a server codenamed 'William Playfair' that serves hypernumbers - named after the Scotsman who invented the graph in 1759



At first glance our product looks like 'a spreadsheet on the web' but our goal is actually to make 'a spreadsheet of the web'

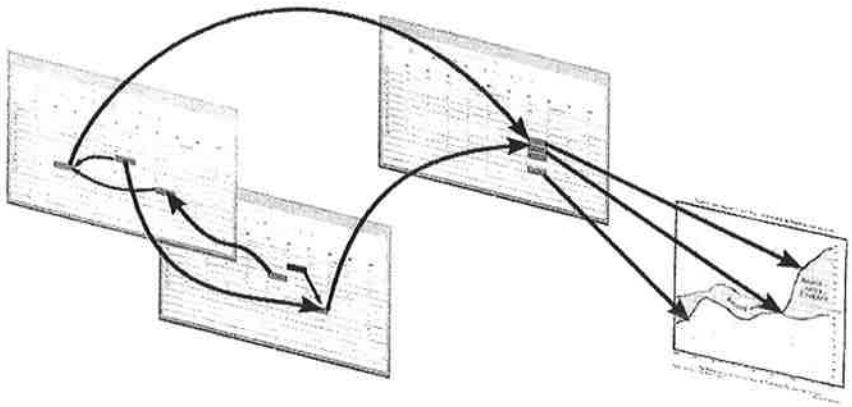


Each page has a URL – to create a new page just type in a new path and populate a cell

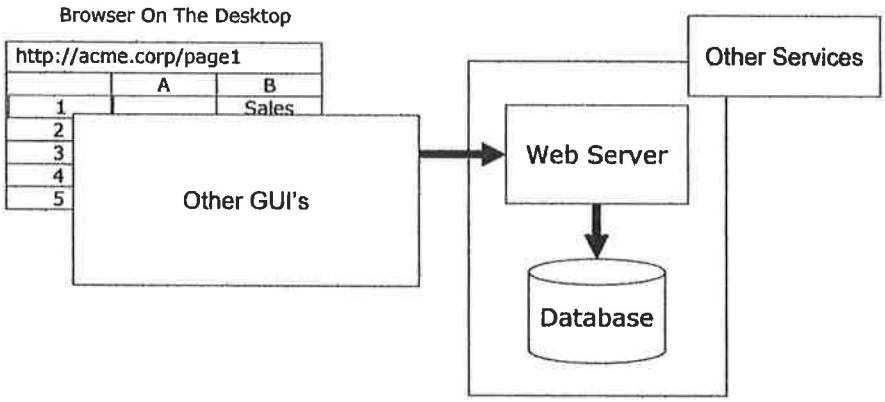
Each cell has a URL

This means that all cells exist in a single namespace – there is therefore only '1 spreadsheet'

So the cells on my websheet can update the cells on yours directly (or your graphs or other applications and components)



The critical thing is that it is a platform for the delivery of hypernumbers and products/GUI's that use them...



The critical decision is to treat 'pages' and 'cells' (or parts of pages) separately, and only use Flash for single pages...

- **a page is terminated by a slash:**

`proto://subbies.domain.tld[:port]/page/path/`

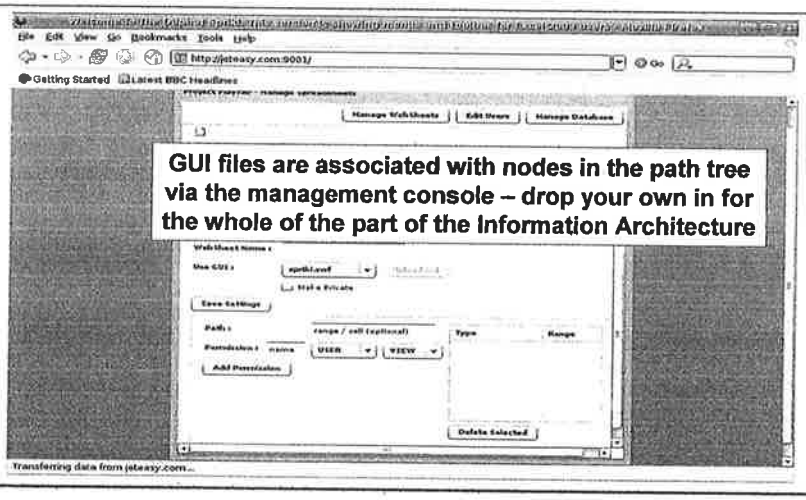
- **a cell is not:**

`proto://subbies.domain.tld[:port]/page/path/ab23`

- **and we use query's to build the API:**

`proto://subbies.domain.tld[:port]/page/path/ab23?toolbar`

This gives us a simple loading cycle – the page URL loads the GUI as a single object and the Flash `init()` event builds the GUI



So how do we actually use a hypernumber – what do the building blocks of this platform actually consist of?

You set the value of a hypernumber with an http(s) POST:

```
protocol://subbies.domain.tld[:port]/page/path/ab23
```

with `action=create`

`value=12345`

value can be:

a number 12345
a string "bob"
a formula =1+2
=a1+b2 cells on the same page
=/some/other/page/a1 absolute page
addressing
=/a/page/name("sub-total") by cell name
=../../relative/page/b1 relative page
addressing
=hypernumber("proto://sub.dom.tld:port/mypage/a3")

and you get the value using an http(s) GET:

protocol://subbies.domain.tld[:port]/page/path/ab23

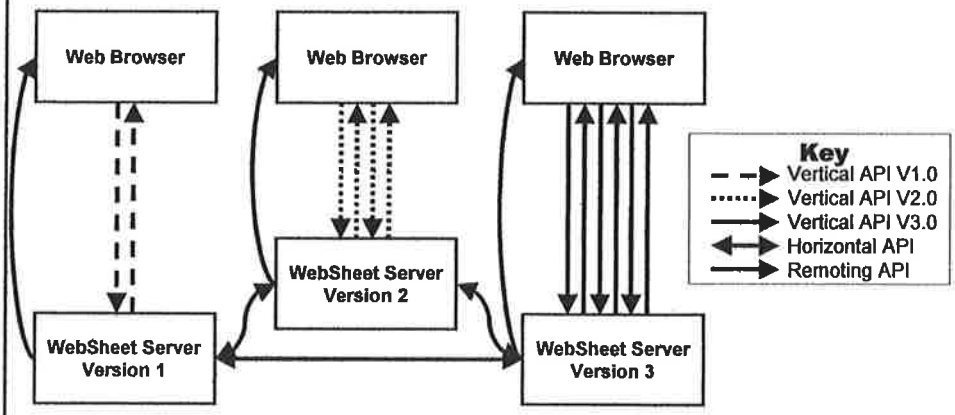
You can use a whole range of formulas (also available in Microsoft Excel!) or write your own server-side one

abs/1,
acos/1,
acosh/1,
address/5,
andf/1,
asc/1,
areas/1,
asin/1,
asinh/1,
atan/1,
atan2/2,
atanh/1,
avedev/1,
average/1,
averagea/1,
betadist/5,
betainv/5,
binomdist/4,
calculate/2,
ceiling/2,
cell/2,
char/1,
chidist/2,
chiinv/2,
chitest/2,
choose/1,
clean/1,
code/1,
column/1,
columns/1,
combin/2,
concatenate/1,
confidence/3,
correl/2,
cos/1,
cosh/1,
count/1,
counta/1,
countblank/1,
countif/2,
covar/2,
critbinom/3,
date/3,
datevalue/1,
day/1,
days360/3,
daverage/4,
db/4,
db/5,
dcount/4,
dcounta/4,
ddb/4,
ddb/5,
devsq/1,
dget/4,
dydx/2,
days360/2,
error_type/1,
even/1,
exact/2,
exp/1,
expondist/3,
fact/1,
false/0,
fdist/3,
find/2,
find/3,
findb/2,
findb/3,
finv/3,
fisher/1,
fisherinv/1,
fixed/1,
fixed/2,
fixed/3,
floor/2,
forecast/3,
frequency/2,
ftest/2,
fv/3,
fv/4,
fv/5,
gammadist/4,
gammainv/3,
gammaln/1,
geomean/1,
gestep/1,
gestep/2,
growth/4,
harmean/1,
hour/1,
hypgeomdist/4,
if/3,
index/4,
indirect/1,
int/1,
integrate/2,
intercept/2,
ipmt/4,
ipmt/5,
ipmt/6,
irr/1,
irr/2,
iserr/1,
iserror/1,
is_list_number/1,
islogical/1,
isna/1,
isnontext/1,
isnumber/1,
ispmt/4,
istext/1,
jis/1,
kurt/1,
large/2,
left/1,
left/2,
leftb/1,
leftb/2,
len/1,
lenb/1,
linest/2,
linest/3,
ln/1,
log/1,
log/2,
log10/1,
lower/1,
max/1,
maxa/1,
mdeterm/1,
median/1,
mid/3,
midb/3,
min/1,
mina/1,
minute/1,
minverse/2,
mmult/4,
mod/2,
mode/1,
month/1,
n/1,
na/0,
not/1,
normdist/4,
normsdist/1,
now/0,
odd/1,
or/1,
permut/2,
pi/0,
pound/1,
pound/2,
power/2,
product/1,
radians/1,
rand/0,
replace/4,
rept/2,
right/1,
right/2,
rightb/1,
rightb/2,
round/2,
rounddown/2,
roundup/2,
search/2,
search/3,
searchb/2,
searchb/3,
second/1,
sign/1,
sin/1,
sinh/1,
small/2,
sqrt/1,
standardise/3,
stdev/1,
stdevp/1,
substitute/3,
substitute/4,
sum/1,
sumproduct/2,
sumsq/1,
tan/1,
tanh/1,
today/0,
trim/1,
true/0,
type/1,
upper/1,
value/1,
var/1,
varp/1,
year/1,
cumulate/1,
percentile/2,
proper/1,
quartile/2,
rank/2,
rank/3,
rows/2,
pearson/2,
moment/2,
skew/1,
slope/2,
steyx/2,
stdeva/1,
subtotal/2,
sumif/2,
sumif/3,
sumx2my2/2,
sumx2py2/2,
sumxmy2/2,
trend/3,
trend/4,
transpose/2,
trimmean/2,
trunc/1,
trunc/2,
vara/1,
varpa/1,
weibull/4

You can address multiple cells in a variety of different ways using URL's as well and these URL's will return lists that can be directly used in formulae

`proto://sub.dom.tld/page/path/ab12:cd34`	a range
`proto://sub.dom.tld/page/path/ab`	a whole column
`proto://sub.dom.tld/page/path/12`	a whole row
`proto://sub.dom.tld/page/path/ab23:cd34,x1:y3`	a union
`proto://sub.dom.tld/page/path/ab23:cd34,x1:y3`	an intersection

There are three substantial API's in the product – all with different characteristics - which are required for different purposes



The horizontal API is how one cell establishes a relationship with another cell on a different server

Request to set up a hypernumber link:

```
protocol://subbies.domain.tld/path/to/a/page/ab23
with    action=register
        registered_URL=the URL of the cell to be updated
        proxy_URL=the URL that the notification is posted to
        biccie=a 'biccie'
```

Notify a registered cell that the value has changed:

```
protocol://proxy.domain.tld/path/to/a/page/ab23
with    action=notify
        type=change
        notifying_URL=the URL of the cell(s) that are notifying
        registered_URL=the URL of the cell to be updated
        value=fully qualified hypernumber
        biccie=a 'biccie'
        version=page_version
```

...and the horizontal API also includes a small number of structural elements that are required to make the system work...

- Notification of structural changes
  - "I have deleted this column on this page"
  - "I have inserted a 3x5 block at this cell"
  - "I have deleted the name 'sales'"
- These require remote sites to rewrite their formulae or mark their values as `#undefined`

...the horizontal API is entirely `POST` based and has three core 'verbs' that provide it with structure...

`register`	set up hypernumber links
`unregister`	tear down hypernumber links
`notify`	notify remote sites of value/structural changes

The vertical API delivers functionality to a particular user or set of users and is a mixture of POST and GET based

Read
- Returns something
- `GET`

Create/Update
- Creates or updates a value in a cell or cells or a named range
- `POST` `action=create`

Clear
- Clears the values in a cell or cells, a named set of cells or a page
- `POST` `action=clear`

Insert
- Inserts new cells causing reorganisation of the grid
- `POST` `action=insert`

Delete
- Deletes cells causing reorganisation of the grid – or deletes a named set of cells (leaving the cells themselves unchanged)
- `POST` `action=delete`

The meat of the read component is provided by decorating the URL's with appropriate queries

- Get a cell formatted as a hypernumber

```
protocol://subbies.domain.tld[:port]/path/to/a/page/a?hypernumber
```

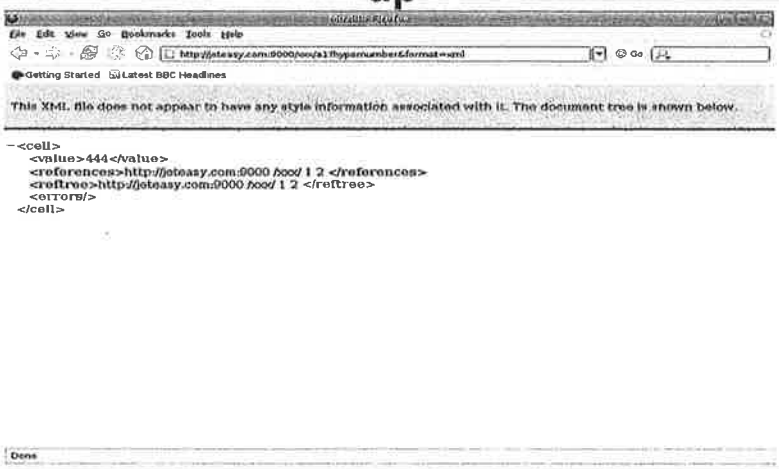
- Get the information required to populate the toolbar for a cell with focus:

```
protocol://subbies.domain.tld[:port]/path/to/a/page/ab23?toolbar
```

- Get details of the cells a particular cell is link_to or linked_from returning the result in a particular format:

```
protocol://sub.domain.tld/page/path/ab23?[link_to|link_from],format=[xml|json]
```

...the hypernumber format is simply a list showing the current value of the cell, the reference tree (used to check for circular references) and any errors that have bubbled up



...the toolbar wants to know where the current cell takes values from and where its value is used, so that you can browse from one page to another – as well as other details

...the remoting API is socket based – a GUI (or a chart) registers that it is looking at a section of the matrix and it can then be notified in real-time of changes to the underlying data

The client uses three verbs:

`authenticate` *`cookie`* **uses the http cookie**

`register proto://sub.dom.tld/some/page/a1:j9`

`unregister proto://sub.dom.tld/some/page/a1:j9`

The server responds with one verb:

`change proto://sub.dom.tld/some/page/a1 99`

...and there are a number of possible extensions – we welcome your views and ideas on these – they include:

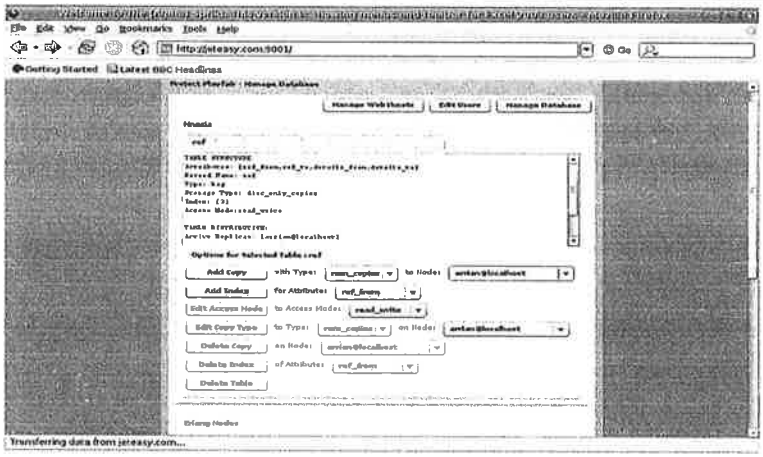
Summary/query URLs:

- `proto://sub.dom.tld/expenses/gordon/2007/Aug`
- `proto://sub.dom.tld/expenses/*/2007/*/sum(name(total))`

Negative, complex and floating cell addresses:

- `proto://sub.dom.tld/page/path/aa.bc+ix-33.3+i5`

...and there are a number of different components of the product that are of interest – in particular the Mnesia management panel...



...but also the delivery platform which is a customised version of Ubuntu with a single-click installer...

- based on Dapper Drake 6.06 (LTS)
- customised with the Ubuntu Customisation Kit (uck)
- basic platform will include an Erlang distribution system (eg CEAN)
  - Ideally with split repositories so that our application can be updated from within the general Erlang distribution
- to be open-sourced

what we have been working on

ADOBE FLEX

Flex is a mark-up language that compiles to Action Script 3.0 – which is an ECMA-262 compliant scripting language – it is a close cousin of Javascript and quite recognisable

Styles:

```
DataGrid {
   backgroundColor: #ffffff;
   horizontalGridLines: true;
   letterSpacing: 0;
   horizontalGridLineColor: #666666;
   useRollOver: false;
   rollOverColor: #666666;
   fontFamily: Tahoma;
   fontSize: 11; }
```

Functions:

```
private function loadLinksFrom(event:ResultEvent):void
{
   link_to.dataProvider = new Array();
   for each (var linksto:XML in event.result.link)
         link_to.dataProvider.addItem({label:linksto.site.toString()+
         linksto.path.toString()});
}
```

...but it is the collections of prebuilt components that can be configured through a mark up language that make it so powerful

```
<menuitem label="Formulas">
   <menuitem label="Maths" data="Maths">
        <menuitem label="Arithmetic">
                <menuitem label="sum" data="sum()"/>
                <menuitem label="product" data="product()"/>
                <menuitem label="abs" data="abs()"/>
                <menuitem label="sqrt" data="sqrt()"/>
                <menuitem label="power" data="power(,)"/>
                <menuitem label="sign" data="sign()"/>
                <menuitem label="exp" data="exp()"/>
                <menuitem label="fact" data="fact()"/>
                <menuitem label="mod" data="mod(,)"/>
        </menuitem>
   </menuitem>
</menuitem>
```

...just drop binary files onto the server docroot and configure the matrix server through a GUI and you are 'good to go'

- loose binary integration
  - very clean
  - allows a secondary market to develop
- powerful platform that is available on around 90% of desktops
- genuinely cross-platform
- socket and http-based connectivity
- well suited to a 'naïve' coding style (eg loads of concurrent connections to yaws)

What happens now

NEXT STEPS

...we are currently seedfunded through a pan-European VC-backed funding contest called Seedcamp but need to step up a notch...

Backers include:
- Eden
- Amadeus
- Venrex
- Atomico
- Northzone

Backers include
- Highland Capital Partners
- Index
- DJFEsprit
- Atlas Venture
- Benchmark Capital

...looking to raise Series 'A' and make some significant technical and non-technical hires in the near future...

- launch before Xmas
- go into Series 'A' funding
- make significant hires
  - including senior Erlang positions

1(6)

Integrating OTP with Enterprise Service Bus

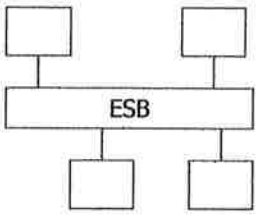
OAS (OTP Application Server)

Leslaw Lopacki
Telenor IS Nordic, Norway
leslaw.lopacki@telenor.com

Background – what is SOA?

What is Service Oriented Architecture?:

- Is a collection of services which communicate with each other
- Uses loosely-coupled relationships between producer and consumer
- Has no direct relationship with software, programming, or technology
- Services can be accessed without knowledge of their underlying platform implementation
- ORB/CORBA is probably a first SOA implementation
- Services are "usually" defined using WSDL/XSD
- Messages are "usually" SOAP/XML based but not only



2

What is OAS and OESB?

- OAS – OTP based Application Server:
  - Provides a simple framework for building Erlang based "Beans" – deployed as Erlang processes – 2 types supported:
    - Dynamic workers – started dynamically on demand
    - Static Workers – registered processes (Singletons)
  - Technically it isn't a JEE App Server – it simply does not support all JEE specs – but that's not an issue here
- OESB – Connects ESB and OAS:
  - uses SOAP 1.1 – ErlSoap 0.4.3

CoMet/Metro/COS at Telenor

How this idea was born?:

- Metro and COS are integration platforms with ca 50 systems (running at Telenor):
  - 2 different Application Servers: IBM WebSphere and BEA WebLogic
  - Close integration of JEE and .NET applications
- CoMet is a new integration platform which integrates Metro, COS:
  - Uses SOA Enterprise Service Bus from BEA
- Very limited/no use of Erlang:
  - Only isolated systems – e.g. Jabber server
- We could consider using OTP/Erlang in some installations if it was easier to integrate:
  - e.g. using SOAP/XSD/WSDL

Motivation for OESB

- Erlang integrates poorly with large enterprise architectures
- But thanks to SOA the systems can be more heterogeneous:
  - i.e. appearance of SOA makes it easier to integrate "small languages" (Erlang ...) with the "big ones" (JEE, .NET, ...)
- OTP is far more cost-efficient than any existing JEE Application Server, e.g.:
  - Built in efficient look up of resources – reduces "glue code"
  - Built in database: mnesia
  - Offensive approach: simpler "exception handling", less code needed to get things done
- SOA and:
  - OO does not integrate so well
  - FP seams to be a perfect match

Conclusion:

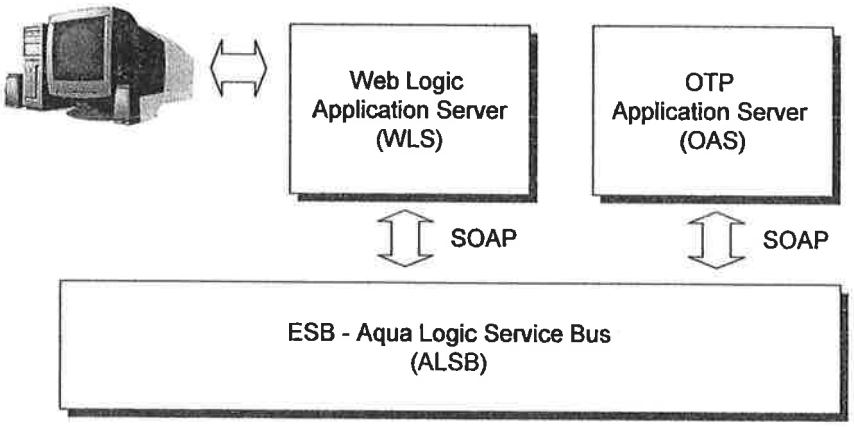
- SOA brings an opportunity to introduce OTP in existing installations

OTP's weaknesses from SOA point of view

- Good built in SOAP libraries are essential:
  - ErlSoap is rather limited – i.e. it does not support WSDL
  - Yaws contains some WSDL support (not evaluated here)
- Lack of infrastructure to quickly deploy applications – application servers
- Not optimal for "frontend" systems:
  - GUI design support is poor ! ☹
  - Poor support for Web Services and Web technologies

4

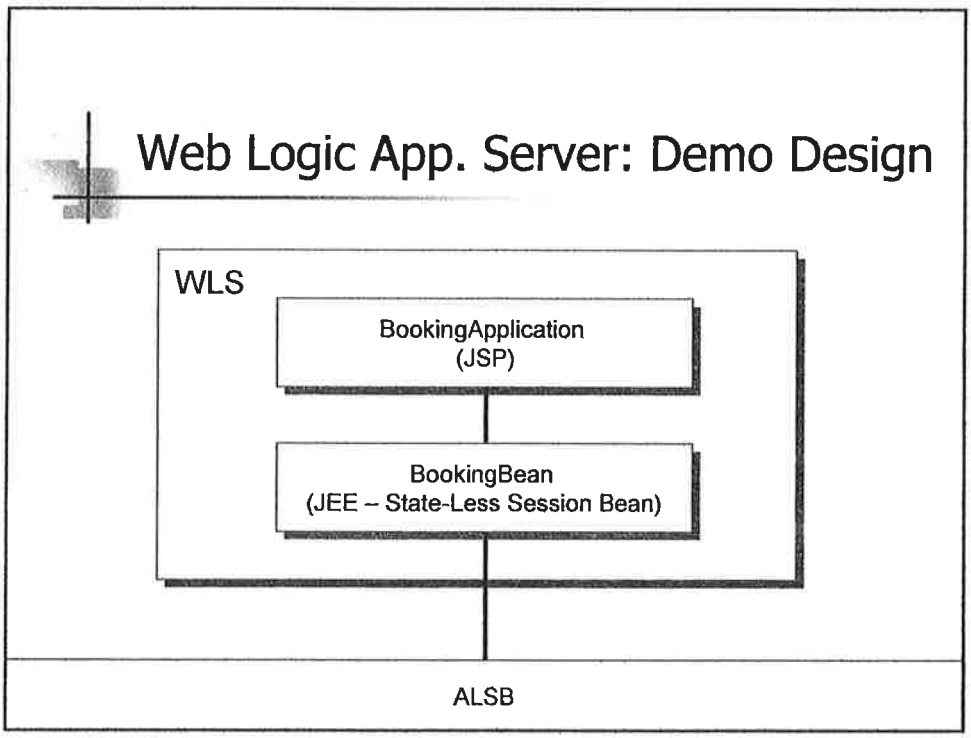
OESB Architecture

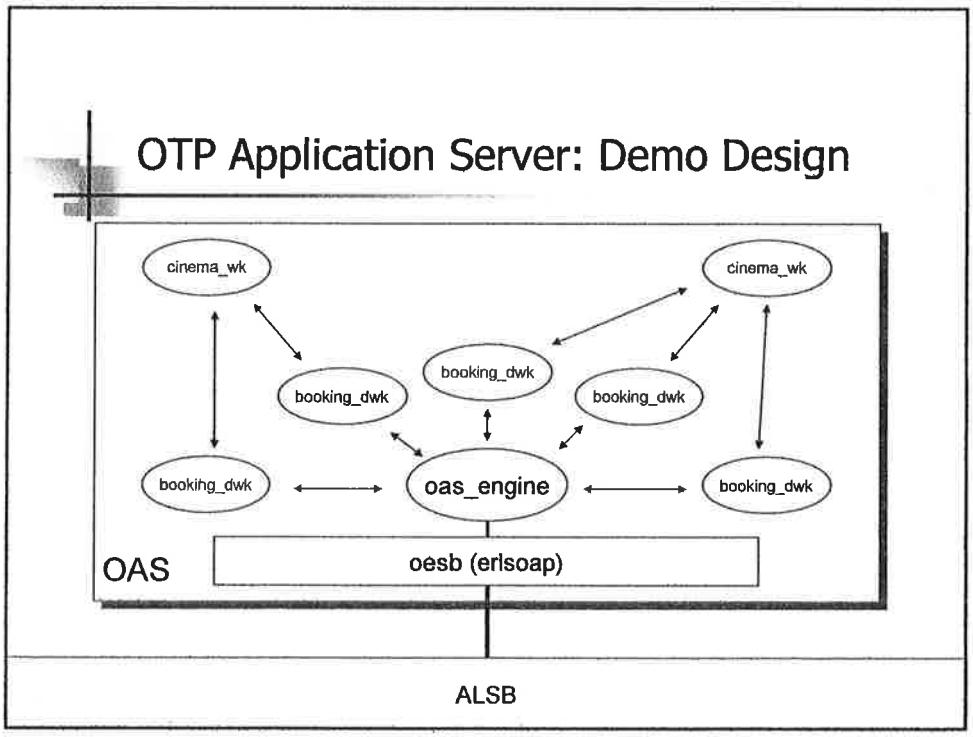


Demo system

- Demo of cinema booking system
- Exemplifies strong sides:
  - of an App Server:
    like Web GUI and integration with legacy JEE systems
  - of OTP/Erlang:
    like high transactional throughput, low latency reliable data storage, FSM support, reliability and scalability

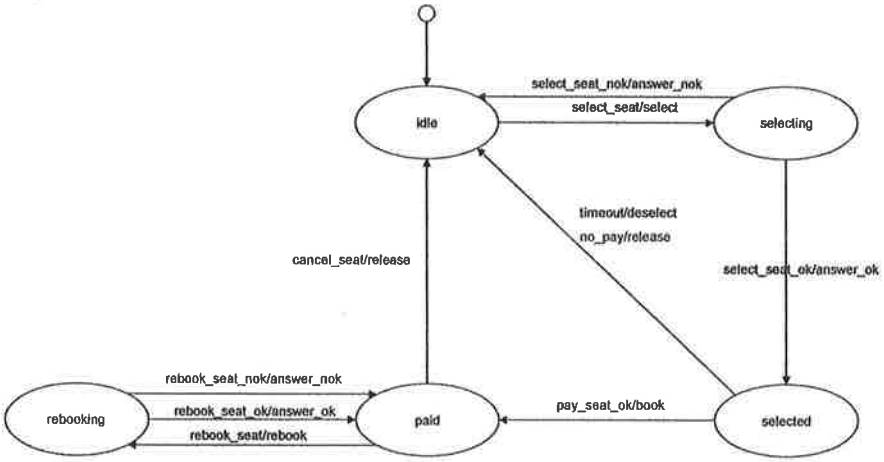
5


Web Logic App. Server: Demo Design
WLS
BookingApplication
(JSP)
BookingBean
(JEE – State-Less Session Bean)
ALSB


OTP Application Server: Demo Design
cinema_wk
cinema_wk
booking_dwk
booking_dwk
booking_dwk
booking_dwk
oas_engine
booking_dwk
OAS
oesb (erlsoap)
ALSB

6

FSM for booking_agent



OESB Enhancements (Future)

- Take full advantage of OTP
  - Make it scalable:
    - Load balancing support
  - Make it more reliable
    - Better redundancy/replication support

1 (16)

Towards Hard Real–Time Erlang

Vincenzo Nicosia [1]

[1] Dipartimento di Ingegneria Informatica e delle Telecomunicazioni
Università di Catania – Italy

Sixth ACM SIGPLAN Erlang Workshop ICFP 2007
5 Oct. 2007 – Frieburg

Outline

HRT systems: what since now ?

- Hard Real–Time (HRT) constraints are common in many application fields, such as:
  - Control systems (locomotion, security....)
  - Manifacturing
  - Signal processing
  - Telecom
- HRT application are often been developed using C, C++, Ada on top of RT operating systems
- Other “main–stream” languages, such as Java, approached the problem of RT only recently
- Nowadays, RT systems are quickly moving towards embedded architectures and solutions

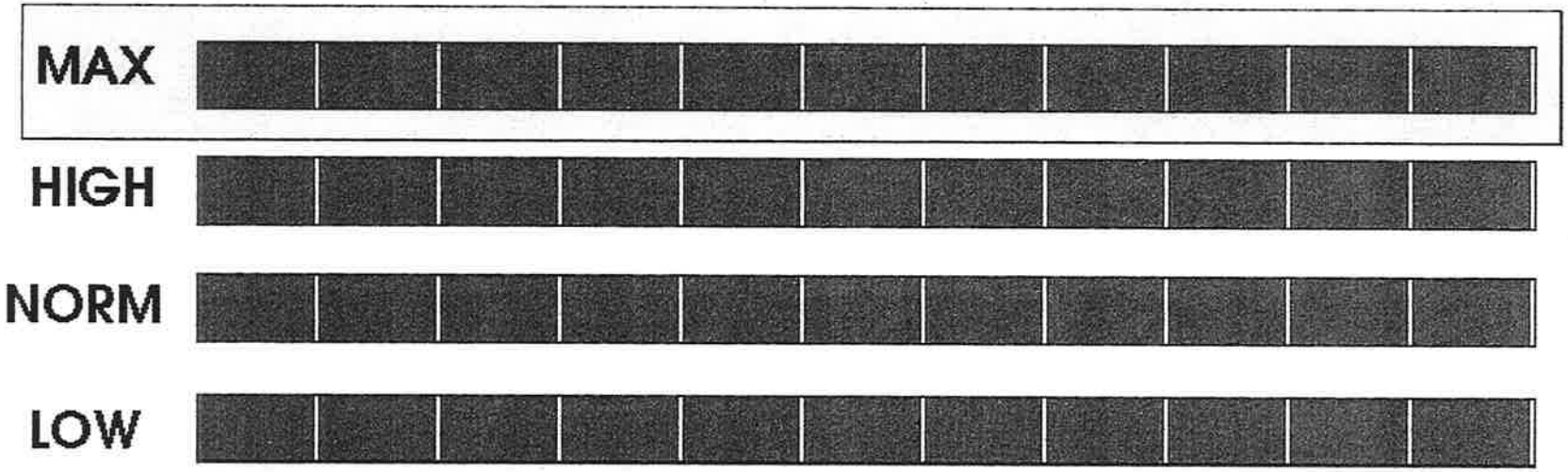
Functional programming for HRT: Erlang ?

- Functional programming paradigm can help a lot in modeling, defining, developing, testing and maintaining RT systems
- In particular, Erlang/OTP has been successfully used for massively concurrent soft real–time systems
- Erlang/OTP gives some basic functionalities that are really useful in developing RT systems:
  - A huge and complete standard library
  - OTP, which gives a lot of power and flexibility to manage large systems with a lot of cooperating processes even in distributed environments
  - The possibility of building and deploy embedded Erlang applications in an easy and reliable way
- We think that Erlang has much to say even in the field of HRT systems, but it lacks native HRT support!

The Actual Emulator Scheduler.....

- The Erlang scheduler does not have support for HRT tasks.
- It is a Multi–Queue Round–Robin scheduler
- There are basically three documented levels of priority for processes: low, normal and high. A fourth priority level (max) is undocumented and reserved for a couple of system processes.
- All user Erlang processes usually run with normal priority, and usage of different priority levels (expecially of high) is highly discouraged
- So the Erlang native scheduler cannot guarantee HRT:
  - No deadline specification for processes
  - No guarantees that a process would finally be scheduled (starvation problems arise using high and normal prio with strange spawning patterns.....)

Scheduler Queues

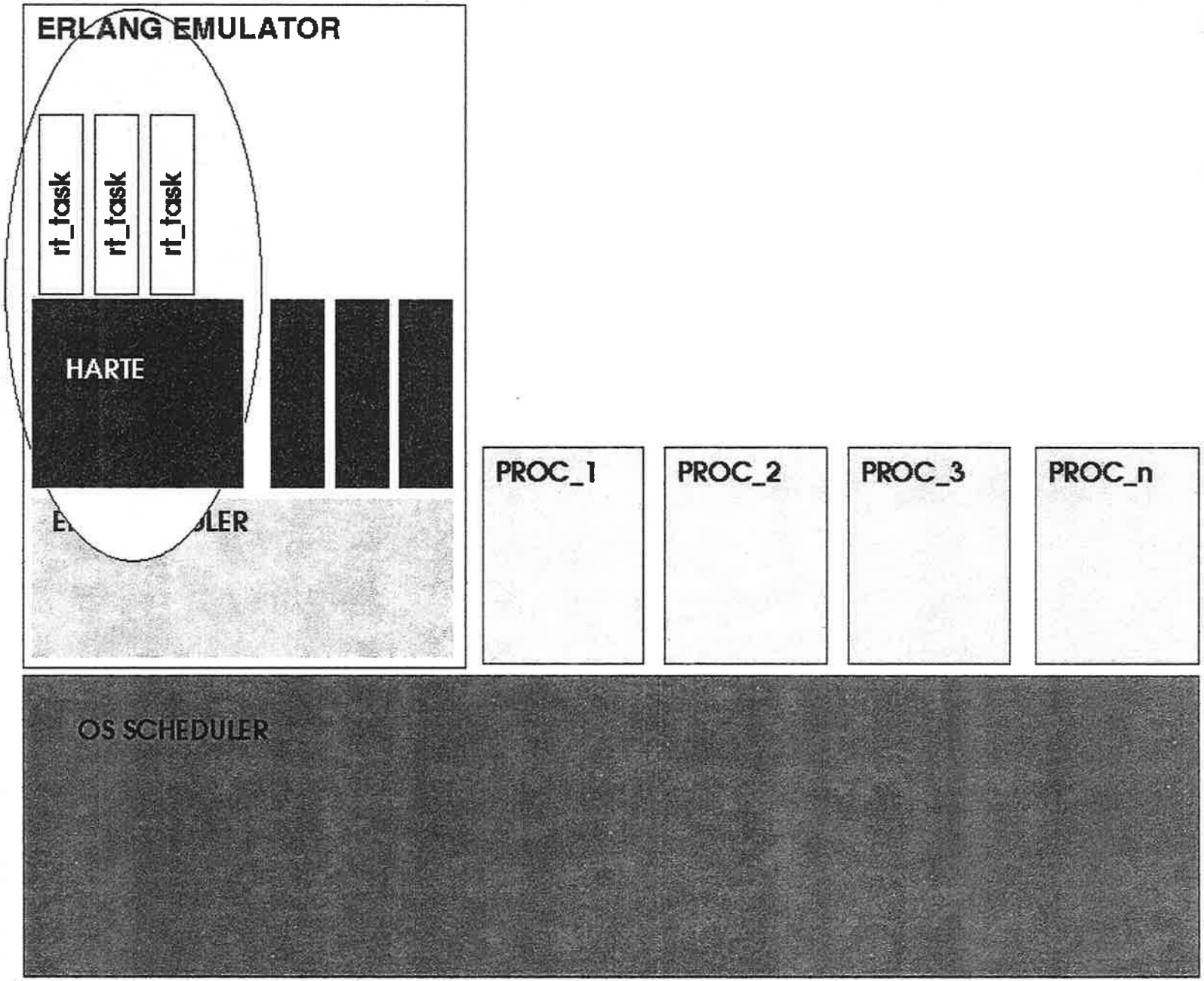


5

Towards RT Erlang processes

In order to have HRT capabilities in Erlang, three different approaches are possible:

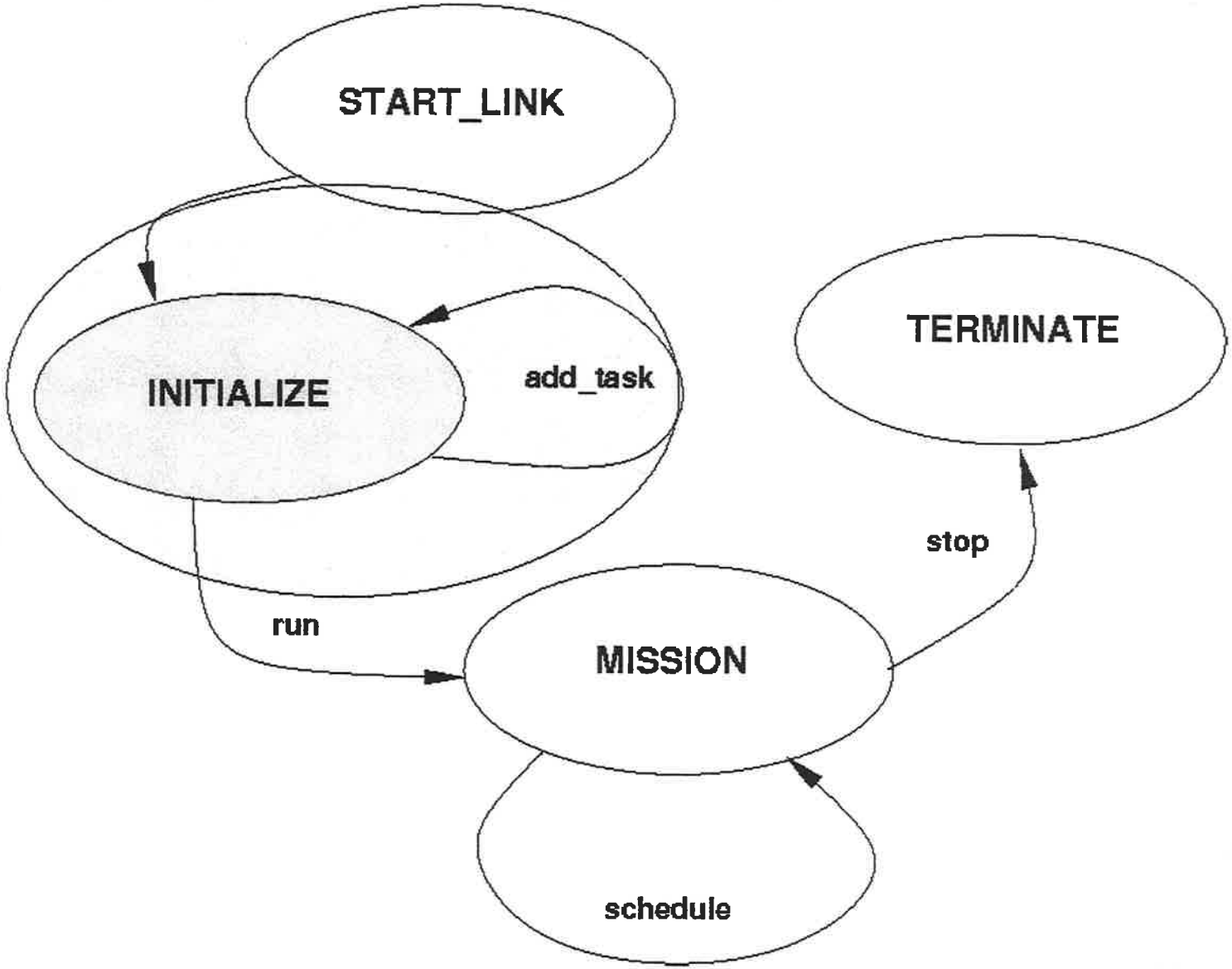
- Writing from scratch a new scheduler for the emulator
  Unfeasible: the scheduler is really entangled with much of the system Existing Erlang code should continue to work anyway
- Modifying the existing MQRR scheduler to support realtime
  Hard: a lot of C code to guarantee HRT
- Adding HRT as a service, which is an erlang application which provides HRT capabilities. This is what this paper is all about :-)


ERLANG EMULATOR
rt_task
rt_task
rt_task
HARTE
PROC_1
PROC_2
PROC_3
PROC_n
OS SCHEDULER

HARTE: an overview

- HARTE is basically an application (a peculiar one), which is in charge of scheduling RT task
- In order to guarantee RT scheduling of task, HARTE itself runs with MAX priority (*)
- All HARTE tasks (i.e. HRT tasks the user would run), are created as low priority tasks and put in a scheduling queue using a Deadline Monotonic (DM) scheduling algorithm
- Then the scheduler is started and it schedules tasks one by one, modifying the priority of the task to be run to HIGH (*)

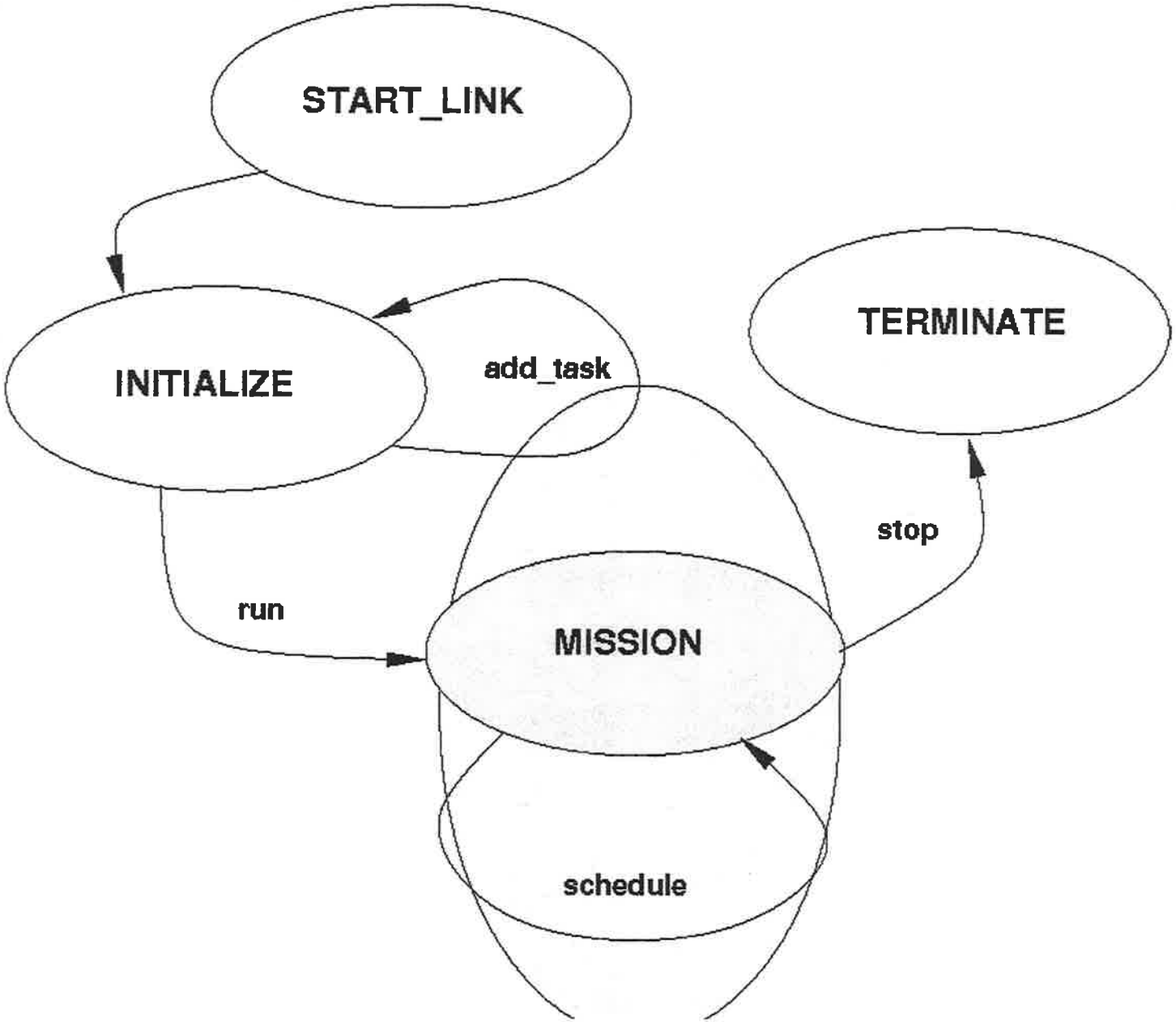
Details: init



Initialisation

- A new behaviour called rt_fsm has been defined. It is basically a gen_fsm with some additional code for RT management
- Each HRT task is represented by an rt_fsm
- In the initialisation phase, all RT tasks are defined and added to the scheduler
- To add a task to the scheduler, the init function of rt_fsm calls rt_scheduler:add_task(),

Details: Mission



Mission

- HARTE scheduler is started by calling rt_scheduler:run()
- From there on task to be run are picked up from the queue and scheduled.
- To schedule a task, we modify its priority from low to high
- In order to do that, the BIF process_flag/3, in order to let a process change the priority of another process to a level not higher than his own priority level
- Note: This trick is really dangerous, and can lead to weird situations, if misused.....or even if used... :-)

Tests

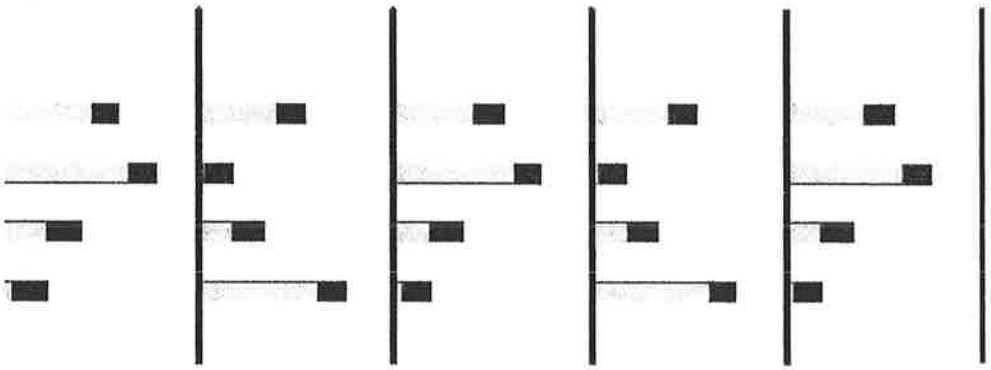
- HARTE is still a proof–of–concept but it seems to work!
- We tested it with a couple of heavy CPU–bound benchmark, running both HARTE tasks and normal erlang processes at the same time
- The scheduler overhead, in different configurations, is reported in table:

Test	Average Overhead (μs)	Std. dev.
RT tasks only	13.1	245.78
RT and Non–RT tasks	11.7	52.5
RT tasks many periods	17.9	132.05

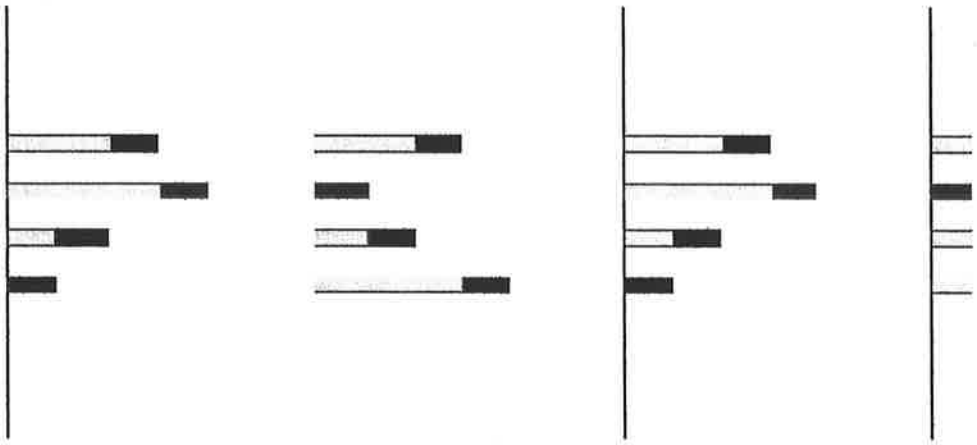
Figure: Average and standard deviation of scheduler overheads

14

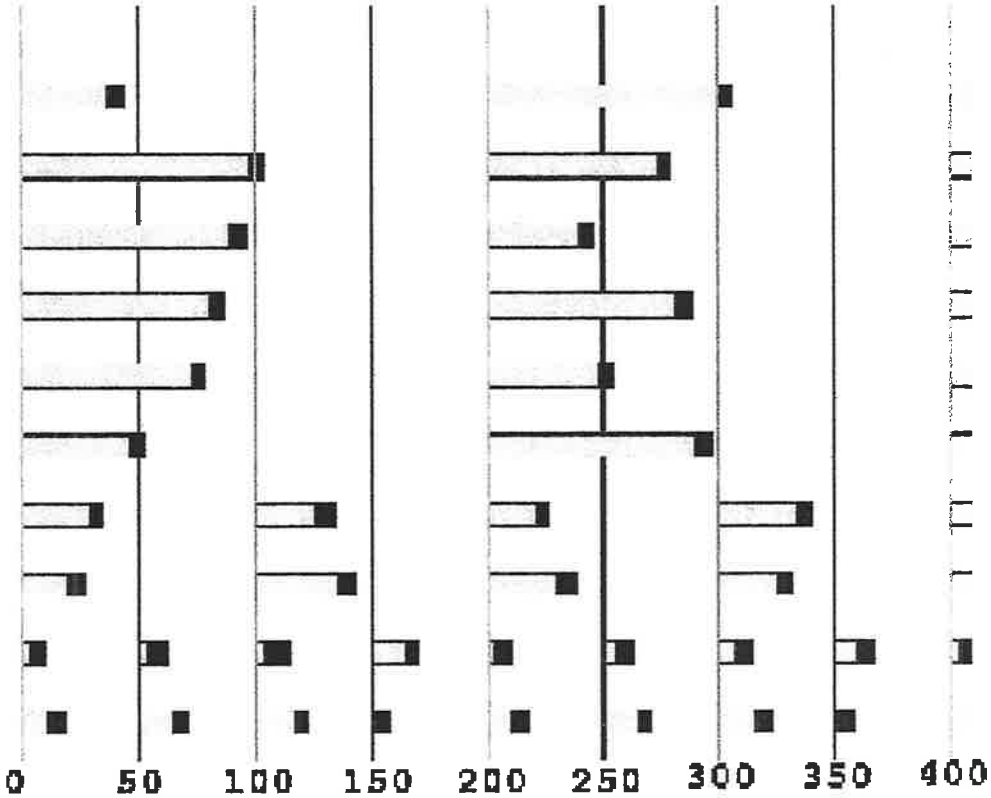
Test 1: four HRT tasks



Test 2: RT tasks and 50 normal erlang processes



Test 3: HRT tasks with different periods



Open Issues

- Erlang Garbage Collector is still a problem. Even if this approach seems to work, the actual GC used by the emulator is not predictable
- Support for real–time message passing has to be introduced (a preliminary solution exists!)
- An EDF scheduling policy should be adopted and becomes cumpolsory when you have RT message passing
- HARTE code needs a bit of refactoring: it was written in the time of two nights.....(those bloody two nights when the flame about priorities exploded on erlang-questions ML) and you can imagine how much it can be improved

Property-based Testing

A STREP proposal for FP7 ICT Call 1

Prepared by the Universities of Sheffield, Kent, Politécnica de Madrid, IT University of Göteborg and Chalmers together with Ericsson, Erlang Training and Consulting Ltd, Quviq AB and Lambda-stream

Concept and objectives

Communication networks, based on telephony, wireless and Internet, have over the last few years been converging. At the present time and for the foreseeable future, more and more services will be added to these merging networks. What is more, these services are becoming more complex, both in themselves and in their interactions with each other and their end users. The telecoms industry has an admirable record in providing reliability and robust services to its clients, and indeed it is the telecoms industry that can point to **5-nines reliability**: that is 99.999% reliability, of their core systems.

The problem addressed in this proposal is that of maintaining 5-nines reliability in future service-oriented networks and systems.

The software for new services and network devices is rapidly growing in complexity, among other things because of the variety of formats and multiplicity of delivery modes evident in modern communication protocols (with thousands of optional fields, for instance). In addition, such software needs to be context-aware, since the requirements vary when the same software is used in different ways. There are several ingredients for ensuring that such complex systems provide the expected reliability, among them choosing a good architecture, using the right technologies, improving the software process, and also being extremely thorough and efficient in testing.

This proposal aims to support the European software industry in its testing methodology for software in network and service infrastructures.

Testing of complex systems is difficult and time-consuming in the extreme, a fact which companies such as Ericsson are well aware of. This indicates that radical approaches are both needed, and will be welcomed by industry—if they work! In this proposal we build upon the innovative idea of using **properties as objects for testing software**. In order to deliver dynamic services and interoperable network applications with *guaranteed properties*, we focus testing around these properties.

We have shown in previous work that we can describe these properties in a concise way, and that we can automatically generate test cases from them. We have carried out industrial case studies of network applications, and shown that random sequences of API calls generated from our properties can test much more, much more quickly, than can traditional, manually constructed test cases. The vast number of formats and options that these modern service-oriented applications support cannot efficiently be dealt with through a manual approach.

Property-based testing will deliver more effective tests, more efficiently, and thus deliver economic benefits to the European software industry.

Properties as objects for testing have only recently been introduced in an industrial setting. From this introduction we learned that this technology is an enabler for other tools and methods to test better and faster. There are many techniques that can be used to create and maintain properties, which were not in reach before.

Testing with properties as objects instead improves the competitiveness of software developers, since they can deliver higher quality software for a lower price. It also allows collaborating companies to improve the definition of their software interfaces and therewith improve the compatibility between their services.

The project will deliver methods and tools to support property-based development of systems.

Property-driven development is a powerful new mechanism for gaining assurance of system reliability and functionality. However, in order to deliver its full benefits we need tools to integrate property-based testing into the development life cycle.

Property discovery. Current testing is based on sets of test cases embedded in test suites; we will provide tools to aid the software developers to extract properties from this test data. Current specifications and models are often informal: we will develop specialised property languages to ease the formalisation of existing specifications.

Test and property evolution. All software systems are subject to change and evolution; we will provide tools to support the evolution of tests and properties in line with the evolution of the system itself.

Property monitoring. Not all properties can be tested in advance of systems being executed; not all faults will be found during testing, be it ever so thorough. We will also provide tools to support the *post hoc* examination of trace details for conformance to (or indeed violation of) particular constraints.

Analysing concurrent systems. At the heart of service oriented systems is *concurrency*: servers will provide services to multiple clients in a seamlessly concurrent way; services will federate to provide complex functionality through concurrently performing parts of a task. We will provide tools by which such concurrent systems can be analysed for fundamental properties.

The project will put its deliverables to the test by putting them into practice.

The property-driven development methodology will be tested in practice in an industrial/ academic partnership to build a substantial case study in an industrial context, reflecting on the tools and method developed, and feeding back into the project itself.

The consortium will build on a strong software development platform.

The aim of the project is to introduce property-driven development into the software engineering process. Property-driven development can be used in a variety of programming languages and systems. The particular platform chosen for initial implementation of the project is Erlang/OTP (Open Telecom Platform), but a crucial aspect of our proposal is the dissemination and adoption of the approach much more widely, particularly into the model-driven development arena (UML) and other implementation languages (C/C++, Java, etc).

Erlang/OTP has been chosen as the implementation vehicle because of its robustness and reliability within the telecoms sector; witness, for example, its success in the implementation of the AXD301 ATM telecoms switch by Ericsson, one of the project partners. Erlang is a practical language, designed from the start with practical application in mind. It also benefits from simplicity, and from being a functional language, which eases the application of theoretical results from the academic programming language community. We see Erlang as a natural common ground between researchers and the telecoms industry, providing a conduit through which research results can be quickly transferred to industrial applications, and thence into a wider industrial context. We see precisely this happening in this project.

The consortium has the right combination of skills to deliver results.

The consortium is a balance of academics, SMEs, and a key larger industrial enterprise. The academics bring experience of testing, formal verification, language development, and refactoring support, delivered in a number of successful national research projects. Quviq are a spin-off from academia, founded to commercialise an innovative property-based testing tool. The remaining industrial partners are system builders (Ericsson, LambdaStream), consultants, and trainers (ETC), who will provide invaluable insights into what is required of practical tools, what properties will need to be checked, and ways of fitting the results from the project into practical software development methods.

The project addresses Objective ICT-2007.1.2: Service and Software Architectures, Infrastructures and Engineering, (b) Service/software engineering approaches.

The project provides a development process and tools that ensure dependable quality of service through directly verifying properties of the systems. In the context of Erlang/OTP, and Open Source systems, this process and the tools will integrate with open development paradigms, and the Erlang/OTP develop community will be addressed directly through the project partners and indirectly through training and open source take-up.

The principal impact of the project will be to allow software developers to bring to market more reliable products on a shorter timescale. Thus, their profitability will be improved, and with that comes an increased ability to compete effectively in a global industry. Also, increased reliability offers the opportunity of building more complex assemblies of systems from assured components.

These innovations will help to nurture the European software service-provider sector, and help it to compete effectively on the global stage. Initial developments will be in

the Erlang/OTP sector, since Erlang provides the platform for the project, but property-based testing in UML and other implementation languages will follow by means of dissemination through Ericsson and its partners.

In summary, the proposal offers a focussed programme of research, based on a novel testing methodology, delivering tools and techniques to the European software sector, large and small, to enable the more efficient development of reliable software services.

For further information contact:
John Derrick (J.Derrick@dcs.shef.ac.uk),
Simon Thompson (S.J.Thompson@kent.ac.uk), or
Thomas Arts (thomas.arts@ituniv.se).

News from the Erlang/OTP Development team at Ericsson

ERICSSON
TAKING YOU FORWARD

What's new in R12B

Runtime system and compiler

- New BIF's tuple_size/1, byte_size/1, bit_size/1
- Binary Comprehensions
- Bitstring + performance improvements for both construction an matching of binaries and bitstrings
- Constant Pool (the compiler now builds constants and put them in a constant pool shared by all processes)
- Percept – a new application for profiling of parallelism on application level.
- Support for SMP (Symmetrical Multiprocessor) on Windows.
- Yecc – produces more compact code which also execute faster than before.
- process_info(
- Enhanced error messages in the Shell
- New module **array** with support for non destructive arrays.
- Several new features in Dialyzer (support for contracts, easier to built PLT's etc)
- Driver_interface with support for creation of threads.

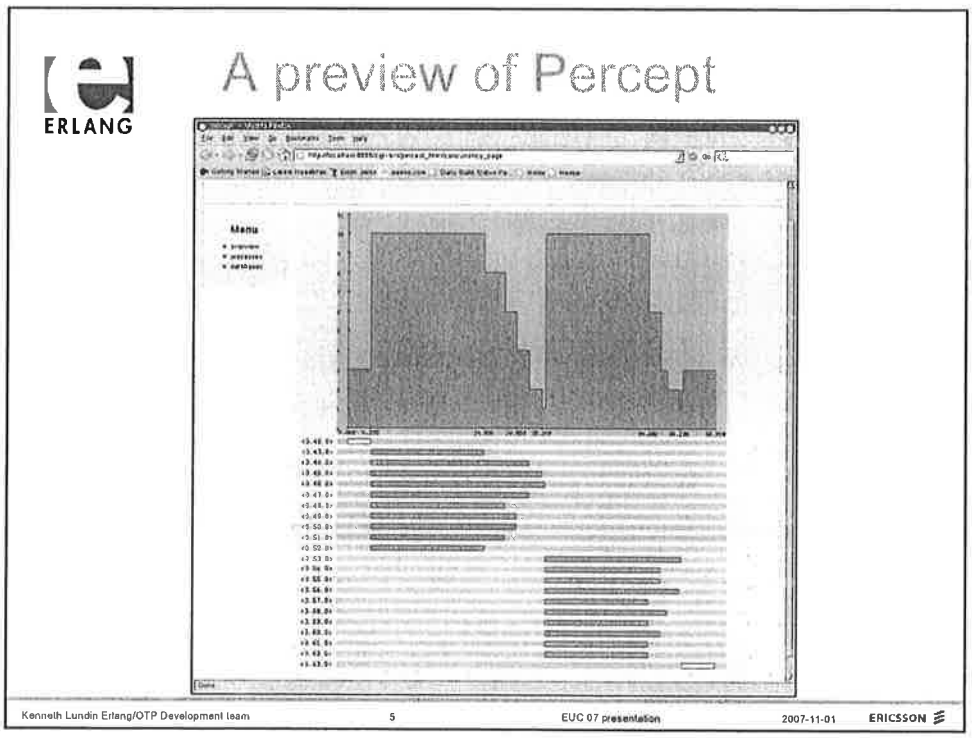
What's new in R12B continued

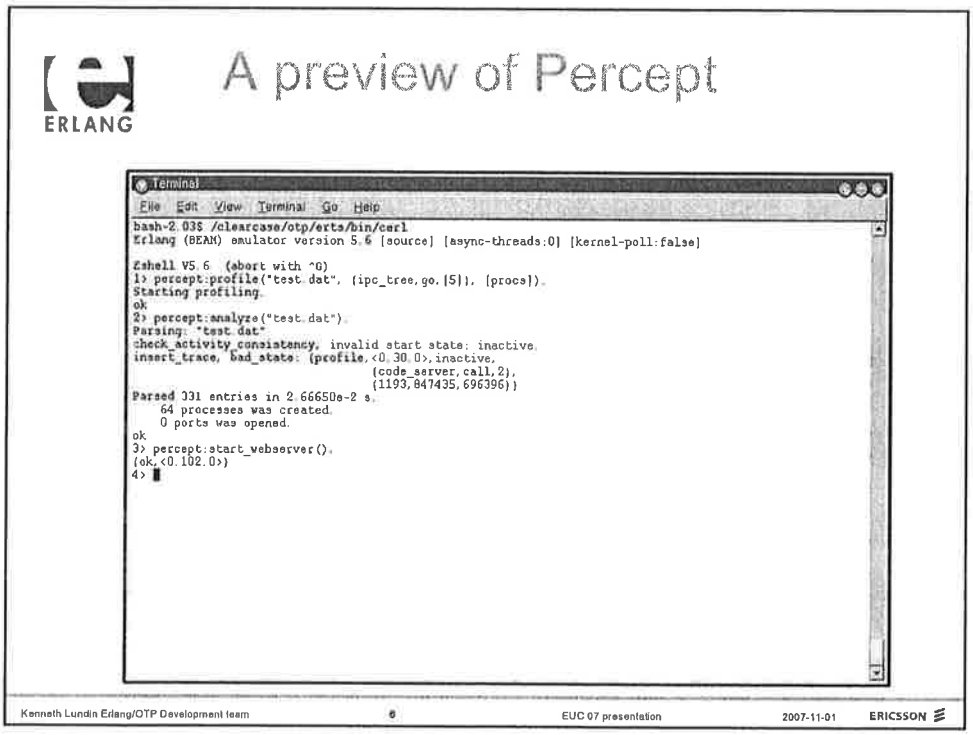
Applications

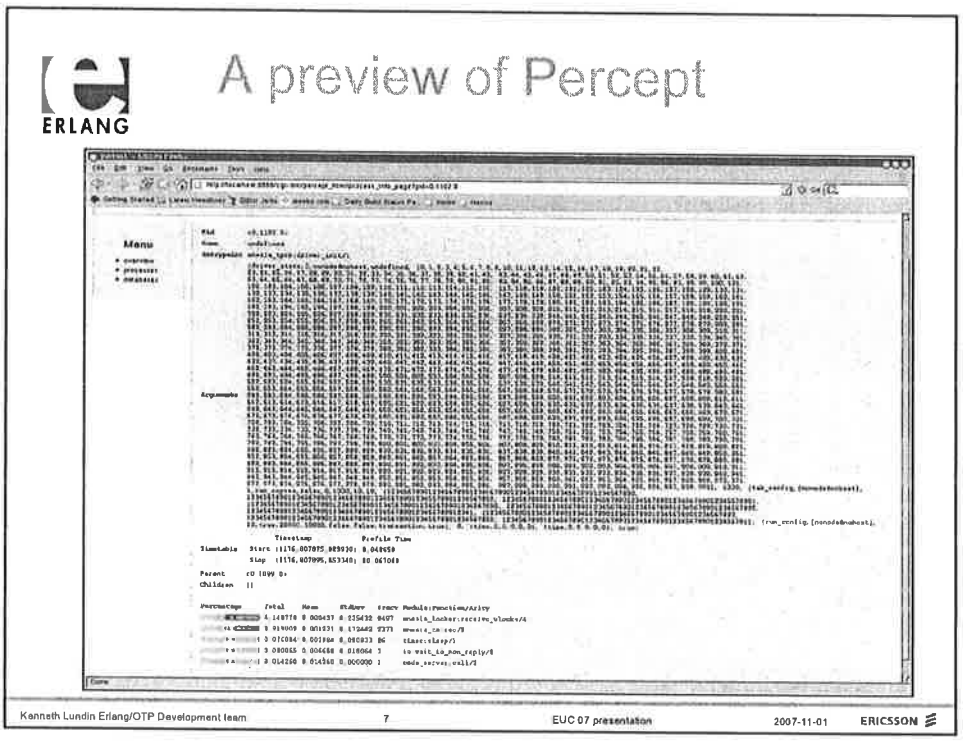
- New SSL implementation written in "pure" Erlang (alfa status) (Old SSL is also delivered)
- Full support for version 3 of H.248 (Megaco)
- Faster ASN.1 (PER) decode thanks to the new bitstring support.
- Hooks for load regulation in SNMP agent
- Inets (HTTP server) more flexible start and config
- CommonTest and TestServer part of the delivery

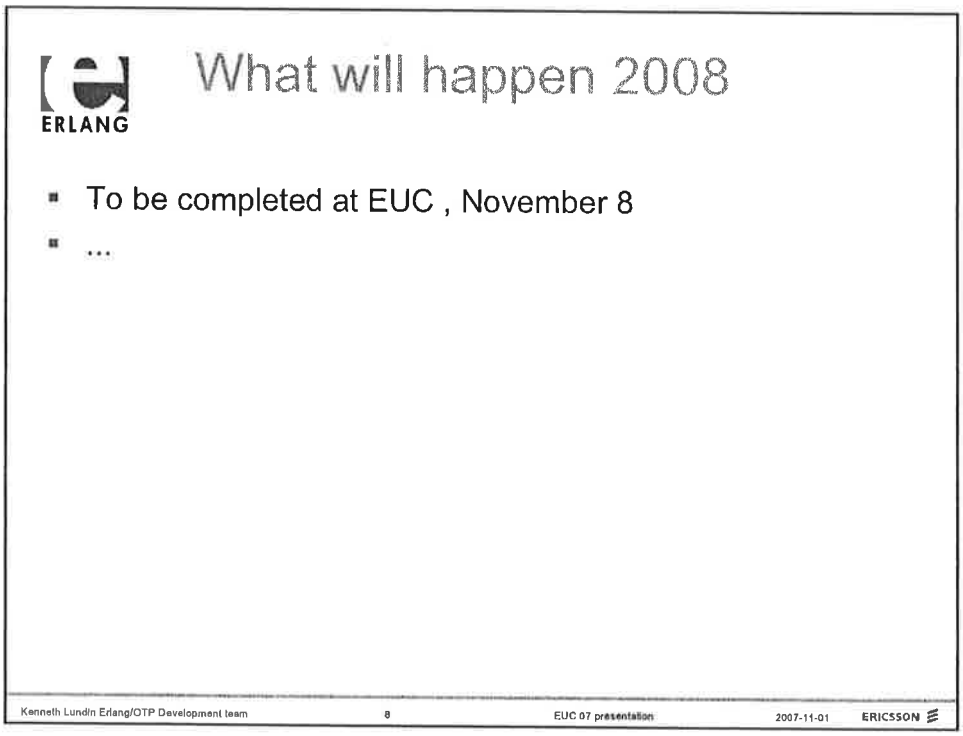
A preview of Percept

- Percept is an application level profiler with focus on parallelism.
- Makes use of new trace points in the virtual machine.
- Collects data about when processes are runnable and waiting.
- Graphical interactive presentation of collected data.


ERLANG
A preview of Percept
Menu
Kenneth Lundin Erlang/OTP Development team
5
EUC 07 presentation
2007-11-01
ERICSSON


ERLANG
A preview of Percept
Terminal
File Edit View Terminal Go Help
bash-2.03$ /clearcase/otp/erts/bin/cerl
Erlang (BEAM) emulator version 5.6 [source] [async-threads:0] [kernel-poll:false]
Eshell V5.6 (abort with ^G)
1> percept:profile("test.dat", {ipc_tree,go,[5]}, [procs]).
Starting profiling.
ok
2> percept:analyze("test.dat").
Parsing: "test.dat"
check_activity_consistency, invalid start state: inactive.
insert_trace, bad_state: {profile,<0.30.0>,inactive,
{code_server,call,2},
{1193,847435,696396}}
Parsed 331 entries in 2.66650e-2 s.
64 processes was created.
0 ports was opened.
ok
3> percept:start_webserver().
{ok,<0.102.0>}
4>
Kenneth Lundin Erlang/OTP Development team
6
EUC 07 presentation
2007-11-01
ERICSSON


ERLANG
A preview of Percept
Menu
Kenneth Lundin Erlang/OTP Development team
7
EUC 07 presentation
2007-11-01
ERICSSON


ERLANG
What will happen 2008
To be completed at EUC , November 8
...
Kenneth Lundin Erlang/OTP Development team
8
EUC 07 presentation
2007-11-01
ERICSSON

Non-Destructive Arrays

Richard Carlsson
and
Dan Gudmundsson

array

Erlang Module

Functional, extendible arrays. Arrays can have fixed size, or can grow automatically as needed. A default value is used for entries that have not been explicitly set.

Unless specified by the user when the array is created, the default value is the atom `undefined`. There is no difference between an unset entry and an entry which has been explicitly set to the same value as the default one (cf. reset/2 [page 59]). If you need to differentiate between unset and set entries, you must make sure that the default value cannot be confused with the values of set entries.

The array never shrinks automatically; if an index `I` has been used successfully to set an entry, all indices in the range [0,`I`] will stay accessible unless the array size is explicitly changed by calling resize/2 [page 60].

Examples:

```
%% Create a fixed-size array with entries 0-9 set to 'undefined'
A0 = array:new(10).
10 = array:size(A0).

%% Create an extendible array and set entry 17 to 'true',
%% causing the array to grow automatically
A1 = array:set(17, true, array:new()).
18 = array:size(A1).

%% Read back a stored value
true = array:get(17, A1).

%% Accessing an unset entry returns the default value
undefined = array:get(3, A1).

%% Accessing an entry beyond the last set entry also returns the
%% default value, if the array does not have fixed size
undefined = array:get(18, A1).

%% "sparse" functions ignore default-valued entries
A2 = array:set(4, false, A1).
[{4, false}, {17, true}] = array:sparse_to_orddict(A2).

%% An extendible array can be made fixed-size later
A3 = array:fix(A2).

%% A fixed-size array does not grow automatically and does not
%% allow accesses beyond the last set entry
{'EXIT',{badarg,_}} = (catch array:set(18, true, A3)).
{'EXIT',{badarg,_}} = (catch array:get(18, A3)).
```

DATA TYPES

`array()` A functional, extendible array. The representation is not documented and is subject to change without notice. Note that arrays cannot be directly compared for equality.

Exports

`default(Array::array()) -> term()`

Returns the value used for uninitialized entries.

See also: new/2 [page 59].

`fix(Array::array()) -> array()`

Fixes the size of the array. This prevents it from growing automatically upon insertion; see also set/3 [page 60].

See also: relax/1 [page 59].

`foldl(Function, InitialAcc::term(), Array::array()) -> term()`

Types:

- Function = (Index::integer(), Value::term(), Acc::term()) -> term()

Folds the elements of the array using the given function and initial accumulator value. The elements are visited in order from the lowest index to the highest. If `Function` is not a function, the call fails with reason `badarg`.

See also: foldr/3 [page 57], map/2 [page 58], sparse_foldl/3 [page 60].

`foldr(Function, InitialAcc::term(), Array::array()) -> term()`

Types:

- Function = (Index::integer(), Value::term(), Acc::term()) -> term()

Folds the elements of the array right-to-left using the given function and initial accumulator value. The elements are visited in order from the highest index to the lowest. If `Function` is not a function, the call fails with reason `badarg`.

See also: foldl/3 [page 57], map/2 [page 58].

`from_list(List::list()) -> array()`

Equivalent to from_list([], undefined) [page 57].

`from_list(List::list(), Default::term()) -> array()`

Converts a list to an extendible array. `Default` is used as the value for uninitialized entries of the array. If `List` is not a proper list, the call fails with reason `badarg`.

See also: new/2 [page 59], to_list/1 [page 61].

`from_orddict(Orddict::list()) -> array()`

Equivalent to from_orddict([], undefined) [page 58].

`from_orddict(List::list(), Default::term()) -> array()`

Converts an ordered list of pairs `{Index, Value}` to a corresponding extendible array. `Default` is used as the value for uninitialized entries of the array. If `List` is not a proper, ordered list of pairs whose first elements are nonnegative integers, the call fails with reason `badarg`.

See also: new/2 [page 59], to_orddict/1 [page 61].

`get(I::integer(), Array::array()) -> term()`

Returns the value of entry `I`. If `I` is not a nonnegative integer, or if the array has fixed size and `I` is larger than the maximum index, the call fails with reason `badarg`.

If the array does not have fixed size, this function will return the default value for any index `I` greater than `size(Array)-1`.

See also: set/3 [page 60].

`is_array(X::term()) -> bool()`

Returns `true` if `X` appears to be an array, otherwise `false`. Note that the check is only shallow; there is no guarantee that `X` is a well-formed array representation even if this function returns `true`.

`is_fix(Array::array()) -> bool()`

Returns `true` if the array has fixed size, otherwise `false`.

See also: fix/1 [page 57].

`map(Function, Array::array()) -> array()`

Types:

- Function = (Index::integer(), Value::term()) -> term()

Maps the given function onto each element of the array. The elements are visited in order from the lowest index to the highest. If `Function` is not a function, the call fails with reason `badarg`.

See also: foldl/3 [page 57], foldr/3 [page 57], sparse_map/2 [page 61].

`new() -> array()`

Creates a new, extendible array with initial size zero.

See also: new/1 [page 58], new/2 [page 59].

`new(Options::term()) -> array()`

Creates a new array according to the given options. By default, the array is extendible and has initial size zero. Array indices start at 0.

`Options` is a single term or a list of terms, selected from the following:

`N::integer()` or `{size, N::integer()}` Specifies the initial size of the array; this also implies `{fixed, true}`. If `N` is not a nonnegative integer, the call fails with reason `badarg`.

`fixed` or `{fixed, true}` Creates a fixed-size array; see also fix/1 [page 57].

`{fixed, false}` Creates an extendible (non fixed-size) array.

`{default, Value}` Sets the default value for the array to `Value`.

Options are processed in the order they occur in the list, i.e., later options have higher precedence.

The default value is used as the value of uninitialized entries, and cannot be changed once the array has been created.

Examples:

```
array:new(100)
```

creates a fixed-size array of size 100.

```
array:new({default,0})
```

creates an empty, extendible array whose default value is 0.

```
array:new([{size,10},{fixed,false},{default,-1}])
```

creates an extendible array with initial size 10 whose default value is -1.

See also: fix/1 [page 57], from_list/2 [page 57], get/2 [page 58], new/0 [page 58], new/2 [page 59], set/3 [page 60].

`new(Size::integer(), Options::term()) -> array()`

Creates a new array according to the given size and options. If `Size` is not a nonnegative integer, the call fails with reason `badarg`. By default, the array has fixed size. Note that any size specifications in `Options` will override the `Size` parameter.

If `Options` is a list, this is simply equivalent to `new([{size, Size} | Options]`, otherwise it is equivalent to `new([{size, Size} | [Options]]`. However, using this function directly is more efficient.

Example:

```
array:new(100, {default,0})
```

creates a fixed-size array of size 100, whose default value is 0.

See also: new/1 [page 58].

`relax(Array::array()) -> array()`

Makes the array resizable. (Reverses the effects of fix/1 [page 57].)

See also: fix/1 [page 57].

`reset(I::integer(), Array::array()) -> array()`

Sets entry `I` to the default value for the array. This is equivalent to `set(I, default(Array), Array)`, and hence may cause the array to grow in size, but will not shrink it. Shrinking can be done explicitly by calling resize/2 [page 60].

If `I` is not a nonnegative integer, or if the array has fixed size and `I` is larger than the maximum index, the call fails with reason `badarg`; cf. set/3 [page 60]

See also: new/2 [page 59], set/3 [page 60].

`resize(Array::array()) -> array()`

Changes the size of the array to that reported by sparse_size/1 [page 61]. If the given array has fixed size, the resulting array will also have fixed size.

See also: resize/2 [page 60], sparse_size/1 [page 61].

`resize(Size::integer(), Array::array()) -> array()`

Changes the size of the array. If `Size` is not a nonnegative integer, the call fails with reason `badarg`. If the given array has fixed size, the resulting array will also have fixed size.

`set(I::integer(), Value::term(), Array::array()) -> array()`

Sets entry `I` of the array to `Value`. If `I` is not a nonnegative integer, or if the array has fixed size and `I` is larger than the maximum index, the call fails with reason `badarg`.

If the array does not have fixed size, and `I` is greater than `size(Array)-1`, the array will grow to size `I+1`.

See also: get/2 [page 58], reset/2 [page 59].

`size(Array::array()) -> integer()`

Returns the number of entries in the array. Entries are numbered from 0 to `size(Array)-1`; hence, this is also the index of the first entry that is guaranteed to not have been previously set.

See also: set/3 [page 60], sparse_size/1 [page 61].

`sparse_foldl(Function, InitialAcc::term(), Array::array()) -> term()`

Types:

- Function = (Index::integer(), Value::term(), Acc::term()) -> term()

Folds the elements of the array using the given function and initial accumulator value, skipping default-valued entries. The elements are visited in order from the lowest index to the highest. If `Function` is not a function, the call fails with reason `badarg`.

See also: foldl/3 [page 57], sparse_foldr/3 [page 60].

`sparse_foldr(Function, InitialAcc::term(), Array::array()) -> term()`

Types:

- Function = (Index::integer(), Value::term(), Acc::term()) -> term()

Folds the elements of the array right-to-left using the given function and initial accumulator value, skipping default-valued entries. The elements are visited in order from the highest index to the lowest. If `Function` is not a function, the call fails with reason `badarg`.

See also: foldr/3 [page 57], sparse_foldl/3 [page 60].

`sparse_map(Function, Array::array()) -> array()`

Types:

- Function = (Index::integer(), Value::term()) -> term()

Maps the given function onto each element of the array, skipping default-valued entries. The elements are visited in order from the lowest index to the highest. If `Function` is not a function, the call fails with reason `badarg`.

See also: map/2 [page 58].

`sparse_size(A::array()) -> integer()`

Returns the number of entries in the array up until the last non-default valued entry. In other words, returns `I+1` if `I` is the last non-default valued entry in the array, or zero if no such entry exists.

See also: resize/1 [page 60], size/1 [page 60].

`sparse_to_list(Array::array()) -> list()`

Converts the array to a list, skipping default-valued entries.

See also: to_list/1 [page 61].

`sparse_to_orddict(Array::array()) -> [{Index::integer(), Value::term()}]`

Converts the array to an ordered list of pairs `{Index, Value}`, skipping default-valued entries.

See also: to_orddict/1 [page 61].

`to_list(Array::array()) -> list()`

Converts the array to a list.

See also: from_list/2 [page 57], sparse_to_list/1 [page 61].

`to_orddict(Array::array()) -> [{Index::integer(), Value::term()}]`

Converts the array to an ordered list of pairs `{Index, Value}`.

See also: from_orddict/2 [page 58], sparse_to_orddict/1 [page 61].

1(6)

Programming Efficiently with Binaries and Bit Strings

Per Gustafsson

Department of Information Technology
Uppsala University, Sweden Ericsson AB, Sweden
pergu@it.uu.se

Abstract

A new datatype, the *bit string*, and a new construct for manipulating binaries, binary comprehensions, are included in the R12B release of Erlang/OTP. In addition to this the implementation of binary construction and matching have been altered to make straightforward programs that operates on binaries or bit strings more efficient.

This paper will describe the new additions to the language and show how they can be used efficiently given the new optimizations of binary pattern matching and binary construction. It also includes some performance numbers to give an idea of the gains that can be made with the new optimizations.

1. Introduction

Binaries have received a makeover in the R12B release of Erlang with the introduction of *bit strings* and *extended comprehensions* as well as optimization of both binary construction and pattern matching.

Binaries have been a part of Erlang for a long time, and there has been a nice syntax for manipulating binaries since the R7B release of Erlang/OTP [3]. There has been some complaints about the using binaries for formats that are bit-oriented rather than byte-oriented since this tends to lead to complicated and error-prone code [2]. Bit strings are introduced to solve exactly this problem.

List comprehensions are used a lot in Erlang. They tend to make programs more compact and readable avoiding boilerplate code. In 2004 Jay Nelson suggested that there could also be binary comprehensions, a compact syntax for operating on binaries. The suggestion was formalized at the 2005 Erlang Workshop [1] and with some syntax changes this proposal was added as an optional feature in Erlang R11B. It will finally be a supported feature in R12B. The feature not only allows binary comprehension but also the use of binary generators in list comprehensions as well as list generators in binary comprehensions. Togeteher we call these features extended comprehensions which give users versatile abstractions for converting data between structured term formats and binary formats.

In addition to this binary comprehensions give the users a surefire way to use the new optimizations of binary construction and pattern matching. The optimization of construction of binaries might be the most important of the two as it makes it possible to build binaries in a piece-wise manner in linear time. This has been a problem in previous versions of Erlang forcing programmers to create lists of binaries which are then concatenated at the end to get efficient algorithms. This pattern tends to make algorithms more complicated than necessary.

The optimization of binary pattern matching is also important as it decreases the need to do unrolling of code that iterates over binary or keeping a counter to iterate over a binary. This optimization tends to make short natural implementations of functions which iterates over a binary efficient. Which is good as the hand-written optimizations above can introduce subtle bugs.

In this paper we will describe the new additions to the language in Section 2 and 3. Then we will give a short introduction to the implementation of operations on bit strings and binaries in Section 4 in order to be able to explain the new optimizations in Section 5 and give the reader some idea of how he should program to make use of them. Finally we have some performance measurements in Section 6 and conlusions in Section 7.

2. Bitstrings and binaries

A new datatype the *bit string* is introduced into Erlang. A bit string is a sequence of bits of any length this separates it from a *binary* which is a sequence of bits where the number of bits is evenly divisible by eight. These definitions implies that any binary is also a bit string.

2.1 Manipulating bit strings using the bit syntax

A bit syntax expression:

```
<<Seg1,...,SegN>>
```

Evaluates to a bit string. If the sum of the sizes of all segments in the expression is divisible by eight the result is also a binary. Previously such expression could only evaluate to binaries and a runtime error was raised if this was not the case.

With this extension the expression `Bin = <<1:9>>` which previously caused a runtime error now creates a 9-bit binary. To be able to use this bit string to build a new bigger bit string we can write:

```
<<Bin/bitstring, 0:1>>
```

Note the use of bitstring as the type. This expands to binary-unit:1 where as the binary type would have expanded to binary-unit:8. Since bitstring is a long word to write in a binary pattern there is an alias *bits* which is used in the rest of this paper, similarily for binary there is a new shorthand *bytes*.

To match out a bit-level binary we also use the bit string type as in :

```
case Bin of
  <<1:1,Rest/bits>> -> Rest;
  <<0:1,_/bits>> -> 0
end
```

This allows us to avoid situations were we previously had to calculate padding.

Example 2.1 A format from the IS 683-PRL protocol which consists of a 5-bit field describing how many 11-bit fields it was followed by. Decoding this format required a complicated calculation of padding to implement in a straightforward manner. The result is shown in Program 1.

Program 1 Decoding a format in the IS 683-PRL protocol

```
decode(<<NumChans:5, _Pad:3, _Rest/binary>> = Bin) ->
  decode(Bin, NumChans, NumChans, []).

decode(_, _, 0, Acc) ->
  Acc;
decode(Bin, NumChans, N, Acc) ->
  SkipBef = (N - 1) * 11,
  SkipAfter = (NumChans - N) * 11,
  Pad = (8 - ((NumChans * 11 + 5) rem 8)) rem 8,
  <<_:5, _:SkipBef, Chan:11, _:SkipAfter, _:Pad>> = Bin,
  decode(Bin, NumChans, N - 1, [Chan | Acc]).
```

With the introduction of bit strings it can be implemented without any padding calculations at all as:

```
decode(<<NumChans:5, Rest/bits>>) ->
  decode(NumChans, Rest, []).

decode(0, _, Acc) ->
  lists:reverse(Acc);
decode(N, <<Chan:11,Rest/bits>>, Acc) ->
  decode(N-1, Rest, [Chan|Acc]).
```

2.2 BIFs for manipulating bit strings

The current builtin functions for manipulating binaries will still only be defined for binaries. We will instead introduce four new BIFs which operate on bit strings .hey are described in Table 1.

3. Bit String Comprehensions

Bit string comprehensions are analogous to List Comprehensions. They are used to generate bit strings efficiently and succintly. Bit string comprehensions are written with the following syntax:

```
<< BitString || Qualifier1,...,QualifierN >>
```

`BitString` is a bit string expression, and each `Qualifier` is either a *generator*, a *bit string generator* or a *filter*.

generator: `Pattern <- ListExpr`
Where `ListExpr` must be an expression which evaluates to a list of terms.

bit string generator: `BitstringPattern <= BitStringExpr`
Where `BitStringExpr` must be an expression which evaluates to a bitstring.

filter : `Expr`
Where `Expr` must be an expression which evaluates to `true` or `false`

The variables in the generator patterns shadow variables in the function clause surrounding the bit string comprehensions. A bit string comprehension returns a bit string, which is created by concatenating the results of evaluating `BitString` for each combination of bit string generator or ordinary generator elements for which all filters are true.

Example 3.1 A simple comprehension which changes all lower case ascii characters in the bit string `Bits` into upper case characters,

```
<< <<(to_upper(X))>> || <<X>> <= Bits >>
```

This has the same semantics as the following expression:

```
bits_to_upper(Bits)

bits_to_upper(<<X,Rest/bits>>) ->
  <<(to_upper(X)), (bits_to_upper(Rest))/bits>>;
bits_to_upper(_) -> <<>>.
```

The translation to Erlang code is pretty straightforward, but the runtime for the Erlang program above is quadratic in the size of `Bits`, whereas the comprehension will be evaluated in linear time.

Since both ordinary list generators and bit string generators are allowed in bit string comprehensions they can be used to convert a list of data structures to a bit string representation.

Example 3.2 Consider the case where you have a list of three tuples where the first value in the tuple is one of 6 different atoms, the second value is a 16-bit integer and the third value is a float. Than you can turn that into a compact format using the following code:

```
<< <<(atom_to_int(Atom)):3,Int:16,Float/float>> ||
     {Atom,Int,Float} <- List >>.
```

Where `atom_to_int` maps the six different atoms to integers between 0 and 5.

3.1 Bit String Generators in List Comprehensions

In addition to introducing bit string comprehensions we also allow bit string generators in list comprehensions. This is useful for turning bit strings into structured data. One example when it is useful is for the problem described in Example 2.1. Using a bit string generator in a list comprehension this can be written as:

```
decode(<<N:5,Chans:N/bits-unit:11,_/bits>>) ->
  [Chan || <<Chan:11>> <- Chans].
```

4. Implementation

In order to describe the new optimizations of binary pattern matching and construction I must first describe how bit strings are represented and bit string operations are implemented.

4.1 The bit string datatype

The layout of bitstrings is a little bit complicated. The actual data in a bit string resides off heap. There is a data structure on the heap that is called a REFC binary that points to the off heap data. Bitstrings are so called sub-binaries which also reside on the heap. They point to REFC binary and they also contain offset and size fields. They never point directly to the off-heap data. The situation is described in Figure 4.1.

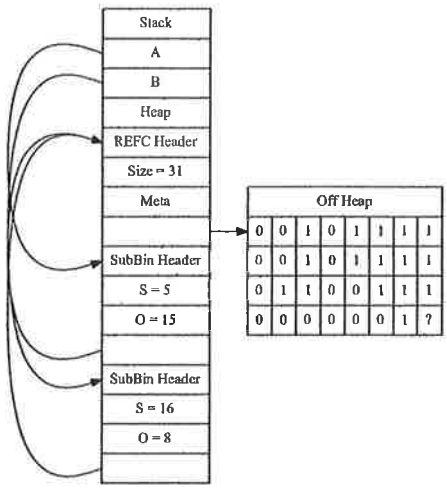
4.2 Bit String Construction

A bit string construction expression that has the form: $\texttt{<<ve}_1\texttt{:se}_1\texttt{/t}_1\texttt{,...,ve}_n\texttt{:se}_n\texttt{/t}_n\texttt{>>}$ is translated as follows. We start by evaluating all the value and size expressions so that we end up with an expression of the form $\texttt{<<v}_1\texttt{:s}_1\texttt{/t}_1\texttt{,...,v}_n\texttt{:s}_n\texttt{/t}_n\texttt{>>}$ where all the $\texttt{v}_i$:s are values and all the $\texttt{s}_i$:s are non-negative integers. If any $\texttt{s}_i$ is a negative value, a run-time exception is raised.

Then the the resulting size of the binary we are building is calculated as $\sum_{i=1}^{n} s_i$. An appropriate amount of off heap space is allocated for the data and the REFC binary is created on the heap. Then each segment is written into the data part. When this is done the sub binary which becomes the result of the expression is created.

Signature	Definition
`bit_size/1::bitstring() -> integer()`	Returns the size of a bit string in bits. This BIF is allowed in guards.
`list_to_bitstring/1::bitstring_list() -> bitstring()`	Concatenates the bit strings and chars in the bitstring list to create a bit string. A bitstring list is an io list which can contain bit strings as well as binaries the chars in the bitstring list are treated as if they were bit strings consisting of 8 bits.
`bitstring_to_list/1::bitstring() -> [char()|bitstring()]`	Turns a bit string into a list of characters and if the number of bits in the bit string is not evenly divisible by eight the last element in the list is a bit string consisting of the last 1-7 bits of the original bit string.
`is_bitstring/1::any() -> bool()`	Returns true if the argument is a bit string, otherwise it returns false. This BIF is allowed in guards.

Table 1. New Builtin Functions for manipulating bit strings



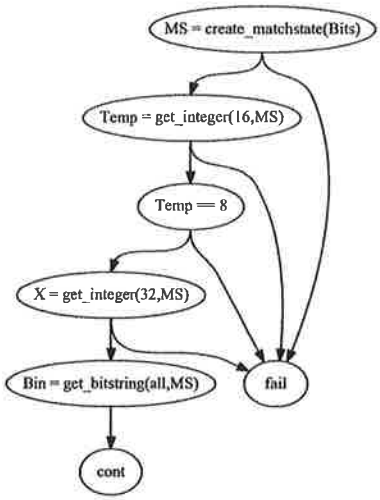


Figure 1. Matching graph for `<<8:16, X:32, Bin/bits>> = Bits`

4.3 Binary Pattern Matching

Consider the following expression:

```
<<8:16, X:32, Bin/bits>> = Bits
```

This gets compiled into the sequence shown in Figure 1. The instruction `create_matchstate` takes a bit string and creates a matchstate to be used during the matching. The matchstate contains the size of the bitstring we are matching against, the offset we are at and a pointer to the data. The `get_integer` instruction takes a matchstate and a size and reads that number of bits, turns it into an integer and updates the offset in the matchstate. The `get_bitstring` function creates a sub-binary from the matchstate.

A more complicated matching with several patterns is compiled into a tree of instuctions for exampleif we have:

```
case Bits of
  <<A:8, 1:8, X:8, Bin/bits>>  -> cont1;
  <<A:8, 2:8, X:16, Bin/bits>> -> cont2;
  <<>>                         -> cont3
end
```

We end up with the tree of instructions shown in Figure 2.

We have some new instructions:

`save_matchstate(N,MS)` This instruction saves the present offset in the matchstate in save slot N.

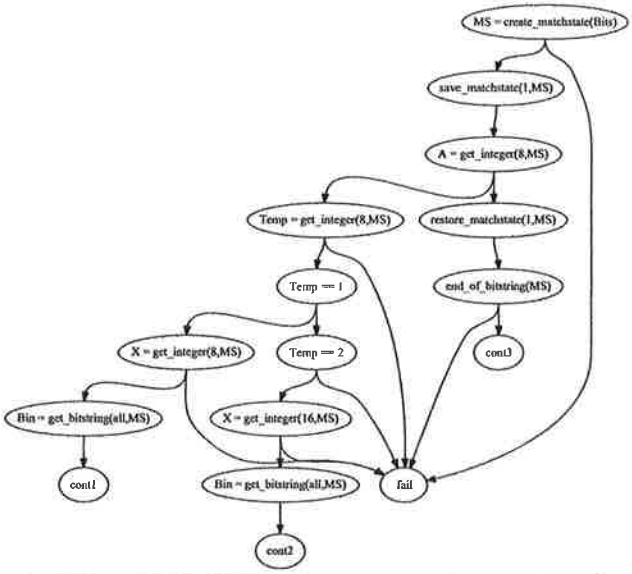


Figure 2. Matching graph for case statement

`restore_matchstate(N,MS)` This instruction loads the offset value from slot N and makes it the present offset value.

`end_of_bitstring` This instruction checks that the offset in the matchstate is equal to the size. That is that we have reached the end of the bit string

5. Optimizations

In R12B both binary construction and binary pattern matching has been optimized. In this section we will describe these optimizations and discuss how to write code that best utilizes trhe optimizations.

5.1 Binary Construction Optimization

The basis of this optimization is that it the emulator can create bit strings with extra uninitialized space, so if a bit string is built by continously appending to a binary the data does not need to be copied if there is enough uninitialized data at the end of the bit string.

`Bits` contains a bit string of 1000 bits followed by 600 bits of uninitialized data.

In the expression

```
NewBits = <<Bits/bits, 12:32>>
```

`NewBits` gets bound to a bit string of 1032 bits followed by 568 bits of uninitialized data, `Bits` on the other hand can no longer be appended to.

On the other hand if we have this expression:

```
NewBits = <<Bits/bits, 12:640>>
```

Since there is not enough uninitalized data `NewBits` becomes a new bit string consisting of 1640 bits followed by 1640 bits of uninitialized data. `Bits` remains the same a bit string of 1000 bits with 600 bits uninitialized data.

What does this mean in practice when your programming?

- It means you can build bit strings piecewise in linear time
- It means that when your building a bit string from a list or from an other bit string and you want to have the same order of your pieces you should use tail calls and an acumulator
- It means that you can reverse a bit string efficiently without turning it into a list

Let us see some examples of efficient programs for building bit strings:

Example 5.1 This function reverses a bit string consisting of 32 bit integers:

```
reverse_32bit(<<X:32,Rest/bits>>) ->
  <<(reverse_32bit(Rest))/bits,X:32>>;
reverse_32bit(<<>>) ->
  <<>>.
```

Not that when we are constructing the answer the first element of the new bit string is the growing bit string.

Note that we use direct recursion in order to get the reverse order in the result in the following example we want to perserve the order of the input.

Example 5.2 This simple function stores a double in 32-bits if it is prefaced by a zero if it is prefaced by a one it uses 64-bits.

```
save_floats(Bits) ->
  save_floats(Bits, <<>>).
```

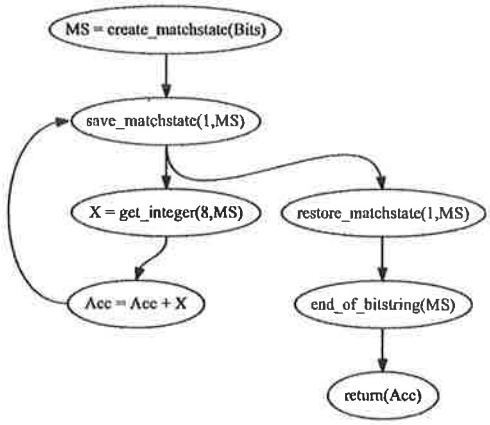


Figure 4. Optimized code for `sum1/2`

```
save_floats(<<0:1,F:64/float,Rest/bits>>, Acc) ->
  save_floats(Rest, <<Acc/bits,0:1,F:32/float>>);
save_floats(<<1:1,F:64/float,Rest/bits>>, Acc) ->
  save_floats(Rest, <<Acc/bits,1:1,F:64/float>>);
save_floats(<<>>,Acc) ->
  Acc.
```

5.2 Binary Pattern Matching Optimization

To describe the new optimization of binary pattern matching consider these two functions which calculates the sum of the bytes in a bit string:

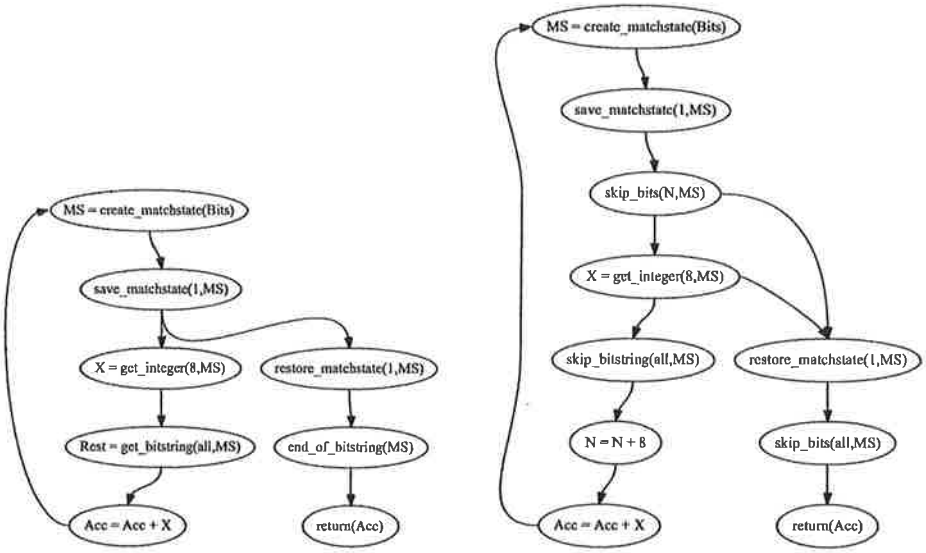
```
sum1(Bits) ->
  sum1(Bits, 0).

sum1(<<X,Rest/bits>>, Acc) ->
  sum1(Rest, Acc+X);
sum1(<<>>, Acc) -> Acc.

sum2(Bits) ->
  sum2(Bits,0,0).

sum2(Bits,N,Acc) ->
  case Bits of
    <<_:N,X,Rest/bits>> ->
       sum2(Bits,N+8,Acc+X);
    <<_/bits>> ->
       Acc
  end.
```

The generated code for `sum1/2` is shown in Figure 3(a). In each iteration of the loop a sub-binary is created from the match sate only to promptly be turned in to a new match state in the next iteration.

For `sum2/3` we avoid creating this sub-binary, but we still have to create the match state in each iteration.

The new optimization of binary pattern matching follows from the observation that it is unnecessary to convert a match state into sub-binary only to immediatly convert it back to a match state. Instead we can keep the match state in the loop. Using this optimization the code for `sum1/2` is shown in Figure 4.

How should we write code to make it possible to apply this optimization? The most important thing is to make sure that the binary we are matching against is not used for anything else in the function. In addition to this we need to make sure that the sub-binary we are creating is only used in a self recursive call.



(a) Generated code for `sum1/2`

(b) Generated code for `sum2/3`

Figure 3. Code generated for two different functions calculating the byte sum of a bit string

```
f(<<Pattern1,...,Rest/bits>>,...) ->
   ... % Rest is not used here
   f(Rest,...);
f(<<Pattern2,...,Rest/bits>>,...) ->
   ... % Rest is not used here
   f(Rest,...);

...

f(<<>>, ...) ->
  ReturnValue
```

Figure 5. Function skeleton that will be optimized

A good model for functions that want to make sure they use this optimization is shown in figure 5.

6. Performance

In this section we will give some performance figures and compare some different approaches to write programs operating on bit strings as well as comparing handling of bit strings in R11B-5 and R12B. All of the benchmarks in this section have been run on a unicore 2.4 GHz Pentium 4 with 1 GB of memory, running Ubuntu 7.10.

6.1 IP-packet checksum

This program exists in four different flavors. Two which creates sub-binaries like the program in Figure 3(a) the difference between these programs is that one of them unrolls the loop eight times whereas the other program does no unrolling. The programs are called *Sub* and *SubUnrolled*. The other two programs use the same type of iteration as the program in Figure 3(b), one of these progams is also unrolled. They are called *Iter* and *IterUnrolled*. They each calculate the checksum for a 658 kB file one hundred times. The runtimes can be found in Table 2. The four different functions can be found in Program [?] in the appendix.

Program	BEAM R12B-0	HiPE R12B-0
Bit String Comprehension	10.43	2.69
Bit String Recursion	14.49	3.41
List Comprehension	10.22	6.21

Table 3. Runtimes in seconds for making 65.8 MB of data all upper case

The results suggest that performance of binary pattern matching in general is better in R12B, but paritcularily when using sub-binaries. The effect of doing unrolling decrease from a factor four in R11B to less than approximatly a factor 1.5, which suggests that good performance can be had without adding ugly unrolling.

6.2 Upper Case

In the second experiment binaries are both constructed and pattern matched on, but it is a pretty simple program. It simply turns a binary string into an all upper case binary string. There are three different versions of the function all of them are shown in Program 2 in the appendix.

It was not really relevant to run this benchmark on R11B-5 since the bit string recursion function had a quadratic cost and bit string comprehensions were very inefficient. They were thus only run on R12B. The input was the same as in the IP-checksum case, a 658 kB file that was turned into an all upper case file one hundred times. The results are shown in Table 3.

The results seem to suggest that with BEAM bit string comprehensions are competitive with operating on a list while it becomes superior when native compilation is used. It is also superior to explicit recursion. This is the case since it is easier to analyze a bit string comprehension and thus construction and matching of bit strings can be optimized further.

The implementation of bit string comprehensions can be improved further. In many cases the size of the resulting bit string can be computed beforehand. This is not done yet, but we expect to implement this in future releses of Erlang/OTP.

Program	BEAM R11B-5	HiPE R11B-5	BEAM R12B-0	HiPE R12B-0
Sub	10.18	3.69	2.66	0.62
SubUnrolled	2.17	0.90	1.13	0.38
Iter	8.31	2.90	5.09	2.15
IterUnrolled	2.16	0.78	1.54	0.58

Table 2. Runtimes in seconds for calculating checksums of 65.8 MB of data

7. Conclusions

This is not a comprehensive description of how to use binaries and bit strings efficiently in your programs. It is simply a short description of how binaries have been extended into bit strings and how various operations on bit strings are implemented. We also try to describe how we optimize these operations. Hopefully this description will help you write shorter and easier and more efficient programs in the future. What we want you to take away from this paper is summarized in the following bullet points.

- Bit strings makes it much easier to deal with bit-oriented data in Erlang
- When you are building new bit strings make sure you append new data to the end of an old bit string
- When you iterate over a bit string use a direct style matching similar to what you would do for lists
- If you are doing a map operation over bit strings use bit string comprehensions to get efficient and concise code
- Write simple straight-forward code first to see if the optimizations makes it fast enough. Then you can try various approaches to make it faster.

References

[1] P. Gustafsson and K. Sagonas. Bit-level binaries and generalized comprehensions in Erlang. In *Proceedings of the Fourth ACM SIGPLAN Erlang Workshop*, pages 1–8. ACM Press, Sept. 2005.

[2] M. Läng. Erlang in the corelatus mtp2 signalling gateway, Oct. 2001. Available at http://www.erlang.se/euc/01/.

[3] P. Nyblom. The bit syntax - the released version. In *Proceedings of the Sixth International Erlang/OTP User Conference*, Oct. 2000. Available at http://www.erlang.se/euc/00/.

8. Appendix

Program 2 Three ways to make a binary all upper case

```
bit_string_comp(Bin) ->
  << <<(to_upper(X))>> || <<X>> <= Bin >>.

bit_string_recursion(Bin) ->
  bit_string_recursion(Bin, <<>>).

bit_string_rec(<<X,Rest/binary>>, Acc) ->
  bit_string_rec(Rest,<<Acc/binary,(to_upper(X))>>);
bit_string_rec(<<>>, Acc) -> Acc.

list_comprehension(Bin) ->
  list_to_binary([to_upper(X) ||
                  X <- binary_to_list(Bin)]).

to_upper(X) when X >= $a, X =< $z ->
  X + ($A-$a);
to_upper(X) ->
  X.
```

Program 3 Four ways to calculate an IP checksum

```
-define(INT16MAX, 65535).

sub(<<N1:16, Rem/binary>>,Csum) ->
   sub(Rem, do_trunc(Csum+N1));
sub(<<N1:8>>,Csum) ->
  sub(<<>>,do_trunc(Csum+(N1 bsl 8)));
sub(<<>>,Csum) when Csum > ?INT16MAX ->
  Val=(Csum band ?INT16MAX) + (Csum bsr 16),
   sub(<<>>,Val);
sub(<<>>,Csum) -> (bnot Csum) band ?INT16MAX.

sub_unrolled(<<N1:16,N2:16,N3:16,N4:16,N5:16,N6:16,
               N7:16,N8:16,Rem/binary>>, Csum) ->
  sub_unrolled(Rem,do_trunc(Csum+N1+N2+N3+N4+N5+N6+N7+N8));
sub_unrolled(<<N1:16, Rem/binary>>,Csum) ->
  sub_unrolled(Rem, do_trunc(Csum+N1));
sub_unrolled(<<N1:8>>,Csum) ->
  sub_unrolled(<<>>,Csum+(N1 bsl 8));
sub_unrolled(<<>>,Csum) when Csum > ?INT16MAX ->
  Val=(Csum band ?INT16MAX) + (Csum bsr 16),
  sub_unrolled(<<>>,Val);
sub_unrolled(<<>>,Csum) ->
  (bnot Csum) band ?INT16MAX.

iter(N,Bin,Csum) ->
  case Bin of
    <<_:N/binary, N1:16,_/binary>> ->
      iter(N+2,Bin,do_trunc(Csum+N1));
    <<_:N/binary, Num:8>> ->
      iter(N+1,Bin,do_trunc(Csum+(Num bsl 8)));
    _ when Csum > ?INT16MAX ->
       Val = (Csum band ?INT16MAX + (Csum bsr 16)),
      iter(N,Bin,Val);
    _ ->
      (bnot Csum) band ?INT16MAX
 end.

iter_unrolled(N,Bin,Csum) ->
  case Bin of
    <<_:N/binary, N1:16,N2:16,N3:16,
      N4:16,N5:16,N6:16,N7:16,N8:16,
      _/binary>> ->
      iter_unrolled(N+16,Bin,
        do_trunc(Csum+N1+N2+N3+N4+N5+N6+N7+N8));
    <<_:N/binary, N1:16,_/binary>> ->
      iter_unrolled(N+2,Bin,do_trunc(Csum+N1));
    <<_:N/binary, Num:8>> ->
      iter_unrolled(N+1,Bin,Csum+(Num bsl 8));
    _ when Csum > ?INT16MAX ->
       Val = (Csum band ?INT16MAX + (Csum bsr 16)),
      iter_unrolled(N,Bin,Val);
    _ ->
      (bnot Csum) band ?INT16MAX
  end.

do_trunc(Csum) when Csum > 16#6ffffff, Csum < 16#7ffffff  ->
  Csum band ?INT16MAX + (Csum bsr 16);
do_trunc(Csum) -> Csum.
```

Erlang/OTP User Conference 2007

Speakers

Richard	Carlsson	IAR Systems AB	Uppsala	Sweden	richardc@it.uu.se
Dan	Gudmundsson	Ericsson AB	Stockholm	Sweden	
Per	Gustafsson	Uppsala University	Uppsala	Sweden	per.gustafsson@it.uu.se
Gordon	Guthrie	Hypernumbers.com	Edinburgh	Scotland	gordonguthrie@backawinner.gg
John	Hughes	Quviq AB	Göteborg	Sweden	john.hughes@quviq.com
Martin	Logan	Orbitz	Chicago	USA	martinjlogan@gmail.com
Leslaw	Lopacki	Telenor Nordic IS	Sandvika	Norway	leslaw.lopacki@telenor.com
Kenneth	Lundin	Ericsson AB	Stockholm	Sweden	kenneth.lundin@ericsson.com
Eric	Merritt		Seattle	USA	cyberlync@gmail.com
Tamás	Nagy	Eötvös Loránd University	Budapest	Hungary	lestat@elte.hu
Anikó	Nagyné Vig	Eötvös Loránd University	Budapest	Hungary	viganiko@inf.elte.hu
Vincenzo	Nicosia	Erlang Training & Consulting	London	England	vnicosia@diit.unict.it
Samuel	Rivas	LambdaStream	A Coruña	Spain	samuel.rivas@lambdastream.com
Erik	Stenman	Kreditor Europe AB	Stockholm	Sweden	Erik.Stenman@kreditor.se
Marcus	Taylor	Erlang Training & Consulting	London	England	marcus@erlang-consulting.com
Fredrik	Thulin	Stockholm University	Stockholm	Sweden	ft@it.su.se

Chairmen

Francesco	Cesarini	Erlang Training & Consulting	London	England	francesco@erlang-consulting.com
Bjarne	Däcker	cs-lab.org	Segeltorp	Sweden	bjarne@cs-lab.org
Mickaël	Rémond	Process One	Paris	France	mickael.remond@process-one.net
Ulf	Wiger	Ericsson AB	Stockholm	Sweden	ulf.wiger@ericsson.com
Claes	Wikström	Tail-f AB	Stockholm	Sweden	klacke@tail-f.com

Participants

Niklas	Adalberth	Kreditor Europe AB	Stockholm	Sweden	Niklas.Adalberth@kreditor.se
Akos	Akos	ROC Development Kft	Debrecen	Hungary	
Kristoffer	Andersson	Synapse Mobile Networks	Stockholm	Sweden	
Peter	Andersson	Ericsson AB	Stockholm	Sweden	
Ingela	Anderton-Andin	Ericsson AB	Stockholm	Sweden	ingela@theheartofgold.org
Adam	Aquilon	Ericsson AB	Stockholm	Sweden	adam.aquilon@ericsson.com
Marcus	Arendt	Marcus Arendt AB	Stockholm	Sweden	marcus@arendt.se
Joe	Armstrong	Ericsson AB	Stockholm	Sweden	erlang@gmail.com
Gösta	Ask	SalveLinus	Stockholm	Sweden	g.ask@telia.com
Henrik	Back	Mobile Arts AB	Uppsala	Sweden	Henrik.back@mobilearts.se

Participants

Tim	Becker	Syngenio AG	Cologne	Germany	tim.becker@gmx.net
Per	Bergqvist	Synapse Mobile Networks	Stockholm	Sweden	per.bergqvist@synap.se
Per	Bergström	Ericsson AB	Stockholm	Sweden	per.bergstrom@ericsson.com
Johan	Bevemyr	Tail-f AB	Stockholm	Sweden	jb@tail-f.com
Xingdong	Bian	Erlang Training & Consulting	London	England	
Martin	Björklund	Tail-f AB	Stockholm	Sweden	mbj@tail-f.com
Johan	Blom	Mobile Arts AB	Stockholm	Sweden	Johan.Blom@mobilearts.com
Jonas	Boberg	Erlang Training & Consulting	London	England	jonas@erlang-consulting.com
Hans	Bokvist	Ericsson AB	Västerås	Sweden	hans.bokvist@ericsson.com
Hans	Bolinder	Ericsson AB	Stockholm	Sweden	
Garry	Bulmer		Coventry	England	gbulmer@gmail.com
Göran	Båge	Mobile Arts AB	Stockholm	Sweden	goran.bage@mobilearts.com
Mats	Cronqvist	Ericsson AB	Stockholm	Sweden	mats.cronqvist@ericsson.com
Graham	Crowe	Ericsson AB	Stockholm	Sweden	graham.crowe@ericsson.com
Björn-Egil	Dahlberg	Ericsson AB	Stockholm	Sweden	
Anders	Danne	Ericsson AB	Stockholm	Sweden	anders.danne@ericsson.com
Wendy	Devolder	Skills Matter	London	England	wendy.devolder@skillsmatter.com
Vlad	Dumitrescu	HiQ Göteborg AB	Göteborg	Sweden	vladdu55@gmail.com
Marcus	Dübois	Stockholm University	Stockholm	Sweden	
Niclas	Eklund	Ericsson AB	Stockholm	Sweden	nick@erix.ericsson.se
Ulf	Eliasson	Erlang Training & Consulting	London	England	ulf@erlang-consulitng.com
Nabiel	Elshiewy	Vinnova	Stockholm	Sweden	Nabiel.Elshiewy@VINNOVA.se
Lars Göran	Ericson	Synapse Mobile Networks	Stockholm	Sweden	
Morgan	Eriksson	Ericsson AB	Stockholm	Sweden	morgan.xe.eriksson@ericsson.com
Jonas	Falkevik	Mobile Arts AB	Stockholm	Sweden	jonas.falkevik@mobilearts.com
Paul	Fleischer	University of Aarhus	Aarhus	Denmark	pf@daimi.au.dk
Rabbe	Fogelholm	Ericsson AB	Stockholm	Sweden	rabbe.fogelholm@ericsson.com
Scott Lystig	Fritchie	Gemini Mobile Technologies		USA	fritchie@snookles.com
Magnus	Fröberg	Kreditor Europe AB	Stockholm	Sweden	Magnus.Froberg@kreditor.se
Francesca	Gangemi	Erlang Training & Consulting	London	England	francesca@erlang-consulting.com
Dmitri	Girenko	Akumiitti Oy	Helsinki	Finland	Dmitri.Girenko@akumiitti.com
Joakim	Grebenö	Tail-f AB	Stockholm	Sweden	jocke@tail-f.com
Rickard	Green	Ericsson AB	Stockholm	Sweden	
Dag	Gruneau	Tail-f AB	Stockholm	Sweden	dag@tail-f.com
Nicholas	Gunder	Motorola A/S	Copenhagen	Denmark	ngunder@motorola.com

Participants

Björn	Gustavsson	Ericsson AB	Stockholm	Sweden	bjorn@erix.ericsson.se
Per	Hallin	Synapse Mobile Networks	Stockholm	Sweden	per.hallin@synap.se
Mazen	Harake	Erlang Training & Consulting	London	England	mazen@erlang-consulting.com
Dale	Harvey	Hypernumbers.com	Edinburgh	Scotland	
Andreas	Hasselberg	Kreditor Europe AB	Stockholm	Sweden	Andreas.Hasselberg@kreditor.se
Dragan	Havelka	Mobile Arts AB	Stockholm	Sweden	dragan.havelka@mobilearts.com
Per	Hedeland	Tail-f AB	Stockholm	Sweden	per@tail-f.com
Pekka	Hedqvist	Optimobile AB	Stockholm	Sweden	Pekka.Hedqvist@OptiMobile.SE
Oscar	Hellström	Erlang Training & Consulting	London	England	oscar@erlang-consulting.com
Johan	Herdegård	Mobile Arts AB	Stockholm	Sweden	johan.herdegard@mobilearts.com
Sean	Hinde	Synapse Mobile Networks	Stockholm	Sweden	
Torben	Hoffmann	Motorola A/S	Copenhagen	Denmark	Torben.Hoffmann@motorola.com
Fredrik	Holmén	Upsala Systemkonsult AB	Uppsala	Sweden	fredrik@upsys.se
Zoltán	Horvath	Eötvös Loránd University	Budapest	Hungary	hz@inf.elte.hu
Victor	Jacobsson	Kreditor Europe AB	Stockholm	Sweden	Victor.Jacobsson@kreditor.se
Karl	Johansson	Erlang Training & Consulting	London	England	karl@erlang-consluting.com
Klas	Johansson	Ericsson AB	Linköping	Sweden	klas.johansson@ericsson.com
Torbjörn	Johnson	Firma Torbjörn Johnson	Stockholm	Sweden	torbjorn.k.johnson@swipnet.se
Fredrik	Jones	Ericsson AB	Göteborg	Sweden	fredrik.xx.jones@ericsson.com
Ruan	Jonker	Mira Networks	Johannesburg	South Africa	ruanj@miranetworks.net
Andreas	Karlsson	Erlang Training & Consulting	Stockholm	Sweden	andreas@erlang-consulting.com
Bertil	Karlsson	Ericsson AB	Stockholm	Sweden	bertil.karlsson@ericsson.com
Martin	Karlsson	Erlang Training & Consulting	London	England	martin@erlang-consulting.com
Mikael	Karlsson	Creado Systems	Stockholm	Sweden	mikael.karlsson@creado.com
Mikael	Karlsson	Mikadako AB	Utö	Sweden	micke@mikadako.com
Roland	Karlsson	Erlang Training & Consulting	Stockholm	Sweden	roland.karlsson@bonetmail.com
Martin	Kjellin	Mobile Arts AB	Stockholm	Sweden	martin.kjellin@mobilearts.com
Bengt	Kleberg	Ericsson AB	Stockholm	Sweden	bengt.kleberg@ericsson.com
Mikael	Laaksonen	Mobile Arts AB	Stockholm	Sweden	micke.laaksonen@gmail.com
Tomas	Langer	Ericsson AB	Stockholm	Sweden	tomas.langer@ericsson.com
Lukas	Larsson	Erlang Training & Consulting	London	England	lukas@erlang-consulting.com
Petter	Larsson	Cybercom Group	Stockholm	Sweden	petter.xa.larsson@ericsson.com
Tobias	Lindahl	Uppsala University	Uppsala	Sweden	tobias.lindahl@it.uu.se
Adam	Lindberg	Erlang Training & Consulting	London	England	adam@erlang-consulting.com
Erik	Lindblom	Synapse Mobile Networks	Stockholm	Sweden	

Participants

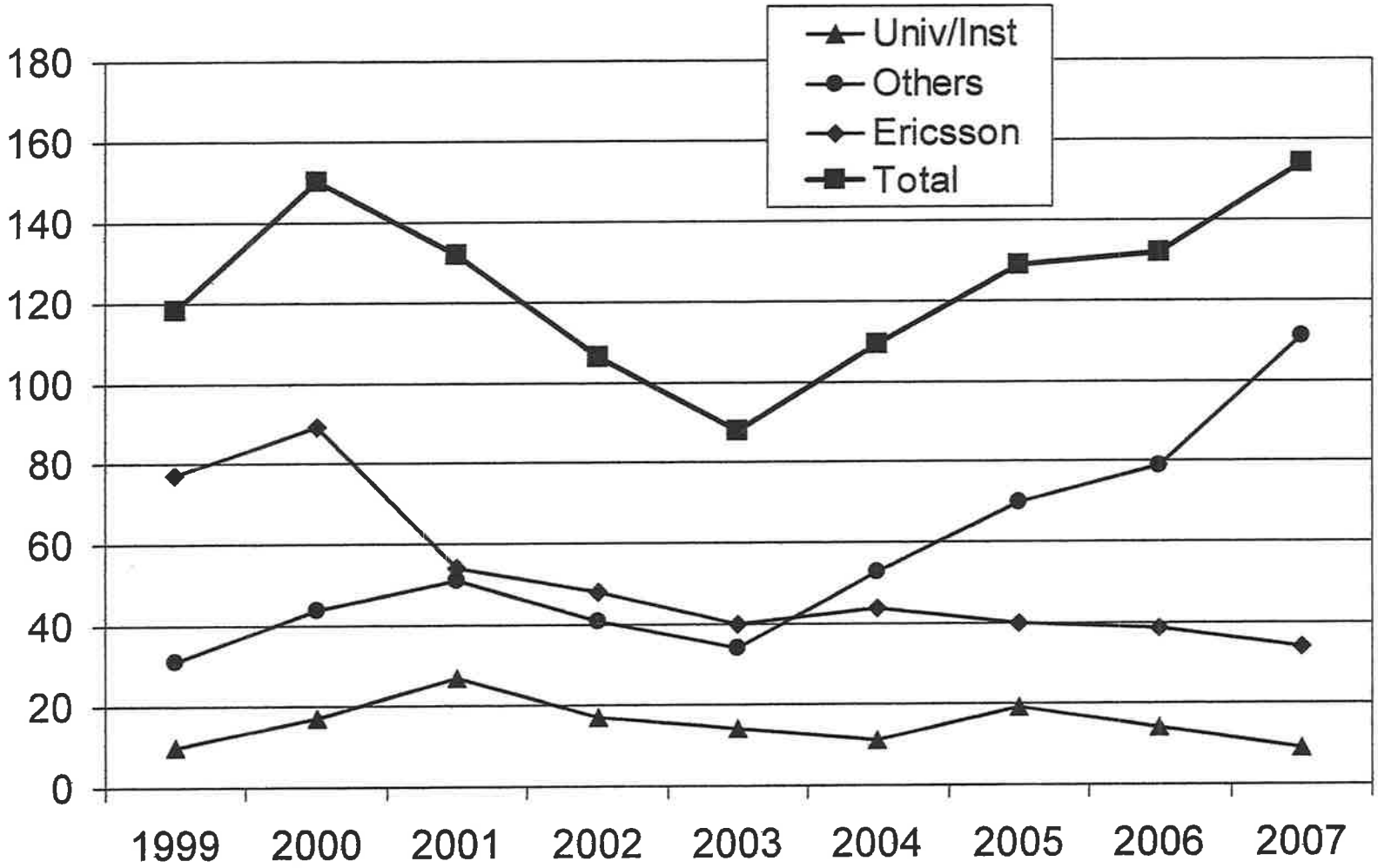
Thomas	Lindgren	Millpond Services Ltd	London	England	thomasl_erlang@yahoo.com
Mikael	Lindmark	Kreditor Europe AB	Stockholm	Sweden	Mikael.Lindmark@kreditor.se
Daniel	Luna	Kreditor Europe AB	Stockholm	Sweden	Daniel.Luna@kreditor.se
Peter	Lund	Synapse Mobile Networks	Stockholm	Sweden	peter.lund@synap.se
Matthias	Lång	Corelatus AB	Stockholm	Sweden	matthias@corelatus.se
Doug	Mansell	Number Play	Geneva	Switzerland	doug.mansell@numberplay.com
Thomas	Mattisson	Mobile Arts AB	Stockholm	Sweden	thomas.mattisson@mobilearts.com
Håkan	Mattsson	Ericsson AB	Stockholm	Sweden	hakan@erix.ericsson.se
Sean	McEvoy	Erlang Training & Consulting	London	England	sean@erlang-consulting.com
Hunter	Morris	Trost & Morris	London	England	hunter@binaryclub.com
Chandru	Mullaparthi	T-Moble	Hatfield	England	chandrashekhar.mullaparthi@gmail.com
Derek	Nangle		Seattle	USA	dnangle@gmail.com
Christer	Nilsson	Ericsson AB	Göteborg	Sweden	christer.n.nilsson@ericsson.com
Hans	Nilsson	Ericsson AB	Stockholm	Sweden	hans.r.nilsson@ericsson.com
Raimo	Niskanen	Ericsson AB	Stockholm	Sweden	raimo@erix.ericsson.se
Linus	Nordberg	Net Insight AB	Stockholm	Sweden	linus.nordberg@netinsight.net
Magnus	Nordén	Stockholm University	Stockholm	Sweden	
Jörgen	Norén	Kreditor Europe AB	Stockholm	Sweden	Jorgen.Noren@kreditor.se
Patrik	Nyblom	Ericsson AB	Stockholm	Sweden	pan@erix.ericsson.se
Jan Henry	Nyström	Erlang Training & Consulting	Uppsala	Sweden	jan@erlang-consulting.com
Filippo	Pacini	S.G. Consulting	Siena	Italy	pacini@sgconsulting.it
Arpad	Pandy	ROC Development Kft	Debrecen	Hungary	
Johan	Petersson	Ericsson AB	Stockholm	Sweden	johan.petersson@ericsson.com
Laurent	Picouleau	Erlang Training & Consulting	London	England	
Dan	Sahlin	Raycore AB	Stockholm	Sweden	dan.sahlin@raycore-fos.com
Sebastian	Siemiatkowski	Kreditor Europe AB	Stockholm	Sweden	Sebastian.Siemiatkowski@kreditor.se
Massimo	Signore		Vicenza	Italy	massimo.signore@esercito.difesa.it
Peter	Sjögren	Mobile Arts AB	Stockholm	Sweden	peter.sjogren@mobilearts.com
Kristtoffer	Skagerberg	Synapse Mobile Networks	Stockholm	Sweden	
Michal	Slaski	Erlang Training & Consulting	London	England	michal@erlang-consulting.com
Håkan	Stenholm	Kreditor Europe AB	Stockholm	Sweden	Hakan.Stenholm@kreditor.se
Sebastian	Strollo	Tail-f AB	Stockholm	Sweden	seb@tail-f.com
Anton	Strydom	Synapse Mobile Networks	Stockholm	Sweden	
Per Einar	Strömme		Stockholm	Sweden	stromme@telia.com
Göran	Stupalo	Ericsson AB	Stockholm	Sweden	

Participants

Ulf	Svarte Bagge	Corelatus AB	Stockholm	Sweden	ulf@corelatus.se
Alfred M	Szmidt	Kreditor Europe AB	Stockholm	Sweden	Alfred.Szmidt@kreditor.se
Lars	Sørensen	Motorola A/S	Copenhagen	Denmark	l.sorensen@motorola.com
Zoltan Peter	Toth	Ericsson AB	Budapest	Hungary	zoltan.peter.toth@ericsson.com
Torbjörn	Törnkvist	Kreditor Europe AB	Stockholm	Sweden	Torbjorn.Tornkvist@kreditor.se
Jane	Walerud	WAPA	Stockholm	Sweden	jane@walerud.com
Marc	van Woerkom	GMX GmbH	Munich	Germany	mvanwoerkom@gmx-gmbh.de
Hasan	Veldstra	Hypernumbers.com	Edinburgh	Scotland	
Mats	Westin	International Data Group	Stockholm	Sweden	mats.westin@gmail.com
Chris	Williams	Ericsson AB	Stockholm	Sweden	sailingareus@gmail.com
Patrik	Winroth	Synapse Mobile Networks	Stockholm	Sweden	patrik.winroth@synap.se
Robert	Virding	Swedish Defense Materiel Administration	Stockholm	Sweden	rvirding@gmail.com
Hao	Zhang	Mobile Arts AB	Stockholm	Sweden	hao@zhang.nu
Ming	Zhao	Mobile Arts AB	Stockholm	Sweden	Ming@zhao.nu
Jonas	Åman	Ericsson AB	Linköping	Sweden	
Lennart	Öhman	Sjöland & Thyselius Telecom AB	Stockholm	Sweden	Lennart.Ohman@st.se
Björn	Öqvist	Kreditor Europe AB	Stockholm	Sweden	biorn.oqvist@kreditor.se
Lennart	Östman	Synapse Mobile Networks	Stockholm	Sweden	

Updated October 28 2007

EUC participation



6