

Conference Proceedings

16th International Erlang User Conference

STOCKHOLM, SWEDEN

16th November 2010



16th International Erlang User Conference

Conference Programme

8:30 - 9:00	Registration, Tea and Coffee	Page
9:00 - 9:05	Welcome and Introduction Bjarne Däcker	1
9:05 - 9:50	Tools@Klarna Jia Wang, Tobbe Tornkvist, David Evans, Jordi Chacón, Tobias Lindahl	3
9:50 - 10:20	RefactorErl: a source code analyzer and transformer tool Zoltán Horváth, Melinda Tóth	23
10:20 - 10:50	Using Erlang to Test non-Erlang Products Graham Crowe	37
10:50 - 11:10	Morning Break	
11:10 - 11:40	PikkoServer, how to scale game servers and enable player density David Almroth	39
11:40 - 12:10	Hibari - Key Value Bigdata Store Joseph Wayne Norton	41
12:10 - 12:40	Agile is Everything Marcus Kern	51
12:40 - 13:40	Lunch	
13:40 - 14:10	Let's ProTest: Update from the Erlang and property based testing project Simon Thompson	53
14:10 - 14:40	Testing what should work, not what should not fail Samuel Rivas	77
14:40 - 15:10	Testing automotive software with Erlang Thomas Arts	79
15:10 - 15:40	Mission Critical with Erlang And QuickCheck: Quality Never Sleeps Raghav Karol, Torben Hoffman	95
15:45 - 16:10	Afternoon Break	
16:10 - 16:40	Continuous integration for Erlang/OTP, enhancing its quality and ease-of-use. Tino Breddin	111
16:40 - 17:10	A prototype state machine "MadCloud" for distributed applications Jacoby Thwaites	113
17:10 - 17:40	Masterless Distributed Computing with Riak Core Rusty Klophaus	115
17:40 - 18:10	A deep dive into some aspects of the multicore support in the Erlang VM Rickard Green	117
18:10 - 18:25	Latest News From the Erlang/OTP team at Ericsson Kenneth Lundin	127
18:25 - 18:30	Closing from Bjarne Däcker	

Bjarne Däcker
Manager of the CSLab at Ericsson
- the birthplace of Erlang



Welcome and Introduction

Abstract

Dear Erlang friends,

It is with the greatest pleasure that I wish to welcome you to the Sixteenth International Erlang/OTP User Conference. Erlang started as a programming language in the telecomms domain but with developments like the Internet, multicore, and Cloud computing, it has found a much wider applicability. Also the combination of concurrency and functional programming has proven to be extremely fruitful opening further areas such as testing, refactoring, databases.

In 2008 we outgrew the Ericsson conference facilities and had to move to the Astoria. This year we have filled it so next year, who knows!

I have heard many times that the talks are but an excuse for Erlang enthusiasts to meet and mingle and exchange experiences at intervals and at the following dinner party. That might be true, but it is also unfair to the speakers who keep telling us of new unexpected applications and of explorations at the very front of current Computer Science research.

So once again, I wish you welcome to a conference that is sure to widen your views of the computing world where Erlang is becoming one of the most significant players.

Biography

Bjarne Däcker joined Ericsson in 1966 as programmer and systems analyst. In 1984 he set up the Computer Science Lab together with Mike Williams to explore, develop and introduce new software technology in Ericsson often in collaboration with university research. The CSLab pioneered things like Unix, A.I., Lisp, Prolog and workstations in Ericsson.

Erlang was created at the CSLab by an initial team of Joe Armstrong, Mike Williams and Robert Virding. Bjarne organised the first Erlang User Conference in 1994. He has had various external commitments such as chairman of the steering committee of the Swedish national research programme in Computer Science 1987-1992 and member of the Evaluation Committee of European Union's ICT-Prize. The CSLab was closed in 2002 in the IT crash.

Tools@Klarna

Abstract

Klarna (currently) operates in six countries. We need to handle translations of PDF's, GUI, Emails, etc. The basis of our i18n system is built around the gettext Erlang application. To help us coordinate the translation work with the development process, we have developed a web-based tool named POLish. With POLish, translators can do their work from anywhere while still cooperating with a particular developer. POLish is released as Open Source and will be described in this talk.

As part of its transformation to agile, Klarna is enhancing its testing toolbox to better support Acceptance-Test-Driven Development (ATDD) and Continuous Validation. Originally working only with Yatsy and Eunit, we are now also utilising Common Test Framework, Fitnesse, Selenium and QuickCheck. The Klarna code base has grown organically for some years now, and so has the code dependencies. In order to create order out of chaos, we resorted to building a tool for dependency analysis and automatic code move. While this is not rocket science, we will share some experiences (and possibly also the actual tool).



David Evans
Agile Services
Director, SQS



Jordi Chacón
Serendipitous
Erlang user



Tobbe Tornkvist
Enthusiastic
Erlang user



Jia Wang
Uncommon tester



Tobias Lindahl
Another Erlang enthusiast



Code Management

Tools @
klarna
Simpler Safer More Fun

Testing

Translation
"Polish"

Tools @
klarna
Simpler Safer More Fun



Code Management



Tools @
klarna
Simple. Safe. Smart.



David
Lindahl

Testing

Tim
Wright

David
Lindahl

Translation
"Polish"



Tobias Lindahl

Code Management

The problem
The solution

The analysis
The solution



The Problem

- Organically grown code.
- Ad-hoc application structure.
- No clear cuts in functionality.

Our solution

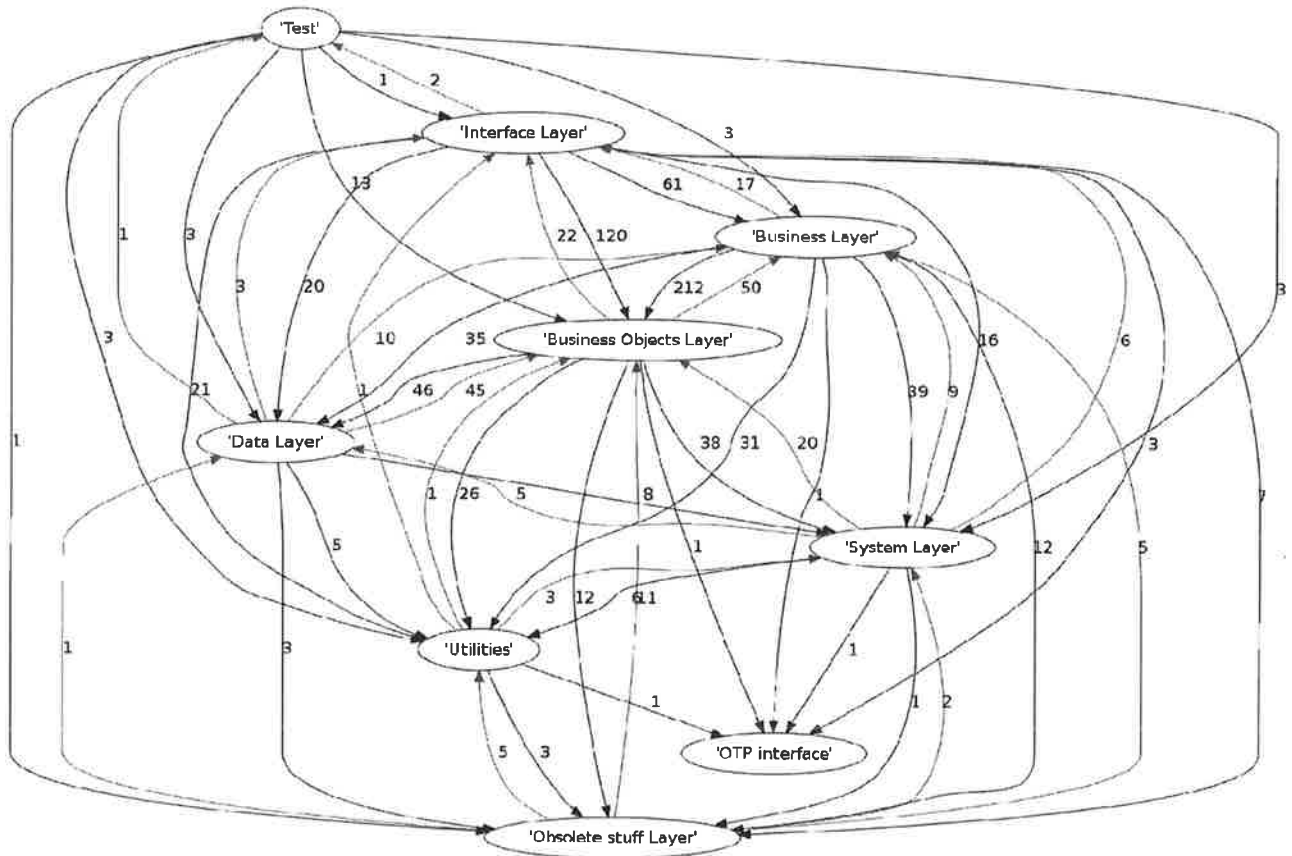
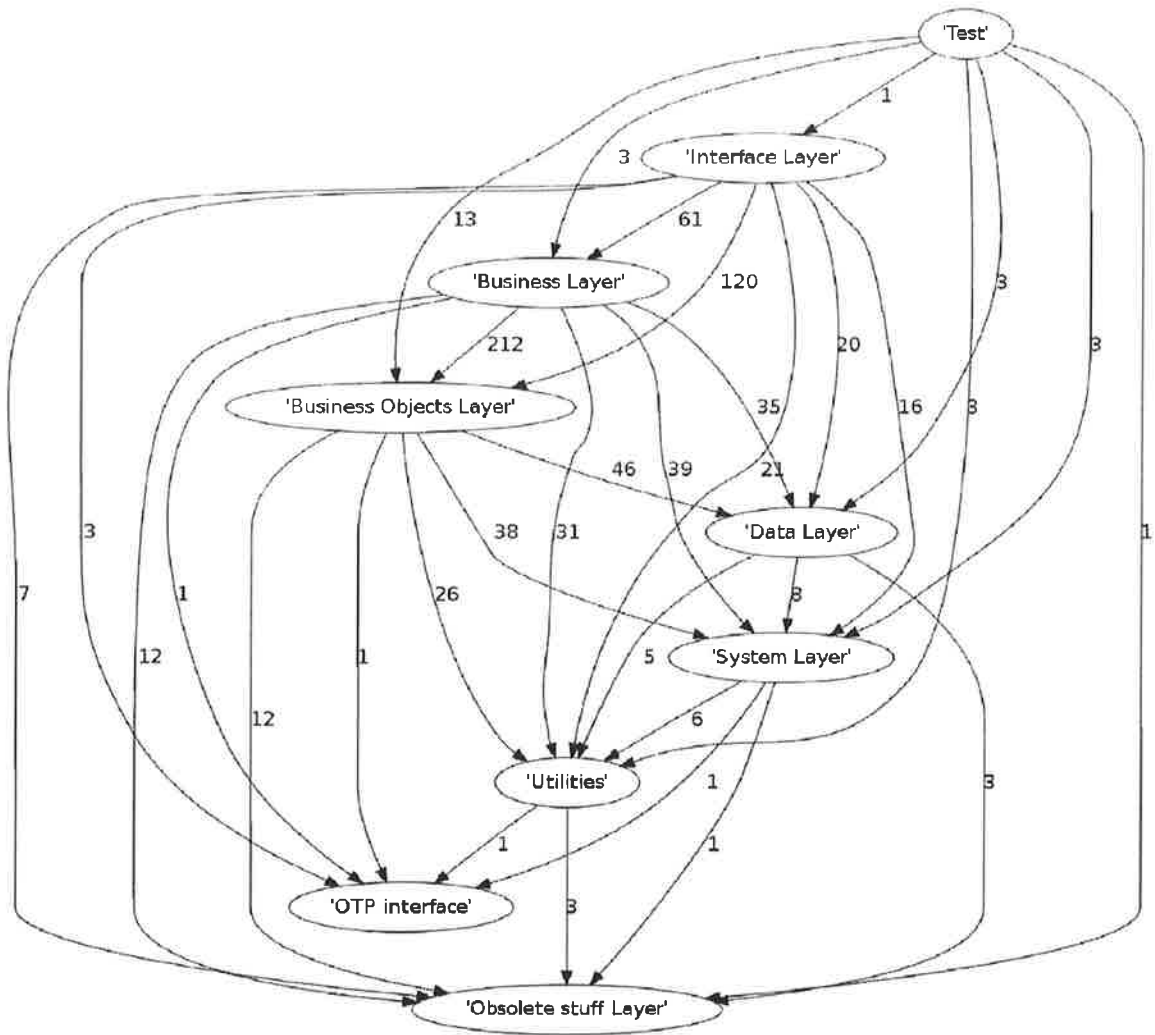
- Move existing modules to new applications.
- Group applications in clusters (layers).
- Put constraints on cluster dependencies.
- Refactor code to satisfy constraints.
- Profit!

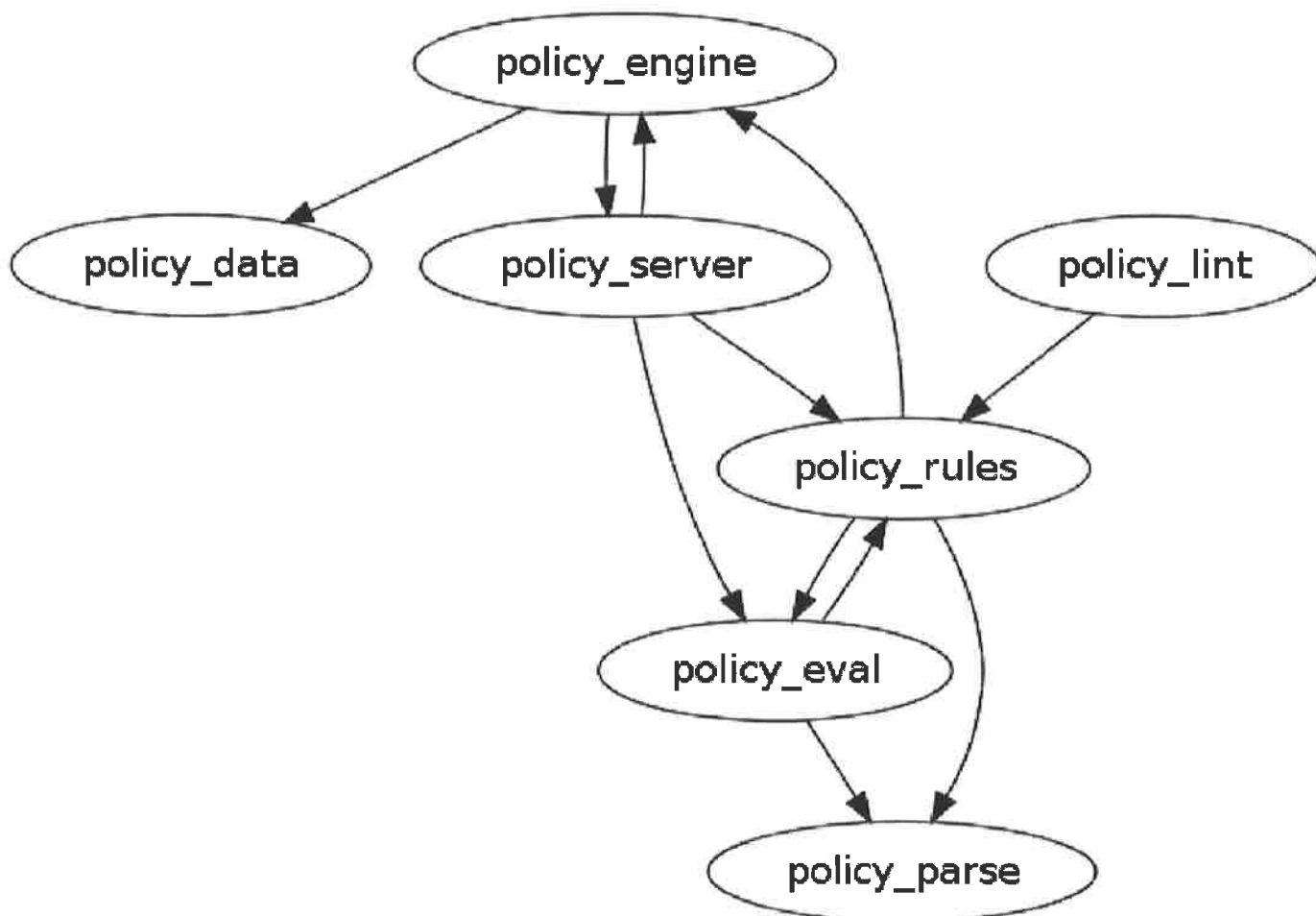
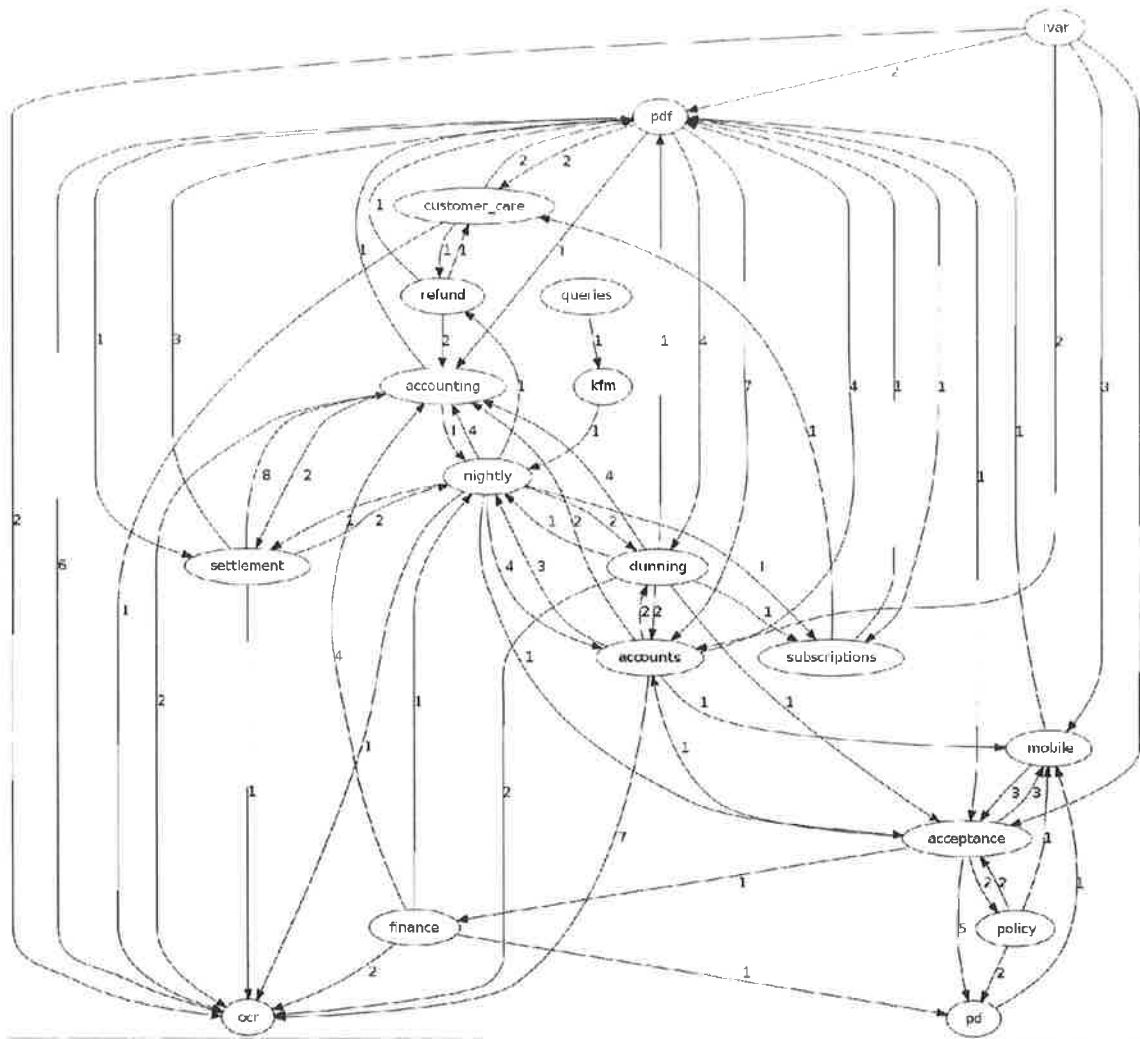
The Analysis

- Find dependencies between modules.
- Divide into applications and layers
- Write boring statistics to a file
- Make extremely cool graphs

The Action

- Generate shell scripts for
 - Creating new application structure
 - Moving the code
 - Patching include paths, Makefiles, etc







Code Management



Tools @ klarna



Torbjörn Törnkvist



Jordi Chacon

Translation "POLish"

Background

Using GetTest

The Problem

The Solution

About POLish

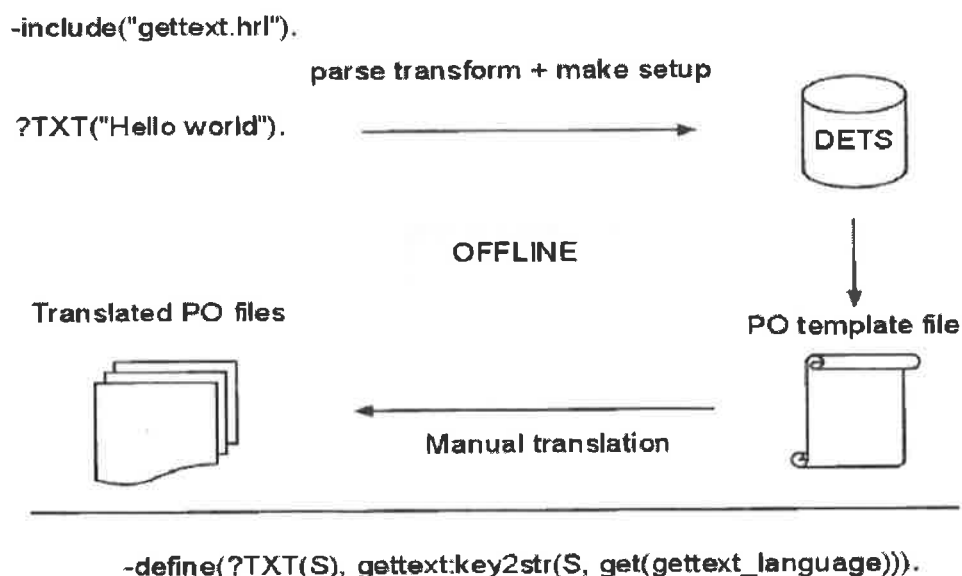
DEMO TIME!

New features coming... or not

Background

- Klarna produces ~50 different PDF documents
- Klarna is serving ~5000 Estores + our own customer care
- We have about ~5000 texts that need to be translated from Swedish to: Norwegian, Finnish, Danish, German, Dutch, English...
- On average, one person working 100% on text changes

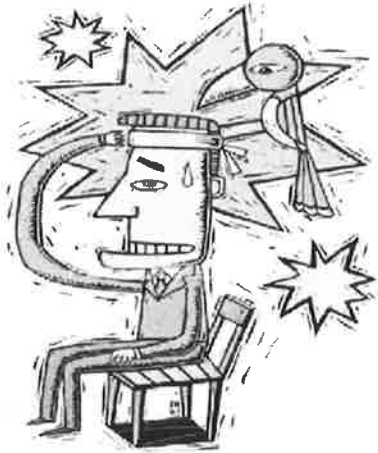
Using GetText



ONLINE

The Problem

is causing increasing headache!



- manually editing PO files is a source of problems
- 25.000-line PO files emailed back and forth
- unfriendly for translators to work on these huge po files
- texts always ended up being taken live with only the Swedish version available
- our customers weren't really happy about that...

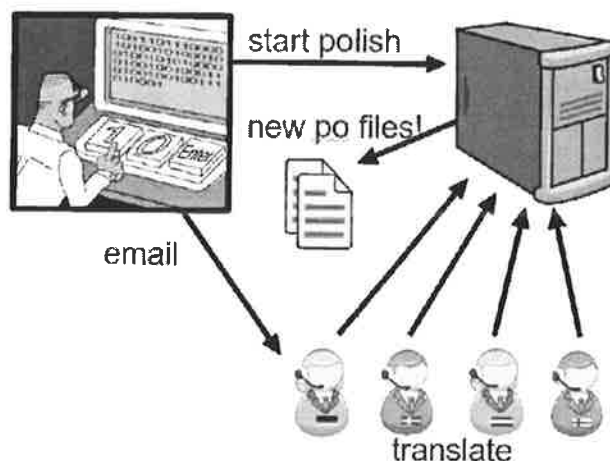
The Solution

A developer is hacking in his/her branch.

Adds a ?TXT macro containing a new Swedish text.

It should be translated to all languages!

What should the developer do?



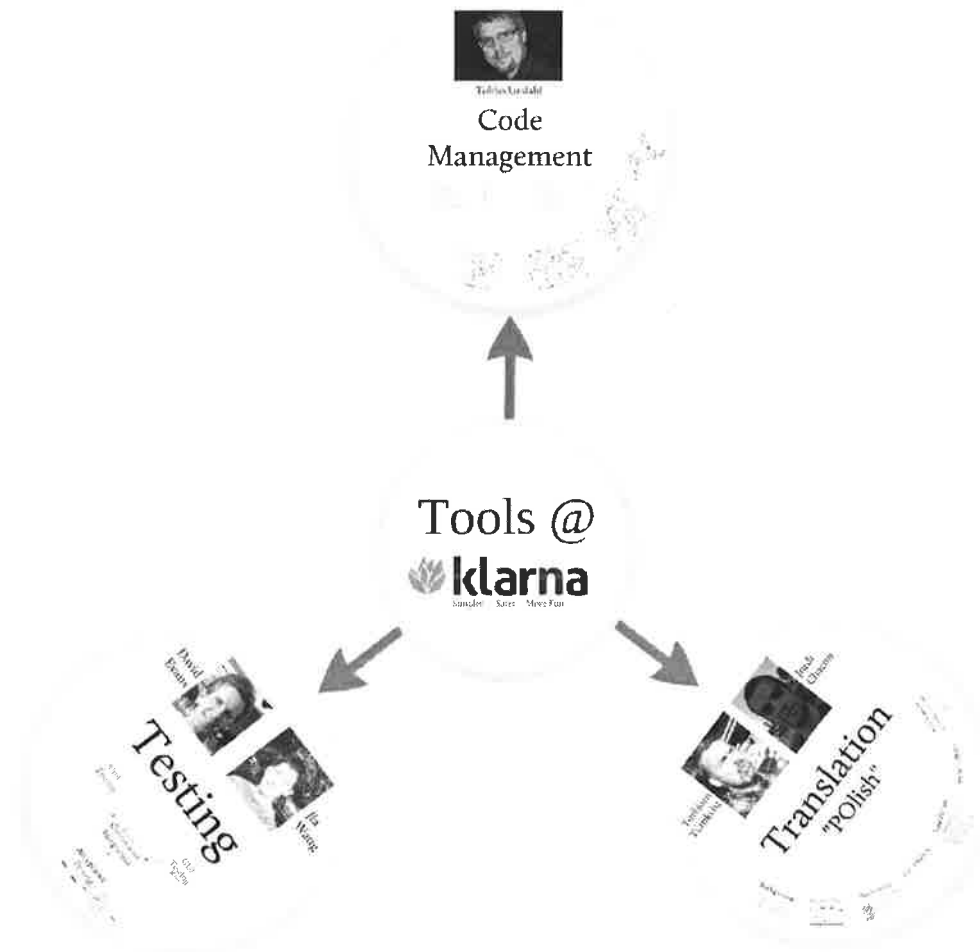
About POLish

- POLish is a utility developed by Klarna
- Web-GUI built on top of gettext to easily access PO-files
- Provides a simple and friendly interface for translators
- Log in via OpenID
- Keeps track of what is untranslated
- Allows translators to search for texts
- Ensures correctness of translations
- Started with a simple commands
- Has been used for the last three months

DEMO TIME!

New features coming... or not

- Translation memory
- Spell checker
- Online service



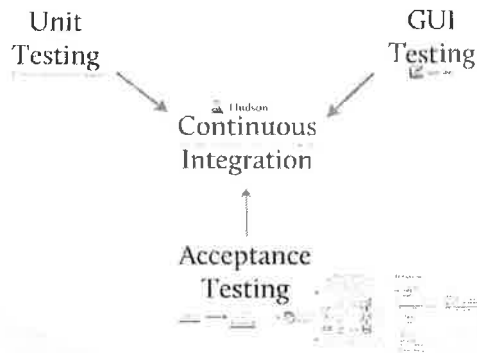
David
Evans



Jia
Wang



Testing



Unit
Testing
JUnit (authored by Richard Carlsson)

GUI
Testing
Selenium

Hudson
Continuous
Integration
JUnit, JUnit4, TestNG, Selenium, FitNesse, FitNesse-HTML, FitNesse-HTML-TestRunner, FitNesse-HTML-TestRunner-TestRunner

Acceptance
Testing
YATSY, Common Test Framework, FitNesse & Sling



Hudson

Continuous

Git SCC Integration Continuous Build Test Suite Execution Code Coverage Analysis

Integration



Unit

Testing

EUnit (authored by Richard Carlsson)

GUI Testing



Selenium

Acceptance

Testing

YATSY
<http://code.google.com/p/yatsy/>

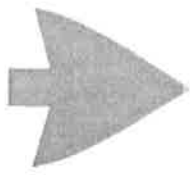


Common
Test
Framework

http://www.seleniumhq.org/docs/5_summary_test/



FitNesse
& Slim



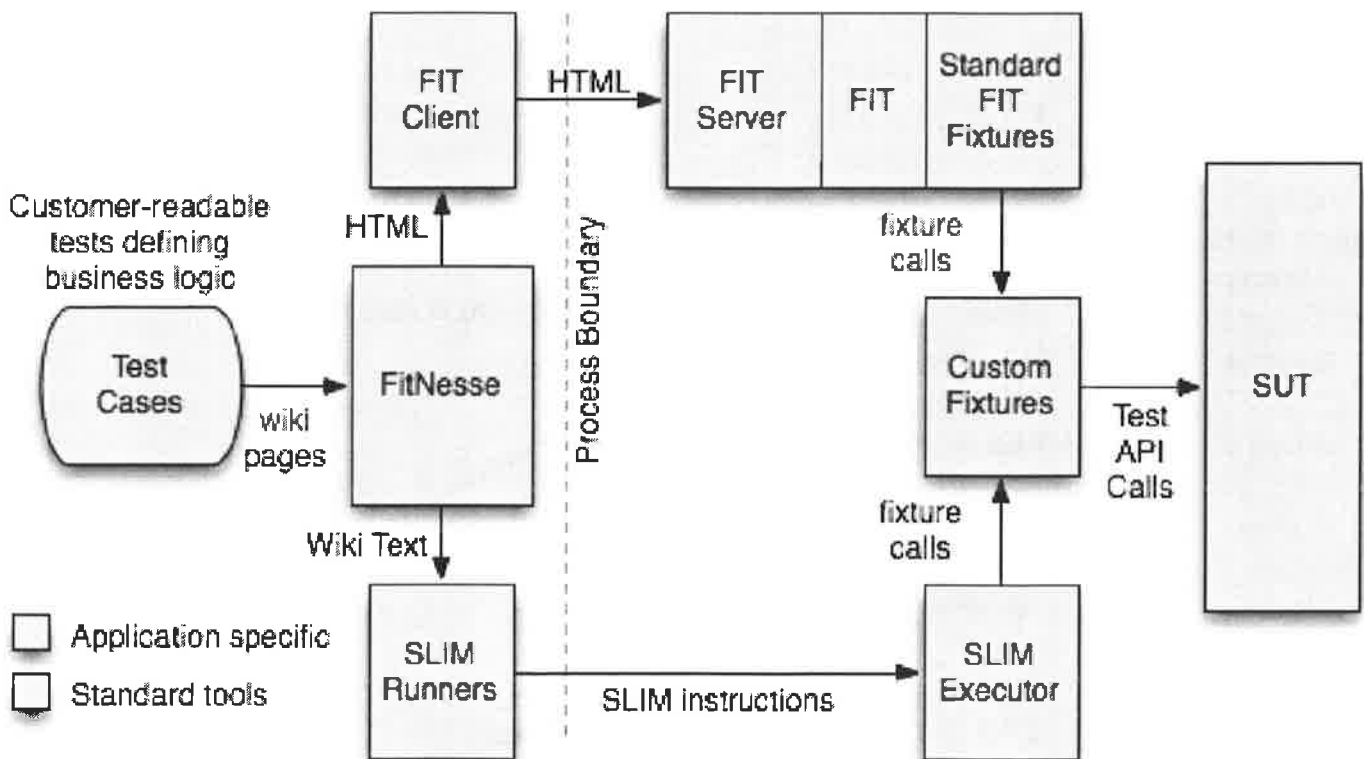
FitNesse & Slim

Custom tests
basir



- Ap
- Sta

- FIT server automates test execution at API level
- FIT implemented in Java, .NET, Ruby, Python



- Application specific
- Standard tools

What is Slim?

- Simple Fitness test runner protocol to replace FIT

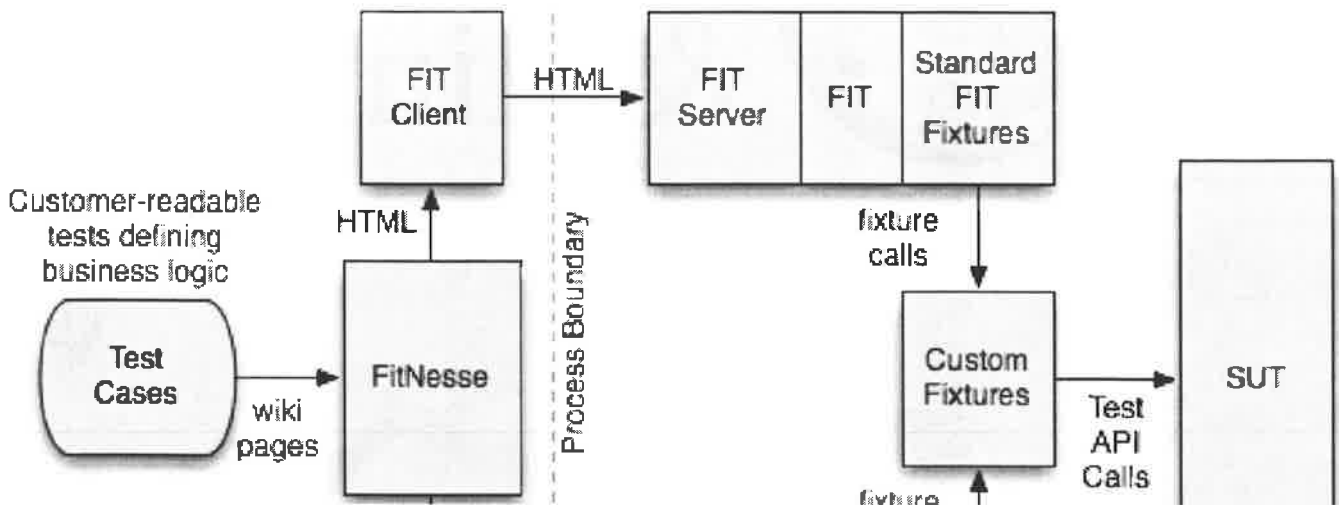
What is Fitness?

Framework designed for agile acceptance testing

- Supports 'Specification by Example'
- Mix plain-language documentation with test data

Wiki-based test authoring wrapper for FIT

- FIT server automates test execution at API level
- FIT implemented in Java, .NET, Ruby, Python



[FrontPage.](#)

ErlangTriangleDemo [add child]

Test

Edit

Properties

Refactor

Where Used

Search

Files

Versions

Recent Changes

User Guide

[Click here for Technical Implementation details](#)

I WANT A FUNCTION THAT RETURNS THE TYPE OF TRIANGLE PRODUCED, GIVEN THE LENGTHS OF THREE SIDES (A, B AND C)

The valid types are:
 All sides equal: Equilateral
 Two sides equal: Isosceles
 No sides equal: Scalene

For example:

another_type_fixture			
a	b	c	type?
2	2	2	Equilateral
2	2	3	Isosceles
2	3	4	Scalene

Plain text (ignored)

Test Data (executed)

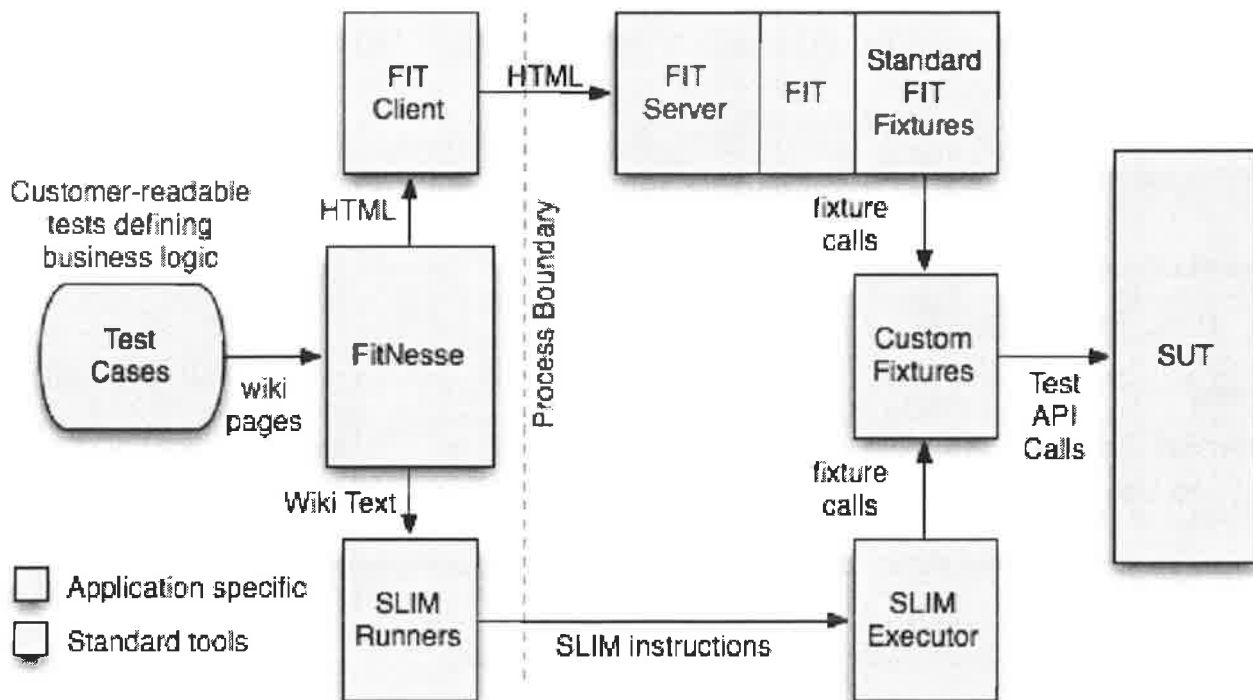
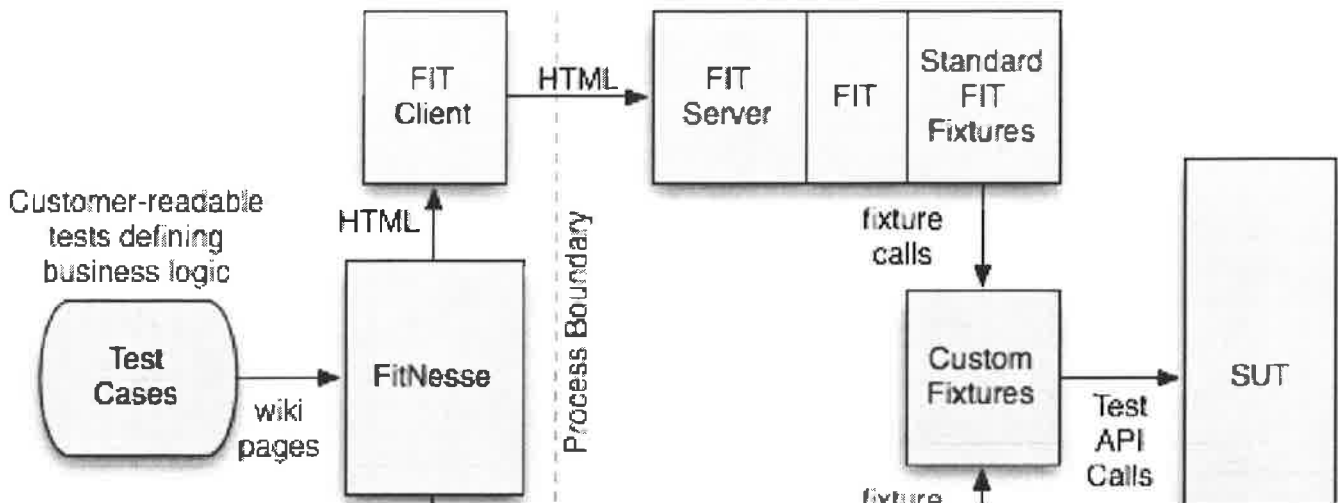
What is Fitness?

Framework designed for agile acceptance testing

- Supports 'Specification by Example'
- Mix plain-language documentation with test data

Wiki-based test authoring wrapper for FIT

- FIT server automates test execution at API level
- FIT implemented in Java, .NET, Ruby, Python



What is Slim?

- Simple Fitness test runner protocol to replace FIT
- Does not parse HTML tables or compare test data
- Much simpler to port to new languages (e.g. SlimErl)

```

emacs@hundhaj
File Edit Options Buffers Tools Erlang Help
-----
%%
%% This fixture code is generated by slim_generator
%%
-----
-module(triangle_type_fixture).

%% Export Callback Functions
-export([ new/0
         , execute/1]).
%% Export setters and assert functions
-export([ setA/2
         , setB/2
         , setC/2
         , type/1
         ]).

-record(triangle_type_fixture, { a
                               , b
                               , c
                               }).

%%
%% Callback functions for FitNesse
%%
-----
new() -> #triangle_type_fixture{}.

%% Setter Functions
setA(Rec, A) ->
    voidFunction(Rec#triangle_type_fixture{a = A}).
setB(Rec, B) ->
    voidFunction(Rec#triangle_type_fixture{b = B}).
setC(Rec, C) ->
    voidFunction(Rec#triangle_type_fixture{c = C}).

execute(Record) -> voidFunction(Record). %% TODO do

%%
%% Assert Functions
%%
-----
type(#triangle_type_fixture{a = A,
                             b = B,
                             c = C} = Rec) ->

%% TODO
B = triangle_type:type(A, B, C),
echoString(Rec, R).

```

Birth Date: The

```

get_person
pno
$JiaPno->[800414-0920]

```

THIS SLIGHTLY MORE COMPLEX EXAMPLE CREATES A NEW PERSON, THEN RETRIEVES THEM

Note that in this test we need to refer to:

► [Included page: CountryCodes \(edit\)](#)

setup_person								
Nationality	Age	First Name	Last Name	Street	Zip Code	City	Country	pno?
2	30	Jia	Wang	Skt. Eriksgatan 112	54321	Stockholm	209	\$JiaPno->[800414-0920]

Now lookup this person and check that we stored the correct details and that we concatenate the names and address elements correctly.

Pacc: By default, a new person will not have a Personal Account (pacc).

Birth Date: The first 6 digits of the Pno determine the date of birth of the person.

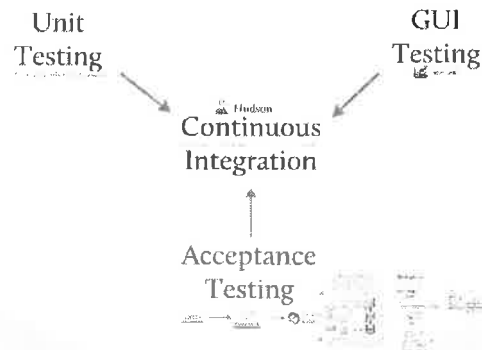
get_person			
pno	has_pacc_account?	birth_year?	get_address?
\$JiaPno->[800414-0920]	false	1980	Skt. Eriksgatan 112 54321 Stockholm Sweden

David
Evans



Jia
Wang

Testing



Tools @



Simpler Safer More Fun

RefactorErl: a Source code analyzer and transformer tool

Abstract

RefactorErl is a source code analyzer and transformer tool aimed at refactoring Erlang software. The tool itself is written in Erlang with a unique approach: semantic analysis results are stored in a Mnesia database to avoid repeated analysis of the same source code, and simple, syntax-based manipulations are available that hide the details of handling separators, comments, and code layout.

Beside the 24 implemented refactoring transformations the tool has a complex analyzer framework. For example, it provides data flow analysis, dynamic function call detection, side-effect analysis, etc and a user level query language to query semantic information or structural complexity metrics about Erlang programs.

Zoltán Horváth

Professor and supervisor of RefactorErl

Zoltán Horváth is Professor at, and Head of, the Department of Programming Languages and Compilers and Vice-Rector for International Affairs at Eötvös Loránd University in Budapest, Hungary. He defended his habilitation thesis in 2004; the title of his thesis was "Verification and Semantics of Mobile Code Written in a Functional Programming Language". Current topics researched under his supervision include language design, construction of programming language processing tools, formal methods and scheduling in grids. He has been supervising the RefactorErl project since 2005.



Melinda Tóth

Researcher at ELTE and RefactorErl project leader
Eötvös Loránd University, Budapest, Hungary

Melinda Tóth is a second year PhD student at the Eötvös Loránd University of Budapest, Hungary. She has been working with Erlang since 2007 with the RefactorErl project. Melinda received her master's degree in Computer Science in 2009 from Eötvös Loránd University. Both her bachelor and master theses were based on Erlang and function related refactorings. In 2008 she spent five months at University of Kent where she worked with Wrangler. Melinda teaches Parallel Programming and Functional Languages at the University: Haskell and Erlang. Her PhD research field is about data and control flow graphs for functional languages, and impact analysis of refactorings.



RefactorErl: a source code analyser and transformer tool ¹

Melinda Tóth and Zoltán Horváth

Department of Programming Languages and Compilers
Faculty of Informatics
Eötvös Loránd University

November 15, 2010

¹Supported by KMOP-1.1.2-08/1-2008-0002, ELTE IKKK, and Ericsson Hungary

Melinda Tóth and Zoltán Horváth

RefactorErl: a source code analyser and transformer tool

Outline

- 1 RefactorErl
 - Framework
 - The tool
- 2 Erlang Semantic Query Language
 - Path expressions
 - Language elements
 - Syntax
 - Examples
- 3 Summary

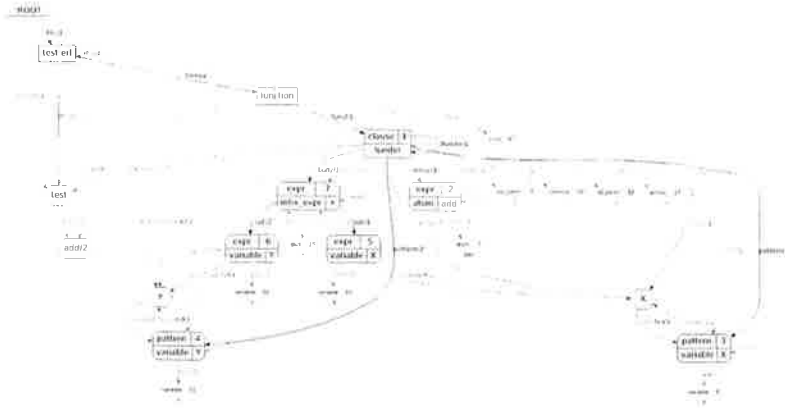
RefactorErl

- Semantic Program Graph: lexical layer + AST + semantic layer
- Efficient information retrieval
- New semantic analyser framework
 - Incremental analysis
 - Modular structure
 - Asynchronous parallel execution
 - 7 times faster initial loading (Intel Core2 Quad, 2.4 GHz)
 - Side effect analysis

The tool RefactorErl

- Store semantic information – Mnesia
- UI: Emacs, Eclipse, Erlang shell, Web based, Command line
- Platform for source code transformation
 - Rename
 - Move definition
 - Expression structure
 - Function interface
- Non-refactoring applications – different analysis
 - Call graph visualization
 - "Bad smell" detection
 - Clustering
- Query Language

Example graph



Path expressions

- Support information gathering for refactoring
- Depends on the representation

```
path() = [PathElem]
```

```
PathElem = Tag | {Tag, Index} | {Tag, Filter} |
           {intersect, node(), Tag}
Tag       = atom() | {atom(), back}
Index     = integer() | {integer(), integer()} | {integer(), last}
Filter    = {Filter, 'and', Filter} | {Filter, 'or', Filter} |
           {'not', Filter} | {Attrib, Op, term()}
Attrib    = atom()
Op        = '==' | '/=' | '<=' | '>=' | '<' | '>'
```

Path expression example

- List the defined functions:

```
path(Module, [{form, {type, '==', func}}])
```

Path expression example

- List the defined functions:

```
path(Module, [{form, {type, '==', func}}])
```

- Set of library module
- Extended evaluation framework

Path expression example

- List the defined functions:
`path(Module, [{form, {type, '==', func}}])`
- Set of library module
- Extended evaluation framework
- `exec(Module, modlib:function())`

Semantic query language

- A user level query language to get information about the Erlang source
- Language concepts:
 - Entities
 - Selectors
 - Properties
 - Filters
- Custom query or predefined query

Syntax of the queries

- `semantic_query ::= initial_selection ['.' query_sequence]`

Syntax of the queries

- `semantic_query ::= initial_selection ['.' query_sequence]`
- `initial_selection ::= initial_selector ['[' filter ']]'`
- `query_sequence ::= query ['.' query_sequence]`
- `query ::= selection | iteration | closure | property_query`

Syntax of the queries

- `semantic_query ::= initial_selection ['.' query_sequence]`
- `initial_selection ::= initial_selector ['[' filter '']]`
- `query_sequence ::= query ['.' query_sequence]`
- `query ::= selection | iteration | closure | property_query`
- `selection ::= selector ['[' filter '']]`
- `iteration ::= '{' query_sequence '}' int ['[' filter '']]`
- `closure ::= '(' query_sequence ')' int ['[' filter '']]`
`'(' query_sequence '+' ['[' filter '']]`
- `property_query ::= property ['[' filter '']]`

Semantic query examples

```

calc(...) ->
  A = ...,
  ...
  {A, ...}.

test(...) ->
  Calc = calc(...),
  ...,
  {First, ... } = Calc,
  First.

run() ->
  some_value = test(...).

```

Semantic query examples

```

calc(...) ->
  A = ...,
  ...
  {A, ...}.

```

- Value of a variable
@expr.origin

```

test(...)->
  Calc = calc(...),
  ...,
  {First, ... } = Calc,
  First.

```

```

run() ->
  some_value = test(...).

```

Semantic query examples

```

calc(...) ->
  A = ...,
  ...
  {A, ...}.

```

- Value of a variable
@expr.origin

```

test(...)->
  Calc = calc(...),
  ...,
  {First, ... } = Calc,
  First.

```

- Call chain
@fun.{called_by}+ or
@fun.{calls}+

```

run() ->
  some_value = test(...).

```

Semantic query examples

```

calc(...) ->
  A = ...,
  ...
  {A, ...}.

```

- Value of a variable
@expr.origin

```

test(...)->
  Calc = calc(...),
  ...,
  {First, ... } = Calc,
  First.

```

- Call chain
@fun.{called_by}+ or
@fun.{calls}+
- Side effect
@fun.dirty

```

run() ->
  some_value = test(...).

```

Dynamic function calls

```

sum([]) ->
  0;
sum([H|T]) ->
  S = sum(T),
  H + S.

```

```

test1(List)->
  Fun = sum,
  test2(?MODULE, Fun, List).

```

```

test2(Mod, Fun, List)->
  apply(Mod, Fun, [List]).

```

Dynamic function calls

```

sum([]) ->
  0;
sum([H|T]) ->
  S = sum(T),
  H + S.

```

• Function references

```

test1(List)->
  Fun = sum,
  test2(?MODULE, Fun, List).

```

@fun.refs

```

test2(Mod, Fun, List)->
  apply(Mod, Fun, [List]).

```

Dynamic function calls

```

test1(List, ArgList)->
  Fun = sum,
  test2(?MODULE, Fun, [List]),
  test2(?MODULE, other, ArgList).

```

```

test2(Mod, Fun, Args)->
  apply(Mod, Fun, Args).

```

```

sum([]) -> ...

```

```

other(A) -> ...

```

```

other() -> ...

```

Dynamic function calls

```
test1(List, ArgList)->
  Fun = sum,
  test2(?MODULE, Fun, [List]),
  test2(?MODULE, other, ArgList).
```

```
test2(Mod, Fun, Args)->
  apply(Mod, Fun, Args).
```

- Dynamic function references
@expr.dynfunref

```
sum([]) -> ...
```

```
other(A) -> ...
```

```
other() -> ...
```

Identifying callback functions

```
request_add()->
  gen_server:call(Module, {req_add, {Phone, Name}}).
```

```
handle_call({req_add, {Phone, Name}}, From, LoopData) ->
```

```
...
```

Callback functions

- mods[name == "gen_server"].
 funs[name == call and arity == 2].
 refs[type == application].
 param[index == 3].

Identifying callback functions

```
request_add()->
  gen_server:call(Module, {req_add, {Phone, Name}}).

handle_call({req_add, {Phone, Name}}, From, LoopData) ->
  ...
```

Callback functions

- mods[name == "gen_server"].
 - funs[name == call and arity == 2].
 - refs[type == application].
 - param[index == 3].
- mods[name == "CallBackMod"].
 - funs[name == handle_call and arity == 3].
 - args[index == 1]]

Checking coding conventions

- mods[line_of_code > 400],
 - mods.funs[line_of_code > 20]
- @file.funs[max_depth_of_cases > 2],
 - @file.max_depth_of_cases,
 - mods[max_depth_of_cases > 2],
 - mods.funs[max_length_of_line > 80]
- mods.funs[no_space_after_comma > 0]
- mods.funs[is_tail_recursive == 0]

Embedded queries

Without:

```
@file.functions.  
  variables[name=="File"].  
  function_definition
```

With:

```
@file.functions  
  [.variables[name=="File"]]
```

Other example:

```
mods[name==mydb]  
  .fun[exported]  
  .refs  
    [.sub[index==1 and type==tuple]  
     .sub[index==1 and value==mykey]]
```

Summary and Future work

- RefactorErl: source code analyzer and transformer
- Query language:
 - understand source code
 - debug information
 - maintenance
- Save and reuse query
- Extend the language: recursion, if, variables

<https://plc.inf.elte.hu/erlang>

Graham Crowe
Ericsson AB



Using Erlang to test non -Erlang products

Abstract

Complex products such as Radio Base Stations require complex Test Environments to achieve test automation with high test coverage. There are many compelling reasons for selecting Erlang/OTP for this task, even if the product itself is not developed with erlang.

Biography

Graham Crowe has been working for Ericsson since the mid 90s, mainly testing Mobile Telecommunication Systems, including GSM, WCDMA and LTE. His main focus in recent years has been Test Automation. He didn't discover Erlang/OTP until 2005 but has been an active user of it ever since.

David Almroth
Server side game architect and
founder of PikkoTekk



PikkoServer, how to scale game servers
and enable player density

Abstract

The dream to build a game server in Erlang has been around for many years. Erlang has been used in games on the server side in some projects but it is not common, not yet. In the gaming industry, C++ has been the favorite tool for most developers when they need performance. This is now changing rapidly. Game programmers are learning and using new technologies, and Erlang is one of them.

In this talk David Almroth will present an ALG product for the gaming industry based on Erlang. It is a middleware product to be used on the server side and where the architecture is transformed from single-core to many-core. The product is used by server side programmers without having to learn any Erlang, which makes it easy to both embrace in the organisation and integrate in the game engine. Allowing a wider use of the power of Erlang.

Biography

After ten years of work with Enterprise Java consulting in banking and finance, David has switched to Erlang and the gaming industry. Games has always been at the forefront of technological advancement and it is in the game industry we can expect some very advanced many-core applications in the near future

He started the company PikkoTekk and has been the leader of a team building an Erlang application-level gateway (ALG), called Pikko Server, for massive multiplayer games. How to utilize a manycore architecture on massive game servers has been Davids area of focus and his team has successfully developed a unique solution that allow for this. Allowing for new types of transaction intensive virtual worlds.

Joseph Wayne Norton
Erlang Enthusiast



Hibari - Key Value Bigdata Store

Abstract

Hibari is a production-ready, distributed, key-value, big data store. Hibari uses chain replication for strong consistency, high-availability, and durability. Hibari has excellent performance especially for read and large value operations. Hibari is implemented in Erlang. As of July 2010, Hibari is open-source software under the Apache 2.0 license.

We'd like to talk about Hibari's major features and how Hibari has been deployed as part of large commercial Webmail system in Asia. We also intend to share recent benchmarking results (gathered by Yahoo!'s YCSB benchmarking tool) of Hibari and a few other open source key-value stores. The target audience are developers.

Biography

Joe Norton is a technical manager, system architect, developer, and Erlang enthusiast active in the mobile industry. Based out of Tokyo, Japan, he works for Gemini Mobile.



Hibari/Erlang/ NOSQL for BIGDATA

November 2010

Gemini Mobile Technologies

Hibari Open Source project: <http://sourceforge.net/projects/hibari/>



Gemini Mobile Technologies

- Founded: July, 2001
- Offices: San Francisco, Tokyo, Beijing
- Investors:
 - Goldman Sachs, Mitsubishi-UFJ, Mizuho, Nomura, Ignite, Access, Aplix
- Accomplishments:
 - Messaging Products
 - Provide MMSC to 3 out of 4 Carriers in Japan (DoCoMo, Softbank, eMobile)
 - Largest MMSC in the world (Softbank Japan)
 - OEM to Alcatel-Lucent and ByteMobile
 - NOSQL / Big Data
 - 2006: First Mobile 3D SNS (Softbank, China Unicom, iPhone App)
 - 4/2010: WebMail, Japanese Mobile Carrier & Internet Provider
 - 7/2010: Hibari Open Source



Customers



Hibari (= Cloud Birds)



What is Hibari?

- Hibari is a production-ready, distributed, key-value, big data store.
 - China Mobile and China Unicom - SNS
 - Japanese internet provider - GB mailbox webmail
 - Japanese mobile carrier - GB mailbox webmail
- Hibari uses chain replication for strong consistency, high-availability, and durability.
- Hibari has excellent performance especially for read and large value operations.
- Hibari is open-source software under the Apache 2.0 license.



5

Environments

- Hibari runs on commodity, heterogeneous servers.
- Hibari supports Red Hat, CentOS, and Fedora Linux distributions.
 - Debian, Ubuntu, Gentoo, Mac OS X, and Free BSD are coming soon.
- Hibari supports Erlang/OTP R13B04.
 - R14B is coming soon.
- Hibari supports Amazon S3, JSON-RPC-RFC4627, UBF/EBF/JSF and native Erlang client APIs.
 - Thrift is coming soon.



6

Why NOSQL?

We needed to build a scalable, high performance web mail system

- Big Data
 - A few million end users from the start
 - A few billion messages in a few month
 - Hundreds of TB data
- Low Cost requirements
 - Customer's business model (Freemium)
 - Distributed >50 PC servers
 - No need for rich and expensive functions of SQL
- Continuous growth of data in the storage
 - Elasticity to expand capacity due to increasing data



What were customer's needs?

- Durability
 - Data loss (e.g., messages, metadata) is not acceptable
- Strong Consistency
 - Because of interactive sessions, consistency is required
- Low Latency
 - <1 sec response time for end user transactions
- High Availability
 - As a branded service to the end user, service must always be available.
- Read Heavy
 - Many more read than write operations
- Big Data and Data size highly variable
 - Large GB mail box as service differentiator was required
 - Mail messages range from a few bytes to many MB attachments



How does Hibari address these needs?

- **Durable updates**
Every update is written and flushed to stable storage (fsync() system call) before sending acknowledgments to the client.
- **Consistent updates**
After an update is acknowledged, no client can see an older version. "Chain Replication" is used to maintain consistency across all replicas.
- **High Availability**
Each key can be replicated multiple times. As long as one copy of the key survives, all operations on that key are permitted.
- **Lockless API**
Locks are not required for all client operations. Optionally, Hibari supports "test-and-set" of each key-value pair via an increasing (enforced by the server) timestamp value.
- **Micro-transactions**
Under limited circumstances, operations on multiple keys can be given transactional commit/abort semantics.



9

Why Erlang?

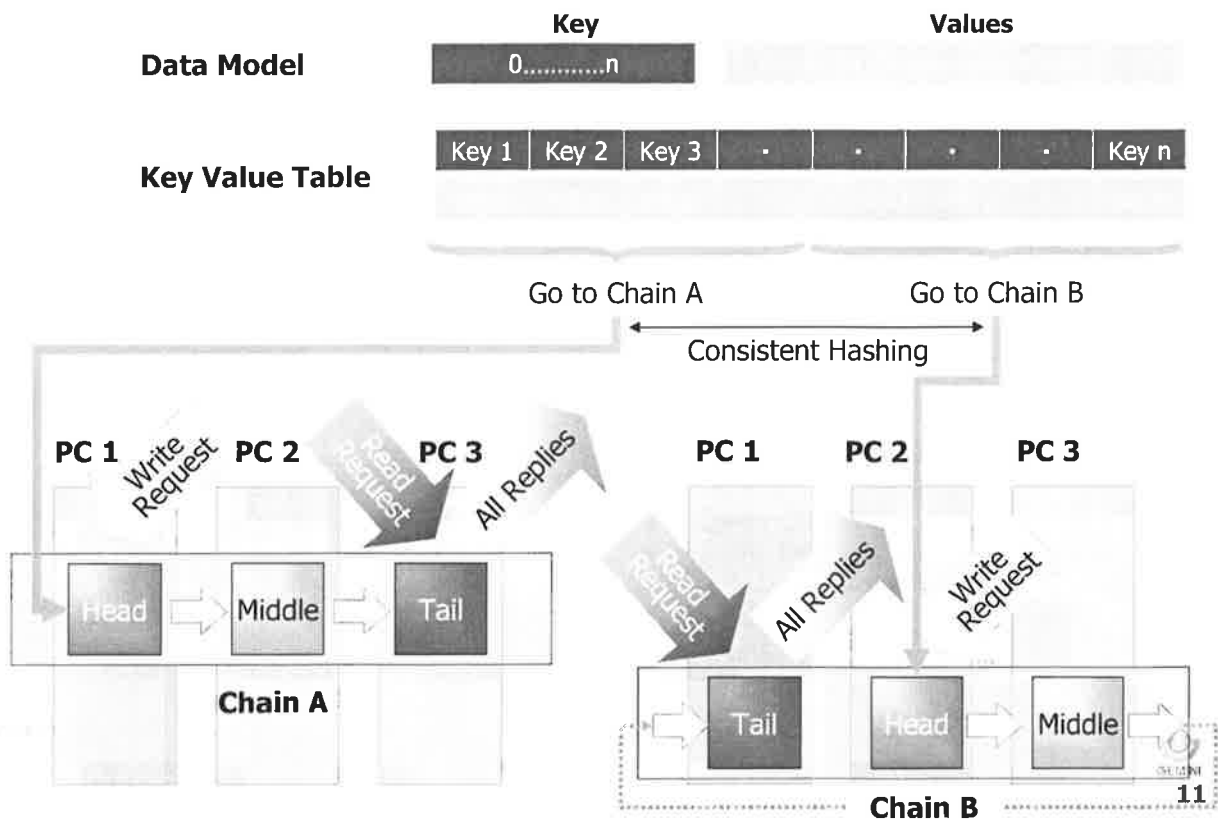
- Concurrency and Distribution
- Robustness
- Efficient garbage collection
- Hot code and incremental upgrade
- Online tracing
- Efficiency and Productivity
 - **Small teams make big impact**
- Ericsson's support of Erlang/OTP is wonderful

Everything you need to build robust, high performance distributed systems



10

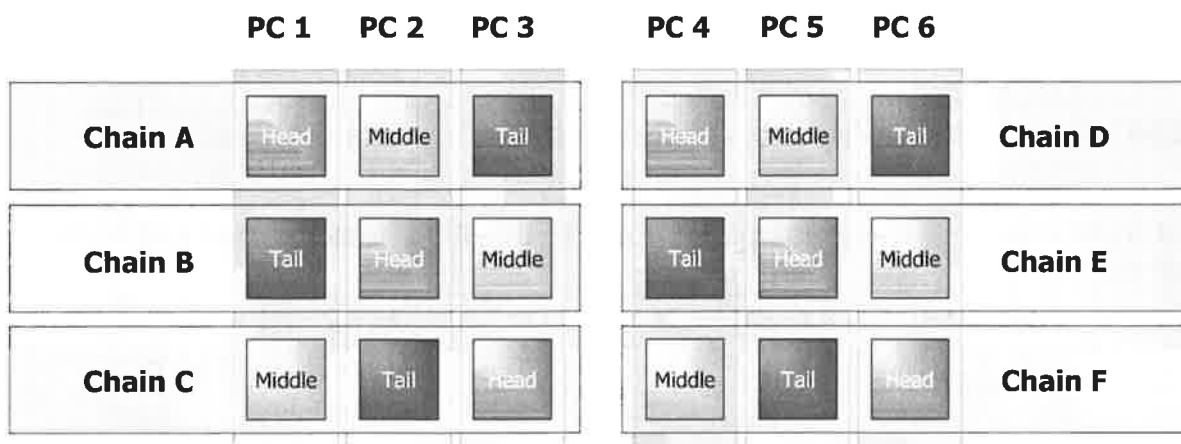
Chain Replication for Strong Consistency



Concept of "Chain"

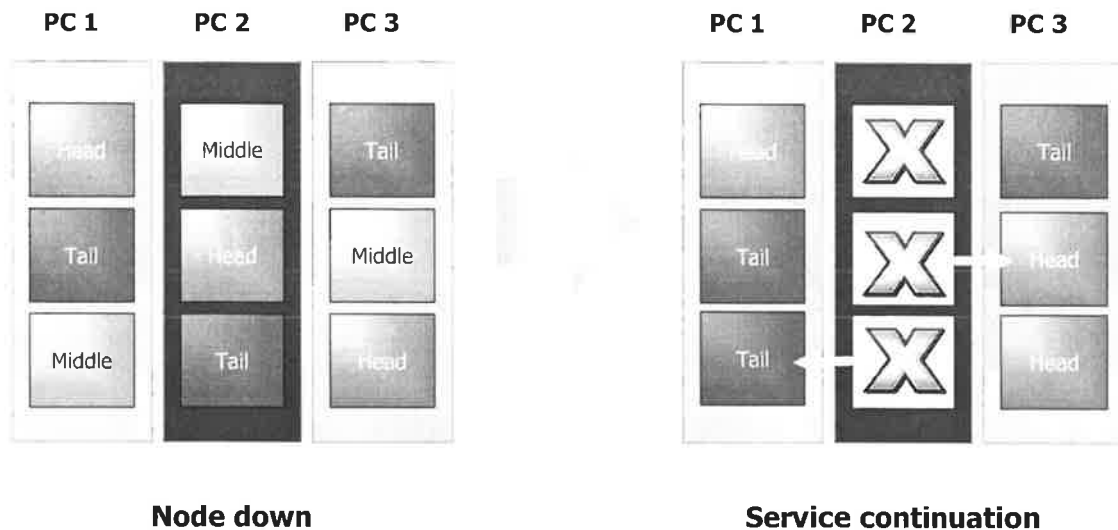
Evenly distributed load in multiple nodes

(Case of 3 replications/6 chains)



Chain Replication for High Availability and Fault Tolerance

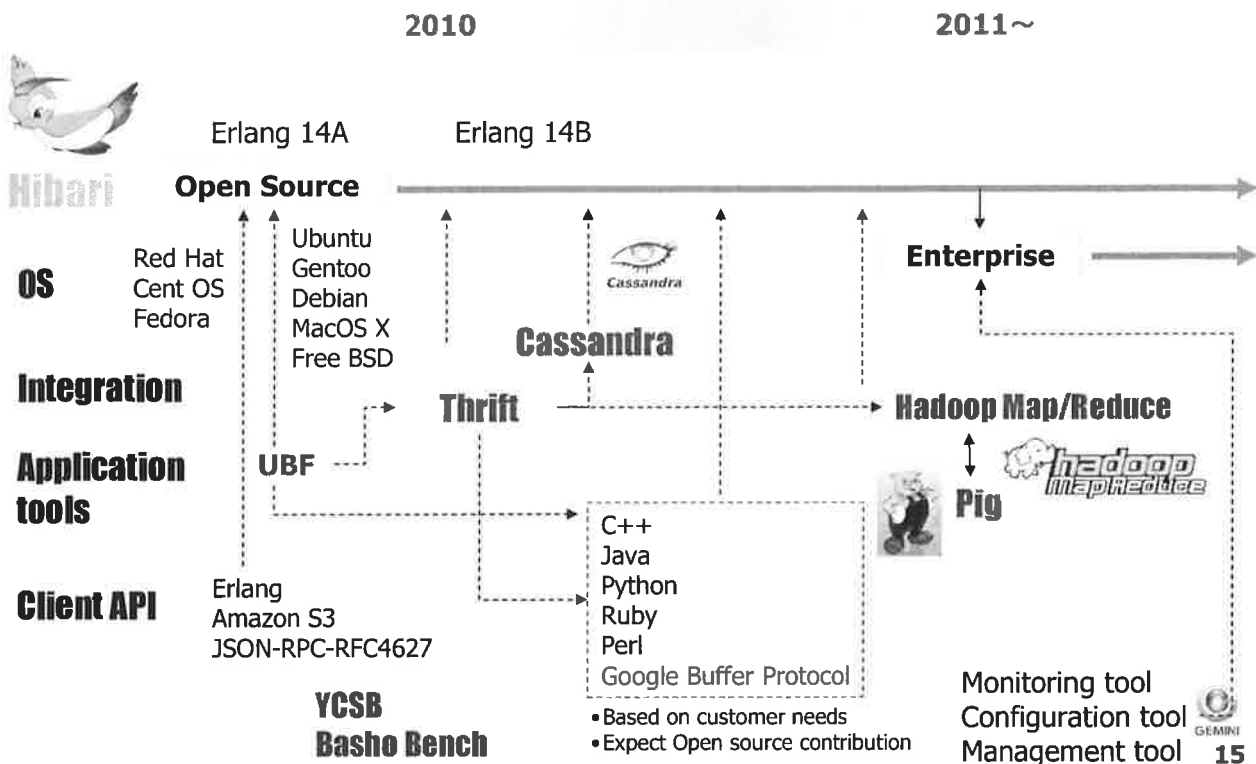
Failover mechanism



“Applications / Customer first” approach leads to mutual complements

FEATURE	(e.g.) Cassandra	Hibari
Data model	Column-oriented	Key-value
Data partitioning	Consistent hashing	Consistent hashing
Data consistency	Configurable	Yes
Data replication	Preference lists	Chain replication
Elasticity	Admin operations, Gossip, Data redistribution	Chain migration
Node health detection	Peer-to-peer monitor, gossip	Admin server monitors
O&M	nodetool, JMX	Admin UI with brick, chain health, statistics
Performance	write-optimized	read-optimized
Implementation	Java	Erlang
API	get/put/delete, scan, map/reduce, atomic row ops	get/put/delete, micro-transactions

"Not Only Hibari" Roadmap



Thank you



Hibari Open Source project: <http://sourceforge.net/projects/hibari/>

Twitter: <http://twitter.com/hibaridb>

Hashtag: #hibaridb

Slideshare: <http://www.slideshare.net/geminimobile>

Marcus Kern
Integrated mobile and digital
communications



Agile is Everything

Abstract

Working together across three different countries was challenging. Working with a language and data storing technology we never touched before seemed outright madness. Experiencing the joy to see it working on the day of our brick-wall deadline was only possible through working together as an Agile Team.

In this talk Kern will attempt to provide you with an insight to MIG's ambitious Mobile Gateway project. The good, the bad and the ugly.

Biography

Marcus Kern started his telecom career in 1998 when he co-founded voiceIT communications, a company re-selling voicemail services. Over the years that followed he became a Senior Technical Architect in Mobile Telecoms working for O2 and T-Mobile. As one of the founding members of Mobile Interactive Group, Kern has been with MIG as CTO since 2005. Over the last 12 months Kern lead the development of MIG's mobile messaging gateway. New to Erlang, MIG has combined established and new software development, storage and delivery concepts.

Simon Thompson
Creator of Wrangler and co-author of
Erlang Programming



Let's ProTest: Update from the Erlang and
property based testing project

Abstract

Update from the Protest Project

Biography

Simon Thompson is Professor of Logic and Computation in the Computing Laboratory of the University of Kent, where he has taught computing at undergraduate and postgraduate levels for the past twenty five years, and where he has been department head for the last six.

His research work has centered on functional programming: program verification, type systems, and most recently development of software tools for functional programming languages. His team has built the HaRe tool for refactoring Haskell programs, and is currently developing Wrangler to do the same for Erlang. His research has been funded by various agencies including EPSRC and the European Framework programme. His training is as a mathematician: he has an MA in Mathematics from Cambridge and a D.Phil. in mathematical logic from Oxford.

He has written four books in his field of interest; Type Theory and Functional Programming published in 1991; Miranda: The Craft of Functional Programming (1995), Haskell: The Craft of Functional Programming (2nd ed. 1999) and Erlang Programming (with Francesco Cesarini, 2009). Apart from the last, which is published by O'Reilly, these are all published by Addison Wesley.

ProTest

property based testing

Simon Thompson, Clara Benac Earle
University of Kent, Universidad Politécnica de Madrid

Test
property based testing



Test
property based testing

ProTest goals

Integrate property-based testing into the development life cycle:

- Property discovery
- Test and property evolution
- Property monitoring
- Analysing concurrent systems



Property-based testing

Describe the required behaviour of a system using logical properties ...

... or abstract state machines.

Test the properties against random data.

Test machine compliance by random execution sequences.



ProTest tools



PULSE

**Exago
Onviso**



**QuviQ
QuickCheck**

State Chum



Focus for this talk



PULSE

**Exago
Onviso**



**QuviQ
QuickCheck**

State Chum



Wrangler



Interactive refactoring
tool for Erlang

Integrated into Emacs
and Eclipse / ErlIDE

Multiple modules

Structural, process,
macro refactorings

Clone
detection
+ removal

Improve
module
structure

Basic refactorings



Refactoring and testing

- Clone detection and elimination in test code
- Property extraction through clone detection and FSM inference.
- Refactoring code and tests: frameworks.
- Refactoring tests in a framework.

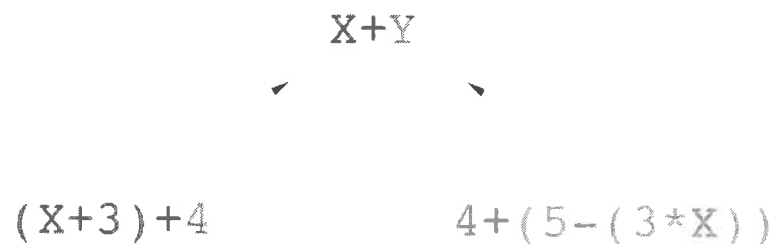


Refactoring and testing

- Clone detection and elimination in test code
- Property extraction theory: clone detection and FSM inference
- Refactoring code and tests: frameworks
- Refactoring tests in a framework



What is 'similar' code?



The anti-unification gives the (most specific) common generalisation.



Step 1

The largest clone class has 15 members.

The suggested function has no parameters, so the code is literally repeated.

```
*eri-output*
New Open Recent Save Undo Redo Cut Copy Paste Help
Similar detection finished with *** 43 *** clone(s) found.
/Users/simonthompson/Desktop/StockholmAug09/code/smm_SUITE.erl:124:1-127:1: This code has
as been cloned 15 times:
/Users/simonthompson/Desktop/StockholmAug09/code/smm_SUITE.erl:124:1-127:1:
/Users/simonthompson/Desktop/StockholmAug09/code/smm_SUITE.erl:124:1-127:1:
/Users/simonthompson/Desktop/StockholmAug09/code/smm_SUITE.erl:124:1-127:1:
/Users/simonthompson/Desktop/StockholmAug09/code/smm_SUITE.erl:124:1-127:1:
/Users/simonthompson/Desktop/StockholmAug09/code/smm_SUITE.erl:124:1-127:1:
/Users/simonthompson/Desktop/StockholmAug09/code/smm_SUITE.erl:124:1-127:1:
/Users/simonthompson/Desktop/StockholmAug09/code/smm_SUITE.erl:124:1-127:1:
/Users/simonthompson/Desktop/StockholmAug09/code/smm_SUITE.erl:124:1-127:1:
/Users/simonthompson/Desktop/StockholmAug09/code/smm_SUITE.erl:124:1-127:1:
/Users/simonthompson/Desktop/StockholmAug09/code/smm_SUITE.erl:124:1-127:1:
/Users/simonthompson/Desktop/StockholmAug09/code/smm_SUITE.erl:124:1-127:1:
/Users/simonthompson/Desktop/StockholmAug09/code/smm_SUITE.erl:124:1-127:1:
/Users/simonthompson/Desktop/StockholmAug09/code/smm_SUITE.erl:124:1-127:1:
/Users/simonthompson/Desktop/StockholmAug09/code/smm_SUITE.erl:124:1-127:1:
/Users/simonthompson/Desktop/StockholmAug09/code/smm_SUITE.erl:124:1-127:1:
The cloned expression/function after generalisation:
new_fun() ->
SetResult = ?SMM_IMPORT_FILE_BASIC(?SMM_RULESET_FILE_1, no),
?TRIAL(ok, SetResult),
AmountOfRuleSets = ?SMM_RULESET_FILE_1_COUNT,
?DM_CHECK(AmountOfRuleSets, ?MP_BS, ets, info, [sbRuleSetTable, size]),
?DM_CHECK(AmountOfRuleSets, ?SGC_BS, ets, info, [smRuleSet, size]),
AmountOfRuleSets.
--** *eri-output* 9% (237.0) (Fundamental Compilation)
```



The general pattern

Identify a clone.

Introduce the corresponding generalisation.

Eliminate all the clone instances.

So what's the complication?



What is the complication?

Which clone to choose?

Include all the code?

How to name functions and variables?

When and how to generalise?

'Widows' and 'orphans'



Clone elimination and testing

Copy and paste ... many hands.

Shorter, more comprehensible and better structured code.

Emphatically not “push button” ...

Need domain expert involvement.



Refactoring and testing

- Property extraction through clone detection and FSM inference.
- Refactoring code and tests: frameworks.
- Refactoring tests in a framework.



Property discovery in Wrangler

Find (test) code that is similar ...

... build a common abstraction

... accumulate the instances

... and generalise the instances.

Example:

Test code from Ericsson: different media and codecs.

Generalisation to all medium/codec combinations.



Refactoring and testing

- Clone detection and abstraction of test code
- Property extraction through clone detection and FBM inference
- Refactoring code and tests: frameworks.
- Refactoring tests in a framework.



Testing frameworks

EUnit, Common Test and Quick Check each give a template for writing tests and a platform for performing them.

Want to refactor code and test code in step.

Extend refactorings while observing

- Naming conventions
- Macros
- Callbacks
- Meta-programming
- Coding patterns



Quick Check example

Callbacks, macros and meta-programming.

```
-export( ..., command/1, postcondition/3, ... ,prop/0]).  
command({N}) when N<10 ->  
    frequency([ {3, {call,nat_gen,next,[]}},  
               {1, {call,nat_gen,stop,[]}}]); ...  
postcondition({N},{call,nat_gen,next,_,R}-> R == N; ...  
prop() ->  
    ?FORALL(Commands,commands(?MODULE),  
    begin {_H,_S,Result} = run_commands(?MODULE,Commands),  
    Result == ok end).
```



Quick Check example

Callbacks, macros and meta-programming.

```
-export( ..., command/1, postcondition/3, ... ,prop/0]).  
command({N}) when N<10 ->  
    frequency([ {3, {call,nat_gen,next,[]}},  
               {1, {call,nat_gen,stop,[]}}]); ...  
postcondition({N},{call,nat_gen,next,_,R}-> R == N; ...  
prop() ->  
    ?FORALL(Commands,commands(?MODULE),  
    begin {_H,_S,Result} = run_commands(?MODULE,Commands),  
    Result == ok end).
```



Refactoring and testing

- State reduction and abstraction in test code
- Property testing through Abstraction and FSM Internals
- Refactoring code and tests: frameworks.
- Refactoring tests in a framework.



Refactoring within QuickCheck

FSM-based testing:
transform state
variable from simple
value to record.

Stylised usage
supports robust
transformation.

Spinoff to OTP libs.

Property refactorings:

Introduce local
definitions (LET)

Merge local defini-
tions and quantifiers
(FORALL).

[EUnit too ...]



www.cs.kent.ac.uk/projects/wrangler/
→ GettingStarted



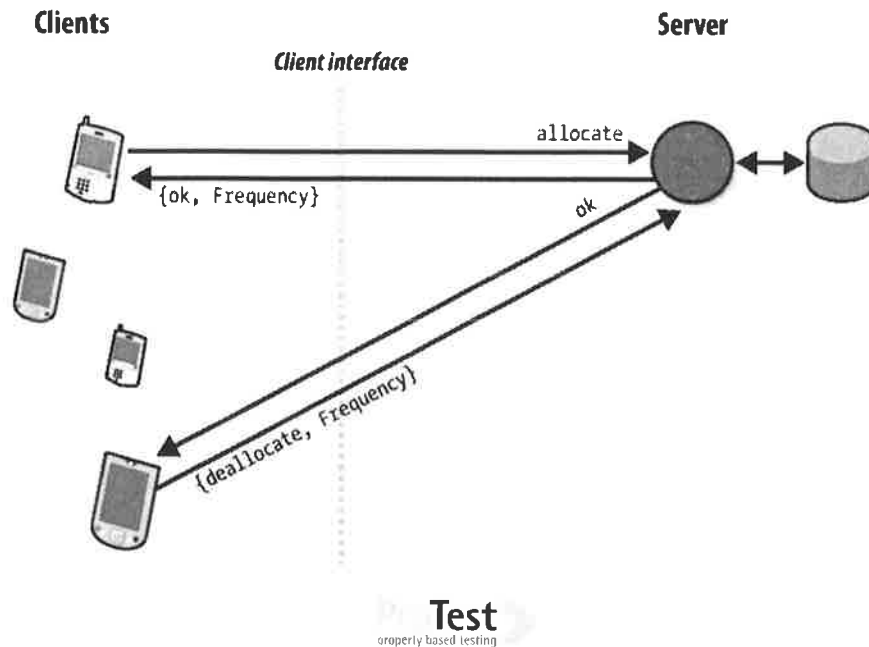
Inferring QuickCheck state machines from Eunit test sets

Thomas Arts, Simon Thompson

Chalmers University, University of Kent



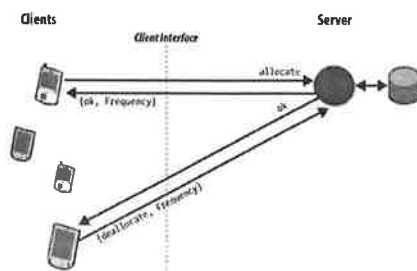
Server for mobile frequencies



Test
property based testing

Server for mobile frequencies

State-based system allows allocation and de-allocation of frequencies from an initial list, once system is started.



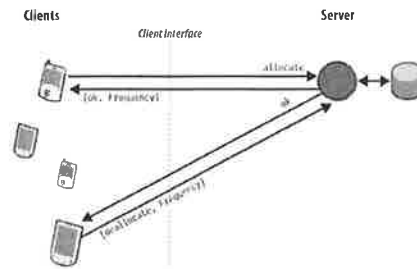
- spec start([integer()]) -> pid().
- spec stop() -> ok.
- spec allocate() -> {ok, integer()} | {error, no_frequency}.
- spec deallocate(integer()) -> ok.

Test
property based testing

Testing start/stop behaviour

EUnit is a unit testing framework for Erlang.

Test start / stop behaviour.



```

startstop_test() ->
    ?assertMatch( ... ,start([])),
    ?assertMatch(ok,stop()),
    ?assertMatch( ... ,start([1])),
    ?assertMatch(ok,stop()).
    
```



Final test set

```

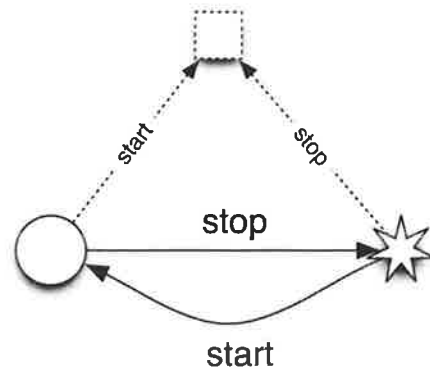
startstop_test() ->
    ?assertMatch( ... ,start([])),
    ?assertMatch(ok,stop()),
    ?assertMatch( ... ,start([1])),
    ?assertMatch(ok,stop()).
    
```

```

stop_without_start_test() ->
    ?assertException( _ , _ , stop() ).
    
```

```

start_twice_test_() ->
    {setup,
     fun() -> start([]) end,
     fun(_ ) -> stop() end,
     fun() -> ?assertException( _ , _ , start([])) end}.
    
```



Improved testing through inductive machine inference

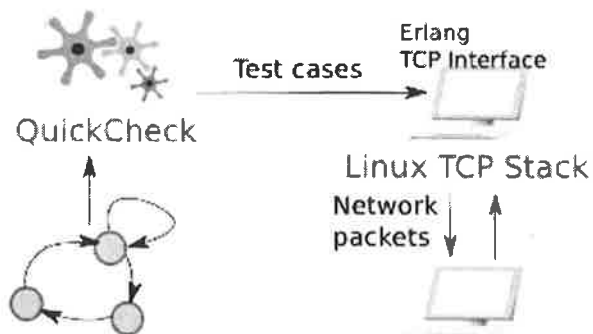
Neil Walkinshaw, John Derrick

University of Sheffield

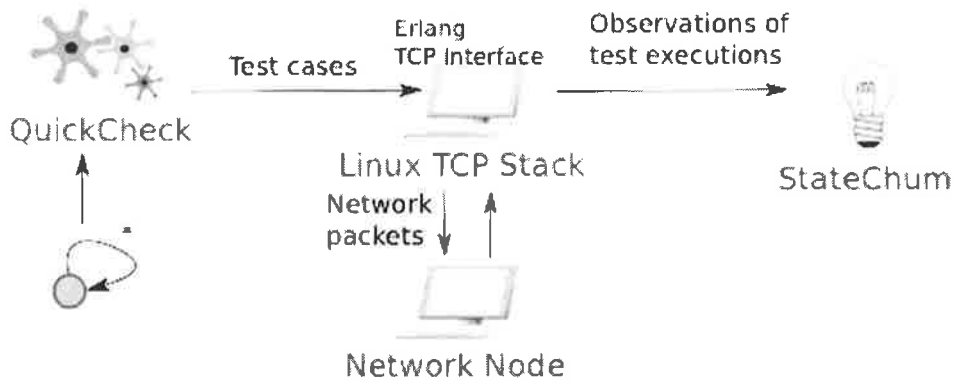


FSM-based testing

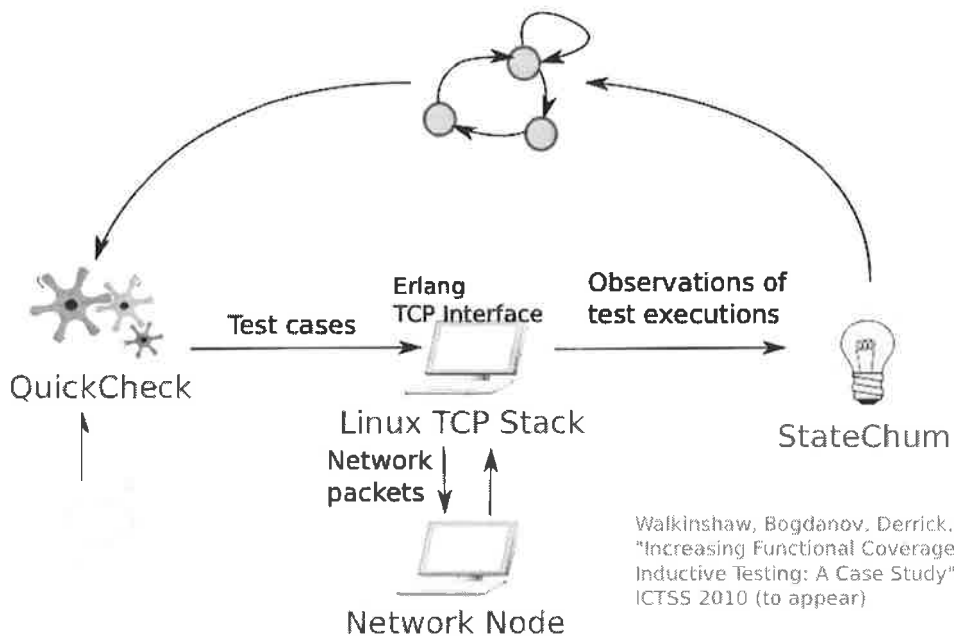
J. Paris and T. Arts,
Automatically Testing TCP/IP Implementations using QuickCheck,
8th ACM SIGPLAN Workshop on Erlang, 2009



Observe test executions



... and improve the FSM



Walkinshaw, Bogdanov, Derrick, Paris -
"Increasing Functional Coverage by
Inductive Testing: A Case Study"
ICTSS 2010 (to appear)

QuickCheck and McErlang integration

Clara Benac Earle, Lars-Åke Fredlund,
Hans Svensson

UPM, Chalmers



McErlang

McErlang is useful for checking concurrent software, not for checking sequential software.

The Erlang runtime system for processes & communication is replaced with a new runtime system written in Erlang (for example, send, spawn... have been re-implemented).

A concurrent program is checked under all possible schedulings.

McErlang is open source, available under a BSD license.



How do the state space traversal methods relate?

Execution path

State space

Faulty part of state space



ProTest
property based testing



QuickCheck



QuickCheck + PULSE

ProTest
property based testing

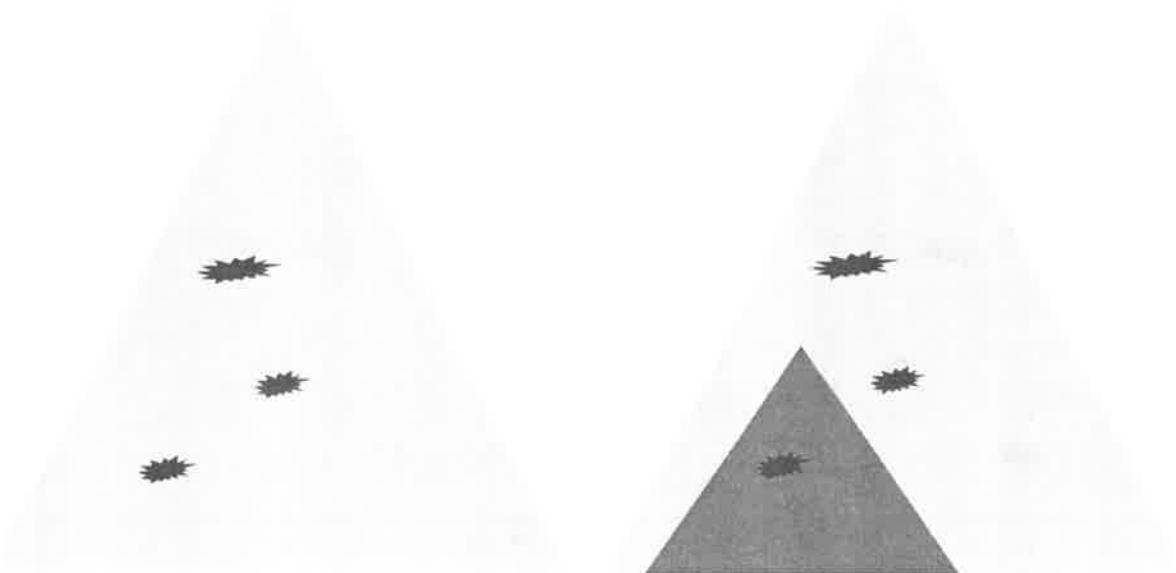
Repeat test N times – ?ALWAYS macro



QuickCheck

QuickCheck + PULSE

Test
property based testing



QuickCheck + McErlang
optimal case

QuickCheck + McErlang
more common case

Test
property based testing

Which verification method to use?

- How large is the state space?
- What is the density of faults?
- How critical is the application?
- What resources (memory/time) do we have?
- Is it better to generate many test cases?
... or to run the same test case many times?
... or explore more of its state space?
- We want to do more experiments and compare!

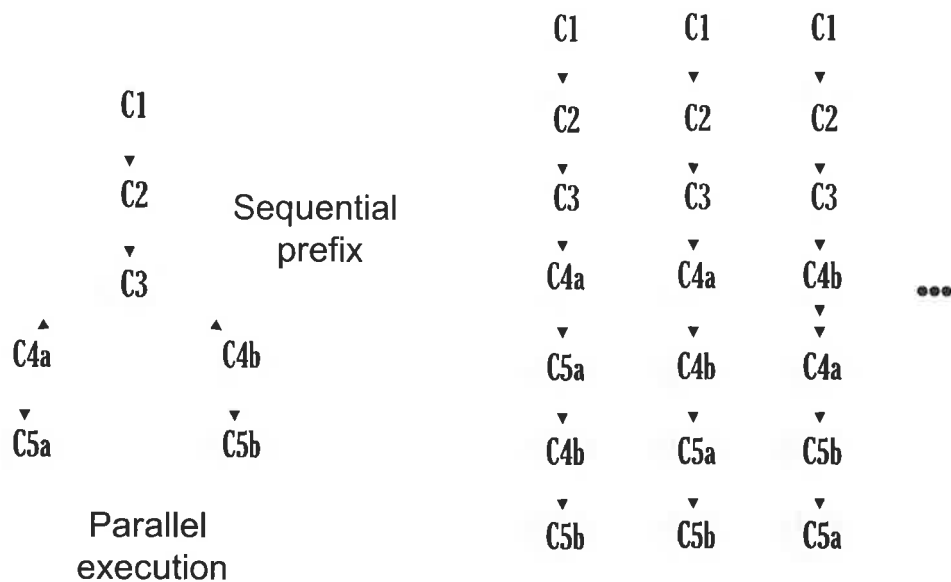


QuickCheck and McErlang integration

- The goal is to provide easy access to the power of model checking to QuickCheck users
- And to make McErlang more accessible through QuickCheck (generators, commands)
- We focus on the QuickCheck state machine library `eqc_statem`
- The `parallel_commands` is a suitable first functionality to integrate



Parallel commands



Is there a linear execution “equivalent” to the parallel one?
(such that all command results are the same)

Implementation - basic QuickCheck

```
prop_testsomething() →
  ?FORALL(PCmds, parallel_commands(?MODULE),
    begin
      {H,S,Res} =
        run_parallel_commands(PCmds),
      ?WHENFAIL(io:format(...),
        Res == ok)
    end).
```

Implementation - PULSE

```
prop_testsomething() →  
  ?FORALL(PCmds, parallel_commands(?MODULE),  
    ?PULSE(  
      [<instrumented-modules>], %Optional?  
      {H,S,Res},  
      begin  
        run_parallel_commands(PCmds)  
      end,  
      ?WHENFAIL(io:format(...),  
        Res == ok))).
```



Implementation - McErlang

```
prop_testsomething() →  
  ?FORALL(PCmds, parallel_commands(?MODULE),  
    ?MCERLANG(  
      [<instrumented-modules>], %Optional?  
      {H,S,Res},  
      begin  
        run_parallel_commands(PCmds)  
      end,  
      ?WHENFAIL(io:format(...),  
        Res == ok))).
```



Conclusions

- Next release of QuickCheck will ship with McErlang integrated
- Benefits to QuickCheck: finding more bugs
- Benefits to McErlang: more users

<https://babel.ls.fi.upm.es/trac/McErlang/wiki/QuickCheck/McErlang>



Samuel Rivas
Erlang Hacker at LambdaStream



Testing what should work, not what
should fail

Abstract

When evolving software we may introduce new corner cases that can pass unnoticed through the test cases in our automated test suite. Since developers rely on other's test suites when changing their code, the more developers work on a module, the more likely it is they slip a bug through the test suite; that's something we thought we had to live with. However, property based testing is emerging as a new hope, our last experiences say that it helps to create test suites more robust to software evolution without forcing us to write tons of "just-in-case test cases."

Biography

Samuel Rivas has been working with Erlang since his last years as a student in the University of A Coruña (Spain). After graduating, he started to work with the MADS (Models And Applications of Distributed Systems) research group, joining to LambdaStream shortly afterwards where he is still working today. In LambdaStream, he has been working in projects about video coding and streaming where he usually has to mix Erlang code with low level C programmes.

Currently, Samuel is leading leading R&D projects, is involved in the quality improvement group, and is the main architect of a number of LambdaStream solutions.

Thomas Arts
Professor and co-founder of QuviQ AB



Testing automotive software with Erlang

Abstract

Modern cars are filled with software. The software for cars is normally written in C and the specifications are at least as bulky as those known from the telecommunication industry. Different from Erlang software, it has some hard real-time requirements, where tasks definitely need to meet certain deadlines. Nevertheless, it is beneficial to test this software by using Erlang as a test framework.

We report on a project of testing an AUTOSAR component by using a combination of Erlang and QuickCheck. We tested the network management module of the Can bus. This network management module is specified as a state machine plus a lot of other details. The most challenging part is dealing with the timers that cause spontaneous transitions from one state to another when time progresses.

Biography

Prof Thomas Arts is the co-founder and CTO of Quviq, a small company that produced QuickCheck, as testing tool for Erlang. Thomas has over 30 publications in various journals and has experience refereeing conferences and workshops. He has successfully introduced some new technologies to the industry, the latest being QuickCheck, a tool for property based testing and aims to support test driven development. Thomas is also a professor at Chalmers University of Technology in Gothenburg, Sweden.

Thomas was one of the members of Ericsson's computer science lab where he worked on program verification and the development of the Erlang programming language. He has also worked in the broad spectrum theoretical computer science, formal methods and industrial case-study research, mainly applying all kind of techniques to systems written in Erlang.

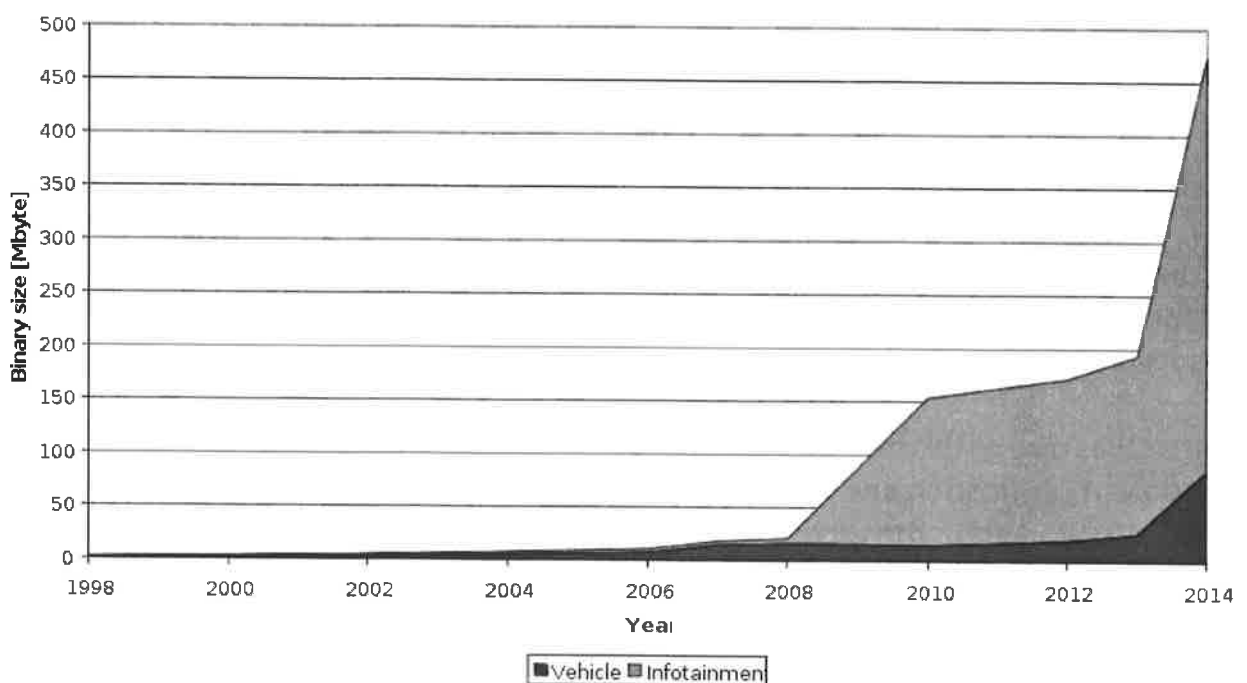
Testing Automotive Software with Erlang

Thomas Arts
Chalmers / Quviq AB

in collaboration with
Juan Puig, Anders Kallerdahl and Ulf Norell
Erlang Solutions Mentor Graphics Quviq

Software in modern cars

S/W size in new car models



source: Ulrik Eklund

Many components that need to communicate with each other

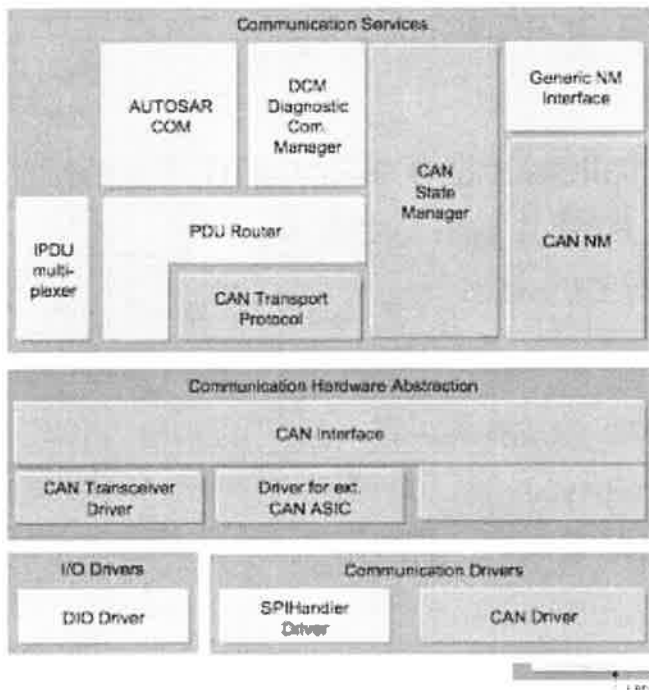
More diversity, faster time to market, higher complexity....

We have seen this before ☺

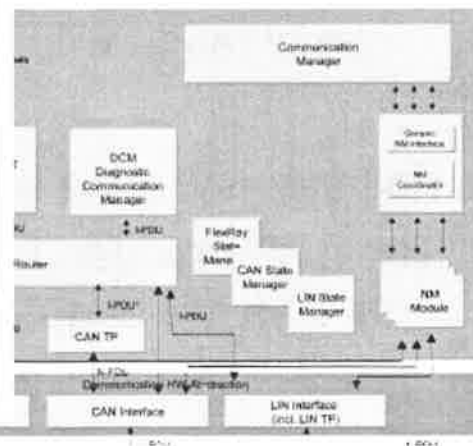
Solutions:

- Standardization of components
- Standard platform (operating system)

AUTOSAR a consortium standard



source: www.autosar.org



AUTOSAR specification open for interpretation.

Even if a component follows the standard, there is no guarantee at all that it will work in combination with other standard components

nothing new... we have seen that before ☺

The evil is hidden in configurations: each Node in the car has typically its own set of options, and software supplier

AutoSAR specification open for interpretation.

Even if a component follows the standard, there is no guarantee at all that it will work in combination with other stan

nothing new... we have seen t

The evil is hidden in configura

its own set of options that is used

Solution:
Spend your budget on testing instead of development

Software systems more complex every day...

- ... more components
- ... more possible configurations per component
- ... more component interactions

Traditional testing insufficient to keep up with this

We need to change our testing methods!

Using Erlang to test C software

- High level language: easier to write test code
- Good tools to support testing

but... we need to connect to C code



All information you need to write marshalling code is in the C (header) files.

Thus, write a C parser in Erlang, extract all type information and generate the link between C and Erlang.

Example



Suppose we have C file `example.c`

```
// Sum an array of integers
int sum (int *array, int len) {
    int n;
    int sum = 0;
    for (n = 0; n < len; n++)
        sum += array[n];
    return sum;
}
```


Example



Erlang shell used to communicate with C

```
1> eqc_c:start(example).  
ok
```

```
2> P = eqc_c:create_array(int, [1, 3, 3, 8]).  
{ptr,int,1048864}
```

```
3> example:sum(P, 4).  
15
```

```
4> eqc_c:free(P).  
ok
```

Example



Erlang shell used to communicate with C

```
1> eqc_c:start(example).  
ok
```

```
2> P = eqc_c:create_array(int,  
{ptr,int,1048864}
```

```
3> example:sum(P, 4).  
15
```

```
4> eqc_c:free(P).  
ok
```

- parse example.c
- create a c program that listens to a socket
- create example.beam and example.hrl with all functions from example.C
- start C program in a separate thread

Example



Erlang shell used to communicate with C

```
1> eqc_c:start(example).  
ok
```

```
2> P = eqc_c:create_array(int, [1, 3, 3, 8]).  
{ptr,int,1048864}
```

```
3> example:sum(P, 4).  
15
```

```
4> eqc_c:free(P).  
ok
```

an array is created in the C thread and the pointer returned points to memory in that thread

QuickCheck



From test case to property

Instead of specifying one or two test cases to demonstrate that the software fulfills a certain property, we specify *the property* and have the tests automatically generated!

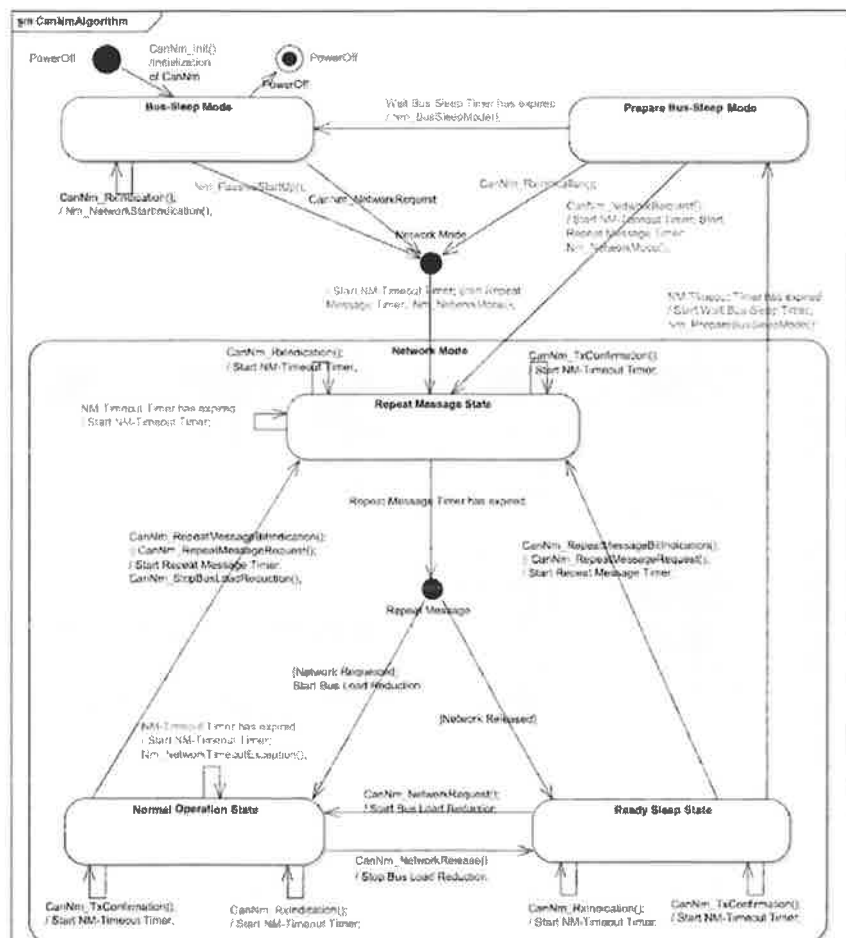
Model based testing with *controlled random generation of test cases*

How difficult is it to test real-time C code?

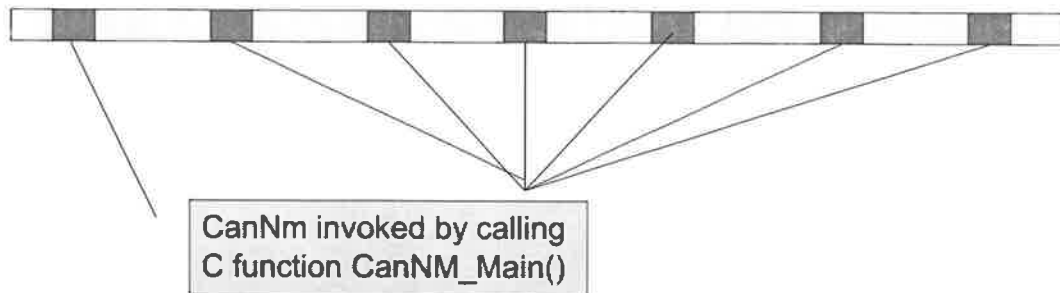
Mentor Graphics hosts master student thesis project to test CanNM with QuickCheck using this C link.

AUTOSAR component as UML state machine

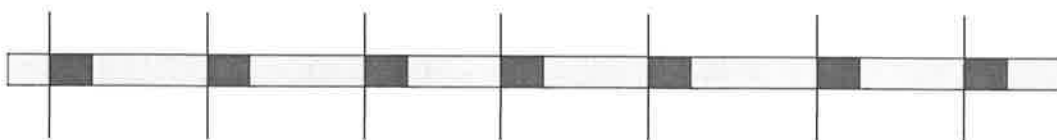
CAN Network Management



CanNM is scheduled as one of many tasks



CanNM is scheduled as one of many tasks

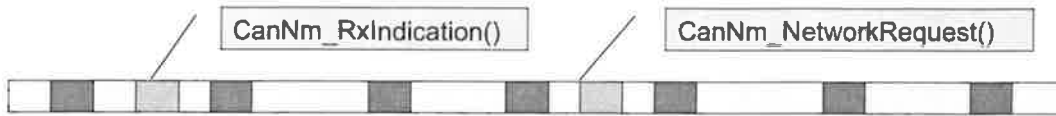


Assumption:

One time unit elapses before CanNm_Main() is called

(In fact, C implementation handles the timers, not the scheduler)

CanNM is scheduled as one of many tasks



Other tasks communicate by calling CanNM interface functions

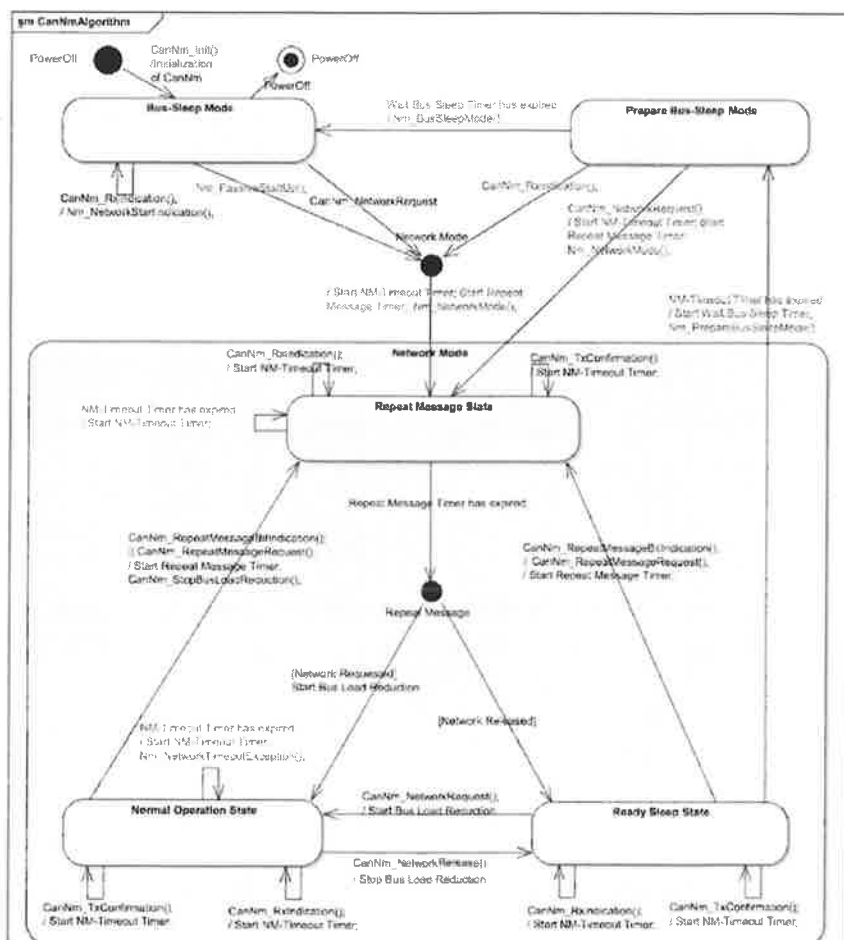
These update data structures in memory

Assumption: Only one interaction in each slot

AUTOSAR component as UML state machine

CAN Network Management

Now... make a QuickCheck model from this state machine





State transitions as Erlang data structure

```

bus_sleep_mode(_) ->
  [ {power_off, {call, ?MODULE, powerOff, []}},
    {bus_sleep_mode, {call, ?MODULE, main, []}},
    {bus_sleep_mode, {call, ?MODULE, 'CanNm_RxIndication', [id(), u8()]}},
    {repeat_message_state, {call, ?MODULE, 'Nm_PassiveStartUp', []}},
    {repeat_message_state, {call, ?MODULE, 'CanNm_NetworkRequest', []}}].

repeat_message_state(_) ->
  [ {normal_operation_state, {call, ?MODULE, main, []}},
    {ready_sleep_state, {call, ?MODULE, main, []}},
    {repeat_message_state, {call, ?MODULE, main, []}},
    {repeat_message_state, {call, ?MODULE, 'CanNm_RxIndication', [id(), u8()]}},
    {repeat_message_state, {call, ?MODULE, 'CanNm_TxConfirmation', [id()]} }].

```

Model how additional state data changes:
timers, network status, ...

```

next_state_data(repeat_message_state, repeat_message_state, S, _V, {_, _, main, _}) ->
  S#can_nm{repeatMessageTimer = S#can_nm.repeatMessageTimer-1,
    nmTimeoutTimer =
      case S#can_nm.nmTimeoutTimer of
        0 -> ?NMTIMEOUT;
        N -> N-1
      end};

next_state_data(repeat_message_state, repeat_message_state, S, _V, {_, _, _, _}) ->
  S#can_nm{repeatMessageTimer = S#can_nm.repeatMessageTimer-1,
    nmTimeoutTimer = ?NMTIMEOUT};

```

Testing hard real-time C



How difficult is it to test real-time C code?

Master student thesis project to test CanNM with QuickCheck using this C link.

Result: - we know how to do it
- it is not that much work
- we found ambiguities in the specification

Testing real-time C



CanNm was modeled using a state machine.

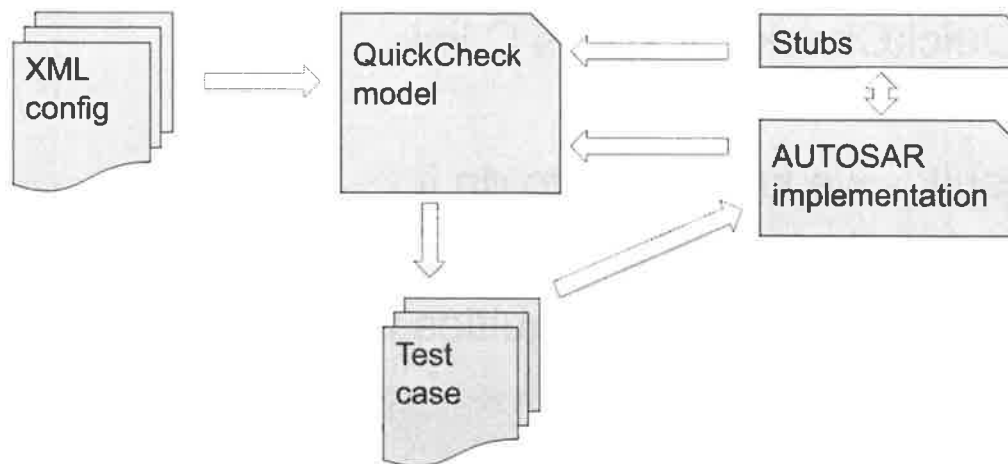
Not all AUTOSAR components are specified as state machines... can we do the rest as well?

Sep/Oct 2010: Experiment (with Mentor Graphics)

- Test COM/PDUR with QuickCheck
- In parallel manual testing of same software (estimated 20 weeks)

approx 8000 lines of C code, representative component

- We have built a model for testing COM and PduRouter



QuickCheck for automotive

We created a model

The model is configurable with an XML config file

Marshalling code is automatically generated from header files

C stub is only a 400 lines of code

QuickCheck model is 800 lines of code

Total: 2 person weeks work

Testing Automotive software

Q...

Conclusions:

We gain productivity

- Erlang less lines of code
- QuickCheck model instead of test cases

We have a scalable solution for AUTOSAR

In the future...

buy a car that has been tested with Erlang!

Mission Critical with Erlang And QuickCheck: Quality Never Sleeps

Abstract

How do you integrate a new component into a mission-critical, legacy system with the key requirement that existing functionality must not be affected? We used QuickCheck to specify the behaviour for the existing system and then we attached our component to validate its behaviour.

In this talk we highlight how property based testing using QuickCheck was beneficial and fun for our project where we lacked full documentation and understanding of the legacy behaviour. Using QuickCheck we were able to produce and validate that our component had the high quality requested as well as allow us to focus our testing into specific areas with very little effort.

Raghav Karol, software engineer at Motorola since 2006.
Worked with network management before becoming an Erlanger in 2009.



Torben Hoffmann, Erlang Priest at Motorola
since 2006.



Mission Critical with Erlang And QuickCheck *Quality Never Sleeps*



Raghav Karol
Motorola Solutions



Torben Hoffmann
Motorola Solutions

Setting the stage

Overview

What did we have to do?

What did QuickCheck help us with?

How is it to use Erlang?

How productive is Erlang/OTP?

What did we have to do?

- ISI Project
 - Gateway to interconnect two TETRA systems
 - Migration for TETRA
 - State-full protocol conversion
 - Motorola Proprietary Call Control and Mobility
 - ISI – ITSI InterSystem Interface open standard
 - QSig, HDLC, LAPD, E1
 - Mobility and Resource Management
 - High concurrency requirement
 - High reliability – required to connect to live customer system

Motivation and Background

Working prototype to be delivered as a product

- Existing codebase created for IOP certification

Connect to live system

- Main requirement – Should **not** crash existing system
- Specification of legacy protocols not complete

Small team, 5 members, no dedicated test resources

ISI application Erlang + C

- Enter Property based testing and QuickCheck

5

Romania

Don't KILL the
existing SYSTEM
in Romania.

And he's just the requirements guy ☺!



September 2010

ISI Stack

ISI

FLM	RTP	IZ	QSIG	HDLC	QSIG
		NETCOM	LAPD		LAPD
TCP	UDP	UDP	E1	E1	E1

Unit test versus Property based testing

- Testing using xUnit like tools
 - Setup Fixture, execute test case, Teardown Fixture
 - Works well --- Used with good success before using property based testing
 - Important to have good test cases
 - Test cases and scenarios easily overlooked
 - Maintenance and refactoring of test cases also always required
 - How is property based testing different
 - *Specify* rules of generating a test case
 - *Specify* pre-conditions when above rules can be valid
 - *Model* expectations once a rule executes
 - Check post-conditions once a rule executes
 - What!?! Can this even work?

Quickcheck

QuickCheck

- Generators
- Properties
- Stateless testing
 - Symbolic test cases
- State full testing
 - State machines based
- Shrinking
- Also available for C --- Not used in our project.

9

Testing a resource manager

- The `rm_ds` is a data structure that manages a range of integers representing unique resources
- API:
 - `lock_next(RM) -> RM`
 - Get the next available resource based on the policy for finding the next available
 - `release(N, RM) -> RM`
 - Release the resource id'ed by N.
 - `lock(N, RM) -> RM`
 - Locks a specific resource (the one id'ed by N)

10

Creating a random resource manager

```
rm[Size) ->
?LET({From, Length, Policy},
      {int(), nat(), oneof([up, down, high, low])}),
?LAZY(
  frequency([[1, rm(0, From, From+Length, Policy)],
             {2, {call, ?MODULE, lock_next, [rm(Size-1)]}},
             {2, {call, ?MODULE, lock, [nat(), rm(Size-1)]}},
             {2, {call, ?MODULE, release, [nat(), rm(Size-1)]}}]
            ]
          )))
```

You create an RM by using a random sequence of commands from its API

11

Example: random resource manager

```
lock_next
lock      17
release   13
lock_next
lock_next
release    4
new       [-11, 6, up]
-----
Lower Range      = -11
Upper Range      = 6
Allocation Policy = up
```

Example property for the resource manager

Locking a specific resource:

```
prop_lock() =>
  ?FORALL(SymRM, N), {create_rm(), int()},
  begin
    RMO = (eval(SymRM)),
    #data{range = Range} = RMO,
    case rm_ds:lock(N, RMO) of
      {invalid_resource, RMO} ->
        not in_range(N, Range);
      {already_used, RMO} ->
        is_locked(N, RMO);
      {N, RM1} ->
        is_locked(N, RM1)
    end
  end).
```

13

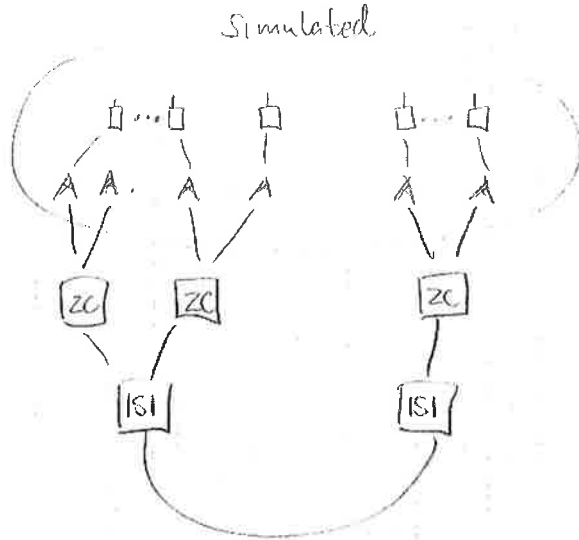
Property based testing in ISI

- Unit Level
 - Queue data structure
 - Resource management and allocation server
- Component level
 - NetComm Layer
- Black box – Box test level

Black box testing of the ISI GW

Simulated sites and mobiles

- Implementation of the ZC-Site protocol
- Pseudo Air Interface protocol



15

Black box test example

```
do_register      [3000001, {7, 2}]]}
do_migrate      [3000001, {5, 3}, {7, 2}]]}
do_register      [3000004, {7, 5}]]}
do_call         [3000004, 3000001, [{simplex_duplex, simplex}]]}
do_answer_call  [3000001, 3000004, {5, 3}]]}
do_dekey        [3000001, 3000004, {5, 3}]]}
```

- The test cases are expressed in terms of actions taken by the mobiles

16

Handling Concurrency in QC

In many cases the success of an operation required several messages to occur

Enter...

- hooks

A hook spawns a listener processes for each message that is expected

Hook Example

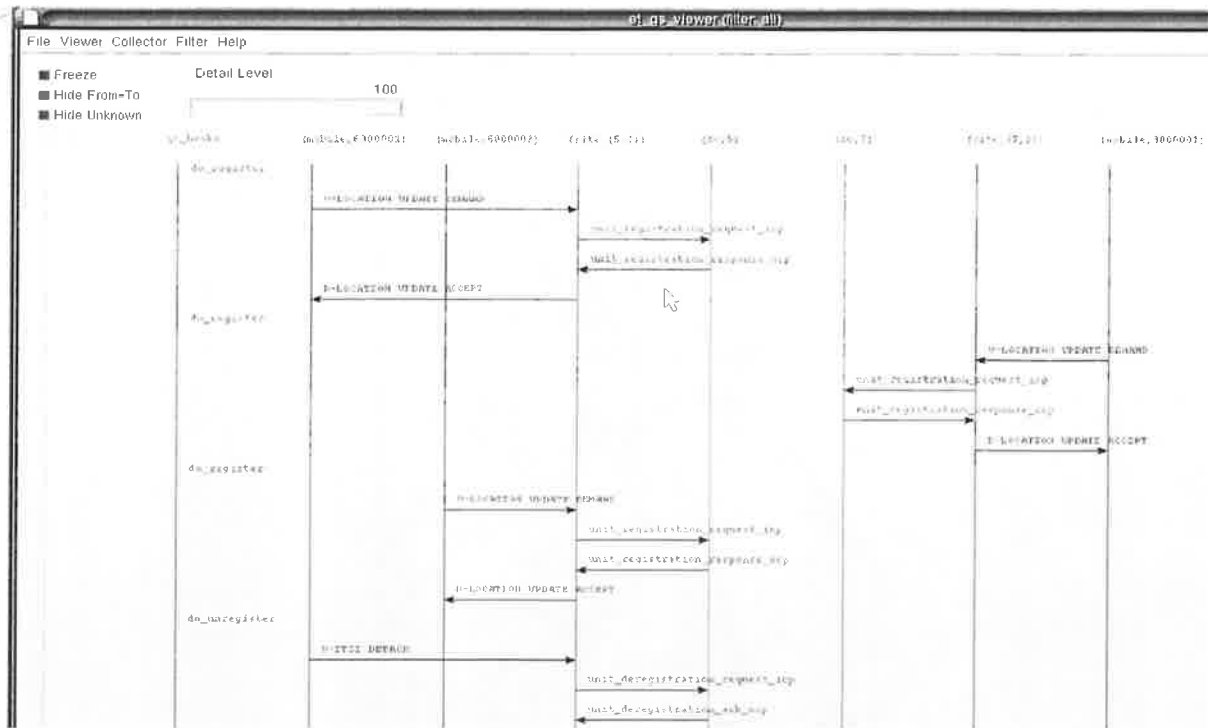
```
%% M1 is answering the call from M2.
do_answer_call(M1,M2,S1) ->
  ?TRACE(do_answer_call,[M1,M2,S1]),
  Pid1 = spawn(?MODULE,listen_for_d_msg,[M1,
                                     msg_type('D-CONNECT ACKNOWLEDGE'),
                                     ?TIMEOUT,self()]),
  Pid2 = spawn(?MODULE,listen_for_d_msg,[M2,
                                     msg_type('D-CONNECT'),
                                     ?TIMEOUT,self()]),
  Pid3 = spawn(?MODULE,listen_for_site_msg,
               [S1,
                msg_type(subscriber_tx_detected_icp),
                ?TIMEOUT,self()]),
  mobile:answer_call(M1),
  Results = collect_results([Pid1,Pid2,Pid3]),
  pause(),
  Res=check_results(Results),
  ?LOG({do_answer_call,[M1,M2,S1]},Res),
  Res.
```

Listen until
stated
message
received

The Operation

Gather results
from listeners

Was everything
as expected?



19

QuickCheck Quality Never Sleeps

- So you made a testing framework – I can do that!
- Yes – every software department in our company, including us has
- Why QuickCheck
 - More thinking, specify, specify, specify, less work
 - Randomness

Visualizing auto tests



Visualizing auto tests



Erlang/OTP Quality never sleeps

Auto-enforcement of a coding standard with Erlang/OTP

Semantics of framework uncomfortably simple

Ease of Distribution

Supervisors - error handling and reliability

Common case ALWAYS works

Understanding software behavior

Resolving Issues

- Resolving issues in the field
- Integration issues with other vendor forces: panic, stress, our reputation

Evaluation of performance

- The eternal problem:
 - Which approach is better?
- Key problem:
 - What is the size of the problem?
- One of the best size measures for software:
 - Function Points
 - Measures input and output and treats the software as a black box
 - Not widely used since it is time consuming to generate FP estimates for a system and even harder to check how many the final system has

Backfiring

Backfiring:

- Counting Function Points by looking at the actual code

Our approach:

- Use epp_dodger to extract incoming messages
- Use xref to extract outgoing messages
- Post-process in Excel to ensure counting the correct messages

Comparing with others

- Function Points are often used by estimation tools
 - Construx Estimate
 - COCOMO II
- Basic project estimation:
 - Inputs:
 - Function Points
 - Programming language
 - Type of project (Telecommunications)
 - Output:
 - Staff Months (SM) to complete the project

Erlang vs X

Language	Vs Erlang effort
Java	3x (2.3-3.9)
C++	4x (3.4-5.3)
C	7x (5.9-9.3)

So for a telecommunications project Erlang/OTP seems to be the right choice...

Conclusions

- Challenged with increased complexity
 - Concurrency, distribution, reliability
- Delivery of features
- Lowering Costs, Development O&M,....
- Losing track of what the customer really wants
- Erlang solves, technical, communication and cost reduction problems
- QuickCheck
 - Leverages Erlang language features
 - Future of testing

Tino Breddin
Embedded Erlanger from Dresden



Continuous integration for Erlang/OTP,
enhancing its quality and ease-of-use.

Abstract

Swarm is a continuous build server implemented in Erlang which presents a innovative twist to software quality assurance. The two core concepts, multi-platform build execution and automatic package generation, give projects the ability to provide users with exceptional support while requiring little manual effort. This talk describes how Swarm is used to enhance Erlang/OTP's development process. Further, the upcoming pre-build and tested packages of Erlang/OTP binaries for many free and commercial operating systems are discussed.

Biography

Tino Breddin is a Systems Engineer at Erlang Solutions Ltd where he spends quality time on building scalable, highly-reliable systems for messaging and data storage. When not using Erlang he prefers using Python or Ruby for anything which needs to be automated. Previously Tino worked at the research labs of SAP Labs LLC in Palo Alto and SAP AG in Dresden focusing on massively scalable systems development.

Jacoby Thwaites
Inventor, designer and implementer



A prototype state machine "MadCloud"
for distributed applications

Abstract

Distributed applications often use client-server techniques across a TCP/IP infrastructure. This talk presents a prototype infrastructure that abstracts the client out of programmers' code and can optimize inter-server traffic across datacenters. It is written in Erlang with an admin UI that is quite pretty because HTML5+CSS3+XSLT+jQuery rocks actually.

Whatever. The infrastructure is very simple, and works by sequencing calls on application-level servers following some initial event. It does this by collecting sets of fields, sending subsets to servers and collecting replies where appropriate, until no further work can be done. The result is, application backends that are declarative.

My feeble hope is that this opens up pathways including applications constructed by search, mutable applications that change many times a second, large-scale caching of server operations, applications that self-render and other things I've not thought of.

Biography

Since 1990 Jacoby Thwaites ran his own company in the UK architecting large-scale networks and enterprise messaging systems. Around 2000 the company got into writing and selling insurance systems in the London Market. He then co-founded a Silicon Valley Web 2.0 startup which folded a couple of years later, and most recently joined Google in 2008 to set up the London AdSense engineering team. He has spent the last 6 months inventing, designing and implementing this state machine as his Google 20% project.

Rusty Klophaus
Senior Engineer at Basho Technologies



Masterless Distributed Computing
with Riak Core

Abstract

In this talk, Rusty will explain why Riak Core was built, discuss what problems it solves and how it works, and demonstrate how a developer can leverage Riak Core to quickly build their own masterless distributed application.

Biography

Rusty Klophaus is one of the core engineers at Basho Technologies where he focuses on building distributed, fault-tolerant applications to store and retrieve Big Data. He is currently on the core engineering team of Riak Search. Before joining Basho, Rusty launched an Erlang-based startup (which spawned the Nitrogen Web Framework); before that he managed multi-million dollar technology products and guided multiple project teams at an Internet advertising company based in New York City. When he's not hacking, Rusty plays guitar and organizes the Hacker News Meetup Group of Washington, DC.

Rickard Green
OTP Team Member and VM Committer



A deep dive into some aspects of the
multicore support in the Erlang VM

Abstract

The first Erlang virtual machine capable of utilizing multicore and multiprocessor hardware, i.e. the Erlang VM with SMP support, first appeared in 2006. Since then work has been ongoing on improving performance and scalability. The increasing number of cores on common processors makes scalability issues very important. The scalability of the first VM with SMP support was quite modest, but has since then improved immensely and will continue to be improved in the future. During this talk we'll look closer at some scalability issues and how they have been addressed.

Biography

Rickard Green is a senior specialist in multicore/multiprocessor utilization and scalability at Ericsson AB. He works with the development of Erlang's virtual machine at Erlang/OTP and has done so for the past ten years. The development of an Erlang VM capable of utilizing multicore/multiprocessor hardware, i.e. the SMP support of the Erlang VM, begun about five years ago. He has since then been the main developer of the SMP support, and has designed and implemented major parts of it.



RWLocks in Erlang/OTP-R14B

Rickard Green
rickard@erlang.org



What Made us Look at RWLocks?

- › Testcase failed
 - Pthread rwlocks on Linux with reader preferred strategy caused starvation of writers
 - Solved by using our writer preferred fallback implementation instead
- Customers complained about poor performance of the fallback implementation
 - Solved by letting them enable the reader preferred pthread rwlocks implementation
- › That is, something needed to be done... ☹️

What is an RWLock?

- › Read/Write Lock
 - Write locked
 - Exclusive access for one thread
 - Read locked
 - Multiple reader threads
 - No writer threads

NPTL Pthread RWLocks

- › Why look at NPTL RWLocks?
 - NPTL (Native POSIX Thread Library) is the thread library used on modern Linux distros
 - Linux is our most important platform
 - The vast majority of our customers run on Linux
 - A lot of the other (perhaps most of the other?) users run on Linux
- › Strategy used during contention
 - Defaults to reader preferred
 - As long as the lock is read locked or readers are waiting for the lock, writers have to wait
 - Can be configured as writer preferred
 - As long as the lock is write locked or writers are waiting for the lock, readers have to wait
 - Both strategies suffer from starvation issues
 - Writer preferred is, however, not as problematic as reader preferred

RWLocks Used in ERTS

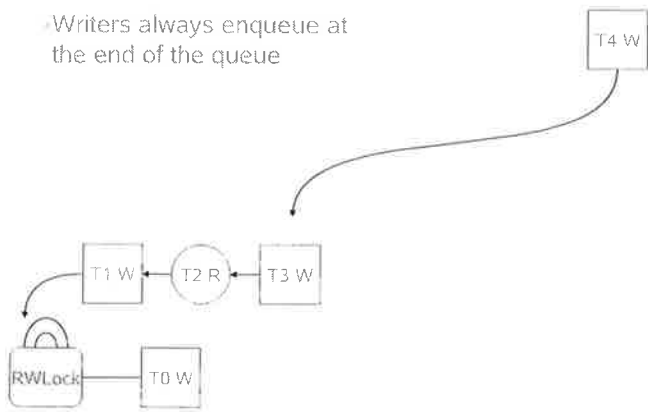
- › ETS tables
- › Internal tables
 - Atom table
 - Registered names
 - Distribution tables

ERTS RWLocks

- › Doesn't use a writer or reader preferred strategy
- › Interleaves readers and writers during contention
- › Fair against readers as well as writers

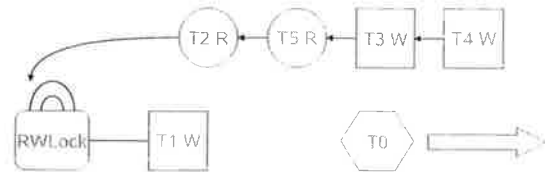
ERTS RWLocks - Enqueue Writer

- Writers always enqueue at the end of the queue



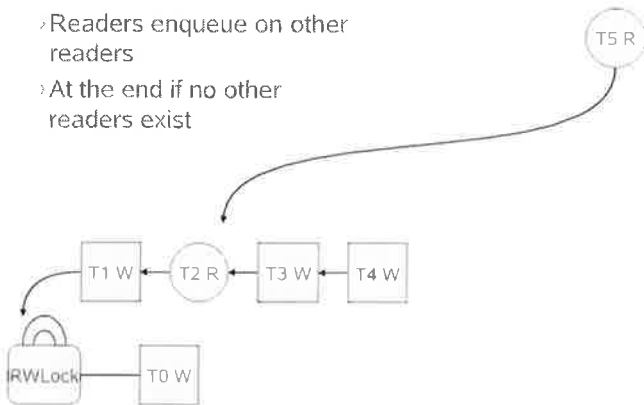
ERTS RWLocks – Dequeue Writer

- Writer at the head of the queue takes over the lock



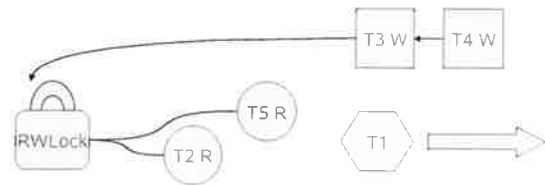
ERTS RWLocks - Enqueue Reader

- Readers enqueue on other readers
- At the end if no other readers exist



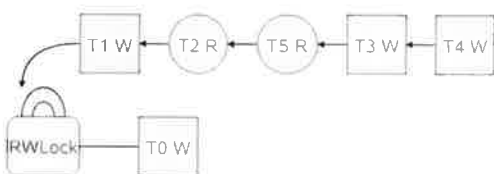
ERTS RWLocks – Dequeue Readers

- All readers at the head of the queue take over the lock



ERTS RWLocks - Queue

- All readers will accumulate at one place in the queue



ERTS RWLocks - Queue

- Ensures that all waiting threads eventually will get the lock
- Able to execute readers as much as possible in parallel
- Writers won't be punished too much by readers going past them
 - Writers won't be punished at all in the case where there are as many threads on the system as cores and each thread read locks for the same amount of time

ERTS RWLocks - Default

Data structure

- Integer flag field
- Queue (double linked list of waiting threads)
- Queue lock

Uncontended case

- Atomic operations on the integer flag field

Contended case

- Atomic operations on the integer flag field
- Locked operations on the queue

ERTS RWLocks – Reader Optimized

Data structure

- Integer flag field (modified)
- Queue (double linked list of waiting threads)
- Queue lock
- Reader groups (integer counters in separate cache-lines)

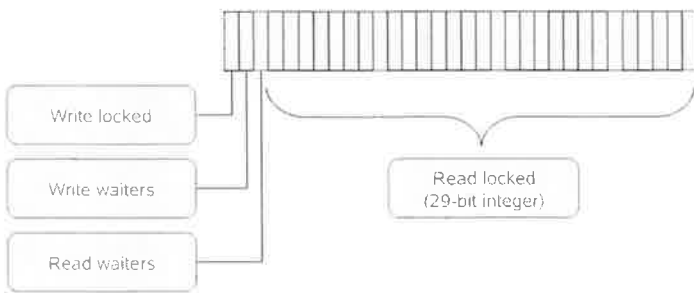
Uncontended cases

- Write lock/unlock
Atomic operations on the integer flag field (not completely true)
- Read lock/unlock
Atomic operations in the readers groups (mostly)

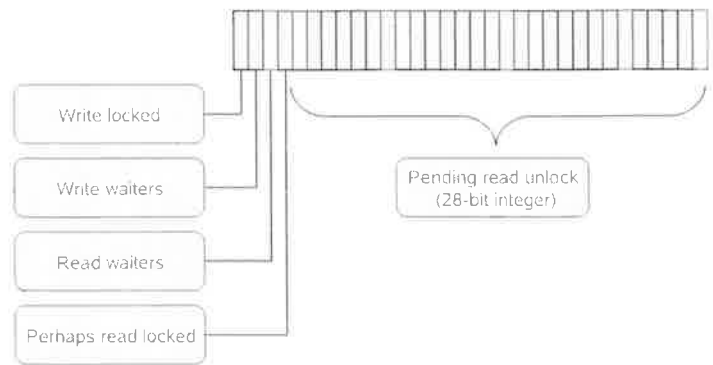
Contended case

- Atomic operations on the integer flag field
- Atomic operations in the reader groups
- Locked operations on the queue

ERTS RWLocks – Flag Field (default)



ERTS RWLocks – Flag Field (reader optimized)



ERTS RWLocks – Performance

- > An improvement compared to NPTL RWLocks, but uncontended read lock case is still a bit disappointing
 - Conceptually only reads of memory, however...
 - Writes to the RWLock cache line is ping-ponged between processors
 - In ETS also other stuff is ping-ponged
 - Meta table lock cache-line
 - Table reference counter cache-line
- > Would be nice to avoid this cache-line ping-ponging
 - Modified reader optimized RWLock implementation using reader groups
 - ETS modifications:
 - Rewrite of the meta table locking to use the new reader optimized rwlocks
 - No use of table reference counter when read locking

ERTS RWLocks – Reader Optimized

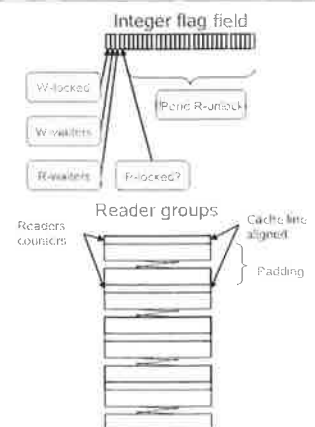
Uncontended read-lock

- Increment reader group counter
- Read flag field.
Verify no "W-locked", "W-waiter", nor "Pend R-unlock"
Set "R-locked?" if not already set

Uncontended read-unlock

- Decrement reader group counter
- If reader group counter reached zero, read flag field
Verify no "{W,R}-waiters" nor "Pend R-unlock"

Note! We do not reset "R-locked?"



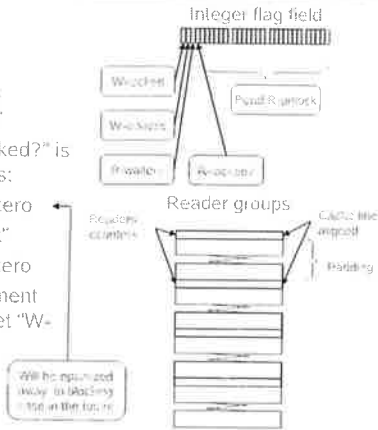
ERTS RWLocks – Reader Optimized

Uncontended write-lock

- Read integer flag field.
- Verify no "W-locked", "{W,R}-waiters", nor "Pend R-unlock"
- Set "W-locked" unless "R-locked?" is set, then check reader groups:
 - Verify that all groups are zero
 - Increment "Pend R-unlock"
 - Verify that all groups are zero
 - Reset "R-locked?", decrement "Pend R-unlock", if zero set "W-locked"

Uncontended write-unlock

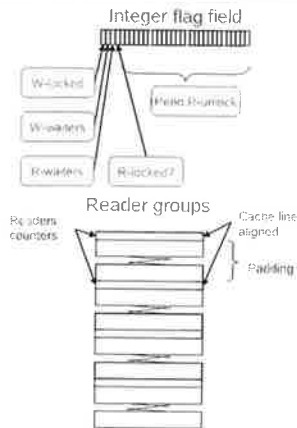
- Reset "W-locked"
- Verify no "{W,R}-waiters"



ERTS RWLocks – Reader Optimized

Contended cases

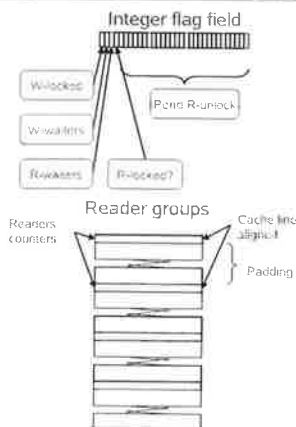
- When a lock operation fail, the thread continues spinning trying to lock actively (a few times); then enqueue and spin passively (on another structure); then block
- When enqueueing the "{W,R}-waiters" flag is set while holding the queue lock; then one last effort to acquire the lock is made
- After this the thread depends on another thread transferring the lock to it and waking it up



ERTS RWLocks – Reader Optimized

Contended cases (continued)

- Write locking when "R-locked?" is set is the complicated case since it can be interrupted by modifications in reader groups
- A thread modifying reader groups aborts or continues its operation if "Pend R-unlock" is set and then tries to complete a read unlock (this operation can also be interrupted; which also...)
- The amount of interrupts is limited when doing blocking operations since threads eventually will end up in the queue
- Non-blocking operations aren't allowed to loop indefinitely



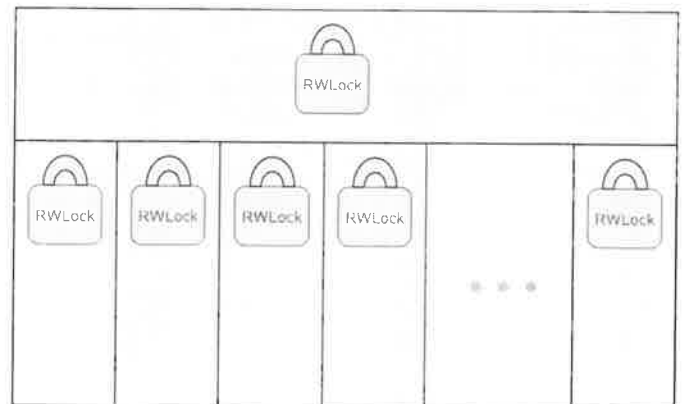
Benchmarking

- The benchmark used can be downloaded from www.erlang.org/~rickard/euc-2010
- 1000 processes accessing a common public ets table of type set
- Accesses consists of `ets:lookup()` (read) and `ets:insert()` (write) in different mixes
- Run with the `thread_spread` scheduler bind type
- R14B NPTL-rwlocks and R14B ERTS-rwlocks only differs in rwlock implementation used
 - R14B NPTL-rwlocks: `./configure force_pthread_rwlock=yes`
- When benchmarking with NPTL-rwlocks
 - No read_concurrency run has been made - same as default
 - No combined read_concurrency and write_concurrency has been made - same as write_concurrency

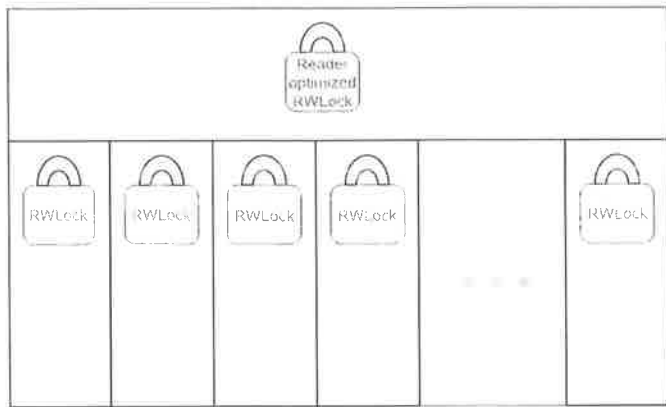
ERTS RWLocks – ETS Options

- The default
 - One table global normal (non-reader optimized) rwlock
- `read_concurrency`
 - One table global reader optimized rwlock
- `write_concurrency`
 - One table global reader optimized rwlock (normally read locked), and multiple normal rwlocks protecting different parts of the table
- `read_concurrency` and `write_concurrency` combined
 - One table global reader optimized rwlock, and multiple reader optimized rwlocks protecting different parts of the table

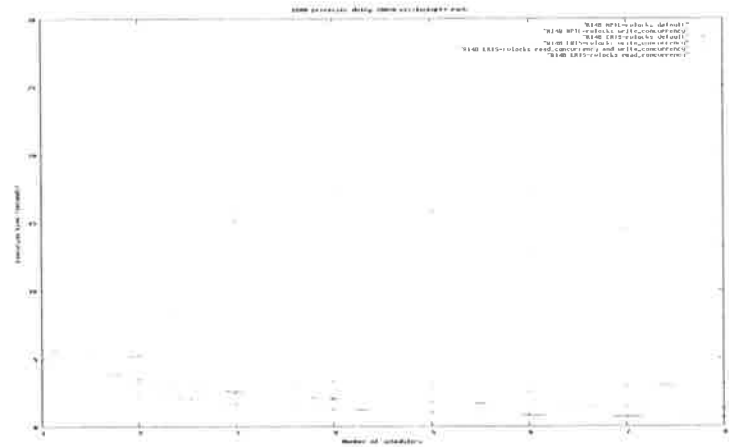
ETS Table with write_concurrency Option (before R14B)



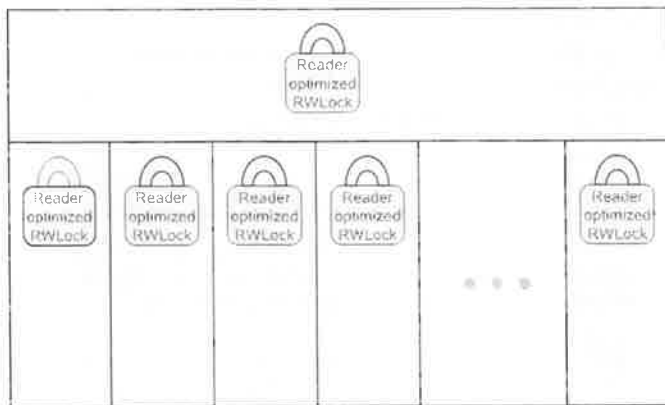
ETS Table with write_concurrency Option (R14B)



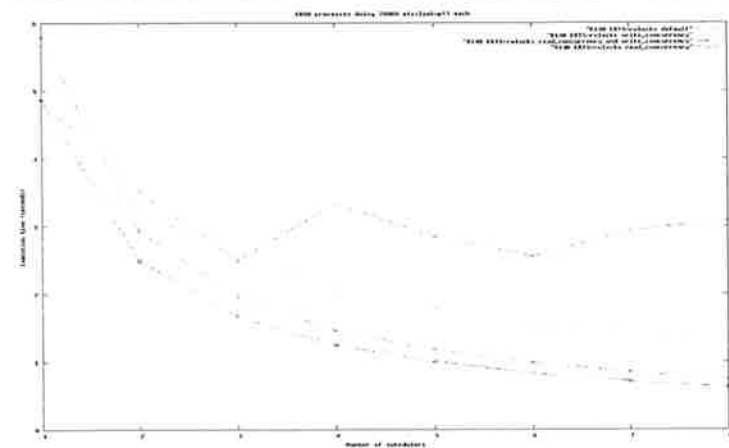
Benchmark Results – Machine A



ETS Table with write_concurrency and read_concurrency Options (R14B)



Benchmark Results – Machine A



Benchmarking



› Machine A

- SLES 10.2
- Kernel 2.6.16.60-0.39.3-smp
- NPTL 2.4
- x86_64
- 2x Intel Xeon L5430 @ 2.66 GHz

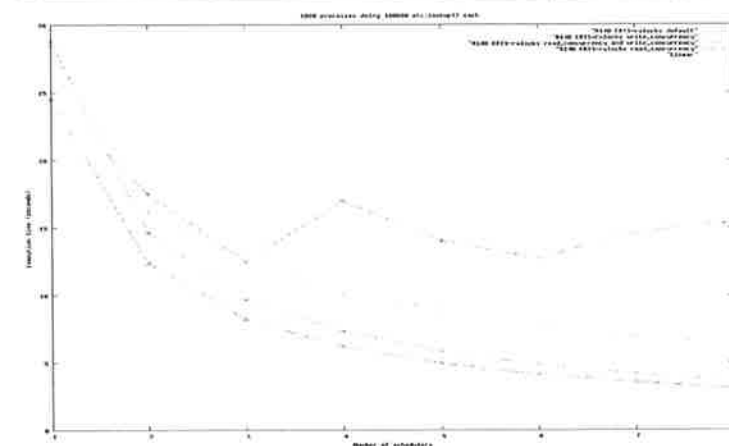
› Machine B

- Ubuntu 9.10
- Kernel 2.6.31-22-server
- NPTL 2.10.1
- x86_64
- 2x Intel Xeon <unknown id> @ 2.8 GHz

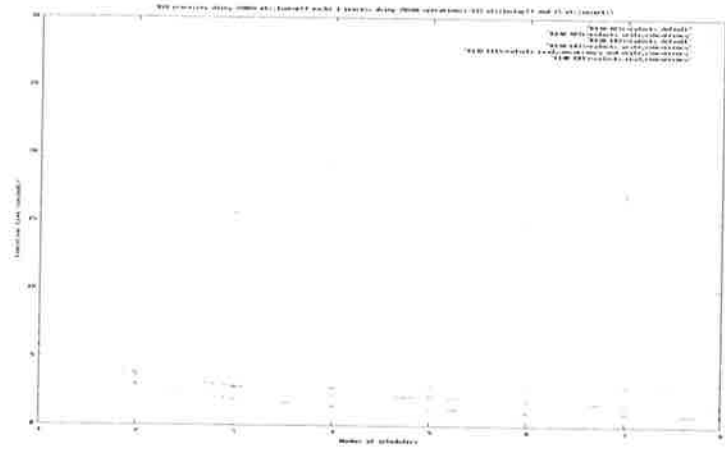
```
1> erlang:system_info(cpu_topology):
[[processor,[[core,{logical,0}],
  [core,{logical,1}],
  [core,{logical,2}],
  [core,{logical,3}],
  [core,{logical,4}],
  [core,{logical,5}],
  [core,{logical,6}],
  [core,{logical,7}]]]]
```

```
1> erlang:system_info(cpu_topology):
[[processor,[[thread,{logical,0}],
  [thread,{logical,1}],
  [thread,{logical,2}],
  [thread,{logical,3}]]]]
```

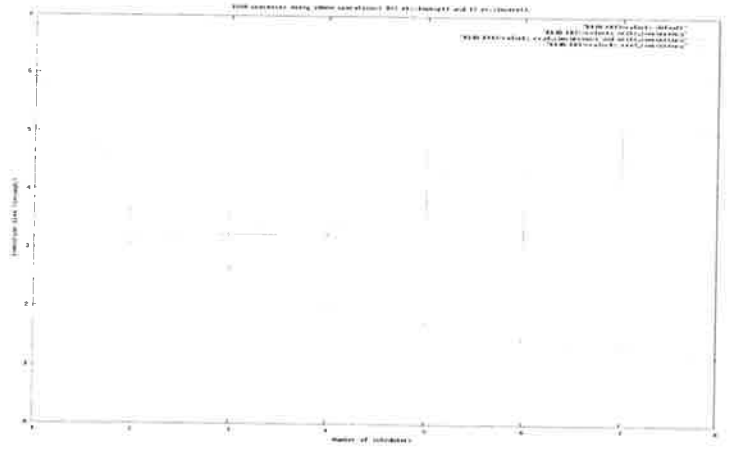
Benchmark Results – Machine A



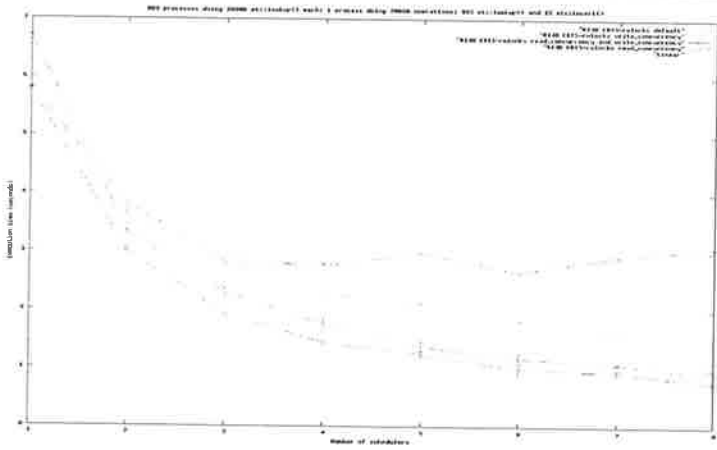
Benchmark Results – Machine A



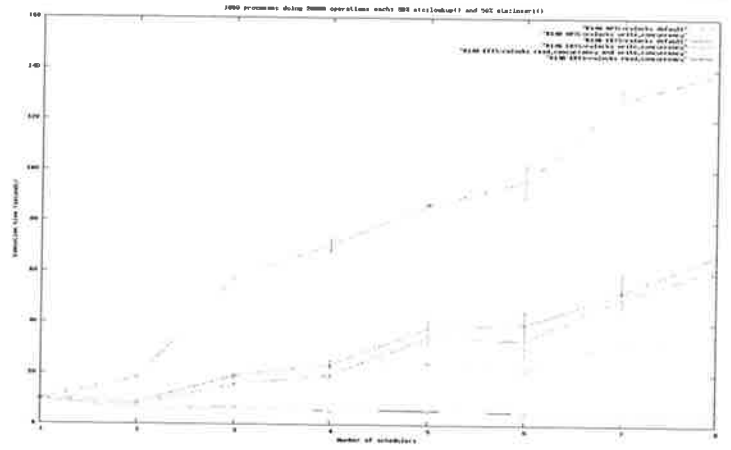
Benchmark Results – Machine A



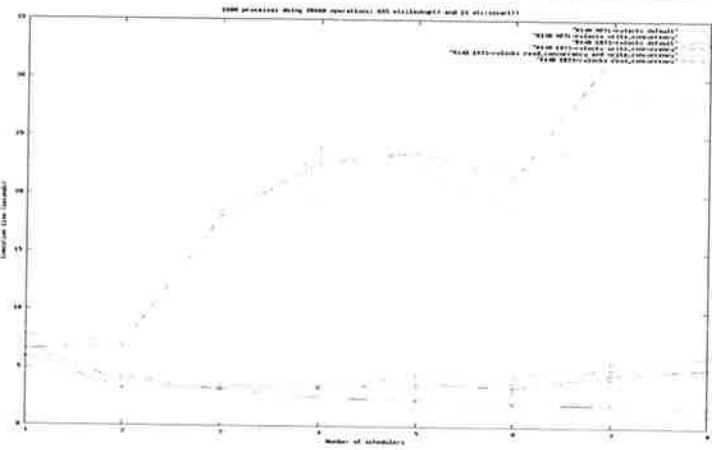
Benchmark Results – Machine A



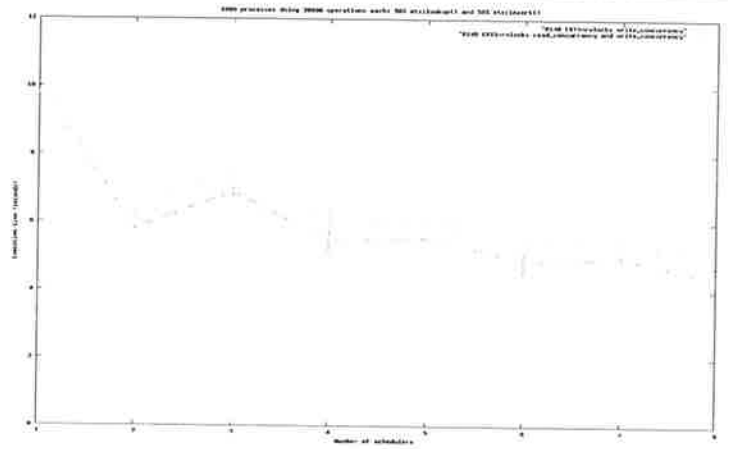
Benchmark Results – Machine A



Benchmark Results – Machine A

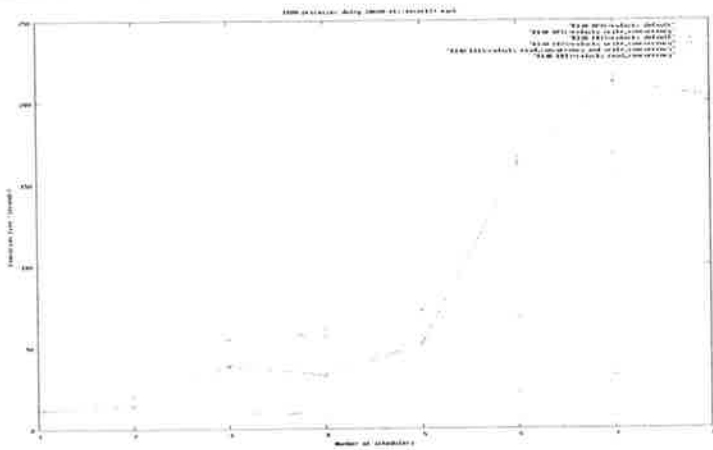


Benchmark Results – Machine A

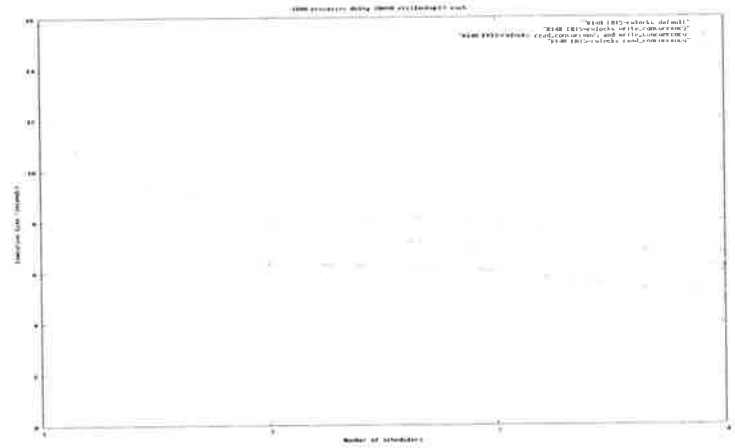




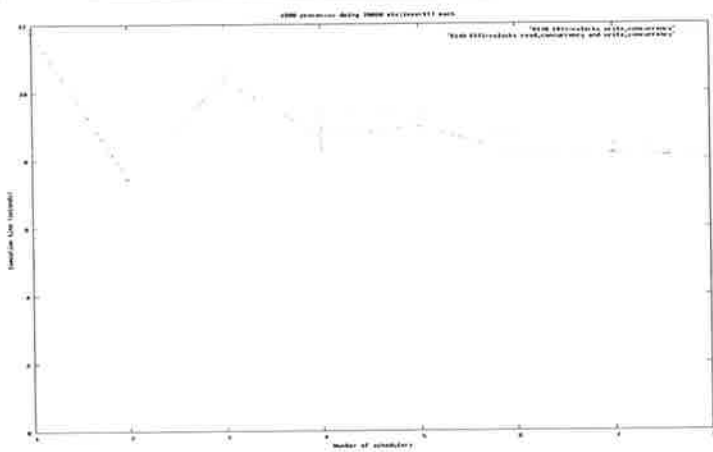
Benchmark Results – Machine A



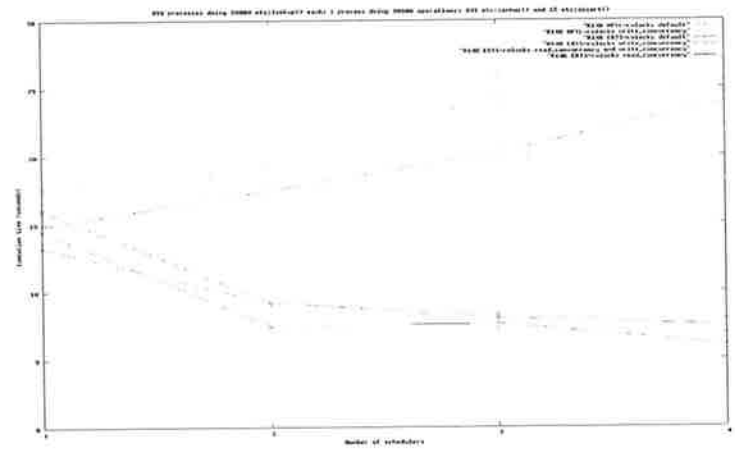
Benchmark Results – Machine B



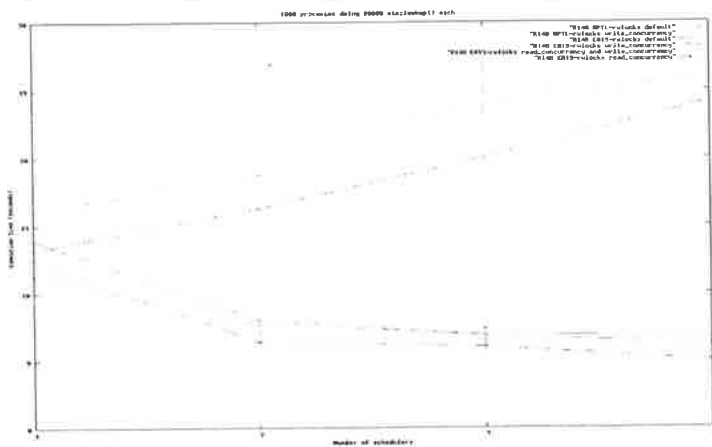
Benchmark Results – Machine A



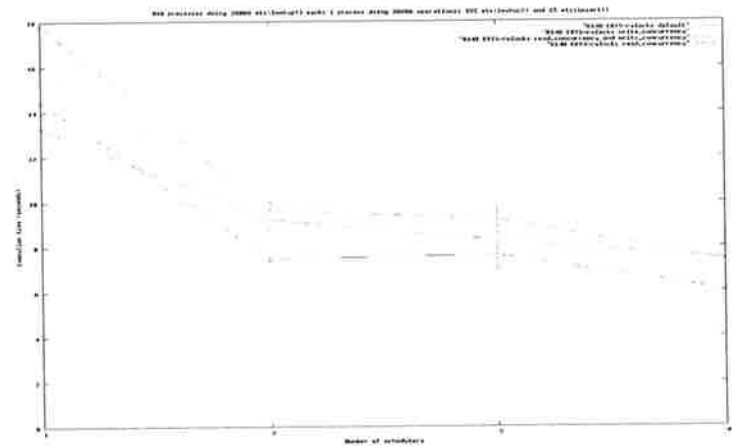
Benchmark Results – Machine B



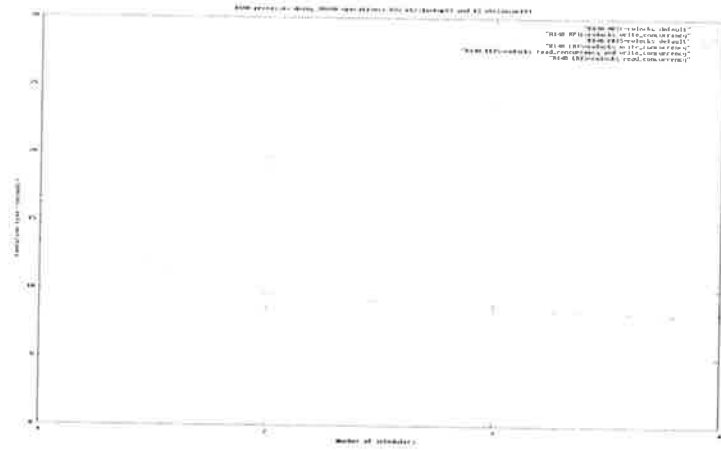
Benchmark Results – Machine B



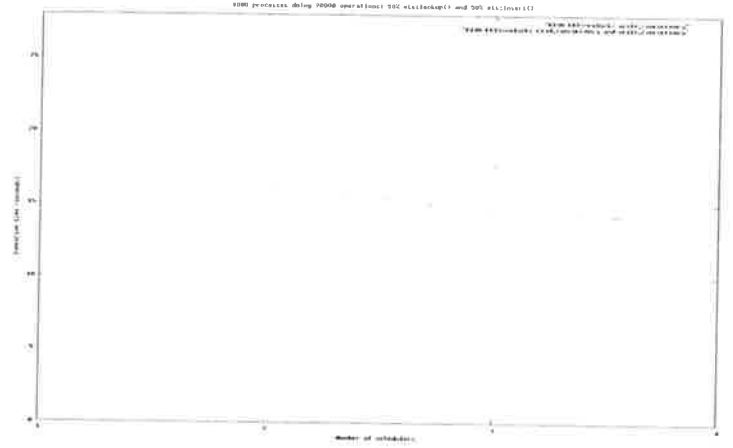
Benchmark Results – Machine B



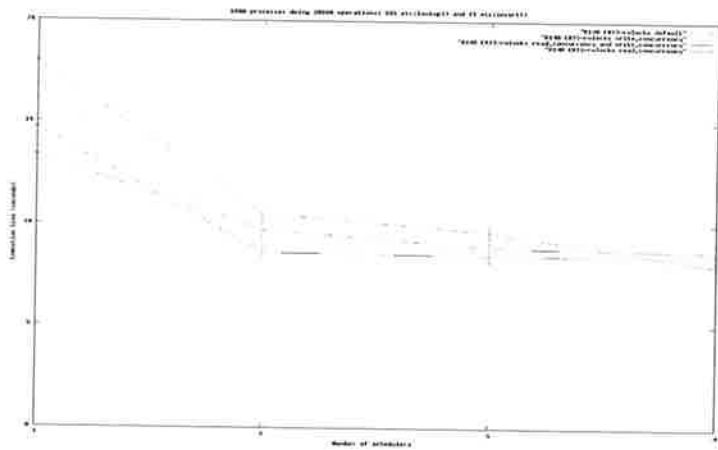
Benchmark Results – Machine B



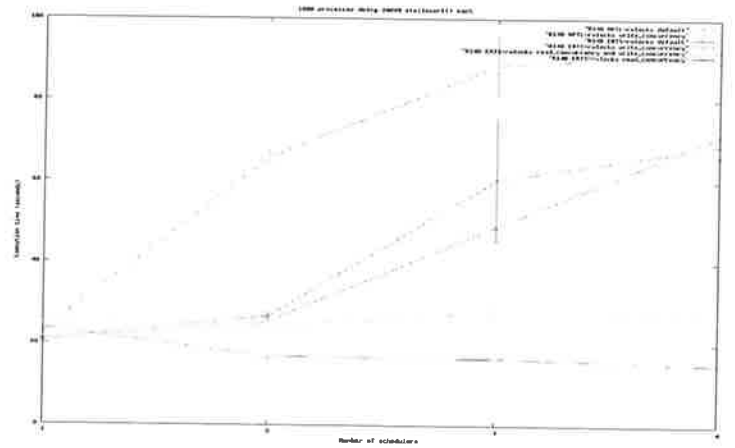
Benchmark Results – Machine B



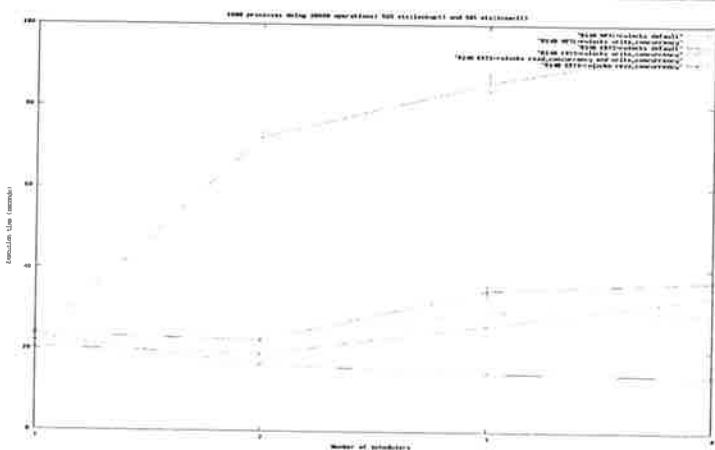
Benchmark Results – Machine B



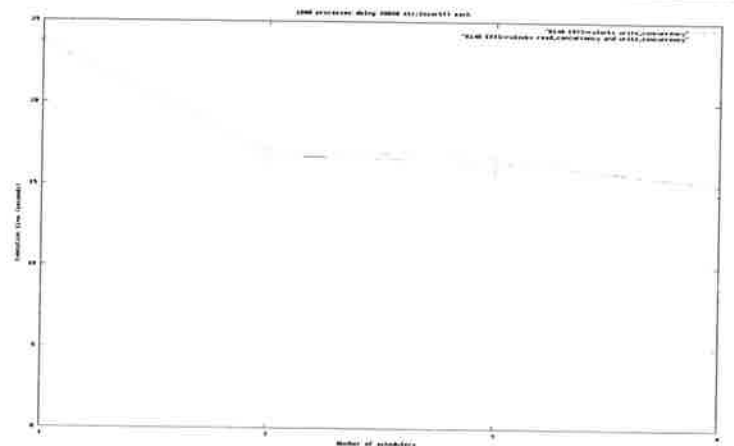
Benchmark Results – Machine B



Benchmark Results – Machine B



Benchmark Results – Machine B





ERICSSON

Kenneth Lundin
Manager of the Erlang/OTP dev team



Latest News From the Erlang/OTP team at
Ericsson

Abstract

Kenneth gives an update of the Erlang/OTP team's work at Ericsson - their current projects and plans for future.

Biography

Kenneth Lundin has been working with SW development since the late 70s. As a curiosity it can be mentioned that Kenneth was one of the pioneers in the use of C++ at Ericsson. Unsurprisingly Kenneth's interest for OO languages has been slightly revised since then. He joined the Erlang/OTP project in it's early stages 1996 and has been working both with application components and the runtime system since then. Has been managing the team for about 10 years now.



ERLANG/OTP LATEST NEWS

ERLANG USER CONFERENCE 2010
Kenneth Lundin



CONTENTS

- › Some highlights from the recent R14B release
- › Coming releases and other work
- › Positive Statistics on usage and contributions



SOME HIGHLIGHTS FROM THE R14B RELEASE



- › New optimized implementation of rwlocks in the Erlang VM
- › New auto imported BIFs
- › The new SSL implementation is now the default (since R14A), A number of bugfixes and improvements have been made thanks to feedback from Open Source users.
- › All together quite a lot of new things in R14A + B

- › R14B was the first release we made directly from GIT (no Clearcase involved from now on).

© 2010 Ericsson AB. All rights reserved.



OPTIMIZED RWLOCKS



- › New optimized implementation of rwlocks in the Erlang VM
- › There is one variant of rwlocks which is "reader optimized" and which gives huge performance improvements when the parallel read operations are dominating. Examples are:
 - When sending messages using a registered name
 - When having many multiple readers of an ets-table (the new table option 'read_concurrency' must be used when creating the table),
A simple benchmark with many parallel readers of an ets-table on a 2 x quad-core machine showed a speedup factor of 5.
- › There also a variant that is neutral (i.e not "reader optimized").
- › Both variants interleaves readers and writers during contention as opposed to NPTL (Linux Thread Library) which uses a reader/writer preferred strategy which can cause starvation.

© 2010 Ericsson AB. All rights reserved.



NEW AUTO IMPORTED BIFS

- › In R14A the semantics changed so that local functions will override auto imported BIFs
- › A change to what it should have been from the beginning
- › The change makes it possible to auto import more functions without introducing incompatibilities.
- › We have now made a number of BIFs from the erlang module auto imported.
- › `monitor/2`, `monitor/3`, `demonitor/2`,
`demonitor/3`, `error/1`, `error/2`,
`integer_to_list/2`, `list_to_integer/2`.

© 2010 Ericsson. All rights reserved.




THE NEW SSL IS NOW THE DEFAULT

- › The new SSL is now the default.
- › All communication is written in Erlang using `gen_tcp` (earlier there was a separate port program written in C built on top of the OpenSSL code) and using the `crypto` module built on `libcrypto` from OpenSSL.
- › Advantages with the new solution:
 - Can use Erlangs SMP support for parallel execution
 - Support for upgrade and downgrade from TCP to TLS and vice versa.
 - Uses less number of ports and file descriptors
 - Efficiency
 - Easier to maintain

© 2010 Ericsson. All rights reserved.





RELEASE PLANS



Preliminary

- › R14B01 to be released on December 8:th

Very Preliminary

- › R14B02 in March 2011
- › R15B in Q3-Q4 2011


© Ericsson 2010. All rights reserved.

WHATS COOKING FOR R14B01

- › New function `inet:getifaddrs` which returns all network interfaces and their addresses
- › All test suites converted to CommonTest format.
- › Compression flag for ets-tables
`(ets:new(...,[compressed,..])` , compression ratio depends on data. But 50% or more for complex data. Of course there is a performance penalty for this.
`erl +tc` , will compress all ets-tables.

© Ericsson 2010. All rights reserved.



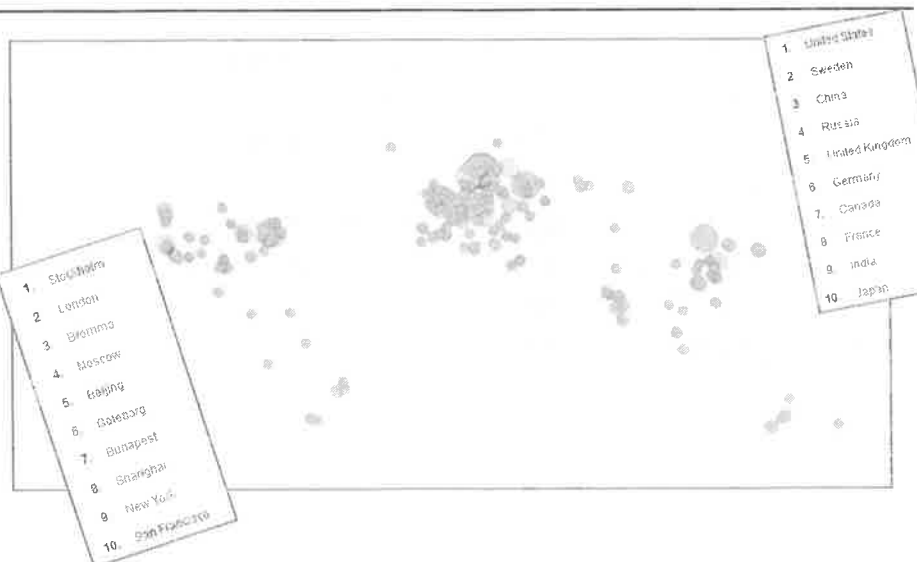
POSITIVE STATISTICS

- › 155 approved contributions from R13B03 until beginning of Nov (roughly a year)
- › From 52 different contributors
- › 7 contributors with more than 5 contributions
- › The activity in the Erlang community is really increasing, a great help in making the Erlang/OTP distribution even better

© Ericsson AB 2010. All rights reserved.



WIDESPREAD INCREASING INTEREST



© Ericsson AB 2010. All rights reserved.



