Functional Programming In Real Life

Erik Stenman
CTO
Klarna

Creative Payment Solutions
Introduction

* I will talk about **klarna**, a company that bet its future on **Erlang**, a functional programming language.

* I will tell you what **klarna** does, how we do it, why we do it this way, and whether it worked out or not... at least, so far.

* I will also tell you about **Erlang**, and **why** and **why not** to use it.
Klarna

* The business model:
  - Bring trust to Internet shopping.
  - Bring old style billing into the new IT-economy.

* Brief background:
  - Founded in **February 2005**.
  - With < $100,000 in venture capital.
  - Live system in **March 2005**.

* The company vision:
  - “Be the coolest company in Sweden.”
The Problem

• Internet shopping is a question of trust.
  – The shop has to trust the customer to get paid.
  – The customer has to trust the shop to send the right stuff.

• Many customers are uncomfortable using credit card over the Internet.

• Many banks are actually worried about the security of Internet shops handling credit card information.
The Solution

• Bring in a *trusted party*, i.e., klarna.

• Send an invoice with the goods to the *customer*.

• The *customer* pays after receiving the goods and takes no risk. The *customer* does not have to trust anyone.

• The *shop* is guaranteed (by contract) to get money from klarna. The *shop* only have to trust klarna with whom they have a written contract.
The Implementation

A customer surfs A webshop

xmlrpc

Credit information

identity kredit check

klarna
The Implementation

A customer surfs a webshop.

Credit information is communicated through xmlrpc.

Klarna stores the credit information in a database (DB).
The Implementation

A customer

surfs

A webshop

Ok, invno=42

Credit information

klarna

DB
The Implementation

A customer

Buy ok

Pack goods, print invoice, ship.

A webshop

Credit information

klarna

DB
The Implementation

A customer

A webshop

Credit information

Bank

klarna

DB

Pay
The Implementation
Some details

- The system is built from scratch using LYME (Linux, Yaws, Mnesia, and Erlang).

- So far we operate in Sweden and Finland, and we sell to Norway and Denmark.

- We have a distributed system with multiple servers to provide a fault tolerant, high availability solution.

- We aim for less than 5 minutes down-time per year, in a setting where we introduce new features in the system every week.

- The problem fits Erlang really well.
Why not use Erlang?

• The main reasons that I have heard of are:

1. Politics – Erlang is not C/Java, company policy.
2. One provider – Concern that Ericsson will stop supporting Erlang.
3. Lack of programmers – Erlang is still not mainstream how can we ensure we get qualified staff?

• When starting a new company, 1 is (usually) not a problem.
• I can't see 2 happening, and Erlang is open source anyway.
• When setting up in Stockholm, 3 is not a problem.
Erlang - Background

★ Developed by the Computer Science Lab at Ericsson.
★ Problem domain - Telephone exchanges
  - High Availability
  - Highly concurrent
  - Real time
  - Distributed
  - Continuous operation
  - In service upgrades
Modern financial applications
Erlang Design Goals*

* How can we build software systems that are as reliable as hardware systems.
* How can we make programming almost as easy as assembling hardware.

* This is my own reconstructed view of what the goal might have been, I was not in any way involved in the design of Erlang.
Erlang - Insight

To make a fault-tolerant system you need at least two computers.
Actually, to make a fault-tolerant system you really need at least three computers*.

* This is an insight about consensus algorithms shown by Leslie Lamport in “The Byzantine Generals Problem” (1982) and in more detail in “Lower Bounds for Asynchronous Consensus” (2004).
The Erlang designers realised “To do fault tolerant computing we need at least two isolated computers.”

This lead to concurrent programming with pure message passing and no shared state.

- Large number of (isolated) processes
- Communication through message passing
- No mutable state in processes
- Pure functional programming
A huge part of the success of Erlang comes from the standard library OTP (Open Telecom Platform).

OTP extends the fault tolerance in Erlang by providing standard patterns (or behaviours in Erlang lingo) for building telco-grade systems.

- Supervisors, with restart polices
- Generic servers
- Generic state machines
- Logging
The **Erlang** Advantage

- **Easy** to build fault-tolerant systems.
- **Rapid** development.
- **Low-maintenance** and easy upgrade.
- **Ability** to leverage multicore technology.
- **Network** programming is easy.
- **Good** way to get great programmers.
Easy to Build Fault-tolerant Systems

- **Erlang** was designed from the ground up with the purpose of making it easy to develop fault-tolerant systems.
- **Erlang** was developed by **Ericsson** with the telecom market in mind.
- **Erlang** supports processes, distributed systems, advanced exception handling, and signals.
- **Erlang** comes with **OTP-libraries** (*Open Telecom Platform*), e.g. *supervisors* and *generic servers*. 
Rapid Development

• **Erlang** has a number of features to support rapid prototyping and fast development:
  – Automatic memory management.
  – Symbolic constants (atoms).
  – An interactive shell.
  – Dynamic typing.
  – Simple but powerful data types.
  – Higher order functions and list comprehensions.
  – Built in (distributed) database.
Low-maintenance and Easy Upgrade

• **Erlang** has a number of features that makes it easy to maintain and upgrade:
  – Hot code loading.
  – Distribution.
  – Interactive shell.
  – Simple module system.
  – No shared state.
  – Virtual machine.
Ability to Leverage Multi Core

• The concept of processes is an integral part of **Erlang**.
• The Erlang Virtual machine (**BEAM**) has support for **symmetric multiprocessing**.
• No shared memory -- easier to program.
• As Joe Armstrong is fond of saying:
  “Each year your sequential programs will go slower. Each year your concurrent programs will go faster.”
Network Programming is Easy

★ Distributed **Erlang** solves many network programming needs.

★ Setting up a simple socket protocol is a breeze.

★ The binary- (and now bit-) syntax makes parsing binary protocols easy.

★ There are simple but powerful libraries for HTTP, XML, XML-RPC and SOAP.
Good Way to Get Great Programmers

• **Nice paradox:**
  The lack of *Erlang* programmers makes it easier for us to find great programmers.

• There are many great C and Java programmers, I’m sure, but they are hidden by hordes of mediocre programmers.

• Programmers who know a functional programming language are often **passionate** about programming.

• Passionate programmers makes *Great Programmers™*. 
Building a Scalable Fault-tolerant 24/7 System

★ You need at least three of everything
★ Start simple
★ Only optimise when and where necessary
★ Scale by adding nodes, make each node the same as all other nodes
★ Monitor everything
Erlang Downsides

★ It is not mainstream*

− There are not many third party libraries yet.**
− Even if we can find great programmers it is hard to find enough programmers who know Erlang or who even want to use Erlang.

* A big advantage according to Paul Graham in “Beating the Averages”.

** On the other hand Erlang doesn’t suffer from DLL-hell.
Erlang is Gaining Popularity

- “Erlang for Concurrent Programming”, by Jim Larson, Google: “Designed for concurrency from the ground up, the Erlang language can be a valuable tool to help solve concurrent problems.” (http://www.acmqueue.com/modules.php?name=Content&pa=showpage&pid=556)


When not to Use Erlang

★ Don’t try to build a stand-alone GUI.
★ Don’t expect to do fancy text handling out of the box.
★ A fancy web-GUI is not easy to do yet.
★ Fast file crunching is not Erlang's forte.

– But **Erlang** can easily interface with other languages and applications, so use **Erlang** as the glue to write the robust server core, and plug in the missing parts.
Did it work?

- **Erlang** has been a great help in providing rapid development with maintained high availability.

- **Klarna** has introduced new services several times per year.

- The system has never been down. Not even while upgrading the system hardware or moving the hosting to a new site.
Some Customers

We have signed with over 4000 internet shops.
(CDON, Dustin, Ginza, Discshop, SIBA, Webhallen, Gymgrossisten)
Did it work?

- The business model has been sound.
- Number of connected stores:
- Number of employees ~150.
- Turnover:
  - 2004: ~0 SEK
  - 2005: 1.5 million SEK
  - 2006: 13.5 million SEK
  - 2007: 35 million SEK
  - 2008: 91 million SEK
  - 2009: 200 million SEK
Conclusion

**Erlang** has proven itself as the right tool for high availability web servers.

Questions?