Safe Sessions for Erlang

Adrián Palacios
MiST, DSIC, Universitat Politècnica de València

Introduction

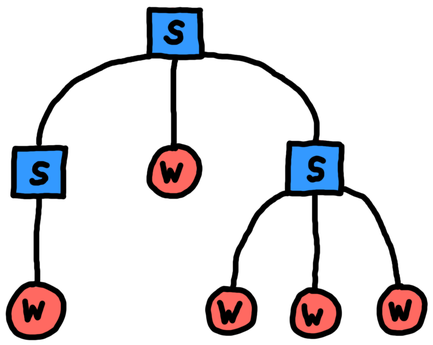
Erlang systems have become an example of fault-tolerant systems thanks to the *Let It Crash* philosophy.

Let It Crash

The *Let It Crash* philosophy discourages excessive error handling in programs, and its advice is to let processes crash in case of error and quickly restart them afterwards.

This is possible by building a process supervision tree where:

- Workers: Do all the hard work
- Supervisors: Restart workers if they crash



But supervisors do not make any verification when restarting workers. That can lead to an **inconsistent system state**.

We propose **safe sessions**, an automatic recovery strategy for Erlang, as a complement to the *Let It Crash* philosophy

In safe sessions, concurrent actions are registered, and the system can return to a safe state in case of error.

This work is based on the reversible semantics for **Erlang** from **Nishida, Palacios and Vidal (LOPSTR'16)**.

Scan QR code to download this poster!

The language

The **Erlang** language:

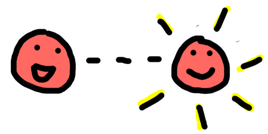
- functional and concurrent features
- concurrency based on the actor model

Some companies that use **Erlang** in their production system

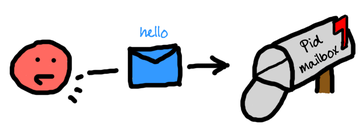
ERICSSON

Concurrent Actions

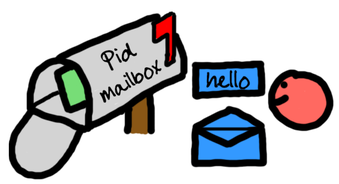
Spawn: Create a new process



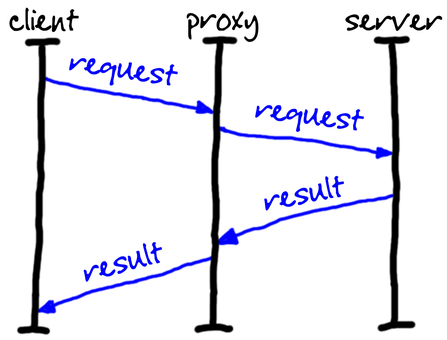
Send: Send a message to another process



Receive: Suspend execution until a message from the mailbox matches any of the receive clauses



An example of **Erlang** computation where the *client* process sends a request to the *proxy* process, which forwards this request to the *server* process, is shown here:



Safe Sessions

We add a new construct to our language:

safetry $expr$ end

Before the evaluation of $expr$, we store a snapshot of the process state. If the evaluation of $expr$ goes wrong, the process is **restored** with the information available in the snapshot.

Causal Consistency

An action may be undone only if every action caused by that action has not been executed yet or has been undone

Restoring the state is not enough to ensure causal consistency, we must also undo the effects of its spawn and send actions. This is solved by "propagating the safety" to other processes.

Safe sessions are implemented using:

- Monitors:
  - Intercept incoming and outgoing messages
  - Send signals between themselves to propagate the safety
- Instrumentation:
  - Enable interaction of processes with their monitors

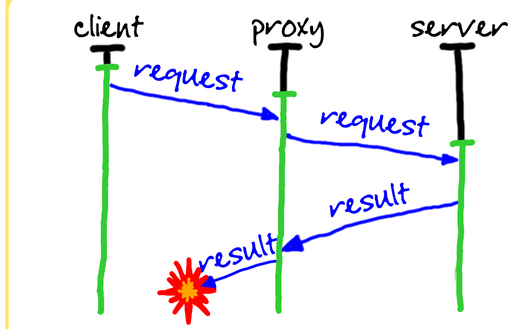
Instrumentation is performed by the auxiliary function $[\![\ ]\!]$ as

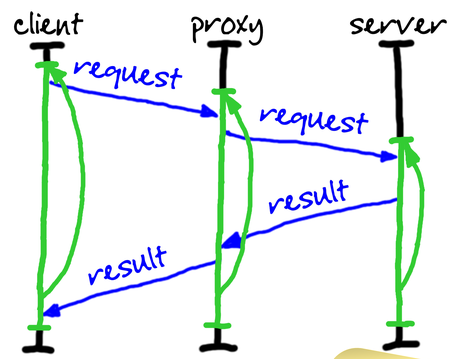
Program Instrumentation

$$
\begin{array}{ll}
[\![\text{safetry } expr \text{ end}]\!]_M & \rightarrow M \,!\, \langle \text{start\_session} \rangle, \\
 & \quad [\![expr]\!]_M, \\
 & \quad M \,!\, \langle \text{end\_session} \rangle \\
[\![\text{spawn}(\ldots)]\!]_M & \rightarrow M \,!\, \langle \text{spawn}(\ldots) \rangle, \\
 & \quad \text{receive } \langle \text{spawn\_with}, P \rangle \rightarrow P \text{ end} \\
[\![\text{self}()]\!]_M & \rightarrow M \\
[\![Pid \,!\, expr]\!]_M & \rightarrow M \,!\, \langle \text{send}(Pid), [\![expr]\!]_M \rangle, \\
 & \quad \text{receive } \langle \text{sent\_as}, E \rangle \rightarrow E \text{ end} \\
[\![\text{receive } clauses \text{ end}]\!]_M & \rightarrow M \,!\, \langle \text{receive}, clauses \rangle, \\
 & \quad Arg = \text{receive} \\
 & \qquad \langle \text{rec\_msg}, Msg \rangle \rightarrow Msg \text{ end}, \\
 & \quad \text{case } Arg \text{ of } [\![clauses]\!]_M \text{ end}
\end{array}
$$

Basically, concurrent actions are replaced by queries to the monitor.

Example





Related Work

Field and Varela (POPL'05) checkpoint-based approach has some similarities with our proposal, although they aim at defining a new language (rather than extending an existing one).

Neykova and Yoshida (CC'17) define an interprocedural recovery strategy based on session types which determines the processes to be restarted in case of error. Our proposal is more fine-grained, and it would allow us to define an intraprocedural recovery strategy in addition to the interprocedural one.

Conclusions

We have presented the basic aspects of an automatic technique for recovery in **Erlang** systems.

In the future, we will:

- refine our design of safe sessions
- develop an implementation
- compare our implementation against other approaches

LaTeX TikZposter