



Erlang Training and Consulting Ltd

Inviso/Onviso

Aniko Nagyné Víg

Stockholm, 13 Nov 2009



Agenda

Goal

Erlang tracing and tools overview

Inviso

Onviso

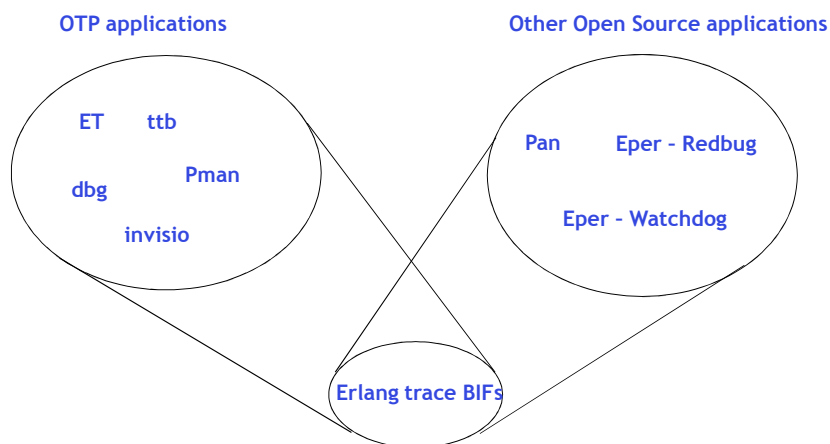
Exercises

Goal

The aim of the ProTest Project is to develop and improve testing and tracing tools.

In this part of the project we would like to create a simple interfaced and safe tracing tool enable to use from a remote node. We chose to extend the existing Inviso application.

Tracing Tools available for Erlang



Erlang trace BIFs

All tracing solutions discussed are based on trace BIFs

Provides data for monitoring execution and memory usage

Events are sent as trace messages of the format

- {trace, Pid, Tag, Data1 [,Data2]}

Events include

- Message passing
- Gc and memory usage
- Process activity

At any one given time, only one process may receive trace events from another process

Copyright 2009



Erlang trace BIFs possible settings

Dynamically enabled. Filtering on pids: existing, new, all, pid()

Flags

- 'send', 'receive' for message passing events
- 'running' for scheduling events: 'in' and 'out' messages sent when a process is scheduled or preempted
- 'exiting' for scheduling exiting processes. Message tags: 'in_exiting', 'out_exiting', and 'out_exited'.
- 'procs' for process-related events. Message tags: 'spawn', 'exit', 'link', 'unlink', 'register', 'unregister', 'getting_linked', 'getting_unlinked'
- 'call' for function calls. Message tags: call, return_from It can be combined with arity, return_to and silent flags. In trace_pattern match_specifications can be defined.
- 'set_on_spawn', 'set_on_first_spawn', 'set_on_link', 'set_on_first_link' sets the inheritance of the trace flags in the new processes
- 'garbage_collection'. Message tags: 'gc_start', 'gc_stop'.
- 'timestamp' and 'cpu_timestamp' for including timestamps to each trace message.

Copyright 2009



Dbg

The most popular debugging tool for Erlang (in Erlang OTP).

Text based debugger, suitable for text based terminals

Provides a friendly and simple interface to the trace and trace_pattern BIFs providing the same functionality:

- By setting trace flags for processes, ports and functions and manipulating the trace patterns even on multiple nodes

Suitable on large systems: Small impact on system performance, but traces can be set only one node at a time, and not remotely.

Not always safe, e.g. dbg:tracer(), dbg:p(all,m) can kill the running system

Copyright 2009



Advantages of invisio & dbg

Invisio

- Configurable output, can be sent to:
 - erlang interface
 - log file
 - text output in erlang shell
- Powerful tool for automatic tests:
 - custom profiling
 - property testing
 - white-box testing
- Strong support for usage in distributed systems

Dbg

- Light tool, very useful for ad-hoc tracing on newly found faults with simple interface
- Helper function (fun2ms) for generating match specifications

Copyright 2009



The invisio Tool

- ✂ The **invisio** tool is an erlang trace tool
- ✂ Provides an interface to the **trace** and **trace_pattern** BIFs
- ✂ Designed with distributed systems in mind
- ✂ Has small impact on system performance
- ✂ Suitable for use on large distributed systems
- ✂ Has more functionality than **dbg**, but too complex to be considered a simple debug tool like **dbg**

Can be used for:

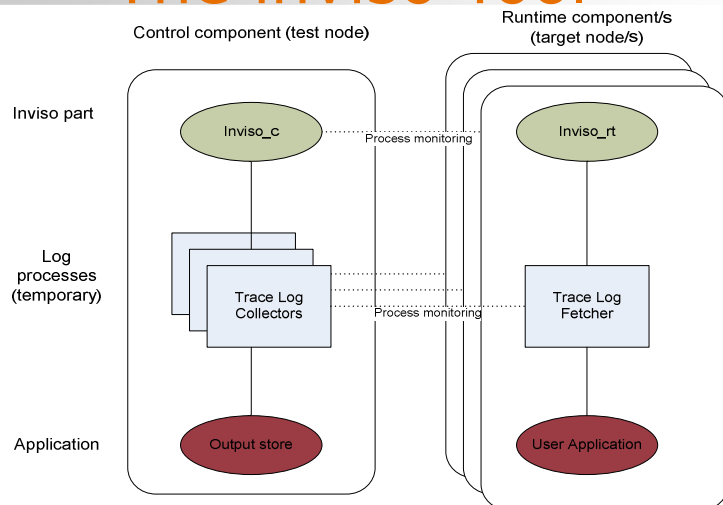
- ✂ **Troubleshooting:** tracing and debugging
- ✂ automatic white-box testing
- ✂ system profiling
- ✂ property checking



Copyright 2009



The invisio Tool



Copyright 2009



The Inviso Tool - Start-up requirements

An instance of RT is started on each node in the system being traced.

These can be controlled from one node, which may be part of the system or a separate, dedicated test node.

Only requirements are:

- ✂ the **runtime tools** application is started on all traced nodes
- ✂ test node can communicate with all the system nodes

Copyright 2009



The Inviso Tool - Trace

Trace information can be:

- ✂ Logged to file
- ✂ Passed as parameters to custom functions
- ✂ Displayed as text in the Erlang shell
- ✂ Sent to another Erlang nodes' runtime system (good for minimizing the performance impact on system under test)
- This other node can be a specialised debugging/testing node

Copyright 2009



The inviso Tool - Performance

Good to design tests and traces to run in a **separate test node** beside the system, because:

- ✂ Easier to divide test from core functionality of the system
- ✂ Tests can have minimal impact on system operation, both configuration & performance wise

System protection under test

- ✂ Optional overload protection, vital for live systems. If a user defined function decides that a node is overloaded then tracing will be suspended on that node
- ✂ Essential in case overloading is caused by the tracing

Copyright 2009



The inviso Tool

Can be used for debugging if the fault is more subtle or harder to find and something more powerful than dbg is needed. Inviso is more suited to designed traces and tests.

The interface is powerful but too complex to be considered a simple ad-hoc debug tool like dbg, it would be a simple task to develop a wrapper to make this more usable for debugging

Copyright 2009



Goal of the Onviso extension

- ◆ Provide an easy to use, remotely usable, safe, online tracing tool
- ◆ Include our extension to the Inviso application within the OTP release (planned meetings with Kenneth Lundin)
- ◆ Include possible extensions for property checking

Onviso - Extension of the Inviso

Easy to use API as a wrapper for the original Inviso

- Using only 2 function to set up tracing on multiple nodes and merge them
- Added short-cuts for common match specifications
- Command line interface

Onviso - Extension of the Inviso

Added extra functionality

- Retrieve the status of the recent traces and configurations allowing to run different merge functions on the same trace data
- Default choices are provided for merging (the simplest is to write every trace into a file) and overload protection
- The merge functionality can be used for property testing

Copyright 2009



Setting up a trace -starting the nodes

Start and initialise all nodes in the system: in this example the server and client nodes

```
> client:init('server@machine').
```

```
> server:start().
```

Start the Inviso node

Set the same cookie on every node:

```
> erlang:set_cookie(node(), invisio).
```

Copyright 2009



Setting up a trace - define the patterns

Onviso uses the following format to specify a pattern:

```
{Module, Function, Arguments, MatchSpecification}
```

Example patterns:

```
{client, get, '_', return}
```

```
{server, loop, '_', []}
```

Please note the difference between the original MS
(`{['_'], [], [{return_trace}]}`) and the return short-cut

Copyright 2009



Setting up a trace

Possible trace calls:

```
>onviso:trace(Patterns, Flags). %traces only the local node
```

```
>onviso:trace(Patterns, Nodes, Flags).
```

```
>onviso:trace(Patterns, Nodes, Flags, OverLoadProtection).
```

To set up the trace on the Inviso node for example:

```
> onviso:trace([server, loop, '_', []],  
               {client, put, '_', []},  
               {client, get, '_', return}],  
               ['server@laptop', 'client@laptop'],  
               {all, [call]}).
```

Copyright 2009



Stop a trace

Every trace call returns a trace reference identifier. This id can be used to stop or merge a trace.

```
>onviso:stop(Id).
```

The traces are collected to files and distributed back to the Inviso control node.

Copyright 2009



Merge a trace - default examples

```
> onviso:merge(Id, void, void, shell). %result in the shell
```

```
> onviso:merge(Id, void, void, file). %result in  
“outputId.txt” file, other easy options {file, Name},  
{file_prefix, Prefix}
```

Please note:

- You can merge the trace files more than once.
- Even if the target node is under constant use, the merged logs will always contain the same result as the first merge, because the tracing stopped then.

Copyright 2009



Merge a trace - custom function

```
> BeginFun = fun(_InitData) -> {ok, 0} end.  
> WorkFun = fun(_Node, _Trace, _PidMapping,  
Count) -> {ok,  
Count + 1} end.  
> EndFun = fun(Count) -> io:format("We  
collected ~p traces.~n", [Count]) end.  
> onviso:merge(1, BeginFun, WorkFun, EndFun).
```

Copyright 2009



Short-cuts -- predefined atoms

To make it easier to adapt the tool some simplification are built in the interface.

1. Predefined match specifications like 'return' and 'caller' for getting the return values
2. Atoms defining the output in predefined EndFuns during merge like 'shell', 'file', {'file', Name}, {'file_prefix', Name}

Copyright 2009



Predefined examples for most common merge functions

1. Predefined merge functions for basic profiling either to shell or file
2. Built in trace and merge with overload protection

Copyright 2009



Exercises

Copyright 2009

