ICT-287510
RELEASE
A High-Level Paradigm for Reliable Large-Scale Server Software
A Specific Targeted Research Project (STReP)

D7.3 (WP7): Exploitation Plan for RELEASE

Due date of deliverable: 30.09.2013
Actual submission date: 14.10.2013

Start date of project: 1st October 2011
Duration: 36 Months
Lead Contractor: The University of Glasgow Revision: 0.1

Purpose: Summarize the exploitation activities to be done by the partners during the project

Results:

- **We have conducted a market analysis by providing an overview how different programming languages support writing scalable, distributed software.**
- **We have summarized the advantages of the Erlang programming language**
- **We have identified the exploitable results of the RELEASE project**
- **We have created a SWOT analysis to investigate market findings at the current stage of the project**

Conclusion: There is a market need to strengthen the language support to write disturbed software.

Project funded under the European Community Framework 7 Programme (2011-14)		
Dissemination level		
PU	Public	

PP	Restricted to other programme participants	(including the Commission Services)	
RE	Restricted to a group specified by the consortium	(including the Commission Services)	*
CO	Confidential only for members of the consortium	(including the Commission Services)	

Exploitation Plan for RELEASE

Author: eva.bihari@erlang-solutions.com
Torben.hoffman@erlang-solutions.com
Francesco.cesarini@erlang-solutions.com
jesper.louis.andersen@erlang-solutions.com

1 Executive Summary

This deliverable defines strategy and plans exploitation activities of the RELEASE project. The exploitation plan is strongly connected to deliverable D7.2: Dissemination and Collaboration Plan. In this deliverable we discuss exploitation of the project results within and beyond the project life time.

We start with identifying the potential exploitable elements. As the main objective is adding new language elements and supporting tools to the Erlang VM and language our market analysis includes by providing an overview of programming languages (Section 4). We define Erlang programming language advantages and challenges related to the RELEASE project, and outline exploitable components implemented during the project. Then in the exploitation initiatives we cover consortium partner intentions to exploit individual project results (Section 5). We conclude with a SWOT analysis that outlines market findings at the current stage of the project (Section 6).

2 Introduction

The RELEASE project aim is to scale the radical concurrency-oriented programming paradigm to build reliable general-purpose software, such as server-based systems, on massively parallel machines (10^5 cores).

The exponential growth in the number of cores requires radically new software development technologies. Many expect 100,000-core platforms to become commonplace, and the best predictions are that core failures on such an architecture will be common, perhaps one an hour. Hence we require programming models that are not only highly scalable but also reliable.

The trend-setting language we use is Erlang/OTP that has concurrency and robustness designed in. Erlang/OTP has inherently scalable computation and reliability models, but in practice scalability is constrained by aspects of the language and virtual machine. Moreover existing profiling & debugging tools don't scale.

The RELEASE consortium is uniquely qualified to tackle these challenges and we work at three levels:

- Evolving the **Erlang virtual machine** so that it can work effectively on large scale multicore systems;
- Evolving the language to **Scalable Distributed (SD) Erlang**, and adapting the OTP framework to provide both constructs like locality control, and reusable coordination patterns to allow SD Erlang to effectively describe computations on large platforms,

while preserving performance portability;
- Developing a **scalable virtualization infrastructure** capable of creating, managing and dynamically scaling super-clusters of smaller heterogeneous clusters, based on capability profile matching.

State of the art tools will be developed that allow programmers to understand the behaviour of massively parallel SD Erlang programs. The effectiveness of the RELEASE approach will be demonstrated by using demonstrators and two large case studies on a Blue Gene. Erlang is a beacon language for distributed computing, influencing both other languages and actor libraries and frameworks. Hence we expect the project to make a strong and enduring impact on computing practice in the next two decades.

The aim of this deliverable is to highlight the potential for further dissemination and exploitation. We start with providing a market analysis and concentrating on the current state of the project's tools and techniques in Section 3. We cover pointers for how best to exploit our work further by discussing the project tools and outcomes individually in Section 4, and providing a SWOT (Strengths, Weaknesses, Opportunities, Threats) analysis in Section 5.

Partner Contribution to WP7. All partners contributed to the deliverable D7.3 by providing corresponding information about the project results and exploitation activities. We had multiple teleconferences and multi-partner face-to-face meetings.

3 Exploitation

RELEASE Project adds new language elements and supporting tools to the Erlang VM and language, therefore it is essential to make the material and the result data as widely available as possible. The real success of the project will be measured according to the dissemination activities throughout the project and the exploitation of the developed material afterwards.

The materials and the results have been exploited as follows:

- **Open Source Model**

The partners who release their frameworks, tools and improvements to Erlang as open source gain the goodwill of the community while making commercial and academic end users more competitive and productive. The open source model will also help with adoption, increasing the size of the overall market.
SD Erlang as an extension to the Erlang programming language will be released as part of OTP that is an Open Source software. (D3.1, D3.2, D3.3).

Scalable Reliable OTP Release (D3.4) will be released as Open Source software as well.

- **Contributing to Research**

Dissemination of research and development throughout the course of the project is being carried out through publication in technical and scientific journals and presentations at academic and commercial conferences. Results of the project are used in academic courses and are the result of PhD and Master's thesis projects.
List of publications are available in D7.2 (Dissemination Plan).
During the development of SD Erlang continuous research activities are carried on, for details see D7.2.

- **Commercial Exploitation**

Growing the Erlang community and the individual tool exploitation allows companies who sell services and tools to not only increase their size of the market and market share when selling training, consulting, support and systems development, but also provide routes into new markets with new products.
Wombat (a tool for deployment and management of Erlang systems) will make easier to deploy applications, which can be an interesting use case for the customers.

Erlang VM improvements (for details see D2.3) will speed up ETS table handling and the newly introduced thread progress functionality allow implementing lock-free algorithm, which will improve the performance of concurrent systems. These benefits will be useful for the whole Erlang community (both for commercial and research activities)

Reliable Sim-Diasca Simulation (D6.1) as a use case can prove the benefits of using SD Erlang and other achievements by the project and can generate customer interest as well.

- **Improving In-house Technology and Tools**

The results of RELEASE has allowed companies and academic institutions to improve their own internal tools and frameworks, reducing the gap to developing and researching resilient distributed systems which scale on multi-core architectures. This route has allowed the end users to be drivers and early adopters of technological innovations.
Online SD Erlang Profiling and Refactoring Tools (D5.2), systematic testing and debugging tools (D5.3) and the Interactive SD Erlang debugger tools will help Erlang developers during their daily work.

Date	**Action**	**Type of activity**	**Target groups**	**Goals during the project's lifetime**	**Goals after the project's lifetime**
Dec 2011	Creation of the Web Site	Web Site of the project	Public	Show the progress of the project to the public. Updates to the Web-site showing the continuous progress of the project.	Show the deliverables of the project
July 2012	Creation of dissemination strategy	Dissemination Plan	Project partners	Full dissemination of the project & its deliverables	Creation of the conditions that will allow the exploitation later of project's results.
August-Sept 2013	Creation of an exploitation strategy	Exploitation plan	Project partners	Coordinate the partners in order to achieve the goals of the exploitation strategy	Exploitation of the project's results commercially after the project has finished
Oct-Dec 2013	Creation of leaflets & promotional material	Leaflets, powerpoint presentation, article about the project brochures	Public	Use of these materials in presentations, meetings&conferences in order to make the project result known	Creation of base documentation which will be used later for the exploitation of the project's result

Dec 2013	Dissemination event (mid-project workshop)	Dissemination Event in London. UK	Target group + Public		Creation a list of contacts of interested parties
Year 2012-2013	Seminars/professional articles	Participation in seminars and conferences	Target group		Make the projects' results known and present the deliverables to potentially interested stakeholders
Oct 2013 – Aug. 2014	Case study to run SD Erlang application in Clouds	Application development	Target group + Public	Compare the performance of distributed Erlang and SD Erlang	Make the results known

4 Market analysis

Introduction

Erlang is the technology of choice for the RELEASE project, but it is not the most widespread programming language, so in this market analysis we will compare Erlang to other programming languages and show where Erlang has an advantage and where it has challenges.

Comparing technologies – in particular programming languages – is often a very subjective matter and often done on without a clear indication of how to do the comparison. In most cases it will be a proponent of a particular technology that chooses what aspect to compare. In this case we will try to single out the most important aspects related to the charter of RELEASE and be as objective about how Erlang stands on these properties compared to other technologies. That said, the properties naturally have a very good fit with the strengths of Erlang, so this comparison has that bias to deal with.

In addition to these properties there is also a set of fundamental problems with the modern and near future machines with many cores that needs to be factored into the comparison.

The properties and problems are as follows:

Multicore space properties

- **Concurrency** — The property of having multiple computations executing simultaneously. For our purpose, concurrency is a property of the programming language, describing what tools are available to express simultaneous computations.
- **Parallelism** — The ability to reduce *cost* of a computation by spreading computation over multiple physical machine cores. Here *cost* most often refers to the time it takes to produce the result of the computation. Thus, parallelism is a property of the machine and focuses of the productivity of the system and its ability to use multiple cores. It is important to stress this is orthogonal to concurrency: it's possible to run a concurrent program on a single-processor machine by simply switching concurrent context from time to time through multi-tasking and time sharing. This switch can happen either cooperatively or pre-emptively depending on the runtime.
- **Resilience** — It's a property, which shows how the system copes with failure. Most notably, partial failure in the system, where one core dies whilst the other cores continue to run on. Also, this addresses the problem where one part of the system crashes due to software failure and whether that causes a total system abort or just a partial system abort.
  For a machine with 10^5 cores, some of them will eventually fail. This means resilience is a must-have
- **Distribution** — The ability for the system to communicate between physical machines in a seamless manner, or a layer needs to be written in order to achieve this.

Problems with 10^5 cores massively parallel machines:

- **Memory bandwidth** – one cannot get data to the cores quickly enough. This means that the memory needs to be split up into banks. In effect, large shared heaps pose a difficulty here, where a message-passing model fits perfectly for a case study.
- **Isolation** - One program must not crash another program running in parallel. The crashes can be due to software mistakes, or hardware faults. There are some lesser variants of this for large programs: If they use libraries written by others, or the team writing code is large and split over several continents. As software grows, the risk of error increases.; and as more hardware is added, the risk of hardware failure increases too. Isolation is a necessary requirement in such a setting.
- **Embrace copying** - If data is copied into a process, it means it now has a local cache of the data. If data is immutable, this is a good thing. It means that one process cannot destroy the data held by another process, however 'badly' the other processes 'operate' this data. Many languages prefer passing pointers due to "speed" or "performance" but that solution is a risk with regards to resilience.

The traditional way of offering concurrency in a programming language is by using threads. In this model, the execution of the program is split up into concurrently running tasks. It is as if the program is being executed multiple times, the difference being that each of these copies operated on shared memory.

Shared memory might have several problems:

- ***lost-update***: For example if two processes try to increment the value of a shared object: acc. If they both retrieve the value of the object, increment the value and store it back into the shared object; as these operations are not atomic, it is possible that their execution gets interleaved, leading to an incorrectly updated value of acc. Using locks can solve this problem.
- ***deadlock*:** If two processes try to acquire the same two locks A and B. When both do so, but in a different order, a deadlock occurs. Both wait on the other, to release the lock, which will never happen.

In order to avoid the previously mentioned problems the **actor model** was introduced.

In the actor model, each object is an actor. This is an entity that has a mailbox and a behaviour. Messages can be exchanged between actors, which will be buffered in the mailbox. Upon receiving a message, the behaviour of the actor is executed, upon which the actor can: send a number of messages to other actors, create a number of actors and assume new behaviour for the next message to be received.

Of importance in this model is that all communications are performed asynchronously. This implies that the sender does not wait for a message to be received upon sending it, instead it immediately continues its execution. There are no guarantees about the order in which the messages will be received by the recipient, but they will eventually be delivered. Note that

in Erlang message delivery is not guaranteed, but the order of messages between two processes will be preserved.

A second important property of the actor model is that all communications happen by means of messages: there is no shared state between actors. Each actor runs concurrently with other actors: it can be seen as a small independently running process.

Language elements that support concurrency

- in Erlang, - processes, messages, sending/receiving messages, monitors.
- in Go goroutines, channels and the operators for sending messages.
- For Java mutexes, the synchronized keyword

Distributed systems

For large scalable systems, there is a requirement to handle a high number of cores as well as provide distribution within several machines. The Erlang model is seamless, i.e. there is no difference between a process on one machine or another machine. This is a problem for concurrency models in other programming languages.

Multi-core scalable programming languages (potential competitors for Erlang)

We summarize below the pros and cons of each language with regards to suitability for multi-core, distributed scaling:

- **Erlang** - the language supports all the previously described properties. We have expanded further in section The Erlang language
- **Scala** combines the object-oriented paradigm with the functional paradigm, has a terse syntax compared to Java, is statically typed and is as fast or sometimes even faster than Java. (For more information see the following benchmark) The drawback for Scala is that within each OS thread, event based actors execute sequentially without preemptive scheduling. This makes it possible for an event-based actor to block its OS thread for a long period of time.
  As this is a general purpose, multi-paradigm, object-oriented, functional language with concurrent elements it is very complex and the learning curve is long.

  According to the TIOBE index the popularity of the language is decreasing, from 30th in 2009 to 42th in 2013.

- **Clojure** is very good for concurrency, but missing distribution, which is one of the critical requirements for the RELEASE project.
  The language does not hit the first 50 on the TIOBE index, not in 2009 or in 2013.
- **Haskell** supports concurrency and distribution.
  Cloud Haskell provides most of the required distributed tooling, however, the Cloud Haskell platform is a little bit immature. Haskell has a very strong concurrent core. It

allows the use of Transactional Memory though the STM monad. It provides One-element mailboxes through the MBox primitive. It has lightweight processes like Erlang. It can also utilize additional cores. It has a concept of a "spark" which is an extra light computation (it doesn't have a stack and shares stack with other sparks). This allows for, finer, grained, parallel execution than Erlang since computations can be broken up at a much finer granularity.

This is the purest functional language, which has a long learning curve but is a worthy contender.

Checking the TIOBE index shows that it was more popular in 2009, 30th to 46th in 2013.

- **Go** is a good choice, but it lacks distribution and resilience. It is possible to 'panic' a Go program if a library is not well written. It violates the your-program-can't-crash-my-program theory. There is a panic/recover feature, but it only works on expected faults, not unexpected faults. Also, the go memory model is built such that it doesn't do copying; therefore the embrace-copying feature is missing.
  The language just hit the first 50 on TIOBE index
- **Ocaml** works on a single core only. Currently there is no support for parallel execution, though an experimental concurrency extension exists. JoCaml exists, based upon the Join-calculus. The runtime would need major surgery to support multiple cores. There is no distribution; it is one large shared heap.
- **Node.js** is a very popular joining of the V8 Javascript engine to a simple epoll()/kqueue() loop written in C(++). It was thought to be very scalable, but there are a few problems: Firstly, there is no parallelism. It can only utilize a single core at a time. Secondly, the concurrency model is based upon continuation passing and callbacks. In effect, large complex systems become reminiscent of GOTO-style programming, and there is a considerable amount of literature that points to a "callback hell" due to this concurrency model. Furthermore, the language sports a very naive runtime, has a shared heap space and provides no protection other than mutexes. It is not clear how to get this system to run on 10^5 cores. There are no distribution features, and we're not sure about resilience.
- **Python:** Several implementations exist. The de-facto CPython implementation is one-core-only and there is a GIL (Giant Interpreter Lock) that hampers parallel execution. One shared heap means that the system has no resilience and no distribution either. Using this language will mean a lot of work to achieve the goal. If JYthon or IronPython are used, they bring no added benefit compared to using Scala or C#.
- **Ruby**: Several implementations exist. This language has the same problems as with Python. Backends are often written in "shared-nothing" style. The basis for this style is that you run a large amount of processes in a UNIX-like system. Intercommunication between the processes would be quite expensive, but in a shared-nothing architecture there is no sharing at all. Each process acts independently. When a Ruby process runs, it usually only accepts a single request, and it blocks, without handling other requests in the meantime. Huge numbers of

processes are needed to utilize a multicore machine, which results in context switch overhead, and wasted memory resource usage, since many of the processes could be using the same resources.

To be able to compare the languages the following table summarize the 4 properties according to the below definition:
C: concurrency — Are the primitives strong enough for our want?
P: Parallelism — Do we need to alter the implementations to utilize multiple cores?
R: Resilience — Can we isolate error correctly? Would it be possible to isolate hardware failure?
D: Distribution — Is it built-in, or do we have to roll-our-own?

Language	**Properties**	**Comment**
Erlang	CPRD	
Scala	CP	due to the Akka framework, ScalaRx
Clojure	CPR	
Haskell	CP(R)D	There are discussions around Haskell resilience
Go	CP	
Ocaml	C (R)	Oacml does not have resilience in the way we want
Node.js	-	Nothing at all we want
Python	-	Nothing at all we want
Ruby	-	Nothing at all we want

Popular languages which do not scale on multi core

- Java, C++: Are using mutexes and locks as programming model, which has problems:
    - It's often leads to deadlock:
        - To maintain any semblance of order, concurrent Java programs use threads that lock pieces of data when a part of the program (let's say Method A) needs access. If another part of the program (Method B) needs access to that same piece of data, Method B is locked out and needs to wait for Method A to finish and unlock the data. The more processors are thrown at a Java program, the more often these data collisions happen, and will eventually hit a point where the whole JVM is simply bogged down in the process of managing and manipulating locks.
    - Locks does not distribute over multiple machines easily
    - Sharing data in the programming model required and it was ruled out as a viable way since the preferable is to copy data around
    - None of the usual High-performance-computing models are using locks, they standardized on message passing (MPI).
    - There is no built-in resilience.

The Erlang language

Erlang has been available as Open Source technology since 1998, and its popularity is growing at an exponential rate. According to the TIOBE September 2013 Programming Community Index, Erlang is among the 30 most popular programming languages, up ten positions from 2009, when it was the 39th most popular language. The Redmonk index is showing similar trends.

Erlang is nowadays represented at most mainstream software developer conferences, and demand for Erlang programmers is on a rising trend. Two strong market and technology trends speak in Erlang's favour:

1. E-commerce, New Media, Messaging and Social Networking have matured to the point where it is no longer enough to present a flashy web page to attract customers; one has to address scalability, response time and fault tolerance in order to get appreciable market share. To this end, leading sites and infrastructure providers like Klarna, Adtech, WhatsApp, Rackspace, EngineYard, Github and Heroku rely on Erlang for important components that allow their infrastructure to scale.
2. Hardware vendors have reached the end of the road on increasing clock speeds on single-core CPUs, and are now focusing on multi-core architectures. This forces programmers to embrace concurrency in order to evenly sustain their performance (as the number of cores increases, each individual core tends to become less powerful). Erlang is one of the few languages with strong multicore scalability.
3. The scalability problems of companies such as Google, Ebay, Yahoo, Facebook and Amazon today affect not just a few, but all new companies entering the market. Anyone who wants to aspire at becoming the next Facebook needs to consider scalability issues from the start. Erlang is one of the very few programming languages that have built in distribution semantics that allows systems to scale horizontally by adding commodity hardware.

A background trend is that languages based on functional programming concepts are enjoying increasing attention, as these concepts appear to hold up very well in the face of multi-core scalability. Elixir is a new language built on top of the Erlang Virtual Machine that is quickly gaining attraction. It provides a new means to program using the Erlang model whilst retaining compatibility with all of the tools and libraries.

Results of the project

As a result of the RELASE project language extension (SD Erlang), improvements to the Erlang VM, tools and case studies are created.

4.1.1 **SD Erlang**

Scalable Distributed (SD) Erlang, is an extension of distributed Erlang functional programming language, that aims to enable Erlang server applications to scale on commodity hardware with at most 10^5 cores. The reason SD Erlang has been introduced because it is not feasible for a node to maintain connections to tens of thousands of nodes, when using the distributed Erlang scheme of transitive connections. SD Erlang reduces the number of node connections and the number of nodes in a namespace, by introducing s_groups, i.e. nodes in an s_group have transitive connections only with nodes from the

same s_groups, but non-transitive connections with other nodes. Each s_group has its own name space, i.e. names registered in an s_group are replicated only to the nodes from the same s_group.

SD Erlang was implemented by introducing new s_group.erl module in lib/kernel/src and by modifying corresponding functions and tables in such modules as global.erl, kernel.erl, and net_kernel.erl. Erlang nodes can join s_groups at launch and dynamically. Other s_group functions include adding and removing nodes from s_groups and deleting s_groups. In SD Erlang free nodes, i.e. the nodes that belong to no s_group, behave identically to distributed Erlang nodes.

The initial experiments that compare performances of distributed Erlang and SD Erlang using DEbench benchmarking tool show that SD Erlang provides a higher throughput. The details of SD Erlang and its implementation can be found in deliverables and publications of WP3. The source code can be found in https://github.com/natalia-chechina/otp/tree/sd_erlang.

The SD Erlang name is used only as a convenient means of identifying the extensions we propose: it is expected the extensions to become standard Erlang in the future.

4.1.2 **Percept2 + Visualization text for RELEASE**

Percept2 (https://github.com/RefactoringTools/percept2) and it is associated visualization tools (https://github.com/RefactoringTools/percept2_online) provide facilities for offline and online monitoring of distributed and SD Erlang systems running on multicore processors.

Percept2 is built on the foundation of Percept (http://www.erlang.org/doc/man/percept.html), the principal existing Erlang concurrency-profiling tool, which is distributed as a part of the Erlang/OTP main distribution. Percept2 extends Percept by providing the following facilities:

- Process migration between run queues: history and number
- Messages sent intra- and inter-core.
- Average size of messages sent/received.
- Structured presentation of process hierarchy.
- Dynamic callpath/count/time generation.
- Structured display of callpath/count/time information.
- Report of function activities during a time interval selected.
- Process communication graph.
- Parallel analysis of trace data.
- Support for sampling-based profiling.
- Selective function profiling of processes.
- User-command interface improved to allow the profiling of a particular aspect of the execution.

.

Percept2 and the visualization tools are available as open source systems on github. This is done in order to ensure the long-term sustainability of the tool during and beyond the life of the RELEASE project through attracting community support and contributions for the

tooling. This ensures that Percept2 and the visualization tools will be available in the long term.

The tools are written in Erlang. Percept2 is built using Erlang plus the DTrace / SystemTap system available for unix (Mac OS X, linux) systems. The visualization and visual aspects of Percept2 are browser-based, using HTML5 and JavaScript libraries, thus ensuring that there is no long-term dependency on any platform-specific framework software.

These tools are usable for general distributed Erlang programs, but have also been customized to work with SD Erlang. The DTrace-based probes only support systems running on the unix platform, but this represents the majority of existing Erlang systems. There are no substantial barriers to adoption beyond the linux dependency of DTrace-based probing; note that Percept2 supports *all* platforms through its support of built-in Erlang tracing.

From the inception of parallel programming, the availability of tools that ease the effective deployment of programs on parallel platforms has been a crucial criterion for success. One of the more effective mechanisms for such tools, is to present a visualization of characteristics of the parallel execution of programs, whether post-hoc (offline) or in real time (online). Visualizations continue to be supplied as a part of programming toolkits for multicore development. A typical example is the Intel Trace Analyzer and Collector [ITAC], which support the analysis, optimization and deployment of applications on Intel processor-based clusters with MPI communication.

The Erlang Virtual Machine is equipped with a comprehensive, low-level tracing infrastructure provided by the trace built-in functions. Built upon this are a number of higher-level facilities, including the debugger (dbg) and the trace-tool builder (TTB), which provides facilities for managing profiling, tracing across a set of distributed Erlang nodes. As the name suggests, the tool is designed to be extensible to provide different types of profiling, tuned to different applications and environments. Percept is also a part of this.

The BUBBA project (https://github.com/duomark/bubba) appears to have used a similar browser-based approach to visualization, but the project had been inactive since 2012.

4.1.3 **Wombat**

Wombat's foundation is a set of tools and libraries which aims at providing help in the development, deployment, monitoring and scalability of Erlang based systems. It makes accessing, controlling and scaling large numbers of nodes easier, so that developers can focus on getting their job done without worrying about patching, deployment or monitoring, and without the need to reinvent the wheel and re-implementing generic functionality within projects. It can be broken down into three independent parts that can be combined to interact with each other or used individually. The three parts consist of:

- A **development stack**, or distro, which alongside Erlang/OTP, contains tools, applications and libraries covering generic functionality needed by most Erlang Systems. Developers using the standard applications from the distro will provide hooks that facilitate monitoring and operations of the final product they are developing.
- **Wombat OAM** handles monitoring and management of Erlang systems through a web based GUI and/or by providing standard interfaces to other OAM tools and services such as (**Graphite, Cacti, or SaaS**) offerings from (**New Relic and Boundary**). Wombat will interface towards Erlang systems (and where present, standard applications from the distro) without the system having to be aware of Wombat's existence.
- **Wombat Orchestration** deploys Erlang systems in heterogeneous public and private cloud, low powered clusters and private servers, allowing the dynamic scaling and shrinking of the system based on preconfigured triggers. Wombat orchestration can interface towards tools such as (Chef, Puppet or ssh).

All tools that offer Wombat's functionality currently are not easy to use with Erlang based systems, making their integration, cumbersome. Wombat will work out of the box and can, (without any changes), be integrated with systems that are already deployed and operational. This increases the current market segment of green field projects, of which Erlang Solutions helps out with half a dozen a year, to the tens of thousands of installations of Erlang based systems.

Pivotal Labs estimates that there exist about 50,000 installations of RabbitMQ[1], an Erlang based open source AMQP interface. In addition, installations of Riak , CouchDB, Ejabberd, MongooseIM and Disco, together account for the majority of Erlang deployments, catering for a market worth millions. If one is to expand outside the Erlang world, the potential is even larger.

The strategy is to first get the Wombat tool to work well in the market segment we know well. The steps to follow this strategy are the following:

- Demo the tool for customers
- Provide access to the tool free of charge for already existing customers and collect feedback
- Use the tool within the project for deploying the case study applications

Once that goal has been achieved, we can address other markets outside Erlang.

4.1.4 **Sim-Diasca**

Sim-Diasca (http://www.sim-diasca.com) is a discrete-time generic simulation engine, implemented in Erlang. It is designed to preserve key model-level evaluation properties while targeting the simulation of very large-scale systems (millions of complex models). It does so by aiming for maximum concurrency, with a mode of operation that is both parallel and distributed.

[1] This comes from Alexis Richardson, one of VMWare's directors of Technology, currently director at Pivotal.

Sim-Diasca (which stands for Simulation of Discrete Systems of All Scales) is developed within EDF since 2007 and has been released as free software (LGPL licence) since 2010.

It has already been used for in-house purposes and also in a partnership project (http://www.cleveronline.org/).

As a scalability-focused tool, various benefits for Sim-Diasca are expected to stem from the RELEASE project, including:

- More knowledge and know-how to make the best use of the language and of its associated tools (to check, benchmark, refactor, tune, etc. our engine)

- The use of the evolutions and extensions of the language (SD-Erlang, s-groups, etc.) to target simulations of higher scales

- The port of the Erlang VM to the Bluegene/Q supercomputer to reach effectively these intended scales

All corresponding tools expected to be released as free software.

Sim-Diasca is written in Erlang. Through the project it will be made more compliant to the OTP principles (notably transformed into a standard release).

Sim-Diasca is one of the very few discrete-time simulation engines that has been designed for scalability.

The engine may be applied to the simulation of most complex systems, and is by nature transverse to many application sectors (from biology to engineering).

We are not aware of any significant competitor in terms of pure, generic, domain-agnostic simulation engines. Most are dedicated to specific themes (ex: to simulate telecom networks, business processes, etc., like OMNeT++) - which induces strong challenges whenever needing to couple domains, or are more integrated tool chains for modeling and simulation - with generally severe limitations in terms of supported scale (ex: AnyLogic).

Being solely free software, the notion of market share hardly applies, however Sim-Diasca is used inside and outside of EDF, in various domains, and, despite the low-volume advertisement that was made, has already be released to more than 30 persons and organizations.

4.1.5 **Benchmarking tools**

DEbench is a benchmarking tool based on Basho Bench tool. DEbench is used to measure distributed Erlang scalability and to compare performance and scalability of distributed Erlang and SD Erlang. DEbench is available in github https://github.com/amirghaffari/DEbench.

BenchErl is a publicly available scalability benchmark suite for applications written in Erlang. In contrast to other benchmark suites, which are usually designed to report a particular performance point, our benchmark suite aims to assess scalability, i.e., a set of performance points that show how an application's performance changes when additional resources (e.g. CPU cores, schedulers, etc.) are added.

The features included in BenchErl allow the execution of applications in various execution environments, the visualization of the results, and the extraction of useful conclusions. Hence, it is a tool that might help the Erlang community make a first step to better understand the parameters that affect the parallel execution of Erlang applications.

Key features of the tool:

- **Unique**: BenchErl is the only benchmark suite that targets the scalability of Erlang applications
- **Configurable**: BenchErl allows the configuration of a large number of parameters that might affect the execution of a benchmark. BenchErl handles the execution of the benchmark with all possible combinations of these parameters.
- **Automated**: BenchErl handles the collection, the execution and the visual presentation of the benchmark execution results.
- **Extendable**: It is straightforward to add new benchmarks and applications to BenchErl.

5 Exploitation initiatives

5.1 Wombat

5.1.1 **Product development strategy**

A lean approach to developing Wombat will be taken. In a first iteration, the Wombat minimal viable product should be able to manage the topology of the system, including self-discovery of nodes. It should handle metrics, alarms and events, and provide a configurable architecture allowing other users to implement their own plug-ins. Wombat will consist of one node running both the master and the OAM applications.

A web-based console will provide a visualization of node clusters, basic alarming, metrics and logging. It will provide a plug-ins for:

- Folsom – the metrics application,
- Lager – the logging application,
- an interface into New Relic - a third party tool, for application monitoring.
- an interface into Graphite - the metrics visualization and database tool.

Once the minimal viable product providing the necessary infrastructure is launched, additional features should be developed based on customer requirements and requests resulting from the live trials.

5.1.2 **Actions to get Wombat used**

- Based on interviews with prospective clients and market research, there is a need for Wombat, as it generically solves problems programmers have to tackle in between projects.

- Wombat will be made available to all customers purchasing business and enterprise support packages of Erlang/OTP from Erlang Solutions. In addition to support of the Erlang VM and OTP applications, Users will get access to Erlang Training, Erlang Conference tickets and a number of hours of Consultative Support. Therefore, whilst Wombat is under development, the plan is to get End Users to purchase the product and influence it's development, and the products will need to be packaged into services and products that customers need. The commercial benefit for Erlang Solutions is understood to be in the sale of additional Support and Training packages.

Wombat will have an open plug-in architecture, allowing developers to implement their own tools. This will allow developers to implement new tools that they can either integrate with wombat or RELEASE as open source. Open source tools can be run on any framework, such as those provided by performance labs.

- In a second phase, Wombat SaaS will be launched. The software as a service offering will have three tiers, of which the simplest one will be free of charge. This will allow Users to test Wombat without having to buy or install it. Other tiers will provide more tools and plug-ins, better visibility and store metrics and logs for longer period of time.

5.2 Other areas

- **VM improvements**

The OTP/Erlang VM improvements we have been working with will all be included in the Open Source project Erlang/OTP. They are all aiming for improving the multicore scalability and performance when the number of cores increases.

The improvements are necessary for keeping Erlang in a top position when it comes to multicore utilization (combined with productivity and robustness).

All of this will make Erlang even more attractive and will grow the community around Erlang. It will result in a broader usage of Erlang in commercial products and in research projects.

- **SD Erlang & Refactoring for API**

SD Erlang is available in github

- https://github.com/natalia-chechina/otp/tree/sd_erlang.

The language extension was also promoted on a number of conference and seminar talks given by WP3 team and by posters.

- **Percept2**

The tools are available on github

- https://github.com/RefactoringTools/percept2
- https://github.com/RefactoringTools/percept2_online

under an Open Source BSD-style license. They have also been reported in a series of conference papers and presentations in both academic (Erlang Workshop, Visual Languages and Computing) and industry-facing venues (Erlang Factory, Erlang User Conference).

- **Bencherl**

The source code is freely available in github:

- https://github.com/softlab-ntua/bencherl

Demo of the tool was presented on several conferences.

- **Sim-Diasca**

The official website (http://www.sim-diasca.com) allows to request access to the sources of the tool.

Sim-Diasca has also been reported in a few presentations in both academic and industry-facing venues, even if it is not a key priority for that tool.

6 SWOT Analysis

Strengths

- The RELEASE consortium has developed language extension (SD Erlang) that will enable scaling the reliability model to preserve Erlang's sophisticated and effective reliability mechanisms, of first class processes and supervision behaviours, in the presence of locality and connectivity controls.
- Complimentary product offerings are available, such as support, training material, scheduled courses, online tutorials, webcasts, screencasts and videos.
- Within the Erlang community, Erlang Solutions is the industry leader with all the sales channels in place to commercialize and disseminate all the tools created by the RELEASE project.
- The Wombat - OAM and Orchestration tool can scale from small clusters to systems with tens of thousands of nodes. No similar products for Erlang exist today.
- There are no other tools and libraries available that enhance programmer productivity in the areas of operations administration and maintenance among Erlang Users.
- Orchestration and dynamic scalability, often comes as an afterthought when designing a system. Using Wombat, provides systems developers these features out of the box, allowing them to focus on the business logic which will automatically scale.

- Distribution and Scalability on multi-core architectures go hand in hand. Wombat bridges the link between the deployment and monitoring of multi-core systems in distributed architectures.
- The DevOps team has had a defacto approach to doing things in Erlang for a long time. The tools the consortium has developed will strengthen this approach, resulting in lower operation costs and more stable systems.
- Benchmarking and refactoring tools will improve the development phase of design activities.
- Sim-Diasca case study will demonstrate the benefits of using SD Erlang and Wombat for deployment. Other customers can use the case study as an example.
- Erlang VM improvements will speed up ETS table handling; newly introduced thread progress functionality allow implementing lock-free algorithm, which will improve the performance of concurrent systems.

Weaknesses

- Only Erlang Solutions has a business case to sell tools and support of the tools provided by the consortium. By providing an open plug-in architecture, tools can be integrated and disseminated through other tools that support this architecture, not limiting the dissemination to paying customers.
- Releasing tools as open source will help make this research widely known and increase the Erlang/OTP User base. As non-Consortium members are interested in capitalizing on the results, this still helps increase the market for the companies who are in the Consortium.
- The market for Erlang development projects is small. RELEASE however helps increase this market. By first getting Wombat right for Erlang based systems, it can be expanded to deploy, manage and monitor systems written in other languages.
- Tools are weakly integrated. The open plug-in architecture will hopefully address this, allowing both community and commercial adoption of the tools.

Opportunities

- Development on multi-core requires a mind shift. There is a growing need for the tool set developed by the RELEASE partners to support this mind shift.
- There is a growing opportunity to develop new lines of business around these tools, including training, support and consulting.
- The results of the RELEASE consortium can be used to enter the non-Erlang market, specifically for managing and deploying systems in hybrid architectures.
- Companies are interested in monitoring tools, since they need to monitor their topologies.
- Shifting mindset towards DevOps (not sure if it's opportunity or threat)

- When these tools have been proven in the Erlang community, there is an opportunity to start making them compatible with other programming languages and development platforms, increasing the size of the market.

Threats

- It is possible that existing open source tools used for deployment and monitoring of clusters, will become as generic, as the tools developed by the consortium. Ensuring the capability, to monetize existing work, will provide funds for future development, ensuring that what has been developed remains ahead of the curve.
- The market for Erlang tools and applications is relatively small. This market can be increased in size by growing the User base and ensuring that the tools we develop can be used across several programming languages.
- Adoption of new tools is slow in the Erlang community. Very few tools are used, even tools that ships with Erlang/OTP are not used much. Actively disseminating the tools and ensuring that they are easy to use can reduce this threat.
- Because Erlang usage is not widespread, there is a shortage of trained and experienced developers. This can be counteracted, by increasing dissemination activities, ensuring that there are adequate online resources, and by working with the community, to grow the User base.

7 Conclusion

This deliverable presents the exploitation plan of the RELEASE project.

It analyzed the RELEASE project business aspects, including the markets to be addressed through the RELEASE solutions. Furthermore, it focused on the potential for the competitive advantages, Strengths, Weakness, Opportunities and Threats.

Exploitation strategy and steps as well as partner collaborations are identified, and will be further followed up within and beyond the projects life time.

8 Change Log

1. First version.

- Sent for internal review 27.09.2013
- Updated and sent for re-review 10.10.2013
- Updated and submitted 14.10.2013