RELEASE

SEVENTH FRAMEWORK PROGRAMME

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

D6.4 (WP6): Homogeneous Deployment of a Load Testing Tool

Due date of deliverable: 31st May 2013
Actual submission date: 31st May 2013

Start date of project: 1st October 2011 Duration: 36 months

Lead contractor: Erlang Solutions Ltd. Revision: 0.2

Purpose: To validate that the prototype developed in WP4 is capable of deploying a real-world Erlang application on a homogeneous virtual infrastructure.

Results: The main results of the deliverable are as follows:

- The development of the WP4 prototype has been aligned with the requirements of a real-world distributed Erlang application.
- It has been validated that the developed prototype is capable of deploying a real-world distributed Erlang application on a homogeneous virtual infrastructure.

Conclusion: The developed prototype, WombatOAM, has been used to successfully deploy a real-world Erlang application, Megaload, on a homogenous virtual infrastructure, Amazon EC2. Upcoming deliverables will build upon the results presented in this deliverable and will focus on demonstrating the ability of the developed prototype not only to deploy applications on a heterogeneous virtual infrastructure, but also to carry out a deployment driven by matching deployment requirements with existing cluster capabilities.

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)	

Homogeneous Deployment of a Load Testing Tool

Enrique Fernandez <enrique.fernandez@erlang-solutions.com>
Csaba Hoch <csaba.hoch@erlang-solutions.com>
Roberto Aloi <roberto.aloi@erlang-solutions.com>
Torben Hoffmann <torben.hoffmann@erlang-solutions.com>

Contents

1 Introduction **2**

2 Introducing Megaload **2**

3 Requirements **3**
3.1 Connectivity . . . 3
3.2 Clustering . . . 4
3.3 Customisation of the Hosting Machines . . . 4
3.4 Execution of Custom Commands . . . 4

4 Deploying Megaload: A Walkthrough **5**
4.1 Specifying node name dependant configuration files . . . 5
4.2 Specifying the commands to start and stop Megaload nodes . . . 5
4.3 Distributed Erlang cookie sharing . . . 6
4.4 Entering infrastructure provider information . . . 6
4.5 Specifying software and hardware characteristics of the host machines . . . 6
4.6 Entering firewall configuration details . . . 7
4.7 Entering clustering information . . . 8
4.8 Specifying allowed Megaload-specific commands . . . 8
4.9 Packaging the WombatOAM-compliant Megaload release . . . 9
4.10 Megaload tarball upload . . . 9
4.11 Creating and Starting Megaload nodes . . . 10
4.12 Interacting with Megaload . . . 10

5 Conclusions and Future Work **11**

6 Change Log **11**

Abstract

This deliverable illustrates how WombatOAM — the prototype being developed in the *Scalable Virtualisation Infrastructure* work package (WP4) — has been used to successfully deploy Megaload — the load testing tool being developed within the context of the PROWESS project — on a homogeneous virtual infrastructure. Thus, validating the work done in WP4. Upcoming deliverables will build upon the results achieved in this deliverable and will focus on demonstrating the ability of the developed prototype not only to deploy applications on a virtual infrastructure where multiple providers are involved, but also to carry out a deployment driven by matching deployment requirements with the capabilities of available clusters.

1 Introduction

This deliverable is part of the *Case Studies* work package (WP6) of the RELEASE [REL11] project. According to the *Description of Work* of the project, the purpose of this work package is *"to demonstrate and validate the tools and methodologies developed within the context of the project"* . This is what this deliverable is all about: demonstrating that WombatOAM — the prototype being developed as part of the *Scalable Virtualisation Infrastructure* work package (WP4) — is capable of deploying a real-world distributed Erlang application on a homogeneous virtual infrastructure. To this end, this document presents the experiences gained throughout the deployment of Megaload — a load testing tool that is being developed within the context of the PROWESS [PRO12] project —, on a single Infrastructure-as-a-Service provider, which has been set to be Amazon EC2 [Ama13b].

The structure of this document is as follows: **Section 2** introduces Megaload, the distributed Erlang application of choice for validating the developed prototype. In **Section 3**, the requirements Megaload has set on WombatOAM are presented. A step-by-step example illustrating how WombatOAM has been used to deploy Megaload is presented in **Section 4**. Finally, the conclusions and future work are drawn in **Section 5**.

2 Introducing Megaload

Megaload is a scalable, high-performance load testing tool that can be deployed on a Cloud environment. Megaload benefits from the Cloud to spawn multiple instances and build a cluster able to generate a massive load to stress a target system.

Megaload instances communicate to one another through their main controllers. In order to get the best performance out of each instance, Megaload balances the load among all the spawned instances. The multiple Megaload instances are coordinated ensuring that the test specification is always met.

A Megaload test scenario defines the functional scenario and traffic profile that the tool must simulate. In Megaload terms, a test consists of:

- General configuration options.
- Lists of plugins to be used.
- Many phases specifying different load conditions.
- Many scenarios per phase. Each scenario can be seen as a list of actions to be performed.

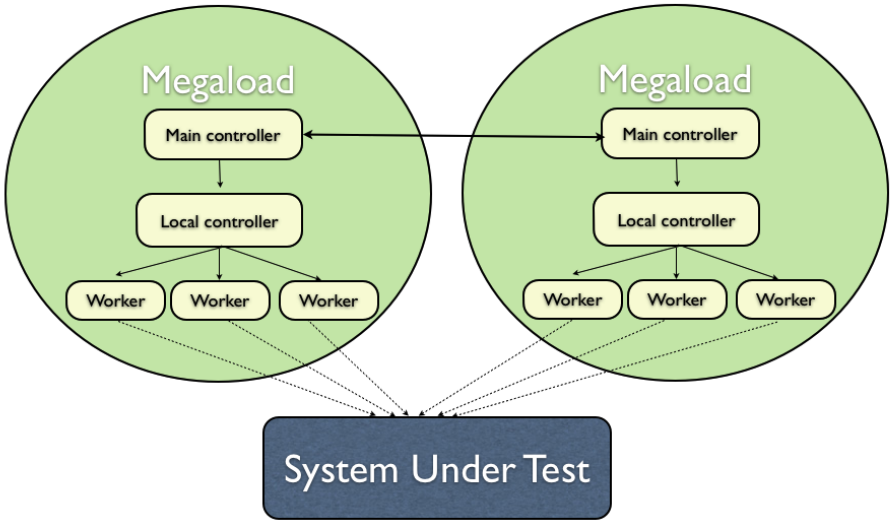

Figure 1: Megaload Overview

3 Requirements

The goal of this deliverable is not only to validate the developed prototype, but also to mature the prototype in the right direction, ensuring that the requirements from real-world applications are met. This section describes the requirements that Megaload — the validating application — has set on WombatOAM — the developed prototype —, and how these requirements have been addressed.

3.1 Connectivity

Deploying an application and not being able to reach it is equivalent to not having deployed it. It is important to bear in mind that end-users are not the only ones who must be able to reach a given node [Eri13a] of a deployed distributed application, but also nodes belonging to the same cluster must be able to talk to one another — throughout this document the word node is used to refer to a running Erlang runtime system (ERTS) [Eri13f] executing one or more Erlang applications [Eri13b] —. Last but not least, WombatOAM also needs to be able connect to the machines responsible for hosting the nodes of the distributed application, so it can garnish them with the required software (including the to-be-deployed Erlang application itself).

In other words, the nodes are running on machines of IaaS (Infrastructure-as-a-Service) providers, e.g. Amazon EC2 or in-house OpenStack. For security reasons, these providers use firewalls to block most of the traffic to and from the Internet. Therefore the firewall needs to be configured to enable the traffic between the nodes and the end-users, between the nodes and WombatOAM, and between the nodes themselves.

In order to address this issue, WombatOAM provides end-users the ability to specify the firewall configuration that will manage the traffic to and from the cluster. Depending upon the nature of the traffic, required firewall rules can be classified as:

1. **Provisioning rules:** Configuration required in order to provision the host machines with the required software, including the Erlang application that is meant to run on those machines.

2. **Application-specific rules:** All other firewall rules required by the application enabling its nodes to be reachable not only by end-users, but also by the other nodes forming the cluster.

3.2 Clustering

It is safe to say that different distributed applications require different clustering strategies. For some applications, choosing a random bootstrap node and using the functions available in the *net_adm* [Eri13d] and *net_kernel* [Eri13e] modules may be enough. However, some other applications may require more complex clustering strategies, the logic of which we expect to be enclosed in a boot script that takes one (or more) bootstrap nodes as input.

In order to deal with this scenario, the developed prototype provides end-users the ability to specify a clustering strategy for the to-be-deployed application. More precisely, WombatOAM allows users to specify:

1. **Bootstrap node selection strategy:** Specifies the strategy to follow in order to select a running node from within the cluster. The selected node will be used by the starting nodes to join the cluster.

2. **Bootstrapping strategy:** Specifies how a starting node interacts with one or more bootstrap nodes in order to join the cluster.

3.3 Customisation of the Hosting Machines

When deploying a host machine on a Cloud environment, users are usually asked to provide details about the operating system and hardware characteristics they want for the to-be-provisioned machine. This is needed because not all applications running on the Cloud have the same software and hardware requirements. Whereas some data-intensive applications may benefit from running on virtual machines that have been allocated reasonably high amounts of RAM, other computationally-intensive applications may benefit from running on virtual machines that have been allocated multiple compute cores instead.

WombatOAM does not limit end-users to deploy their applications on standard pre-provisioned host machines that may not meet the requirements of their applications, but provides them the ability to specify the concrete hardware and operating system requirements of their applications.

3.4 Execution of Custom Commands

Deploying an application on the Cloud may result being totally useless if no mechanism for interacting with the deployed application is provided. Moreover, an end-user should not need to know where the nodes of his distributed Erlang application have been deployed in order to be able to start using them. To this end, the developed prototype provides both: 1. a way to specify the Erlang commands allowed to be remotely executed (expressed in the typical Erlang *Module Function Arguments (MFA)* form), and 2. a mechanism enabling end-users to perform the remote execution of those commands. WombatOAM prevents users to execute un-authorised commands by matching all incoming requests with the lists of allowed commands specified when building the WombatOAM-compliant release of the Erlang application to be deployed.

4 Deploying Megaload: A Walkthrough

This section walks the reader through the process of using the prototype developed in WP4 to deploy a real-world distributed Erlang application on a homogeneous virtual infrastructure. To this end, a step-by-step example is provided to illustrate how WombatOAM has been used to deploy Megaload on Amazon EC2.

Please note that the present example assumes the following statements to be true:

- The user holds a valid Amazon EC2 account.
- The user has access to an OTP-compliant release of Megaload.
- WombatOAM is up and running.
- A Deltacloud service has been properly started in the background, so that it can talk to Amazon EC2 backend (deltacloudd -i ec2).
- WombatOAM has access to a pair of private and public keys. These keys are used to enable the access to the machines that will be hosting the Megaload application.

4.1 Specifying node name dependant configuration files

WombatOAM uses the same Erlang release tarball for all the Erlang nodes deployed within the same cluster. This means that the node name cannot be hardcoded in the configuration file. Otherwise, there would be no way to identify which node is which. To address this issue, WombatOAM uses the dynamically generated node names to automatically update all configuration files depending on them. This requires all the configuration files depending on node names to be turned into templates that can be automatically updated by WombatOAM when deploying a new node (an example of how to avoid hardcoded node names is given in Listing 1). These templates must be specified in the *oam.config* configuration file using the **node_name_templates** configuration flag. Listing 2 shows how this have been done when deploying Megaload.

Listing 1: Turning a configuration file (megaload/etc/vm.args) into a template file

```
...
-name {{ node_name }}
...
```

Listing 2: Entering node name dependant configuration files in oam.config

```
{node_name_templates, ["etc/vm.args"]}.
```

4.2 Specifying the commands to start and stop Megaload nodes

The next step is to let WombatOAM know how to start and stop a Megaload node. WombatOAM expects the user to specify how to start and stop his application by setting the **start_cmds** and **stop_cmds** configuration flags in the *oam.config* file.

Listing 3 shows how Megaload start and stop commands look like. Note that multiple commands could have been specified. In that case, WombatOAM would execute those in the same oder they are listed in *oam.config*.

Listing 3: Specifying the start and stop commands in oam.config

```
{start_cmds, ["megaload/bin/megaload start"]}.
{stop_cmds, ["megaload/bin/megaload stop"]}.
```

4.3 Distributed Erlang cookie sharing

WombatOAM relies on the Erlang distribution protocol for interacting with the deployed Erlang nodes (e.g. starting and stopping nodes, executing custom commands, ...). In order to be able to do this, WombatOAM needs to share a distributed Erlang cookie with the deployed nodes. The way to let WombatOAM about the cookie being used by the deployed Erlang nodes is to set the **distributed_erlang_cookie** configuration parameter in *oam.config*, as shown in Listing 4.

Listing 4: Setting the distributed Erlang cookie in oam.config

```
{distributed_erlang_cookie, megaload}.
```

4.4 Entering infrastructure provider information

The user must specify where Megaload nodes will be deployed on. This is done by editing the *oam.config* configuration file as shown in Listing 5. Since we are deploying Megaload on Amazon EC2 — a virtual infrastructure provider —, the **provider_type** configuration flag needs to be set to *virtual*. The details (username and password) of a valid Amazon EC2 account are set using the **provider_username** and **provider_password** configuration flags, respectively. Finally, information about where to reach the Deltacloud service interfacing with Amazon EC2 is set using the **provider_endpoint** parameter.

Listing 5: Entering provider information in oam.config

```
{provider_type, virtual}.
{provider_username, <AMAZON_EC2_ACCESS_KEY>}.
{provider_password, <AMAZON_EC2_SECRET_KEY>}.
{provider_endpoint, <DELTACLOUD_SERVICE_URL>}.
```

4.5 Specifying software and hardware characteristics of the host machines

One more thing that is needed is to specify the hardware and operating system requirements of the machines that will be hosting Megaload. This can be easily done by setting the **hardware_profile** and **virtual_disk** configuration variables. Since we are deploying on Amazon EC2, the simplest thing to do is go to [Ama13a] and [Ama13c] and choose the most suitable instance type (hardware_profile) and AMI (virtual_disk), respectively. Listing 6 shows the ones we have chosen when deploying Megaload, which correspond to a 64 bits Ubuntu virtual disk and a hardware profile consisting of 3.75GB of RAM, 410GB of storage and 1 virtual core.

Listing 6: Entering the requirements of the host machines in oam.config

```
{hardware_profile, "m1.medium"}.
{virtual_disk, "ami-0cdf4965"}.
```

4.6 Entering firewall configuration details

WombatOAM relies on the SSH protocol for the deployment of an Erlang application on a target host machine. This means that TCP port 22 needs to be enabled in the firewall managing the traffic to and from the cluster. Besides this, all the communication between WombatOAM and the deployed Erlang nodes (e.g. starting/stopping nodes, executing custom commands, ...) happens atop the Erlang distribution protocol. Thus, TCP port 4369 and the ports used for distributed Erlang traffic must also be enabled.

By default, Erlang does not specify any range to be used for distributed Erlang traffic. Instead, Erlang uses the first available port in the system. In order to simplify things, WombatOAM assumes the distributed Erlang application to explicitly narrow the ports used for distributed Erlang traffic using the **inet_dist_listen_min** and **inet_dist_listen_max** environment variables of the *kernel* [Eri13c] Erlang application. To get this addressed in Megaload, we have modified its configuration file as follows.

Listing 7: Setting ports used for distributed Erlang traffic in etc/app.config

```
{kernel, [
  {inet_dist_listen_min, 9100},
  {inet_dist_listen_min, 9105}
]}
```

Since Megaload relies on the distributed Erlang protocol for all the communication across nodes, there is no need to specify any other application-specific firewall rules.

Listing 8: Entering firewall configuration details in oam.config

```
{firewall_name, "megaload"}.
{firewall_rules,
  [
    [ %% SSH
      {protocol, "tcp"},
      {ports, {"22", "22"}},
      {sources,
        [
          {address, "0.0.0.0/0"}
        ]
      }
    ],
    [ %% EPMD
      {protocol, "tcp"},
      {ports, {"4369", "4369"}},
      {sources,
        [
          {address, "0.0.0.0/0"}
        ]
      }
    ],
    [ %% distributed Erlang traffic
      {protocol, "tcp"},
      {ports, {"9100", "9105"}},
      {sources,
        [
          {address, "0.0.0.0/0"}
        ]
      }
    ]
  ]
}.
```

4.7 Entering clustering information

Megaload relies on a master-slave architecture with one master node per cluster. It uses the gen_leader [Gar13] library to globally select a leader (the master node) among all the nodes in the cluster. If the leader node happens to die, a new one is elected.

In this architecture, a master node is used to keep the test phase synchronised across all nodes present in the cluster. This node is also responsible for monitoring all slave nodes and rebalance the cluster when needed (e.g. slave nodes are under/over-loaded). Rebalancing the cluster may result in spawning one or more nodes if the cluster is overloaded. Thus, a mechanism to dynamically add new nodes into an existing Megaload cluster is needed.

The gen_leader library version used in Megaload has the singularity that supports the dynamic addition of nodes into the candidates set — list of nodes that can eventually become the leader node of the cluster — (previous versions only supported a static list of candidate nodes). Dynamically added nodes must know about all other nodes in the cluster before starting the gen_leader. This can be achieved by connecting to the Erlang cluster using the functions available in either the net_kernel or net_adm modules. Once joined the Erlang cluster, the result of calling *erlang:nodes/0* can be used as input to start the *gen_leader* process using either *gen_leader:start_link/6* or *gen_leader:start/6*.

The way WombatOAM can be used to express this behaviour is as follows. Since there is no strict requirement on what node should be selected as a bootstrap node, selecting a random running node from the cluster suffices. The details about how the joining node uses the selected bootstrap node to connect to the Erlang cluster and how the gen_leader process is started must be enclosed in a boot script that will be called right after the joining node is started. Listing 9 illustrates how this can be expressed in the *oam.config* file.

Listing 9: Specifying enabled custom commands into oam.config

```
{bootstrap_node_selection_strategy, random}.
{bootstrap_strategy, custom}.
{bootstrap_opts, [{cmd, "megaload/bin/bootstrap"}]}.
```

4.8 Specifying allowed Megaload-specific commands

As mentioned earlier in this document, one of the requirements Megaload has set on WombatOAM is to provide users the ability to specify the list of application-specification commands they will be able to execute using the WombatOAM REST API (that is, without having to manually connect to the nodes where the Megaload nodes have been deployed). When specifying an application-specific command, the user is expected to enter it in the {*CmdName, M, F, A = [*{*ArgName, ArgType, ArgDescription*}*, ...], CmdDescription*} format. Listing 10 illustrates how these Megaload-specific commands have been specified in *oam.config*.

Listing 10: Specifying application-specific commands in oam.config

```
{cmds,
  [
    {"Load configuration file",
      loader,
      load_configuration_file,
      [{"File", string, "Full path to the configuration file"}],
      "Load configuration file with one/several configuration items"},
    {"Start load test",
      loader,
      start_load,
      [{"Test id", string, "Test identifier"}],
```

```
      "Starts the load test identified by Test id. It must be previously loaded in the
          configuration"},
    {"Stop load test",
      loader, stop_load,
      [],
      "Stops the load test"},
    {"Increase rate",
      loader,
      increase_rate,
      [{"Rate", integer, "Percentage rate"}],
      "Increases the call rate by the percentage specified with regards to
          configuration rate"},
    {"Decrease rate",
      loader,
      decrease_rate,
      [{"Rate", integer, "Percentage rate"}],
      "Decreases the call rate by the percentage specified with regards to
          configuration rate"},
    {"Reset rate",
      loader,
      reset_rate,
      [],
      "Resets the call rate to configuration values"},
    {"Abort load test",
      loader, abort_load,
      [],
      "Aborts the load test. All load is stopped immediately"}
  ]
}.
```

4.9 Packaging the WombatOAM-compliant Megaload release

Reached this point, the user is done with the manipulation of the *oam.config* file and all what is left in order to get a WombatOAM-compliant release is to package the OTP-compliant Megaload release as a tarball.

It is important to note that WombatOAM assumes the release to be packaged in tarball with the *.tar.gz* extension. The first element of which must be a directory with the same name we will give to the release when uploading this tarball. Besides standard OTP release files and directories, the top-level directory of the tarball must also contain the *oam.config* configuration file that has been produced in the previous steps.

For the sake of completeness, Listing 11 shows how the Megaload tarball has been produced.

Listing 11: Producing the Megaload tarball

```
tar czf megaload.tar.gz /path/to/the/directory/containing/the/megaload/release
```

4.10 Megaload tarball upload

The next step is to upload the tarball produced in the previous step to WombatOAM. The developed prototype assumes the tarball to be encoded in *base64* [IET06]. Listing 12 shows how the WombatOAM REST API must be used to perform the upload of the tarball.

Listing 12: Uploading the Megaload release to WombatOAM

```
curl -X POST http://<URL_TO_WOMBAT_WEB_SERVICE>/api/releases
     -H "Accept:application/json"
     -H "Content-Type:application/json"
     -d "{'release':
            {
              'name': 'Megaload Release',
              'description': 'Megaload release used in D6.4',
              'tarball': `base64 /path/to/megaload/tarball.tar.gz`
            }
          }"
```

4.11 Creating and Starting Megaload nodes

Once the Megaload WombatOAM-compliant release has been successfully uploaded to the system, we are ready to continue with the creation and starting of the Megaload nodes. Listing 13 shows the *curl* command and the request body used for creating a given number of Megaload nodes.

Listing 13: Creating Megaload nodes

```
curl -X POST http://<URL_TO_WOMBAT_WEB_SERVICE>/api/nodes
     -H "Content-Type: application/json"
     -d "{'amount': <AMOUNT_OF_NODES_TO_BE_CREATED>}"
```

Once created, the nodes are started issuing the *curl* command shown in Listing 14.

Listing 14: Starting all deployed nodes

```
curl -X POST http://<URL_TO_WOMBAT_WEB_SERVICE>/api/nodes/start
```

4.12 Interacting with Megaload

Now that we got all deployed nodes up and running, we can start using the Megaload application. The first thing we must do now is to load the configuration file containing the test cases. This can be easily done using the *loader:load_configuration_file/1* command we previously specified in *oam.config*. To trigger the execution of this command we can do as shown in Listing 15.

Listing 15: Running a custom command on all deployed nodes

```
curl -X POST http://<URL_TO_WOMBAT_WEB_SERVICE>/api/nodes/run_command
     -H "Content-Type: application/json"
     -d '{'command':
            {
              "mod":"loader",
              "fun":"load_configuration_file",
              "args":["megaload_test.txt"]
            }
          }'
```

Finally, once the Megaload configuration file has been loaded, we can trigger a test case by using the *loader:start_load/1* command, as shown in Listing 16.

Listing 16: Running a custom command on all deployed nodes

```
curl -X POST http://<URL_TO_WOMBAT_WEB_SERVICE>/api/nodes/run_command
     -H "Content-Type: application/json"
     -d '{'command':
             {
               "mod":"loader",
               "fun":"start_load",
               "args":["test1"]
             }
           }'
```

5 Conclusions and Future Work

In this deliverable we have teamed with the Megaload development team to gather requirements from a real-world distributed Erlang application. We have also ensured that the gathered results are met by the prototype being developed in WP4. This collaboration has led to the validation of the developed prototype when it comes to deploy real-world applications on a homogeneous virtual infrastructure.

Upcoming D6.5 and D6.6 deliverables will build upon the results achieved by this deliverable and will focus on demonstrating the ability of the developed prototype not only to deploy applications on a heterogeneous virtual infrastructure, but also to carry out a deployment driven by matching deployment requirements with the capabilities of the available clusters.

6 Change Log

Version	Date	Comments
0.1	07/05/2013	First Version
0.2	10/05/2013	Revised version (internal review)

References

[Ama13a] Amazon Web Services, Inc. Amazon EC2 Instance Types, 2013. http://aws.amazon.com/ec2/instance-types/.

[Ama13b] Amazon Web Services, Inc. Amazon Elastic Compute Cloud (Amazon EC2), 2013. http://aws.amazon.com/ec2/.

[Ama13c] Amazon Web Services, Inc. Amazon Machine Images, 2013. https://aws.amazon.com/amis/.

[Eri13a] Ericsson AB. Distributed Erlang, 2013. http://www.erlang.org/doc/reference_manual/distributed.html.

[Eri13b] Ericsson AB. Erlang Applications, 2013. http://www.erlang.org/doc/design_principles/applications.html.

[Eri13c] Ericsson AB. Erlang kernel application, 2013. http://erlang.org/doc/man/kernel_app.html.

[Eri13d] Ericsson AB. Erlang net_adm module, 2013. http://erlang.org/doc/man/net_adm.html.

[Eri13e] Ericsson AB. Erlang net_kernel module, 2013. http://erlang.org/doc/man/net_adm.html.

[Eri13f] Ericsson AB. The Erlang Run-Time System Reference Manual, 2013. http://www.erlang.org/doc/apps/erts/.

[Gar13] Garret Smith. gen_leader library, 2013. https://github.com/garret-smith/gen_leader_revival.

[IET06] IETF. RFC4648: The Base16, Base32, and Base64 Data Encodings, 2006. http://tools.ietf.org/html/rfc4648.

[PRO12] PROWESS. Property-based Testing for Web Services, 2012. http://www.prowessproject.eu/.

[REL11] RELEASE. A High-level Paradigm for Reliable Large-scale Server Software, 2011. http://www.release-project.eu/.