Bachelor Thesis: Extending a SCADA framework to support high availability

1 Client and Supervisor

Client: mindclue GmbH

Client Contact: Andy Rohr, andy.rohr@mindclue.ch

Supervisor: Prof. Dr. Farhad Mehta, HSR Rapperswil

2 Students

- Patrik Wenger, pwenger@hsr.ch
- Manuel Schuler, mschuler@hsr.ch

3 Setting

The company mindclue GmbH, located in Ziegelbrücke, develops SCADA[1] applications for controlling systems used in traffic systems, energy, and water supply. For that purpose, the company developed the *Roadster* framework, which provides the basis for project specific applications.

Roadster is implemented in Ruby and is architecturally based on the Actor model [1], which means that multiple parallel running, single-threaded processes (actors) are coupled via messaging (shared-nothing architecture). The messaging layer is based on ZeroMQ (ZMQ [2]) and has an asynchronous/non-blocking nature. Several different messaging patterns/messaging protocols are used. Additionally, the system includes a web UI which is based on ember.js and connected to the messaging via WebSocket. Fundamentally, the system follows the Reactive Manifesto [3].

4 Goals

The main aim of this thesis is to extend the *Roadster* framework to support high availability.

Roadster currently lacks the following features:

1. A *Roadster* application is currently limited to one node (one instance). The goal is to be able to build systems which consist of multiple nodes. Example: A master node forms together with multiple subordinate nodes a system (basically a distributed system). The subordinate nodes are responsible for their respective subtask of a facility and communicate with their components (e.g. PLCs). The subsystems are integrated into the master node to form an overall view of the facility, which is visualized in the web UI.

   This requirement implies:

[1]Supervisory Control And Data Acquisition, see https://en.wikipedia.org/wiki/SCADA

- Extension of the messsaging protocols to allow the communication between nodes across levels in the hierarchy.
- Encryption of the communication.

2. A *Roadster* application has to have the ability to be run as a highly available active/passive cluster. Two nodes (primary and backup) at the same level in the hierarchy form a hot-standby cluster, where the two nodes stay in constant connection with each other. In case the active node fails, the passive node immediately takes over and becomes the new active node.

   This requirement implies:

   - Extension of the messaging protocols to allow the communication between nodes within the same level in the hierarchy.
   - Implementation of resilient failover mechanisms.
   - Encryption of the communication.

3. With (2), it is possible to implement a highly available OPC UA server. OPC UA [4] includes a concept for redundant UA servers. *Roadster* already implements a OPC UA server, although not highly available.

   This requirement implies:

   - Extension of the OPC UA implementation to support OPC UA HA mechanisms.

The client essentially wants *Roadster* to be extended by the three features described above, whereas the **third one is optional** and is only to be approached in case there is time for it. The **same applies to the encrypted communication requirement**.

5 Tasks

Here is an overview of the currently planned tasks that need to be performed:

1. Getting familiar with the concepts and implementation of *Roadster*, particularly the messaging layer. For that, Andy Rohr (mindclue GmbH) will provide an extensive introduction.

2. Elaboration of a subnode concept. This includes the design, implementation, and testing of extensions of the existing messaging protocols for the communication between nodes of different levels in the hierarchy.

   One of the most important *Roadster* messaging protocols is called *Clone State Protocol* and is based on the *Clone Pattern* described in the zguide [5]. It provides means to replicate the current state of the domain model into the different actors within an application, as those actors behave according to the following principle [6]:

   > "Don't communicate by sharing state; share state by communicating."

   For the communication between nodes, the protocol has to be extended accordingly. The messages are basically Ruby objects serialized using `Marshal.dump` and transported over ZMQ sockets.

3. Elaboration of a HA concept. This includes the design, implementation, and testing of extensions of the existing messaging protocols for the communication betwen nodes within the same level in the hieararchy, including resilient failover mechanisms.

   The *Binary Star Pattern* [7][8] forms the basis of the HA concept. However, the concept will have to be adapted to fit *Roadster*'s needs. Availability has to be ensured under the following scenarios:

   - hardware or software failure of the primary node
   - network failure

   A more detailed definition will have to be worked out during the elaboration phase of the thesis.

4. Implementation of encrypted communication. The current implementation is based on ZMQ 3 and the Ruby binding ffi-rzmq [9]. However, encryption has been introduced in ZMQ 4, which isn't supported by ffi-rzmq. On top of that, ffi-rzmq is not being maintained anymore. A possible solution is CZTop [10], which is based on CZMQ [11] and authored by Patrik Wenger.

   In case CZTop is used, it would have to be extended to allow the watching of ZMQ sockets by EventMachine [12], e.g. `EM.watch(socket_file_descriptor)`. *Roadster* uses EventMachine as a reactor implementation [13].

5. Extensions of the *Roadster* OPC UA server implementation to support HA mechanisms. This part of *Roadster* is a Ruby extension, which is implemented based on the Unified Automation C++ SDK [14]. The extension is written in C++ and uses rbplusplus [15].

6 License

To grant mindclue GmbH unrestricted usage of the student's contributions, the student's code changes and additions shall be protected under the ISC License [16], which is functionally equivalent to the MIT license and the Simplified BSD license, but uses simpler language.

7 Guidelines

The students and the supervisor will plan weekly meetings to check and discuss progress. The student will schedule meetings with the client as and when required (recommendation: 1 meeting per week of 1 hour duration).

All meetings are to be prepared by the students with an agenda. The agenda will be sent at least 24h prior to the meeting. The results will be documented in meeting minutes that will be sent to the supervisor.

A project plan must be developed at the beginning of the thesis to promote continuous and visible work progress. For every milestone defined in the project plan, the temporary versions of all artefacts need to be submitted. The students will receive a provisional feedback for the submitted milestone results. The definitive grading is however only based on the final results of the formally submitted report.

8 Documentation

The project must be documented according to the regulations of the Computer Science Department at HSR [17]. All required documents are to be listed in the project plan. All documents must be continuously updated, and should document the project results in a consistent form upon final submission. All documentation and work artefacts have to be completely submitted in three copies on CD/DVD (one copy each for the client, university, and supervisor). Three printed copies of the report need to be submitted (one copy each for the client, external examiner, and supervisor).

9 Important Dates

Please refer to `https://www.hsr.ch/Termine-Diplom-Bachelor-und.5142.0.html`.

10 Workload

A successful Bachelor thesis project results in 12 ECTS credit points per student. One ECTS point corresponds to a work effort of 30 hours. All time spent on the project must be recorded and documented.

11 Grading

The HSR supervisor is responsible for grading the master thesis. The following table gives an overview of the weights used for grading.

Facet	Weight
1. Organisation, Execution	1/6
2. Report	1/6
3. Content	3/6
4. Final Presentation & Examination	1/6

The effective regulations of the HSR and Department of Computer Science [18] apply.

Rapperswil, Wednesday 28th September, 2016

Prof. Dr. Farhad Mehta

References

[1] URL: https://en.wikipedia.org/wiki/Actor_model.

[2] URL: http://zeromq.org/.

[3] URL: http://www.reactivemanifesto.org/.

[4] URL: https://opcfoundation.org/about/opc-technologies/opc-ua/.

[5] URL: http://zguide.zeromq.org/page:all#Reliable-Pub-Sub-Clone-Pattern.

[6] URL: https://www.igvita.com/2010/12/02/concurrency-with-actors-goroutines-ruby/.

[7] URL: http://zguide.zeromq.org/page:all#High-Availability-Pair-Binary-Star-Pattern.

[8] URL: http://zguide.zeromq.org/page:all#Adding-the-Binary-Star-Pattern-for-Reliability.

[9] URL: https://github.com/chuckremes/ffi-rzmq.

[10] URL: https://github.com/paddor/cztop.

[11] URL: http://czmq.zeromq.org.

[12] URL: http://www.rubydoc.info/gems/eventmachine.

[13] URL: https://en.wikipedia.org/wiki/Reactor_pattern.

[14] URL: https://www.unified-automation.com/products/server-sdk/c-ua-server-sdk.html.

[15] URL: https://github.com/jasonroelofs/rbplusplus.

[16] URL: https://en.wikipedia.org/wiki/ISC_license.

[17] URL: https://www.hsr.ch/Allgemeine-Infos-Bachelor-und.4418.0.html.

[18] URL: https://www.hsr.ch/Ablaeufe-und-Regelungen-Studie.7479.0.html.