UNIVERSITY OF BIRMINGHAM

DESIGNING A FRAMEWORK FOR TESTING ALGORITHMS IN MULTIAGENT SYSTEM

4th year Internship Report - InfoTronique

Presented by

Flavien Bailly
Promotion Hall 2010/2013
Based at the University of Birmingham

Internship supervisor : Dr Peter Lewis
University supervisor : Dr Nader Mbarek

2011/2012

Contents

List of Figures

Acknowledgements

That is not always easy to find an internship, let alone if it is in a foreign country. Therefore, my internship would have not happened if Dr Peter Lewis and Dr Rami Bahsoon had not accepted me, I'm very grateful for this. More, it is a pleasure to work with people like them, with serious and good cheer.

In addition, I am thankful to Francois Jacob for his help to get this internship. In fact, he has done his work placement in the same place last year so he assisted me to get in touch with my supervisor and advised me about Birmingham.

I would like to thank ESIREM teachers for education and knowledge they give us. Everything we learned is useful and our formation is really suitable for the working world. Also, I wish to thank especially Mr Nader Mbarek, my university supervisor, who followed me and gave me advice.

Finally, I would like to express my heartfelt thanks to my parents for their assistance and wishes for the successful completion of this internship. Also, many thanks come to my beloved girlfriend for her encouragement and blessing.

1 Introduction

Within the framework of my 4th year internship, I have been lucky to work as part of the EPiCS project at the School of Computer Science at the University of Birmingham. I wanted to do it in United Kingdom to improve my English. So I packed my luggage, and crossed the English Channel with Eurostar Train direction B'ham.

I was working under the supervision of Dr Peter Lewis who is a post-doctoral research fellow at the Centre of Excellence for Research in Computational Intelligence and Applications (CERCIA) in the School of Computer Science at the University of Birmingham. He is working on the European FP7 project EPiCS and he's part of the Natural Computation research group in the school. At present, his research is primarily concerned with investigating algorithms and techniques for achieving and analysing the effect of self-awareness and self-expression in decentralised computational systems.

Therefore, I was working as an IT technician, implementing and testing programs as part of the EPiCS project. The final output of my internship should be a framework that allows people (EPiCS team) to test their own algorithms. So, first of all, I will present my internship place, the campus in general and the building where I've been which is the School of Computer Science. I will also make a brief presentation about the EPiCS project. Next, I will speak about a framework that we want to design. I will explain its purposes, way and tools used to implement it and discuss about communication between nodes. After, I will speak about two algorithms I have implemented through my multiagent system. Typically, I will show two different transmission mode, synchronous and asynchronous, and what we have done with results. Finally, I will end this report with a conclusion.

2 Internship place

2.1 University of Birmingham

The University grew out of the radical vision of Joseph Chamberlain, the first Chancellor. Founded in 1900, Birmingham represented a new model for higher education. This was Englands first civic university, where students from all religions and backgrounds were accepted on an equal basis. Birmingham has continued to be a university unafraid to do things a little differently, and in response to the challenges of the day.

The University of Birmingham was established by Queen Victoria by Royal Charter in 1900 and was the UKs first civic or "redbrick" university. The first phase of building work on the campus was completed in 1909 under the auspices of the esteemed architect Sir Aston Webb. They celebrated the centenary of those buildings in July 2009. [1]

Figure 1: University of Birmingham - Aerial view

This university is a really good place to study. It is structured into five colleges : Arts and Law, Engineering and Physical Sciences, Life and Environmental Sciences, Social Sciences, and Medical and Dental Sciences, each of which is divided into a number of schools and departments. So that, everybody can find his way. Even if it is an old university, it is a modern campus too, with good place for practicing sport, eating and having a drink.

2.2 School of Computer Science

Computer Science at Birmingham dates back to the late 1950s. This School of Computer Science became one of the first academic departments in the UK to undertake research and teaching in this field. The term "computing" covers every kind of digital technology that we use to create, store, communicate, exchange and use information. As such, it is the foundation for small and large businesses to build their strategies and grow. It is also the key to making our personal lives easier and more fun : mobile phones, online shopping, MP3s... we owe them all, and a lot more besides, to computer science.

The School of Computer Science at the University of Birmingham offers a full range of degrees courses in Computer Science, from Bachelors (B.Sc.) and Masters (M.Sc.) to Doctorates (Ph.D.). Their research lies mainly within the following themes :

- Nature-Inspired and Intelligent Computation
- Computing Systems
- Theoretical Computer Science
- Software Engineering

Each theme contains sub-themes and there is a grouping of academics associated with each of them. [1]

My office was at the second floor of this building. There was about eight persons working in, including my internship supervisor. Most of them are research fellow at the Centre of Excellence for Research in Computational Intelligence and Applications (CERCIA). The main aim of CERCIA is to exploit research present in the Natural Computation Group of the School of Computer Science for the benefit of industry and business in the West Midland region. [2]

The day of my arrival, stuff has been made available to me. First, a personal card has been made in my name to access the building. More, they gave me a key to open and close our desk and other rooms containing a lot of things. I had also a computer, login for access network and a padlock for my personal computer.

2.3 EPiCS project

As I said before, I was working as an IT technician, implementing and testing programs as part of the EPiCS project. EPiCS is a transnational multi-disciplinary research project which aims at laying the foundation for engineering the novel class of proprioceptive computing systems. The EPiCS project is an Integrated Project (IP) funded by the European Union Seventh Framework Programme and is one of five projects funded in the FET proactive initiative "Self-Awareness in Autonomic Systems".

EPiCS is implemented by a consortium of 8 institutions from 5 countries :

- Imperial College, London
- EADS Innovation Works Munich
- University of Birmingham
- Austrian Institute of Technology
- University of Paderborn
- University of Oslo
- Klagenfurt University
- ETH Zurich

Innovations from EPiCS are based on systematic integration of research in concepts and foundations for self-aware and self-expressive systems with novel hardware/software platform technologies and architectures for engineering autonomic compute nodes and networks. EPiCS drives and validates the research by the requirements of three challenging application domains that cover both high-end computers and embedded systems, as well as embeddings into technical and non-technical contexts.

Proprioception, self-awareness, and self-expression are concepts mainly known from psychology, philosophy and medicine. These concepts are rather new to the domains of computing and networking and are thus not yet sufficiently investigated and understood. The innovative claim of EPiCS is that the successful transfer of these concepts to computing and networking domains will help create the powerful and versatile heterogeneous and distributed systems of 2020 and beyond. [3]

3 Designing a framework for multiagent system

The day of my arrival, I had a meeting with Dr Peter Lewis, Dr Rami Bahsoon and two Ph.d students. They presented me their work and introduced concept like "self-awarness" and "self-decision". Then, my internship supervisor talked about a multiagent system which execute reporting tasks thanks to a simulator. It allows EPiCS team to test some of their algorithms. Then, he introduced the main purpose of my internship, a framework that allows people to test their own algorithms inside a multiagent system. Each node of this multiagent system has to be able to execute reporting tasks and to communicate with his neighbors.

3.1 Purposes

Dealing with concepts and foundations for self-aware and self-expressive systems, EPiCS team often use networks. For example, a part of this team is focusing on object tracking handover in a network of smart camera. So that, they need tools to test their algorithms because that's a difficult task in real life.

There is already existing tools that allow people to test their algorithms like *MASON* [4] or *NetLogo* [5], but they present disadvantages. The main of them is that they work with a network simulator using its own language or its own functions. It forces people to write their algorithms twice, one for the simulator, and one for the real life. Because of this, the framework that we try to develop has to be most heterogeneous as possible. With this word, I say that people can write their algorithms using programming language they want, and the framework should work with that.

Also, the framework must allow people to run their algorithms in both synchronous and asynchronous mode. The synchronous one consists to manage algorithm steps in a synchronously way between each nodes of the network. For this, we need to use a "*heart*" or a *monitor* that tells nodes when achieve a part of their work. This is a diffi-

cult part because it isn't always easy to define "steps" in algorithms. With the asynchronous way, which is easier, each node performs his job without taking care about others.

The picture below shows the framework flowchart. The three blocks and links have to be set up by the framework. *Message handler* block is focused on communication between nodes. It can also extract data from the *local knowledge store* to show them to users. *Action engine* block contains users algorithms. Jobs are called to run through the *subscription* to *local knowledge store*. This *subscription* is managed by *publication* action from *Message handler* block.

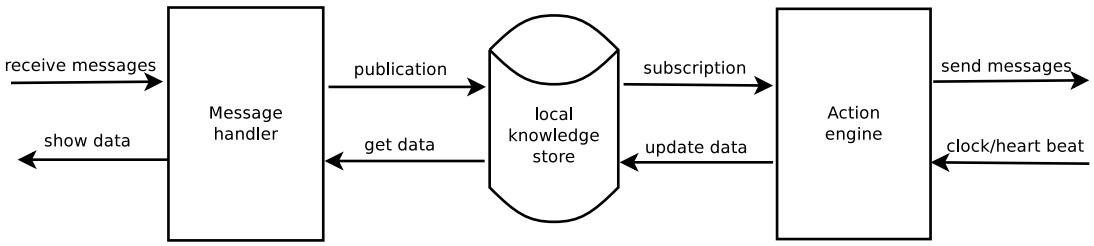

Figure 2: Framework flowchart

Finally, it would be nice if the framework enables people to produce some graphs and network visualisations about multiagent systems they test.

3.2 Implementation

To implement the multiagent system, some tools have been imposed to me. First, the main programming language used : **Ruby**. It is an open source, high level and dynamic object-oriented programming language with a focus on simplicity and productivity. More, my internship supervisor has a very good experience with this programming language, so, he can help me easily if I encounter problems. We can use it with most of Operating Systems. I got all necessary documentation from a book [6]. It also includes "*irb*", an interactive command-line interpreter which can be used to test code quickly. This is very useful to try parts of code in large project. You can see a sample on the next page.

```
$ irb
irb(main):001:0> puts "Hello IRB !"
Hello IRB !
=> nil
```

Figure 3: Basic use of IRB with a terminal

For the *local knowledge store*, we use **Redis**. It is an open source, advanced key-value store. It is often referred to as a data structure server since keys can contain strings, hashes, lists, sets and sorted sets [7]. Redis has two great advantages. First is that it stores data both in RAM and Hard Drive, that makes it very fast. Second is that Redis has got two powerful functions : *publish* and *subscribe*. When a program publish a message on a given channel in the Redis server, every program subscribing to this channel will get the message. This is a good stuff for us for giving signals to our *Action engine*.

At this day, I only ran multiagent system on *localhost*. Each node is characterised by his name and his port number (I will explain this more specifically in the next subsection). These data are stored in the *local knowledge store*. Also, each node must know his neighborhood to set up the network. For this, we store a hash that contains names and port numbers from linked neighbors. Sometimes, depending on the algorithm used in *action engine*, we save other data like values that comes from computing or from neighbors.

3.3 Communication

In order to make nodes communicate, my internship supervisor chose to use **HTTP** protocol [8]. This is an easy way to send and receive messages. More, a lot of programming language can communicate through this protocol, so, it is a good stuff to manage framework heterogeneity.

To realised the *Message handler* block, we used **Sinatra** [9]. This is a free and open source DSL[1] written in *Ruby* for quickly creating web applications in *Ruby* with minimal effort.

[1]domain-specific language

```
# sample.rb
require 'sinatra'

get '/hi' do
   'Hello Sinatra'
end
```

Figure 4: Sinatra - sample.rb

These few code lines allow us to start a *HTTP server* on *localhost:4567*. So, if we go to the URL *http://localhost:4567/hi* on a web browser (*HTTP client*), we will see the message "Hello Sinatra". This library enables to use all HTTP method[2] in a similar way. It is a good stuff in order to receive messages and to show data using a web browser.

For sending messages, *Action engine* block needs to be able to speak *HTTP*. Most of programming languages can do it. In our case, we used *net/http gem*[3] with *Ruby*. Typically, running on *localhost*, each node runs its own sinatra application (Message handler block) using different **port number**. As well, we was able to send messages between them using it. For example, if we have two nodes, "A" and "B", running respectively on two different port number, "2222" and "3333", "A" can send a message to "B" using *http://localhost:3333*, and vice versa.

Finally, we used **JSON**[4] string to send data. It is a lightweight data-interchange format that is easy for humans to read and write. It is a text format that is completely language independent but uses conventions that are familiar to programmers of a lot of programming language [10]. This stuff is really useful when we want to send objects. A process that wants to send an object only has to serialized it as a *JSON string* then send it. Following, the receiver process can deserialized the *JSON string* and then use the object.

[2] *get*, *post*, *delete*... see [8]
[3] gems, in Ruby, are packages
[4] JavaScript Object Notation

4 Testing algorithms

Before going on framework implementation, I tested algorithms, both in synchronous and asynchronous way through multiagent systems. During these tests and according to results, framework design has grown.

4.1 Newton-Raphson

Newton-Raphson method is a simple algorithm. My internship supervisor suggested me to perform it on each node of a multiagent system in both synchronous and asynchronous way. The main purpose was for me to get used with tools like *Sinatra*. In fact, all tools I have used was totally new for me.

4.1.1 Algorithm presentation

Newton-Raphson method, in numerical analysis, is a method for finding, step by step, better approximations to the roots of a function. It is based upon a knowledge of the function tangent. More, this is an iterative process so that we can easily define "step" for our use.

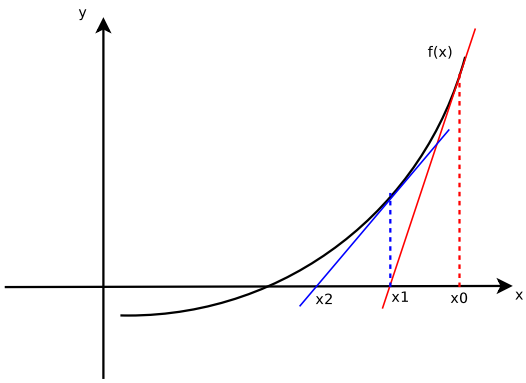

Figure 5: Newton-Raphson

The picture above shows exactly how this algorithm works. It begins with a starting point, here, **x0**. We draw the function tangent at this point and take its intersection with the x-axis. We obtain **x1**. Then, we repeat this process until we reach an accurate value of function root. **x1**, **x2** ... **xN** are steps that we want to use.

4.1.2 Implementation

This part was mostly for testing **Sinatra**. Therefore, we used only one process running it to perform both *Action engine* block and *Message handler* block. In fact, this process can be called a node. It is identified with a name and a port number. We was able to run it both in synchronous and asynchronous mode by giving a parameter at the beginning.

The *Local knowledge store* was made with a simple file, and each nodes had its own. Inside it, they wrote a couple of data at each step of Newton-Raphson method : current step number and current x-value.

There was also a monitor process that manage nodes thanks to *HTTP* methods. It was able to collect data from nodes through *JSON* documents and then could make results interpretation and graphs. The following picture shows monitor management and differences between synchronous and asynchronous node.

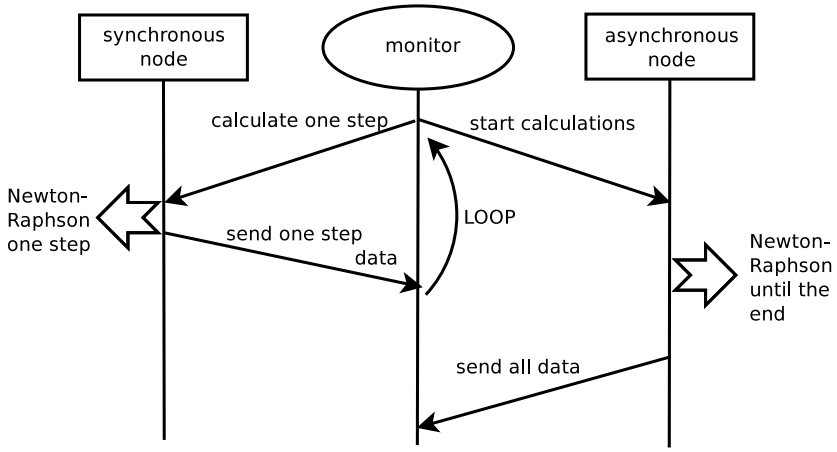

Figure 6: Newton-Raphson in a multiagent system - flowchart

4.1.3 Results interpretation

To try this method, we chose a function and a precision that gave us a large number of step : $f(x) = x^7 - 7$ with a 0.000000001 accuary. With this, we had about 30 steps before finding the result. However, each node gave us the same result because every of them began with

the same starting value (**x0**). As a result, we made it stochastic. For this, each node must choose its starting value between 40 and 70. Then, depending of this value, results was different.

The, when I had a lot of data files thanks to *BASH* scripting, I got used with **Gnuplot** for showing graphs. The following picture shows a sample we have done, where standard deviation and mean have been calculated across among 30 independent runs.

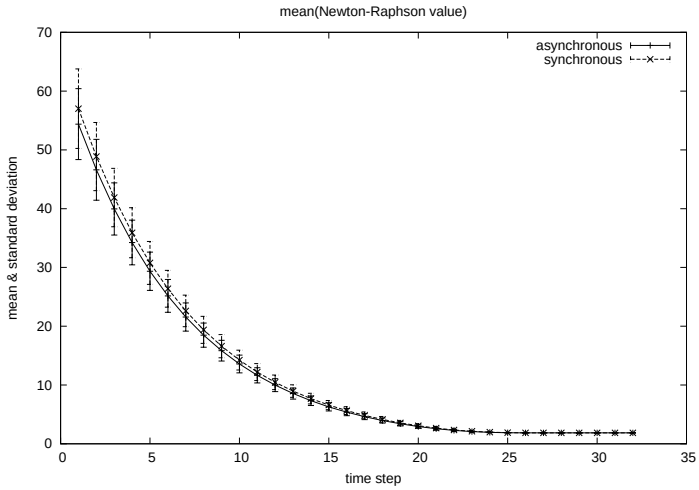

Figure 7: Gnuplot graph

4.2 Graph-Coloring

Graph-Coloring algorithms are more difficult than Newton-Raphson method. In fact, they need interactions between nodes in order to find a solution. This kind of algorithm is close to what EPiCS team wants to do.

4.2.1 Algorithm presentation

The main goal of this algorithm is to assign colours to nodes from a network. But, there is a constraint which is that neighbors cannot have the same colour. It can look easy like this but, the more nodes and links a network have, more this algorithm becomes complex.

It exists various version of this algorithm, each one using different approach. On our part we used the **Asynchronous Backtracking** method. I used a very good paper ([11]) to learn how it works and then deal with it.

In this algorithm, the priority order of nodes is determined, and each node communicates its tentative value assignment to neighboring agents via *ok? messages*. Each node maintains the current value assignment of other nodes from its viewpoint called *agent view*. The priority order is determined by the alphabetical order of node identifiers. A node changes its assignment if its current value assignment is not consistent with the assignments of higher priority nodes. If there exists no value that is consistent with the higher priority nodes, the node generates a new constraint called a *nogood*, and communicates it to a higher priority node via *nogood messages*. Then the higher priority node changes its value.[11]

A *nogood* is a subset of an *agent view*, where the node is not able to find any consistent value with the subset. In the simplest case, it can use the whole agent view as a *nogood*. Here is an example :

$$x_i = 1 \wedge x_j = 2 \wedge x_k = 3 \rightarrow x_l \neq 1$$

This nogood could be generated by the node x_l because is value is not consistent with its *agent view*. In fact, we see that x_i has the same value.

4.2.2 Algorithm implementation

I have made two version of this algorithm implementation. The first one looks like the Newton-Raphson method implementation. I used only one program to perform both *Message handler* block and *Action engine* block. However, results was not what we expected, this implementation way has made this process deterministic. If we run it many times, we always obtain the same result. This happened because we used *HTTP* functions which are blocking. Therefore, when a node sends a message, it waits for his receiver. And, with this algorithm, it often happens that a node sends messages when it receives one. So, nodes was waiting in "cascade" in the same order every time.

In order to avoid the problem explained above, we tried an other way focused on the framework flowchart I showed before. We separated the process in two different block (*Message handler* block and

Action engine block) and we added the *Local knowledge storage* with a *redis server*. The two blocks communicate with it and we don't encounter the problem anymore because receiving a *HTTP message* involve a publication on the *redis server* and not an other *HTTP message*.

The completeness[5] of the algorithm is guaranteed, but the time and the number of operation are unknown. It is a good implementation in order to make tests because it reflects a real network where computing and transmission time are variable.

4.2.3 Network visualisation

In order to watch our results with this algorithm, I have developed a monitor that gets colour back from each node. Then this monitor is able to show a colored network on a web page.

For making this, I have used *Sinatra* with a *Ruby* program to ask colours and to put the web page online. Also, I have used **Cytoscape Web**, a *jQuery*[6] plugin that visualises graphs [12]. With it, we can easily make pretty network visualisations. Here is a sample that we obtained.

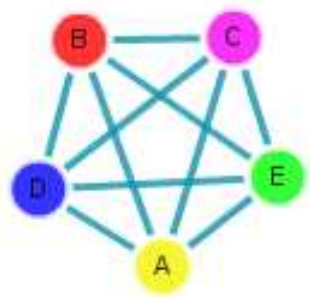

Figure 8: Network visualisation - Cytoscape web

The disposition is managed automatically and we can drag nodes. We can also add some information about nodes and links. Nodes colour are not show in real-time but I'm working on it.

[5] always founds a solution if one exists, and terminates if no solution exists

[6] jQuery is a new kind of JavaScript Library.

5 Conclusion

At the time of writing my report, a month is remaining, and the framework is not ended, not even really started. The main goals will be to try to define steps inside our graph coloring algorithm, and then manage nodes progression with a monitor. Following, I think we will focus on framework development and documentation. The figure below shows time I spent on each part.

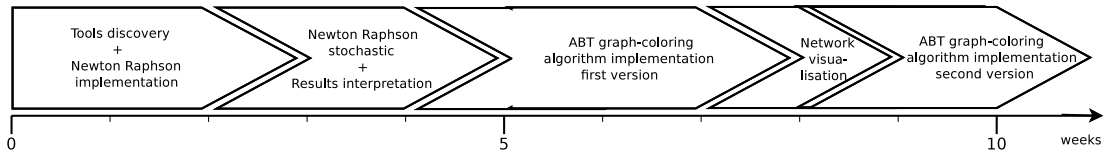

Figure 9: Chronology

This internship has been very instructive for me. First, I improved my english. I progressed my listening and I am more able to speak. Also, I worked with a lot of tools I never used before, particularly web stuff. This is a good thing for my curriculum vitae. Finally, I have been lucky to participate at EPiCS meeting. It was very informative and interesting.

I recommand the University of Birmingham, the School of Computer Science and my internship supervisor Dr Peter Lewis for futur ESIREM student. Even if this internship was unpaid, the juice is worth the squeeze. It is a very good experience in the field of research, such as it made me like it.

References

[1] University of Birmingham, About us, a brief history, School of Computer Science,2012, *[online]: www.birmingham.ac.uk.*

[2] CERCIA, About us, History, 2009, *[online]:http://www.cercia.ac.uk/.*

[3] EPiCS project, Home, Project, Consortium, 2012, *[online]:http://www.epics-project.eu/.*

[4] MASON : Mutli-Agent Simulator Of Networks, George Mason University's Evolutionary Computation Laboratory, GMU Center for Social Complexity, *[online]:http://cs.gmu.edu/ eclab/projects/mason/.*

[5] NetLogo, Center for Connected Learning (CCL),Uri Wilensky, *[online]:http://ccl.northwestern.edu/netlogo/.*

[6] Programming Ruby 1.9, The pragmatic programmers' guide, Dave Thomas, Chad Fowler, Andy Hunt, 2009, *[book] The Pragmatic Programmers.*

[7] Redis, Salvatore Sanfilippo, Pieter Noordhuis, VMware, 2009-2010, *[online]:http://redis.io/.*

[8] HTTP, Internet RFC 2616, R. Fielding, et al, 2004, *[RFC]:http://www.ietf.org/rfc/rfc2616.txt.*

[9] Sinatra, Blake Mizerany, and the rest of the GitHub crew, *[online]:http://www.sinatrarb.com/.*

[10] JSON,D. Crockford, IETF, Network Working Group, 2006, *[RFC]:http://www.ietf.org/rfc/rfc4627.txt*, *[online]:http://www.json.org/.*

[11] Algorithms for Distributed Constraint Satisfaction, Makoto Yokoo, Katsutoshi Hirayama, Autonomous Agents and Multi-Agent Systems, Vol 3, No 2, 2000.

[12] Cytoscape Web, *[online]:http://cytoscapeweb.cytoscape.org/.*