NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

FACULTY OF EXACT SCIENCES
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

BACHELOR THESIS

RHEA: A Reactive, Heterogeneous, Extensible and Abstract Framework for Dataflow Programming

Orestis Melkonian

Supervisors: **Panos Rondogiannis,** Professor EKPA
Angelos Charalambidis, Researcher NCSR

ATHENS

APRIL 2016

BACHELOR THESIS

RHEA: A Reactive, Heterogeneous, Extensible and Abstract Framework for Dataflow Programming

Orestis Melkonian
A.M.: 1115201000128

SUPERVISORS: **Panos Rondogiannis,** Professor EKPA
Angelos Charalambidis, Researcher NCSR

ABSTRACT

Summary here

SUBJECT AREA: Programming Languages

KEYWORDS: dataflow, stream, frp, distributed systems, model-driven architecture program synthesis declarative languages node placement

”τὰ όντα ιέναι τε πάντα καὶ μένειν ουδέν”
(all entities move and nothing remains still)
- Heraclitus

ACKNOWLEDGEMENTS

I would like to thank Angelos Charalambidis for his immensely helpful supervision and guidance throughout the whole period of 6 months that I was present in NCSR.

I would also like to thank Professor Panos Randogiannis for being a major influence in my current research interests through the undergraduate courses ”Theory of Computation” and ”Principles of Programming Languages”, where I was introduced to a much more declarative way of programming and started appreciating mathematical sensibility in computing.

CONTENTS

LIST OF FIGURES

PROLOGUE

This bachelor thesis is a continuation of my internship at the National Centre for Scientific Research ”Demokritos”, particularly in the Software and Knowledge Engineering Laboratory (SKEL).

The main task I was assigned was the implementation of a framework for robot programming using a dataflow approach. During that internship, I came to realize that my work could be easily generalized to cover a much broader application area than just robot software.

The name of the framework stems from the ancient Greek Titaness Rhea(*Ρέα*), daughter of earth goddess Gaia(*Γαία*) and sky god Uranus(*Ουρανός*) and etymologically derives from the verb *ρέω*(to flow).

1. INTRODUCTION

1.1 Main concept

My main contribution is the design and implementation of a framework for dataflow programming to be deployed anywhere, ranging from low-performance robots and sensors to clusters of computer and even the Cloud.

The main idea is to provide the programmer with a different execution model, the dataflow model, which allows for a more abstract way of thinking and has the advantage of exposing opportunities for parallelism (amongst CPU cores) and distribution (amongst computational machines) that the ”intelligent” underlying system can automatically realize. Therefore, the programmer will be able to gain good performance and utilization of the available computational resources, while at the same time reducing development time and cost and maintaining a much cleaner and easier-to-refactor software system.

1.2 Motivation

1.2.1 Declarative languages

Software is becoming increasingly more complex each year, as computing capabilities are strengthened and user needs become more and more demanding. Thus the need for higher abstraction becomes mandatory, as it provides a more structured, easier to debug and maintainable way of developing software. In other words, abstraction in computer science acts as a mean to overcome complexity.

In programming languages, the level of the aforementioned abstraction is measured regarding the amount of low-level details a programmer has to specify. Therefore, languages can be divided in two categories: the imperative ones, which specify what needs to be done and how to do it, and the declarative ones, which only specify what needs to be done and rely on the underlying compiler/interpreter to produce the exact commands that will realize the desired behaviour. The most well-known declarative programming paradigms are functional and logic programming, each providing higher abstraction in different aspects. My approach was greatly influenced by the functional paradigm.

1.2.2 Data versus Computation

1.2.3 Dataflows in Robotics

1.2.4 Dataflows in Internet of Things

1.2.5 Dataflows in Big Data

1.3 Outline

There are nine chapters which compose this thesis: *Introduction, Background, Approach, Implementation, Applications, Related Work, Future Work, Conclusions*.

Background introduces the reader to basic background knowledge, necessary for complete understanding.

Approach presents the main characteristics of my approach.

Implementation gives a more detailed specification of the framework.

Applications present some use-cases, ranging from general mathematical problems to real-life robot scenarios.

Related Work discusses relevant concepts and technologies, which influenced major decisions concerning the design and implementation of the framework.

Future Work suggests some interesting topics for future research, whose embedding in the framework is meaningful.

Conclusions sums up.

2. BACKGROUND

2.1 The dataflow computational model

The increased interest in parallelism during the 70's gave rise to the dataflow execution model, which is an alternative to the classical "von-Neumann" model. In the dataflow model, everything is represented in a dataflow graph, where nodes are independent computational units and edges are communication channels between these units. A node/unit is fired immediately when its required inputs are available and therefore no explicit control commands are needed for execution. Figure 1 shows a dataflow graph enumerating the set $\mathbb{N}$ of natural numbers.

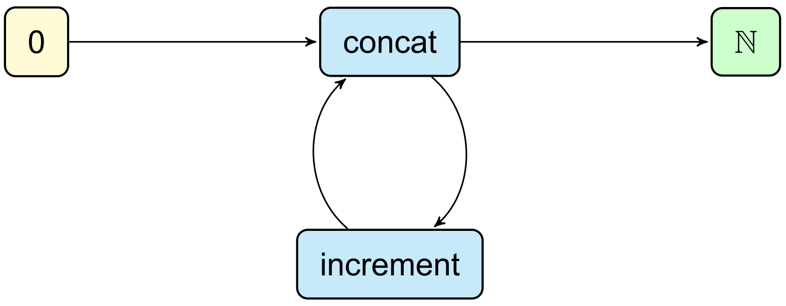

Figure 1: Natural numbers

The main advantage of the dataflow model is its implicit parallelism, deriving from the fact that the computational units are totally independent and therefore can be executed in parallel. The communication can either be an in-memory data storage or even a TCP connection across the network. Its great flexibility and composability make it a good candidate for the underlying architecture of a framework with a high level of abstraction.

2.2 Functional reactive programming

A relatively recent model of programming is Functional Reactive Programming (FRP), which provides a conceptual framework for implementing reactive(i.e. time-varying and responding to external stimuli) behaviour in *hybrid systems* (i.e. containing both continuous and discrete components), such as robots, in functional programming languages. It first appeared as a composable library for graphic animations (cite FRAN), but quickly evolved into a generic paradigm (cite Yale papers).
Although appealing at first, FRP was not appropriate for systems with real-time constraints (cite EventFRP), due to uncontrollable time- and space- leaks. The solution was a generalization of monads called *arrows* (cite Hughes), which provided the necessary guarantees that the aforementioned common errors do not occur. Let's see the example of calculating a robot's x-coordinate. Here is the mathematical formula drawn from control theory:

$$x = 1/2 \int (vr + vl) \cos \theta$$

Below is the FRP code corresponding to the formula above:

```
x = let
  v = (vrSF &&& vlSF) >>> lift (+)
  t = thetaSF >>> arr cos
  in (v &&& t) >>> lift (*) >>> integral >>> lift (/2)
```

As the above may seem counter-intuitive, a new notation was conceived, notably the *arrow notation* (cite arrow notation):

```
x = proc inp -> do
    vr <- vrSF -< inp
    vl <- vlSF -< inp
    theta <- thetaSF -< inp
    i <- integral-< (vr+vl) * cos(theta)
returnA -< (i/2)
```

The main advantages of FRP are its close correspondence to mathematics, which make it an ideal framework for modelling real-time systems, and its concise representation of time-varying values via *signals*.
For a more detailed view of the FRP idiom and all of its variants please see the literature (cite Push/Pull, RealTimeFRP, Yampa).

2.3 Publish-Subscribe model

Publish/Subscribe(PubSub) is a communication paradigm that became popular due to the loose coupling of its components, suited for the most recent large-scale distributed applications.

There is no point-to-point communication and no synchronization. *Publishers* advertise messages of a given type to a specific message class or *topic* that is identified by a *keyword*, whereas *subscribers* listen on a specific *topic* without any knowledge of who the publishers are. The component responsible for relaying the messages between machines and/or processes and finding the cheaper dissemination method is called *message broker*. Figure 2 is an abstract graphical representation of the PubSub model.

2.4 ROS: Robot Operating System

ROS is an open-source middleware for robot software, which emphasizes large-scale integrative robotics research(cite ROS). It provides a *thin* communication layer between heterogeneous computers, from robots to mainframes and tt has been widely adopted by the research community around the world, due to its flexibility and maximal support of reusability through packaging and composability. It provides a nice solution to the

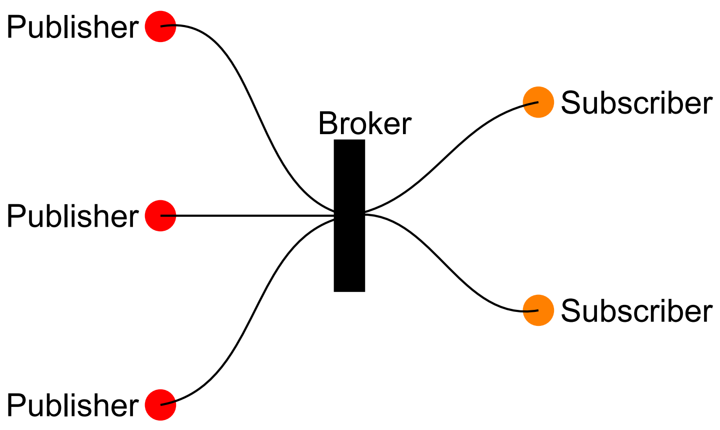

Figure 2: PubSub typical layout

development complexity introduced by complex robot applications that consist of several modules and require different device drivers for each individual robot.

It follows a peer-to-peer network topology, implemented using a topic-based PubSub messaging protocol and its architecture reflects many sound design principles. Another great property of ROS is that it is language-agnostic, meaning that only a minimal specification language for message transaction has been defined, so contributors are free to implement small clients in different clients.

A typical development scenario is to write several *nodes*, that subscribe to some topics

2.5 Internet of things

3. APPROACH

3.1 Reactive

3.2 Heterogeneous

3.3 Extensible

3.4 Abstract

4. IMPLEMENTATION

4.1 The Reactive Streams Standard

4.2 Stream variables

4.3 Stream operators

4.4 Deployment

4.5 Optimization

4.5.1 Heuristic-based graph transformations

4.5.2 Network-aware node placement

4.6 Serialization

5. APPLICATIONS

5.1 Hamming numbers

5.2 Camera surveilance

5.3 Robot control panel

5.4 Robot hospital guide

6. RELATED WORK

6.1 GoogleDataflow

6.2 TensorFlow

6.3 Akka

6.4 dispel4py

6.5 Flowstone

6.6 Spark

6.7 Naiad

6.8 NoFlo

6.9 NodeRed

6.10 Yampa

7. FUTURE WORK

7.1 More evaluation strategies

7.2 Dynamic reconfiguration

7.3 Advanced network profiling

7.4 Integration with other technologies

7.5 Domain-specific operators

7.6 Stream reasoning

8. CONCLUSIONS

TERMINOLOGY TABLE

A table of used scientific terms follows.

κλάσση	class
εντολή	command
περιβάλλον	environment

ABBREVIATIONS, INITIALS AND ACRONYMS

A table of all abbrevations used throughout the thesis follows.

FRP	Functional Reactive Programming
JVM	Java Virtual Machine
NCSR	National Centre for Scientific Research
ROS	Robot Operating System
IoT	Internet of Things
CPU	Central Processing Unit
TCP	Transmission Control Protocol
PubSub	Publish/Subscribe

REFERENCES

[1] E. A. A. William W. Wadge, *Lucid, the Dataflow Programming Language*. Academic Press, 1985.

[2] H. Jass, “A big paper,” *The journal of big papers*, vol. MCMXCVII, 7991.