ID: MSCA-2020-CDRoover01

Title: A General Theory for Distributed Reactive Programming

Reactive programming is getting ever more popular. Many frameworks and programming languages exist. However, most of them are:

- *single node*: they don't have a purposefully designed built-in distribution and partial failure model.
- *crafted atop an existing programming language*: they actually consist of 2 programming languages, one to build the reactive signals/event sources and another one (often functional) to write the code that is made reactive.
- *implemented in a very naive way*: the implementation constructs a DAG and the DAG is traversed to generate the reactive behaviour.
- textual: good IDEs for dealing with reactive programming do not exist.
- *small-data*: reactive signals can be built for reacting to thermometers or clicks on a web page but there is no support for reacting to enormous amounts of data streams at the same time.

Our research is constructing a new Reactive Programming Paradigm based on first principles. The key element is a “reactor”: a first-class building block that represents a reactive program.

Research is conducted:

- *in language design*: Which reactor composition operators are necessary to compose reactive programs? Can these operators also be expressed as reactors themselves and can we give a meaningful semantics to higher-order reactors? Can we combine reactors with imperative languages features (e.g. storage data structures, actors, …)?
- *in formalisation*: What is the denotational semantics of reactors? Can we give a simple formal description of the streams of events and values that are handled by a reactor composition?
- *in implementation technology*: Instead of relying on the naive DAG-representation of a reactive program. We are developing a reactive virtual machine with static memory requirements and *guaranteed throughput.*
- *in meta-programming*: We are developing an implementation of reactive programming in reactive programming. The goal is to design a open reactive programming language that can be extended in itself.
- *in applications*: How far does the application space of reactive programming reach? We are applying RP to neural networks, sorting networks, big data processing systems, app development and cyber-physical systems. How do we need to enrich or adapt existing RP languages in order to cater to such domains.
- *in security aspects*: In the last couple of years we see a clear trend towards the streaming paradigm and RP are just perfect to design and implement such systems. However, this also means that streaming data should not end up in the wrong hands.
- *in distribution*: the defining feature of distributed systems is that they can fail partially. I.e. even if part of the system is gone, the remaining parts keep on functioning. Handling such partial failures in reactive systems is unexplored territory.
- *in programming environments*: In a SmallTalk-like sense, we would like to express an IDE for RP as a reactive programming itself. The IDE is a giant reactor that is consuming the inputs of the programmer and reacting to them appropriately.

Supervisor: wdmeuter@vub.ac.be
Research Group: **http://soft.vub.ac.be/soft/**
To apply: https://www.vub.ac.be/en/european-liaison-office#apply-msca-if