# Frabjous|| : A Declarative Framework for Agent-Based Modelling

Ivan Vendrov
Dept. of Computer Science
University of Saskatchewan
Saskatoon, SK, Canada
ivan.vendrov@usask.ca

Christopher Dutchyn
Dept. of Computer Science
University of Saskatchewan
Saskatoon, SK, Canada
dutchyn@cs.usask.ca

Nathaniel Osgood
Dept. of Computer Science
University of Saskatchewan
Saskatoon, SK, Canada
osgood@cs.usask.ca

## ABSTRACT

-todo-

## Categories and Subject Descriptors

D.3.2 [**Programming Languages**]: Language Classifications - *functional, parallel, data-flow*; I.6.5 [**Simulation and Modeling**]: Model Development; Modeling Methodologies; J.3 [**Life and Medical Sciences**]: Health

## General Terms

Languages, Experimentation

## Keywords

Functional reactive, functional programming, simulation, dynamic model, domain-specific language, agent-based simulation, agent-based modelling, data parallel programming

## 1. INTRODUCTION

For systems that evolve continuously in space and time, the language of differential equations (DE's) - honed by centuries of application to the physical sciences - has no substitute. Its syntax is extremely terse, with precise mathematical semantics that permit sophisticated analysis. While differential equations are not a natural fit for modelling populations of individuals (these being essentially discrete), the System Dynamics community has used them with great success to model the dynamics of large populations *(insert citations / examples)*.

There are, however, a number of processes that are difficult to express with differential equations, such as those involving networks or a high degree of heterogeneity in the populations being modelled [6]. The need to model these processes is addressed by agent-based (AB) modelling, a more general approach, which involves specifying the behaviour of each individual in the population and allowing the global dynamics to emerge from the interaction of individuals.

The generality of agent-based modelling comes with a number of costs. With existing tools and frameworks, agent-based models are significantly harder to create, extend, and understand; significantly more expensive to calibrate and run; and significantly harder to mathematically analyze relative to models based on systems of differential equations [7].

Although the increased cognitive and computational costs of agent-based models are to some degree unavoidable due to the models' increased complexity and generality, we argue that these costs have been exacerbated by the use, in many AB modelling frameworks, of imperative languages like Java and C++. While these languages are well-suited for general-purpose programming, they are not good specification languages, due to their verbose nature and hiding of essential details. They generally force modellers and users to think at a low level of abstraction, and fail to cleanly separate the relationships at the heart of the model from implementation details such as input/output, the time-stepping mechanism, and the data structures used [6].

On the other hand, the underlying language of DE models is not imperative but declarative - rather than explicitly specifying rules by which model variables change, differential equations specify relationships between model variables that hold at all times. We believe that the declarative nature of DE models accounts for much of their success by simplifying model creation, modification, and analysis. It then stands to reason that AB models could be similarly simplified by basing them on an appropriate declarative language. To support this hypothesis, we develop such a language and use it to implement a number of standard models from the literature.

## 2. BACKGROUND

In this section, we briefly describe the existing languages and technologies we used to create Frabjous||, as well as explain why we chose them.

### 2.1 Haskell

Haskell is a purely functional programming language; that is to say, a Haskell program is a list of equations, each defining a value or a function. As an example, consider the following Haskell code:

```
b = True
f x = 2 * x + 1
g = f 2
```

Here `b` is defined to be the Boolean value True, `f` is defined to be the function $f(x) = 2x + 1$, and g is defined as $f(2)$, i.e 5.

Note that the equal sign in a Haskell definition denotes mathematical equality, not the assignment of a value to a variable. The values of `b`, `f`, and `g`, once defined, cannot change for the duration of the program.

Since Haskell lacks a mechanism for changing the value of a variable, it comes very close to the declarative ideal - specifying what things are, not how they change - and reaps the associated benefits: Haskell programs are often an order of magnitude shorter than programs written in imperative languages, are clearer to read, and are much easier to analyze mathematically (CITE). For these reasons, we have chosen Haskell as the base language of FrabjousII, in that FrabjousII code is largely composed of segments of Haskell code, and compiles directly to Haskell. *Chris: more about the process*

## 2.2 Functional Reactive Programming

A weakness of functional programming is the difficulty of representing systems that vary with time, since there is no mechanism for changing state. As pioneered by Elliott and Hudak, functional reactive programming (FRP) is a paradigm that augments functional programming with "behaviours" - values that change over time - as well as a set of primitive operations on these values [2]. This allows complex, time-varying, locally stateful systems to be written in a declarative fashion, as demonstrated by Courtney et al [1].

For example, we might model an agent's immunity to a particular disease by the following FRP equation:

```
immunity = 2 + (sin time)
```

where `immunity` is a time-varying value - for instance, it has value 2 at time 0, and value 1 at time $\pi$.

Then suppose the agent has, on average, three contacts with infective agents per day. We could model this as a Poisson process:

```
contact = rate 3
```

where `contact` is `True` for a few instants during each day, and `False` the rest of the time.

Finally, suppose that we want to keep track of the number of *exposures* - the number of times the agent had a contact with an infective while her immunity was particularly low ($< 1.5$). This informal specification translates directly to code:

```
exposures = count (contact && (immunity < 1.5))
```

Here `count` is an FRP operator that counts the number of instants for which the its argument has value `True`.

FRP can in fact be viewed as a generalization of differential equations. An FRP program is essentially a set of equations between time-varying values (in other words, functions of time). But where differential equations are limited to the standard mathematical operations such as addition, multiplication, and the differentiation operator, FRP adds a number of operators (like `rate` and `count` above) that act on and produce a much richer variety of functions. The differentiation operator is only defined on smooth functions in a vector space, but FRP operators can produce piecewise and step functions on arbitrary sets, which permits us to specify fundamentally discrete processes.

We use FRP as the basis for our ABM language because its declarative nature provides the transparency, clarity, and concision usually associated with declarative modelling [5, 6], and because FRP has an explicit, formal semantics [2] that makes it possible to reason mathematically about programs written in it.

## 2.3 Frabjous

The generality of FRP comes at a cost, however. Understanding the syntax used in existing FRP libraries such as Yampa [1] or Netwire [9] requires familiarity with advanced functional programming concepts such as monads [10], arrows [3], and applicative functors [4]. While these concepts allow for a great deal of conceptual elegance and generality, much of that generality is not necessary for ABM, and so a less general language with syntax oriented towards modellers rather than functional programmers would be desirable. This was precisely the motivation for the development of Frabjous, a domain-specific language for agent-based modelling built on top of FRP [8].

Frabjous demonstrated that core mechanisms of ABM (specifically, state charts) could be expressed in FRP in a much more concise and human-readable form than that of Java code generated by the popular AnyLogic framework [8], giving evidence to our claim that declarative languages are better suited to ABM than imperative languages, at least for concerns of software engineering. But Frabjous was a proof of concept, not a usable language for ABM; it placed a number of restrictions on agent state, behaviour, and network structure that drastically reduced its generality. In this paper, we redesign and extend Frabjous to yield a language that is still concise and readable, but is general enough to describe, in principle, any agent-based model.

## 3. SUMMARY

## 4. FUTURE WORK

## 5. ACKNOWLEDGEMENTS

This work was funded in part by a Natural Science and Engineering Research Council of Canada (NSERC) Undergraduate Student Research Award (USRA).

## References

[1] A. Courtney, H. Nilsson, and J. Peterson. The yampa arcade. In *Proceedings of the 2003 ACM SIGPLAN workshop on Haskell*, Haskell '03, pages 7–18, New York, NY, USA, 2003. ACM.

[2] C. Elliott and P. Hudak. Functional reactive animation. *International Conference on Functional Programming*, pages 263–273, 1997.

[3] J. Hughes. Generalising monads to arrows. *Science of Computer Programming*, 37:67–111, 2000.

[4] C. McBride and R. Paterson. Applicative programming with effects. *J. Funct. Program.*, 18(1):1–13, Jan. 2008.

[5] N. Osgood. Systems dynamics and agent-based approaches: Clarifying the terminology and tradeoffs. *Proceedings of the First International Congress of Business Dynamics*, 2006.

[6] N. Osgood. Using traditional and agent based toolsets for system dynamics: Present tradeoffs and future evolution. *Proceedings of the 2007 International Conference on System Dynamics*, 2007.

[7] H. Rahmandad and J. Sterman. Heterogeneity and network structure in the dynamics of contagion: Comparing agent-based and differential equation models. *Proceedings of the 2004 International Conference on System Dynamics*, 2004.

[8] O. Schneider, C. Dutchyn, and N. Osgood. Towards frabjous: a two-level system for functional reactive agent-based epidemic simulation. In *Proceedings of the 2nd ACM SIGHIT International Health Informatics Symposium*, IHI '12, pages 785–790, New York, NY, USA, 2012. ACM.

[9] E. Soeylemez. Netwire, 2012. [Online; accessed 29-August-2013].

[10] P. Wadler. Monads for functional programming. In J. Jeuring and E. Meijer, editors, *Advanced Functional Programming*, volume 925 of *Lecture Notes in Computer Science*, pages 24–52. Springer Berlin Heidelberg, 1995.