Comparing Repast Java and Functional Reactive ABS

jonathan.thaler@nottingham.ac.uk

August 10, 2017

Abstract

This study we compares the Agent-Based Simulation programming libraries Repast and FrABS. As both are of fundamentally different programming paradigms - Repast uses Java, FrABS Haskell - both propagate fundamental different approaches in implementing ABS. In this document we seek to precicesly identify these fundamental differences, compare them and also look into general benefits and drawbacks of each approach.

Contents

Chapter 1

Introduction

TODO

1.1 FrABS

This library was developed as a tool to investigate the benefits and drawbacks of implementing ABS in the pure functional programming paradigm. As implementation language Haskell was chosen and the library was built leveraging on the Functional Reactive Programming paradigm in the implementation of the library Yampa.

1.2 Repast Java

This library TODO:

1.3 Focus

We have to be very clear about a thing: there is a huge difference between Repast and FrABS. To paraphrase Fred Brooks [**?**]: Repast is a final *programming system* ***product*** whereas FrABS is merely a (yet unfinished) *programming system*. Repast easily surpases FrABS features-set and this is probably best reflected in Repasts high usability. It has a user-Interface which allows to customize the appearance and parameters of the simulation with a few clicks and visualize and exporting of data created by the simulation. Clearly usability is FrABS major weakness: it completely lacks a user-interface [1] and although FrABS allows to customize the appearance of the visualization and allows exporting of data through primitive file-output (e.g. constructing a matlab-file),

[1] So far there are no plans to add one, also because GUI-programming in Haskell is not straight-forward, constitutes its own art and would in general amount to a substantial amount of additional work for which we just don't have the time.

everything needs to be programmed and configured directly in the code. So from the perspective of usability, Repast is just very much better and thus we won't spend more time in investigating this and just state that so far usability is the big weakness of FrABS. So considering the big conceptual difference between the two libraries and their different feature-sets, the question is then, how to compare them, on what to focus. It is important to emphasize that ultimately both libraries provide support in programming ABS and thus this is what we will focus on: compare how to implement agent-based models directly in code and to see if there are fundamental differences and if yes which they are. Note that we refrain from comparing library-features isolated and instead look at them always directly in the context of the use-cases as described below.

1.4 Evolution

It is important to note that during conducting this study we sometimes slightly extended and refactored FrABS to express problems easier. One could say that this is cheating as Repast does not has this advantage but we argue that this is a legitimate approach because:

- The fundamental differences are already set and cannot be changed due to the fundamental differences of the programming paradigms and the resulting different approach to implementing ABS.
- Repast can be considered to a finished product with more features, whereas FrABS is still in its Alpha-Phase.
- It shows how easily and quickly (or not) FrABS is able to adapt features already available in Repast.

Most notable additions were:

- Introduction of single/multi occupant/non-occupant cells in the Discrete2D environment and supporting functions for accessing these.
- Introduction of a mapping of a generic type to 2d-continuous positions in the Continuous2d environment and supporting functions for accessing / changing these.

These functionalities existed already in part in other examples (e.g. Agent_Zero, Sugarscape) and were thus only refactored back into the FrABS library.

1.5 Use-Cases

As use-cases on which we conduct the study we implement the following models in *both* libraries:

- JZombies - the 'Getting Started' example from Repast Java [2] - a first, very easy model, as there exists code-listing for Repast Java there is a 'standard' implementation, so comparison can be straight-forward and will serve as a first starting point.
- System-Dynamics SIR - study how the libraries deal with continuous time-flow.
- ABS SIR - study how the libraries support time-semantics.
- Sugarscape Model as presented in the book "Growing Artificial Societies - Social Sciences from the bottom up" by Joshua M. Epstein and Robert Axtell [1] - this highly complex model serves as a use-case for investigating how the libraries deal with a much more complex model with a big number of features. In this model there are no explicit time-semantics like in the SIR model: agents move in every time-step where the model is advanced in discrete time-steps.

[2] `https://repast.github.io/docs/RepastJavaGettingStarted.pdf`

Chapter 2

FrABS

TODO: read papers

- Functional Reactive Programming from First Principles
- Functional Reactive Programming, Continued
- Arrows, Robots, and Functional Reactive Programming
- The Yampa Arcade
- Functional Reactive Programming, Refactored
- On the Mathematical Properties of Monadic Stream Functions
- Back to the Future: Time Travel in FRP
- Testing and Debugging Functional Reactive Programming
- Structure and Efficiency of Computer Programs

Chapter 3

Repast Java

3.1 JZombies

3.1.1 Repast Java

3.1.2 FrABS

Parallel vs. Sequential -¿ Sequential is as in Repast -¿ Parallel is novel, but need to collapse environment: but how?

needed to implement standard-cells in discrete2d: -¿ empty / non-empty with single / multiple occupier -¿ supply helper-functions -¿ occupy / unoccupy / isoccupied -¿ allOccupiers -¿ supply functions which construct these discrete2d with standard-cells

agents are implemented as the union of ADT

3.1.3 Comparing dynamics

Both use the same parameters, the one specified in the Repast Example.

FrABS leads much faster to a full infection: in case of using shuffling we arrive at full infection around 80 steps. When NOT using shuffling, is is happening around 140 steps.

Repast Java leads to a full infection after around 200 steps.

I think the reason for this is a different scheduling for the human agents. The zombies agents move in every time step but the humans have a watch added which gets triggered as soon as a zombie enters their moore-neighbourhood: $@Watch(watcheeClassName = "jzombies.Zombie", watcheeFieldNames = "moved", query = "within_moore1", whenToTrigger = WatcherTriggerSchedule.IMMEDIATE)$ Thus humans are scheduled right after an approaching zombie which allows them to flee thus prolonging the time until all humans are infected thus impacting the dynamics. In the case of FrABS the result is that although this specific zombie who enters the moore-neighbourhood does not directly pose a threat for the human as this human is still on another patch, until the human is scheduled, other

zombies can close in thus reducing the action-radius of the human considerably.

This Watcher-Mechanism is not possible in FrABS and there are no plans to implement such as it would need a completely different approach. Also we cannot think of a mechnism which allows to specify such a meta-programming construct in Haskell.

TODO: try using conversations which allows humans to react to zombies too near. should result in similar dynamics? TODO: implement watcher-mechanism by using some Schedule-ADT which is very similar to conversations but without the whole infrastructure of conversations

3.2 SIR System Dynamics

3.2.1 Repast Java

`https://repast.github.io/docs/RepastSystemDynamicsGettingStarted.pdf`

3.2.2 FrABS

3.3 SIR Agent-Based Simulation

3.3.1 Repast Java

3.3.2 FrABS

3.4 Sugarscape

3.4.1 Repast Java

3.4.2 FrABS

References

[1] Epstein, J. M., and Axtell, R. *Growing Artificial Societies: Social Science from the Bottom Up*. The Brookings Institution, Washington, DC, USA, 1996.