Comparing Repast Java and Functional Reactive ABS

jonathan.thaler@nottingham.ac.uk

July 28, 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

0.1 Introduction 3
0.2 JZombies 4
0.3 SIR System Dynamics 4
0.4 SIR Agent-Based Simulation 4
0.5 Sugarscape 4

0.1 Introduction

TODO

0.1.1 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.

0.1.2 Repast Java

This library TODO:

0.1.3 Focus

One of the main benefits of Repast is its high usability. It has a User-Interface, allows to customize the appearance of the simulation with a few clicks 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 simulation and exporting of data through primitive file-output (e.g. constructing a matlab-file), 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. Instead the main focus will be now in comparing how to implement the 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.

0.1.4 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.

[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.

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

- 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.

0.2 JZombies

0.2.1 Repast Java

0.2.2 FrABS

0.3 SIR System Dynamics

0.3.1 Repast Java

0.3.2 FrABS

0.4 SIR Agent-Based Simulation

0.4.1 Repast Java

0.4.2 FrABS

0.5 Sugarscape

0.5.1 Repast Java

0.5.2 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.