A model for pure functional agents

Jonathan Thaler

November 4, 2016

Abstract

I want to develop a powerful pure functional agent-model which allows convenient multi-step conversations in continuous time as i feel yampa is not the perfect match

1 Introduction

Start from wooldridge 2.6 (look at the original papers which inspired the 2.6 chapter) and weiss book and the original papers those chapter is based upon. Look into denotational semantics of actor model. Look also in functional models of czesar ionescu. why: this is the major contribution of my thesis and is new knowledge. Must find intuitive, original and creative approach.

no concurrent execution: deterministic
deterministic: can use random-numbers but to be reproducible/deterministic one has to specify the same seed or even provide an own RNG-implementation (which is easily possible using the RNG in haskell)
could use STM for state-propagation to other agents
read Improving Performance of Simulation Software Using Haskell's Concurrency & Parallelism
Sketch EDSL for ACE model of interest or maybe more general ACE models.
In the end it all boils down to 1. the agent-model and 2. how the agent-model is implemented in the according language. Of course both points influence each other: functional languages will come up with a different agent-model (e.g. hybrid like yampa) than object-oriented ones (e.g. actors).

2 Agents

3 FRP

paradigm for programming hybrid systems which combine continuous and discrete components. Time is explicitly modelled: there is a continuous and synchronous time flow.

there have been many attempts to implement FRP in frameworks which each has its own pro and contra. all started with fran, a domain specific language for graphics and animation and at yale FAL, Frob, Fvision and Fruit were developed. The ideas of them all have then culminated in Yampa which is the reason why it was chosen as the FRP framework. Also, compared to other frameworks it does not distinguish between discrete and synchronous time but leaves that to the user of the framework how the time flow should be sampled (e.g. if the sampling is discrete or continuous - of course sampling always happens at discrete times but when we speak about discrete sampling we mean that time advances in natural numbers: 1,2,3,4,... and when speaking of continuous sampling then time advances in fractions of the natural numbers where the difference between each step is a real number in the range of [0..1])

time- and space-leak: when a time-dependent computation falls behind the current time. TODO: give reason why and how this is solved through Yampa. Wan and Hudak (2000)
Yampa solves this by not allowing signals as first-class values but only allowing signal functions which are signal transformers which can be viewed as a function that maps signals to signals. A signal function is of type SF which is abstract, thus it is not possible to build arbitrary signal functions. Yampa provides primitive signal functions to define more complex ones and utilizes arrows Hughes (2005) to structure them where Yampa itself is built upon the arrows: SF is an instance of the Arrow class.

Fran, Frob and FAL made a significant distinction between continuous values and discrete signals. Yampas distinction between them is not as great. Yampas signalfunctions can return an Event which makes them then to a signal-stream - the event is then similar to the Maybe type of Haskell: if the event does not signal then it is NoEvent but if it Signals it is Event with the given data. Thus the signal function always outputs something and thus care must be taken that the frequency of events should not exceed the sampling rate of the system (sampling the continuous time-flow). TODO: why? what happens if events occur more often than the sampling interval? will they disappear or will the show up every time?

switches allow to change behaviour of signal functions when an event occurs. there are multiple types of switches: immediate or delayed, once-only and recurring - all of them can be combined thus making 4 types. It is important to note that time starts with 0 and does not continue the global time when a switch occurs. TODO: why was this decided?

Yampa has been used in multiple agent-based applications: Hudak et al. (2003) uses Yampa for implementing a robot-simulation, Courtney et al. (2003) implement the classical Space Invaders game using Yampa, the thesis of Meisinger (2010) shows how Yampa can be used for implementing a Game-Engine, . Note that although all these applications don't focus explicitly on agents and agent-

based modelling / simulation all of them inherently deal with kinds of agents which share properties of classical agents: game-entities, robots,...
Nilsson et al. (2002)

4 Model

use yampa as basic framework for "mainloop" with continuous time where each agent has a signalfunction which is triggered when he receives a message or conversation request. in a conversation all other agents are then halted and only the 2 are communing, then time continues. all is running in sync and messages are transfered via non-concurrent STM so no rollback/retry/... stuff is needed. also need to deal with how new agents are created and inserted into the system and how existing ones can be removed when died

References

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

Hudak, P., Courtney, A., Nilsson, H., and Peterson, J. (2003). *Arrows, Robots, and Functional Reactive Programming*, pages 159–187. Springer Berlin Heidelberg, Berlin, Heidelberg.

Hughes, J. (2005). Programming with arrows. In *Proceedings of the 5th International Conference on Advanced Functional Programming*, AFP'04, pages 73–129, Berlin, Heidelberg. Springer-Verlag.

Meisinger, G. (2010). Game-engine-architektur mit funktional-reaktiver programmierung in haskell/yampa. Master's thesis, Fachhochschule Oberösterreich - Fakultät für Informatik, Kommunikation und Medien (Campus Hagenberg).

Nilsson, H., Courtney, A., and Peterson, J. (2002). Functional reactive programming, continued. In *Proceedings of the 2002 ACM SIGPLAN Workshop on Haskell*, Haskell '02, pages 51–64, New York, NY, USA. ACM.

Wan, Z. and Hudak, P. (2000). Functional reactive programming from first principles. In *Proceedings of the ACM SIGPLAN 2000 Conference on Programming Language Design and Implementation*, PLDI '00, pages 242–252, New York, NY, USA. ACM.