AN API FOR ECONOMIC AGENTS

DAVID R. PUGH, DANIEL F. TANG, J. DOYNE FARMER

1. Motivation...

Our objective is to create a *user-friendly* toolkit for building *scalable*, *data-driven* agent-based models (ABMs) of *economic* systems.

1.1. **Requirements.**

2. Implementation

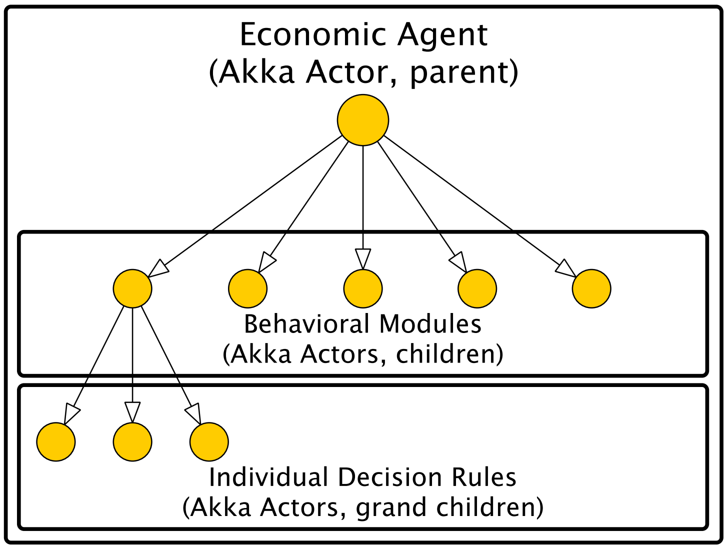

Figure 1. An economic actor in our framework is represented as a hierarchical tree of Akka Actors.

3. Model Layer

An economy is populated with many seemingly disparate types of agents (i.e., consumers, producers, financiers, government, etc). Our goal is to distill the core essence (in terms of data and behaviors) of these different agents into a multi-layered Application Programming Interface (API) defining a generic *economic agent* that can then be specialized to the various types of economic agents needed for any particular model.

Date: September 17, 2015.

Several key features distinguish *economic* agents from more generic types of agents. At a minimum these features include: purpose driven (or goal oriented) behaviour, an ability to learn, and the ability to anticipate future events. This suggests that our framework will need APIs for...

- Goals or objectives (and their associated behavioral rules): Our goals/objectives API needs to be as un-opinionated as possible as prospective users of ScalABM are likely to have strong opinions on how to define appropriate goals and decision rules for their agents. At the same time, we will need to have some type of underlying null model of agent goals/objectives. Utility maximization, Belief-Desire-Intent (BDI), probabilistic discrete choice (AKA, random utility maximization) are possibilities. These frameworks seem to have much in common.
- Learning rules: A large number of various learning rules/mechanisms have been proposed in the academic literature. Roughly, learning rules seem to fall into two camps: learning through previous experience (i.e., reinforcement learning) and learning through observation (i.e., belief learning). Useful references for learning rules for ABMs are the two handbook chapters Brenner (2006) and Duffy (2006).
- Expectations formation rules: A large number of various expectation formation rules have been proposed in the literature. Useful references for expectation formation rules are Hommes (2006), Anufriev and Hommes (2012), Woodford (2013), Assenza et al (2014).
- Decision rules: Depending on type, an economic agent might need to make a number of supply or demand decisions.

Concurrent communication between real economic agents is a fundamental fact of economic life. While a substantial amount of communication between economic agents is direct (or "peer-to-peer"), perhaps the majority of communication between agents is indirect via market institutions. Our framework leverages the Actor model of concurrency (as implemented in Akka) to model both direct and indirect communication between economic agents via concurrent asynchronous message passing.

This suggests that our framework will need APIs for...

- Communications: In order to constrain the communication problem, we need to develop an API that defines both the type and the content of messages that can be passed between agents. Our agent communication API is influenced by, but not slave to, the Foundation for Intelligent Physical Agents (FIPA) compliant Agent Communication Language (ACL).
- Market institutions: In addition to being able to communicate with economic agents, a market needs to have an auction mechanism for determining prices and quantities (i.e., double-auction mechanism,

market clearing, etc) and a market needs to have a clearing mechanism for processing transactions (although this might simply involve linking to some underlying payment system).

3.1. **An API for agent communication.**

Concurrent communication between real economic agents is a fundamental fact of economic life. While a substantial amount of communication between economic agents is direct (or "peer-to-peer") the majority of communication between agents is indirect via market institutions.

We have developed an API for agent communication which specifies:

- a set of abstract message types that impose structure on the messages passed between a group of economic agents,
- a set of abstract protocols that impose structure on conversations (i.e., sequences of messages) between a group of economic agents.

Each abstract message type can be thought of as defining an "envelope" containing the actual content of the message that is to be exchanged between a group of economic agents. Defining envelopes containing messages is useful because it allows agents to react based on the type message received. Each abstract protocol defines a particular subset of message types that can be sent by an agent in response to a particular type of message that it has received.

Our agent communication API is influenced by, but not slave to, the Foundation for Intelligent Physical Agents (FIPA) compliant Agent Communication Language (ACL).

3.1.1. *Examples.*

An important example of an abstract message type is the `RequestLike` message. In our agent communication API this abstract message has three concrete types:

- `Request`: Agent (i.e., `sender`) sends a `Request` message to another agent (i.e, `receiver`) if the `sender` wants the `receiver` to perform some action as specified in the message content.
- `RequestWhen`: Agent (i.e., `sender`) sends a `RequestWhen` message to another agent (i.e, `receiver`) if the `sender` wants the `receiver` to perform some action as specified in the message content when some precondition is satisfied.
- `RequestWhenever`: Agent (i.e., `sender`) sends a `RequestWhenever` message to another agent (i.e, `receiver`) if the `sender` wants the `receiver` to perform some action as specified in the message content whenever some precondition is satisfied.

Another important abstract message type is the `ResponseLike` message. In our agent communication API this abstract message has two concrete types:

- `Agree`: A message sent from some agent (i.e., `sender`) to another agent (i.e., `receiver`) indicating that the sender has agreed to perform some action(s) as specified in a `RequestLike` message previously sent by the `receiver`.
- `Refuse`: A message sent from some agent (i.e., `sender`) to another agent (i.e., `receiver`) indicating that the sender has refused a `RequestLike` message.

Finally consider four additional message types:

- `Inform`: A message sent from some agent (i.e., sender) to another agent (i.e., receiver) informing the receiver that some proposition is true.
- `NotUnderstand`: A message sent from some agent (i.e., sender) to to another agent (i.e., recevier) informing it that some previously received message was not understood.
- `Cancel`: A message sent from some agent (i.e., sender) to another agent (i.e., receiver) indicating that a previously received Request message (i.e., request) to perform some action should be canceled.
- `Failure`: A message sent from some CommunicatingActor (i.e., sender) to another agent (i.e., receiver) indicating that some action was attempted, but that the attempt failed.

With these message types in hand, we define a `RequestHandler` protocol that specifies a valid sequences of these message types (i.e., a "request conversation").

(1) An agent (i.e., `initiator`) sends a `RequestLike` message to another agent (i.e., `participant`).
(2) The `participant` responds with either an `Agree` or `Refuse` message (or in extreme cases with a `NotUnderstood` message).
(3) Assuming the `participant` agreed to the request, the `participant` attempts to perform necessary actions and then responds with either `Failure` or `Inform`, depending.

There is an important exception to the protocol. At any point the `initiator` of the request can send a `Cancel` message to the `participant` indicating that it wishes to `cancel` its previous request.

Examples of additional abstract communication protocols can be found on the FIPA web site. Concrete implementations of these abstract protocols will define behavioral strategies that will be mixed together to create an specific economic agent.