Davoud Taghawi-Nejad (2013)

Modelling the Economy as an Agent-Based Process: ABCE, A Modelling Platform and Formal Language for ACE

Journal of Artificial Societies and Social Simulation 16 (3) 1

<<http://jasss.soc.surrey.ac.uk/16/3/1.html>>

Received: 30-Oct-2012 Accepted: 25-Nov-2012 Published: 30-Jun-2013

Abstract

In this paper, I argue that the key innovation of Agent-Based Economics is not the introduction of the individual agent as an ontological object, but the fact that the economy is modelled as a process. I propose a formal language to express economic models as processes. This formal language leads to ABCE, a modelling platform for Agent-Based Economic models. ABCE's core idea is that the modeller specifies the decisions of the agents, the order of actions, the goods and their physical transformation (the production and the consumption functions). Actions, such as production and consumption, interactions and exchange, are handled automatically by the modelling platform, when the agent decided to do them. The result is a program where *the source code contains only economically meaningful commands*. Beyond the decisions and the setup, ABCE handles everything in the background. It scales on multi-core computers and cloud computing services, without the intervention of the modeler. ABCE is based on python, a language which is characterized by highly readable code.

Keywords:

Agent Based Modeling, Macroeconomics, Ontology, Economics, Process, Platform

Introduction

- 1.1 ABCE is a discrete event platform for rapid development of Agent-Based Economic models. The core design goal has been that the model's source contains only economically meaningful code. Everything that has no economic meaning is hidden and automatized by the platform.
- 1.2 What is special about ABCE is that it handles all goods, production, consumption, interaction and trade. After specifying the order of actions, endowments, consumption and production functions, the modeller can concentrate on the economic decisions the agents make. For example, when an agent decides to trade, he instructs the platform to do so. ^[1] The platform then offers the counter-party the choice, and clears the trade when the counter-party accepts.
- 1.3 This automation allows the platform to run the simulation on multi-processor servers or cloud-computing systems, and further, to use the full power of multi-core processors. But the modeller does not have to deal with the complexities or know anything about distributed computing.
- 1.4 This paper has three sections. In the first section I will discuss the ontological philosophy that underlies ABCE. Here I will illustrate how agent-based modelling is more than merely a model with agents: it is also explicitly modelling objects (goods) and their transformation (production and consumption). ABMs model a process where agents transact (trade), interact and transform. While these considerations are interesting in their own right and help us to understand ABCE, the impatient reader can skip directly to the second section, which explains how to use the ABCE modelling platform. The third section goes into the technical details of ABCE, such as multiprocessing, cloud computing and the python language, licensing, and open-source development of the ABCE itself.

Ontological Considerations

- 2.1 Say (1880) defines economics as the science of production, distribution, and consumption of wealth. In agent-based modelling the actors, which Say mentions only implicitly, are modelled. However, this quote draws attention to the objects missing in our agent-based ontology: wealth, the *goods*, or more generally *objects* that agents possess. The possessions of all agents constitute the distribution of wealth. What is more, Say identifies the two transformations: production and consumption.
- 2.2 Thus the economy is a set of *agents*. Each agent possesses a set of *goods* (wealth) which can be *transformed* or *transferred* between agents. Furthermore, agents can *interact*.
- 2.3 A simulation of an agent-based model is a sequence composed of the three types of actions executed by the agents: the *process*. The core difference between agent-based modelling and conventional economics is not the presence of individual agents, but the modelling of the economy as a *process*, a sequence of *interaction*, *transformation of goods* and *transfer of goods* between *agents*, guided by the constraints and the agents' decisions.
- 2.4 The actions in the process are constrained by physical laws and the possession of goods by the individual agents. The law of conservation of energy says: nothing can be created and nothing can be destroyed. In a physically consistent system this has to be obeyed. Objects therefore can only be transformed. In economic science this transformation is generally a production function, which maps a vector of input goods to a vector of output goods. A special kind of transformation is consumption. In economics this is represented by a utility function: a vector of goods is used up by an agent. The subjective usefulness of the consumption for this agent is represented by utility. Some goods are perishable: that means that, left alone, they transform themselves - often to a less useful form. Furthermore, an agent-based model of a macro-economy must be physically consistent. For the object of study of economists this subset of the physical laws is sufficient. In simulations in which the survival of the agents is modelled, the consumption function is an additional physical constraint.
- 2.5 In many economic models we encounter services. These are transformations of goods for other agents, usually in exchange for another good: money. However it is often convenient to model labour as a good. Labour then is part of the transformation input vector. This pretended labour good is non-durable, as it cannot be transferred between periods and replenishes every round.
- 2.6 A model has an initial endowment of agents and goods. Within the boundaries of the physical constraints, agents' actions are the process that convey the model from period to period. A model then is a set of
 - *agents* and their *endowments of goods*
 - *physical constraints* (production and consumption functions, the characteristics of goods^[2])
- 2.7 When the *process* — the agents' *decisions* and *actions* (interaction, transformation and transfer) — is simulated, the output of the process reveals emergent patterns in the actions and the distribution of goods.
- 2.8 One way of seeing the ABCE, the agent-based modelling platform, is that it is an implementation of this formal language. The modeller defines the physical world (goods, goods properties and transformations), the agents' decisions and the order of actions. All transformations, actions and interactions are decided by the agents, but handled completely automatically by the ABCE modelling platform. The result is a model where almost every line of the source code has economic meaning.

ABCE: a brief introduction

- 3.1 In ABCE, the modeller has to define the initial endowments, the physical reality as represented by the production and consumption functions and properties of some goods, and the order of actions that is repeated in every round. What remains is to define the agents' decisions and behavior.

Initial endowments and the physical reality

- 3.2 In the initiation of each agent:
 - production functions
 - consumption functions
 - initial endowment of goods
- 3.3 In start.py for the whole simulation:
 - goods, that perish
 - goods, that get renewed every round (round endowment, e.g. labour, renewable resources)
 - normal goods, which do not have to be explicitly declared.

The agents' decision and process

- 3.4 In ABCE all *interactions* and *transfers* between agents and *transformations* of the goods by the agents are automatically handled. That means that the modeller only has to specify the decision of the agent.^[3]

Agents' interaction and transformation methods

```

transformation of goods:
    produce(input_goods)
        Produces output goods given the specified amount of inputs
        and the agent's production function
    consume(input_goods)
        consumes input_goods returns utility according to the agent's
        consumption function
transfers of goods between agents:
    sell(receiver, g, q, p)
        offers to sell the quantity q of good g at price p to agent receiver
        The agent can receive it at the beginning of the next round.
    get_offers(g)
        returns all offers of the good g ordered by price.
    accept(offer)
        accepts the offer
    rejects(offer)
        rejects the offer
    give(receiver, g, q)
        give quantity q of good g to agent receiver.

interaction:
    message(receiver, content)
        sends a message to an agent.
        The agent receives it at the beginning of next round.
    shout_message(receiver_group, content)
        sends a message to an agent group or 'all'.
        Agents receive it at the beginning of next round.
    messages()
        returns all new messages

```

3.5 Furthermore, there are methods that predict transformation methods and inform the agent about his possessions:

Agents' possessions and transformation predictors

```

possession(g)
    how much of good g an agent possesses
predict_produce_output(goods)
    Predicts the output of a certain input vector and given the
    production function
net_value(goods, prices)
    Calculates the net_value of a goods_vector for a particular price_vect
    and the technology.
predict_utility(goods)
    Predicts the utility of a vector of input goods

```

Example

- 3.6 Now we will expose a very simple economy comprising one firm and one household. This example has no decisions, just actions. It is left as an exercise to the reader to fill this model with decisions so that it represents an economic problem. The model 'one_household_one_firm' can be found in the example folder of the ABCE platform. In start.py, the agents are imported, the action_list and goods with special properties are declared.^[4]
- 3.7 This defines the order in which the agents trade. The good 'GOOD' has no special properties: it can be used without declaration. Labour, on the other hand, has two properties: first, if it is not used in one round it simply disappears. Second, every household gets one unit of labour per adult in every round::

```

w.declare_perishable(good='labor')
w.declare_round_endowment('adult', product='labor')

```

- 3.8 As promised, the code of the firm and the household are almost completely economically meaningful. Accordingly, we reproduce it in full::

```

import abceagent

```

```

class Household(abceagent.Agent, abceagent.Household):
    def __init__(self, wp, ap, _pass_to_engine):
        abceagent.Agent.__init__(self, *_pass_to_engine)
        self.create('adult', 1)
        self.set_cobb_douglas_utility_function({"GOOD": 1})

    def sell_labor(self):
        """ offers one unit of labor to firm 0,
        for the price of 1 'money' """
        self.sell('firm', 0, "labor", quantity=1, price=1)

    def buy_goods(self):
        """ receives the offers and accepts them one by one """
        oo = self.get_offers("GOOD")
        for offer in oo:
            self.accept(offer)

    def consumption(self):
        """ consumes everything and logs the
        aggregate utility. current_utility """
        current_utility = self.consume_everything()

```

- 3.9 The agent inherits from `abceagent.Household`: this means it has a utility function and can consume. At the beginning of the simulation in the `__init__` method, we endow the household with one working adult. We set the utility function to a degenerated Cobb-Douglas function:

$$utility = GOOD^1$$

```
import abceagent
```

```

class Firm(abceagent.Agent, abceagent.Firm):
    def __init__(self, wp, ap, _pass_to_engine):
        abceagent.Agent.__init__(self, *_pass_to_engine)
        self.create('money', 1)
        self.set_cobb_douglas("GOOD", 1, {"labor": 1})

    def buy_labor(self):
        """ receives the offers and accepts them one by one """
        oo = self.get_offers("labor")
        for offer in oo:
            self.accept(offer)

    def production(self):
        """ uses all labor that is available and produces
        according to the set cobb_douglas function """
        self.produce_use_everything()

    def sell_goods(self):
        """ offers one unit of labour to firm 0,
        for the price of 1 "money" """
        self.sell('household', 0, "GOOD", self.possession("GOOD"), 1)

```

- 3.10 The firm agent is of type `abceagent.Firm`, which means that it has a production function and can produce. In the `__init__` method, that function is executing once at the beginning of each simulation. One unit of money is created. With `set_cobb_douglas`, we set a Cobb-Douglas production function of the type:

$$GOOD = labor^1$$

- 3.11 When we start the simulation, the agents call and execute the actions as specified in the action list::

```

action_list = [
    ('household', 'sell_labor'),
    ('firm', 'buy_labor'),

```

```

('firm', 'production'),
('firm', 'sell_goods'),
('household', 'buy_goods'),
('household', 'consumption')
]

```

- 3.12 At the beginning of the round, the households receive one unit of labour for every adult, as specified by `declare_round_endowment`. When we execute the program, first the household sells labour to firm 0.^[5] The modeller could fill this in with a more sophisticated logic for setting the price. The offer arrives in the next sub-round, meaning the next action in the action list. In this case it is ('firm', 'buy_labor'). The firm receives all offers `oo` and can inspect them and accept them one by one.^[6] Here, we accept all offers. In the next round the offer is cleared.
- 3.13 The firm produces the good using all the labour it has and sells it in a similar fashion to how the agents sold labour. Subsequently the households consume. Then, in the next round, the simulation starts again with the action list, and repeats as many times as is specified in `simulation_parameters.csv`.

Technical Consideration

The programming language

- 4.1 Because agents have variables—things they know or possess and things they can do—object-oriented programming is the natural representation for agent-based models.
- 4.2 Scientific computing has additional requirements to programming language:
1. The code should be readable and all lines should have an economic meaning.
 2. The time of development is more important than the execution speed.
- 4.3 The programming languages traditionally used in economics, though much less in computer science, are C++, Matlab and Java. These languages are verbose; variables need to be declared; in the case of C++, the memory needs to be managed and many other lines of code have no economic meaning.^[7] Modern high-level programming languages, such as Ruby, Python and Perl, are much more concise. Of the three, Python has been designed explicitly to be readable. The high readability works as a hedge against mistakes and wrong results, both on the programming level as well as on the model level.
- 4.4 Development time is crucial, because of the pressure to publish and the fact that researchers have to be funded when they do the modelling. Computer power, on the other hand, is readily available. It is easy to rent an Amazon elastic cloud and create a network of computers with hundreds of cores for a modest price. Modern internet startups such as Dropbox, Google, Mendeley and many others use python as their staple scripting or programming language.
- 4.5 Another key advantage of python is that it is very easy to learn. The author of this paper learned Python on a long weekend. A huge and very helpful community can be found on <http://www.stackoverflow.com>. It is therefore advantageous to use this new, but simpler tool, rather than a complicated one, just because it is already known.
- 4.6 One of the common critiques of scripting languages is that they are slow. The answer to this is that the biggest part of the execution time of a program is spent only in the inner loop. For Python Numpy and Scipy are available, packages that make calculations with the speed of C++. Furthermore, as ABCE scales over cores, processors and computers, just buying processing time solves this problem. The speed advantage thus becomes less important.

Cloud computing and multiprocessors

- 4.7 As Moor's Law approaches its physical limit (Gorder 2007), the increase of computing performance is largely obtained by multi-core processors, multi-processor servers and cloud computing. However, running a program created for a single core computer on more than one core or processor does not lead to any speed advantages. The program's tasks need to be divided into subtasks that can be run concurrently. This can only be done manually. There are tasks that are inherently indivisible.
- 4.8 Agents in agent-based models are by definition separate entities. They have their own memory and processes. So the simulations can be separated and the agents can be separated on different computers or processor cores. Creating simulations that use the full power of several cores, processors or servers is therefore not prohibitively difficult for an experienced programmer. But it is time-consuming and requires knowledge of multiprocessing/threading and messaging frameworks. Fortunately ABCE handles multi-core and processors automatically: the modeller does not even see it. If you use more than one computer, this is a simple matter of configuring the addresses correctly.
- 4.9 We can view an agent as a separate program that interacts with other programs. When you create your model in ABCE, you essentially write *agent-programs*. The agent-program inherits a method that allow it to communicate and send 'pseudo-physical' objects to other agents. Further agents wait for the 'beat' of the ABCE platform in order to synchronize their actions. The

implementation is done with zeromq. Zeromq is a "socket library that acts as a concurrency framework" (Hintjens 2013). In other words, much like the Internet messaging protocol, it routes messages between programs, computers, processors or cores. Zeromq was chosen for several reasons. First, it is available on a wide range of platforms: Windows, Linux, and all modern supercomputer or cloud computing facilities support it. It is backed by Intel and is generally expected to be included in the Linux kernel in the near future.

Messaging and synchronization

- 4.10 In ABCE, all agents are autonomous processes. Two processes organize the Simulation. `abce.Simulation` sets up the simulation and controls the order of Actions, and the hidden `abce.Communication`^[8] process synchronizes the agents' actions and passes messages from agent to agent.
- 4.11 In a situation where many agent-processes exist parallel to each other in the computer, the consistency of time is not fixed. Agents could run 'faster' than other agents and the simulation would be systematically biased or the simulation could simply not work.
- 4.12 The consistency of time is important only when agents interact. In a discrete event simulation, time is a sequence of actions. The sequence is cyclically repeating, and one full cycle is a round. We will refer to an action as a sub-round. A sub-round is a block of code that does something. As this could be more than one thing at a time the word 'action' is misleading.^[9]
- 4.13 There is a correspondence between simulation time and simulated time (real time). If one agent runs faster, it is still known where in the sequence he is, and the corresponding point in simulated time is therefore known too.
- 4.14 If in reality two persons interact, both are at the same point in time. Therefore inconsistency arises when agents interact when they are at different points in time. In order to avoid this, the simulation needs to be synchronized whenever agents interact.
- 4.15 The solution ABCE has is the following. Interaction—messaging and as we see later the exchange of goods—only happens between sub-rounds. The messages are synchronized:
 1. agents send their message for other agents to the `abce.Communication()` process
 2. agents signal when their sub-round is finished
 3. when all agents signaled that they have finished `abce.Communication()` forwards the messages to the receiver
 4. Agents receive messages, the internal engine of `abce.Agent` receives and processes the messages, and it starts the next sub-round when messages are processed.
- 4.16 For the modeller, this is much less complex: a message or an object is sent, and in the following sub-round it arrives with the recipient. If an agent asks a question at sub-round t , his answer would arrive at $t+2$.

Physical objects

- 4.17 In the ontological part of this paper, we established two physical things that can happen to objects: they can be transformed or exchanged. ABCE handles this all automatically. The modeller just has to establish the transformation functions at the beginning of the simulation.^[10]
- 4.18 All physical objects are possessed by agents.^[11] Agents can transform them by instructing ABCE to do so. `(self.produce(input_vector), self.consume(input_vector))` In the case of production functions, a vector of goods is transformed into a different vector of goods according to the production function. In the consumption case they are consumed and utility, which is not a physical good, is reported to the agent.
- 4.19 Perishing, and the creation of goods that are declared to be recreated every round by `declare_perish` and `declare_round_endowment`, takes place automatically under the ABCE simulation.
- 4.20 The transfer of goods between agents is handled by ABCE. Consistency in these transfers is enforced by ABCE. In other words, if an object is sent, it gets subtracted from one agent and added to the other. However, ABCE also ensures that objects cannot be transferred twice: if an agent gives an object to another agent it is immediately taken from his possessions and credited only in the next round. In case of a sale offer, or analogously a purchase offer, the good gets committed to the sale as soon as the offer is made:

<i>sub-round 1</i>	Agent A offers the object <code>self.sell('receiver', receiver_id, 'good', quantity, price)</code>	
Agent A	<ul style="list-style-type: none"> the quantity of the good gets subtracted from his possessions immediately. 	
<i>sub-round 2</i>	Agent B accepts <code>self.accept(offer)</code>	Agent B rejects <code>self.reject(offer)</code>
Agent B	<ul style="list-style-type: none"> the money gets subtracted, immediately the good added, immediately 	

sub-
round 3
Agent A

- the money gets added to his possessions, at the beginning of the round

- the good is refunded, at the beginning of the round

`self.buy` analogously. With `quote_sell` agents can quote prices without committing a good. Once a quote is accepted, it gets transformed into a committed purchase.

Obtaining ABCE and licensing

- 5.1 ABCE can be downloaded for Linux, Mac and Windows from <https://github.com/DavoudTaghawiNejad/abce>.
 - 5.2 The documentation is available at <http://davoudtaghawinejad.github.com/abce/>.
 - 5.3 For those who are not familiar with Python Chapter 2, 3 and 5-5.5 of Diving into Python is a good introduction. ABCE has a permissive Apache 2.0 licence. That means that it can be freely used, derivative works can have any kind of licence, closed or open source. Users of ABCE are kindly asked to quote this article.
 - 5.4 It is my hope and intention that a community around ABCE will evolve. Contributions to the source code are highly welcome.
-

Notes

¹`self.sell('agent', agent_id, 'bread', quantity=2, price=2.95)`

²The fact that the goods perish or that they regenerate as plant or renewable energy and pretend goods such as labor.

³However, for the modeller's convenience, there are short-cuts and an extension of the firm class `FirmMultiTechnologies`, that supports multiple production functions.

⁴The full code of `start.py` is here . It is worthwhile looking at it even if you have no experience with python

⁵ABCE automatically makes sure that a good cannot be offered twice. There is a separate price quoting facility.

⁶For example: `if offer['price'] < 0.75: self.accept(offer)`

⁷Not to speak of completely meaningless header files.

⁸It is invoked by `abce.Simulation()`

⁹Nevertheless, for the modeller it is better to have one action per sub-round, thus for other purposes than the detailed exposition of how ABCE internals work, thinking of it as an action is better.

¹⁰The transformation functions can be changed, when for example a firm changes its production function.

¹¹In order to simulate objects that are not possessed by an agent, a nature agent can be introduced that possesses the non-possessed objects.

References

GORDER, Pam Frost (2007) *Multicore Processors for Science and Engineering. Computing in Science and Engineering*, 9:3-7. [doi:10.1109/MCSE.2007.35]

HINTJENS, Pieter (2013) *Zeromq Guide*. O'Reilly.

SAY, Jean-Baptiste (1880) *A Treatise on Political Economy; or the Production, Distribution, and Consumption of Wealth*. Philadelphia.