

# Modeling Autonomous Adaptive Agents with Functional Language for Simulations

Richárd Legéndi, László Gulyás, Rajmund Bocsi, and Tamás Máhr

AITIA International Inc.,  
Czetz János utca 48–50 1039 Budapest, Hungary  
{rlegendi,lgulyas,rbocsi,tmahr}@aitia.ai  
<http://mass.aitia.ai>

**Abstract.** The basic concept of agent-based modeling is to create adaptive agents to operate in a changing environment. Agents make autonomous decisions and modify their environment through continuous interactions. The Functional Agent-Based Language for Simulations (FABLES) is a special purpose language for ABM that is intended to reduce programming skills required to create simulations. The aim of FABLES is to allow modelers to focus on modeling, and not on programming. This paper provides an overview of FABLES, explaining the traits and the design concepts of this hybrid language that merges features of object-oriented, functional and procedural languages to provide flexibility in model design. To demonstrate some of these issues, we describe modeling with FABLES via the popular El Farol Bar problem from a user perspective, by means of example.

**Key words:** functional programming, agent-based simulations, multi-formalism, El Farol Bar problem

## 1 Introduction

Creating agent-based models is quite a challenging task. It requires considerable knowledge in the abstraction of the analyzed real-world phenomena (*modeling*), and expert skills in at least one programming language as well. Without a deep insight in a proper programming language, the modeler is not capable to implement the model as desired, and it may not behave as expected. Solving these issues usually consumes a lot of resources and requires a great effort.

ABM is an interesting field for social scientists, economists and sociologists as well [1, 2]. It is a great computational model to find emergent behaviour that propagates from lower (micro) level interactions to higher (macro) level regularities in the formalized phenomena. The method simulates individuals (connected autonomous adaptive agents) in the examined system [3, 4]. Accordingly, the model of the El Farol Bar Problem [5], as we discuss in this paper, was also created to investigate autonomous individuals with bounded rationality in economics.

Researchers interested in such problems usually do not possess the necessary programming skills. As an alternative way, they may use the help of a technical

expert, but since the scientist’s cognitive model of the simulation is very likely to differ from the programmer’s cognitive model, it could lead to several pitfalls – and the communication between the partners requires exceptional care.

FABLES was specially designed to require minimal programming skills, as its formalism is similar to the mathematical formalism used in publications in the subject. It is intended to facilitate the concise and efficient definition of agent-based models. It combines the strengths of functional and imperative programming with object-oriented paradigms, providing unique means to implement agent-based simulations.

### 1.1 Motivation

The main motivation behind FABLES was that the recent tools for agent-based simulations are mostly too complex for social scientists. Various software tools and modeling environments already exist to support modelers. However, some of these tools are only application programming libraries and the users are forced to learn the corresponding programming languages before they could be used effectively (e.g. Java in the case of Repast J [6], Objective-C for Swarm [7]).

On the other hand, there are several agent-based modeling environments (like NetLogo [8] and AnyLogic [9]). These tools offer easy-to-use interface commonly allowing graphical model building, however, without extending the functionalities of the environment (i.e. writing some parts of the simulation outside of the simulation environment and import them as external resources) their usage may limit the ‘space’ of possible simulations.

Most recent tools allow full control over the simulation with visual model building (Repast Symphony [10], metaABM [11]), eliminating the dependency on a programming language. They provide a natural way to construct simulations by exploiting the opportunities of graphical model building.

With FABLES, we have chosen a different approach. We didn’t want to eliminate the process of model implementation – we believe the description of the model should have a precise semantic to reflect the cognitive model of the researcher. The formalism used for model description should be close to the mathematical formalism because it is natural for the scientists. However, we tried to reduce the efforts required to perform any other side tasks with automatised components (like several wizard dialogs to create visualization for the model).

Our primary goal with FABLES was to create a language that:

- requires minimal programming skills,
- is able to describe the model focusing on the nature of the model, and leaves the implementation to the compiler,
- is general enough to accommodate possibly any agent-based model, but should focus on the common techniques and methods,
- supports the creation, observation and control of simulations with the help of interactive wizards,
- has a formalism similar to the mathematical formalism used in publications related to agent-based modeling. In these papers there’s no space to publish algorithms and thus models are described by formulas and quantors.

We have chosen a basically functional language, because the syntax is close to the formalism of mathematical functions. To be able to control the simulation flow we allow sequential execution in specific parts of the model. The user can create special structures, *schedules*, that are used to define the events and their dispatched order. In addition, to describe agents in the model we introduced *classes*. With classes one can specify both the attributes and behaviour of the agents.

## 2 The El Farol Bar Problem

In this section we introduce the El Farol Bar problem [5] that is used in the article to discuss language concepts of FABLES. We created the implementation of the model based on the NetLogo version [14], which may give a good opportunity to compare the results<sup>1</sup>. This model is basically simple, but it is complex enough to demonstrate the work-flow with FABLES, and it provides a good possibility to show how the user may create agent-based simulations within this environment.

In the center of the problem is a popular bar in Santa Fé that is regularly visited by the workers of the Santa Fé Institute. There is a fixed population ( $N = 100$ ), and at every Thursday each worker decides independently to either visit the bar with Irish music or stay at home.

Unfortunately, the bar is a too small and the atmosphere is relaxing only if not too many people shows up. In the original problem, if less than the 60% of the population goes to the bar, they will enjoy their stay there. If more than 60% of the population visits the bar, they will have a worse time due to overcrowdedness than if they stayed home. Agents have to decide about their actions at the same time simultaneously, and they do not know anything about the other agents' opinions. Hence they cannot wait and see if it is worth visiting the bar.

One of the aspects of the problem is that there is no deterministic pure strategy for the agents: they have to use mixed strategies. If each agent use the same deterministic strategy pattern and decide in the same way, it would certainly mean they do the same actions as the others and come to the same bad decision. For instance, if a strategy implies that the bar will not be crowded, everyone (following the same strategy) goes to the bar thus it becomes crowded. Otherwise, if the strategy implies that the bar will be crowded, nobody visits the bar thus it remains empty.

To solve this problem, we can supply agents with a limited personal memory to store the results of the used strategy of the past few weeks. Using this memory agents can develop different strategy patterns and adapt to each other, giving the possibility to maximize the results of their efforts.

The fact that Yi-Cheng Zhang and Damien Challet conceived Minority Games<sup>2</sup> (players who decide to be on the side that is in minority win) from this

<sup>1</sup> The full source code of the implemented FABLES model can be downloaded from the MASS Model Library at <http://mass.aitia.ai/downloads/model-lib>

<sup>2</sup> See <http://www.unifr.ch/econophysics/minority> for detailed information about research on minority games.

model, and that there are mathematical versions available in [12] and [13] makes this small model even more interesting. Fogel et al. also built a version of this model using a genetic algorithm [15], and Duncan describes the model in the context of reinforcement learning [16]. The reference implementation of the model was used to investigate if there is a connection between the agents' computational efforts and the bar attendance level [17].

### 3 Modeling with FABLES

There are several different implementation strategies for ABM [18]. In the following sections we are going to build up the model as follows:

1. The modeler specifies global properties of the model. These may be parameters, variables, global relations (facts), etc. specified in a declarative way.
2. The modeler specifies the basic structure of the agents used in the simulation, including their own local properties (variables, behavioral functions, etc.). Agents are considered independent objects, hence they can be defined with the use of classes.
3. The model dynamics and agent interactions may be specified with the use of schedules. FABLES uses the discrete time, discrete event paradigm for its scheduler, and to allow the user to implement action sequences.
4. The creation of visual components and charting is performed with automated tools. The initialization and execution of the model is performed in an enhanced version of the Repast J interface, featuring:
  - easy to use interface allowing simple initialization and customization of the model,
  - direct point and click activation of functions for observation and debugging reasons,
  - automated integration of charts, and
  - video recording capabilities and direct export of chart data to CSV files.

#### 3.1 Basic Structure of the Model

**Parameters** The first step is to create a FABLES model. The syntax is similar to the class definitions of other programming languages: it may contain constants, variables, functions, agent definitions (other classes), etc. We define our model *ElFarol* with the **model** keyword and with some global parameters:

**Listing 1.1.** Basic Components

```

1 model ElFarol {
2   param numberOfAgents = 100, overcrowdingThreshold = 60
3     memorySize = 5, numberOfStrategies = 10 ;
4 };

```

Parameters are defined with the **param** keyword. They are constant expressions, but their value may be altered before model initialization. In this case, we have defined *numberOfAgents* to specify the number of agent instances during the simulation run. The parameter *overcrowdingThreshold* declares the threshold value for the occupancy in the bar. Moreover, we define *memorySize* to denote how many preceding weeks agents can remember, and *numberOfStrategies* to specify how many strategies they can use.

**Variables** In FABLES, we consider simulations as state-transitions. The state of the simulation is the state of the model and the state of its components (agents). We have introduced variables into the language to make it convenient to handle these states.

In the El Farol model, we keep track of the last few weeks' history of occupancy, to allow agents to determine their best strategy. To specify this history, we introduce a new variable with the **var** keyword:

**Listing 1.2.** Variable definition

```
1 var history ;
```

The modeler is not required to declare types since they can be inferred from assignments. By sorting out redundant type definitions, we believe the syntax of the language becomes extremely clear and simple for a beginner user.

**Set and Sequence Expressions** In FABLES, both set and sequence types are native components of the language, and there is a huge set of built-in functions to give the opportunity to use them effectively. Basically, there are two kinds of sequences in FABLES: intervals and composite sequences.

Intervals may be defined with the [ and ] characters. The definition  $[a..b]$  denotes a closed interval from  $a$  to  $b$  containing the elements  $a, a+1, a+2, \dots, b$ . Composite sequences have a more general construction. We tried to make the syntax of the language as close to the formalism used in publications in the subject of ABM as possible. Composite sequences may be created with generators and a filter in the form of  $[< element > : < iteration_1 >, < iteration_2 >, \dots \text{ when } < condition >]$ . For example, the expression of  $[n^2 : n \text{ is } [1..10]]$  means a sequence of numbers in the form of  $n^2$ , where  $n$  denotes the numbers of  $1, 2, \dots, 10$ .

**Listing 1.3.** Variable definition

```
1 var history := [ discreteUniformFromTo(1, numberOfAgents)
2 : i is [ 1 .. 2 * memorySize ] ] ;
```

In Listing 1.3 we defined a variable with a default value (its value is assigned to the variable at model initialization). Function *discreteUniformFromTo()* cre-

ates random numbers between 1 and *numberOfAgents*, consequently, the value of *history* is a sequence containing exactly  $2 * \textit{memorySize}$  random elements<sup>3</sup>.

### 3.2 Defining Agents

In the FABLES code we can define a new agent structure with the **class** keyword. Just as the model, a class is also considered as an aggregate object: it may contain other constants, variables, functions, etc.

In the current implementation of the El Farol Bar model, each agent has a set of randomly initialized autoregressive models (sequences of AR coefficients) that are not fitted. We declare three variables for the agents: the sequence of *strategies*, the *best strategy* found so far, and a boolean flag that indicates its decision about *attendance*.

**Listing 1.4.** Agent definition

```

1 class Agent {
2   var strategies ;
3   var bestStrategy ;
4   var attend := false ;
5 };

```

Agents are initialized with a random strategy pattern. A convenient way to handle this is to declare a global method that creates a series of random strategies:

**Listing 1.5.** Sequence construction

```

1 randomStrategies () =
2 [ [ uniform(-1, 1) : j is [1..memorySize+1] ]
3 : i is [1..numberOfStrategies] ] ;

```

Listing 1.5 specifies a function that returns a sequence of strategies (a strategy is another sequence filled with random numbers created by the *uniform()* built-in function from the interval  $(-1, 1)$ , having a size of *memorySize*), containing *numberOfStrategies* elements. Therefore we can assign the following default values to the agent variables:

**Listing 1.6.** Default Values for Agent Variables

```

1 var strategies := randomStrategies () ;
2 var bestStrategy := strategies (0) ;

```

This means that each agent is going to have a set of random strategies by default, and they declare the first strategy as their best strategy in the beginning

<sup>3</sup> In this sequence construction, *i* is a new loop variable that even might have been used to create the elements of the collection. In the current definition it was required only to specify the number of elements in the sequence.

(sequence constructions could be indexed from 0 to acquire the element at the specified position). To get the current attendance level of the bar, we define the following global *attendance()* function:

**Listing 1.7.** Attendance Level

```
1 attendance() = count( [ a.attend : a is Agent ] );
```

The function *count* returns the number of *true* elements in the specified collection (which contains the *attend* logical variable of all of the agents). Therefore its value equals to the current attendance of the bar.

**Defining Agent Behavior** We have finished declaring the basic structure of the model, hence we can start implementing the behavioral definitions. In each simulation step agents have to rethink their beliefs by evaluating their best strategy depending on their current predictions. Therefore we declare function *updateStrategies()* for the agents.

In general, functional languages do not contain block structures, however, they offer the possibility to declare local definitions. In FABLES, local definitions may be declared for a given function with the use of the **where** keyword. Local definitions are constants that are visible only in the scope of the corresponding function and may be used to shorten the definition of the function.

In the following definition of *updateStrategies()* we define three local constants: *threshold*, *predictions* and *strategyPos*. The *threshold* local constant is used to make agents keep their actual strategy if the overall rating of the currently evaluated best one is lower than a specified value. The sequence *predictions* contains the sum of differences between the actual and the predicted attendance level (evaluated by the function *evaluateStrategy()* which is described soon) for each of the preceding weeks (as far as the agents' memory allows to determine it) for each of the agents' strategies. We consider the strategy with the minimal difference from its prediction as the best one, and *strategyPos* denotes its index in the strategies sequence.

**Listing 1.8.** Updating Agent Strategies

```
1 updateStrategies() =
2   ( predictions( strategyPos ) < threshold ) =>
3     bestStrategy := strategies( strategyPos )
4   where (
5     threshold = memorySize * numberOfAgents + 1 ,
6     predictions = [ sum( [ evaluateStrategy( week, str )
7                       : week is [1..memorySize] ] )
8                   : str is strategies ],
9     strategyPos = minPlace( predictions )
10  ) ;
```

The body of the function is a conditional expression. In FABLES, the general form of creating conditional functions is (*< condition<sub>1</sub> >*) => *statement<sub>1</sub>* | (<

$condition_2 > ) => statement_2 \mid \dots \mid otherwise => statement_N$  where some of the branches may be omitted, and it means "If  $condition_1$  is true then perform  $statement_1$ . Otherwise, if  $condition_2$  is true then perform  $statement_2$ , ... If none of the above conditions were true then perform  $statement_N$ ". The function `updateStrategies()` assigns a new value to `bestStrategy` if the prediction of the currently found best strategy is better than the specified threshold value (which is  $memorySize * numberOfAgents + 1$ ).

We also have to define how the function `evaluateStrategy()` works. It determines the current prediction for the specified week and strategy:

**Listing 1.9.** Agent Strategy Evaluation

```

1 evaluateStrategy( week, str ) =
2   abs( currentAttendance - prediction )
3   where (
4     currentAttendance = history( week - 1 ) ,
5     prediction = predictAttendance( str ,
6       [ history(j): j is [week..(week+memorySize-1)] ] )
7   ) ;

```

The value returned by the function is the difference between the current attendance level and the prediction with the given strategy for the given week, as the `abs` function returns the absolute value of the difference. The value of `currentAttendance` is the week<sup>th</sup> element in the history, and prediction is determined by the `predictAttendance` function that requires the current strategy and the list of attendance values preceding the week<sup>th</sup> element of history. Formally, the function should return the following  $p(t)$  prediction described in the original model:

$$p(t) = w(t) + \sum_{i=t-M}^{t-1} w(i) * a(i - 1)$$

where  $t$  is the current time,  $w(i)$  is the weight,  $a(i)$  is the attendance level and  $M$  is the memory size. In the FABLES implementation of the model, strategies represent weights, so for a specified strategy and subsequence of history the implementation of the function is as follows:

**Listing 1.10.** Predicting Attendance

```

1 predictAttendance( strategy , subhistory ) =
2   strategy(0) + sum( [ strategy(i) * subhistory(i-1)
3     : i is [1..memorySize] ] ) ;

```

### 3.3 Defining Simulation Events and Agent Actions

The final step to implement the simulation is to define the dynamics of the model. Describing the dynamics of a complex system often causes the modelers several

problems. The situation is getting worse when it comes to agent interactions. In FABLES, there are basically two ways to describe simulation actions. One group of the actions have to be performed for proper initialization of the model, and the other group is used to describe the runtime dynamics of the simulation.

For initialization, we have *startUp* blocks both for agent classes and the model. The *startUp* block of the model is executed when the simulation is started; the *startUp* block of an agent is executed when the agent is created.

When the simulation is started, we have to instantiate the specified number of agents:

**Listing 1.11.** Model Initialization

```

1 startUp() {
2   for each i in [1..numberOfAgents] do create Agent ;
3 };

```

When an agent is created, we have to call its *updateStrategies()* function to allow it to initialize its *bestStrategy* attribute.

**Listing 1.12.** Agent Initialization

```

1 class Agent {
2   ...
3   startUp() {
4     updateStrategies() ;
5   };
6 }

```

To describe the simulation dynamics, FABLES uses *schedules*. In general, schedules may be cyclic or acyclic, named or anonymous ones, and both the agents and models may have separate and multiple schedules. Acyclic schedules are executed only once, while the events of a cyclic one is executed successively with a specified cycle rate. The difference between named and anonymous schedules is that named ones can be dynamically manipulated during runtime: they may be stopped, restarted, their events may be altered, etc.

Inter-agent communication may be implemented in schedules as well. FABLES does not have a fixed communication scheme, agents can interact through function calls in schedules. In the El Farol Bar model, there is no direct communication, thus the common variable *history* is the only way they interact with each other using a blackboard like approach.

For the El Farol simulation, we need two schedules: one for the agents to make them determine if they would like to attend the bar; the other one is a global one to update the *history* variable that the agents use to predict their optimal strategy.

**Listing 1.13.** Defining Agent Actions

```

1 class Agent {
2   ...
3   updateAttendance() =
4     attend := prediction <= overcrowdingThreshold
5     where (
6       prediction = predictAttendance( bestStrategy ,
7         [ history(i) : i is [0..memorySize-1]])
8     ) ;
9
10  schedule cyclic 1 {
11    1 : updateAttendance() ;
12    2 : updateStrategies() ;
13  };
14 };

```

**Listing 1.14.** Defining Global Events

```

1 schedule cyclic 1 {
2   1.1 : history := [ attendance ] ++
3     [ history(i) : i is [0..size(history)-2] ] ;
4 };

```

### 3.4 Visualization

The FABLES Integrated Modeling environment has a powerful component called the Charting Wizard. This tool allows the modeler to create visualization for a simulation via a point and click interface. The wizard offers a selection of charts, capable of displaying derived simulation data, and contains several built-in statistics for data processing.

For the discussed El Farol model, we create a simple Time Series chart from the agents' attendance variables. The usage of the Charting Wizard is self-explanatory, we have to select the *attendance* variable in the drop-down list and add it to a Time Series chart (the wizard is not discussed here in detail due to limitations of space). The results can be seen in Figure 1.

## 4 Conclusion and Future Works

We have developed the above described language for agent-based simulation, its integrated development environment and a built-in observation wizard. During the development we have tried to automate as many tasks as possible, allowing the user to concentrate on models instead of the coding details and the implementation of visualisations.

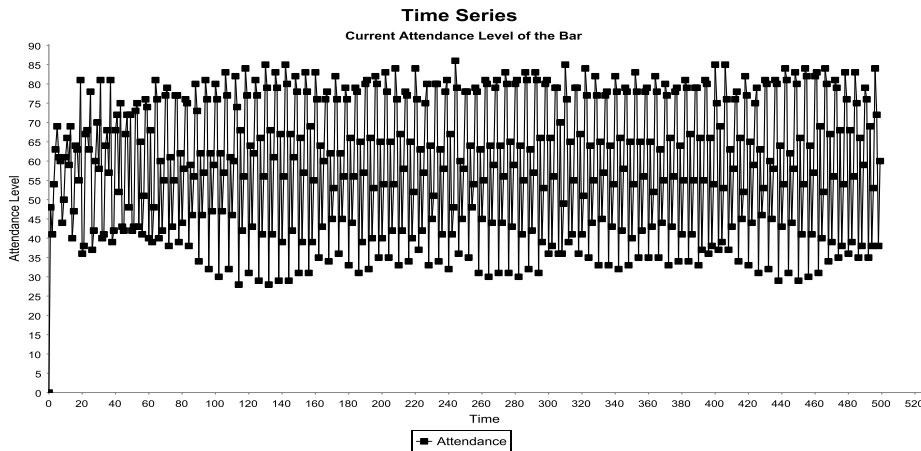


Fig. 1. The results of the implemented simulation where  $N = 100$  with an attendance level of 60. The figure shows this simple model has the fluctuation of the original model.

We discussed the development of FABLES models as a four step process: defining global variables, defining agents, adding interactions and dynamics, and creating charts to visualize the simulations. A working reproduction of the original El Farol Bar model was described that may be compared with the original version [14]. Slight runtime differences may occur because the generated random values. The model consists of pure mathematical definitions, and no additional coding was required. Besides, visualization for the model was created with a few clicks. In addition, there are several extensions to the original model. For instance, in some versions agents are permitted to communicate with a set of other agents to discuss their opinions. This could be easily implemented in FABLES as well.

We are in the process organizing and evaluating user tests, hence we hope to have the necessary feed-back to further improve FABLES' built-in functions. Optimizing the generated code is also an important ongoing task.

**Acknowledgements.** The work reported here benefited from a grant of the Hungarian Government (grant #GVOP-3.2.2-2004.07-005/3.0). The partial support of the European Commission via the FP6 STREP projects QosCosGrid (contract #033883) and EMIL (contract #033841) is also gratefully acknowledged.

## References

1. Epstein, J.M., Axtell, R.L.: Growing Artificial Societies: Social Science from the Bottom Up, volume 1 of MIT Press Books. The MIT Press, January 1996.
2. Samuelson, D.A., Macal, C.M.: Agent-based Simulation Comes of Age. August 2006.

3. Axelrod, R., Tesfatsion, L.S.: A guide for newcomers to agent-based modeling in the social sciences. Staff General Research Papers 12515, Iowa State University, Department of Economics, March 2006.
4. Tesfatsion, L.S.: Agent-based computational economics: Modeling economies as complex adaptive systems. Staff General Research Papers 12974, Iowa State University, Department of Economics, August 2008.
5. Arthur, W.B.: Inductive Reasoning, Bounded Rationality and the Bar Problem. Working Papers 94-03-014. Santa Fe Institute, New Mexico, USA, 1994.
6. North, M.J., Collier, N.T., Vos, J.R.: Experiences Creating Three Implementations of the Repast Agent Modeling Toolkit, *ACM Transactions on Modeling and Computer Simulation*, Vol. 16, Issue 1, pp. 1-25, ACM, New York, New York, USA (January 2006).
7. Minar, N., Burkhart, R., Langton, C., Askenazi, M.: The Swarm simulation system: A toolkit for building multi-agent simulations. Working Paper 96-06-042, Santa Fe Institute, Santa Fe, 1996.
8. Wilensky, U. 1999. NetLogo. <http://ccl.northwestern.edu/netlogo/>. Center for Connected Learning and Computer-Based Modeling, Northwestern University. Evanston, IL.
9. Borshchev, A., Karpov, Y., and Kharitonov, V.: Distributed simulation of hybrid systems with AnyLogic and HLA. *Future Gener. Comput. Syst.* 18, 6 (May. 2002), 829-839. 2002.
10. North, M.J., Tatara, E., Collier, N.T., Ozik, J.: Visual Agent-based Model Development with Repast Symphony, *Proceedings of the Agent 2007 Conference on Complex Interaction and Social Emergence*, Argonne National Laboratory, Argonne, IL USA (November 2007).
11. Parker, M.: MetaABM Agent-based Modeling Software. <http://metaabm.org> 2009.
12. Challet, D., et al: Shedding light on El Farol. *Game Theory and Information* 0406002, EconWPA, 2004.
13. Challet, D., et al: *Minority Games: Interacting Agents in Financial Markets*, Oxford University Press, USA, 2005.
14. Rand, W., Wilensky, U. (2007). NetLogo El Farol model. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL. <http://ccl.northwestern.edu/netlogo/models/ElFarol>.
15. Fogel, D.B., Chellapilla, K., Angeline, P.J.: Inductive reasoning and bounded rationality reconsidered, *IEEE Transactions on Evolutionary Computation*, 1999, v3n2, p142-146.
16. Whitehead, D.: The El Farol Bar Problem Revisited: Reinforcement Learning in a Potential Game. ESE Discussion Papers 186. Edinburgh School of Economics, University of Edinburgh, 2008.
17. Rand, W. and Sondahl, F. : The El Farol Bar Problem and Computational Effort: Why People Fail to Use Bars Efficiently. *Agent 2007*, November 15-17, Evanston, IL, USA, 2007.
18. Gulyás, L.: On the transition to agent-based modeling: Implementation strategies from variables to agents. *Social Science Computer Review*, 20(4):389-399, 2002.