

Geographic Automata Systems and the OBEUS software for their implementation

Itzhak Benenson^{1,2}, Vlad Kharbash¹

¹Revson Environment Simulation Laboratory of the Porter School of Environmental Studies,

²Department of Geography and Human Environment,

University Tel Aviv, 69978, Tel-Aviv, Ramat-Aviv

<http://www.tau.ac.il/~benny>

benny@post.tau.ac.il

Geographic Automata Systems and the OBEUS software for their implementation

Abstract

Recently, an automata-based approach to portraying urban systems has been proposed. It is based on the concept of Geographic Automata System (GAS) (Benenson and Torrens 2004). According to the GAS concept, any urban system consists of unitary automata of different types, fixed and non-fixed. Non-unitary urban entities are considered as ensembles of unitary automata that satisfy a predetermined set of predicates. It is shown that the majority, if not all, Cellular Automata and Multi-Agent urban models proposed until now can be reformulated in a GAS framework.

In parallel, the above object-based view of urban systems is implemented as a software environment (Benenson, Aronovich et al. 2004). After the first, experimental, stage, a user-friendly shareware version of the software (OBEUS) is developed, adjusted to the GAS theory, and presented in the paper.

From arbitrarily spatial units to geographic objects

During the first two decades of their development (1960s-1980s), urban and regional modeling and simulation were almost exclusively based on cybernetics - black-box - views of systems. Beginning from 1990s, Cellular Automata (CA) and, recently, Multi-Agent System (MAS) model approaches took the stage (Benenson and Torrens 2004). CA and MAS models flourished during last decade (Batty 1997; Clarke, Hoppen et al. 1997; Portugali 2000; White and Engelen 2000; Benenson, Omer et al. 2002). Compared to a regional modeling approach, standard CA and MAS are much more intuitive, but are constrained by regular partition of space, which engenders problems regarding representation of the network elements, spatial entities of different size, spatial relations between mobile and immobile objects, and so forth. Recently introduced Geographic Automata Systems represent geographic space by means of the objects operating there (Benenson and Torrens 2004) and resolve this limitation.

GAS substitutes partition of space, as a conceptual basis, by discrete *automata objects* that directly represent real-world urban entities. Automata can be of different nature, spatial extension, and hierarchical rank; they can represent spatially fixed and non-fixed entities. Spatial relationships between automata would thereby determine the structure of urban space. Although the object approach to the representation of urban reality has been mentioned in the literature (Erickson and Lloyd-Jones 1997; Semboloni 2000; Galton 2001), the idea has remained inchoate to date.

The recent GIS boom provides strong empirical support for a GAS approach: Layers of urban GIS are nothing but collections of urban objects of the same kind. A next step is simply to build GAS-based software, and this paper presents the latest progress in developing OBEUS (Object-Based Environment for Urban Simulation) system that makes the GAS view operational.

We assume below that the reader is familiar with the backgrounds of Relational DBMS and Entity-Relational data Model (ERM) (Howe 1983) and the Object-Oriented (OO) programming paradigm (Booch 1994).

Short introduction to Geographic Automata System (GAS)

The GAS concept considers urban infrastructure and social objects as *spatially located* discrete automata. Generally, automata A are characterized by (multidimensional) states \mathbf{S} , and the states of unitary automata change in discrete time t depending on input \mathbf{I} , following state transition rules \mathbf{T} . To specify the geographic

automata (GA), Benenson and Torrens (2004) re-interpret **S**, **I**, and **T**, to enable the explicit consideration of *space* and *spatial behavior*:

States S:

- In addition to non-spatial components, the state of GA **G** contains its locations.
- In addition to coordinate location, **G** can be located indirectly, by pointing to the other GA.

Input I:

The input information **I** comes from many sources. Benenson and Torrens (2004) consider **I** as defined by **G**'s neighbors, just as in CA scheme. We extend this definition:

- **G** receives input information **I** from the automata it is *related to*. The GAS approach thus focuses on **G**'s spatial relationships.

Specification of transition rules - T:

- Transition rules of GA **G** specify the changes of non-spatial components of a state vector - *state transition rules* and location - *movement rules*.
- The *relationships* of **G** can also change in time; these changes are governed by *relationship transition rules*.

Benenson and Torrens (2004) consider Geographic Automata System **G** as consisting of different *types* of automata **K**. A set of GAS states **S** consists then of the subsets of states **S^k** characteristic of automata of each type **k** ∈ **K**. A set of *state transition rules* **T_S**, determine how automata states change over time. The *geo-referencing conventions* **L** determine how automata are located in space, and the *movement rules*, **M_L**, govern changes in their location. The relationships **R** and the *relationship transition rules* **N_R** specify how automata relate to the other automata and how these relationships change in time.

Altogether, a GAS **G**, may be defined as consisting of seven components:

$$\mathbf{G} \sim \langle \mathbf{K}, (\mathbf{S}, \mathbf{T}_S), (\mathbf{L}, \mathbf{M}_L), (\mathbf{R}, \mathbf{N}_R) \rangle \tag{1}$$

And for geographic automaton, **G** characterized at t by state **S_t**, location **L_t**, and relations **R_t**, the state, location and relations at time t + 1 are determined by the transition rules **T_S**, **M_L**, **N_R** as follows:

$$\mathbf{T}_S : (\mathbf{S}_t, \mathbf{L}_t, \mathbf{R}_t) \rightarrow \mathbf{S}_{t+1}$$

$$\mathbf{M}_L: (S_t, L_t, R_t) \rightarrow L_{t+1} \quad (2)$$

$$\mathbf{N}_R: (S_t, L_t, R_t) \rightarrow R_{t+1}$$

From Geographic Automata System to a software

Implementing GAS view, we have to translate all the components of (1) into software objects and methods:

Automata of a given type $k \in K \rightarrow$ Instances of *population class*

Urban objects always deal with information found on levels above the individual level, and OBEUS **Population** is a container for these meta-data.

Individual automata of type $k \rightarrow$ Class of *entities of a type k*

At an abstract level, OBEUS considers unitary **entities**. (Benenson, Aronovich et al. 2004), and distinguishes between **fixed** and **non-fixed entities** (Benenson and Torrens 2004). Fixed entities — buildings, parks, road links, traffic lights, etc. — usually do not change location being established, while non-fixed entities may do that.

There is principal difference in the way fixed and non-fixed entities are geo-referenced. Fixed objects are **located directly**, in a manner similar to vector GIS, that is, by means of a coordinate list. Consequently, the location of the fixed object is one of components of its state **S**. Non-fixed urban objects are located **by pointing** to one or several fixed ones. For example, householders are located by pointing to their habitats (Benenson, Aronovich et al. 2004). It is easy to notice, thus, that geo-referencing by pointing is actually a relationship.

In OBEUS we assume that objects of a given type $k \in K$ are either fixed or non-fixed, and we are not aware of 'natural' objects that cannot be easily classified in this respect. Consequently, the second pair — (**L**, **M_L**) — of the GAS definition (1) either becomes special case of the (**S**, **T_S**) pair in the case of fixed objects or special case of the second pair (**R**, **N_R**) in case of non-fixed objects. GAS is thus restricted in OBEUS to

$$\mathbf{G} \sim \langle \mathbf{K}, (\mathbf{S}, \mathbf{T}_S), (\mathbf{R}, \mathbf{N}_R) \rangle \quad (3)$$

The specificity of the GAS software rests on automata *relationships*, and their priority is the core of OBEUS.

Relationships between automata \rightarrow Class of *relationships*

OBEUS considers relationships between entities explicitly, as a software object additional to the entities, which, as usual in ERM, can have their own properties.

To illustrate the use of relationships, the decision to sell or develop a lot might depend on (a) the neighborhood's development potential (**lot-lot** relationships between **lot** entities) and (b) the relationship between the landowner and the land estate (**landowner-estate** relationship).

To ensure consistency when applying transition rules T_S , and, especially N_R , we impose essential limitations on the semantics of the relationships in OBEUS:

- we assume that relationships between fixed objects are also fixed;
- we assume a **leader-follower pattern** of relationship update (Noble 2000). According to this pattern, non-fixed objects have exclusive update rights, while the related fixed objects can only query the relationship's attributes;
- we do not permit direct relationships between non-fixed objects and define relationships between them in a transitive way (OBEUS presents built-in functions for that)

The distinction between objects and relationships offers immediate advantages in applications. In general, it ensures that the algorithm of updating will result from *intentional* thought. Specifically, it provides an identical framework for CA and explicit GIS-based land-use models, both of which are based on neighborhood relationships. The only difference between CA models and GIS-based irregular partitions of a plane is the variation in degree of neighborhood relationship from parcel to parcel (Flache and Hegselmann 2001; Benenson, Omer et al. 2002).

Transition rules T_S and R_N → Automata behavior rules

Geographic automata 'behave', and in terms of OBEUS their behavior results in changing states and altering relationships with the automata they are related to. The behavioral rules are represented in OBEUS by *class methods*, which are formulated by the user.

In OBEUS, we separate between **assessment rules**, aimed at estimating the parameters agents react to and actual **behavior rules** that describe the act of the behavior itself.

Self-organization in OBEUS

OBEUS considers *collective phenomena* from the perspective of the object-based approach. It introduces ensembles of unitary entities (patterns) and implements the specific case of 'foreseeable' self-organization, which can be recognized by the user-defined *a priori* set of predicates. This structure is yet sufficient to recognize the typical geographic spatial self-organization, as areas populated predominantly by individuals of a certain socio-economic group (poor, rich, immi-

grants) (Benenson and Torrens 2004), as well as for the deltatrons (Clarke 1997; Candau, Rasmussen et al. 2000), etc.

Synchronization of events in OBEUS

The famous patterns of the Game of Life disappear when the asynchronous updating scheme is employed (Schonfisch and de Roos 1999). This is an example of a general problem - management of events that occur simultaneously yet influence one another induces synchronization problems of various kind, which are intensively discussed in CA, MAS, temporal GIS and other research dealing with discrete-time object-based systems (Berec 2002). Conceptually, there is no general solution to the synchronization problem (Zeigler, Praehofer et al. 2000) and OBEUS implements two synchronization modes:

Synchronous: In this mode, the state transition rules are applied to all entities simultaneously. The calling order of the objects of a certain class has no influence on the model outcome.

Asynchronous: In this mode, objects change in turn, with each observing the urban reality as left by the previous object. In asynchronous mode the modeler define an order of object actions in advance: random sequence, sequence in order of some characteristic, etc.

Relationships transition rules are always applied asynchronously, no matter which mode is chosen for objects updating.

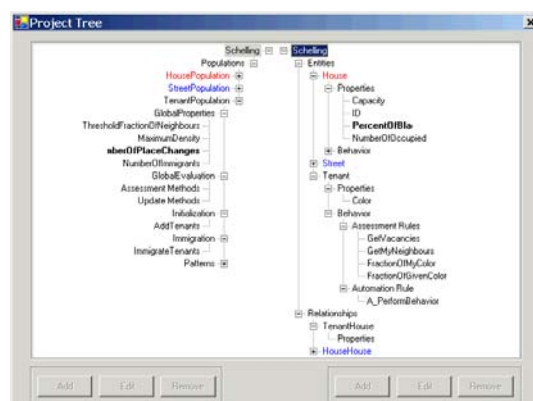
User's view on OBEUS

To formulate the model, a user of OBEUS should define **Model Tree**, **Synchronization Scheme**, and **Behavioral Rules**. The latter is the most important part of any model, and we incorporate into OBEUS the Borland C# compiler as an engine for formulating these rules. The universality of OBEUS is provided by the set of **Built-in methods** aimed at operating with the entities and relationships.

Model tree

Through the Model Tree, a user defines classes of automata, relationships, their properties, and properties of the populations of automata (Figure 1).

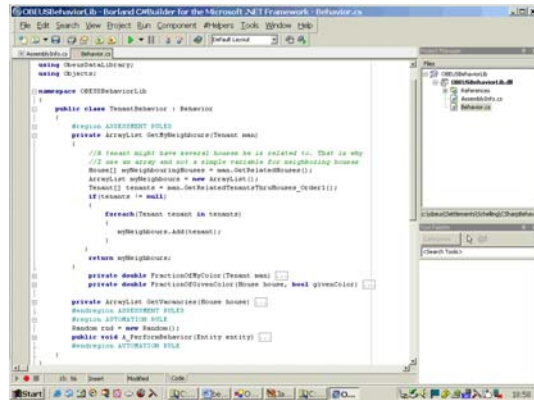
Figure 1: Model Tree for the generalized Schelling model



Behavioral Rules

For the modeler, the core of the model design is in use of a .NET Borland C# environment for defining **Assessment Rules** and **Behavioral Rule** (Figure 2).

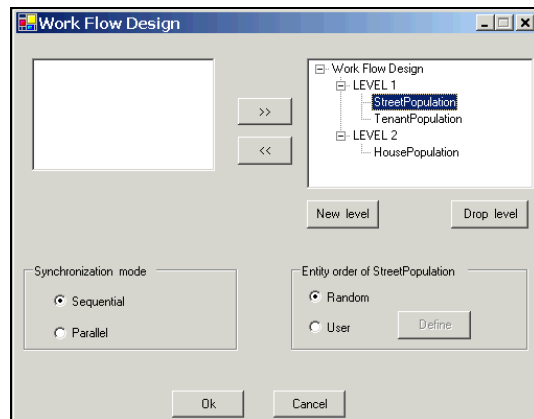
Figure 2: Behavioral component of the generalized Schelling model



Synchronization scheme

The synchronization mode is the third basic element of OBEUS model (Figure 3).

Figure 3: Dialog box for establishing synchronization mode



Built-in methods

Implementation of GAS theory demands instantaneous access from the software objects representing model entities to those representing relationships, and visa versa. In the same time, to satisfy OOP requirements, the methods responsible for this functionality must be encapsulated within entities' and relationships' classes. OBEUS resolves these demands by automatic construction of the access methods with every new entity and/or relationship class defined. The idea can be illustrated with the Schelling model, which needs at least **Tenant** and **House** entities and **TenantHouse** relationship classes. The automatically constructed methods of the **Tenant** class are such as **Tenant.GetRelatedHouses()**, **Tenant.SetRelationship(House house)**, **Tenant.RemoveRelationship(House house)**, etc. The built-in methods are available to model developer just as the other methods of C# developer environment.

Output

OBEUS output exploits its GIS/Database background. The attributes and features requested by the modeler are presented on the screen (Figure 4) and stored in the database for further processing.

Figure 4: Typical OBEUS output for GIS-based generalized Schelling model



To demonstrate usage of OBEUS, we rebuild various urban high-resolution models published during last decades. We also compare OBEUS to RePast and NetLogo, acknowledged as most helpful in recent reviews (Tobias and Hofmann 2004).

References

- Batty, M. (1997). "Cellular automata and urban form: A primer." Journal of the American Planning Association **63**(2): 266-274.
- Benenson, I., S. Aronovich, et al. (2004). "Let's Talk Objects: Generic Methodology for Urban High-Resolution Simulation." Computers, Environment and Urban Systems **Forthcoming**.
- Benenson, I., I. Omer, et al. (2002). "Entity-based modeling of urban residential dynamics - the case of Yaffo, Tel-Aviv" Environment and Planning B, **29**: 491-512.
- Benenson, I. and P. M. Torrens (2004). Geosimulation: automata-based modeling of urban phenomena. London, Wiley.
- Berec, L. (2002). "Techniques of spatially explicit individual-based models: construction, simulation, and mean-field analysis." Ecological Modelling **150**: 55-81.
- Booch, G. (1994). Object-oriented analysis and design with applications. Menlo Park, Ca, Addison-Wesley.
- Candau, J. T., S. Rasmussen, et al. (2000). A coupled cellular automata model for land use/land cover change dynamics. 4th International Conference on Integrating GIS and Environmental Modeling (GIS/EM4): Problems, Prospects and Research Needs, Banff, Alberta, Canada.
- Clarke, K. C. (1997). Land Transition Modeling With Deltatrons, <http://www.geog.ucsb.edu/~kclarke/Papers/deltatron.html>. **2002**.
- Clarke, K. C., S. Hoppen, et al. (1997). "A self-modifying cellular automata model of historical urbanization in the San Francisco Bay area." Environment and Planning B: Planning and Design **24**(2): 247-261.
- Erickson, B. and T. Lloyd-Jones (1997). "Experiments with settlement aggregation models." Environment and Planning B-Planning & Design **24**(6): 903-928.
- Flache, A. and R. Hegselmann (2001). "Do Irregular Grids make a Difference? Relaxing the Spatial Regularity Assumption in Cellular Models of Social Dynamics." Journal of Artificial Societies and Social Simulation vol. 4, no. 4 **4**(4): <<http://www.soc.surrey.ac.uk/JASSS/4/4/6.html>>.
- Galton, A. (2001). "Space, Time, and the Representation of Geographical Reality." Topoi **20**: 173-187.
- Howe, D. R. (1983). Data analysis for data base design. London, Edward Arnold.
- Noble, J. (2000). Chapter 6. Basic Relationship Patterns. Pattern Languages of Program Design 4. N. Harrison, B. Foote and H. Rohnert, Addison-Wesley:73-89.
- Portugali, J. (2000). Self-Organization and the City. Berlin, Springer.
- Schonfisch, B. and A. M. de Roos (1999). "Synchronous And Asynchronous Updating In Cellular Automata." Biosystems **51**: 123-143.
- Semboloni, F. (2000). "The growth of an urban cluster into a dynamic self-modifying spatial pattern." Environment and Planning B-Planning & Design **27**(4): 549-564.
- Tobias, R. and C. Hofmann (2004). "Evaluation of free Java-libraries for social-scientific agent based simulation." Journal of Artificial Societies and Social Simulation **7**(1): <<http://jasss.soc.surrey.ac.uk/7/1/6.html>>.
- White, R. and G. Engelen (2000). "High-resolution integrated modelling of the spatial dynamics of urban and regional systems." Computers, Environment and Urban Systems **24**(5): 383-400.
- Zeigler, B. P., H. Praehofer, et al. (2000). Theory of modeling and simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems. New York, Academic Press.