5th World Congress of the International Microsimulation Association (IMA)
Luxembourg, 2-4 September 2015

The Event Pump: An Agent Based approach to Microsimulation

Peter Stephensen
DREAM

Abstract

Events are a basic element in most dynamic microsimulation models. Events are what people do: birth, death, moving etc. According to the Event-Pump Architecture, the core driver of the model is a loop called the event pump. This loop controls when and where things happen in the model. The feature is inspired by the message pump, which is a central element in how Windows is programmed (Petzold, 1998). The method is used in the microsimulation model SMILE-DK (Stephensen, 2013) , in the official Danish population forecast (Hansen & Stephensen, 2014) and in a forthcoming agent-based analysis of the Danish housing marked.

Keywords: population projections, education, household projections, housing demand, microsimulation

The Event Pump: An Agent Based approach to Microsimulation

1. Introduction

In the Danish microsimulation model SMILE a novel approach is used, called the *Event-Pump Architecture*. As in most other dynamic microsimulation models a basic element of the model is *events*. Events are what people do: birth, death, moving etc. The core driver of the model is a loop called the *event pump*. This loop controls when and where things happen in the model. The feature is inspired by the *message pump*, which is a central element in how Windows is programmed (Petzold, 1998).

2. The SMILE model

The SMILE[1] model is a Danish, dynamic, data-driven microsimulation model. The current version 3 forecasts demography, education level, socioeconomic characteristics, housing demand, income, taxation, public benefits and labor market pensions for the period 2014-2050.

The SMILE model is *dynamic* in the sense that an initial population (the entire Danish population of approximately 5.5 million persons) is forecasted into the future. Demographic events such as death, birth, immigration, emigration etc. are modelled. Projections of death probabilities are based on the Lee-Carter econometric method (Lee & Carter, 1992). The model is subdivided in 98 regions and a matching algorithm called SBAM (Stephensen & Markeprand, 2013) is used.

The modelling of *education* decisions is based on a regionally subdivided transition probabilities calculated from Danish register data and it thus forecasts education levels by employing historical educational behavior. The modelling of income and labor market dynamics is based on (Bækgaard, 2013).

The moving probability is divided by background characteristics of the household and by characteristics of the household's current dwelling. Together this results in a lot of explaining variables why the moving probability is calculated as a CTREE (Rasmussen, 2013).

The SMILE model is a *data-driven model*, based on rich Danish register data. The data cover the entire Danish population on annual basis in the period between 1986 and 2013. On each individual our dataset contains information about the person himself (gender, age, educational background, labor market participation, income etc.), the person’s family situation (single/couple, number of children living at home etc.) and information about the dwelling that the person’s household lives in (location, owner/rental status, dwelling type and size etc.). We derive data from seven different

[1] Simulation Model for Individual Lifecycle Evaluation.

sources made available through Statistics Denmark. The main data sources are the Danish Civil Registration System (*CPR-registret*), the Housing Register (*Bygnings- og Boligregistret, BBR*), the education register (*Uddannelsesregistret*) and the labor force statistics (*Registerbaseret Arbejdsstyrkestatistik, RAS*).

3. Basic concepts

The basic concepts in the development strategy have been *object orientation* and *agent based modelling*. The model is programmed in an object orientated programming language (C#). The advantages of object orientation are often described by the words *encapsulation* and *separation*. By encapsulation you mean that a lot of functionality can be hidden inside an object. That makes it easier for others to use it. This is best demonstrated by an example. Figure 1 shows the *Agent Tree*. This is basically the model. The Simulation object at the top level controls the simulation. The left branch contains the actual agents of the system: the households and the persons. The right branch contains different kinds of functionality. The Demographics object contains information on the behavior of the households and persons, and the Statistics object gathers information (output data). The Demographics object contains all the transition probabilities and (importantly) how to use them. All this functionality is *encapsulated* in the object. When programming the left branch you can use the Demographics object without actually knowing/understanding what's inside it. You could now replace the Demographics object with a Swedish version (ie. with Swedish transition probabilities) and you would have a Swedish instead of a Danish model. This example also demonstrates the related concept *separation*. It is considered as good programming practice to separate various tasks. This makes it easier to maintain code and to find errors. The structure of the model naturally separates these tasks.

In the last 10 to 15 years Agent Based Modelling has been more and more common. The basic idea is that modelling should be done at the individual level, and that interaction is important. Microsimulation and agent based modelling is related traditions but never the less different. Both traditions work at the individual level. But when it comes to interaction among the agents there are differences. The most important example of interaction in microsimulation models is the formation of couples. In many microsimulation models (also in SMILE) this problem is solved with a top-down approach. In an agent based model the approach should always be bottom-up. Agent based models are more principled, use more theory and less data. Microsimulation models are data-driven and more pragmatic.

Despite these differences SMILE is basically build as an agent based model. It is probably fair to say that microsimulation models is a pragmatic subset of agent based models. The basic argument for using an agent based modelling strategy is therefore generality. Using this approach opens up for a lot of future possibilities of interaction between the agents.

The decision to follow an agent based path let to the idea of a basic Agent object. This object is the Lego block from which the model is build. An Agent is a thing that 1) Potentially has children 2) Potentially do stuff. By 'having children' we mean pointing to other Agents. It is this property that makes it possible to build an Agent Tree like the one shown in Figure 1. By ‘do stuff’ we mean having a generic method/function that defines what to do under different circumstances. This method (a ‘function’ is called a ‘method’ in C#) is called `EventProc(idEvent)`. The input `idEvent` is an event ID that defines the current event. The generic behavior (the so called base implementation) of `EventProc` is to do nothing and just send `idEvent` to all its children Agents (if any). If you therefore sends an `idEvent` to the `EventProc` of the top Simulation object in Figure 1, the event will automatically be send down the tree an eventually reach all Agents. If you want your `EventProc` to do something (a Person to die or a Household to move), you just program this behavior before you send the event to the children Agents. Or if you know the event is of no interest for the Agents further down the tree, you can chose to stop the event. This can be a source of speed gains.

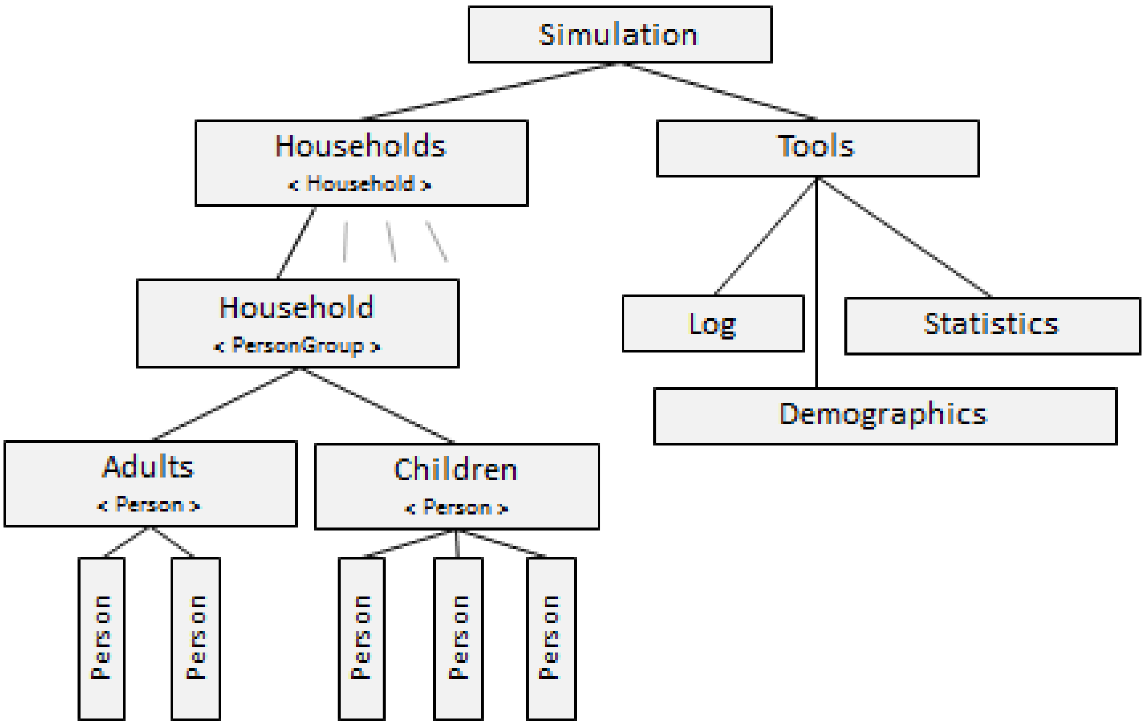


Figure 1 The Agent Tree

Having constructed the Agent tree from our Agent Lego blocks, we just need to send sequences of events down the tree to make the model run. This is done by the *Event Pump*. The Event Pump is a loop located in the top Simulation objects `EventProc`. It sends events down the tree in the correct order. Typically it repeats the same sequence of events every period.

The Event Pump got its name from the so called Windows Message Pump that is a fundamental concept in the original programming of Windows (the Win32 API, see Petzold 1998). In Windows every window is controlled by a callback function called `WinProc`. The input to this function is (among other) a parameter `iMsg`. This is a message ID that defines which message the window receives (pressing a button, moving the mouse over the window etc.). The Message Pump controls the order of these `iMsg`'s. The parallel to our approach is obvious.

The Event Pump Architecture has proven itself very useful and flexible. It is easy to make changes to the model and we have yet to face microsimulation/Agent-based issues that do not have an Event Pump implementation.

4. Events

The 'Event Pump' pumps events into the Agent tree. In the simplest version of the SMILE model, five events are defined:

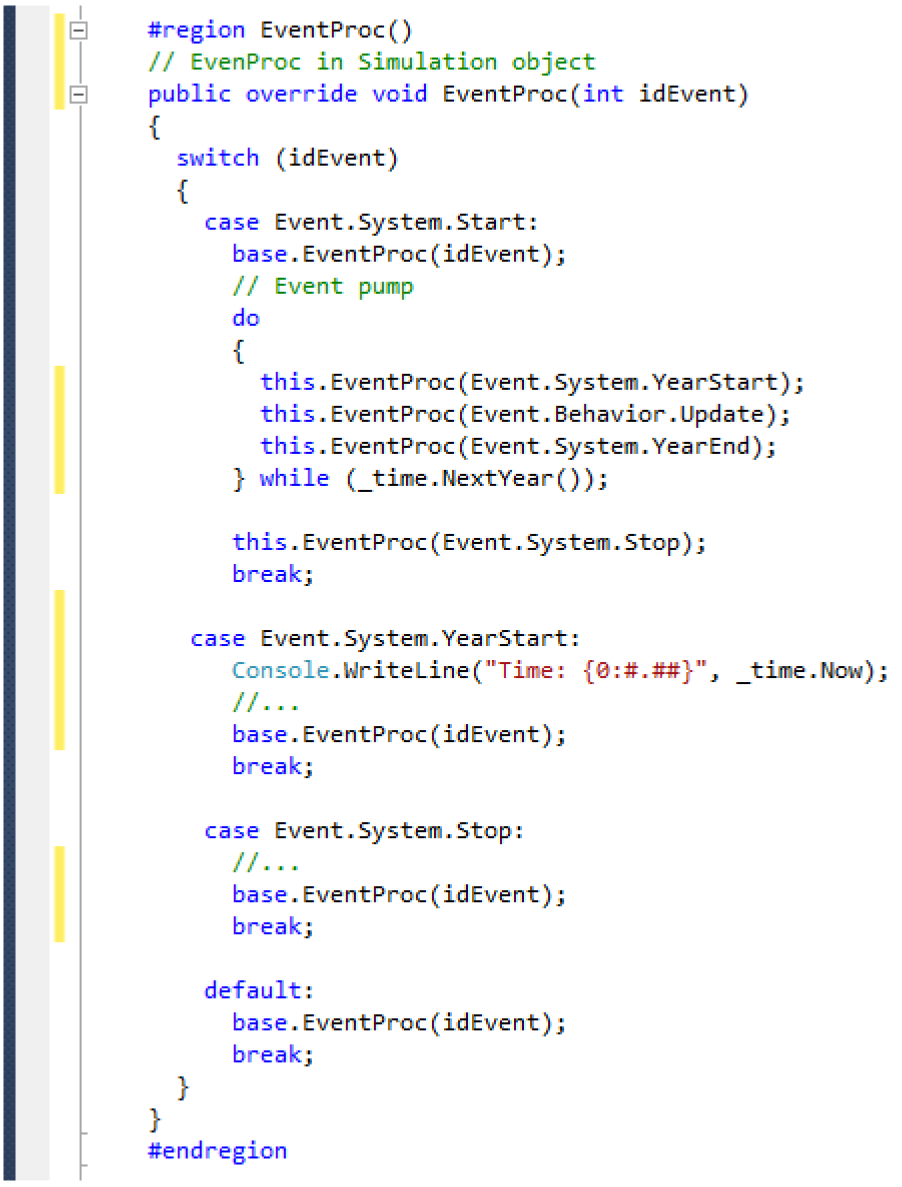
```
Event.System.Start

Event.System.End

Event.System.YearStart

Event.System.YearEnd

Event.Behavior.Update
```

The four system events are used to control the flow of the model. The last update-event is used to trigger the actual behavior of the agents. In more elaborated versions, the update-event will typically be split up in more events (death, birth, move etc.).

In code example 1 a simplified version of the C# implementation of the `EventProc` in the top Simulation object is given. The basic structure of the `EventProc`-method is a so called switch that provides a differentiated response on the various events. The model starts by sending an `Event.System.Start` event to the Simulation objects `EventProc`. This initiates the Event-Pump. A time object _time controls the length of the simulation. The method `_time.NextYear()` returns True every year until the end of the simulation. Hereafter it returns False, such that the Event Pump ends.

In a given year, the Event Pump sends the 3 events `Event.System.YearStart`, `Event.Behavior.Update` and `Event.System.YearEnd` to the Simulation objects EventProc (recursive calls as the Event Pump is itself part of this method; This explains the 'this').

In code example 1 the implementation of `Event.System.YearStart` and Event.System.End is showed. The YearStart-event writes the current year to the console, and sends the event down the Agent tree to its children with the method base.EventProc(idEvent). This is the so called base implementation of the EventProc in all Agent objects. All it does is sending the event to its children (if any). The System.Stop event just sends the event down to its children.



```
#region EventProc()
// EvenProc in Simulation object
public override void EventProc(int idEvent)
{
  switch (idEvent)
  {
    case Event.System.Start:
      base.EventProc(idEvent);
      // Event pump
      do
      {
        this.EventProc(Event.System.YearStart);
        this.EventProc(Event.Behavior.Update);
        this.EventProc(Event.System.YearEnd);
      } while (_time.NextYear());

      this.EventProc(Event.System.Stop);
      break;

   case Event.System.YearStart:
      Console.WriteLine("Time: {0:#.##}", _time.Now);
      //...
      base.EventProc(idEvent);
      break;

    case Event.System.Stop:
      //...
      base.EventProc(idEvent);
      break;

    default:
      base.EventProc(idEvent);
      break;
  }
}
#endregion
```

Code example 1 The EventProc in the Simulation object

Every year all agents in figure 1 receive the events `Event.System.YearStart`, `Event.Behavior.Update` and `Event.System.YearEnd`. The role of the Statistics object is to collect output data from the simulation. This object will use `Event.System.YearStart` to reset all its counters to zero and it can use `Event.System.YearEnd` to write output data to file. It uses `Event.System.Stop` to close the output files. The agents that react on `Event.Behavior.Update` will typically only be the agents in the left household branch in figure 1. When a Person object receives the update-event, it will serially make all its decisions (death, birth, education, move etc.). The decisions are made by asking the Demography object. The

Demography object looks at the asking agent, and gives an answer given its data (transitions probabilities, estimated models etc.) and one or more draws from the random generator. When there is only one update event we call it a *Serial approach*, as opposed to the *Parallel approach* when we have multiple update events.

5. Alignment by slicing

Alignment plays an important role in microsimulation. Alignment basically works by manipulating the probabilities of the model, aiming to do model calibration, comparative analysis (static or dynamic) and/or to eliminate Monte-Carlo noise. In SMILE we use the alignment method described in (Stephensen, 2014).

If the model is build, following the serial approach (only one update event) it is hard to do alignment. Under this approach, all calculations in a given year are done first for agent 1, then for agent 2 etc. Alignment is typically done by gathering all individual transition probabilities and then manipulate these such that a given aggregated target is reached. This is done much easier under the parallel approach: by *slicing* the update event.

For the sake of argument, assume we model death, birth, education, income and tax, and that we only want to align education to given national levels. Under the serial approach we would have only one update event, in which death, birth, education, income and tax would be calculated one agent a time. To do alignment we would slice the update event in four:

`Event.Behavior.Update1`	Death, birth
`Event.Behavior.EducationeAlignInit`	Probability gathering
`Event.Behavior.Education`	Education
`Event.Behavior.Update2`	Income, tax

Under `Event.Behavior.Update1` and `Event.Behavior.Update2` death, birth, income and tax is modelled without alignment. Under the event `Event.Behavior.EducationeAlignInit` all education probabilities are gathered and handed to an Aligment object together with the aggregated targets. This object do the actual alignment calculations. Under the event `Event.Behavior.Education` education behavior is modelled with the manipulated probabilities.

6. References

Bækgaard, H. (2013): *A Bayesian approach to labour market modeling in dynamic microsimulation*, DREAM Conference Paper, December 2013. The paper can be downloaded from www.dreammodel.dk/SMILE

Hansen, J. Z., Stephensen, P. & Kristensen, J. B. (2013): Household Formation and Housing Demand Forecasts, DREAM Report, December 2013. The report can be downloaded from www.dreammodel.dk/SMILE

Hansen, J. Z. (2013): *Modeling Household Formation and Housing Demand in Denmark using the Dynamic Microsimulation Model SMILE*, DREAM Conference Paper, December 2013. The paper can be downloaded from www.dreammodel.dk/SMILE

Lee, Ronald D. & Carter, Lawrence R. (1992): *Modeling and Forecasting U.S. Mortality*, Journal of the American Statistical Association, Vol. 87, No. 419, 659-671.

Rasmussen, N. E. (2013): *Conditional inference trees in dynamic microsimulation - modelling transition probabilities in the SMILE model*, DREAM Conference Paper, December 2013. The paper can be downloaded from www.dreammodel.dk/SMILE

Petzold, Charles (1998): *Programming Windows*. 5th edition. Microsoft Press. 1479 pages.