Chapter 12
Design Principles for Micro Models

Einar Holm and Kalle Mäkilä

12.1 Background

When looking back on about 30 years of model design for dynamic microsimulation, spatial microsimulation, and agent-based microsimulation, it is obvious that the software platform is not stable. The dream of using generic modeling packages, or at least generic software packages, instead of programming each application model from the ground up has not yet materialized as a generally available alternative, despite the calls from practitioners for greater cooperation in the construction of such expensive models (e.g., Harding 2007). For special classes of applications and for models with not too many object instances, promising efforts have emerged like the ModGen toolset from Statistics Canada (Statistics Canada, no date) and all the R-based simulation tools contained in the UrbanSim open-source project (Waddell and Ulfarsson 2004), just to mention two examples out of several development efforts. Most current applied dynamic microsimulation models so far, however, like the MOSART model in Norway (Fredriksen et al. 2011), the Australian APPSIM model (Bacon and Pennec 2007; Harding et al. 2010), and the Swedish SESIM model (Brouwers et al. 2011) are hard-coded for a specific national dataset and application range, as were the original Dynasim and .CORSIM models (Caldwell and Morrison 2000). Several observers have highlighted the problems when it comes to effective sharing of knowledge and development efforts, one late effort being the creation of EURODYN, the European network for dynamic microsimulation (Dekkers and Zaidi 2011).

Of course, this problem has caused an enormous amount of extra work for those who develop microsimulation models. Many quite useful and theoretically interesting models do not survive because of the development effort required. If the microsimulation

E. Holm (✉) • K. Mäkilä
Department of Social and Economic Geography, Umea University, 901 87 Umea, Sweden
e-mail: einar.holm@geography.umu.se; k.makila@bredband2.com

R. Tanton and K.L. Edwards (eds.), *Spatial Microsimulation: A Reference Guide for Users*, Understanding Population Trends and Processes 6, DOI 10.1007/978-94-007-4623-7_12,

community is going to continue and thrive, then we have to invent some clever methods to save mental resources.

The purpose of this chapter is to respond to this need by promoting design principles that have gradually emerged from our long-standing efforts to create agent-based dynamic microsimulation models in Sweden. In order to enable a judgment on the relevance of our specific experiences, a short story of our different models is presented below.

Based on Torsten Hägerstrands time geography (Hägerstrand 1970, 1991), one early model containing much of the intellectual content of later models was the "HÖMSKE" model developed as an application demonstrating an extension of time geography into a time geographic theory of action (Holm et al. 1989). That model contained individuals in families giving birth, dying, moving, getting educated, working, etc. The model contained 1,000 synthetically created individuals, the practical maximum for the computers of those times (a DEC-10 mainframe).

The next step was to access an early version our huge longitudinal individual database (ASTRID), now containing several generations of all Swedes with more than a hundred annually updated socioeconomic attributes, including place of living and working with 100 m of spatial resolution. This meant that there was no longer a need for a synthetic population to start a model with, and we could now estimate behavioral equations on a complete set of longitudinal individual data including individualized subsets of attributes of other persons and workplaces surrounding each person (e.g., the number of people with the same profession within a 50-km radius). The quest to model all Swedes individually soon emerged, in order to grasp the potential of the database. The network of relatives (based on mother and father pointers) is useless in a sample. A 10% random sample does not contain the mothers of 90% of the children. In addition, the distribution of many attributes and resources in the context of the sampled population rapidly becomes distorted if the information for an individual does not contain all object instances – however rare their attributes are.

However, this huge dataset and potential for modeling the whole population also created significant problems. In particular, the modeling needed to provide a representation of all individuals in the population on a not too specialized workstation. This effort resulted in a series of models jointly named the "SVERIGE" models (Holm et al. 1996, 2002, 2007; Clarke and Holm 1987; Holm and Sanders 2007). Specific applications include assessments of local and regional effects of nuclear waste disposal (Berner et al. 2011; Holm et al. 2008), plant shutdown (Rephann et al. 2005), immigration (Rephann and Holm 2004), diffusion of sick leave (Holm and Öberg 2004; Holm et al. 2004a), modeling pandemics (Holm and Timpka 2007), labor supply (Holm et al. 2004b), and the aging population (Strömgren and Holm 2004).

Dynamic microsimulation models invariably contain localized individuals and often interaction between individuals within and outside the family, and interaction between individuals and firms and schools, similar to the ambition of agent-based simulation as reported in Boman and Holm (2004) and Holm and Sanders (2007). A few models applied in countries outside Sweden further the techniques for creating synthetic populations from aggregate data and surveys while otherwise conforming to the SVERIGE tradition (Strömgren and Holm 2010; Aschan-Leygonie et al. 1999).

Other applications include the development of a "twin"-based model applied to exploring the development of the balance between local supply and demand for elderly care, a regional population projection system (ACPOP), and a model for assessment of the endogenous growth impact of infrastructure investments ("InfraSim").

12.2 The Wish List

There are a number of aims that we want to achieve when developing a spatial microsimulation model. These include:

- Using the most modern software
- Using standard methods, shared by many users
- Backward compatibility (so keeping our old models and subsystems running)
- Avoiding relearning
- Developing solutions that are theoretically well designed
- Transferring knowledge and know-how to new colleagues

Achieving all these aims fully in one model is going to be very difficult, as some of them could be taken as contradictory. For example, we want to use the most modern software, but we also want backward compatibility. When looking at the whole model, we can see three levels of components involved in the model building process:

1. The internal logic of the model (the kernel)
2. The input-output system and other general tools used in the model (the tools)
3. The surface of the model (the user interface and presentation of results)

At the extreme ends, the choice is simple: if you have found a useful and stable programming environment, keep using it. This advice applies at least for the kernel but also for the tools. While new and useful methods have been emerging using other environments, the cost of moving to a new environment can be significant in terms of programmer training in the new environment. Most end users also want to use the interface design of the current generation of software, particularly if there is menu-driven interaction with the program.

Most of the remarks below are connected to the tool level. We argue that the kernel and the tools should be kept apart but still kept in the same stable programming environment. These two together, which we refer to as the *application program*, should be strictly independent of the surface level – the user interface. That is not to say that the user interface has to remain the same for different application programs – typically, it is as easy to construct a tailor-made user interface in a certain application as it is to do it in Excel.

To design a generic package that enables model building without programming is desirable but so far unrealistic. It might be possible, but the risk is that it would end up as a new language that is far more technically complicated than the one that was being avoided. Instead, the model design can be constructed at a number of levels:

- Identify the fundamental functionality needed.
- Separate clearly the kernel of the model from the rest of the model, which will be either parameters for controlling the model experiments, code for handling the simulated objects, or output which enables the user to analyze the outcome of the experiments.
- Only elaborate the kernel and the user interface in detail as parts of the simulation program. These are the parts that are most specific to each application.
- The rest of the model (the tools) should be able to be shared between many models.

12.3 Quasi-Independent Subsystems

The choice of design principles and other functionality will be discussed under the following headings:

- Parameter input
- Matrix input
- Equation evaluator
- Result aggregation
- Biography aggregation
- Memory allocation
- Random number generation and use
- Handling of sets
- Random choice between many alternatives
- Primary and secondary attributes
- Parallel execution
- Twins or equations

In the models developed by us over the last 20 years, all of these subsystems have been part of the process. In a few cases, something that is close to a generic package has been emerging, i.e., a module that can be utilized as a ready-made component in later models. More typical has been the copying and modifying of code from one model to another, sometimes simplifying it, sometimes adding features required for the new application. Therefore, we have not presented these subsystems as existing packages or modules. Instead, we are content to contribute some methodological remarks concerning each of the points above.

12.3.1 Parameter Input

Parameters in this case signifies single values, either describing the world simulated, the experiment done, or references (e.g., file names) to more elaborate data. We recommend that this is kept very simple: just a text file with pairs (parameter name:parameter value). Of course, a more elaborate user interface is an advantage,

but we prefer to keep this outside the application, and then it can take many different forms in the interface, so values can be provided by menu choices to support updates, and error checks can be used to avoid illegal values.

All of this can be designed very ambitiously if a model is used heavily and less ambitiously when it is used once or twice and then forgotten, but it is not part of the application. In order to get started with the most difficult parts of the modeling, it is better to start out with very simple parameter handling that can be copied from earlier models and then modified.

12.3.1.1 Matrix Input

Often, there are some fundamental data that occur in many models, typically two-dimensional tables (e.g., age and sex), or just one column of numbers or names. Input of these matrices can often be done by a standard procedure common to many models.

12.3.2 Equation Evaluator

It is convenient and relatively straightforward to implement behavioral equations so that they can be changed without recoding instead of being directly hard-coded, computing an expression like the following:

$$V = p_0 + p_1 * v_1 + p_2 * v_2 + \ldots p_i * v_i$$

where the p_i are estimated parameters and the v_i are values. Some of them are attributes of the simulated objects or other values dynamically calculated in the model. The final calculation is nearly always simple and can often be done by calling a general function. The parameters are then inputted from an external table, which is simple to do. The values are a bit more difficult: sometimes they are simply attributes of the simulated object and sometimes results of rather elaborate calculations based on these attributes and other state variables in the model. Most of this can be defined in a common top-level case statement where each case corresponds to one of the variables. Usually, we call this procedure *eval* and there is one eval function for each kind of simulated object.

12.3.3 Result Aggregation

There are two fundamental output streams from a microsimulation model:

1. The yearly object (population, firms, regions, etc.) status
2. The stream of events

These streams of events must be monitored through a set of well-designed functions that are called at the end of the year and whenever an event (a change of an attribute for a person or family) occurs. These procedures should produce aggregate information of a general multidimensional kind and also micro-data in a compact form. Any other ad hoc printing of output should be avoided. Ideally, the application will construct a set of standard tables from the aggregate data and print them on a set of standard files in a form that makes them directly suitable for post-processing and analysis work after the simulation. It is usually not sufficient to restrict this output to the most basic kind of log files at the microlevel. This will only add additional workload and delay for the user.

12.3.4 Biography Aggregation

A useful tool for debugging is biographies. A biography is all the historical data associated with a person in a microsimulation model. To be informative enough, biographies must be able to track a substantial part of the context of each event (e.g., for a birth, not only the id of the mother but also other attributes, also information on the father, the region of birth, etc.). The log files for biographical information will be very large and will have to be in a compact binary form (if not constrained to a chosen subset of the object instances), but still contain a lot of information to capture the context of each event.

12.3.5 Memory Allocation

For smaller samples (a few hundred thousand persons), it is possible to store all persons as individual objects in an object-oriented program like C++ and keep them in primary storage.

It is also possible to keep all the families in primary storage as class objects linked together in a linked list. Within each family, the members are also built as class objects and linked together in a list. This imposes an enormous overhead, considering that the memory allocation mechanism in C and C++ will use about 15 bytes extra for each item stored. There is also a large overhead in computing time to maintain these structures. However, it is not necessary to allocate all objects by individual calls to the C++ new function. Instead, a large buffer can be allocated by one call in the program, and then the persons in the sample should be allocated as objects through pointers into this buffer. This will save a lot of storage and also processing time.

When the number of objects reaches the level of millions, then another approach must be taken. They can still be kept in primary storage, but now they must be compacted as well. The internal program logic will then be similar to an old-fashioned

data processing program where one family at a time is read from a file, and another file with the updated families is written. When reading the family, it is expanded to the normal problem-oriented class objects. And at the end of the year, the output file is closed and reopened as the input file for the next year.

So the processing is similar to reading and writing compressed or encrypted files, unpacking them one family at a time. However, the files are kept in primary storage, and there is one important reason for this apart from the processing speed for reading and writing on disk as compared to moving data from buffers. The reason is that it must sometimes (e.g., for the matching algorithms) be possible to reach any other person in the population while treating another person. This is solved by saving pointers to the starting points of the family and the relative number within the family for the person.

In the simplest kind of model, there is only one kind of person object. But within the engine that drives the scanning of the buffers, reducing the number of types of people can be used to save space. As an example, if we define people of three kinds: children, active adults, and passive pensioners. Only active adults will need a complete storage of all attributes. For the others, it is enough to set a flag that will cause a number of standard values to be assigned to certain attributes. Stored in this way, nearly 8.6 million persons – with about 50 attributes each – can be fitted into a buffer of about 340 Mb.

While constructing another model (InfraSim) with only about 15 attributes per person, an explicit comparison was made between directly using a conventional object representation of each of the nine million persons in Sweden and using a vector with 15 million (to cover expected population increase) elements indexed by person id for each of the 15 attributes. The object representation required close to 20 Gb of core memory. The vector representation fitted within 300 Mb, showing that significant increase in efficiencies can be gained using a vector representation.

12.3.6 Random Number Generation and Use

Very simple and efficient algorithms can be used to produce pseudorandom numbers. This section recommends some criteria for evaluating these algorithms:

- *Degree of distribution*. Usually, if one million numbers between zero and one are drawn, they will have a continuous uniform distribution. However, intervals near 0 and 1 may need to be checked closely to ensure a continuous uniform distribution.
- Check for autocorrelation between one number drawn and the next few. Ideally, this correlation should be very low, but there are algorithms where this correlation is high. Such algorithms may still work fine in many models as long as the continuous uniform distribution is maintained, but in some cases, they might cause very biased results. One example is when very small numbers tend to be followed by other small numbers. This might result in events with low probabilities

occurring more frequently than expected because they occur concurrently due to the autocorrelation.
- In many cases, these algorithms repeat an identical sequence after a specific number of draws. This is OK, but the number of draws used should not be too short.
- The inbuilt random number generator of some development environments sometimes deteriorates if used too many times with the same start seed. One remedy is to create a new random variable at certain points in time and execution.
- In some cases, simulations can be biased due to the order that the objects are processed in. Usually, the objects are stored in a specific order, and this order is kept from 1 year to the next. Often this is quite OK, but when the model involves matching (searching and forming combinations with other objects), the order may cause a bias. There is one simple remedy: scramble the whole database at the end of each year. This may sometimes take more time than the simulation itself, so make this optional so that scrambling is skipped during test and development.

12.3.7 Handling of Sets

There are many general tools for this in the Microsoft class libraries. Sometimes they are OK, but when dealing with large sets of objects, care needs to be taken. In many cases, it is better to design some code yourself. When doing this, efficiency with storage and computing time is essential. Bitmaps are useful for representing sets of objects, provided they have id numbers as dense series of integers. Arrays are usually efficient and it is not too difficult to design a class where arrays of dynamic size are allocated and expanded when needed. Care needs to be taken with structures involving lists. They are easy to create but might be wasteful of space. In particular, removing elements might be a waste of time. For each deleted item in the list, the method wrapper copies all elements but the removed one into a new vector and then replaces the old vector with the new one. To do that a million times within a loop through, the whole population is an enormous waste of computing resources. It is possible to keep using lists if the remove method is avoided and replaced by giving the elements to be removed a flag and then performing the actual remove by copying as if the list was a vector, i.e., at the end of the year.

12.3.8 Random Choice Between Many Alternatives

Sometimes in a model, choices are made between many alternatives according to a probability distribution. If this is done only a few times, then it can be done in a loop where a drawn random number is compared to the limits in a cumulative distribution. But when this is done for a large population and the number of choices is large (like when choosing between the 290 local administrative areas of Sweden, the municipalities), then this will take a lot of time. The process can be sped up considerably by generating

an auxiliary data structure before the simulation is started or at the start of each year if the probabilities change over time. To do this, the program needs to generate a large vector of values, and for each alternative, store it repeatedly in proportion to the probability for that alternative. Then a choice can be made by just drawing one single random number. In the case of migration between municipalities, the large municipalities will have a large number of alternatives, so then a large vector is needed to represent them all with some precision. Smaller municipalities will have fewer alternatives so then a smaller vector can be used. While these vectors will take a lot of memory space, the gain in computing time will be enormous.

The same method can be used for other attributes, e.g., professions, and also in matching procedures, combining persons with other persons or with workplaces.

As a general point, there has been a shift of balance between storage and computing time. We are often impressed by the great improvement in computing time but tend to forget that the improvement in terms of memory in computers, both in terms of size and price, is even bigger. A large amount of memory can now be devoted to auxiliary data that will speed up the search in large sets of data.

12.3.9 Primary and Secondary Attributes

The simplest models only deal with one kind of object (the person). Other models contain several object classes like family, workplace, and municipality. The additional objects may possess static as well as dynamic attributes which can change during the simulation (e.g., the number of members in families or workplaces, or the mean income in a workplace). Persons have their own attributes, but are also connected by pointers to other objects, and so they can access attributes of the other objects. All of these attributes can be used in the model equations. This means all attributes should be accessed in the same way through a function (eval) containing a case statement with cases for both the primary attributes and other cases that will call the eval functions of the connected objects or other functions that make additional calculations to transform or recombine the stored attributes. The inbuilt "property" construct might sometimes give a useful alternative.

12.3.10 Parallel Execution

One obvious emerging option to increase the speed of a large simulation is to make use of the threading capabilities of current multi-core computers. There are now tools available to organize this at the source code level, e.g., by partitioning the program and assigning different processors to simulate these partitions in parallel. This works only if each partition is strictly independent of the others, so it's easy to achieve only for the simplest kinds of models. We have moved towards more and more interaction between different parts of the model, so these mechanisms are not easy to use in our models without complicating the source code considerably.

For certain parts of the program, high-level methods like the canned parallel for loop of the .NET framework are sometimes useful. One advantage of using such constructs is that it just doesn't work if the parameters of the called function are not strictly independent of the calculations in the function. That gives a kind of additional run time consistency check.

The increased speed seen using parallel execution is larger for the development phase than it is for final production runs of an application. During development, one very soon approaches the situation when it is necessary to run the full model many times with the entire population in order to discover remaining bugs. This can be very tedious if the compile–run cycle takes a long time.

A typical production run often requires many replications and/or different alternative experiments. In these cases, it is easier to use the parallel machinery at process (operator) level. The simplest solution is to run several simulation experiments in parallel. Another way that was implemented in the SVERIGE model is to implement some of the housekeeping activities needed to collect aggregates during the simulation or to maintain auxiliary data that supports rapid access in memory as parallel processes running concurrently with the simulation.

12.3.11 Twins or Equations

A demanding analytical task is to accurately model the outcomes for large choice sets, like the choice of destination when moving geographically or the choice of specific education or profession or place of work. Geographic mobility and connected family changes are core events in spatial population modeling. A Wilson/Fotheringham-type interaction model tends to require fine-grained alignment in order to produce reasonably accurate local outcomes, almost to the point where the output largely reproduces the alignment factors. Therefore, especially in situations with a large and diverse choice set like destination choice combined with simultaneous family changes, imitating empirical twins might stand out as a simple viable alternative to multiple logit-based equations or interaction models. Instead, a random assignment scheme is applied based on imitating observed behavior of similar individuals ("twins") as discussed by Klevmarken (1997).

We have tested this alternative as demonstrated by the following simple example model tested on a database of the Swedish population. The example model contains three time-independent variables (birth year, sex, and place of birth) and four time-dependent variables (municipality, education level, civil state, and disposable income). Municipality is fully represented (290 values) but the other variables have been reduced to make a complete match of twins possible. Twins are picked for one particular year in the example (1993). The year after the twin year is the result year. So the whole simulation procedure can be described as:

1. Scan the whole population and locate a twin with an identical or very nearly identical set of attributes.
2. Pick up the set of attributes for this twin in the result year.

3. Let the simulated person inherit the whole set of attributes as values for the next simulated year.

Altogether, there are 2*4*512*290 = 1,187,840 possible combinations for the time-dependent variables distributed over 2*4*64 = 512 combinations of the time-independent variables (sex, origin, and age class). Before the simulation can be done, the behavior of the individuals for the past years must be aggregated in such a way that those having a specific combination can be easily accessed. For each combination occurring in the dataset, a number of people are found, each of them having a particular combination of result variables. All of this must be loaded in the core before the simulation.

Note that twins are not unique or limited to just a few individuals. Many of the technically possible 1,187,840*512 cells are actually empty, but others might contain thousands of individuals. It is not always possible to find an exactly matching twin. In the example using data for the Swedish population, we applied a set of strategies for locating "proxy twins" which are those that are similar except for one or two dimensions. When simulating about nine million individuals, we get approximately the following outcome in terms of hits:

In full detail	7,000,000
Within age +/−1 or +/−2	1,000,000
Within proximate income	600,000
In the most frequent income	100,000

The hypothesis is that twin replication might sometimes outperform behavioral equations, especially when the outcome is complex, not binary or scalar like the destination choice of movers; when several events interact simultaneously like family formation and mobility; or when it is important to maintain a realistic heterogeneity in the long run like an income distribution.

Advantages of using twins are that heterogeneity in all attributes is maintained automatically and that latent information not obvious from the attributes can affect the results. In addition, twins might in some cases function as a consistent alternative to alignment.

The disadvantages of using twins' replication are that results easily get locked into the sample of the empirical twins. Problems can occur when the simulation approaches a state not experienced by any empirical twin. Experiments changing behavior are easier to perform using analytical behavioral equations. In addition, the definition of and the criteria for selecting a twin are not obvious except in very simple cases.

It has been demonstrated that for a simple simulation, twin replication produced somewhat superior results at least for the age distribution of movers and the destination of movers. For complex events, it might in some cases give a convenient alternative to equations that maintain consistent heterogeneity better than using a table look up. Questions like how to define similar, how to select twins, and how to use biographical individual information still need to be solved if using models with individuals with many attributes that make them unique. So, in this case, we have no general recommendation to make.

12.4 Conclusion

Our own answer to the relevance question raised in the background section is that most of the discussed design principles might be even more relevant in other countries which don't have the rich longitudinal individual data that exists in Sweden. However, these principles are recommendations about how to code and reuse code, and there are no ready to use software modules or modeling packages for dynamic spatial microsimulation. We would recommend that these principles be implemented while constructing new national and regional simulation models, and we would particularly emphasize the use of the proposed design principles in efforts to create generic high-level software for dynamic microsimulation.

One alternative would be to integrate core parts of the discussed design principles into something like the already well-developed toolset of ModGen. This would then contribute somewhat towards moving the resulting software into a tool, replacing the need for a large portion of our own (as well as others') application specific coding for different models. It would then be possible for the next generation of social scientists in academia and in agencies to apply micro-based dynamic modeling on urgent problems instead of entirely relying on regression analysis with its obvious shortcomings when it comes to representing the complex dynamic interactions of different agents within society.

References

Aschan-Leygonie, C., Baudet-Michel, S., Gautier, D., Holm, E., Lindgren, U., Mäkilä, K., Mathian, H., & Sanders, L. (1999). Micro modelling of the population dynamics in a region with strong urban growth. In S. E. Van der Leeuw (Ed.), *Archeomedes*.

Bacon, B., & Pennec, S. (2007). *APPSIM – Modelling family formation and dissolution* (Online Working Paper – WP2). http://www.canberra.edu.au/centres/natsem/

Berner, B., Drottz Sjöberg, B., & Holm, E. (2011). *Social science research 2004–2010, themes, results and reflections*. Stockholm: SKB. ISBN: 978-91-978702-2-1

Boman, M., & Holm, E. (2004). Multi-agent systems, time geography, and microsimulations. In M.-O. Olsson & G. Sjöstedt (Eds.), *Systems approaches and their application: Examples from Sweden* (pp. 95–118). Dordrecht: Kluwer.

Brouwers, L., Ekholm, A., Janlöv, N., Johansson, P., & Mossler, K. (2011, June 8–10). *Simulating the need for health- and elderly care in Sweden – A model description of Sesim-LEV*. Paper presented at the Third General Conference of the International Microsimulation Association Stockholm.

Caldwell, S., & Morrison, R. (2000). Validation of longitudinal microsimulation models: Experience with CORSIM and DYNACAN. In L. Mitton et al. (Eds.), *Microsimulation in the new millennium*. Cambridge: Cambridge University Press.

Clarke, M., & Holm, E. (1987). Micro-simulation methods in spatial analysis and planning. *Geografiska Annaler Series B, Human Geography, 69*(2), 145–164.

Dekkers, G., & Zaidi, A. (2011). The European network for dynamic microsimulation (EURODYM) – A vision and the state of affairs. *International Journal of Microsimulation, V4*(1), 100–105.

Fredriksen, D., Knudsen, P., & Martin Stølen, N. (2011, June 8–10). *The dynamic cross-sectional microsimulation model MOSART.* Paper presented at the Third General Conference of the International Microsimulation Association Stockholm.

Hägerstrand, T. (1970). *What about people in regional science, regional science association papers, Vol. XXIV*. Heidelberg: Springer.
Hägerstrand, T. (1991). Tiden och Tidsgeografin. In G. Carlestam & B. Sollbe (Eds.), *Om tidens vidd och tingens ordning, T21*. Stockholm: Statens råd för byggnadsforskning.
Harding, A. (2007, August 21). *Challenges and opportunities of dynamic microsimulation modelling.* Plenary paper presented to the 1st General Conference of the International Microsimulation Association, Vienna. Available at http://www.euro.centre.org/ima2007/programme/day2.htm
Harding, A., Keegan, M., & Kelly, S. (2010). Validating a dynamic microsimulation model: Recent experience in Australia. *International Journal of Microsimulation, 3*(2), 46–64.
Holm, E., & Öberg, S. (2004). Contagious social practice? *Geografiska Annaler, 86B*(4).
Holm, E., & Sanders, L. (2007). Spatial microsimulation models. In L. Sanders (Ed.), *Models in spatial analysis* (Geographical information systems series). Newport Beach: ISTE.
Holm, E., & Timpka, T. (2007). A discrete time-space geography for epidemiology: From mixing groups to pockets of local order in pandemic simulations. *Studies in health technology and informatics, 129*, 464–8.
Holm, E., Mäkilä, K., & Öberg, S. (1989). *Tidsgeografisk handlingsteori – Att bilda betingade biografier*. Gerum Rapport 8, Department of Geography, Ume University.
Holm, E., Lindgren, U., Mäkilä, K., & Malmberg, G. (1996). Simulating an entire nation. In G. Clarke (Ed.), *Microsimulation for urban and regional policy analysis*. London: Pion.
Holm, E., Holme, K., Mäkilä, K., Mattsson-Kaupi, M., & Mörtvik, G. (2002). *The SVERIGE spatial microsimulation model – Content, validation, and example applications* (Gerum Kulturgeografi 2002, Vol. 4). Umeå: Umeå Universitet.
Holm, E., Lindgren, U., Eriksson, M., Eriksson, R., Häggström Lundevaller, E., Holme, K., & Strömgren, M. (2004a). *Transfereringar och arbete* (Arbetsrapport R2004, Vol. 16). Östersund: ITPS – Institutet för tillväxtpolitiska studier.
Holm, E., Lindgren, U., & Malmberg, G. (2004b). *Arbete och tillväxt i hela landet – betydelsen av arbetskraftsmobilisering* (Vol. 22). Östersund: ITPS – Institutet för tillväxtpolitiska studier.
Holm, E., Lindgren, U., Häggström Lundevaller, E., & Strömgren, M. (2007). SVERIGE. In A. Gupta & A. Harding (Eds.), *Modelling our future, population ageing health and aged care: International symposia in economic theory and econometrics* (Vol. 16). Amsterdam/Boston: Elsevier.
Holm, E., Lindgren, U., & Strömgren, M. (2008). Socioekonomiska effekter av stora investeringar i Oskarshamn. SKB R-08–76.
Klevmarken, A. (1997). Behavioral modeling in micro simulation models: A survey (Working Paper Series 997:31). Uppsala University, Department of Economics.
Rephann, T., & Holm, E. (2004). Economic-demographic effects of immigration: Results from a dynamic, spatial microsimulation model. *International Regional Science Review, 27*, 379–410.
Rephann, T., Mäkilä, K., & Holm, E. (2005). Microsimulation for local impact analysis: An application to plant shutdown. *Journal of Regional Science, 45*, 183–222.
Statistics Canada. (no date). *Modgen 15 years of creating models.* http://www.statcan.gc.ca/microsimulation/pdf/modgen-hist-eng.pdf. Accessed 20 Aug 2011.
Strömgren, M., & Holm, E. (2004). *Åldrande befolkning och framtida behov av kommunalskatt*, Kulturgeografiska institutionen, Umeå universitet.
Strömgren, M., & Holm, E. (2010). *Using downscaled population in local data generation* (Technical Report). ESPON 2013 Database.
Waddell, F., & Ulfarsson, G. (2004). Introduction to urban simulation: Design and development of operational models. In B. Stopher & H. Kingsley (Eds.), *Handbook in transport, Volume 5: Transport geography and spatial systems* (pp. 203–236). New York: Pergamon Press.