Multi-Agent Model to Multi-Process Transformation

A housing market case study

Gerhard Zimmermann

University of Kaiserslautern

Keywords: Agent technology, User activity modeling, User activity simulation, Software engineering, Code generation, Software process

Abstract: Simulation is a means to help urban planners and investors to optimize inhabitant satisfaction and return on investment. An example is the optimal match between household preferences and property profiles. The problem is that not enough knowledge exists yet about dynamic user activity models to build reliable and realistic simulators. Therefore, we propose a modeling and software technique that produces simulator prototypes very efficiently for the development, test, and evaluation of many different user activity models, using executable models, code generation, and a domain specific software process. As a specific feature, the model is based on many agents acting independently from each other and that are mapped in several refinement steps into the same number of concurrent processes. The housing example is used as a case study to explain the process and show performance results.

1. INTRODUCTION

User activities in urban and building environments gain more and more interest in architectural design and evaluation tasks. Since user activities are non-deterministic and since model abstractions of these activities are not yet well understood or validated, tools are necessary to develop and test user activity models and to apply such models in design decision processes. Computer simulation is a means to implement such models and to execute, observe, and analyze user activities with different initial and boundary conditions.

Although a number of simulators exist for applications such as motorized and pedestrian traffic, models and simulators for other kinds of user behavior...
in urban and building environments still have to be developed and tested. One problem is that of finding the appropriate models for the desired applications. We specifically concentrate on dynamic models of many individual users instead of stochastic models of user groups because traffic simulations have shown that human behavior is modeled more realistically that way. Appropriate in this context means that because of the high complexity of such models the abstraction of the reality should be as simple as possible without missing essential details. Therefore, many experiments with different abstractions are necessary to arrive at models with the right amount of details.

This leads to the next problem, the implementation, execution, and variation of models. Advances in computing and software techniques in the last decade have provided us with a number of options to solve this problem as efficiently as possible. Efficiency is very important because during the development phase of models a large number of different experiments are necessary. Therefore, we propose and demonstrate a modeling and software prototyping process that is based on executable models and code generation tools. Executable models are formal models which include dynamic behavior and can be either interpreted as or compiled into executable computer programs. We will use a case study to demonstrate the process and show efficiency data.

2. STATE OF THE ART

Our work is based on three main areas: user activity modeling, simulation, and software engineering. We cannot cover all of them completely and will restrict ourselves to some representative examples.

Static user activity modeling in building environments was introduced by Eastman and Siabiris (1995) and by Eckholm (2001). In the IFC models user activities are not yet included. We also presented an effort to integrate user activities into a comprehensive building system model (Zimmermann, 2003) as a basis for extending it to user dynamics.

Models exist or are under development for pedestrian movements (Kukla et al. 2002) or in shopping centers (Borgers and Timmermans, 2005). The models are either based on cellular automata with a very restricted horizon and no differentiation of individuals or on agent systems with a much larger horizon and the possibility of giving different properties to each individual. Still, cellular automata exhibit much more realistic pedestrian behaviors than stochastic models and because of the regularity, synchrony, and simplicity can be executed with very large numbers of individuals. Agent models can implement much more complex behaviors and are in principle asynchronous. The possible size of such simulations has to be seen.
Models of other user activities are also under development. For our case study we adopted models from (Devisch et al. 2005) for the dynamic migration behavior of households in towns. Devisch is modeling individuals, households, realtors, and properties as agents and is implementing very sophisticated decision and negotiation models.

Simulation has become a mature tool in the area of building performance (e.g. ESP-r). Such simulators are based on modeling the physical environment by differential equations and numerical solutions at run-time. Also in the area of pedestrian simulation and visualization, for example for the purpose of emergency evacuation (Klüpfel, 2005), tools exist and are in use. These are based on cellular automata or agent technologies. Integrating the different simulation approaches is difficult. Therefore, we successfully experimented with performance simulators that are based on mapping physical entities such as walls and windows to autonomous objects and executing all objects concurrently (Zimmermann, 2002). This approach lends itself to integration with agent models.

The construction of simulators can be very time consuming tasks. This is true for many reactive systems and because of the typically low number of sold systems, development cost is an important factor. For this purpose intensive research has been conducted in the field of software technique for reactive systems. At the same time software engineering has made large progress, especially by better modeling techniques such as the Specification and Description Language SDL (Olsen et al. 1994) and the Unified Modeling Language UML. Both environments allow formal models of asynchronous dynamic behavior and tools are available for automatic code generation. This code may not be as computational efficient as manually programmed code, but the generation enhances process efficiency by magnitudes. We are using SDL for modeling and simulation purposes. Environments such as SIMULINK and Modelica are more orientated towards continuous simulation and not so suited for modeling user activities. Efficiency also depends on the applied software process. For the purpose of user activities special processes for reactive systems are of advantage, for example (Braek and Haugen, 1993). We have tailored such a process for building control systems and performance simulators (Zimmermann and Metzger, 2004) and will show in this paper that this process can also be applied efficiently to user activities in the case study.

3. MODELING ENVIRONMENT

The model used to describe the structure follows the principle idea of our object-based building system model (Zimmermann, 2003). The model has several levels of refinement. The top level in Figure 1 represents the relation
between the domains, in our case the *User Activity* and the *Functional Unit Domain*. The next level refines these domains to represent general types for urban models, for example *Group Role Type* in the *User Activity Domain* and *Organizational and Functional Unit Type* in the *Functional Unit Domain*. The *Application Domain Level* further specializes each *Domain Level Model* for a specific area, for example population migration.

For a specific project like the one presented here, specific object types are derived from the types above the dashed line by instantiation. A *Household* type is an instantiation of the *Group Role Type* with the function of controlling and changing (by moving) the utility gained from the inhabited property. The *Property* type is an instantiation of *Functional Unit Type* and a *Market* type of the *Organizational Unit Type*. The function of a market is the management of properties and making offers on request. Finally, the simulator at run-time is a further instantiation of these types into individual objects. Objects can exhibit agent properties and we will also call them agents. *Figure 4* shows the object type diagram for our case study project as an example. For automation purposes we also use tabular representations of the object type structures.

Modeling the dynamic behavior is a more demanding task. We use several representation types for different refinement levels. With structured text we describe *Needs, Tasks*, and *Strategies* informally. Structured text means that identifiers are assigned to text blocks to be able to store, retrieve, and reuse the blocks in a structured way and to be able to define relations between blocks. This is important for consistency checks and for documentation and annotation purposes. In *Chapter 5* we will show some examples.

With *Needs* we describe what the system is supposed to do for its user. These *Needs* are then translated into *Tasks* and subtasks that are assigned to object types. From the *Tasks* we derive *Strategies*. 

---

*Figure 1. Modeling levels. Triangles depict inheritance, straight lines instantiation.*
The first refinement of the Strategies is MTCs (Message Transition Chart). To understand these charts we have to introduce the model and software architecture of the target simulators in the language SDL (Olsen, 1994). All objects are mapped into threads that execute concurrently and communicate by messages. These threads act like agents and we will use this term from now on. Agents are internally modeled as extended finite state machines (EFSM). They are represented by state transition diagrams. Messages trigger state transitions. Messages are saved in an input queue for each agent. If a queue is empty, the agent becomes inactive. Time can be introduced by timers that also send messages at the set time.

Both the agent-message level and the EFSM level individually are not very complex when both are well structured. We do this by using aggregation hierarchies of agents in SDL to keep individual agent types reasonably simple. The difficulty to design and understand models arises when both levels are combined. This is necessary to understand the dynamic behavior of the whole system. MTCs are a means to abstract both levels as much as possible to understand the main dynamic actions and interactions of the agents. With MTCs we only model the messages that are exchanged between agents and the state transitions that are triggered by messages and create new messages. All other state changes, especially data transformations are neglected. MTCs do not have to be complete. Individual scenarios can be modeled as action-reaction chains. Further abstractions show agents as composed states. Figure 5 and Figure 6 show an example from the case study.

4. THE SOFTWARE GENERATION PROCESS

A software generation process PROBAnD (Metzger and Queins, 2002) was developed for reactive systems in general and especially for building control systems. The efficiency of the process is based on advanced modeling and software generation tools, but also on tailoring the process to the specific domain. Because of the latter, we can make use of predefined document and model types and architectures, specific process structures, and reuse partial descriptions and models.

We first adapted this process with small changes to building performance simulation (Zimmermann, 2002) and have now also used it for user activity modeling. Here we only describe the main steps of the process and explain it in more detail using examples from the case study in Chapter 5.

Figure 2 gives an overview. Starting point are the Needs that build the problem description and information about the environment such as layouts and other design information. Especially the environment structure is used to build an object structure, using the same terminology. This is important for
the necessary discussions between modeler and customers, in this case urban planners. The mapping of *Needs* into *Tasks* now depends on the chosen *object structure* shown as a strong relation in Figure 2.

The modeling of the concurrent interaction between objects (agents) can be quite complex. The MTCs are a means to divide this task into *scenarios* which are orthogonal to the object structure. MTCs are optional and the direct path from the *problem description to strategies* is also possible.

*Strategies* are first described informally. A formal description is derived in several refinement steps. For this purpose we use two representations. The first is a textual formal language that describes all state transitions individually and is structured in tabular form. It is closely related to the task and MTC structure. The advantage of a textual over a graphical language is that reuse of text components is easier than of graphical components.

The step from the textual to the graphical representation of SDL can be done manually, but is error prone and time consuming. Therefore, we automated this by the tool PROTAGOnIST (Metzger and Queins, 2003). The graphical layout follows simple rules and is done automatically. SDL graphs can be further manipulated and analyzed by the commercial tool environment SDT (Telelogic).

The last step is the automatic generation of C-code using SDT. This also produces a run-time environment for the concurrent execution of agents and the message and timer administration. Finally the code is compiled into an executable program and can be run in simulation experiments. SDL provides different generators to create code for interactive debugging, real-time code,
efficient code without debugging aids, and application code that can be run stand-alone without SDT licenses.

In Chapter 5 we will demonstrate the process with examples. Figure 2 shows efficiency data in the form of person hours spent on different document types for the case study. The data are automatically recorded by a simple document management and versioning environment, based on internet browsers. Adding all hours results in 89 person hours or 11 work days. The large number of 61 hours for SDL is mostly debugging and refinement time because of being to “hasty” in earlier phases. We can assume that one person can create such a simulator within three weeks. We achieved the same result with other applications as well. This is a very reasonable time frame. Creating initial and boundary data take at least this time and simulation experiments as well.

5. THE CASE STUDY

This case study was adopted from ongoing research at the Eindhoven University of Technology (Devisch, 2005) to be able to compare models, process, and results. It is not intended to copy all model details, nor to compete in any sense. All decision processes have been greatly simplified. Our goal is the demonstration of the software process and the efforts necessary to produce a simulator prototype and run some initial experiments.

5.1 The Problem

It is of interest for city planners to be able to predict the result of changes in the built environment for citizens and commercial enterprises regarding their satisfaction with the gained utility. It is also of interest to analyze the result on yield for owners and investors and on city tax income. Example changes are the construction of new properties, the change of the environment and quality of buildings, or of renting and buying prices. It is also of interest to predict changes in the population itself and the result on satisfaction and necessary changes in the built environment. This includes migration into and out of cities.

In this case study households are considered as basic entities. Households are created, change in size, and disappear, mostly because of biology and of decisions of individuals to join or separate. Other properties such as income, work place position, or preferences can also change over time. Households occupy properties in buildings. Properties can be owned or rented. The dynamics are mainly visible as moves of. Other changes are property improvements to increase satisfaction. Both events are determined by
decisions of the households and by the property market. Therefore, we need models of the dynamic behavior of households and of property markets. Since we model all entities individually, they should not be uniform but display the whole spectrum of different behaviors. The behavior should also have some random components, modeling non-determinism.

One of the problems of such models is complexity. This is not so much a computational problem. If models have too many parameters, we have no way to determine a correct set of parameters by real life experiments. In our approach we therefore start with as simple models as possible and allow for extensions when necessary.

5.2 Basic Model

The basic model of a small town consists of five property markets, one in the center called city and four suburbs. Figure 3 shows the layout. With this simple geometry tests for the assignment of positions to markets is simple and decision processes can still be tested. We also use a fixed assignment of one realtor agent per market as a further simplification. This agent manages all properties of its assigned market.

Households are not modeled as groups of individuals, but as basic agents. Households are grouped into 15 types according to the number of persons (1..5), employment status (student, employed, retired), and income level (low, high). Unlikely combinations are excluded.

Households are characterized by profiles and preferences. Profiles are data about the household, as for example income, current property, and position of workplace. Preferences describe how households value property profiles and how flexible households react to different property options. Household and property profiles are evaluated against the preferences to calculate the utility of a property for a household. Details of the data and the calculations are in Section 5.3.
Properties are characterized by property profiles. The profiles include typical data such as type, location, size, ownership, value and cost.

All households and properties can have different data. The data are stored in external files and are loaded into the simulator before the simulation starts (initial condition) and during simulation to communicate changes (boundary condition). The classification into household types is only a means to create sample data that reflect statistical data of real communities.

5.3 Utility Metrics

The goal of households in selecting properties and deciding on moves is the optimization of the utility it gains from using a property. Many factors influence the utility evaluation. Other goals such as minimizing the cost of providing the urban environment for the city administration and to optimize the return on investment for property owners can also be considered. In this case study we concentrate on utility metrics only, although the other goals can be observed as well using the resulting data.

In this case study we have selected eight criteria out of a much larger number of possible ones (see also Devisch, 2005) for the calculation of utilities. The following list contains a maximum of three choices for each criteria and short explanations:
1. urbanisation: city, suburb, rural. The first two correspond to the sections of our model town, rural is set equal to external.
2. tenure: renting and owning a property.
3. garden: no, small, large garden belonging to the property
4. type: apartment, row house, free standing house
5. size: <80%, 80-120%, >120% of the ideal floor space for a household which is a profile value.
6. distance: >10, 2-10, <2 km rectangular distance between property and centre of external household activities, for example the work place, the school, or a shopping centre.
7. remaining income: <50%, 50-70%, >70% of the total household income reduced by rent or sales price financing of the property.
8. resistance to move: single value that reduces the utility of a property in case a move to it is necessary. Can be extended to resistance to renovate.

Using a larger number of criteria and choices would in principal refine the utility calculation, but would also require a much larger number of realistic preference parameters to be determined by real life experiments and would not change the decision process in principal.
For each criteria $i$ and choice $j$, a preference value $b_i(j_i)$ typically for a household has been defined. The range of $b_i$ is $[0\text{-}100\%]$ such that

$$b_i(1) + b_i(2) + b_i(3) = 100\%$$

(1)

The maximum values of a choice for each criteria show the relative main preference, the values of the other choices show the household’s tolerance towards other choices. For example, if a household would prefer a free standing house, but would find a row house also ok, but not an apartment, it could express this as $b_x = (0, 30\%, 70\%)$. In order to express the importance of each criteria in relation to the others, weight factors $w_i$ are introduced.

We can now calculate the utility $u$ for a household with known preference values and known $j$-vector as

$$u = \sum_{i=1}^{8} b_i(j_i) \cdot w_i$$

(2)

The resistance to move weight $w_8$ has to have a negative value in the case of a necessary move and zero otherwise.

We calculate three different utilities. $u_{\text{ideal}}$ is the best value for a given preference matrix if for each criteria the best choice is selected. For the currently inhabited property its profile is compared against the household profile in order to determine the $j$-vector and to calculate $u_{\text{curr}}$. The same method is applied for offered properties, resulting in $u_{\text{offer}}$.

Next we need a value that shows the relative satisfaction $s$ of a household with its current property or with offered properties to decide if a move is appropriate. We calculate the satisfaction by comparing the ideal utility with the current or offered one. A satisfaction factor $sf$ describes the reduction of the ideal utility that is still seen as satisfactory. In our simulations, $sf = 75\%$ has been assumed to give a positive satisfaction. This leads to

$$s = \left( u_{\text{curr}} - sf \cdot u_{\text{ideal}} \right) / u_{\text{ideal}}$$

(3)

If this value is positive, the household is assumed to be content with its situation; if negative it will search for an alternative with a better satisfaction value. The possible range is of $s$ is $[0.25, -0.75]$.

Naturally, a household will first search for offers with a utility close to the ideal one. Therefore, three attempts are made with a minimum offer acceptance level of 90% of the difference between ideal and current utility and then going down in 8 steps of 10%. If still no offer is found, the search is
aborted until a later date. This strategy may lead to a move into a property with negative satisfaction and also trigger a new search, but not immediately to allow the market to change.

The satisfaction of a household may change over time and trigger a search. Such changes can stem from changed preferences and profile values, as for example household size or new work place. The change may also stem from changed property profiles such as depreciating quality of it over time.

5.4 Model Structure

The structure of the model is static. All households are modeled as individual agents. We have limited the number to 500 to be able to generate and run simulation prototypes in the range of 10 minutes for a two year simulation time period. The five markets are also agents. A minimum of 500 properties are distributed to these markets but are not modeled individually. The aggregating agents are objects that can perform global bookkeeping, instrumentation, and data collection. Figure 4 shows the object type structure. The cardinalities define the number of objects that are aggregated by the object type. The object BigCity functions as simulation time management and user and file interface.

![Object type structure](image)

*Figure 4. Object type structure, the diamond denotes aggregation, the numbers the cardinality.*

5.5 Model Dynamics

The model functions as follows: Households with negative satisfaction send a request to the city and the suburb markets nearest to their center of activity. The request contains a floor area and a cost range and the tenure preference. Both markets select a maximum of three offers each from a list of free properties that match the request and send the property id to the household. The household polls the property data, calculates offer utilities and sorts them accordingly. For offers that are acceptable an *accept* is sent to the market. Since the markets can send the same offer to many households simultaneously, acceptance messages are handled on a first-come-first-serve
basis. If the property is still free, a confirmation is sent to one household, taking the property from the free list. When the household has moved, the old property is set on the free list. This is in short what happens.

All initial data and all dynamic changes are fed to the simulator from files that we derive from spreadsheets. This gives us the opportunity to run experiments with different sets of households and properties without changing the simulator. Result data are also stored in files and evaluated by spreadsheets. The amount of output data can be controlled by a simulation parameter. Real time simulation is possible and time compression is controlled by another factor.

5.6 Software Process

The process has been outlined in Chapter 4 and we will follow it with examples of documents from the case study. Because of the limited space we have selected short document sections.

The problem description is composed of needs with identifiers, as in Table 1:

<table>
<thead>
<tr>
<th>Need1</th>
<th>Given an urban plan with all housing properties, and given an assignment of households to properties, simulate and observe the migration of households within the local market</th>
</tr>
</thead>
</table>

An example for the environment is Figure 3. The object type structure is shown in Figure 4. From need Need1, Task3 for object type Household and Task5 for LocalMarket can be derived:

<table>
<thead>
<tr>
<th>Task3</th>
<th>Calculate satisfaction and request offers from near local markets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task5</td>
<td>Receive requests from households, find properties and send offer back</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Task3</th>
<th>Calculate satisfaction and request offers from near local markets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task5</td>
<td>Receive requests from households, find properties and send offer back</td>
</tr>
</tbody>
</table>

As an option, MTCs can be created to design the interfaces between the objects (Figure 5) and the basic state transitions. The folded corner in the compound states mean that refinements exist. The diagram has been produced and edited with a tool environment DOME (dome).
Figure 5. Household – LocalMarket MTC interaction diagram. Arrows are messages with message name. Message parameters not shown.

Figure 6. MTC of object type Household, scenario 'get offer'.

Multi Agent Model to Multi-Process Transformation 215
The detailed internal view of the main scenario is shown in Figure 6. It has to be pointed out that this is not the typical state-transition diagram. States can occur several times in one diagram. The primitive is the transition with two thin arrows and the diamond in the middle. Thick lines denote messages. Conditions are shown in square brackets.

Figure 6 is the main scenario of a handshake of a household that is no longer satisfied with its current property and asks a local market (realtor) for offers. If a satisfactory offer is found, the household sends an acceptance message. Since there is no guarantee that the property has not already been taken by another household, it has to wait for a positive confirmation, before it can take the property. Otherwise, it has to select another property from the offer or try again (not shown). Also not shown are additional built in delays, modeling the times necessary to evaluate and decide. The delay unit is one day, with a resolution of 1/1000. Delays can be fixed or made dependent on the urgency of a move.

5.7 Simulation Experiments

We ran several experiments, first with a small number of households and then with 500. The problem we faced was getting the right data for the household preference matrix and household and property profiles. We used what statistics and rules we found from public sources and guessed the rest using common sense.

One of the problems was to find a reasonable initial assignment of households to properties. We solved this problem with a trick: In Experiment 1 we let all households be externals that want to move into the town at the same time. The town initially has no inhabitants. We assigned all households the same urgency and let them compete for offers kind of synchronously. This is a kind of extreme test for the simulator, creating a huge concurrent activity with a large number of acceptance rejections until more and more household find a property and settle down for a while. The assumption was that if the supply of properties is fair, all will finally find something, maybe not really satisfied but content for a while. This stable situation was assumed to be a good initial condition for other experiments.

In Experiment 2 we softened the concurrency some by giving each household a random urgency, proportional to the amount of dissatisfaction. All delays were calculated as reverse proportional to the urgency with some offset.

In Experiment 3 we used the result of Experiment 2 after 100 days and kind of started a new phase. Now dynamic changes in household preferences and profiles and additional properties were introduced to observe a normal migration behavior. Still, also this experiment is not meant to give new
Multi Agent Model to Multi-Process Transformation

insides into human behavior, but rather a validation of the models, decision processes and parameters.

The results from Experiment 1 and 2 are nearly identical, although in 1 it takes 170 days before a stable condition is reached, compared to 65 days in 2. We therefore only report on the first 70 days of Experiment 2. During this time 471 households found a property, all local markets together sent 3594 offers, each with a maximum of three properties. In addition, 1673 offer came back empty. 1823 accept messages were sent, but 1352 rejected as already occupied by a household that had been faster. The relation of 7 offers per household seems to be high, but it has to be taken into account, that the offer selection is only based on tenure, size and cost requests only and many are turned down as not satisfactory. The fact that 75% of the accepted offers are already occupied is not astonishing, given the initial condition. Figure 7 shows how quickly the average satisfaction changes from negative to positive and in parallel the rent income grows. All properties together would reach 500 thousand Euro if occupied.

![Figure 7. Change of average households and total rent paid during the first 10 weeks of Experiment 2.](image-url)

In Experiment 3 the first 100 days are set up as in experiment 2 to assign properties to households. At 100 a new satisfaction evaluation starts with the goal of moves in the town to improve the utilities. In addition, over a period of 120 days 100 households get new random work locations which is also a trigger to reevaluate the satisfaction. First tests showed that under these boundary conditions no moves occur because the existing free properties are not attractive. Therefore, 80 additional free properties in the suburbs are
added at day 130. The result is an average satisfaction of 0.04 at day 100, 42 additional moves with a final average satisfaction of 0.09 after 800 day. This is still far off the maximum of 0.25, but we did not try to adjust the housing market to the household preferences in the first place.

Common to all three experiments is the run-time of 10 minutes for a simulation period of 2 years. Memory requirements are about 5 MBytes. The simulation is executed with a laptop with an 850MHz CPU.

6. CONCLUSION

It has been shown that a fully documented and tested prototype simulator for a housing case study can be developed in three person weeks, using the introduced software process, modeling techniques, and appropriate tools. It has also been shown that the agent based model is capable of simulating individual instance behavior with a large range of properties, using a small number of object types. The used models do not claim to be correct abstractions of real user activities and decision processes. Also, the parameters used for user preferences and profiles are not based on empirical study.

But despite these deficiencies, the simulation results show reasonable and sometimes astonishingly realistic household behaviors. With better statistical data we are sure that it can be further improved. But, most of all, the simulator is meant to be a basis for experiments with more sophisticated decision models. Such models can be easily implemented and tested by stepwise refinement of the agents. As such, it can be used as a tool for research in this field.

The experiments were limited to 500 agents. At 2000 agents model compile times became too large for fast turn-around times. Beyond that number compiler errors occurred. If larger numbers are required, the model documents can be used as specifications for simulator implementations in more efficient agent environments. We are looking forward to results from Devisch (Devisch 2005).

7. REFERENCES

ESP-r, http://www.esru.strath.ac.uk/Programs/ESP-r.htm
Telelogic Tau SDL Suite, http://www.telelogic.com