What Software Engineering has to offer to Agent-Based Social Simulation

By Peer-Olaf Siebers (peer-olaf.siebers@nottingham.ac.uk) and Franziska Klügl (franziska.klugl@oru.se)

Please cite as: Siebers PO and Klügl F (in press) 'What Software Engineering has to offer to Agent-Based Social Simulation'. In: Edmonds B and Meyer R (Eds). Simulating Social Complexity: A Handbook - 2e

6.1 Introduction

...

6.2 Review of Formal Approaches to Model Development

...

6.3 Illustrative Example: Normative Comparison in an Office Environment

Up to now we have seen that Software Engineering in general and AOSE in particular offers a lot of support for developing ABSS models. Most of this support can be coined "formal": at the heart are clearly given process models describing the different steps to go through when doing a simulation study. This is particularly important for less experienced modellers as these process models help to solve the problem of translating vague mental representations of models into descriptions that are more and more refined. These methodologies help to know where one should start when doing a simulation study.

In the following we show based on an illustrative example that there is no need to be afraid of formal approaches, but that they can indeed be useful to support awareness about the actual model content when developing a model.

6.3.1 Our Structured Approach

When aiming to develop ABSS models one faces the question of how to build them and where to start. This can be challenging not only for novices in the field but also for multidisciplinary teams where it is often difficult to engage everyone in the modelling process. Over the years we have developed a quite sophisticated "plan of attack" in form of a framework that guides the model development and can be used by either individuals or teams.

When used by individuals, they need to consider the perspective of potential team members (i.e. slip into their roles) during each process step. When used by teams, co-creation is an important aspect. Team members need to be open minded about the use of new tools and methods and about the collaboration with researchers from other domains and business partners. This is often not easy either for researchers trained in more traditional approaches or for business partners who often expect researchers to act like consultants, providing them with a report and a list of recommendations (Mitleton-Kelly 2003).

The framework supports model reproducibility through rigorous documentation of the conceptual ideas, underlying assumptions and the actual model content. The framework provides a step-by-step guide to conceptualising and designing ABSS models with the support of Software Engineering tools and techniques. Figure 2 provides an overview of the steps that make up the development process.

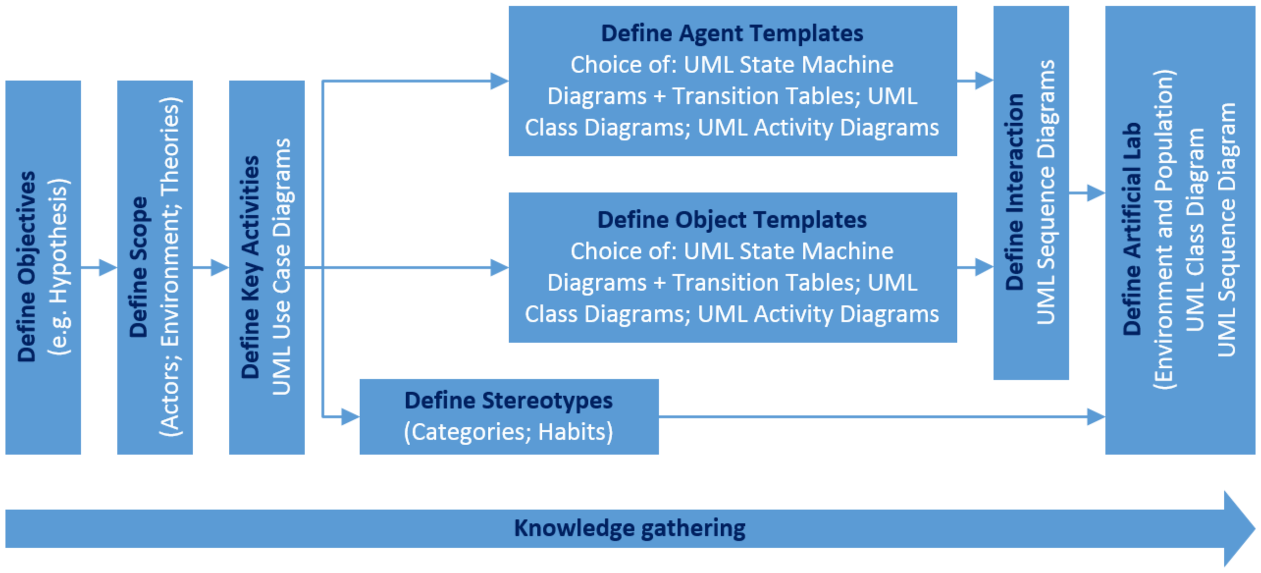


Figure 2: Overview of our ABSS development framework

While the framework represents a structured modelling approach, there will always be iterations required by the users to improve definitions from previous tasks. When stepping through the framework the users may realise that they did not consider important elements/details in a previous step or that they considered too many or that they considered them wrongly. In particular *discussions in focus groups* unearth these kinds of issues and are therefore extremely valuable for the model development process. The framework is suitable tool for well organised discussions and to capture the knowledge and ideas coming out of these discussions in a formal way. While there is a given sequence of steps that users should follow they need to be prepared to go back to a previous task if required and apply changes. Consequently this means that the users do not have to worry too much if in the initial rounds they get things wrong or things feel incomplete. They should simply move on to the next task if they feel that they have some form of contribution. Our experience is that it is necessary to revisits each task 4-5 times before there is a satisfying result that is acceptable for all stakeholders. In that sense, the approach somehow resembles "agile" approaches of Software Engineering with frequent interactions with stakeholders and frequent iterations, and not investing a lot of time into specifications that are obsolete after the next discussion.

While this framework will not work perfectly for all possible cases, it provides at least some form of systematic approach. The user should be prepared to adapt it to fit individual needs. In the following we will explain each step (including the necessary tools) and exemplify its application.

In order to demonstrate the use of our structured approach we use an illustrative example which is based on work by Zhang et al. (2011), Susanty (2015), and Bedwell et al. (2014). In this example we focus on the simulation model development to support studying the impact of normative comparison amongst colleagues with regards to energy consumption in an office environment. Normative comparison in this context means giving people clear regular personalised insight into their own energy consumption (e.g. "you used x% more energy than usual for this month") and allowing them to compare it to that of their neighbours (e.g. "you used x% more than your efficient neighbours"). A simulation study could compare the impact of "individual apportionment" vs. "group apportionment" of energy consumption information on the actual energy consumption within the office environment.

6.3.2 Gathering Knowledge

The task of knowledge gathering is one that happens throughout the structured modelling approach and in many different ways. The main ones we use in our framework are *literature review*, *focus group discussions*, *observations*, and *surveys*. The knowledge gathering is either a prerequisite for tasks (e.g. a literature review) or is embedded within the tasks (e.g. focus group discussions). For our study all focus groups were led by a Computer Scientist (the initiator of the study) and the participants consisted of a mixture of academics and researchers from the fields of Computer Science, Business Management, and Psychology. In this example study we did not engage with business partners. The team consisted of five core members that would participate regularly in the focus groups. Over the years we have made the experience that for our purposes smaller focus groups work best. Whenever we describe a task in the following we also briefly mentioned when and how the required knowledge was gathered.

6.3.3 Defining the Objectives

The first step within the framework is to define objectives in relation to the aim of the study. In our case this was done through a combination of a *literature review* and *focus group discussions*. After some iteration we came up with the following:

- Our aim is to study normative comparison in an office environment
- Our objective is to answer the following questions:
    - What are the effects of having the community influencing the individual?
    - What is the extent of impact (significant or not)?
    - Can we optimise it using certain interventions?
- Our hypotheses are:
    - Peer pressure leads to greener behaviour
    - Peer pressure has a positive effect on energy saving

With the objectives defined we then need to think about how we can test these objectives. For this we need to consider relevant experimental factors and responses. Experimental factors are the means by which it is proposed that the modelling objectives are to be achieved. Responses are the measures used to identify whether the objectives have been achieved and to identify potential reasons for failure to meet the objectives (Robinson 2004). In other words, experimental factors are simulation inputs that need to be set initially to test different scenarios related to the objectives while responses are simulation outputs that provide insight and show to what level the objectives have been achieved. In our case the hypotheses are very helpful for defining an initial set of experimental factors and responses:

- Experimental factors
    - Initial population composition (categorised by greenness of behaviour)
    - Level of peer pressure ("individual apportionment" vs. "group apportionment")
- Responses
    - Actual population composition (capturing changes in greenness of behaviour)
    - Energy consumption (of individuals and at average)

The experimental factors and responses defined at this stage are still very broad and need to be revisited when more information about the model becomes available.

6.3.4 Defining the Scope

At this stage we are interested in specifying the model scope. This requires some initial knowledge gathering. We did this through a *literature review* and *observation* of the existing system. With the help of the knowledge gathered we were then able to define the scope of the model. Decisions were made through *focus group discussions*. To guide the discussion and to document the decisions made in a more formal way we used an adaptation of the conceptual modelling *scope table* proposed by Robinson (2004) specially tailored towards ABSS modelling. The general categories we consider are: "Actor", "Physical Environment", and "Social / Psychological Aspect".

In order to make decisions about including or excluding different elements within these categories we asked ourselves, amongst others, the following questions:

- What is the appropriate level of abstraction for the objective(s) stated before?
    - This would define the level of abstraction acceptable.
- Do the elements have a relevant impact on the overall dynamics of the system?
    - Then they should be included.
- Do the elements show similar behaviour to other elements?
    - Then they should be grouped.

After some *discussions within the focus group* we decided that "transparency" would be the key driver for our decision making and that we want to abstract/simplify as much as possible while still keeping a realistic model (i.e. we aimed to explicitly follow the KISS principle mentioned in Section 6.2.1). In order to have easy access to data we decided to use our own offices (University of Nottingham; School of Computer Science) as the data source. Table 1 presents the resulting scope table in which we state for every element whether we want to include or exclude it and why we decided either way.

Category		Element	Decision	Justification
Actor		Staff	Include as group (User)	Regularly occupy the office building
		Research fellows		
		PhD students		
		UG+MSc students	Exclude	Do not have control over their work environment
		Visitors	Exclude	Insignificant energy use
Physical Environment	Appliance	HVAC (Heating + Ventilation + Aircon) system	Exclude	We only need one major energy consumer to test the theory; we decided to go for electricity
		Lighting	Include	Interacts with users on a daily basis; controlled by user
		Computer	Include	Interacts with users on a daily basis; controlled by user
		Monitor	Exclude	Modelled as part of the computer
		Continuously running appliances	Exclude	Constant consumption of electricity; not controllable by individuals
		Personal appliances	Exclude	No way to measure consumption
	Weather	Temperature	Exclude	Not necessary for proof-of-principle
		Natural light level	Exclude	Not necessary for proof-of-principle
	Room	Office	Include	Location where electronic appliances are installed
		Lab	Exclude	Mainly used by UG+MSc
		Kitchen	Include as group (Other Room)	Common areas frequently used by "users"
		Toilet		
		Corridor	Include	Commonly used when "users" move around
Social / Psychological Aspect		Comparative feedback	Include	Effective strategy to reduce energy consumption in residential building
		Informative feedback	Include	Effective strategy to remove barriers in performing specific behaviour
		Apportionment level	Include	Potential strategy to reduce energy consumption in office building
		Freeriding	Include	Behaviour that differentiate two apportionment strategy
		Sanction	Include	Factor to encounter freeriding behaviour
		Anonymity	Include	Factor to encounter freeriding behaviour

Table 1: Scope table for our illustrative example

6.3.5. Defining Key Activities

Interaction can take place between actors and between an actor and the physical environment it is in. Capturing these at a high level can be done with the help of *UML use case diagrams*. In Software Engineering *UML use case diagrams* are used to describe a set of actions (use cases) that some system or systems (subject) should or can perform in collaboration with one or more external users of the system (actors). These diagrams do not make any attempt to represent the order or number of times that the systems actions and sub-actions should be executed. The relevant components of a use case diagram are depicted and described in Table 2.

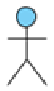
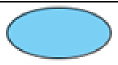
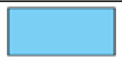
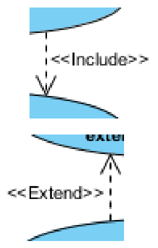
Component	Symbol	Description
Actors		Entities that interface with the system (this can be people or other systems). Think of actors by considering the roles they play.
Use cases		Denotes what the actor wants your system to do for them.
System boundary		Indicates the scope of your system: The use cases inside the rectangle represent the functionality that you intend to implement.
Relationships		There are different types of relationships. In a relationship between use case and actor the associations indicate which actors initiate which use cases. A relationship between two use cases specifies common functionality and simplifies use case flows. We use <<Include>> when multiple use cases share a piece of same functionality which is placed in a separate use case rather than documented in every use case that needs it. We use <<Extend>> when activities might be performed as part of another activity but are not mandatory for a use case to run successfully. We are adding more capability.

Table 2: Relevant use case diagram components

While in Software Engineering the actors are outside the system boundaries (they are usually the users of software, and the software represents the system) when using use case diagrams in an ABSS context the actors are inside the system (representing the humans that interact with each other and the environment). The system boundaries are the boundaries of the relevant locations (which in our case would be the building boundaries of the office environment). It is important to understand that the purpose of these diagrams is to promote understanding; as long as they capture the ideas and help to explain them they are very useful. The use case diagram which we developed for our illustrative example through *focus group discussions* is depicted in Figure 3.

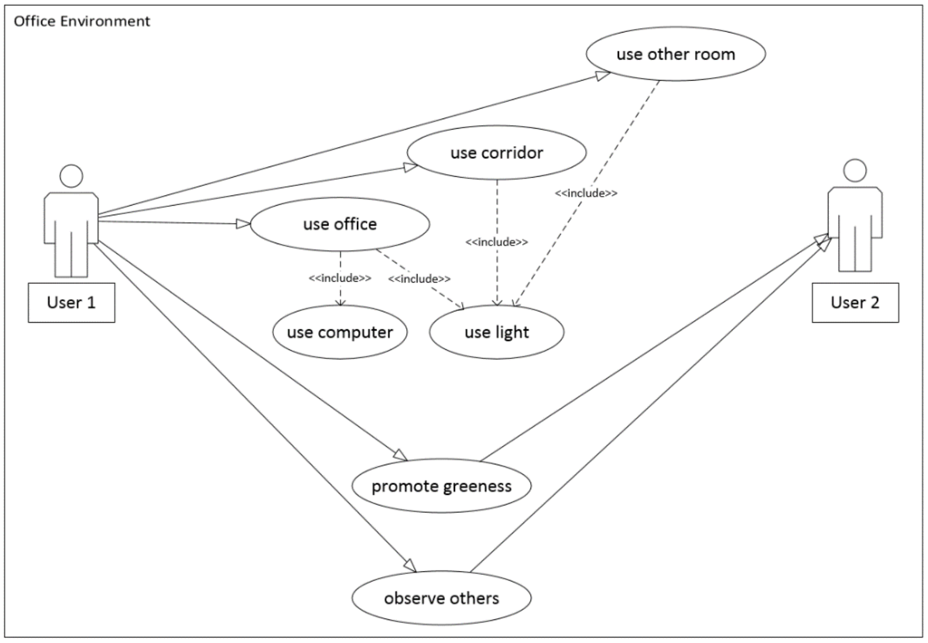


Figure 3: Use case diagram for our illustrative example [drawn with Visio]

6.3.6 Defining Stereotypes

In social psychology, a stereotype is a thought (or belief) that can be adopted about specific types of individuals or certain ways of doing things (McGarty et al. 2002). In order to be able to represent a specific population in our simulation models we define stereotypes that allow us to classify the members of this population. We derived our stereotype templates (categories, habits to be considered, and type names) through *focus group discussions* and through considering the *knowledge gathered previously*. Getting the stereotype templates right is more an art than a science. After long debates we decided to have two categories of stereotypes: one related to "work time", the other related to "energy saving awareness". Once the categories were identified we had to come up with the habits that describe these stereotypes:

- Habits for work time category
  - Arrival time at office
  - Leaving time from office
- Habits for Energy Saving Awareness category
  - Energy saving awareness
  - Likelihood of switching off unused electric appliances
  - Likelihood of promoting greenness

To get the information we needed to fully define the stereotypes we conducted a *survey* amongst our school's academics, researchers, and PhD students, anonymously asking them questions about their habits towards work time and energy saving awareness. We then *analysed the data* through cluster analysis to come up with the stereotype groups, assigned some speaking name and populated the stereotype tables with the "habit" information. The stereotype definitions we ended up with can be found in Table 3 and 4.

Stereotype	Working days	Arrival time	Leave time
Early bird	Mon-Fri	5am-9am	4pm-7pm
Time table complier	Mon-Fri	9am-10am	5pm-6pm
Flexible worker	Mon-Fri	10am-1pm	5pm-11pm
Hardcore worker	Mon-Fri + Sat	8am-10am	5pm-11pm

Table 3: User stereotypes defining work time habits

Stereotype	Energy saving awareness [0-100]	Probability of switching off unnecessary appliances	Probability of sending emails about energy issues to others
Environmental champion	95-100	0.95	0.9
Energy saver	70-94	0.7	0.6
Regular user	30-69	0.4	0.2
Big user	0-29	0.2	0.05

Table 4: User stereotypes defining energy saving habits

6.3.7 Defining Agent and Object Templates

For each of the relevant actor types we have identified in our scope table we have to develop an agent template containing all information for a prototypical agent. These templates will act as a blueprint when we later create the actor population for each simulation run. When it comes to modelling the environment we need similar templates for everything relevant we have identified in the scope table that lends itself to be represented as an object (e.g. the appliances). For other things (e.g. the weather) we need to consider other modelling methods. From a technical point of view there is no big difference between agents and objects. Thus we can use the same types of diagrams to document their design. We will therefore use the term "entity" when we talk about both. There are three different diagram types that are of relevance for defining entity templates: UML class diagrams (to define structure), UML state machine diagrams (to define behaviour), and UML activity diagrams (to define logic). Often only a subset of these is required. When developing the templates we create the different diagrams in parallel and in an iterative manner as often one informs and inspires the development of the other. As with the stereotypes, getting the entity templates right is not hard science and will therefore require many iterations.

In Software Engineering *UML class diagrams* are used to define the static structure of the software to be developed by showing classes (which are blueprints to build specific types of objects) and the relationships between classes. These relationships define the logical connections between classes (association, aggregation, composition, generalisation, dependency). UML class diagrams can be very complex and for our purposes it is often enough to consider individual classes. Therefore we focus on how to define individual classes here. In UML classes are depicted as rectangles with three compartments. The first compartment is reserved for the class name. This is simply the name of the entity as defined in the scope table (e.g. "User" for our user agent template). The second compartment is reserved for attributes (constants and variables). Often we would capture key state variables (e.g. "energy saving awareness"), key parameters and key output variables (e.g. "own energy consumption") here. The third compartment is reserved for operations that the user may perform. For each operation, we define some function names that indicate what kind of additional code we have to produce later (e.g. "moveToNewLocation()"). The brackets indicate that this is a function. Figure 4 shows as an example the user class definition we developed in parallel with the other template diagrams in several *focus group discussion* sessions.

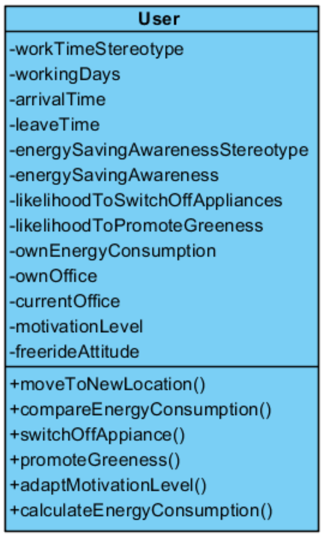


Figure 4: User class definition [drawn with Visual Paradigm]

In Software Engineering *UML state machine diagrams* (sometimes just called "state chart") are used to represent the dependencies between the state of an object and its reaction to messages or other events. State machine diagrams show the states of a single object, the events or the messages that cause a transition from one state to another and the action that result from a state change. A state machine diagram has exactly one state machine entry pointer which indicates the initial state of the agent. A state in a state machine diagram models a situation during which some invariant condition holds. Usually time is consumed while an object is in a specific state. A simple state is a state that does not have sub-states while a composite state is a state that has sub-states (nested states). The relevant components of a state machine diagram are depicted and described in Table 5.

Component	Symbol	Description
Entry pointer	|•→	Indicates the initial state after an object is created
State	▭	Represents a locus of control with a particular set of reactions to conditions and/or events
Initial state pointer	•→	Points to the initial state within a composite state
Final state	◉	Termination point of a state chart
Transition	→	Movement between states, triggered by a specific event
Branch	◇	Transition branching and/or connection point
Shallow history	Ⓗ	The state chart remembers the most recent active sub state (but not the lower level sub-states)
Deep history	Ⓗ*	The state chart remembers the most recent active sub state (including the lower level sub states)

Table 5: Relevant state machine diagram components

In our case we use state machine diagrams to define the behaviour of our entities. This type of diagram is particular useful as it can be automatically translated into source code by IDEs who support such features. One can use several diagrams (e.g. one representing physical states and one representing mental states) for the same entity. A state machine diagram is not always meaningful (e.g. if there are no relevant states that need to be represented to capture the behaviour) or necessary (e.g. "energy saving awareness" could be expressed in states "aware" and "not aware" but also as a state variable that represents the level of awareness). There is nothing wrong with having entity templates without state machine diagrams.

While for Software Engineering the descriptions of how transitions are triggered are usually embedded within the diagram (in a rather cryptic language) it might be a good idea to present them in a separate table, to make the diagram easier to understand.

Many people find it difficult to get started with developing the state machine diagrams for agent templates. In order to come up with potential states that an agent can be in it helps to think in terms of *locations* (e.g. "in office"). The next step would be to think about *key time consuming activities* within these locations (e.g. "working with computer"). It is important to consider only key locations and key activities as otherwise the state chart gets too complex. One should only define as much detail as is really necessary for investigating the question studied. The above steps are just suggestions and do not always work. In case they do not work one has to use intuition and try to draft something that "feels right".

Figure 5 shows as an example the user state machine diagram we developed in parallel with the other template diagrams in several *focus group discussions*. Here we have defined location states based on the relevant rooms we identified in the scope table and added one location ("outOfOffice") to represent the outside world. The ideas for the activity states stem from our use case diagram (Figure 3). We then added transition arrows to represent the possible transitions between the defined states. Transitions with a question mark symbol are condition triggered while transitions with a clock symbol are time triggered. If there are more than one transition connecting states we have considered different triggers for state changes. This becomes clearer when we look at the transition definitions in Table 6. Here we can see that for example a state change from "outOfOffice" to "inCorridor" can happen for all user stereotypes during the working week and only for hard-core worker user stereotypes during the week end.

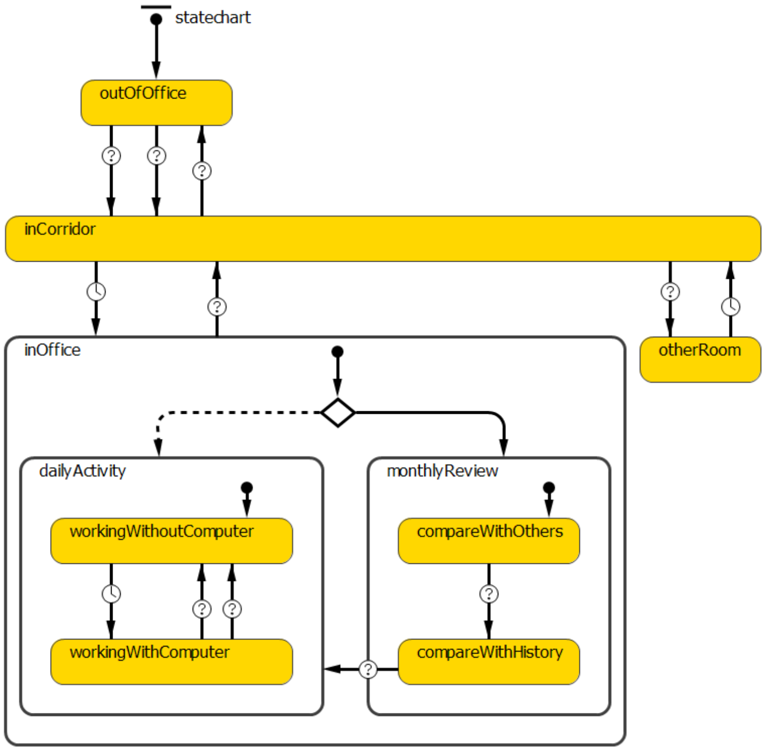


Figure 5: User state machine diagram [drawn with AnyLogic]

From state	To state	Triggered by	When?
outOfOffice	inCorridor	Condition	At typical arrival time during the working week for all
outOfOffice	inCorridor	Condition	At typical arrival time on Saturdays for hard-core workers only
inCorridor	outOfOffice	Condition	At typical leave time
inCorridor	inOffice	Timeout	At average after 5 minutes
inOffice	inCorridor	Condition	At random while at work or when leaving
inCorridor	otherRoom	Condition	At random while at work
otherRoom	inCorridor	Timeout	At average after 10 minutes
...	...	...	...

Table 6: User state machine transition definitions (excerpt)

In Software Engineering *UML activity diagrams* describe how activities are co-ordinated (the overall flow of control). They represent workflows of stepwise activities (while state machine diagrams show the dynamic behaviour of an object) and actions with support for choice, iteration and concurrency. Often people describe activity diagrams as just being fancy flow charts. The relevant components of an activity diagram are listed in Table 7.

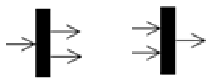
Component	Symbol	Description
Activity		Named box with rounded corners (a state that is left once the activity is finished)
Activity edge	→	Arrow (fires when the previous activity completes)
Synchronisation bar		Represent the start (split) or end (join) of concurrent activities
Decision diamond		Used to show decisions
Start marker	◉	Indicate entry point of the diagram
Stop marker	●	Indicate exit point of the diagram

Table 7: Relevant activity diagram components

Amongst others, we can use these activity diagrams as a formal way to describe a decision making process (logic flow). In our case we use it to describe the logic flow of the normative comparison process. In order to define the logic flow we use the information we gathered from our *literature review* on psychological factors in the scoping phase. Figure 6 shows as an example the actions happening when the user agent is in the state "compareWithHistory" (which in the model is triggered once per simulated month). It is good practice to provide some evidence from the literature for the rationale behind the decision making process. This would come from our scoping phase *literature review* but might also require some additional resources. As an example, let's take the case "Less than former month? = no / Group? = yes / Sanction? = yes / Not Anonymous?". In the literature we find that using mechanisms to identify freerides and implement sanctions (social (e.g. gossip) or institutional (e.g. fines)) reduces the likelihood of further free-riding (Fehr et al. 2002). This is our justification for adding the action "decrease freeriding" for this case. In the end we would evaluate our logic flow by *discussing it in the focus group*.

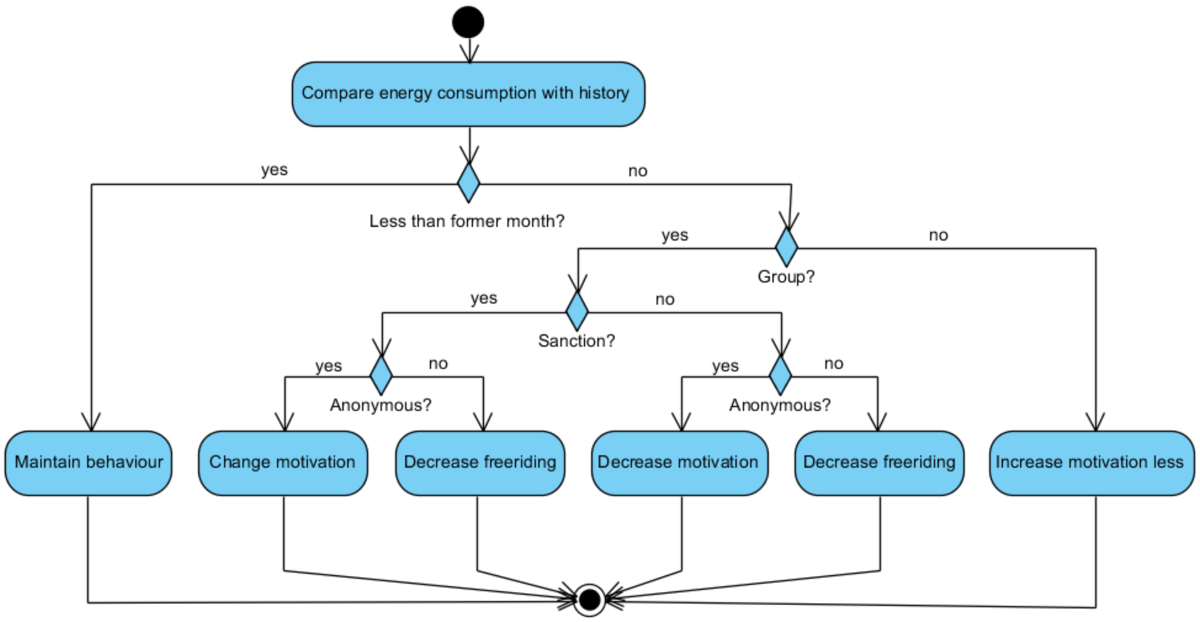


Figure 6: Activity diagram for user agent state "compareWithHistory" [drawn with Visual Paradigm]

6.3.8 Defining Interactions

As we saw in section 6.3.5 capturing interactions on a high level can be done using *UML use case diagrams*. Capturing interactions in more detail can be done by using *UML sequence diagrams*. This can be used to further specify use cases that involve direct interactions (usually in form of message passing) between entities (agents and objects).

In Software Engineering *UML sequence diagrams* are used primarily to show the interactions between objects in the sequential order that those interactions occur. Often they depict the actors and objects involved in a specific use case realisation and the sequence of messages exchanged between the actors and objects needed to carry out the functionality of the use case realisation. But sometimes they also capture wider scenarios that go beyond a specific use case. The relevant components of a sequence diagram are listed in Table 8.

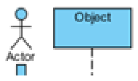
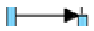
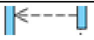
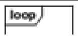
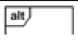
Component	Symbol	Description
Lifeline		Named element which represents an individual participant in the interaction
Message		From sender to receiver
Message		Return message
Execution		Represents a period of time in which the participant is active
Message		Self message
Loop		Wrapper for representing loops (has one compartment)
Alternative		Wrapper for representing alternatives (has as many compartments as alternatives exist)

Table 8: Relevant sequence diagram components

In our case we discussed the technical way of implementing the "observe others" use case during one of our *focus group discussions*. Figure 7 shows the sequence diagram we developed during our discussion for this use case. The entities involved are users and units that provide information. Users interact with information units and with each other. Information units interact with the users and with each other.

Creating this diagram sparked a discussion around if we should consider a database that stores historic information in our model or not. It is currently not represented in the scope table (Table 1). In the end we agreed that for our initial model we will leave it out but keep a record of it in the scope table as it might be something we want to consider in the future. We then removed it from the final version of our sequence diagram.

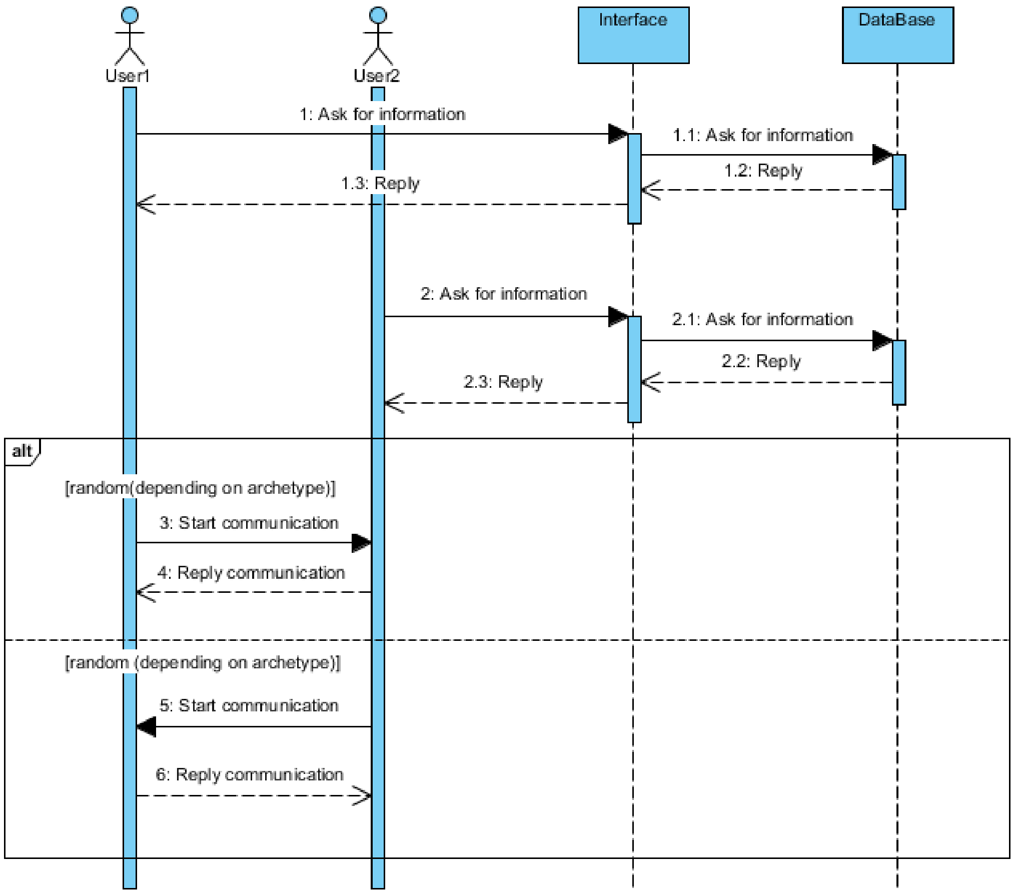


Figure 7: Initial sequence diagram for the use case "observe others" [drawn with Visual Paradigm]

6.3.9 Defining the Artificial Lab

Finally we need to define an environment in which we can embed all our entities and define some global functionality. We call this environment our "artificial lab". For the development of our artificial lab we use a class definition as described in Section 6.3.7. Within this class definition we consider things like global variables (e.g. to collect statistics), compound variables (e.g. to store a collection of agents and objects), and global functions (e.g. to read/write to a file). We also need to make sure that we have all variables in place to set the experimental factors and to collect the responses we require for testing our hypotheses. We derive our class content through *focus group discussions.* To inform these discussions we need to look at our list of objectives (see Section 6.3.3) and our scope table (see Section 6.3.4). The final class definition should only contain key variables and functions. Figure 8 shows the "Artificial Lab" class definition for our illustrative example. Variable names including "[]" represent collection variables.

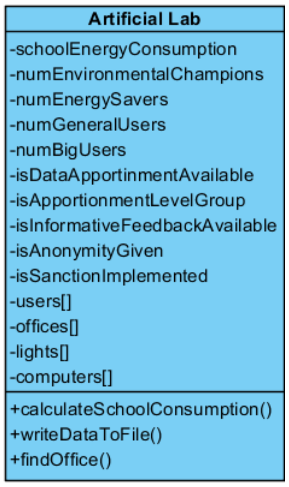


Figure 8: Artificial Lab class definition [drawn with Visual Paradigm]

Sometimes it can be helpful to create a sequence diagram as described in Section 6.3.8 to visually show the order of execution describing the actions taken on various elements at each step of the simulation from a high level approach. The way and order in which all entities are initialised, as well as the way and order how they are updated and how their interactions are handled, is often not trivial and a major source of artefacts. Therefore, in such a case it needs to be clearly documented and specified. In our illustrative example we do not have any obvious complex order dependencies and therefore it was not necessary to create such a high level sequence diagram.

At this point we have all the information together. Using the collection of diagrams and tables that we produced, the model to be handled should be fully specified and as well understood as it can be without running it. The next step is to take all the information and either start with the implementation or pass on the details to let a professional software developer deal with it.

6.4 Conclusion

...

Further Reading

There is a host of literature on the topic of Software Engineering. A book that provides a comprehensive yet easy to understand entry to most of the Software Engineering topics discussed in this book chapter is Lethbridge and Laganiere (2005). If you are mainly interested in learning more about UML then Fowler (2003) is sufficient. A lot of ideas for ABSS stem from the Computer Science field of Artificial Intelligence, and here in particular Multi-Agent Systems. A good overview on the wide area of topics (including AOSE) is Weiss (2013). Finally, the JASSS special issue "Engineering ABSS" (Siebers and Davidsson 2015) provides lots of information and case studies

References

Bedwell B, Leygue C, Goulden M, McAuley D, Colley J, Ferguson E, Spence A (2014). Apportioning energy consumption in the workplace: a review of issues in using metering data to motivate staff to save energy. Technology Analysis & Strategic Management, 26(10):1196-1211

Fehr E, Fischbacher U, Gächter S (2002) Strong reciprocity, human cooperation, and the enforcement of social norms. Human nature, 13(1):1-25.

Lethbridge TC, Laganiere R (2005) Object-Oriented Software Engineering: Practical Software Development Using UML and Java: Practical Software Development, McGraw Hill

McGarty G, Yzerbyt VY, Spears R (2002). Social, cultural and cognitive factors in stereotype formation. McGarty G, Yzerbyt VY, Spears R (eds.) Stereotypes as explanations. New York: Port Chester, 1-15.

Mitleton-Kelly E (2003). Complexity Research - Approaches and Methods: The LSE Complexity Group Integrated Methodology. A Keskinen, M Aaltonen, E Mitleton-Kelly (eds.) Organisational Complexity. Tutu Publications. Finland Futures Research Centre, Turku School of Economics and Business Administration, Turku, Finland, pp. 56-77.

Robinson S (2004). Simulation: The practice of model development and use. John Wiley & Sons: Chichester, UK

Siebers PO, Davidsson P (2015) Engineering Agent-Based Social Simulations: An Introduction (Special Issue Editorial). Journal of Artificial Societies and Social Simulation, 18(3) < http://jasss.soc.surrey.ac.uk/18/3/13.html>.

Siebers PO, Figueredo GP, Hirono M, Skatova A (2017) Developing Agent-Based Simulation Models for Social Systems Engineering Studies: A Novel Framework and its Application to Modelling Peacebuilding Activities. Garcia-Diaz C, Olaya Nieto C (eds.) Social Systems Engineering: The Design of Complexity. John Wiley & Sons.

Susanty M (2015) Adding psychological factors to the model of electricity consumption in office buildings. MSc Dissertation, Nottingham University, School of Computer Science.

Weiss G (2013) (ed.) Multiagent Systems, 2nd Edition, MIT Press, Cambridge

Zhang T, Siebers PO, Aickelin U (2011) Modelling electricity consumption in office buildings: An agent based approach. Energy and Buildings, 43(10), 2882-2892.