

# Refactoring of Agent-based Simulation Models

Cornelia Triebig and Franziska Klügl

Department for Artificial Intelligence  
University of Würzburg  
Am Hubland  
D-97074 Würzburg  
{trieb, kluegl}@informatik.uni-wuerzburg.de

**Abstract:** There is a general agreement that the design and development of multiagent simulation models is highly time and effort consuming – like the development of any simulation model. Reusability of complete or partial models would reduce the modeling effort enormously and thus is highly attractive also for agent-based simulation models. However, there are some prerequisites that can be fulfilled using refactoring methods: Improved readability, understandability and thus maintainability of the model, performance gains and configuration ease. In this contribution we therefore elaborate categorized refactoring methods for agent-based simulation models and relate them to an abstract methodology for developing reusable MASim framework models.

## 1 Motivation

Multiagent Simulation (MASim) is a powerful modeling paradigm. It forms a technique that is especially apt for analyzing systems consisting of intelligent autonomous entities interacting with each other and their dynamic environment. Due to its ability to provide appropriate means for representing flexible behaviour, complex interactions or variable and heterogeneous structures, MASim experiences increasing popularity. In some application areas like social science or biology, it is becoming an established simulation technique with a broad usage. Successful uses of MASim could have been also documented for technical domains in industrial context. An example is the application of MASim for simulating virtual high bay warehouses for testing their complex control software [Tr05].

However, the development of MASim models is still a highly time and effort consuming task. Models in general, are thoroughly constructed and validated for fulfilling the objectives of the simulation study they are developed for. Thus, the development of a model means a great investment of time and thus money, especially when the model is more complex as MASim models are usually in comparison. This is due to the fact that they tend to reproduce their original system on a less abstract level than more traditional simulation paradigms do. Reusability of models or model components is therefore a basic prerequisite for actual usage in industrial context as it can be seen in different application domains. For example, traditional microscopic traffic simulation models for reproducing traffic flow are used for different traffic networks by merely calibrating them: their parameters are adapted in a way that given data is reproduced by the reused simulation model. Also, standard tools for object-oriented simulation of production lines,

etc. are equipped with a library of predefined elements that resemble certain types of machines that can be reused as they are after setting the components' parameter.

Despite the number of platforms, tools and Integrated Development Environments (IDEs), nothing comparable to such reusability of simulation models is possible in MASim. Even more basically, there is still a lack of knowledge on how to support MASim construction in analogy to the development of Multiagent Systems (MAS) in general [DG05]. However, such support is necessary for developing models with a clear structure, easily understandable behaviour definitions, etc. Especially, for simulation models such properties are highly important, as credibility forms an essential prerequisite for model use.

Object-oriented software development is already well supported by methodologies for a systematic development, various refactoring methods, and appropriate languages for specification as well as for documentation of the resulting software. These techniques are on a mature level and frequently applied. They allow and support developing software that is well structured, provides appropriate interfaces, and can be configured using a suitable parameter set.

Central mechanisms for improving the structure of the software are the already mentioned refactoring methods. In this contribution we examine which refactoring methods known from object-oriented software engineering [Fo02] can be reasonably executed in MASim code improvement and how refactoring can be imbedded in a larger methodology for developing reusable MASim models.

The paper is structured as follows: We first give an introduction to issues in reusability of MASim models. In section 3, we present well-known refactoring from software engineering and relate them to requirements for MASim models. Then, we suggest a categorization of refactoring methods for MASim and present methods that we implemented and integrated into the agent-based simulation platform SeSAm [Se07]. After giving a short practical example for illustration, the paper ends with a conclusion.

## **2 Developing Reusable MASim Models and the Role of Refactoring**

Two forms of reusability can be identified for simulation models in general. On the one hand, a complete model can be reused when it is advanced to a general and configurable framework. On the other hand, partial models can be reused when they are further developed to self-contained building blocks or components providing appropriate interfaces. Additionally, MASims form a special kind of software, but they are software with all the problems of software that is frequently modified – e.g. by model tuning for improving the simulation performance, during the validation phase when the model is adjusted, etc. The improvement of the implemented simulation model using refactoring towards a restructured, comprehensible and cleaned up model is desirable.

Before suggesting appropriate refactoring methods for MASim model implementations, we would like to tackle an abstract methodology for developing reusable MASim framework models that integrates refactoring and shows when it is applied best. The starting point for our reusability method is a fully elaborated and validated model that ideally already has proven its usefulness in actual application. The goal is, to treat this model for allowing its application to similar problems scenarios. We suggest a process for transforming the given full model to a generic, configurable, but domain specific framework. Figure 1 illustrates the relation between complete model reuse (on the left) and the proposed process (on the right) schematically.

The first step in this transformation process forms the refactoring process as it provides the basis for the successive procedure. By refactoring a model it will become well-structured and comprehensible. This facilitates maintenance that has to be guaranteed for future modifications. After improving the model structure by applying proper refactoring methods, the model is parameterized by inserting or working out overall model parameters. This elaborated model is then documented completely. Well elaborated model documentation supports the comprehensibility and reproducibility not only of the model itself but also of the simulation run and its results. Since complete model reuse is based on pure adaptation and modification of an existing model exclusively, the model has to be fully or partially accessible or at least extensively configurable. The configurability procedure therefore, forms the concluding step towards a generic domain specific framework. A critical issue in all mentioned processes is to regard characteristics of MAS and MASim in particular.

Hence, we focus on refactoring of MASim models as essential preparatory work towards reusable models.

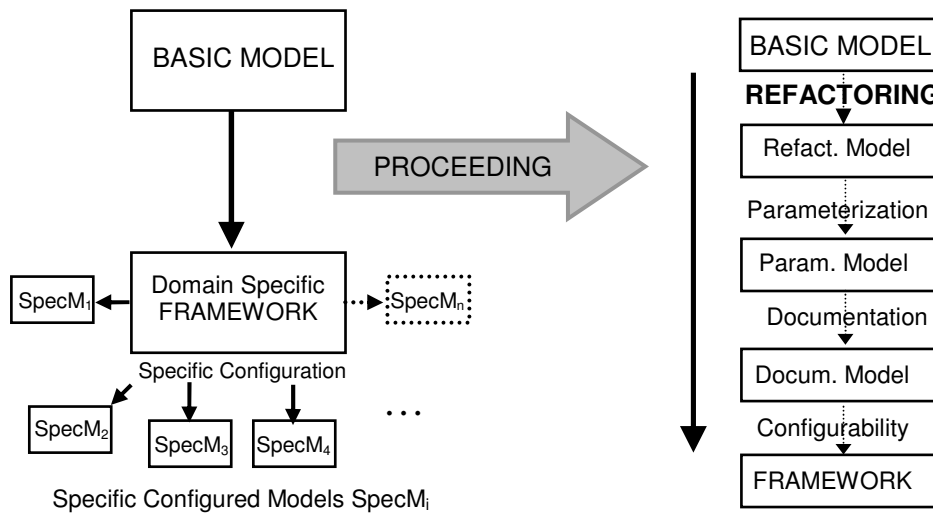


Figure 1 Model reusability: Proceeding to gain a configurable framework from an existing model.

### 3 Software Refactoring vs. MASim Model Refactoring

According to [Fo02] refactoring is seen as “the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure.” The focus of general software refactoring lies on treating the structure to facilitate maintainability and extensibility and to improve the readability of software. Refactoring is heavily tied to thorough (automatic) testing of the software for ensuring that structural changes do not affect the behavior of it. Clearly, it is not trivial to apply these methods.

Refactoring is a common practice in software development independently whether it is acknowledged and named as in rapid prototyping or agile software development [Be00] or it is simply done on the fly or less systematically. Refactoring is also desirable for MAS software and implemented MASim models. This was already expressed in [DG05] where basic refactoring functionalities are asked such as renaming, changing signatures, or moving elements for facilitating and accelerating model development.

As with software in general, model implementation takes time and is not a static specification to implementation phase. Model development is actually an iterative process as already shown in [Wi94]. Also, agile methods have already been motivated [Ri05] [CC07], however without reference to necessary refactoring support. In general, the motivation for refactoring in software development can be transferred to model implementation: improved comprehensibility of the agent implementation, facilitated understandability and maintainability of the model code. By improving the general structure of the simulation model, the credibility of the model is improved which is particularly important for simulation applications when models should be reused that were not implemented in the project itself. In the scope of reusable models model parameterization entailed by refactoring is also significant. Furthermore refactoring augments the testability of models by providing a cleared model structure.

Refactoring methods for MASim models however, pose some more challenges on their application than standard software does as testing becomes more effortful: When software behaviour and functionality have to be tested, dealing with simulation models the complete validation cycle has to be performed again for ensuring that the model is still sufficiently valid. Additionally, MASim models are per definitionem white box models that resemble the basic structure of the original Multiagent System. That means their structure is relevant – and should result in structural validity. However, there are always parts in a model responsible for more low-level functionality – if an agent sorts its preferences, the used sort mechanism is hardly part of the structurally valid aspects, but is selected for performance reasons.

## 4 Refactoring Methods for MASim Models

Software refactoring methods as introduced in [Fo02] are specific mostly for object-oriented languages. In the case of MASim, refactoring methods can be based on specific high-level representations for agent-based simulation models. Although it is in principle possible to implement a MASim model using standard programming languages, like Java, it makes more sense to consider specific high-level languages. Meanwhile, development tools are also available for high-level development.

Here, we want to focus on activity graph like behavior like in SeSAm [Se07] or hierarchical reactive plan representations like in RAP [FF00] or PRS-related agent architectures [GI89]. We apply refactoring as a process of changing a MASim model so that it does not alter the external behavior of the simulated system, yet improves its internal model structure without losing structural validity.

We remain it to be discussed, in which order to apply the following refactorings as this depends on several aspects like actual model structure, necessity of improvements, etc., which cannot be mapped onto a suggested refactoring proceeding.

**Extract-Parameter:** During rapid model development, absolute quantities like thresholds, action parameter, or initial values can be used directly in the model. These absolute values are hidden in model code. However, for testing the effect of these parameters systematically or calibrating their values, their treatment must be explicit. `Extract-Parameter` alters the model by replacing the absolute values by variables. The parameter is associated with the agents or other model parts where the quantity is actually used. This procedure lays the basis for model configurability and supports readability.

**Extract-Activity:** As a consequence of iterative model development, the overall behavior program, e.g. consisting of activities might become complex, the same partial plan is implemented in only a slightly different way again and again. The overall structure of the activity program can be clarified and its structure improved, when parts of the complex activity are extracted to new activity and an activation of the new one is inserted. This has not only the effect of improved structure, but again lays the foundation for reusability or simplified reproducibility of the extracted part. Similar refactoring methods can be defined for smaller elements of the behavior description (**Extract-Method**).

**Extract-Partial-Graph** corresponds to re-organization of the reasoning engine by inserting additional levels of aggregation – if the used language allows for such hierarchical behavior structures. It is applied when the behavior description becomes large and thus in-transparent without tackling additional levels of behavioral hierarchy. A partial skeleton plan or a partial activity graph is extracted and placed as a kind of subroutine.

**Inline-Activity:** Basically, this is the reverse refactoring to `extract-activity` and should be applied when the overall activity graph contains many, small activities with only simple connections between one-action activities. Such a large graph appears to be complex and thus confuses the modeler. `Inline-Activity` then merges activities to a new, more meaningful activity. Also, on a smaller grain, **Inline-method** can be defined analogously. For tackling larger structures with unnecessary complex hierarchical structures, one may define **Inline-Partial-Graph**.

**Rename:** The naming of model elements (e.g. variables, parameter, agents, etc) may be sub-optimal, as e.g. during its introduction, its naming was not important or the meaning slightly changed during development. However, the name of the element must correspond to its function and must be sufficiently long for actually being able to express the necessary information. `Rename` changes the model element name at every occurrence. This is a standard refactoring method that is provided by nearly every development tool. It improves readability and transparency of the model enormously.

**Insert-Assertion:** Frequently, assumptions made in a model, are only described some accompanying document or worse, are only stored in the brain of the modeller. `Insert-assumption` is a refactoring where assertions, guards, etc. are inserted into the behaviour code, stating what conditions have to be fulfilled at certain steps of the program. The failure of an assertion indicates an error by stopping the simulation. Assertions thus improve testability of a model and spare simulation time by cancelling simulations that run out of scope.

**Make-Calculation-Explicit:** A relict from fast prototyping can consist in condensed representations of calculations. Many elements are connected in one formula that can hardly be overseen by a human. `Make-Calculation-Explicit` identifies interesting partial values and introduces auxiliary variables for storing these values. A new variable is added used in the formula that hereby becomes accessible for human control. Auxiliary variables also support testing, etc. The reverse form of this refactoring corresponds to the inline refactoring, where unnecessarily often done auxiliary computations are integrated into one larger formula.

**Collect-Parameter:** Using the `extract-parameter` refactoring, we replaced absolute quantities in the model by explicit parameter. This makes them explicit and thus improves model clarity. However, for comfortable configuration, these parameter should be at hands of the modeller. When one assumes the existence of an explicit, overall model documentation and interface description), we could associate these parameter with this overall facility and thus define the interface of the overall simulation.

**Reduce-Parameter-Set:** The set of model parameters, like a threshold in the behaviour description of an agent, may also be too high (see *Extract-Parameter* for a related refactoring). In MASim, parameters tend to be highly dependent on others. Thus, there is a chance, that some of them can be computed based on the others' values. *Reduce-Parameter-Set* replaces a parameter by its calculation. Introducing a equation instead of a fixed value, relations between parameters are made explicit and noticeable in the model itself.

**Replace-by-Value:** Instead of using parameters calculations based on constants every time the value is accessed, one may use a refactoring similar constant folding to code optimization: computations are replaced by constant values. This refactoring is the reverse to *Make-Calculations-Explicit*. The aim of this refactoring is improving simulation speed. However, it is traded against the price of clarity, explicit relations are getting lost.

**Delete-Partial-Graph:** This refactoring method deals with dead and redundant code, much like dead-path-elimination in code optimization. This situation can clearly be the result of the iterative modeling process applied without permanent re-design efforts. With *Delete-Subgraph* redundant sub-graphs are cut off and removed. This refactoring may also be applied on lower levels of granularity, like **Delete-Activity**, etc.

**Delete-Redundant-Parameters:** During iterative model development, parameter may have been defined that turn out to be un-used or without influence in a sensitivity analysis. These redundant parameters are removed from model.

Agents may interact in a variety of ways: indirectly by manipulating the environment, directly and message based, etc. Here, we again can treated interaction refactoring on arbitrary detail. We introduce the interaction refactoring on a quite high abstraction level, based on so called "ports", instead of defining the ACL for communication or the protocols the agent follow. A port may be a shortcut for message-based interaction, another one for the reproduction of continuous emissions.

**Delete-Redundant-Output-Ports:** Output ports define with which actions the agent moves through its environment; some of them may turn out to be unused throughout the simulation runs. Thus, if no other agent addresses these ports, they can be deleted. **Delete-Redundant-Input-Ports** works analogously to the latter, the main difference is the determination when the ports are not used. This may happen because of a changed agent behaviour or environmental program.

**Bundle-Messages:** Also message-based protocols may develop in a way that design improvements are necessary. One example is sending only small parts of information per message. Depending on the type of message and the ordering of available information combining small messages to a larger one may reduce at least traffic on the communication channel, but also simplifies communication protocols. **Split-message** forms the reverse refactoring method that may be applied when protocols require sending really large messages.

**Fix Interaction Partner:** In MASim models, interaction partners for certain agents can be determined in several ways. Firstly, the agent evaluates its perceptions and addresses the agent that it has recently perceived again and again as in purely reactive agent architectures. The second option lies in using a mediator. Thirdly, the address of the other interaction partner could be stored. **Fix-Interaction-Partner** introduces the address of the interaction partner; other refactorings may exchange addressing methods appropriately and can be defined in a similar way.

Not only behavioural structures and the interactions need improvement, but also the (simulated) environment. However, there seems to be no significantly different refactoring methods existing for environments of MASims.

## 5 Experiences: Applied refactorings on High Bay Warehouses Models

In the scope of the cooperation project between the Department for Artificial Intelligence at the University of Würzburg and SSI Schäfer Noell GmbH (Giebelstadt) we apply MASim technology to an industrial problem: testing the control software for automatic high bay warehouses using MASim [Tr05]. For that the real warehouse (respectively its components) is mapped to a MASim model. The simulated warehouse therefore, serves as a testbed for the project specific control software. Meanwhile more than 20 virtual high-bay warehouses have been developed, illustrating the need for constructing a re-usable framework.

Different modules like conveyor elements, scales and scanners, storage units, but also human operators are the basic elements of a high bay warehouses. Each of these elements is treated as an agent. In addition to modelling interactions within the virtual high bay warehouse – e.g. a scanner agent tells a conveyor agent, that the transported piece is too heavy – also communication between virtual warehouse and the real warehouse control software has to be established

All warehouses are similar as far as this concerns the functionality of most of the basic elements. For supporting re-usability, we already developed a number of component agents that represent specific elements. However, it turned out that virtual high bay warehouses are quite individual and component level reuse may be too effortful. Thus, the complete model is reused and developed into a re-usable framework. Additionally, all simulation projects started before the actual construction of the warehouse. The consequence was that the system that we had to reproduce was not fully specified with partially large uncertainties how it will actually look like. Without the availability of

easily applicable refactoring, the overall project goal would not be fulfilled with that level of quality that we reached. In the following, we will shortly give details on what refactoring methods we used in what order.

We started the refactoring with an initial model cleanup by deleting all redundant model parts like test parameter, dead paths, ... This was followed by the renaming of all elements with a well engineered naming concept as we found a lot of very specific expressions which had to be generalized. Further we extracted and also inlined several agent activities to construct concise agent reasoning. Thereby, temporal constraints of the activities had to be regarded. A lot of hidden values had to be extracted to parameters for a facilitated model. Furthermore we made assertions explicit as this helps us to identify whether the model or the connected software to test is incorrect. For the purpose of further debugging we made calculations explicit and added auxiliary variables. For the starting and adjustment of the model we made a parameter collection. Thus, it is clear and concise which values can or have to be adjusted. At least we fixed interaction partners of several agents by storing their identification directly.

We experienced that permanent refactoring is essential in our project application. On the one hand this is due to several technical innovations which have to be integrated into the model. On the other a group of five people is working with the model. Hence it is hard to pursue a totally common modeling style.

## **6 Conclusion**

In this paper, we suggested a number of – partially quite obvious – refactoring methods for agent-based simulation models. Despite of the introduction of many structured engineering processes in general simulation engineering as well as for agent-based simulations, developing a model has quite artistic elements that require iterative proceedings with frequent modifications [Wi94]. Thus, being confronted with real-world requirements and constraints on the modelling process, refactoring – and that means basically an apparatus for reliable and comfortable model clean-up – becomes inevitable. Although many of the proposed refactorings are not particularly agent-system-specific, however we showed how the standard ones can be adapted and made the experience how useful they can be in an industrial context.

Refactoring has to be integrated into a systematic overall methodology for constructing re-usable MASims, much as we shortly suggested in figure 1. There, we did not consider elements like sensitivity analysis that is used to evaluate the influence of parameter values and other basic assumption on the simulation dynamics. Basically, such a sensitivity analysis is a prerequisite for many of the proposed refactoring as it is the main contributor for identifying redundancy.

A second critical aspect is that of structural validity. MASims can be seen as white box models that means as models that aim at resembling the structure of the original system. That means e.g. that there cannot be `move` refactorings arbitrarily executed as parameters or activities belong to an agent, not because of efficiency reasons, but as it is like this in the original system. We suggested a number of generally allowable refactoring, although some of them actually modify the structure of the systems.

## References

- [Be00] Beck, K.: *Extreme Programming Explained. Embrace Change*. 1st Edition, Addison Wesley, 2000
- [CC07] Clynh, N. Collier, R.: SADAAM: Software Agent Development An Agile Methodology. LADS/Durham Agents 2007,
- [DG05] Dastani, M., Gomez-Sanz, J.: Programming Multi-Agent Systems Technical Forum. Notes on PROMAS TF Meeting Budapest (2005)
- [FF00] Fitzgerald, W., Firby, R.J.: Dialog is Task Execution; Task Execution is Best Done Reactively; Therefore, Dialog Systems Call for a Reactive Task Execution Architecture, 2000
- [Fi05] Fipa: <http://www.fipa.org/> (2005)
- [Fo02] Fowler, M.: *Refactoring: Improving the Design of Existing Code*. Boston, MA, USA, 9<sup>th</sup> ed., Addison-Wesley (2002)
- [GI89] Georgeff, M. P., Ingrand, F.F.: Decision-Making in an Embedded Reasoning System. In: Sridharan, N. S. (editor): *Proceedings 11th Int. Joint Conf. on AI (IJCAI-89) USA*, 972–978. Morgan Kaufmann, San Mateo, Calif., November 1989.
- [KI07] Klügl, F.: *Towards a formal framework for multi-agent simulation models* (2007)
- [Ri05] Rixon, A., Moglia, M. & Burn, L.S. 2005, 'Bottom-up approaches to building agent-based models: discussing the need for a platform', presented to CABM-HEMA-SMAGET 2005 Conf.: Joint Conf. on Multi-Agent Modelling for Environmental Management, Bourg-Saint Maurice-Les Arcs, France, 21–25 March 2005.
- [Se07] Sesam: <http://www.simsesam.de/> (2007)
- [Tr05] Triebig C. et al.: Simulating Automatic High Bay Warehouses Using Agents. In *CEEMAS (2005)* 653-656
- [Wi94] Willemain, T. Insights on Modeling from a Dozen of Experts, *Operation Research*, 42(2):213-222 (1994)