

# Flexible Models of Service Systems Based on the ABAsim Architecture

Michal Lekýr  
Norbert Adamko

Faculty of Management science and Informatics  
University of Žilina  
Univerzitná 8215/1, 01026 Žilina  
E-mail: lekyr@utcpd.sk, Norbert.Adamko@fri.utc.sk

## KEYWORDS

Agent simulation, ABAsim architecture, service system modeling, CASE tool

## ABSTRACT

In the process of designing new service systems and studying operation of existing systems computer simulation becomes very useful for its flexibility and similarity of modeled systems with the real world. This paper gives a brief overview of agent oriented architecture ABAsim, which provides its users with high quality founding base for modeling large-scale service systems. ABAsim architecture is based on hierarchically organized rather reactive than proactive agents. This architecture was applied to model the operation of various types of service systems. These models have been designed according to the general model of a service system described in this article. Specialized CASE tool for design and creation of simulation models developed under the architecture is also discussed.

## INTRODUCTION

Computer simulation is an experimental method which is often used to study the behavior of various kinds of service systems. The process of modeling such systems is rather complicated task with high potential for errors, especially when the modeling is carried out by scientists holding their expertise in other fields than computer science. To reflect real system behavior in the modeled system requires experience even beyond the field of computer science. Defining the complex methodology for service systems simulation and modeling and creating a CASE tool supporting model design and maintenance based on the methodology is the effort to make this process easier. This paper briefly describes the ABAsim architecture of simulation models. This architecture provides flexible base for creation of simulation models of large service systems. The methodology of modeling such systems is the main focus of this article. It also gives an overview of the

supporting tool for this architecture called the ABAbuilder.

## THE ABAsim ARCHITECTURE

Architecture *ABAsim* was developed mainly for simulations of large service systems. The simulation models of simple real systems could be composed of only one agent; however the simulations of complex service systems are obviously connected with *multi-agent approach* using the agents within some organizational structures. Let us remark that the philosophy of ABAsim architecture was also partly inspired by the paradigm of reactive agents, which is based on the society of rather reactive than proactive agents (Nwana, H. 1996). The intelligence of that society *emerges* when observing the whole community and not the separate (relatively little intelligent) members.

Multi-agent hierarchical system can be demonstrated on the example shown on Figure 1. On the top of the hierarchy is agent *A-0* (agent of simulation, also called boss). This agent utilizes the set of subordinated specialized agents in order to fulfill defined tasks connected with surroundings (agent *A-1*) or with service system itself (agent *A-2*). Agent *A-1* is responsible for the management of whole surroundings of the system (arrival of customers from surroundings, departure of customers from the system, etc.). The agent divides the management to three partial managements of different transportation modes (road, railway and water). We say that agent *A-1 delimits* the surroundings management to three peer agents *A-3*, *A-4* and *A-5*, which inform their superior agent about the important facts connected with their mode of transportation. Agent *A-1* can also send important multi-modal information to its *subordinates*. It is expected that all three mentioned modal agents carry out the same kinds of management operations.

The agent responsible for the service system itself is *delegating* some portion of its competences to hierarchically lower ranked colleagues. It is obvious from the mentioned figure that agent *A-2* utilizes its subordinate agents *A-6* to *A-9* to manage all required tasks.

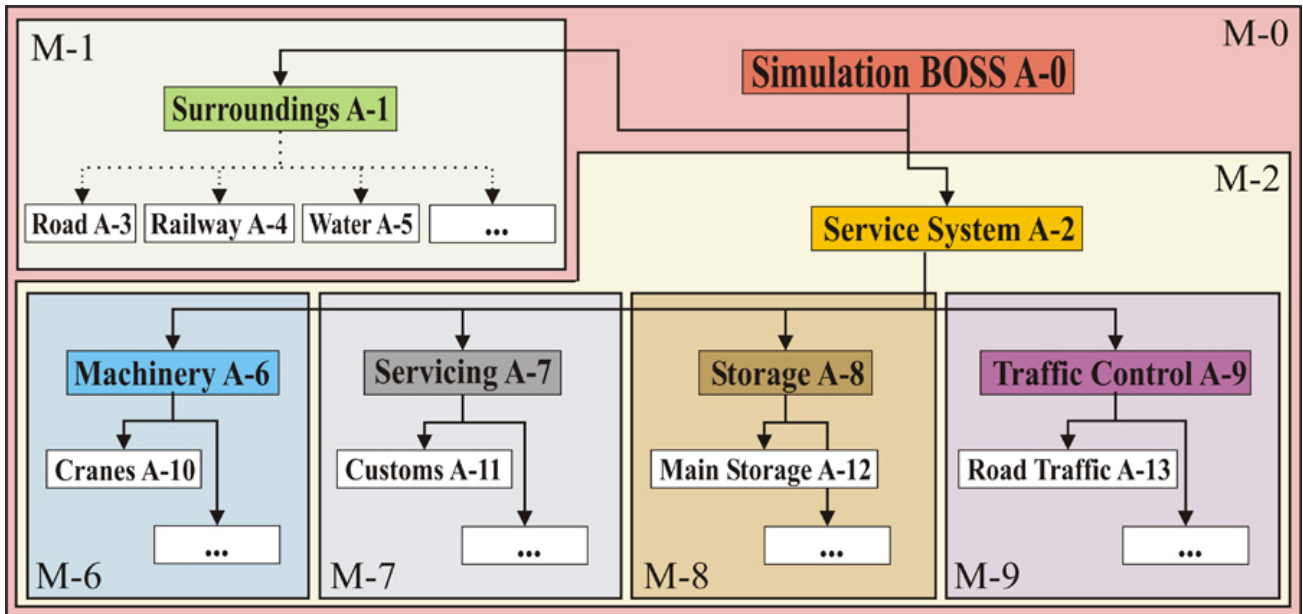


Figure 1: Agent Hierarchy in a Simulation Model of a Service System

Another point of view is that agent  $A-0$  is a representative (boss) of the whole model  $M-0$  (simulation model of service system). That fact is denoted as:  $A-i = boss(M-i)$  and  $M-i = model(A-i)$ . The models  $M-1$  and  $M-2$  represent sub-models of model  $M-0$ . The models on the lowest hierarchical level (the leaves) are always realized as one-agent models. Let us finally denote the structure (down to the third level) of model  $M-0$  in the form of algebraic expression:  $(M-1[M-3, M-4, M-5], M-2(M-6, M-7, M-8, M-9))$ , where the pairs of brackets encapsulate the models, which were created as an act of delimitation and the pairs of parentheses encapsulate the models with delegated competences.

### Agent decomposition

Each of agents can be decomposed to four following groups of *internal components*:

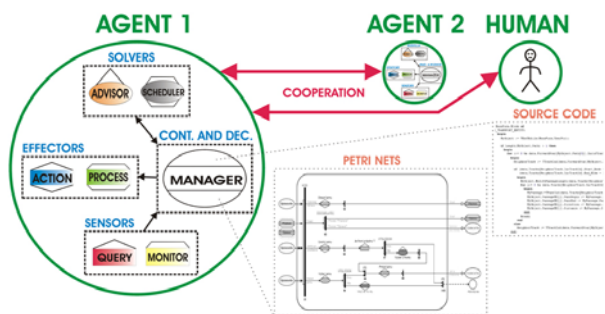


Figure 2: Agent Decomposition

- The first *control* and *decision making* component, called *manager*, is responsible for making decisions and inter-agent communication. In addition, the manager represents the central agent component because it initiates the work of other internal components and also can communicate with all of them.
- The group of *sensors* is specialized to information mining from the state space. That group is composed of two kinds of components - the *query* delivers the required kinds of information instantly and the *monitor* makes scanning of state space during some time interval and continuously brings important pieces of information to the manager.
- The next group of *solvers* provides problem solutions for the manager, which can accept them or asked for alternative ones. The *advisor* is passive component that is able by return to react only to the manager's request for delivery of problem solution proposal. The typical forms of an advisor can be represented e.g. by optimization algorithms, neural networks or human operator. On the other hand the *scheduler* (focused on the restricted scope of problems) works for manager continuously on the base of either a priori created rosters/schedules (e.g. connected with allocations of resources) or making its own dynamic forecast for defined time interval.
- The last component group includes *effectors* (actuators), which make changes of system status after getting corresponding instructions from the manager. The other kinds of agent components are not allowed to make the mentioned changes. The *action*-component makes instant state changes (e.g.

switch a traffic light over, close the train doors), while the *process* (e.g. the crane movement) makes them continuously till its tasks are finished.

The effectors, sensors and solvers are briefly called manager's *assistants* and can be further distinguished as:

- *Continual assistants*, activities of which take some interval of simulation time (processes, monitors and schedulers).
- *Instant assistants* (actions, queries and advisors), which are active only within one instant of simulation time.

### Inter-agent communication

Let us present basic principles of solving the above-mentioned problem within ABASim architecture profiting by hierarchical organization of sub-models. The model  $M = \{M-0, M-1, M-2, \dots, M-s\}$  represents a set of sub-models reflecting a set of agents  $A = \{A-0, A-1, A-2, \dots, A-s\}$ . For all  $I = 1 \dots S$  we denote:

- $I_{Ag}(A-i)$  – a set of messages directly addressed to an agent  $A-i$ , which is able to respond to them (a set of *direct-messages* of an agent  $A-i$ ),
- $I_{Sub}(A-i)$  – a set of direct messages of all agents, which are subordinated to  $A-i$  (a set of *mediated-messages* by an agent  $A-i$ ),
- $I_{Mod}(A-i) = I_{Ag}(A-i) \cup I_{Sub}(A-i)$  – a set of messages, which can be elaborated by a *model(A-i)* regardless of its structure (a set of *model-messages* in respect of *model(A-i)*).

Each message kind can be utilized in three following variants:

- Message of kind  $mess(A-i, A-j, agent)$  is sent by an agent  $A-i$  to an agent  $A-j$ , which accepts it only if  $mess \in I_{Ag}(A-j)$  – it is a standard *addressed message*.
- Another message type  $mess(A-i, A-j, model)$  is dispatched by a sender in case it is not known a concrete target addressee/agent (or due to flexibility it is not desired to be determined). However, it is evident that *model(A-j)* is qualified for its elaboration. That message is treatable if  $mess \in I_{Mod}(A-j)$  and is called *partially addressed message*.
- Finally a message kind  $mess(A-i, *, *)$  represents a *non-addressed message*, which is elaborated in case that  $mess \in I_{Mod}(A-0)$ .

### MODELING SERVICE SYSTEMS

A *service system* is understood as a system focused on elaboration of *orders* (attendance of *customers*) and

execution of *services* related to them. The mentioned services can further initiate another set of orders. The service systems include a wide class of various systems e.g. factories, any transportation and logistic nodes/junctions, repair shops etc. The natural and technical systems do not belong to that domain.

The service system structure can be considered (from the viewpoint of order elaboration) as strictly *hierarchical*. The order (the customer) entering the system (e.g. produce a car) calls recursive sequence of suborders according to the rules of competence redistribution. All system elements (subsystems) work synergically, unlike the majority of natural systems, with the common goal to elaborate the order. Thus, the architecture is not determined to work with the processes of the following kinds: evolution, competition and parasitism. The entities of a service system (orders/customers and resources) can be divided into specialized classes with the same behavioral rules for included entities. It means that the responsibility for the behavior of entities is taken by their superior subjects (agents) and hence there is no reason to consider entities as agents.

The service systems represent usually large-scaled systems. It is mostly necessary to transfer service resources to the customer (or vice versa), in order to realize the service activity. Hence, there are typical frequent complex transposition processes within those systems.

### System Entities

The main function of a service system is to serve customers which enter and leave the system. To fulfill this task, resources of various kinds are necessary. The customers and all the resources are represented by system entities, which are controlled by system agents. Each entity is controlled by one agent at a time. Agents can relegate the control over specific entity to another agent. Entities are not capable of managing their own acts. There are several dimensions to classify system entities:

Firstly entities may be classified by their mobility, i.e. by their ability to be moved around some network by their controlling agent:

- *Static entities* – the change in their position in space and time is not possible (e.g. building, gate, parts of the infrastructure, etc.)
- *Dynamic entities* – the position change can be achieved by their controlling agent. They are further subdivided into:
  - *Mobile entities* – an agent controls their movement without support of another entity (e.g. car, plane, vessel, etc.)

- *Stationary entities* – their movement has to be supported by another entity (e.g. container, chassis, etc.)

Second classification is according to their appearance in the system:

- *Endogenous entities* – they originate inside the system (e.g. train sets, etc.)
- *Exogenous entities* – they enter and leave the system, originate outside of the system (e.g. car, plane, etc.)

Entities in the system are stored in so called *the entity layer* which is part of the ABAsim architecture. Each entity has *access rights* defined, so that only authorized agents are allowed to change their status. This approach concentrates all of the system entities at a single location and makes them easily accessible to all of the system agents.

### The Model Design

To design the system architecture, the system has to be divided according to its main tasks. Therefore, a different kind of agent for each one of the main tasks to be done has been modeled. Unlike in (Rebollo et al. 2000) where each system entity is controlled by a separate agent, in our system each agent controls all the entities belonging to the same category. For example in a system with multiple cranes, such as gantry cranes, the operation of these cranes would be controlled by one agent called *Agent GantryCranes*.

The ABAsim architecture, which is designed for modeling large service systems, allows us to divide the problem into subproblems. Each subproblem can be solved by a specific agent, which uses its subordinate agents to accomplish desired tasks. This approach enhances flexibility of the whole system.

According to our understanding of service system, we propose the hierarchical architecture of agents in a simulation model of a service system shown on Figure 1. The whole model *M-0* is represented by its boss agent *Simulation BOSS A-0*. This agent has two subordinated agents: *Surroundings A-1* and *Service System A-2*.

Submodel *M-1* represented by agent *A-1* is used to model the behavior of the surroundings related to the modeled service system. Agent *A-1* can have various number of underling agents. The number of these agents depends on the service system characteristics. For example, when modeling a container terminal, agents *Water* and *Road* will be used to supply “customers” for the service system. Agent *Railway* would be alternatively used in case railway directly interacts with the system. Agents in the *M-1* submodel are responsible for management of *exogenous entities* before entering and after leaving the system.

Agent *Service System A-2* represents the submodel *M-2* which is the model of the service system itself. It’s delegating its competences to hierarchically lower ranked colleagues: *Machinery A-6*, *Servicing A-7*, *Storage A-8* and *Traffic Control A-9*. Both *endogenous* and *exogenous* entities are being managed by *M-2*. The main features of agents in *M-2* are specified in the following section.

*Machinery A-6* is the head of the subsystem *M-6* and it utilizes all of the agents managing the operation of *endogenous* entities (e.g. cranes, reach stackers, straddle hoists, etc.) Each *M-6* member controls entities of the same category and it’s capable of making decisions about the usage of each entity it manages. Decision making of these agents is supported by their *advisors*, which are commonly implemented as some kind of optimization algorithm. The way the logic is implemented influences the function of all controlled entities.

Agent *Servicing A-7* is responsible for submodel *M-7* which has been divided according to the different services. Each service type has assigned one agent which is responsible to fulfill desired tasks. Agents in this group are responsible for serving “customers” at all checkpoints in the system.

*M-8* is a subsystem with its leading agent *Storage A-8*. This part of the model is responsible for managing system storage resources. Agent *A-8* has a function of the director of all the storage areas in the system. Subordinated agents provide rules for storing entities in different storage sections of the system.

Finally the traffic control subsystem *M-9* is controlling all the traffic in the entire system. The process of transportation of an entity from starting point to its destination point is guided by corresponding agent. For example, when truck is moving from the entrance gate to the control bridge, this process is controlled by the *Road Traffic A-13* agent. This agent is responsible for safe journey, avoiding all possible collisions with other entities and positioning in the queue of waiting vehicles.

### Modularity and Model Re-Use

Structure of a simulation model as well as of individual agents represents such features, which highly support a flexibility of simulation model within ABAsim architecture. In spite of that fact the message-oriented communication mechanism does not enable such level of flexibility, which would allow changing a sub-model structure without a need of additional changes within other model parts.

### THE ABAbuilder TOOL

Using traditional methods of model design introduces high risk of making errors; therefore the ABAbuilder tool which was created upon the existing architecture

brings a great improvement in this process. ABAbuilder is a CASE tool based on the ABAsim architecture and it supports several aspects of model development including system analysis, design of communication, and use of methodologies, prototyping, model maintenance and re-engineering (Figure 3).

One of the greatest contributions of the architecture is the flexibility of model configuration. ABAbuilder is capable of creating large palette of alternative internal components, which can be used to construct the simulation scenario. The experimenter is allowed to change executive characteristics of each of the system agents by using different components with pre-programmed behavior and decision making algorithms. Even less experienced programmers can design their own experiments. This is due to similarity of modeled system with the real world.

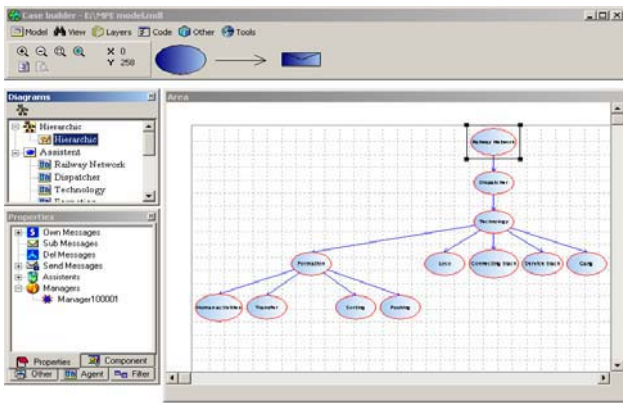


Figure 3: Screen Capture of the ABAbuilder Tool

A simulation model can be described by the hierarchy of system agents. From the ABAbuilder point of view, not only agents, but also their internal components are considered as model building base. There are several phases each model has to undergo, before experiments can be carried out:

1. design of the hierarchy and communication of system agents
2. implementation of the simulation model
3. verification of the model
4. parameters setup
5. source code generation

To be able to accomplish these particular phases the CASE tool had to be divided into four interacting modules: The *visual environment* displays the simulation model from different perspectives. Its main purpose is to display the agent hierarchy and internal components of each agent. It allows users add new

agents to the hierarchy, edit their internal components and define the message flow inside the model. Visual Petri nets editor is used to define the reaction of agent's manger to received messages. Users are allowed to look at all aspects of the hierarchy at once, or they can apply various sets of filters.

The *error localization* module works continuously and it checks the model each time a change has been applied. It provides user with two kinds of messages: *error messages* and *warnings*. Errors messages appear when user tries to do something which is not allowed by the methodology of creating models based on the ABAsim architecture. Warnings inform users of possible unwanted problem solutions.

When the decision is made by the user that the model is ready for deployment the *source code generation* module comes into use. This module is capable of creating the framework of the simulation model. It generates source files in chosen destination language (only Object Pascal is currently supported). These files contain all agent classes, their internal components, communication and the hierarchy. User then finalizes the simulation model in external development environment (e.g. Borland Delphi). Libraries containing the simulation kernel, which allow compiling a standalone application, are provided.

To be able to reconstruct the model from source files to its graphical representation a *source code parser* module is necessary. This module analyzes given files, which were previously generated and it provides data for the visual environment.

ABAbuilder tool enables to construct new models and modify existing ones with the help of prefabricated model parts, which are selected from bases of agent components (assistants, managers), agents, sub-models and their various configurations. It makes the whole process of modeling much easier to accomplish and more understandable for more as well as less experienced user.

## CONCLUSIONS

In this paper the agent-oriented architecture ABAsim was presented and its openness for creating flexible models of large-scale service systems. The general model of a service system presented in this paper is being customized to build more models of real world systems. Once these models are configured they are being verified and validated and experiments are being carried out. ABAbuilder tool makes process of model customization much easier for developers. One of the goals to achieve in the future is to try to make modeled systems cooperate by the methods of distributed simulation which are also supported by this architecture.

## REFERENCES

- Gulyas, L.; Koszik, T. 1999. "Model Design Interface – A case tool for the Multi-Agent Modeling Language", In: *Journal of Artificial Societies and Social Simulation* vol. 2, no. 3
- Jennings, N. R. 2001. "An agent-based approach for building complex software systems", *Communications of the ACM*, Vol. 44, No.4, pp.35-41
- Kavička, A.; Adamko, N.; Klima, V. 2000. "Simulation support for railway junctions infrastructure and processes economization", In: *Proceedings of MI 2000 conference*, Zilina, Slovak republic, 2000
- Klima, V.; Kavička, A. 2000. "Simulation support for railway infrastructure design and planning processes", In: *Proceedings of COMPRAIL 2000 conference*, Bologna, Italy, 2000, pp.447-456
- Kovacs, G.; Kopacsi, S.; Nacsa, J.; Haidegger, B.; Groumpos, P. 1999. "Application of software reuse and object-oriented methodologies for the modelling and control of manufacturing systems", In: *Computers in Industry 39 1999*, pp. 177-189
- Nwana, H. 1996. "Software Agents: An Overview", In: *Knowledge Engineering Review*, Vol. 11, No 3, pp.1-40
- Rebollo, M.; Julian, V.; Carrascosa, C.; Botti, V. 2000. "A Multi-Agent System for the Automation of a Port Container Terminal".
- Wooldridge, M.; Jennings, N. 1995. "Intelligent Agents: Theory and Practice", *The Knowledge Engineering Review 10 (2)*, pp. 115-152

## AUTHOR BIOGRAPHIES

**MICHAL LEKÝR** - got his Master degree at the University of Zilina in 2003, where he is now doing his PhD. study.

**NORBERT ADAMKO** - graduated at the University of Zilina in 1997, where he in the same year started PhD. study. Currently he is working as an assistant professor, teaching courses on Modeling and Simulation, Data Structures and Discrete Simulation.

## ACKNOWLEDGEMENT

This work has been supported by the Slovak grant foundation under grant No. 1/1049/04 "Agent Oriented Architecture of Service Systems Simulation Models"