RAPID PROTOTYPING AND VERIFICATION OF SIMULATING SYSTEMS APPLYING ABASIM ARCHITECTURE

Emil Řezanina[(a)], Antonín Kavička[(b)]

[(a),(b)]Faculty of Electrical Engineering and Informatics, University of Pardubice, Czech Republic

[(a)]emil.rezanina@upce.cz [(b)]Antonin.kavicka@upce.cz

ABSTRACT

When designing and developing computer simulation models it is possible to employ different architectures of simulate systems. Quite often Agent-based architectures are used. They provide a number of advantages to the needs of construction, modification and expansions of relevant models. In the field of computer simulations a variety of agent-based architectures are applied. They are available in a number of simulation tools, frameworks and programming languages.

Nowadays, when the emphasis is put on the reduction of time for software systems development, it is often required to have a graphic support for rapid prototyping of simulation models available, using the appropriate architecture.

The paper is aimed at presenting the development of a software framework (working name ABAframe) supporting the development of simulators using one particular type of Agent-based architecture of simulation models called ABAsim. Said architecture is particularly suitable for creating simulators of service, logistics and transport systems.

The main purpose of the ABAframe framework within an integrated development environment is to enable both rapid and partially automated graphical prototyping of simulation models based on autonomous agents and also support the implementation of simulation experiments including evaluation of their results. An important functionality the stated framework features is providing the verification of communication interfaces and communication of inter-agents within a built prototype of simulator.

Keywords:
Agent-based architecture, simulation model, rapid prototyping, prototype verification

1. INTRODUCTION

Currently, a number of simulation tools, frameworks, and programming languages (Zheng 1992) are available for the needs of production of (monolithic) simulation models, where different architectures, or rather approaches, are applied. The following architectures could be used as an example:

- events or process-oriented architecture,
- architecture based on continuous activities,
- agent-based architecture,
- combined / hybrid architecture.

Within the existing tools a support is usually available to facilitate the building of simulation models.

In this paper an attention is paid to one of the used agent-based simulation model architectures called *ABAsim* (Kavička, Klima and Adamko 2005). Using this architecture is particularly suitable for complex simulating systems reflecting traffic within the service, logistics and transport systems.

To simplify and accelerate the creation of simulation models the methodology of rapid (graphically supported) prototyping can be used. This methodology facilitates the implementation of quick configuration of a simulation model prototype, in this case composed of agents. The advantage of rapid prototyping is the possibility to bring the simulation model prototype under formal correctness control.

In order to increase the attractiveness of *ABAsim* architecture it was requested by many users to build a support for a quick and partially automated prototyping of complex simulation models, including the possibility to examine the correctness (verification) of the models. In this connection, the *ABAframe* framework and the *ABAframeIDE* tool, which is constructed above the framework, are being developed.

The concept of *ABAframe* framework, including its functionalities, is described below. Attention is paid mainly to the process of rapid prototyping of simulating systems and to the implementation of relevant prototype verification. Also a case study is presented where the individual parts of creation and verification of a prototype are demonstrated.

2. ABASIM ARCHITECTURE

Agent-based *ABAsim* architecture allows the creation of simulation models using autonomous units, the so-called agents. These agents constitute an encapsulated computer system that is able to flexibly and autonomously operate in a specific environment to achieve the stated objective (Jennings and Wooldridge 200). To "flexibly operate" means reactive, pro-active and social behaviour of the agent. The following figure shows the agent's behaviour when achieving the goal.

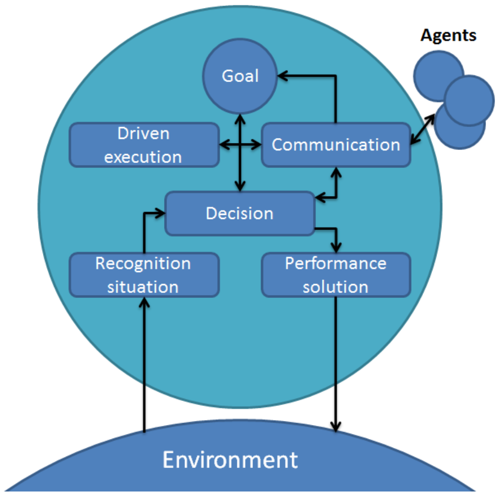


Figure 1: The structure of the agent's behaviour (Adamko 2013)

In the *ABAsim* architecture the simulation model consists of cooperating agents organized in a hierarchical structure. The architecture offers two types of agents - control and dynamic agents. Control agents are static entities which exist throughout the simulation run. These agents represent a high-level system entities that cooperate together in order to achieve system goals. The second type of agents is represented by dynamic, autonomous and proactive entities. They are always current under the management of one control agent, but may be transferred under the management of another control agent. Dynamic agents mutually interact with the environment (other dynamic agents, entities) and communicate with the control agent. Dynamic agents are created and cancelled during the simulation run and receive local goals by managing agents (Adamko 2013, Kormanova, Varga and Adamko 2014).

2.1. Composition of Agent

The *ABAsim* architecture supports decomposition of individual autonomous agents into specialized components that focus on performing certain activities. The main component of each agent is a *manager*. This component focuses on the control and communication activities of an agent. The remaining internal components of agents are called assistants. They are used to support the *manager*. Assistants could be classified as either *prompt* or *continuous*. The difference between them is that prompt assistant performs the action immediately, whereas continuous assistant needs some time. Assistants can be divided into several groups according to the type of activity:

a) Group of *sensors* has the task of monitoring the area of an agent. This group includes a *query* component and a *monitor* component. The query component is a *prompt* assistant that examines the surroundings at the request of the *manager* and immediately inform it of the outcome. The *monitor* component is a continuous assistant that unlike *query* component monitors the area for some time. If there is any monitored event at this time (arrival of a customer) it informs the *manager*.

b) Group of *effectors* represents assistants who perform executive actions in the surroundings of an agent. For immediate performance a prompt assistant *action* is employed and for continuous operation a continuous assistant *monitor* is applied.

c) Group of *advisors* is being used by the *manager* as a decision support. This group includes prompt assistant *adviser* and continual assistant *planner*.

2.2. Communication

Basic communication mechanism in *ABAsim* architecture is a mechanism of messaging sending. This mechanism is used both for inter-agent communication and internal communication. Messages can be processed in several different ways. Messages are saved in a mailbox of an agent or in a central mailbox of a simulation model. Another possibility is to process the message immediately. Each message contains a timestamp that represents the time when the message should be processed.

For better organization of communication and for simplification of the simulation model design the *ABAsim* architecture defines several types of messages. Message types actually enable messages to better illustrate what kind of message it is and how it should be processed. Communication via message types can be seen in the following Figure.

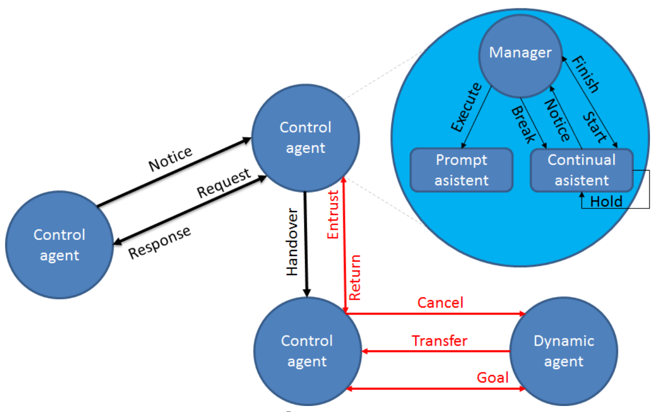


Figure 2: Inter-agent and intra-agent communication (Adamko 2013)

Message types that are used for inter-agent communication are divided into (Fikejz and Řezanina 2013, Kocifaj and Adamko 2014):

– *Notice* – a message used to inform agents about the situation that has arisen. The message only indicates given facts and doesn't wait for a response (e.g. resource was returned).
– *Request / Response* - *Request* message type is used as a request for resource. The request is sent to the addressee (e.g. Deliver mobile source). To the extent possible a response is expected in association with this type of message. Message type *Response* is used as a reply and it is sent back to the original sender.

Messages related to work with dynamic agents are also used for inter-agent communication. It is the possibility of transferring a temporary management (*Entrust / Return*) or permanent management (*Handover*) to a dynamic agent, assignment of goals (*Goal*) and cancelling of goals (*Cancel*) to a dynamic agent. In contrast, the dynamic agent sends to the managing agent that administers it a message that indicates that the goal was achieved (*Done*) or request to move the management of dynamic agent elsewhere (*Transfer*).

Intra-agent communication is open to all its components, but they must follow certain rules for this communication. The *manager* can communicate with all the other components of the agent. Other components do not communicate with each other (Adamko 2013). *Manager* and his assistants communicate through the following types of messages:

- *Start* - This type of message is sent to a continuous assistant. The addressee starts to perform its autonomous activity after receiving this message.
- *Break* - through this type of message the manager affects the autonomous run of continuous assistant. Assistant cease working after receiving this message.
- *Execute* - type of message sent to prompt assistants. Assistant responds with an instant processing of the message and returning results to this type of message.

Continuous assistants are initiated by the *manager* via message Start mentioned above. During their existence, continuous assistants can send the following types of messages to the *manager*:

- *Finish* - this type of message is sent when a continuous assistant completes its work.
- *Notice* - a message used by a continuous assistant to inform the *manager* about important matters that require reaction.
- *Hold* - is the only type of message that can have higher timestamp of the message than the current local virtual time of the agent. Message type Hold is used in order to shift simulation time.

More specifications of inter-agent and intra-agent communication can be found in resource (Adamko 2013).

3. RAPID PROTOTYPING

Rapid prototyping is one of the ways to quickly create simulation models of complex systems. The result of prototyping is a simulation model with limited functionality, so-called prototype. Prototypes of simulation models based on Agent-based architectures can be composed of empty agents and communication links among them. In *ABAsim* architecture so-called pre-defined agents are used instead of empty agents. These agents are perceived as agents that contain external interface for receiving messages from other agents as well as a set of internal components of an agent. Mentioned components include internal communication interface, but without internal logic of incoming messages processing.

Tools that provide a graphical user interface are very often used for prototyping. These tools lead to a significant speed-up of the creation of simulation models and to convey its visualization. Visualization brings greater knowledge of the proposed model. This is very useful for complex solutions.

Another advantage of prototyping is an early control of the model. Formal correctness does not relate directly to a simulation model but to its prototype.

4. VERIFICATION

Verifying the correctness of the simulation model consists of verification and validation of the model. Model verification is a process of checking the formal correctness of the model. In contrast, model validation is a process to verify that the outputs of a simulation model correspond to the realistic characteristics of the modelled system.

When rapid prototyping simulation model in *ABAsim* architecture we deal only with the verification of the model prototype. Validation cannot be applied in this case because components of pre-defined agents do not contain internal logic and therefore the simulator is not yet built.

Verification of the prototype of agent-based simulation model determines whether all communication interface settings are correct. These interfaces can be divided in two groups, namely:

- The first group consists of the *communication interface of agents*. Every agent knows what messages it receives from other agents and what messages it sends.
- The second group is composed of *communication interfaces of internal components* of individual agents. When processing incoming messages from another agent the *manager* communicates with his

assistants. The communication interface of each assistant of the manager must be correctly configured so that the *manager* is able to communicate with the assistant.

The verification of the prototype also focuses on inter-agent communication and the involvement of agents in the simulation model.

The main advantage of prototype verification, as mentioned, is the possibility of early correctness checks. This allows to eliminate errors that might have arised at the beginning of the draft before complete implementation of the simulation model's internal logic.

5. ABAFRAME FRAMEWORK

ABAframe Framework uses a special simulation core, which allows the simulation above the simulation models created in *ABAsim* architecture. Simulation core works with discrete events that represent moments of message processing in the *ABAsim* architecture. Simulation core performs simulation above the simulation model in two phases. Synchronization algorithm of the simulation core is based on the principles of the resource (Adamko 2013).

ABAframe Framework enables the user to create a simulation model prototype using graphical user interface (or more precisely integrated graphical development environment). The prototype consists of individual pre-prepared (partially formalized) agents and specifications of inter-agent communication.

5.1. Creating a prototype

For creating a prototype of agent's simulating system within the environment provided by the *ABAframe* Framework a tool called *ABAframeIDE* is used. This tool divides the creation of a prototype into several phases:

(1) A draft of a model, which consists of empty agents and defined communication links among agents.

(2) Specifications of individual internal components an agent will consist of.

(3) Setting the communication interfaces of internal components of individual agents.

(4) Implementation of internal model for incoming messages processing within the component manager.

(5) Generation of frames.

In the phase of a *draft of the model at the level of agents* the user determines what agents, agent's models and inter-agent's communication links the model will consist of. Inter-agents communication link represents the sending of one message. Any agent can be a sender and the recipient of the message can be either a specific agent or also an agent model. The message must also specify the type (e.g. *Notice*), code (e.g. *Incoming customer*) and parameters (e.g. *Customer*) of the message. For message type R*esponse* it is necessary to set to which message type R*equest* it replies.

The internal composition of every created agent must be set. Setting the composition of an agent comes under the *agent's internal components specification* phase. For each agent a tab appears that allows insertion and removal of manager assistants (i.e. *monitor, action*).

The next phase is the *setting of communication interfaces of internal components*. This phase is used to set what messages assistants receive and how they respond. In addition, for continuous assistants it is also necessary to define what messages they will send during their operations (to themselves or the *manager*).

The penultimate phase of developing the prototype represents an *implementation of an internal model of incoming messages processing within a manager*. At this stage, there are several ways how to implement the internal model. An imperative approach (using the resource code of the programming language) or declarative approach (based on the graphical specification of the relevant - coloured - Petri nets) may be used. If the first option is chosen, the implementation of an internal model is omitted for the time being. The programming of this model must be finished after the generation of the prototype.

The last phase is the *generation of frames*. In this phase, first of all a formal inspection of correctness of individual parts' settings from previous stages takes place. If this goes through successfully, an export of a simulation model prototype to a resource code of the relevant higher programming language (in which the model will be further developed) ensues.

5.2. Verification

It is possible to conduct a formal control of correctness within the *ABAframe* framework. It is divided in two levels.

The first level of control is performed prior to the generation of frames in the *ABAframeIDE* tool. The main objectives of this level are to check the correctness of the setting of agents' identifiers and their internal components. Furthermore, it is examined whether some communication links between two agents do not share the same type and code.

The second level of control is performed after the export of a prototype. This level is composed of the following verification:

(i) verification of communication interfaces of agents,
(ii) verification of communication interfaces of agents' internal components,
(iii) verification of inter-agents communication and
(iv) verification of the involvement of agents in a complex simulation model.

Verification of communication interfaces of agents controls which messages agents receive and which messages they send. Each message must pass the scrutiny of their requirements. These requirements include e.g. type code, sender, recipient, etc.

Verification of communication interfaces of agents' internal components controls each interface of manager's assistants. This control should check whether the interface of each manager's assistant is properly applied in the context of the internal logic of incoming messages processing. This verification does not occur if an internal model of incoming messages processing of the *manager* has not yet been specified.

Verification of inter-agents communication checks whether each message sent from agents has its existing recipient who has a relevant routine utilities of this message. Lists of agents' incoming and outgoing messages are used for this purpose. During the control it is checked whether a message from the list of agent's outgoing messages has the same message notation in the list of recipient's incoming message. Messages are the same if they have the same type, code and parameter list.

Verification of the involvement of agents in a complex simulation model aims to reveal the single alone agents. These are agents that do not communicate with any other agent, and thus are not utilized within the model.

If the model prototype fails to pass the verification successfully the framework informs the user about the formal faults of the model so that they could be removed subsequently.

The second level of control do not have to be used for generated simulation model prototype only. It can also be used for specific simulation model.

6. DEMOSTRATION APPLICATION

A procedure of creation of a simplified simulation model of queuing is illustrated here to demonstrate the work with the *ABAframe* framework and *ABAframeIDE* tools.

6.1. Description of a model of queuing

Customers enter the model from its surroundings, attend two types of service and leave the system after their completion. While the first type of service is linked to a stable resource to which the customer must transfer, the second resource is mobile, and thus transfers itself to the customer in order to perform the second service. If the customer comes to a service that does not have a free resource, he will join the appropriate queue waiting for this resource and stay here until taking his turn (Kavička, Klima and Adamko 2005).

6.2. Creating a queuing model prototype

To create a queuing model prototype the *ABAframeIDE* tool is used. The procedure of generation of a model using this tool will be shown in this section.

In the first stage a simulation model was modelled. It includes *surroundings*, *service manager* and *resource manager* agents. Each of these agents has a specific role. In addition, it was necessary to set up communication between these agents.

Surroundings agent is responsible for the contact between the model and its environs (arrivals and departures of customers). This agent cooperates with *service manager agent*, which is in charge of organizing the service. Arriving customer will undergo two types of services here and leave the model after completing these services. To work with resources the *service manager agent* communicates with the *resource manager agent*. This agent is responsible for allocation and relocation of resources to the service. In addition it deals with queues of applicants for resources.

The design of individual inter-agents communication links consists of two steps. The first step was setting the resource and destination of the message. The second step was to set the type, code and list of parameters of the link. List of parameters contains the names of parameters the message will transmit. E.g. communications link "Incoming customer", which leads from the *surroundings agent* to the *service manager agent*, carries the Customer parameter in this list.

The result of this phase is shown in the following figure.

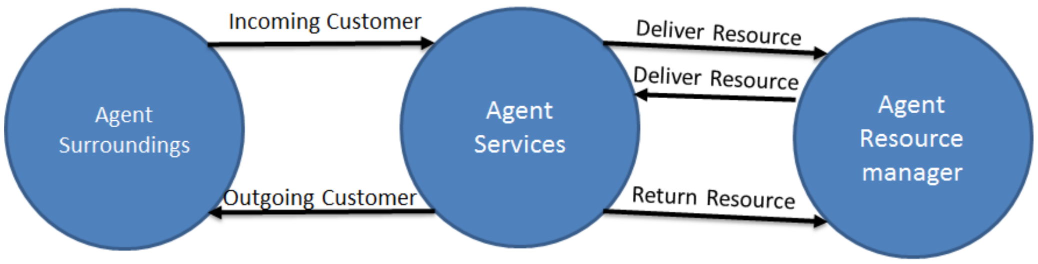
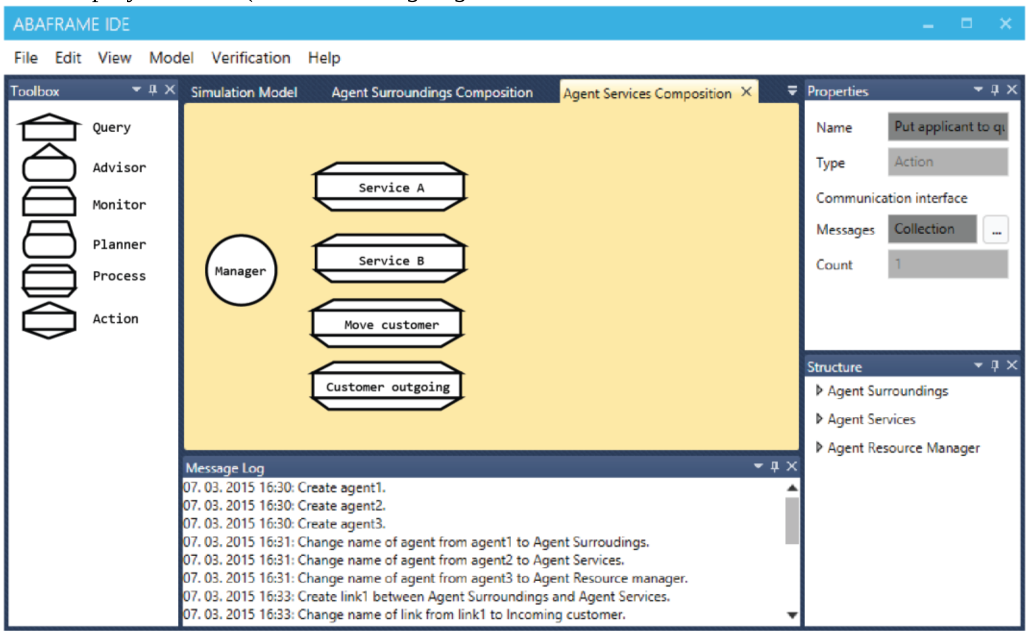


Figure 2: Design of simulation model of queuing.

The second phase dealt with the composition of individual agents. In this step components that the manager will use for his work were selected for each agent.

Surroundings agent contains only a component process *Entering Customers*. This component aims to generate periodic messages announcing the arrival of a new customer into the system.

The manager of *Service agent* operates with four processes. The first process is the execution of service A (*Service A* process). The second process is responsible for the transfer of a customer from service A to service B (*Move customer* process). This process is followed by the process of performing service B (*Service B* process). After completion of service B the customer leaves the service and heads to the exit of the system. The next process is employed here (*Customer outgoing* process). The compositions of this agent in *ABAframeIDE* tool is shown in Figure 3.



Name: Put applicant to qu
Type: Action
Communication interface
Messages: Collection ...
Count: 1

```
07. 03. 2015 16:30: Create agent1.
07. 03. 2015 16:30: Create agent2.
07. 03. 2015 16:30: Create agent3.
07. 03. 2015 16:31: Change name of agent from agent1 to Agent Surroudings.
07. 03. 2015 16:31: Change name of agent from agent2 to Agent Services.
07. 03. 2015 16:31: Change name of agent from agent3 to Agent Resource manager.
07. 03. 2015 16:33: Create link1 between Agent Surroundings and Agent Services.
07. 03. 2015 16:33: Change name of link from link1 to Incoming customer.
```

Figure 3: The compositions of service agent in *ABAframeIDE*

The last agent, the *resource manager* agent, has two groups of manager's assistants. The first group helps the *manager* with allocation and relocation of service resources. This group consists of the following components:

- action for allocation of resources (*Assign resource*)
- action for recovery of resources (*Return resource*)
- advisor for selection of free resources (*Selection of free resource*)
- request to determine whether it is necessary to relocate the resource (*Need move resource*) and
- process of transferring the resource (*Move resource*).

The second group is in charge of the queue of customers waiting for service. The queue depends on the occupancy of resources. The group includes an action for adding an applicant to the queue (*Put applicant to queue for resource* action) and actions for removing the applicant from the queue (*Remove applicant from queue* action).

Apart from setting the name of every assistant it was also necessary to set a communication interface. The setting of *Put applicant to queue for resource* action assistant will be used as an illustration. This component represents a prompt assistant that should add the sent applicant to the queue for the resource – as expected by the *manager*. The manager does not communicate with this assistant in any other way. Communication interface of the assistant includes only one record consisting of a *notice* message type, "*Put applicant to queue for resource*" message code and a parameter list. The parameter list contains only one parameter called "*Applicant*". There is no need to set a return value in this case.

After setting all the agents assistants it is possible to proceed to the stage of setting the internal logic of incoming message processing. Every manager must respond to an incoming message (inter-agents and intra-agents). In our model, for example, the *Surroundings* agent must respond to three types of messages. The first type of messages comes from the *Service* agent, which sends information about an outgoing customer. The other two types of messages are from continuous component - *Entering customer* process. This assistant informs the *manager* of the arrival of a customer or of the termination of its activities.

For the implementation of internal logic a possibility of programming in the resource code was selected.

ABAframeIDE tool was used for the prototype verification within each particular phase.

The completion of each phase and successful verification was followed by the generation of the simulation model frames.

6.3. Completion of the simulation model

The result from the previous chapter can be added to the project C # programming language. It is sufficient to then add a reference to *ABAframe* framework into the project, which will allow further work with this simulation model, whether it concerns verification or launching of the simulation model.

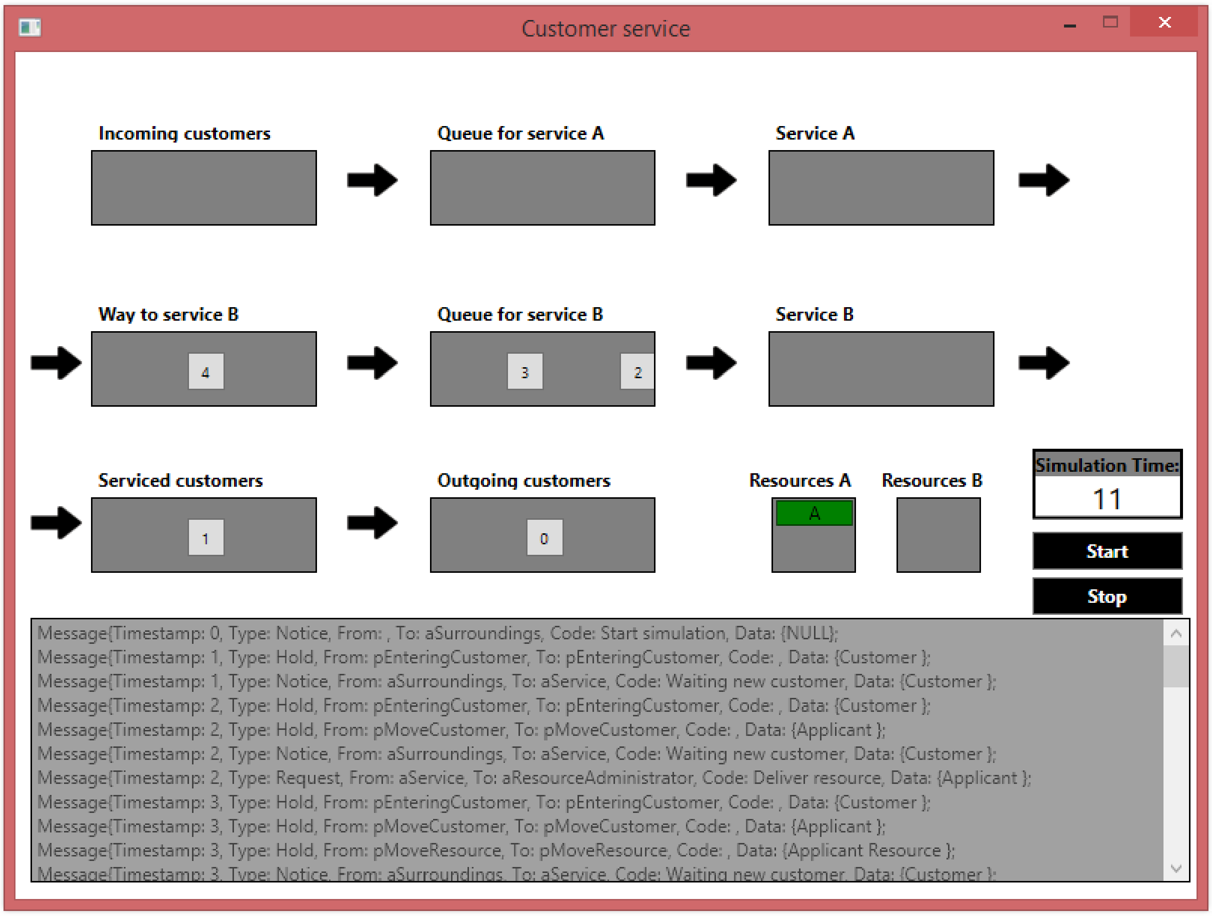
The generated simulation model is not yet complete. The creation of an internal model of message processing by the manager was omitted and the internal logic of generated individual assistants is missing. The missing parts must be programmed by the user.

After completion of the simulation model the *ABAframe* framework verification may be used to control the formal correctness.

Figure 4 shows the possible implementation of a complete simulation of queuing

7. CONSLUSION

This article discusses rapid prototyping and verification of simulation models in the agent oriented *ABAsim* architecture. The *ABAframe* framework was introduced here together with its tool *ABAframeIDE that* allow rapid prototyping and verification of simulation models in this architecture. A demonstration application dealing with queueing was created for better understanding of the design of simulation models using the *ABAframe* framework. The process of creating a prototype of this application is described in this article.



```
Message{Timestamp: 0, Type: Notice, From: , To: aSurroundings, Code: Start simulation, Data: {NULL};
Message{Timestamp: 1, Type: Hold, From: pEnteringCustomer, To: pEnteringCustomer, Code: , Data: {Customer };
Message{Timestamp: 1, Type: Notice, From: aSurroundings, To: aService, Code: Waiting new customer, Data: {Customer };
Message{Timestamp: 2, Type: Hold, From: pEnteringCustomer, To: pEnteringCustomer, Code: , Data: {Customer };
Message{Timestamp: 2, Type: Hold, From: pMoveCustomer, To: pMoveCustomer, Code: , Data: {Applicant };
Message{Timestamp: 2, Type: Notice, From: aSurroundings, To: aService, Code: Waiting new customer, Data: {Customer };
Message{Timestamp: 2, Type: Request, From: aService, To: aResourceAdministrator, Code: Deliver resource, Data: {Applicant };
Message{Timestamp: 3, Type: Hold, From: pEnteringCustomer, To: pEnteringCustomer, Code: , Data: {Customer };
Message{Timestamp: 3, Type: Hold, From: pMoveCustomer, To: pMoveCustomer, Code: , Data: {Applicant };
Message{Timestamp: 3, Type: Hold, From: pMoveResource, To: pMoveResource, Code: , Data: {Applicant Resource };
Message{Timestamp: 3, Type: Notice, From: aSurroundings, To: aService, Code: Waiting new customer, Data: {Customer };
```

Figure 4: Resultant simulation of queuing

proto the article.

REFERENCES

Zheng, Hong, et al. A Primer for Agent-Based Simulation and Modeling in Transportation Applications. No. FHWA-HRT-13-054. 2013

KAVIČKA, A., KLIMA, V., ADAMKO, N. Agent-based simulation of transportation nodes, EDIS, University of Žilina, 2005 (in Slovak), ISBN 80-8070-477-5.

JENNINGS, N. R., WOOLDRIDGE, M. Agent-Oriented Software Engineering. Artificial Intelligence. 2000, roč. 117, s. 277–296.

ADAMKO, N. Agentovo orientovaná simulácia zložitých obslužných systémov. Habilitation thesis, Žilina, Faculty of management science and informatics, University of Žilina, 2013.

KORMANOVA, A., VARGA, M., ADAMKO, N. Hybrid model for pedestrian movement simulation. In: The 10th International Conference on Digital Technologies 2014 [online]. 2014 [cit. 2015-06-09]. DOI: 10.1109/dt.2014.6868707.

KOCIFAJ, M., ADAMKO, N. Modelling of container terminals using two-layer agent architecture. In: 2014 IEEE 12th International Symposium on Applied Machine Intelligence and Informatics (SAMI) [online]. 2014 [cit. 2015-06-09]. DOI: 10.1109/sami.2014.6822416.

FIKEJZ, J. - ŘEZANINA, E. Simulation of localization rolling stock within the railway network model utilizing agent-based simulation. In The European Simulation and Modeling Conference 2013. Ostend: EUROSIS-ETI, 2013. s. 290-296. ISBN 978-90-77381-79-3.