EXTENSIVE TRANSPORTATION SERVICE SYSTEMS AND THEIR FLEXIBLE SIMULATION MODELING

Michal Lekýr[1], Valent Klima[1]

[1] University of Žilina, Department of Transportation Networks,
Faculty of Management Science and Informatics, Slovak Republic

Michal.Lekyr@fri.uniza.sk, Valent.Klima@fri.uniza.sk

Abstract

Largely developing area of computer simulation brings many unresolved issues when it comes to building extensive simulation models of complex service systems. For the purpose of creating the simulation models of such systems it is necessary to choose appropriate tools and approaches, which allows designers to respect the real system structure and create models with highest values of model quality indicators. Such models must be flexible and easy to maintain. The fact is that without existence of adequate supportive environment the work with complex simulation models, regarding their size and need of detailed modeling becomes very hard to accomplish and in some cases even unmanageable. One of the contributions of our work is to push forward imaginary frontiers, when the simulation model is still manageable towards more complex models. Presented solutions enable us to create far more extensive models as we could accomplish in the past. For these purposes we are using our own agent architecture ABAsim and our software tool for building generic simulation models called ABAbuilder. This paper also talks about one generic simulation model called Villon, which is suitable for simulating processes in railway transportation terminals.

Keywords: extensive service systems, conceptual modeling, nonprocedural algorithms, generic simulation models, large scale simulation modeling

Presenting Author's biography

Michal Lekýr graduated at the University of Žilina in 2003, where he in the same year started PhD study and graduated in 2007. For two years he worked at the Boston University, in Brain and vision research lab as a research assistant and scientific programmer. Currently he is working as an assistant professor, teaching and doing research in simulation and computer graphics as well as teaching assembly language programming course.

1 Transportation service systems

The most important and evidently the most resource consuming part of a logistic chain is the transfer process, often called *transportation logistics*. The transfer process is a combination of the actual movement of commodities along the desired route as well as the necessary operations which are being carried upon transported commodities in logistic junctions. These necessary operations (manipulations) are also known as the *service processes* and are being conducted at specialized locations known as *transportation terminals*. At transportation terminals, an individual transported quantum or - considering from the service system viewpoint - a transported entity may originate, may be disposed, and at the same time these entities are the main subject of services provided by these terminals.

Undoubtedly, the most complicated situation from the manipulation perspective and process control occurs in service systems, where the infrastructure is composed mostly of railway tracks, where the mobile service resources are mostly locomotives and where the elements undergoing various services are mostly railway vehicles as well. Railway logistic terminals and terminals with a significant portion of railway infrastructure belong to this category of service systems. Terminals with such characteristics are in the main focus of our attention.

Let's name a few examples of railway logistic terminals, where the common problems concerning the infrastructure design and planning and the traffic control appears, and also where the similar approach to their solution is applicable: marshalling yards, passenger railway stations, industrial sidings, specialized railway logistic centers (railway depots), multimodal transportation terminals (e.g. container terminals).

We can say that a transportation node can be considered as a generalized service terminal which from the point of view of the infrastructure layout and traffic operations should be understood in a wider sense as an integration of terminals of several types. For example, a need may arise to integrate a freight train station a passenger railway station and a corresponding depot into a single transportation node.

2 Problems of design, implementation and model usage / re-usability

As we go from smaller and less detailed models to more detailed models with higher number of elements, we run into problems with handling such model design and creation and also with the modification and model re-use.

More specifically, these are the following issues:

- How to ensure that the construction of large-scale model is manageable or even how to increase the limiting frontier when a model is still construction manageable.
- How to reduce the time needed to construct and verify a model.
- How to ensure the greatest possible flexibility of the model and re-usability of its parts.
- How to extend the area of applications of a model (generic model building).

3 Simulation principles, architectures and software tools

For the most efficient means to manage the creation of simulation models of large-scale transportation systems, we consider the use of appropriate agent architecture (some of the existing architectures are described in [1]). Our emphasis is on the notation of nonprocedural algorithms - in particular the possibility to directly execute the conceptual model.

3.1 Hierarchical agent architecture

Management structure of service systems (thus transportation logistics systems) is typically hierarchical. Since we consider the hierarchy to be important premise for solutions leading to desired model flexibility, agent architecture must reflect the structure of management entities in hierarchical structure of model agents. Based on the strictly required hierarchical structure, model designers can take advantage of the following features:

- Replacing any agent or submodel with another agent or submodel. This way designer can modify the management of respective subsystem - typical and often utilized reason for such a change is the requirement to model the subsystem at more (or less) detailed level.
- Merging of more submodels (agent trees) to one, realized through adding a new boss agent responsible for coordination of merged submodels.
- Model configuration by selecting from library of reusable agents or submodels.

Thanks to the hierarchical structure of model agents three types of addressing can be implemented. Each inter-agent message can be send in following ways:

- as a standard addressed message – in this case, the message addressee field is filled with the address of responsible agent, which can process the message (the message is listed in agent's direct-messages register),
- as a partially-addressed message – if the message is addressed to a submodel (represented by the boss agent of the model; this will then, based on its

mediated-messages register, determine the agent responsible for message elaboration,

– as a non-addressed message – if the sending agent is unable or not willing (due to the flexibility requirements) to determine responsible agent or even submodel, the message can be sent with empty addressee field.

If a non-addressed or partially-addressed message is to be delivered, then a special addressee searching algorithm, based on systematic hierarchical investigation of message registers, is automatically initiated by run-time infrastructure. Thus, it depends only on a model designers if they prohibit a selected submodel from addressing its agents (except boss) – such a command enables the making of safer structural changes/modification to that submodel. Total flexibility is reached only if non-addressed messages are allowed; however, this concept is connected with more time-consuming demands.

3.2 Nonprocedural algorithms

A possible way of increasing the flexibility of a simulation model is the utilization of nonprocedural (graphically editable) algorithms, which are being executed at model runtime.

3.2.1 Network charts

Network charts [2] seem to be a suitable tool for describing service algorithms (technologies), which typically include parallel activities (e.g. service of a train, after entering the marshalling yard). Network charts provide clear evidence of activity relationships (parallelism, dependency) and they can be constructed without knowing the duration of actual activities. Each edge of the chart is representing a single *activity*. Nodes of the chart can be considered as synchronization points of the operating procedure.

Parameterized derived activities can be incorporated into network charts - they define succession and mutual dependence of activities in a service process. Defined derived activities are reusable and can be used in more than one technology. Network charts may be created in a comfortable graphical editor with support for automatic validation of entered technologies (guarding required succession of some activities and appropriate resource handling).

A finished network chart is then assigned to a customer (e.g. a train or truck). Once the technology has been defined it can be reused – the same technology can be applied to a different customer with the same attendance procedure. Since the network chart is a human readable notification it makes the modification of service procedures of a customer very flexible – user can modify parameters or resources, change mutual dependency of activities (change the network chart drawing) or even exchange the whole technology at once (simply by assigning another technology from the list of defined technologies)

3.2.2 Petri nets

Petri nets represent an appropriate formalism for design of logic related to selected agents [3]. It means in fact that using the mentioned formalism enables rapid and flexible prototyping of relevant Petri nets, which are consequently involved into a simulation model based on agent architecture.

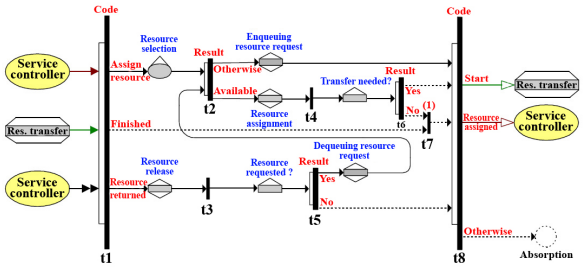

Fig. 1 Petri net example - internal logic of a simulation agent

So we can claim that utilization of Petri nets within agent architecture supports high degree of flexibility, because it is possible to make readily different alternative variants of agents (within the frame of corresponding editor without the need to change the source code of simulation model). In addition, Petri nets can be properly analyzed and verified before becoming a part of a simulator. An example of this formalism can be seen on figure 1.

3.3 Simulation model live cycle

The process of simulation modeling can be divided into several separate stages, each stage should be completed before starting the next stage. On figure 2 the different phases of simulation model development are shown.

The first phase is the *analysis phase* - we insist on understanding the system behavior and naming requirements and its specification. It is not possible to speed up this phase using some supportive software environment. Analysis should be done deeply to avoid later questions about system functionality or desired objectives.

The analysis of the modeled system is followed by the *design* or also called *conceptual modeling* [4] phase. When creating a conceptual model the modeler must utilize gathered knowledge from the previous stage create the design of the simulation model. In our case, the focus is on simulation models of extensive transportation systems which requires - as stated before - design of the following features:

a) hierarchical architecture of model agents

b) communication (message) mechanism between agents

c) model behavior design, which is based on the incoming messages into model agents in our case Petri nets can be used

d) technologies described by a chosen formalism in our case Network charts can be used

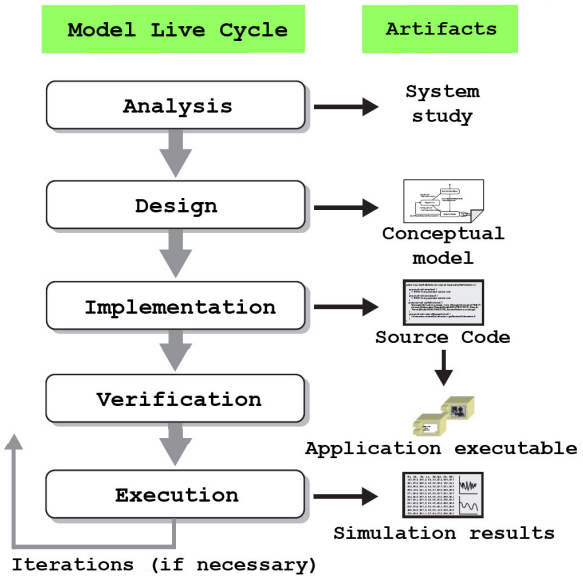

Fig. 2 Live cycle of a simulation model

The efficiency of the conceptual modeling phase could be greatly enhanced by automation using software tools. Following article will explain more on this subject.

After having created the conceptual model, the *implementation phase* consists of coding the simulation model in a chosen programming language. The computer program is being created based on the conceptual model from the previous phase. At the end of the implementation phase, the created program has to be transferred into binary executable.

Verification phase is important to review previous phases and verify, whether there are any design or implementation errors. Next step is the model *validation*, where the real system behavior is compared to the behavior of the simulation model.

Execution phase follows after all the previous phases have been successfully fulfilled, the simulation model is ready to be used and the modeler can carry out simulation experiments.

3.4 Simulation model quality indicators

The essential aim of simulation model designers and users is to design and work with high-quality simulation models. The term "quality of a model" is too general, therefore we choose some *external quality factors* (perceived by the user of a model) as well as *internal quality factors* (indicative for the quality of the internal design of a model) to describe the criteria of the quality of simulation models:

Accuracy is the ability to precisely model the situation for which it has been designed for. It is a primary factor in assessing the quality of a model. If the simulation model doesn't have the proper behavior all the other criteria of quality becomes secondary.

Robustness of the simulation model is the ability of the model to work under unexpected or abnormal conditions. It describes how the model is able to behave and keep running when it's exhibited to unexpected data inputs or situation that is not specified. Robustness criteria can be viewed in a couple of ways:

a) The robustness of the model as a software product - in this respect, the model is robust if it can correctly respond to unexpected internal situations. Internally robust model must react to an unforeseen situation, warning the user with an understandable report of the problem (e.g. invalid attempt to read data, etc.). Robust model does not lead to "catastrophic events", i.e. generate general software exception.

b) The robustness of the model as an experimental environment - in this respect the model is robust if it can treat unexpected internal stochastic impacts (e.g. late arrival of a train into the station, etc.).

When we speak about the robustness of the model we have in mind the robustness of the model as a software product since the other criterion of the robustness is influenced by the simulation model design.

Flexibility is the quality factor, which defines how difficult it is to adapt the simulation to changes in specification or convert it for simulation purposes of other systems belonging to the same category. The flexibility issue arises when models grow in complexity and become extensive. Large models require high levels of flexibility in these aspects:

a) Design clarity - understandable and clear architecture of agents allows designers to make changes more easily.

b) Decentralization - the more self-sustaining components are forming the model, the more likely we have a situation where a simple change will affect only one or a few interacting components and this change will not cause a chain reaction of necessary changes throughout the system.

c) Independence - changes in one part of the model will not cause necessity to make changes in different parts of the model.

Re-usability is the ability of different parts of the model to be used repeatedly or the ability of the whole model to be used as a submodel of another model. Since it appears that some parts of the model could be used in other models, we should be able to exclude those parts and re-use them. A special case is the

situation where we substitute the whole model or its parts for another model, which can serve the same tasks, but using another internal algorithms. This ability also plays important role in the process of working with extensive models.

Efficiency means good utilization of hardware components (i.e. CPU, memory, etc.). In terms of the functionality of the model this is an important aspect of quality assessment, especially as the criterion for simulation speed and memory requirements. Negative influence on the model efficiency may have inappropriate data structures or algorithms. It must be admitted that large simulation models in the aspect of efficiency do not always have satisfactory results.

The *ease of use* is not the criterion of how easy or difficult it is to work with the model itself, but in our case when we are talking about this criterion we have in mind the simplicity of conceptual modeling, implementation, verification and experimenting with the model.

Some of the factors for assessing the quality of the model may be mutually incompatible. It is sometimes necessary to make certain compromises and to reduce the degree to which individual factors are fulfilled.

3.5 Model development approaches

At present, two main approaches exist. In first case, the conceptual model is created in a written form, and is followed by the next phase, where the conceptual model from the written form is being transferred into the source code of a program, which after compilation becomes standalone application. This approach is ineffective, especially in development of extensive and complex simulation models. Converting conceptual model from a written form into the source code is a time consuming task which introduces a high potential of making errors. Also, modifications of the system are mostly in larger systems extremely complicated, therefore the achieved level of flexibility and re-usability is very limited. Therefore this approach is not recommended when making large and detailed simulation studies.

The second commonly used approach is more suitable. It uses a compact software environment which allows the conceptual modeling in a software tool. We recognize two types of such tools, it's either a CASE tool allowing the creation of the conceptual model in a graphical form and then automatically creating model source codes using embedded source code generator (e.g. [5]). Another type of supportive software environment (e.g. [6]) commonly used is a simulation tool allowing model designer and user to carry out all the phases of the model live cycle except the analysis phase, which cannot be automated. This approach is slightly preferable. CASE tool can convert the conceptual model into the source code automatically jumping over the implementation phase. Although this approach is slightly better it also has some serious shortcomings.

When using simulation CASE tools we run into problems with the flexibility. The problem arises when modeler needs to make changes into the conceptual model. After such change the source codes have to be re-generated. The problem comes forward especially when the programmer made custom changes in the source codes and added some information (which is always necessary in order to make the model functional). This information is lost after re-generating the source codes, and must be manually inserted again. This significantly decreases the level of flexibility and re-usability of the system. It also introduces a high potential in making implementation errors by accidently changing the generated source codes. These errors are very hard to trace.

With the usage of simulation packages we often run into problems with a limiting set of components that are provided by the simulation tools. These tools often provide bounded set of tools and the modeler can't go beyond this set. Another limitation is in size of the model, these tools often have limitations in number of components that are used in the model. Such tools often provide the user with their own scripting languages, but most of the times user can't go beyond the set of provided simulation tools. All these factors are negatively influencing the construction of extensive simulation models with a large number of system elements.

As mentioned above the current software tools bring number of unresolved problems into the large model development process, therefore we suggest another approach and techniques.

3.6 Software support as aim of speeding up model live cycle phases and increasing quality indicators

Some of the phases of simulation model live cycle are quite time consuming, especially the design and implementation phase.

The fact is that in the implementation phase a high potential for making errors exists. Our effort is to the most possible extent decrease the potential for making errors in the model development process and automate as much as possible some of the time consuming phases. One possibility to improve the model design efficiency would be to create an environment, in which all the phases of the model development are carried out. In this type of environment the model would have to be designed (by the GUI), implemented (where a conceptual model becomes both machine and human readable) and executed. On one hand this mentioned approach is advantageous, because it allows the modeler to carry out all the modeling phases in one compact environment. On the other hand, this approach brings some limitations.

The programmer cannot directly interfere with the simulation model source code. This means, that the simulation model can be constructed only from components contained in the environment. When trying to build large and detailed models *this factor is very limiting*.

Suggested solution is oriented to minimize unwanted effects coming along with realization of particular stages of simulation model development. Basic idea comes from a *strong relation between the conceptual model and the implementation.*

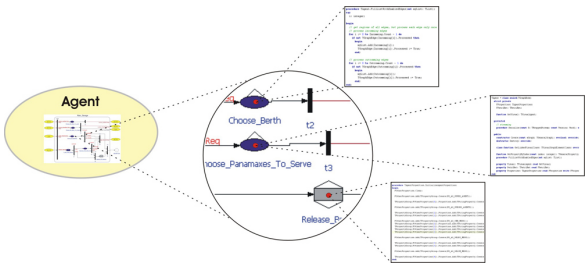

Fig. 3 Linkage between agents, their internal logic (Petri nets, Network charts) and the executable methods written directly into the source codes by the GUID.

We suggest that the conceptual model (the output from the design phase) is directly used for the simulation purposes. That means the conceptual model has to be both human and machine readable. The fact, that the conceptual model becomes machine readable greatly decreases the amount of the generated source codes, because most of the generated source codes would have been used to describe the basic model structure. The conceptual model should be exported in a binary format and the simulation kernel must be capable of loading and using this format for simulation purposes.

To give the programmer virtually unlimited versatility in the simulation model enhancement, we had to solve the problem of interconnection of the exported binary structure and the programmed source codes. This was solved by giving every element of the simulation model (e.g. agent, message edge, assistant etc.) a unique identifier GUID (globally unique identifier), which corresponds to an *execution method* containing the source code of this element. By this mechanism (figure 3) we were able to create a hybrid system, where the simulation kernel is directly using the conceptual model which is interconnected with the simulation model source code.

In application of this design, the term *conceptual model* got new enhanced meaning. Now it represents not only the simulation model structure and configuration, but it also represents an executable part of the simulation model. The simulation kernel is capable of using the binary format of the conceptual model without requiring the source codes of the conceptual model. In application of this approach, conceptual model becomes human readable and also readable and executable by the computer. As a result we have significant reduction of simulation model source codes, which indirectly increases the flexibility of the simulation model, and its robustness. This approach also greatly contributes to the implementation phase and reduces the error potential in model programming.

This approach has a positive effect on various quality indicators. First of all, as mentioned earlier it increases the *flexibility* of the system, by reducing the amount of the source codes: Most of the source codes describing the model structure and model logic don't have to be generated at all. With lowering the amount of source codes used, we increased the *robustness* of the model, since the modeler is not allowed to access the model structures and therefore cannot accidently or on purpose change the behavior of the model. Since all the design and modeling is done in a graphical form, we increased the *ease of use* quality indicator. It's more natural for modelers to use graphical language to describe the model structures and reactive logic utilizing Petri nets and Network charts. Another positively influenced quality factor is the *re-usability* of the model or of its parts. This factor is very significant when trying to build generic simulation models.

Our hybrid system with all the advantages of CASE tools give the model designer virtually unlimited space to enhance the model behavior by writing the source codes. Each component of the model has its execution method, which can be implemented in a standard development environment (e.g. Microsoft Visual Studio, Delphi, etc.). These execution methods are connected with the model structure using GUID, therefore there is no need to recreate these source codes. The only need is to configure the linkage methods, which is done semi-automatically with the help of built-in simple source code generator.

4 Implementation example

The previous section of the article describes the lifecycle of a simulation model and introduces our approach in improving the modeling process. In this section we will talk about the generic model builder software environment called ABAbuilder, which is based on our own agent architecture ABAsim (Agent Based Architecture of Simulation Models). More about this architecture can be found in [7].

Next section briefly talks about generic simulation model Villon. This simulation model has been designed for making simulation models of a wide range of transportation service systems.

4.1 ABAbuilder software environment

In this section we will briefly discuss the features of the ABAbuilder software environment which proved its important role in conceptual modeling process. For designer it makes the designing process semi-automated, with a variety of functions for altering the model, such as changing the hierarchy of agents, testing different message models, etc. From the programmers perspective it speeds up the model implementation with utilizing the source code generator which is capable of generating the basic model structures.

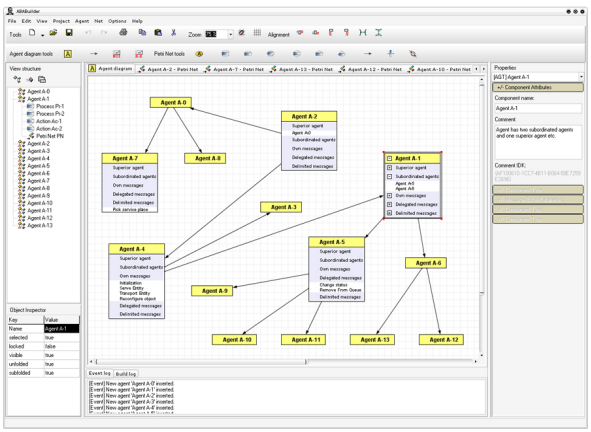

Fig. 4 Screen capture of the ABAbuilder software environment

ABAbuilder is a software environment based on the ABAsim architecture and its principles. It supports several aspects of model development including system analysis, design of communication, and use of methodologies, prototyping and model maintenance. 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 preprogrammed behavior and decision making algorithms.

The similarity of control structures from the real world allows relatively easy design of control components of the simulation model using ABAbuilder software environment. A simulation model structure can be described by the hierarchy of system agents (figure 4). From the ABAbuilder point of view, not only agents, but also their internal components are considered as model building base.

To be able to accomplish individual phases of the simulation model live cycle this environment had to be divided into three 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 (Petri nets). It allows users to add new agents to the hierarchy, edit their internal components and define the message flow inside of the model. Petri Net editor is used to define the reactive logic of each agent. 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 issues, that are not critical for the model but might be dangerous during the runtime.

When the decision is made by the user and the model is ready for deployment the *source code generator* module comes into use. This module first of all creates so called execution methods of model components. These methods are empty and the programmer is capable of implementing custom model behavior. In the next step it generates the method which interconnects these execution methods with model elements by the GUID. Libraries containing the simulation kernel, which allow compiling a standalone application, are provided as a part of the ABAbuilder software environment. These libraries are actually part of the simulation project.

Creating complex simulation models in the ABAsim architecture such as the model of a container terminal or a marshalling yard would be very hard to accomplish, if not almost impossible without existence of this software environment.

4.2 Generic simulation model Villon

As a bright example of architecture possibilities and flexibility, generic simulation model Villon [8] can be mentioned. Villon is detailed microscopic simulation model of logistic junction operation with predominant transportation processes. The junction is understood to be e.g. railway marshalling yard, railway station for passenger transport, factory sidings, airports, container terminals, etc. Utilizing the ABAsim architecture, Petri nets and Network charts Villon tool is flexible enough to model different means of transportation considering their specific attributes.

Using Villon one can build simulation models of railway marshaling yards with only railway traffic, but also combined models, with inter-modal operation. Villon was successfully used in commercial environment to provide services for many clients, among them are national railway companies from Germany, Austria and Switzerland, private companies BASF, Volkswagen and many others.

5 Conclusion

Given the current state of the art information technologies and simulation methods, the important decisions making process in the management of complex service systems should not be taken without the previous modeling of the consequences by the means of discrete simulation. Costs associated with the simulation study represent only a fraction of the cost of the damage that could be caused by bad decision applied in reality.

In developing extensive models of complex transportation service system systems it is necessary to choose the right methods, that allow modelers to respect the structure of the real system and allows them to create a flexible, open and easily maintainable simulation models. The fact remains that without appropriate architecture e.g. ABAsim and supporting software environment e.g. ABAbuilder the task of detailed modeling becomes very complicated, in some cases might be even unmanageable. The solutions presented in this article unlike from the past allows us to create much larger models and models meeting high quality requirements.

6 References

[1] Davidsson, P., Henesey, L., Ramstedt, L., Törnquist, J., Wernstedt, F.: *Agent-Based Approaches to Transport Logistics*, Whitestein Agent Technology series, 2005

[2] Adamko,N., Klima,V.: *Definition of Operating Procedures in Generic Agent Based Simulation Model of Transportation Logistic Terminal*.In: Proceedings of European Simulation and Modelling Conference 2008 *pp. 312-316*, Le Havre, France 2008.

[3] Kavicka, A.: *Petriho síť s rozhodovacími přechody aplikovaná v rámci ABAsim architektury simulačního modelu*, Sborník přednášek z 37.mezinárodní konference "MOSIS '03"- konané v Brně, MARQ - Ostrava, 2003, str.373-380, ISBN 80-85988-86-0

[4] Takashi, I., Yoshiaki, M., Nozomu, A.: *From Conceptual Models to Simulation Models: Model Driven Development of Agent-Based Simulations*, Faculty of Policy Management, Keio University, 2003

[5] Féliot, D., Bellissard, L., De Palma, L.: *OLAN Tools*, SIRAC Project, St. Martin, France

[6] http://www.simcreator.com/

[7] Adamko, N., Klima, V., Kavička, A., Lekýr, M.: *Flexible hierarchical architecture of simulation models*, Proceedings of European simulation and modelling conference, Eurosis, Paris, 2004, pp.30-34

[8] Klima, V., Kavička, A., Adamko, N. 2001. "Software tool VirtuOS – simulation of railway junction operation", In: *Proceedings of ESM 2001 conference*, Prague, Czech republic

ACKNOWLEDGEMENT

This work has been supported by the National Research Program of Czech Republic under project No. MSM 0021627505 "Theory of Transportation Systems"