Int j simul model 6 (2007) 1, 13-24
ISSN 1726-4529 Original scientific paper

SIMULATIONS OF TRANSPORTATION LOGISTIC SYSTEMS UTILISING AGENT-BASED ARCHITECTURE

Kavicka, A.[*]; Klima, V.[**] & Adamko, N.[**]
[*]Jan Perner Transport Faculty, University of Pardubice, Studentska 95,
532 10 Pardubice, Czech Republic
[**]Faculty of Management Science and Informatics, University of Zilina, Univerzitna 8215/1,
010 26 Zilina, Slovak Republic
E-Mail: antonin.kavicka@upce.cz; valent.klima@fri.utc.sk; norbert.adamko@fri.utc.sk

Abstract
This paper presents a methodology related to rapid and flexible prototyping of simulation models reflecting large-scale transportation logistic systems. This methodology is based on a proprietary architecture (called ABAsim) inspired by the paradigm of autonomous agents. The architecture was designed within the framework of research and development of an integrated computer environment, which is specialized for simulations of a wide class of transportation logistic systems.

Basic characteristics of the mentioned architecture are briefly described, whereas a flexibility of simulation models, developed within that architecture, is emphasised. The high degree of flexibility is enabled due to consistent hierarchical structure of relevant conceptual models and indirect addressing mechanism utilised for inter-agent communication. Because of the applied approaches, the simulation model configuration can be built by replacing relevant sub-models, individual agents or their internal components. In addition, exploitation of Petri nets for definitions of control procedures and utilisation of a proprietary CASE-tool also supports rapid and flexible model prototyping.

There is also presented a complex simulation tool Villon (developed within ABAsim architecture) and corresponding experience with its utilisation within simulations mirroring compound systems from the field of transportation logistics systems.
(Received in May 2006, accepted in February 2007. This paper was with the authors 3 months for 2 revisions.)

Key Words: Agent-Based Simulation, Simulation Model Architecture, Petri Nets, Simulation of Transportation Logistic Systems

1. INTRODUCTION

There are many systems, the basic processes of which correspond to orders of various kinds of services and the realization of those services - such systems are called *service systems*. The realization of a service is usually related to further orders, i.e. the mentioned systems are strictly hierarchically oriented. The most complex service systems involve complicated transportation processes, which represent substantial portion of all studied processes (e.g. railway stations related to passenger or freight traffic, multimodal terminals etc.). Those systems are called *transportation logistic systems*.

Essential questions should be asked before we start building simulation model of a complex transportation logistic system. Are we able to create that model? Is that potential model flexible enough in order to carry out simulations following various scenarios or even reflecting different kinds of transportation logistic systems? Final success depends namely on an applied methodology and a kind of model architecture that has to include the following features and methodologies.

Model designer should cooperate closely with technologists of a modelled system within the frame of all stages of simulation study. This cooperation is possible only on a level of conceptual model. So, the conceptual model structure should plausibly mirror the structure of modelled system.

Once the model (or its parts/components) is built, it should be utilised many times (also because of the costs of its development) within various stages of design, building, operation or maintenance of the modelled system. Therefore it is needed that the applied model architecture supports *reusability* of individual components and sub-models. In addition, the simulation model is supposed to be configured on a conceptual level, i.e. the model can be structured of ready-made sub-models (components) or it can be used replacing sub-models with rough granularity by sub-models applying more detailed granularity. These features are denoted as model *flexibility*.

The architecture ought to support non-procedural programming to minimise writing source codes. Non-procedural (declarative, graphical) parts of the simulation programme substantially increase its flexibility. Finally, an appropriate architecture should be supported by a CASE-tool for realisation of an effective model development.

Facing the challenge of developing software tool for simulation of transportation logistic systems, we have decided to base its development on proprietary agent-based simulation architecture, which provides the needed level of flexibility, manageability and extensibility (Section 2). Section 3 presents those features of a discussed architecture, which substantially support model flexibility. At the end (Section 4) we illustrate all described approaches and methodologies on the case of simulation model built within presented architecture and utilised for simulations of many complex transportation logistic systems in real applications.

2. AGENT-BASED ARCHITECTURE

Architecture *ABAsim* (initially mentioned in [1] and elaborated in [2], [3], and [4]) was mainly developed for simulation of large service systems. A *service system* is understood to mean a system focused on an execution of *orders* (attending to *customers*) and the realization of *services* relating to them. These services can further initiate another set of orders. Such service systems include a wider class of various systems – e.g. factories, transportation and logistic nodes/junctions, hospitals, repair shops, etc. Natural and technical systems do not belong to this class.

Let us mention the most important features of service systems, features of which essentially influence the architectural properties:

- *Service* system structures can be considered (from the point of view of order elaboration) strictly *hierarchical*. The order (the customer) entering the system (e.g. production of a car, treatment of a patient) initiates a recursive sequence of suborders, according to the rules of competence redistribution.
- All system elements (subsystems) work in a synergic way, unlike the majority of natural systems, with the common goal of executing the order. Thus, the architecture cannot be expected to work with the processes of: evolution, competition, or parasitism.
- The entities within a service system (orders/customers and resources) can be divided into specialized classes with the same behavioural rules for all included entities. This means that the responsibility for the behaviour of these entities is taken over by their superior subjects (agents). Hence, there is no reason to consider individual entities as agents.
- Service systems usually represent large-scale systems. It is necessary in most cases to transfer service resources to the customer (or vice versa), in order to actualize the service activity. Frequent and complex transposition processes are typical within such systems.

2.1 Agent

First let us remind the generally-respected agent definition [5]: *Agent is an encapsulated computer system situated in some environment and capable of flexible, autonomous action in that environment in order to meet its design objectives*, where the agent features are as follows:

- *Autonomy* – i.e. an agent is able to work autonomously without exogenous interventions, entirely able to control its activities and inner status.
- *Social behaviour* – this is made manifest by the agent's interaction with other agents (or with human beings) by means of some communication mechanism or language.
- *Re-activeness* – an agent responds to external influences from its surrounding.
- *Pro-activeness* – an agent acts with initiative and goal-orientation.

In addition, the agent is potentially able to "improve" continuously its behaviour through the ability to learn.

The main agent functions in ABAsim architecture are described in Fig. 1. A given task or goal is assigned to each agent. The agent then realizes, according to its mission, its own life-cycle: *sensing – decision making – acting* (within its life space) using the support of *solving* (focused on making solution proposals) and *communicating* with other agents (eventually with human operators). If the agent detects a problem or a situation beyond its delegated competence, it informs other agents about the need for a corresponding solution.

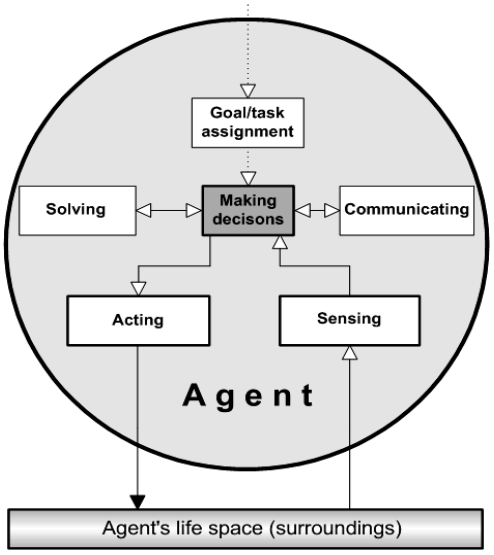

Figure 1: The agent functions.

2.2 Agent components

Each agent can be decomposed into the following groups of *internal components* (Fig. 2):

a) The first, *control* and *decision making* component (called the *manager*) is responsible for making decisions and for inter-agent communication. In addition, the manager represents the central agent component because it initiates the work of other internal components and can also communicate with all of them.

b) The group of *sensors* is specialized for mining information from a state space. This group is composed of two kinds of components - the *query* delivers the required forms of information instantly, and the *monitor* scans the state space in some time interval and continuously brings important information to the manager.

c) The next group, called *solvers*, provides solutions for problems to the manager, which can accept them or asked for alternative ones. The *advisor* is a passive component able, by return, to react only to the manager's requests for delivery of proposals for problem-solving. The typical advisor can be represented e.g. by optimization algorithm, neural network, fuzzy regulator or a human operator. On the other hand, the *scheduler* (focused on a restricted scope of problems) works continuously for the manager, on the basis of either a priori rosters or schedules, which have been created (e.g. connected with the allocation of resources), or by making its own dynamic forecast for a defined time interval.

d) The last component group includes *effectors* (actuators), which make changes to system status after receiving corresponding instructions from the manager. No other agent components are allowed to make these changes. An *action*-component makes instant state changes (e.g. switch traffic lights, close a train's doors), while a *process*-component (e.g. a crane's movement) makes them continuously until its task is finished.

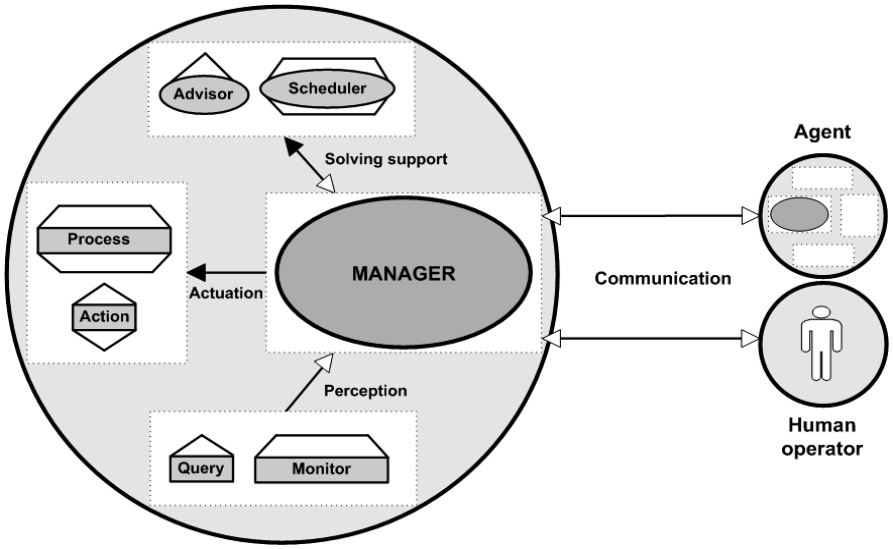

Figure 2: Agent's decomposition.

The effectors, sensors, and solvers are, for brevity, given the umbrella term of manager's *assistants*, and can be further distinguished as:

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

The question arises, how to realize appropriately the internal agent components. They can be described alternatively either

- using *imperative approach* (implementation of program routines constructed in a given source code), or
- by means of *declarative forms* (connected with some kind of symbolic formalism), which are reflected by a structured input data and "vitalized" by a corresponding interpreter. Petri nets [6] [7] represent one of the most effective formalisms appropriate for describing agent internal components.

2.3 Community of agents and its structure

Simulation models for simple real systems could be composed of only one agent; however, the simulation of complex service systems is obviously connected with a *multi-agent*

approach (e.g. mentioned in [5]), using the agents within some organizational structure. Let us remark that the philosophy of ABAsim architecture was also partly inspired by the paradigm of reactive agents [8], which is based on a society of reactive rather than proactive agents. The intelligence of such society *emerges* when one observes the whole community and not its separate members (individually of relatively low intelligence).

A multi-agent hierarchical system can be demonstrated through the following example (Fig. 3), where A_0 (agent of service centres) divides the management of the relevant service company into two partial managements of affiliated service centres. We say that agent A_0 *delimits* the company management to two peer affiliate agents A_1 and A_2, which inform their *boss* about important facts concerning the entire company. Agent A_0 can also send important supra-affiliate information to its *subordinates*.

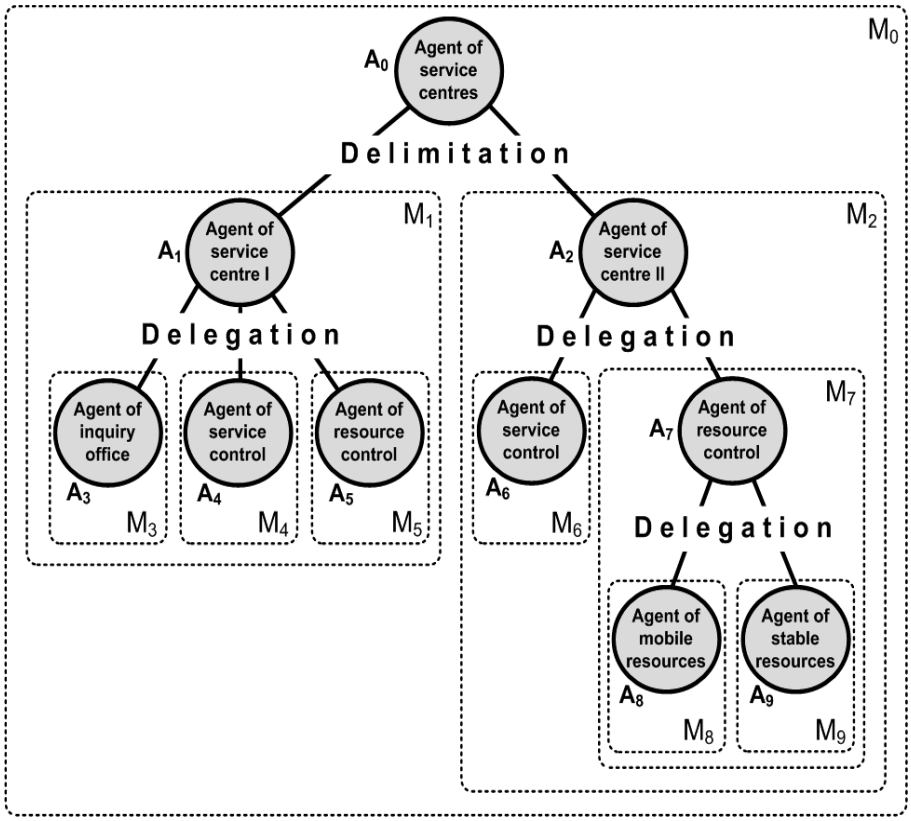

Figure 3: Hierarchical organization of agents and models.

It is expected that both affiliates mentioned above carry out the same kind of management operations. Each of the affiliate agents further utilizes a set of subordinated specialized agents in order to fulfil its own defined tasks. This means, in fact, that affiliate agents *delegate* some portion of their competences to hierarchically lower-ranked colleagues. It is obvious from the above-mentioned figure that agent A_1 utilizes a different structure of subordinates than its colleague A_2 (it reflects specific needs of each affiliate). Another point of view is that a hierarchical structure of agents (set A) represents a hierarchical structure of models (set M, $|M|=|A|$), whereas each model $M_i \in M$, ($M_i \subseteq A$) is composed of a tree of agents with the root/agent $A_i \in A$, i=0,..., $|A|$-1. The agent A_i is a representative (boss) of the whole model M_i – we can alternatively utilize the notation ${}^{model}A_i \equiv M_i$. 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. Finally, it is possible to denote the structure of model M_0 in the form of an algebraic expression: $M_0[M_1(M_3,M_4,M_5),M_2(M_6,M_7(M_8,M_9))]$, 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.

To summarize the philosophy of ABAsim operation: The control role is played by mutually communicating managers (supported by sensors and solvers), which initiate the activities of effectors at the correct time instants and under particular conditions.

2.4 Communication mechanism

One way to realize inter-agent communication is to use standard communication languages (e.g. KQML [10] or FIPA-ACL [11]). Another approach is to implement a customized communication mechanism able to reflect, in the best way, the features of the respective architecture.

Communication within ABAsim architecture is based on a simple, original mechanism applied to *inter-agent* and also *intra-agent* communications. As was already mentioned, inter-agent communication is made by manager components, and intra-agent communications are realized between the managers and their assistants. Both kinds of ABAsim-communications utilize exclusively the paradigm of sending messages (from this viewpoint, ABAsim represents *message-oriented architecture*).

The following description simply characterizes selected kinds of messages used within ABAsim architecture. *Notice*–messages contain some information for the addressee without expecting any answer, *Request*–messages carry specific demands, which are expected to be satisfied or supplied by means of corresponding *Response*–messages. Continual assistants are initiated by *Start*–messages (sent by superior managers), whereas *Finish*–messages (sent by continual assistants) delivered to corresponding managers, indicate completion of an activity related to relevant continual assistant. In addition, managers exploit *Execute*–messages in order to obtain promptly required results from their inferior instant assistants. Finally, *Hold*–messages exclusively mediate the augmentation of simulation time. They involve so-called *time stamps*, which define the durations of their deliveries (equal or greater than the current simulation time). The attributes *sender* and *addressee* contain the same values – i.e. the continual assistants send those messages to themselves with some time delay. Thus, after sending *Hold*–message, the continual assistant remains idle and resumes its activity after the message returns. We have to emphasize that the augmentation of the simulation time is realized exclusively by continual assistants, i.e. synchronization of simulation time is based on synchronization of these components.

2.5 Run-time infrastructure

The current implementation of ABAsim's runtime infrastructure provides means for execution and synchronisation of discrete and combined discrete-continuous simulation models [12]. Runtime infrastructure is modular and comprises three different modules: discrete simulation module (main kernel module), continuous simulation module and animation module. Animation module provides basic support for animation of modelled discrete and continuous activities, utilizing the time-stepping approach.

The newest run-time infrastructure also supports execution of simulation models in distributed environment, synchronising distributed sub-models by means of proprietary developed hierarchical conservative synchronisation algorithm [4]. The set of basic services, which run-time infrastructure offers to simulation model designers, is supplemented by support for flexible message addressing and interpreter of Petri nets.

2.6 ABAsim versus other agent-based architectures

Seeing that general paradigm of autonomous agents influenced the design and development of ABAsim architecture, it is only natural that the architecture shares some common principles with other agent based simulation architectures that were inspired by the same paradigm. Among many, we can mention for example Cougaar architecture [13] (with similar hierarchical organisation of agent communities or agent decomposition to simpler executive units) or architecture HIDES [14], which shares the same view on importance of hierarchical structure of agents reflecting modelled system and supports forming of agent communities responsible for specific tasks.

Since its beginning, ABAsim architecture was oriented to creation of simulation models of complex large-scale service systems, with emphasis on flexibility for simulation model designers, programmers as well as for end-users of simulation models.

3. FLEXIBILITY OF ABAsim ARCHITECTURE

To be successful in application of ABAsim architecture, it is essential to enable fast and flexible prototyping of simulation models reflecting studied systems. The bellow mentioned features present applied approaches, methodologies and tools, which support those expectations.

3.1 Hierarchical structure

Management structure of service systems is typically hierarchical. Since we consider the hierarchy to be important premise for solutions leading to desired model flexibility, ABAsim architecture reflects the structure of management entities in hierarchical structure of model agents. Based on the strictly required hierarchical structure, model designers can enjoy following features:

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

Implementation of the flexible agent and sub-model configuration abilities is dependent on the existence of adequate inter-agent message addressing mechanism.

3.2 Message addressing

The modularity of a simulation model, as well as of individual agents, represents features, which highly support flexibility of a simulation model within ABAsim architecture. For example, if the structure of sub-model $M_1(M_3,M_4,M_5)$, illustrated in Fig. 3, should be changed to $M_1(M_3(M_{10},M_{11}),M_4,M_5(M_{12},M_{13}))$; then, within other model parts, it may be necessary to modify the identifiers of the addressees (agents) hierarchically subordinated to M_1. The reasons for changing a sub-model structure can be motivated e.g. by an additional decision to implement that sub-model on a more detailed level. In spite of that, simple standard message-oriented communication mechanism will not enable such a level of flexibility, which would allow changes to a sub-model structure without the need to make additional changes within other model parts. Therefore, ABAsim architecture utilizes flexible message addressing system, which is exploiting hierarchical structure of models and is based on message-processing registers. Each agent keeps two distinct message registers:

- a register of *direct-messages*, which holds messages, agent can directly process and
- a register of *mediated-messages*, which contains messages that could be processed by agent's subordinated agents.

The union of these registers gives the set of messages, which can be elaborated in agent's model. Each inter-agent message can be send in following ways:

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

If a non-addressable or partially-addressable 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 designer if he/she prohibits a selected sub-model from addressing its agents (except *boss*) – such a command enables the making of safer structural changes/modification to that sub-model. Total flexibility is reached only if non-addressable messages are allowed; however, this concept is connected with more time-consuming demands.

3.3 Flexibility supported by Petri nets

Petri nets represent (as already mentioned above) an appropriate formalism for design of logic related to selected agent components, namely managers. 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 ABAsim architecture. Let us adduce an intuitive illustrative example of coloured Petri net (Fig. 4) defining the logic of a manager component encapsulated by agent of resource control (A_5) from the Fig. 3. Presented net disposes of two special places (p_1, p_{14}), where p_1 (input place) accepts tokens/colours, which reflect incoming messages of studied agent and p_{14} (output place) receives tokens representing outgoing messages from that agent. The transitions denoted as $a_1, \ldots, a_7$ correspond to relevant instant assistants, which are respectively executed during the net evolution. Transitions s_1 and s_2 realise sending outgoing messages through the place p_{14}. And finally, transitions $d_1, \ldots, d_4$ represent points of conditional branching.

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

3.4 CASE tool

To ease the sometimes tedious task of creating complex simulation models based on the ABAsim architecture, computer aided software engineering (CASE) tool is provided for simulation model designers and programmers. The tool, named ABAbuilder, supports several aspects of model development including conceptual modelling, design of communication, model maintenance and re-engineering. ABAbuilder provides visual graphical environment

for the users to define hierarchical agent structure of the model, to design communication by definition of messages to be sent and processed by agents, to define components and edit Petri nets of model agents. As an output, the tool produces source code frames of the model in chosen programming language (currently only Object Pascal is supported), programmers then only fill-out executive commands of assistants. ABAbuilder supports re-engineering and model flexibility by its ability to analyse existing source code (not necessarily generated by ABAbuilder) and converts it to graphical representation, so users can adjust the model structure, exchange agent components, redefine Petri nets or modify other model properties in visual environment.

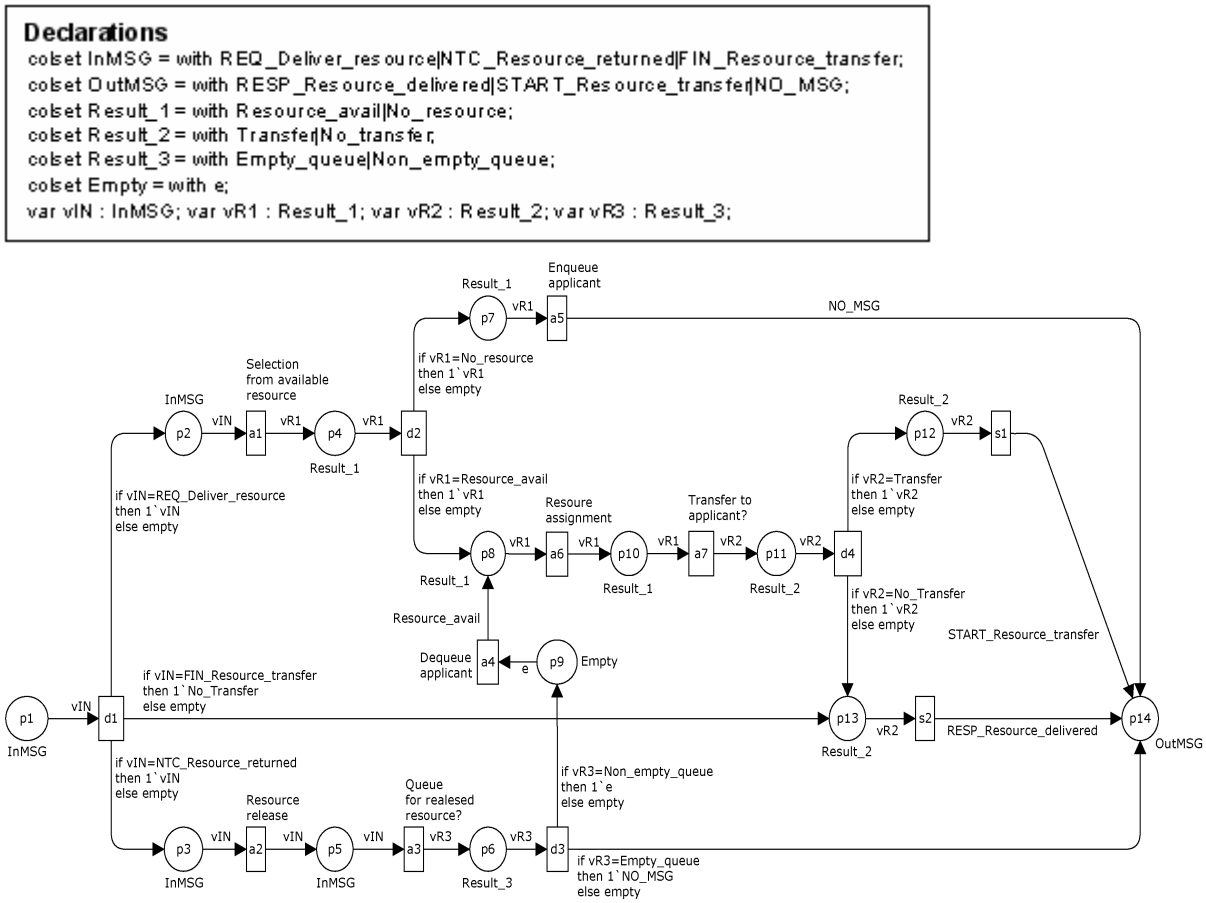

Figure 4: Coloured Petri net reflecting a manager related to the agent of resource control.

4. APPLICATION OF THE ARCHITECTURE – SIMULATION TOOL VILLON

The properties of ABAsim architecture made it possible to design and implement complex simulation tool for modelling of transportation logistic systems, called Villon [15] [16]. Simulation tool Villon allows users (professionals in the field of logistics) to create simulation models of logistic systems, to run prepared scenarios as well as to evaluate results of simulation runs, without the need to write a single line of program code – utilizing only Villon's user-friendly interface. The creation of complex simulation models of logistic systems, of course, requires a certain level of experience and knowledge, however, using the Villon simulation tool, even less experienced users are able to create simulation models of simple logistic systems within a short period of time (few days).

Villon is a complete simulation system; it provides the user with comfortable user-friendly editors to edit all needed data to run a simulation model, supports customisation of many aspects of simulation runs, offers animated output of modelled activities in 2D or 3D view (Fig. 5) as well as extensive set of post-run evaluation tools (including statistics, graphical protocols, etc.).

Even though the development of this tool was motivated by the ambition to create complex simulation model of a marshalling yard, nowadays Villon is able, thanks to the architecture flexibility and its other valuable properties, to support modelling of various types of logistic transportation nodes (e.g. railway passenger stations, train depots, factories including road transportation, ground handling within airports etc.).

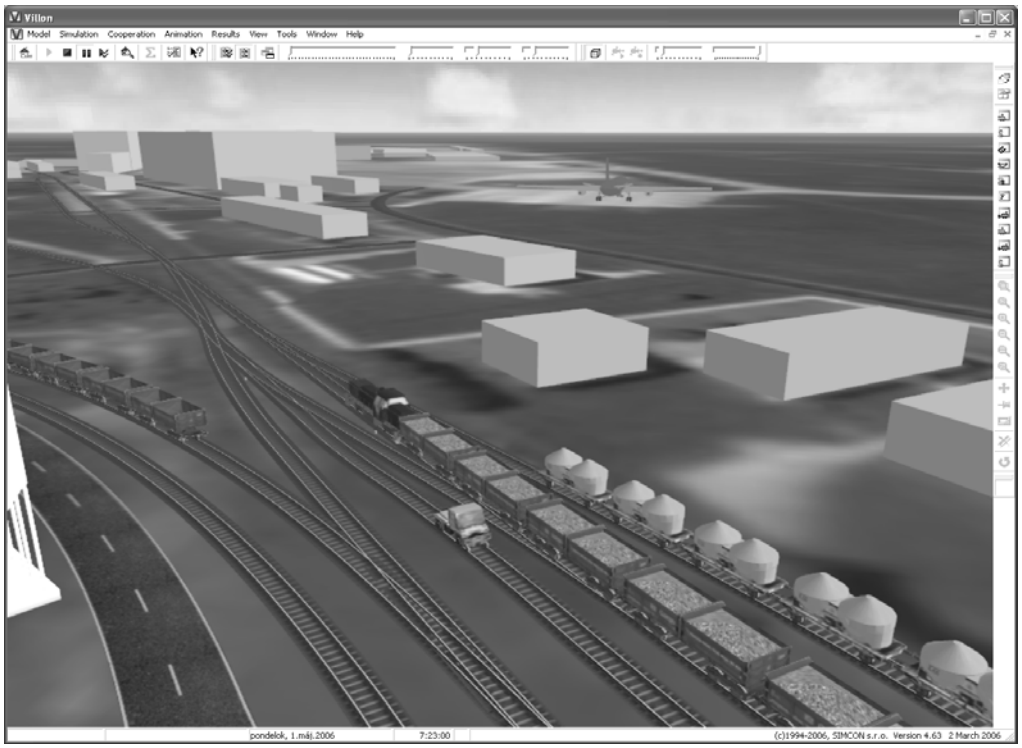

Figure 5: Simulator's animation output using 3D visualisation.

Utilization of ABAsim architecture in the development of the tool gave programmers the possibility of comfortable modelling functionality augmentation of the tool (e.g. by adding more agents and sub-models to the existing model structure) without the need for extensive program code changes in existing parts of the tool. For example, programmers can simply add the support for another type of service resources (e.g. fork-lifters) by adding the responsible agent to the hierarchical structure. At present, Villon's structure contains more than a dozen of agents responsible for various tasks, organised in a hierarchical manner. To mention only a few, there is *Dispatcher agent*, set of resource agents (infrastructure, locomotives and personnel), *Movement agent*, *Crossings agent*, *Surroundings agent*, etc. Even at present we are continuously improving Villon by adding more modules (agents and sub-models) and by modification of existing agents and their assistants. During the configuration of a simulation model, experimenter is able to specify modules that the model will use, which leads to corresponding structure of agents (agents responsible for not requested functionality are not included in the model structure).

Features provided by the ABAsim architecture are not only used by programmers during development and implementation of the Villon tool, they are also mediated to Villon's users

(who are usually not programmers). For example, the users have the chance to choose suitable algorithm for resource assignment. This is in Villon implementation translated as a substitution of advisor component of Resource Agent for another one (which implements alternative algorithm of resource assignment) from the pre-programmed palette of available advisors. If there is a need to alter agent's behaviour in greater extent, the user has the possibility to modify the Petri net describing agent's behaviour; however this task requires basic Petri net knowledge and certain algorithmic abilities.

Extensibility and flexibility of Villon tool even surpass the features directly offered by ABAsim architecture, however without the support from the architecture, these additional features would not be so easy to implement. Let us mention only one example: Since Villon has to support modelling of various logistic node types, it does not contain any hard-coded technological procedures – simply said, Villon itself is not able to perform any task without proper "program", which is entered in a form of a flowchart by the user during creation of the model. Flowcharts are created in a comfortable graphical editor and are composed of activities, which are chosen from predefined set of activities known to Villon (these are prepared by programmers, e.g. loading, resource assignment, etc.). User has the chance to modify various parameters and resources needed for the activity to be executed (e.g. movement speed). Ready flowchart is then assigned to a customer (e.g. a train). During simulation run, specialised agent in Villon interprets defined flowcharts – each activity is simply translated to a message that is at a proper time (respecting the order and dependences between activities defined in the flowchart) sent to the agent, which carries out the activity.

Finally, we adduce some important applications for the mentioned simulator. It was applied within simulation studies, which paid attention mainly to the marshalling yards in Austria (Vienna, Linz), in Germany (Hamburg Alte Süderelbe, Oberhausen-Osterfeld), in Switzerland (Lausanne, Basel), and in China (Mudanjiang, Harbin). In addition, other types of railway and logistic centres were also modelled using this simulator – e.g. railway depot in Ulm (Germany), the factory sidings of the chemical plant BASF Ludwigshafen (Germany), the internal railway traffic of the paper producing company SCA Laakirchen (Austria) and steel production company Voest Alpine Linz (Austria), the passenger station of Beijing (China), the internal traffic of the car production company VW Bratislava (Slovakia), etc.

5. CONCLUSIONS

On the basis of actual experience with the application of ABAsim architecture, we can emphasize namely its following advantages:

- Simulation model structure is very close to the structure of a simulated system mainly from the point of view of the organization/hierarchy of the system control units. This feature is useful not only for model designers, but also for improvement of communication with customers.
- Clear decomposition of the entire model enables to utilize and to reuse its sub-models, agents as well as its individual components.
- Operator can be integrated into a simulation model in a very natural way – he/she can be understood either as an agent or as one of its components (advisory, sensorial, etc.).
- It supports formation of versatile and flexible simulation models rather than single-purpose ones. It is natural to build bases/libraries of alternative components, agents, and models. It is viable to "mix" the required model version/alternative with the help of those predefined elements – which means that rapid configurations and scenario preparations are assisted. Thus, designer can easily modify (mostly because of the non-procedural ways of programming):

  - executive properties of an agent (selecting from its effectors),
  - decision-supporting features (choosing from a set of sensors and solvers),
  - agent control strategies (using various agent "brains", i.e. manager-components),
  - complex parts of simulation model (activating alternative agents and sub-models).
- An experimenter can form model configurations and scenarios of experiments by means of editing tools, without the need to modify the code of a simulation program.
- The concept of message-oriented architecture enables distributed simulations.

Having a long-term experience with practical application of the ABasim architecture, we are convinced that it pushes forward the complexity limit of transportation logistic systems, for which we are able to create flexible and maintainable simulation models.

6. ACKNOWLEDGEMENT

This work has been supported by the Czech National research program under project MSM 0021627505 "Theory of transportation systems" and by the grant of Slovak Ministry of Education VEGA 1/4057/07 "Agent-oriented models of service systems".

REFERENCES

[1] Klima, V.; Kavička, A. (1996). Agent-based simulation model design, *Proceedings of European simulation multiconference*, SCS, 254-258
[2] Kavička, A.; Klima, V. (2006). Agent-based simulations of transportation nodes - methodology and applications (invited paper), *Proceedings of 40th Spring international conference Modelling and simulation of systems - MOSIS '06*, Czech and Slovak Simulation Society, 9-20
[3] Kavička, A.; Klima, V. (2005). ABAsim: Agent-Based Architecture of Simulation Models, Šnorek, M.; Štefan, J. (Editors), *Simulation Almanac*, Czech and Slovak Simulation Soc., 63-72
[4] Adamko, N.; Klima, V. (2005). Distributed Agent Based Simulation Architecture with Hierarchical Time Synchronisation, *Proceedings of ESM conference*, Eurosis, 123-127
[5] Jennings, N. R. (2001). An agent-based approach for building complex software systems, *Communications of the ACM*, Vol. 44, No. 4, 35-41
[6] Girault, C.; Valk, R. (2002). *Petri Nets for Systems Engineering*, Springer Verlag, Berlin
[7] Jensen, K. (1997). *Coloured Petri nets – basic concepts*, Springer Verlag, Berlin
[8] Brooks, R. (1991). Intelligence without representation, *Artificial Intelligence J.*, No. 47, 139-159
[9] Maes, P. (1990). *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back*, MIT Press, Cambridge
[10] Finin, T.; Labrou, Y.; Mayfield, J. (1997). KQML as an agent communication language, Bradshaw, J. (Editor), *Software Agents*, MIT Press, Cambridge, 291-316
[11] Foundation for intelligent physical agents. FIPA ACL message representation, from *http://www.fipa.org/specs/fipa00070/XC00070G.html*, accessed on 17-05-2006
[12] Kavička, A.; Klima, V.; Adamko, N.; Fabian, P. (1996). System for combined simulations, *Proceedings of European Simulation Symposium & Exhibition - ESS '96*, SCS, 240-244
[13] Helsinger, A.; Thome, M.; Wright, T.: Cougaar: A Scalable, Distributed Multi-Agent Architecture, from *http://cougaar.org/docman/view.php/17/136/cougaar-bbn-0617-submitted.pdf*, accessed on 20-06-2005
[14] Henoch, J.; Ulrich, H.: HIDES: Towards an Agent-Based Simulator, from http://www.ifor.math.ethz.ch/publications/2000_towardsagentbasedsimulator.pdf, accessed on 11-03-2005
[15] Kavička, A.; Klima, V.; Niederkofler, A.; Zaťko, M. (1999). Simulation model of marshalling yard Linz Vbf (Austria), *Proceedings of The international workshop on Harbour, Maritime & Logistics Modelling and Simulation*, SCS, 317-320
[16] Kavička, A., Klima, V., Adamko, N. (2006). Analysis and optimization of railway nodes using simulation techniques, *Proceedings of COMPRAIL 2006 conference*, WIT-Press, 663-672