Multi-Agent System Model for Urban Traffic Simulation

Alexandru Cicortas, Norbert Somosi
Computer Science Department
Mathematics and Informatics Faculty
West University of Timisoara
Email: cico@info.uvt.ro, wolfsomosin@yahoo.com

Abstract: Multi-agent systems models are frequently in many complex real systems. They allow expressing event-based multi-agent, providing environment, and messaging, execution and sensor services.
The development of agent based applications is difficult suffering from insufficient standards and tools and on the other hand deployment issues are little researched and supported.
The cooperation in multi-agent systems is one of the most important features that can be understood in many ways. In the paper is suggested that the designers must make the cooperation possible using appropriate tools.
Urban traffic problems are very complex and highly interactive. The multi-agent systems (MAS) approach may however provide a new hopeful direction. A simulation system is needed to understand and explore the difficulties in a MAS-based traffic control.

Keywords: multi-agent, traffic modeling.

1 MAS Framework and Design

Real systems [3] require abilities for accurately measure the influence of different multi-agent co ordinations strategies in an unpredictable environment. A significant advantage multi-agent systems (MAS) have over traditional designs is the fact that the system is distributed. The decentralized, partially autonomous and redundant nature of such a system makes them less sensitive to certain classes of faults or attacks. This decentralization, however, also makes it difficult to analyze these systems.

Multi-agent systems (MAS) are composed of autonomous, interacting, more or less intelligent entities [1]. The agent metaphor has proven to be a promising choice for building complex and adaptive software applications, because it addresses key issues for making complexity manageable already at a conceptual level. Furthermore, agent technology can be seen as a natural successor of the

object-oriented paradigm and enriches the world of passive objects with the notion of autonomous actors. Therefore, one would suppose agent applications to be in widespread use in academic as well as in industrial projects. The contrary is the case. Even though many agent applications are developed in various domains, most of them are specialized solutions that are deployed in at most one setting.

The MASs are not yet effectively distributed. One reason for this is that the development of MAS is inherently difficult and error prone, because of several intricate issues. First, the development process for building agent applications is in most cases ad-hoc and not based on a generally accepted methodology, like for example the well-known Unified Process for UML in the object-oriented world. For agent systems, no such common ground exists due to different agent architectures and missing standards. In consequence, a methodology has to be chosen independently for each project among several alternatives. This choice is crucial for the project's success and is constrained by domain and implementation aspects. In addition, whatever methodology is selected, the tool support is always relatively poor and does not cover all phases of the development process.

Besides the methodology, the development of agent-based applications is difficult, because the software is distributed and dynamic in nature and demands various new skills and a new way of thinking from the developers. E.g. an object-oriented software engineer cannot easily change to the agent paradigm without considering ontology descriptions and studying the abstract speech-act based agent communication. Additionally, intelligent agents often use mentalist notions or employ rule-based approaches.

Another important reason for the scarce distribution of commercial off-the-shelf agent applications is that there is currently no support for the deployment of agent applications. In areas such as distributed object systems, systematical guidelines and mechanisms for all activities concerned with deployment issues have been developed.

These guidelines ensure that a properly developed distributed application can be packaged into a reusable, maintainable and configurable piece of software. However, although multi-agent systems composed of autonomous proactively (inter-)acting entities differ considerably from distributed object systems, the issue of appropriate deployment techniques for MAS is not yet very much researched.

One way is to specify agent applications at high-level using constraints to declare what system properties need to be fulfilled for the application to work properly. E.g. one could demand certain services and agent roles to be available, whereby the deployment environment has the task to interpret and supervise these constraints and has to start agent instances accordingly. As a first step towards this high-level deployment for MAS we propose a reference model for the launching of distributed multi-agent applications that are specified by declaring which and how many agent instances shall be instantiated in what order. As part of the reference model a generic meta-model for the specification of agent applications is

described, which consists of one layer for the definition of agent types and another one for the ordered composition of agent instances belonging to a certain application scenario. To underline the applicability of the proposed model a prototype implementation is presented.

The complexity of real systems imposes to develop powerful simulators for Multi-Agent Systems [3]. More of these are discrete, event-based providing environment, messaging and sensor services. The motivation for such simulators is based on the two simply but conflicting objectives:

- the ability to accurately measure the influence of different Multi-agent coordination strategies in an unpredictable environment;
- realistically modeling adaptive behavior in multi-agent systems within a static environment.

The MAS allow solving problems collaboratively by coordinating the knowledge, goals and plans of autonomous intelligent agents. It offers certain advantages of: faster response, increased flexibility, robustness, resource sharing, graceful degradation and better adaptability of integrating pre-existing and stand-alone systems. These concepts are the basis for our model.

1.1 Formal Framework

In [6] was proposed a conceptual framework for agents. A model for formalizing agents was also proposed. The formal framework must satisfy at least the following requirements:

- a formal framework must precisely and unambiguously provide meanings from common concepts and terms and also do so in a readable and understandable manner;
- the framework should be sufficiently well structured to provide a foundation for subsequent development of new and increasingly more refined concepts;
- alternative designs of particular models and systems should be able to be explicitly presented, compared and evaluated with the relative ease within the framework.

The Z language is recommended as an important tool [2].

As a view of the hierarchy of the entities, objects, agents and intelligent agents in the environment in Figure 1 is shown that every entity is a refinement of other entity i.e., an agent is a refinement of an object and so on. The entity is considered to be a collection of attributes. The environment can be defined as a collection of entities.

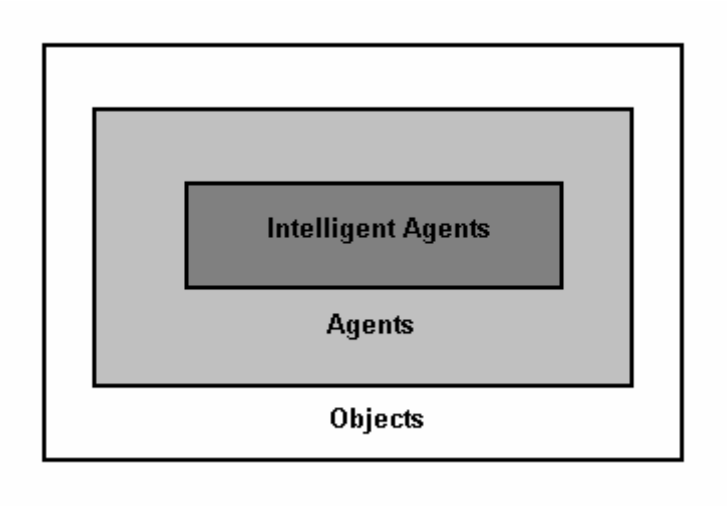

Figure 1
The Agent hierarchy

2 Deployment Considerations

First of all we most make distinction between the agent type and agent instance. To support the launching of distributed multi-agent applications several basic services can be identified. First of all, services are needed for starting and stopping agent. For invoking these services, at least the following information has to be supplied. The start of an agent instance should be based on a given agent type definition which has to contain a reference to the agent implementation (e.g. a Java class) and should declare the parameters that can be supplied to an agent of this type. To instantiate an agent, its type definition, the name for the agent instance (according to FIPA) and the assigned values for the parameters have to be supplied. To stop an agent instance only the agent identifier has to be known.

In order to launch distributed applications these basic services should be available remotely, therefore issues of security and accounting have to be considered. In addition, it is desirable that only minimal requirements are necessary for the manual configuration of network nodes, which could be achieved by code distribution and a service that allows remotely starting new agent platforms. The basic services additionally require a launch process management that has to make sure, that the correct agents and platforms are launched at the correct nodes at the correct times. One can imagine several ways to specify this. At the concrete level, it is possible to directly define the dependencies between agent instances. The

launch process management can then determine the launch order based on a topological sort of the dependency graph.

Constraints that are more abstract such as dependencies to specific services or roles can be employed to define application characteristics already at the type level (i.e. in agent type definition). In addition, application specific constraints and network load characteristics can be used to determine the allocation of agent instances to the available network nodes.

The monitoring and reconfiguration of the running agent instances should be supported. On the one hand, an administrator might want to observe a running application and manually add or remove agent instances or reallocate mobile agents to new network nodes. On the other hand a monitoring service should take care of the constraints and dependencies specified in the type and instance definitions and perform appropriate actions when the constraints get violated, e.g. by starting additional service agents to assure a given response time. By detecting failures and re-launching of agents, as well as detecting agents which are no longer needed by any application, the monitoring service can increase the robustness of agent applications.

The exact mechanisms available to the monitoring service to alter a running system have to be customized carefully for each application to reflect the varying degree of autonomy for each agent. To support monitoring and reconfiguration of agents and applications it is necessary to provide the responsible monitoring entities with relevant state information about the monitored entities and vice versa to be able to communicate back reconfiguration commands to the relevant agents. In addition, the reconfiguration of a larger application often requires a coordinated set of reconfigurations against the individual agents that constitute the system. Furthermore, reconfigurations need to assure that the system is in a consistent state after the reconfiguration has been performed.

Figure 2 depicts the structure of an agent type specification. An agent will be defined the agent type specification e.g. when it is requested to instantiate an agent of that type. The agent element captures important properties of an agent such as the agent's implementation class and the type, which identifies the required agent platform. The single-valued parameters and multi-valued parameter sets represent typed arguments that can be supplied when creating a new instance of the agent. Additionally, it is possible to specify one or more (for parameter sets) default values when no explicit value is provided for the creation of a specific agent instance. Both kinds of parameters can be further elaborated with additional constraint elements, used for restricting the set of allowed values for the parameter.

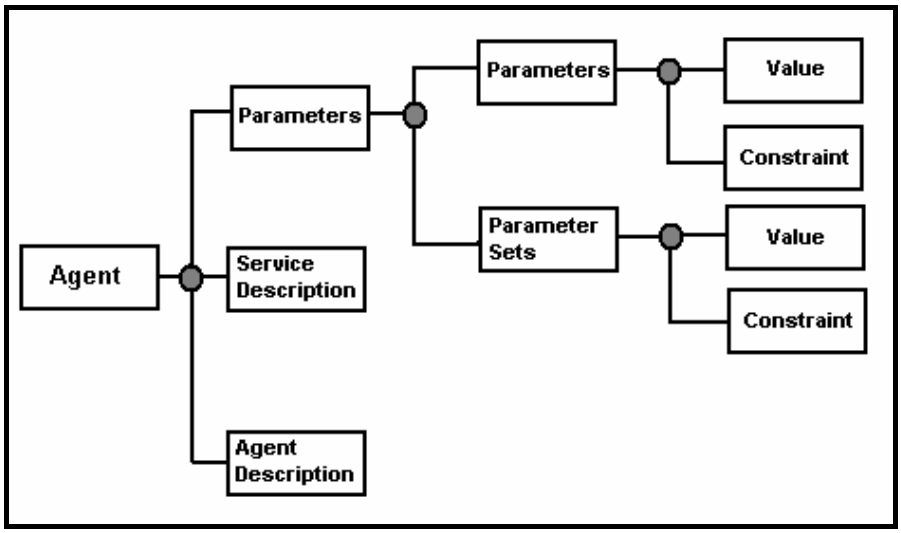

Figure 2
The agent meta-model

3 Cooperation, a Communication Model

Cooperation [4] can be thought a lot of meanings depending upon the system where it is used. So the cooperation can also be understood as working-together. In this case, it must allow to the agents of the system to have tools for exchange of information, sharing of knowledge, having a local way of doing, to capture the local distinctions that make a particular activity significant and meaningful to the other agents from the system. The cooperation and cooperative activities can be viewed as a situation in which ambiguity is accepted as a structural element of the interaction.

Cooperation is a way that allows to the agents that work together in order to improve some goals, their activities seems to be inseparable and the order of these activities is well defined. The agents must have the same language to express the terms of their know ledges and interactions. The know ledges are shared in appropriate ways such that the agents can be able to use them.

In the case of an urban traffic simulation system, the communication model that we used is based on the following main idea: we can consider that a map/network of routes is composed from several nodes and that these nodes are linked with route lines. Every node in this system represents an agent, while the links between them represents the communication channels between those agents. Every agent can communicate only with those agents, for which a communication channel to

the original agent exists. An example for this communication model is presented in the following figure:

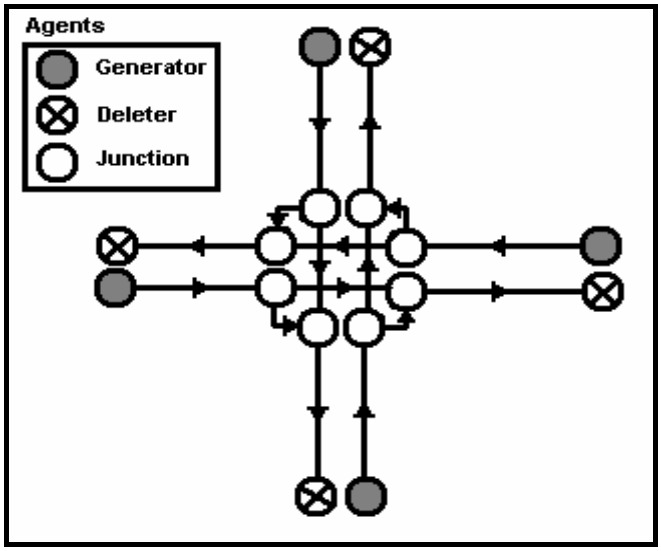

Figure 3
Communication between agents in a simple crossing

In the previous figure is presented a communication model between agents in a crossing with 4 in-entries and 4 out-entries. In this example we have three types of agents. The generator-agent will produce/generate messages (in our case these messages are cars/information about cars) and will send this messages to those agents, which are linked to it. An agent of type junction will send a received message to another agent, chosen randomly, which is linked to it. When a message comes to an agent of type deleter, it will be eliminated from the system and will not be considered any more.

In the case of the presented model, the agents communicate between each others by using special messages. These messages are in fact some objects, in our case some cars, while the links between the agents are some buffers (queues) in which the messages/cars are stored. A buffer or queue with N messages means, that on the route between two nodes corresponding to two agents, is situated a set of N cars.

There are two types of agents used in this communication model:

- **Normal Agents:** when an agent of this type receives a message from another agent, he will send this forward to the rest of the linked agents, without making any modifications on the content of the received message. For example if we have a very long route, which is made of a set o shorter routes (Figure 4) and these routes are linked with nodes, then such a node can be considered as an

normal agent. A car which comes to a such of node, will continue its way o the next shorter part of the route, which is linked to the node.

- **Intelligent Agents:** when an agent of this type receives a message from another agent, modifies the content of the received message, or it waits a time interval and only after then sends the message forward to the rest of the agents. For example a node of type junction can be considered as an intelligent agent, because it will send the received message forward to the agents of the system, only when its state is green. When its state is red, he will stay and keep the message until its state will change to green.

The working idea of this communication model is based on the followings: In the system there exist some agents who will produce/generate cars (Car-generators). These cars are generated randomly and stored in the buffers/queues which links the generators with the rest of the agents of the system. Every queue has a well specified length. When a car is inserted into a queue, it will be stored to the first position of the queue .At every moment of time this car will change its position, and will skip to the next free position in the queue, so the positions before the current position of the car will be cleaned for other cars. A car can change its position until he comes to the end of the queue that means that it is arrived to another nod/agent. Now, depending from this agent, the car will be deleted from the queue and inserted into another queue. This new queue is chosen by the agent, depending on the type of the agent and on the number of queues linked to this agent.

When an agent of type generator generates a message/car, the message will be sent from one agent to another in the system until it arrives an agent which can not send it forward to none of the other agents. At this moment the message/car will be eliminated from the system.

A simple example for demonstrating the functionality of this communication model is presented in Figure 4. In the presented example we have three agents: Generator, Agent1 and Agent2. Between Generator and Agent1 we have a link (a queue of length 3), while between Agent1 and Agent2 a queue of length 5.We can observe that every queue has a traversing order, or a specified order of inserting messages. This means, that when we have a link from agent 1 to agent 2, then only agent 1 can send messages to agent 2, contrarily is not true. We must have a link from agent 2 to agent 1, if we want a two-sided communication channel. Whit other words every route has a traversing order. A car, which is situated on a route can not move in both directions on the respective route. At the moment $t=1$ the Generator agent generates a message/car A .This message is inserted into the buffer, which links the Generator with Agent1. At the moment $t=2$ the message A skips its position to the next available in the buffer. At the moment $t=3$ the message A arrives the end of the queue, while the Generator generates another message B, which is inserted into the first position in the buffer. An important moment is at $t=4$.At this moment the message A comes to Agent1. This agent

deletes it from the buffer, and puts it into the next buffer between Agent1 and Agent2.When a message comes to Agent2, it will be deleted from the second buffer and destroyed.

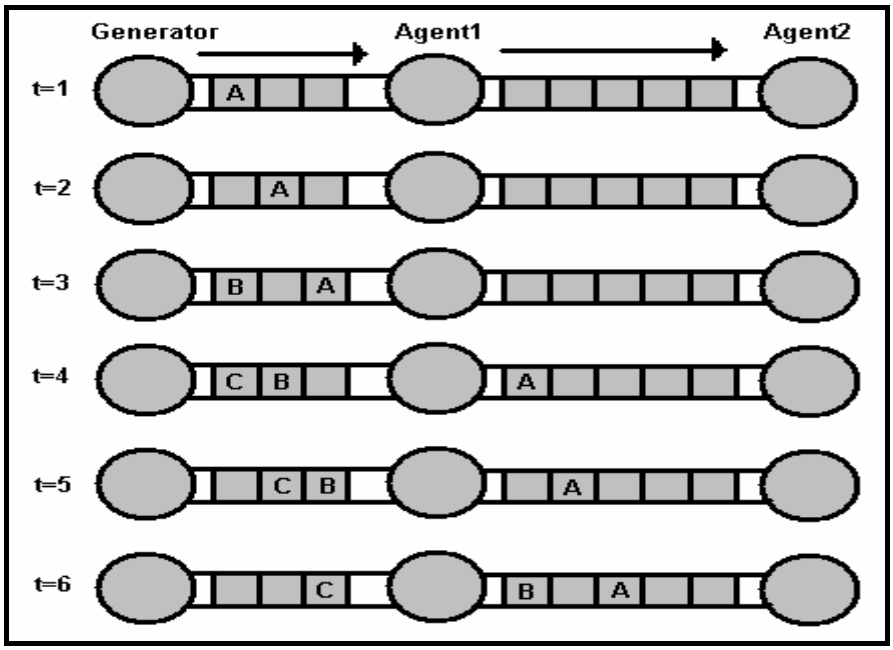

Figure 4
Communication model with buffers

In practice this is more complicated. Every car could have a different speed, so it can skip more then only one position in the buffer at a given moment, or an agent can be linked to more then one other agents and in this case we have more buffers than one, in which we can store a received message. The new buffer can be chosen randomly or using a specified method. For example if a car comes into a node of type junction, it will have more possibilities to continue its way ,depending on the crossing settings, from which the junction takes part.

An implementation of this model is presented in the next paragraph.

4 Traffic Modeling, Application: TraffSim V1.0

The evolution of systems concerning traffic control reflected the changes in optimization criteria that evolved gradually based on accumulated experience [5], [9], [10], [7]. Were tacked into account some performances indices like the length of queues on traffic lights, the number of vehicles stops during a journey. Another

systems use the on-line traffic flow data to modify the split/cycle/offset timing of the road junction lights in response to random fluctuations of traffic intensities.

In [10], shows how can provide current and predictive data on the basis of simulations and historical data. The on-line simulation provides current information, i.e., real-time data about the traffic state, like link travel times. The basic framework for an on-line simulation is introduced. It is based on the combination of an agent-based model, with real-world data stemming from inductive loops. The application of this framework to real road network is given. In order to provide predictive information, historical data is incorporated into the simulation. Historical data is analyzed and used to develop heuristics, the basis of a forecast. The impact of such predictive data on the current traffic patterns is analyzed.

[1] gives a general concept of the traffic information system that allows;

- constructing a simulation model in that the movements required by driving a car are in detail developed;
- implement of an adequate topology;
- visualization of the traffic state.

The application TraffSim V1.0, which implements the multi-agent model described in the previous paragraph, it was implemented in Visual C++ 6.0. The application has two main parts:

- a part allows the construction of a network of routes and traffic components like crossings, car generators etc. and the setting of the attributes of these components
- and a part of simulation, which allows the visualization in real time of the traffic activities on a constructed map/network of routes.

The construction of an arbitrary map can be made easily using the mouse. The user has the possibility to construct nodes, routes and crossings, and for each of these components to specify specific attributes. For example for a route the user can set the maximal speed which is allowed for cars on the respective route etc.

The agents are implemented using nodes. The application allows four types of nodes/four type of agents: junction, car-generator, connection to another map and normal node. For each of this node types we can specify different attributes. For example for a node of type car-generator we can specify how many cars we want to be generated in N time units, or for a node of type junction we can set the time interval between the junction states (i.e., green and red state) etc.

Figure 5
The GUI of the Application

The graphical user interface (GUI) of the application is presented on figure 5. The user can construct any type of traffic map, with an arbitrary configuration of nodes and routes, crossings with variable number of entries, signalized and unsignalized junctions and car generators which can generate cars using different methods of generating random numbers. The maps can be saved and reused. We can add new components to an existing map and we can change the attributes of the existing components.

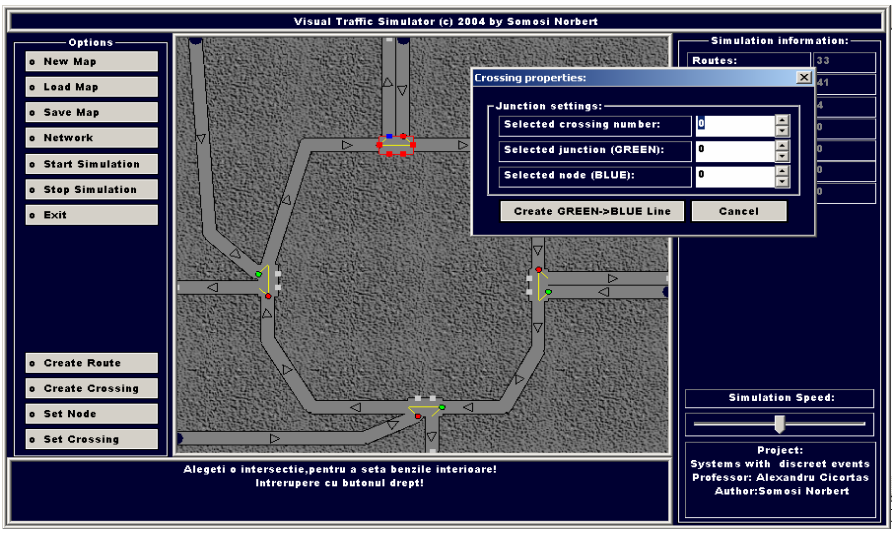

Figure 6
The GUI for modifying the map components

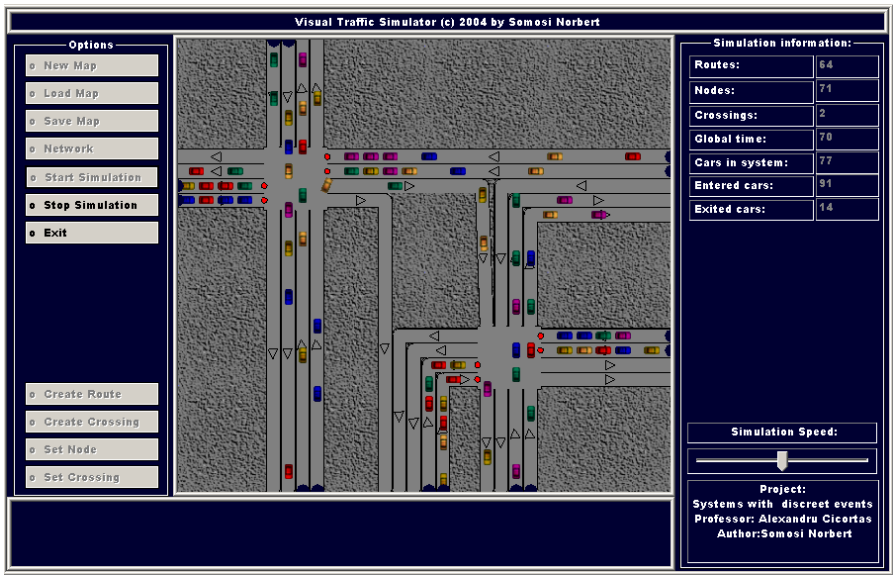

Figure 7
A Running Simulation

The user can change at every moment the properties of a given node. For example we can run a simulation on a map with a car generator, which generates cars using some specified rules and after then to rerun a new simulation on the same map with the same car generator, but this time using another generation rule. Te user can continue a stopped simulation, with new properties of the map components or restart the simulation with new values for the attributes of the map components. An example of simulation in presented in figure 7.

The construction of a map/network of routes has the following steps:

- specifying the positions of the nodes and the construction of the routes, which links these nodes;
- specifying the type of these nodes (For example: a map is only then valid, if it contains at least one car generator);
- specifying the crossings positions and setting these crossings. We can create crossings with variable number of in/out-entries, and with variable junctions. We can set the communication links between all the junctions, which take part of a particular crossing.

At the time of a simulation on the screen are visualized information about the number of cars in the system, the number of exited cars from the system, the number of entered cars into the system, the global simulation time and the number of map components. The user can also change at every instant the simulation

speed, to stop a running simulation, to modify the properties of some components and to restart the simulation with these new changes.

The application is very extensible, we can implement and add new types of components later and has a client-server distributed part, which can be used for maps with larger size. A larger map can be partitioned in multiple maps with smaller size and these smaller maps are distributed on different computers in a network. The communication between these maps is realized by some special type of agents, which communicates using the TCP/IP protocol between the different parts of the map, on distinct computers.

The application contains implementations of synchronization methods between the agents of the system. Every agent of type junction knows the state of the other junctions, which take part from the same crossing. So an agent of type junction, before changing its local state, looks at the states of the rest of the junctions and corresponding to their states will change its own state. So we can avoid the eventually collisions between cars in the inside of a crossing. Every agent will allow for a car to change its route, only then when on the corresponding route there is a free place for the car. The car-generators will not stop generating cars when on the screen a route is full.

Conclusions

In this paper we discussed some basic concepts about the problematic of multi-agent systems and we presented a simple multi-agent communication model for a particular case: traffic simulation. The presented model was based on a graph of nodes with unidirectional links between these nodes, while the communication model between these nodes was based on the using of multiple message buffers.

In the second part of the paper we presented shortly an implementation of the model (in the project application TraffSim V1.0).The application allows the construction of maps with four types of agents, presented in this paper. We can now add new type of components (ex. Train lines etc). The distributed part of the application is also implemented, we work now on the implementation of the agents, who will realize the communication between the distributed map parts of a larger map.

References

[1] Braubach, L., Pokahr, A., Krempels, K. H., Winfried Lamersdorf, W., Deployment of Distributed Multi-Agent Systems, Fifth International Workshop on Engineering Societies in the Agents World, Eds., Marie-Pierre Gleizes and Andrea Omicini and Franco Zambonelli, 2004

[2] Bowen, J., Formal Specification and Documentation using Z: A case Study Approach, International Thomson Computer Press, London, 1996

[3] Horling, B., Lesser, V., Vincent, R., Multi-Agent System Simulation Framework, 16th IMACS World Congress 2000 on Scientific Computation, Applied Mathematics and Simulation, August 2000

[4] Introna .L. D., Cooperation, Coordination and Interpretation in Virtual Environments: Some Thoughts on Working Together, Cognition, Technology &Work Vol. 3, 2003, pp. 101-110

[5] Li, M., Chong, K. W., Chan, S., Hallan, J., Agent-Oriented Urban traffic Simulation, The 1st International Conference on Industrial Engineering Application and Practice, 1996

[6] Luck M., d'Inverno, M., A Conceptual Framework for Agent Definition and Development, The Computer Journal, 44(1), 2001, pp. 1-20

[7] Mazur, F., Chrobok, R., Hafstein, S. F., Pottmeier, A. and Schreckenberg, M., Future of Traffic Information - Online-Simulation of a Large Scale Freeway Network, IADIS International Conference WWW/Internet 2004, Procs Vol. 1, Madrid, Spain, October 2004, pp. 665-672

[8] Moldt, D., Wienberg, F., Multi-Agent-Systems based on Coloured Petri Nets, 18th International Conference on Application and Theory of Petri Nets, Toulouse, June 23-27, 1997

[9] Peychev, E. T., Bargiela, A., Parallel Simulation of City Traffic Flows using "PADSIM", 2001

[10] Wahle, J., Schreckenberg, M., A Multi-Agent System for On-Line Simulations based on Real-World Traffic Data, in: Proc. of the Haiwai International Conference on System Science (HICSS), (IEEE Computer Society, 2001)