Proceedings of the 1st Human-Centered Transportation Simulation Conference, The University of Iowa, Iowa City, Iowa, November 4-7, 2001 (ISSN 1538-3288).

A MULTI-AGENT, MICROSCOPIC TRAFFIC SIMULATION ARCHITECTURE INCORPORATING ENTITIES WITH ADAPTIVE BEHAVIORS

Nikos Manouselis[1], Pythagoras Karampiperis[2], Elias Kosmatopoulos[3]

[1]Centre for Research and Technology- Hellas
Informatics and Telematics Institute
Advanced e-Services for the Knowledge Society Research Unit
Kyvernidou 1, Thessaloniki, 54639, Greece
e-mail: nikosm@iti.gr

[2,3]Dynamic Systems and Simulation Laboratory
Dept. of Production and Management Engineering
Technical University of Crete, Chania,
73100, Greece
Tel: +3082137237
e-mail: pythk@dssl.tuc.gr
e-mail: kosmatop@dssl.tuc.gr

Submitted August 07 2001
Revised October 21 2001

ABSTRACT

In this paper we present an agent-based architecture for microscopic traffic simulations. Agents representing vehicles utilize database modules containing microscopic models and adaptive algorithms, in order to achieve precise simulation of real-life traffic entities. The analysis and design of the simulation system is made using the Gaia methodology, a formal way of going from a statement of requirements to an agent-based, detailed design that can be directly implemented.

INTRODUCTION

Research indicates that a critical feature of next generation scenario control systems for driving simulation will be the capability to generate and control natural-looking ambient traffic, while still maintaining the ability to meet experimenter's situational needs. At the lowest level, ambient traffic requirements may be specified, by associating traffic property goals with individual road sections in the road network. Specifiable traffic properties include flow, density, average speed, distribution of vehicle types, and so on. Various queuing models are used for the motion of the vehicles along the road sections. When a vehicle enters a section, its behavior model has to be calculated according to its velocity and type, and length, capacity and number of cars that are moving on that road section.

Our approach is based on various microscopic traffic interaction models, simulated as a distributed set of autonomous agents, each of which controls its own vehicle and responds to the actions of the other agents on the specific road section. The method we propose is based on improvising driver-vehicle entities' behavior using various adaptive algorithms, in order to "fit" microscopic models on each vehicles behavior, thus being able to perfectly simulate them in future generated scenarios. This is achieved by introducing a "learning" stage where, using real traffic data, system agents monitor vehicles entering the road section under surveillance and adopt a microscopic traffic model to simulate each entity's behavior. Furthermore, each agent uses adaptive learning techniques to adjust the model's parameters in order for the simulated behavior to be as realistic as possible.

An interesting agent-based approach of autonomous vehicles defining their position in a traffic network is described in (1). Trafficopter is a multi-agent system that collects and propagates traffic information in an urban setting using distributed methods. Agents into the vehicles themselves collect and propagate traffic related information in a decentralized, self-organizing fashion with no single point of failure. The ideas in this system are the use of the vehicles/agents themselves as a way of collecting traffic data and the way those data are distributed to the interested vehicles. The objective is to have all the vehicle's agents know what the traffic conditions are in a metropolitan area by using data collected by other vehicles' agents. Each vehicle agent broadcasts a message object with its local information (position, speed, direction).

Although Trafficopter was intended for application in the real traffic field, it shows the feasibility and potential of using an agent-based approach to the analysis and design of traffic systems. This fact is moreover outlined by another system applied in on-line simulations and predictions at the urban network of Duisburg and the freeway network of North Rhine-Westphalia, Germany. The OLSIM system (4) uses real-world traffic data and an agent-based traffic flow model, in order to derive the traffic state of a complete network on-line. Typically, locally fixed detectors like inductive loops or cameras are used in collecting traffic data. Nevertheless, a lot of road networks are not adequately equipped with detection devices to gather information about the present traffic state in the whole network. A possible way to derive information for those regions that are not covered by measurements is to combine local traffic counts with the network structure (i.e. type of roads, priority regulations at the nodes or on- and 0ff ramps) under consideration of realistic traffic flow dynamics. This is the basic idea of this on-line simulation: local traffic counts serve as input for traffic flow simulations to provide network-wide information.

In this paper, we will describe a similar approach to a different traffic problem: the creation and control of natural-looking ambient traffic with microscopic models. To be more specific, the Section that follows describes the basic ideas incorporated into the system, whereas the next Section introduces and applies the Gaia methodology (2) for agent-oriented analysis and design. Finally, some basic conclusions and remarks are stated.

BASIC IDEAS

We introduce an agent-based architecture with agents controlling each vehicle, and each one of the vehicle agents communicating with a coordinator agent. Since the primary field of application of our approach is that of traffic simulators, we define this coordinator agent as the information center responsible of updating the simulation data and providing the information needed by the vehicle agents. This general architecture is depicted in Figure 1.

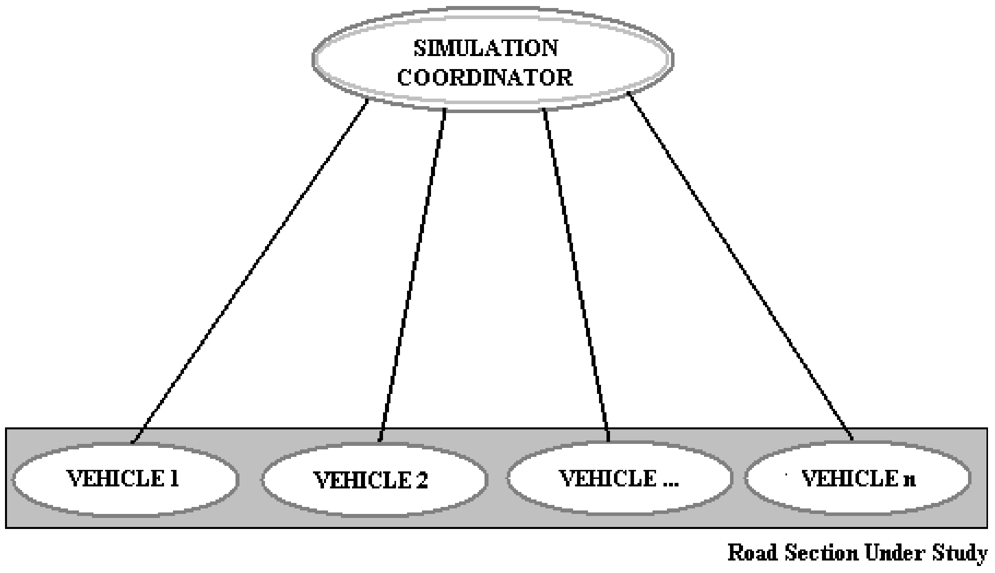

FIGURE 1 The general agent-based simulator architecture

One basic idea is that each vehicle agent is equipped with a sufficient number of microscopic models that are used in modeling the vehicle’s behavior. It was our aim, the agents to be able to choose which model is appropriate for the simulated vehicle, so we created a models’ hierarchy. This hierarchy contains several well-known and widely used microscopic models that have proven very effective in simulation programs. The basic relations used are the following:

$\ddot{x}_{n+1}(t+T) = \lambda_+[\dot{x}_n(t) - \dot{x}_{n+1}(t)]$, for the acceleration phase,
$\ddot{x}_{n+1}(t+T) = \lambda_-[\dot{x}_n(t) - \dot{x}_{n+1}(t)]$, for the deceleration phase.

The relations defining the factors λ_+, λ_- are the following:

$$\lambda_+ = a_+ \frac{\dot{x}_{n+1}(t+T)^m}{[x_n(t) - x_{n+1}(t)]^l}$$
$$\lambda_- = a_- \frac{\dot{x}_{n+1}(t+T)^m}{[x_n(t) - x_{n+1}(t)]^l}$$

where a_+, a_-, *l*, *m* are the free parameters. If we assume that the acceleration and deceleration factors are identical ($\lambda = \lambda_+ = \lambda_-$), the model becomes:

$$\ddot{x}_{n+1}(t+T) = \lambda[\dot{x}_n(t) - \dot{x}_{n+1}(t)]$$

where

$$\lambda = a_0 \frac{\dot{x}_{n+1}(t+T)^m}{[x_n(t) - x_{n+1}(t)]^l}.$$

This form, by properly adjusting the free parameters, can be degraded into well-known models found in literature (3) as the following three:

$\ddot{x}_{n+1}(t+T) = a_0[\dot{x}_n(t) - \dot{x}_{n+1}(t)]$ (Simple linear),

$\ddot{x}_{n+1}(t+T) = a_0 \dfrac{\dot{x}_n(t) - \dot{x}_{n+1}(t)}{x_n(t) - x_{n+1}(t)}$ (Gazis *et al*. 1959),

$\ddot{x}_{n+1}(t+T) = a_0 \dfrac{\dot{x}_n(t) - \dot{x}_{n+1}(t)}{[x_n(t) - x_{n+1}(t)]^2}$ (Pipes 1953).

The selection of the appropriate microscopic model depends on the field measurements acquired through the learning phase. The basic adaptation algorithm (Figure 2 – *in Page 8 temporarily*) includes definition of the position function in terms of coefficients φ_i and, regarding the number of coefficients that can be neglected, selection of the appropriate model.

When the microscopic model is selected, we wish to regularize the free parameters in order to fit the model to the specific driver. This can be accomplished using adaptive algorithms. This procedure is called Parameter Identification (PI). PI involves the determination of the parameters of certain mathematical models that describe the response of static or dynamical systems. PI is a fundamental block of adaptive control and is also crucial for adaptive filtering, prediction, diagnostics and other applications. There are two classes of PI algorithms: on-line and off-line. In the off-line case the response data are available as a complete block of information and they are processed simultaneously to generate the parameter estimates with no strict time limits on the process of analysis. In our case (the on-line case) the data are processed sequentially during each sampling period and the parameter estimates are recursively updated within the time limit imposed by the sampling period.

Therefore, if we consider the parametric model $Z(t) = \theta^{*T}\phi(t)$, where $\theta^* \in \Re^n$ is the unknown constant vector and $Z \in \Re, \phi \in \Re^n$ are known sequences and $t = 0,1,2,\ldots$, the objective is to process the sequences $Z(t), \phi(t)$ for $t = 0,1,2,\ldots$ in order to generate a recursive estimate for θ^*. Let $\theta(t-1)$ be the estimate of θ^* at time $t-1$. Then the estimate $\theta(t)$ of θ^* at time t may generated as $\theta(t) = \theta(t-1) + H(t)e(t)$, where $H(t)$ is some gain matrix that depends on $\phi(t)$ and $e(t)$ is an error signal that represents a measure of how far $\theta(t-1)$ is from θ^*. Different choices for $H(t)$ and $e(t)$ lead to a wide class of PI algorithms with sometimes different convergence properties. Our will is to include as more adaptive algorithms as possible into the vehicle agents database and let the agent select the appropriate one, according to time and computational resources constraints.

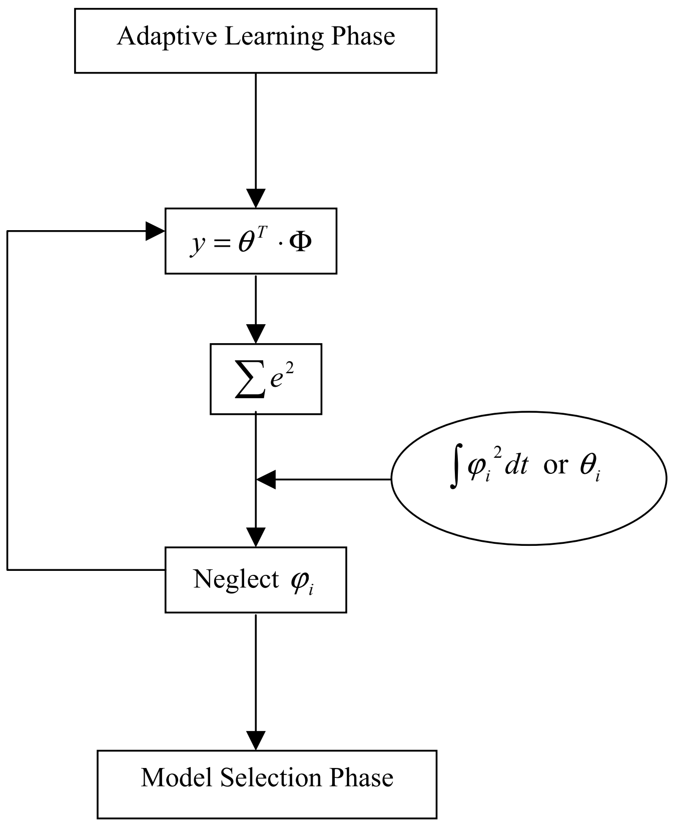

FIGURE 2 The basic adaptation algorithm.

The basic functionalities of the adaptation algorithm are the following:

- The position function is expressed in terms of coefficients φ_i, as $y = \theta^{\mathrm{T}}\Phi = \theta^{\mathrm{T}}\sum \phi_i$.
- The error function is minimized, in accordance to the real-world measurements of the learning phase.
- The less important coefficients are neglected, so that the function is expressed in the simplest possible form.
- Depending on the final form of the position function, the appropriate microscopic model is selected and its free parameters are determined.

In the previous paragraphs we outlined the basic ideas behind our agent-based architecture with adaptive entities' behavior. Let us now use a formal way to analyze and design our system, in order for the architecture to be implemented.

AGENT ORIENTED ANALYSIS AND DESIGN

In order to deploy agent-based architectures that can be used to develop complex distributed systems like traffic simulators, it is a necessity to follow system analysis and design approaches that can adequately capture the agent's flexible, autonomous problem-solving behavior, the richness of an agent's interactions and the complexity of an agent-based system's organizational structures. Existing software development techniques (for example object-oriented analysis and design) have proven simply unsuitable for this task. For these reason, we briefly introduce the basic ideas of a methodology called Gaia, which has been specifically tailored to the analysis and design of agent-based systems.

The Gaia Methodology

Gaia (2) is intended to allow an analyst to go systematically from a statement of requirements to a design that is sufficiently detailed that it can be implemented directly. In applying Gaia, the analyst moves from abstract to increasingly concrete concepts. Each successive move introduces greater implementation bias, and shrinks the space of possible systems that could be implemented to satisfy the original requirements statement. Analysis and design can be thought of as a process of developing increasingly detailed models of the system to be constructed. Gaia provides an agent-specific set of concepts through which a software engineer can understand and model a complex system. In particular, Gaia encourages a developer to think of building agent-based systems as a process of organizational design. The main concepts of Gaia can be divided into two categories: abstract and concrete. Abstract entities are those used during analysis to conceptualize the system, but which do not necessarily have any direct realization within the system. Concrete entities, in contrast, are used within the design process, and will typically have direct counterparts in the run-time system.

The objective of the analysis stage is to develop an understanding of the system and its structure (without reference to any implementation detail). In our case, this understanding is captured in the system’s organization. We view an organization as a collection of roles, that stand in certain relationships to one another, and that take part in systematic, institutionalized patterns of interactions with other roles.

A role is defined by four attributes: responsibilities, permissions, activities, and protocols. Responsibilities determine functionality and, as such, are perhaps the key attribute associated with a role. An example responsibility associated with the role of company president might be calling the shareholders meeting every year. Responsibilities are divided into two types: liveness properties and safety properties. Liveness properties intuitively state that “something good happens”. They describe those states of affairs that an agent must bring about, given certain environmental conditions. In contrast, safety properties are invariants. Intuitively, a safety property states that “nothing bad happens”.

In order to realize responsibilities, a role has a set of permissions. Permissions are the “rights” associated with a role. The permissions of a role thus identify the resources that are available to that role in order to realize its responsibilities. In the kinds of system that we have typically modeled, permissions tend to be information resources. The activities of a role are computations associated with the role that may be carried out by the agent without interacting with other agents. Finally, a role is also identified with a number of protocols, which define the way that it can interact with other roles. Thus, the organization model in Gaia is comprised of two further models: the roles model and the interaction model.

The aim of the Gaia’s design process is to transform the analysis models into a sufficiently low level of abstraction that traditional design techniques (including object-oriented techniques) may be applied in order to implement agents. To put it another way, Gaia is concerned with how a society of agents cooperate to realize the system-level goals, and what is required of each individual agent in order to do this. Actually how an agent realizes its services is beyond the scope of Gaia, and depends on the particular application domain.

The Gaia design process involves generating three models. The agent model identifies the agent types that will make up the system, and the agent instances that will be instantiated from these types. The services model identifies the main services that are required to realize the agent’s role. Finally, the acquaintance model documents the lines of communication between the different agents.

Analysis

The analysis stage of Gaia can be summarized as:

1. Identify the roles in the system. In this step we form a prototypical roles model; that is a list of the key roles that occur in the system, each with an informal, unelaborated description:

- Coordinator: collects data from all vehicles and informs the simulation system, defines id's and queue positions for the car-following models, provides certain user-defined parameters of the models when appropriate.
- Measurement Control: collects state variables in a form of a function with several coefficients.
- Vehicle control: utilizes model results in order to determine next positions.
- Parameter estimator: after the model is selected defines the values of the model parameters.
- Model selector: receives variables, determines which coefficients of the state variables will be used and which will be discarded and defines the model to be used.

2. For each role, identify and document the associated protocols. Protocols are the patterns of interaction that occur in the system between the various roles. The output of this step is an interaction model, which captures the recurring patterns of inter-role interaction (Figure 3).
3. Using the protocol model as a basis, elaborate the roles model. The output of this step is a fully elaborated roles model, which documents the key roles occurring in the system, their permissions and responsibilities, together with the protocols and activities in which they participate. Fully detailed and beyond the scope of this paper, which is to outline the architectural structure of the proposed system.

There are basic functionalities in the multi-agent system, represented in the interactions model of Figure 3. For example, throughout the learning phase, the position regarding to a leader vehicle at each time step is measured (Measure), in order for the microscopic model that best resembles the real position function is selected (SelectModel) and its free parameters are determined (EstimatedParam). As the vehicle is simulated, it engages the VehicleEntrance protocol in order to enter the road section under study. It is assigned a leader vehicle to follow and it requires this vehicle's position (RequirePos) at each time step, so that its own position is computed (DeterminePos).

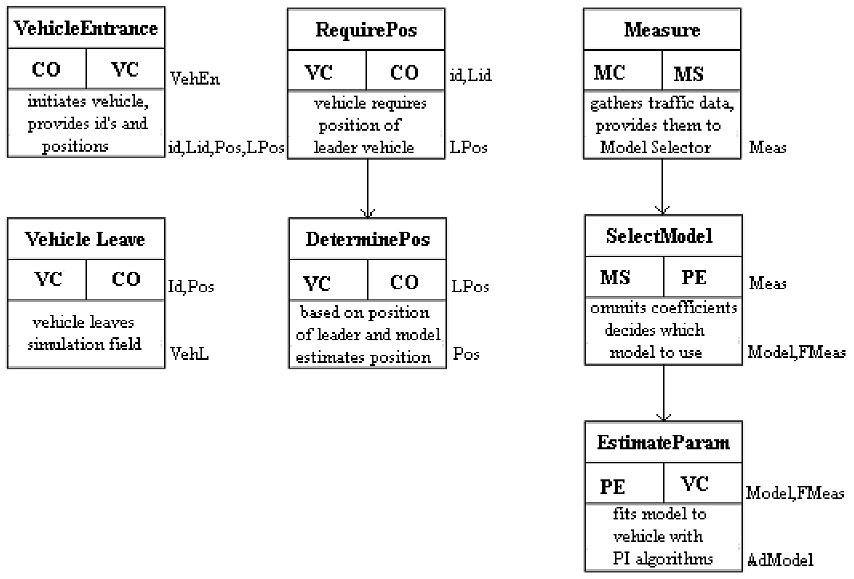
VehicleEntrance
CO
VC
initiates vehicle, provides id's and positions
VehEn
id,Lid,Pos,LPos
RequirePos
VC
CO
vehicle requires position of leader vehicle
id,Lid
LPos
Measure
MC
MS
gathers traffic data, provides them to Model Selector
Meas
Vehicle Leave
VC
CO
vehicle leaves simulation field
Id,Pos
VehL
DeterminePos
VC
CO
based on position of leader and model estimates position
LPos
Pos
SelectModel
MS
PE
ommits coefficients decides which model to use
Meas
Model,FMeas
EstimateParam
PE
VC
fits model to vehicle with PI algorithms
Model,FMeas
AdModel

FIGURE 3 The interactions model.

Design

The Gala design stage can be summarized as:

1. Create an agent model that includes aggregating roles into agent types, and refining them in order to form an agent type hierarchy; moreover, the agent model documents the instances of each agent type using instance annotations (Figure 4).

2. Develop a services model, by examining activities, protocols, and safety and liveness properties of roles (based on the elaborated roles model).

3. Develop an acquaintance model from the interaction model and agent model (Figure 4).

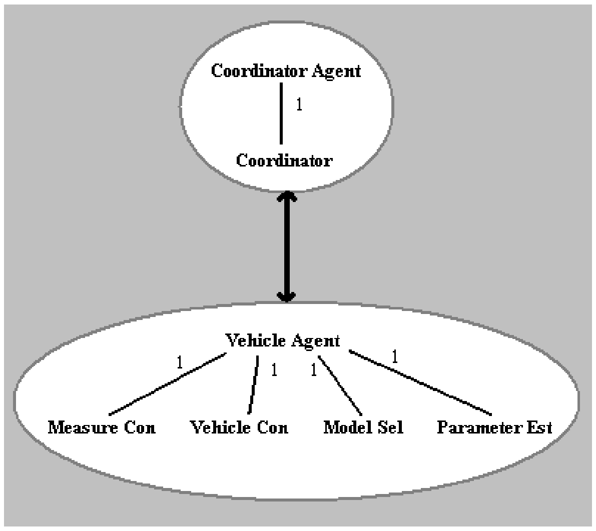

FIGURE 4 The Acquaintance Model incorporating the Agent Model

Simulation results

The proposed architecture has been implemented and tested in an experimental testbed. Although this architecture is part of still ongoing research, the first simulation results indicate that when different microscopic models are used for each vehicle's behavior, more realistic simulation results are provided. There is still much work to be carried out though, in order for the model architecture to be validated.

DISCUSSION AND CONCLUSIONS

As Gaia was applied, the following characteristics had to stand for the overall system (2):

- Agents had to be coarse-grained computational systems, each making use of significant computational resources.
- It was assumed that the goal was to obtain a system that maximizes some global quality measure, but which may be sub-optimal from the point of view of the system components.
- Agents had to be heterogeneous, as different agents may be implemented using different programming languages, architectures, and techniques. We made no assumptions about the delivery platform.
- The organization structure of the system had to be static, in that inter-agent relationships would not change at run-time.

- The abilities of agents and the services they provide had to be static, in that they would not change at run-time.

- The overall system should contain a comparatively small number of different agent types (less than 100).

All of the above requirements were satisfied in the case of our agent-based microscopic simulation architecture.

Despite the preliminary stage of our research, there are several interesting conclusions made. The ideas of incorporating adaptivity modules into the agent-based infrastructure of a simulation system, has proven to be valuable and encouraging for future applications. Moreover, engaging the Gaia methodology for agent-oriented analysis and design of the system, provided an extremely useful tool for creating detailed schemas that can be directly implemented, in a functional and feasible way.

REFERENCES

1. A. Moukas, K. Chandrinos, and P. Maes. Trafficopter: A distributed Collection System for Traffic Information. In: Klusch M., Weiss G. (Eds.). *Cooperative Information Agents II, LNAI 1435*. Springer, 1998.
2. M. Wooldridge, N.R. Jennings, and D. Kinny. The Gaia Methodology for Agent-Oriented Analysis and Design. *Journal of Autonomous Agents and Multi-Agent Systems*, 2000.
3. J.F. Gabard, Microscopic Models. In: Papageorgiou, M. (Ed.). *Concise Encyclopedia of Traffic and Transportation Systems*. Oxford, Pergamon Press, 1991.
4. J. Wahle, O. Annen, C. Schuster, L. Neubert, and M. Schreckenberg. A dynamic route guidance system based on real traffic data. *European Journal of Operational Research,* 2000.