Multi-Agent Approach to Modeling and Simulation of Urban Transportation Systems

P. Gruer, V. Hilaire, A. Koukam

Université de Technologie de Belfort Montbéliard.
Laboratoire Systèmes et Transports
Belfort Technopôle
90000 Belfort, France
E-mail: pablo.gruer@utbm.fr

Abstract

This work presents an approach to modelling and simulation of urban public transport network. The main aspects of the approach are agent orientation, combined use of formal languages Object-Z and Statechart.
We are convinced that the adoption of formal languages currently encoutered in the software engineering field, facilitates the construction of the simulation model and introduces new benefits, such as reuse of model components. Additionally, formal languages allow to verify some qualitative properties related to the correctness of the model before facing the simulation phase. Steel, the formal approach allows for rapid construction of and execution of the simulation itself, by using industrial working environments such as Statemate.

1 Introduction

Urban public transports have received an increasing amount of attention as they often thought of as a rational alternative to the intensive use of existing urban networks by private cars. The expansion of this transport mode implies the infrastructure improvements, the master of the urban network traffic and other operational problems. Modeling and simulation constitute an approach which can be valuable in tackling some of these problems. This paper presents a part of a research project, namely Urban Bus Networks that intends to design a framework based on generic components for modeling and simulation of public transport network dedicated to the bus operations in an urban area. Toward this end, we propose a modular approach based on agent oriented modeling and simulation. The choice of agent technologies derives basically from two observations. First, an urban public transport network is a complex system which involves a set of distributed and interacting entities. Secondly, the global system behavior is made of several emergent phenomena that result from the behavior of the individual entities and their interactions. MAS have already been used in the transportation domain. Indeed, the agent abstraction allows the conception of microscopic models which generates, at least conceptually, arbitrarily realistic simulation of transports phenomenon [5]. For example, MAS have been applied for planning, optimising and monitoring road haulage in the TELETRUCK sytem [4]. The AGENDA system [3] provides cooperation methods between agents in order to solve scheduling problems. Other projects aim to simulate traffic [5, 1, 2]. The perspective is to help the decision process. Many problems, like computing tractability or models accuracy, arise when one try to simulate traffic with the multi-agent paradigm [5] and different solutions for these problems have been found. Despite the existence of several successful simulation environment, the problem of reusing all or part of their underlying models still remain complex and difficult. We belive, and the xeprerience bears this out, that a formal modeling is fundamental to handle the complexity related to building and reusing such models. The purpose of this paper is to present a formal approach to multi-agent systems that fits in with simulation of urban public transport networks. The system is viewed as an organization which federates a set of interacting agents. Each agent defines an abstract characterization of the behavior of an active entity in the organization.

2 Agents for transportation simulation

2.1 Multi-Formalism Based Specification Language

Many specification formalisms can be used to specify entire system but few, if any, are particularly suited to modeling all aspects of such systems. For large or complex systems, like MAS, the specification may use more than one formalism or extend one formalism.

Proceedings of the 2001 IEEE Systems, Man, and Cybernetics Conference

The multi-formalism approaches [7, 6] compose two or more formalisms in order to specify more easily and naturally than with a single formalism. Indeed, the multi-formalism approach deals with complexity by applying formalisms to problem aspects for which they are best suited and to prove properties with proofs rules and transformation techniques available in a specific formalism.

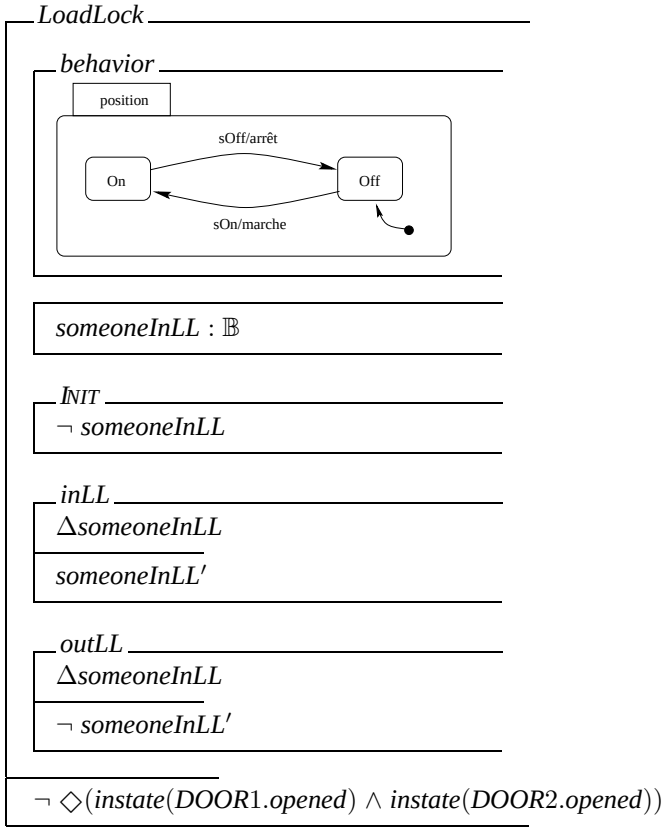
Our choice is to use Object-Z to specify the transformational aspects and statecharts to specify the reactive aspects. Object-Z extends Z with object-oriented specification support. The basic construct is the class which encapsulates state schema with all the operation schemas which may affect its variables.

Statecharts extend finite state automata with constructs for specifying parallelism, nested states and broadcast communication for events. Both language have constructs which enable refinement of specification. Moreover, statecharts have an operational semantic which allows the execution of a specification.

Our method for composition of Object-Z and statecharts relies on the meta-method of Paige [6]. The main step of this method is the definition of an heterogeneous basis which is a set of notations, translations and formalizations that provides a formal semantics to multi-formalism specifications. In our case the main concept of the heterogeneous basis is the integration of statecharts in Object-Z classes. We have extended the expressive capabilities of each formalism by features available in the other. The role of the heterogeneous basis is to provide formal means of expression without translating a formalism in the other. In other words the heterogeneous basis furnishes mean of communication between partial specifications written in either Object-Z or statecharts.

The class describes the attributes and operations of the objects. This description is based upon set theory and first order predicates logic. The statechart describes the possible states of the object and events which may change these states. A statechart included in an Object-Z class can use attributes and operations of the class. The sharing mechanism used is based on name identity. Moreover, we introduce basic types $[Event, Action, Attribute]$. *Event* is the set of events which trigger transitions in statecharts. *Action* is the set of statecharts actions and Object-Z classes operations. *Attribute* is the set of objects attributes. These types also belong to the heterogeneous basis.

The *LoadLock* class illustrates the integration of the two formalisms. It specifies a *LoadLock* composed of two doors which states evolve concurrently. Parallelism between the two doors is expressed by the dashed line between *DOOR*1 and *DOOR*2. The first door reacts to *activate*1 and *deactivate*1 events. When someone enter the *LoadLock* he first activate the first door enter the *LoadLock* and deactivate the first door. The transition triggered by *deactivate*1 event execute the *inLL* operation which sets the *someoneInLL* boolean to true. Someone which is between the first and the second door can activate the second door so as to open it. The temporal invariant at the end of the class specifies that the statechart must not be in $DOOR1.opened$ and $DOOR2.opened$ states simultaneously. This invariant uses the predicate $instate(S)$ which is true whenever *S* state is active.



The notation for attribute modification consists of the modified attributes which belongs to the Δ-list. In any operation sub-schema, attributes before their modification are noted by their names and attributes after the operation are suffixed by '.

The result of the composition of Object-Z and statecharts seems particularly suited in order to specify MAS. Indeed, each formalism have constructs which enable complex structure specification. Moreover, aspects such as reactivity and concurrency can be easily dealt with. In fact, available constructs enable natural specification of "low" level aspects inherent to MAS. Higher level aspects like coordination are expressed by roles, interactions and organizations classes which we present in the following section.

2.2 Modeling of timed actions

Statecharts provide means of dealing with qualitative analysis of discrete event systems. They use a logical model of time based on a partial order on event occurrences, subject to restrictions such as causality. However, for quantitative analysis, it is necessary and useful to introduce time delays associated with actions to be performed by the system. Additionally, we are interested in specifying the delays randomly to model the system under uncertainty. Models such as those proposed by stochastic Petri nets [1], introduce timed transitions to represent the delay associated with actions. A timed transition has an exponentially distributed firing time that expresses the delay from the enabling to the firing of the transition. This solution is not feasible here as it is in conflict with the formal semantic of statecharts [6]. Indeed, a transition in statecharts should be instantaneous. We overcome such a problem by associating the time delay to states or, in other words, we represent actions as states. In addition, we define modeling capabilities to represent two optional important features: first, associating a precondition to the modeled action and second, defining the effect of the action on the value of the state variables of the model.

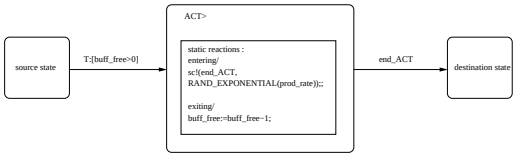


Figure 1. A statechart model for a timed action.

We illustrate the proposed approach to the modeling of timed actions through an example: consider an action ACT that consists of producing data items and storing them in a data buffer. A new data item can be produced only if there is a free location in the buffer. Producing a new data item consumes an amount of time that is uncertain. Nevertheless, the production rate prodrate , i.e., the average number of data items produced per unit of time by action ACT, is known. Figure 2 illustrates the proposed construct, based on static reactions associated with state ACT, which represents the action execution. In statechart notation, the symbol "¿" following the state identifier indicates that static reactions have been associated with the state. We introduce state variable bufffree to represent the number of free locations in the buffer. Transition T activates state ACT, if precondition bufffree 0 is verified. A static reaction, executed upon entering ACT, consists in delaying an occurrence of event endACT. To determine the delay amount, the predefined randexponential function is used. Consequently, state ACT remains active for a duration determined randomly, with an exponential law. The effect of the action ACT on state variable bufffree is represented by the static reaction executed upon exiting the state: as a new data item has been stored in the buffer, the number of free locations decreases.

2.3 Structural conflict

Structural conflicts represent non deterministic situations where many mutually exclusive actions can be started. One of those actions is randomly selected to be executed. The modeling formalism should offer means to assign probabilities to each one of the possible evolutions. Those modeling constructs are frequently called n-way random switches, with n equal to the number of mutually exclusive issues. Generally, an n-way random switch is considered to be instantaneous, as the probabilistic choice is supposed to take no time.

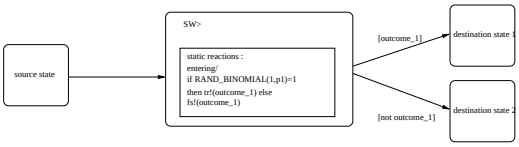


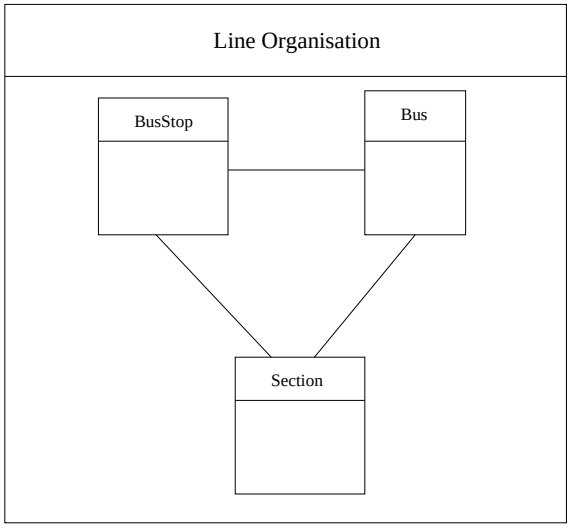
Figure 2. A statechart model for a two-way random switch.

Statecharts can resolve a structural conflict with two possible issues, (i.e., a two-way random switch) in the way illustrated by figure 3. Upon entering state SW, a static reaction is executed. It consists in randomly assigning value true or false to a boolean variable outcome1. The standard function rand-binomial with parameters n=1 and p=p1 is used to this end. The value of boolean variable outcome1 is then used in conditions outcome1 and not outcome1 to select one of two possible outgoing transitions. Consequently, p1 is the probability of outcome1 being assigned value true. The proposed statechart model of a two-way random switch can be considered to be instantaneous, provided that the asynchronous time model is adopted during simulation. If, on the contrary, the synchronous time model is adopted, the random switch operation will consume one unit of simulated time.

3 Application

3.1 Formal specification

Two basic types need to be introduced for this specification $[StopId, LineId]$. These types define identifiers for respectively, BusStop and Line. The *BusStop* class specifies an agent type which handle bus stops. Each of these agent has a specific identity, *id*, and may connect several lines.



These lines are specified by the *Connections* constant set. The behavior of the *BusStop* agent is defined by the statechart included in the *Behavior* sub-schema. The attributes for this type of agent represent the number of persons waiting. We distinguish people waiting for the direction D1 or D2 of the line identified by lineId and people waiting for a specific connection. The *passMouvAtStop* is executed as soon as a bus enters the *BusStop* and takes as input parameter the number of passengers exiting a bus. These people are distributed among the different connections following a binomial law. This is done by the predefined RandomBinomial function. This operation outputs the n! umber of passengers waiting for the bus to the *Bus* agent and assigns zero to these attributes. Moreover, the *GTNSD*1 event is scheduled by the predefined function *Schedule* in order to simulate the time during which the bus stays in the stop.

BusStop

Agent
Connections : $\mathbb{P}\,LineId$
id : *StopId*
lineId : *LineId*

Behavior

$passAtStopD1 : LineId \rightarrow \mathbb{N}$
$passAtStopD2 : LineId \rightarrow \mathbb{N}$
$waitingForConnectionD1 : LineId \rightarrow \mathbb{N}$
$waitingForConnectionD2 : LineId \rightarrow \mathbb{N}$

passMouvAtStop

$\Delta(BusStop)$
$passDown? : \mathbb{N}$
$passIn! : \mathbb{N}$

$passIn! = passAtStopD1(lineId)$
$\quad + waitingConnectionD1(lineId)$
$passAtStopD1' = 0$
$\quad \wedge\ waitingConnectionD1(lineId)' = 0$
$\forall\, l \in Connections \bullet (l \neq lineId) \Rightarrow$
$\quad waitingConnectiond1(l)' =$
$\quad\quad RandomBinomial(passDown?, pd1(l))$
$\quad \wedge\ waitingConnectionD2(l)' =$
$\quad\quad RandomBinomial(passDown?, pd2(l))$
$Schedule(GTNSD1(id, lineId), DelayAtSTop(passIn!))$

The *Bus* agent class defines the type for the agent representing buses. Each of this agent has a maximum capacity and a number of passengers which must be less or equal to it. The two operations of the agent are for respectively people exiting and entering the bus. These operations are interleaved with the *passMouvAtStop* operation of the *BusStop* agent. Indeed, when a bus arrives in a stop the squence *busPassOutAtStop* $\parallel$ *passMouvAtStop* $\parallel$ *busPassInAtStop* is executed and each operation outputs are equated with the inputs of the next operation.

Bus

Agent
capacity : $\mathbb{N}^+$

Behavior

passengers : $\mathbb{N}^+$

$passengers \leq capacity$

busPassOutAtStop

$\Delta(Bus)$
passDown! : $\mathbb{N}$

$passengers' = passengers - passDown!$

busPassInAtStop

$\Delta(Bus)$
passIn? : $\mathbb{N}$

$passengers' = passengers + passIn?$

References

[1] J. Cremer, J. Kearney, and Y. Papelis. HCSM: A framework for behavior and scenario in virtual environments.

ACM Transactions on Modeling and Computer Simulation, 5(3):242–267, July 1995.

[2] K. Erol, R. Levy, and J. Wenthworth. Application of agent technology to traffic simulation. In *Complex Systems, Intelligent and Interfaces*, 1998.

[3] K. Fischer, B. Chaib-draa, J. P. Müller, M. Pischel, and C. Gerber. A simulation approach based on negotiation and cooperation between agents: A case study. *IEEE Transactions on Systems, Man and Cybernetics*, 1999.

[4] K. Fischer, J. P. Müller, and M. Pischel. Cooperative transportation scheduling an application domain for DAI. Research Report RR-95-01, Deutsches Forschungszentrum für Künstliche Intelligenz, Deutsches Forschungszentrum für Künstliche Intelligenz GmbH Erwin-Schrödinger Strasse Postfach 2080 67608 Kaiserslautern Germany, 1995.

[5] K. Nagel, R. J. Beckamn, and C. L. Barrett. TRANSIMS for transportation planning. In *Proceedings of the International Conference on Complex Systems*, 1998.

[6] R. F. Paige. A meta-method for formal method integration. In J. Fitzgerald, C. B. Jones, and P. Lucas, editors, *FME'97: Industrial Applications and Strengthened Foundations of Formal Methods (Proc. 4th Intl. Symposium of Formal Methods Europe, Graz, Austria, September 1997)*, volume 1313 of *Lecture Notes in Computer Science*, pages 473–494. Springer-Verlag, Sept. 1997. ISBN 3-540-63533-5.

[7] P. Zave and M. Jackson. Conjunction as composition. *acm Transactions of Software Engineering and Methodology*, 2(4):379–411, Oct. 1993.