Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 00 (2012) 1–7

Procedia Computer Science

www.elsevier.com/locate/procedia

The 1st International Workshop on Agent-based Mobility, Traffic and Transportation Models, Methodologies and Applications

A flexible framework for developing integrated models of transportation systems using an agent-based approach

Vadim Sokolov[a,*], Joshua Auld[a], Michael Hope[a]

[a]*Argonne National Laboratory, 9700 S Cass Ave, Argonne, IL, 60439, USA*

Abstract

Travel demand, traffic flow and land-use models are typically modeled in a decoupled way, i.e. each of the components is modeled separately assuming the others are fixed. Moreover, the models are often developed by different groups for different contexts, requirement, etc. In this paper we present a prototype of a software framework which allows the user to develop an integrated simulation of a transportation system and also to link additional models to the new simulation in a standardized way . We use an agent-based approach as the basis of such a model. Integrated transportation system models allow model users to overcome the limitations of traditional aggregated, independent transportation models, particularly with respect to sensitivity to behavioral aspects of the travelers. Another requirement, which the software is to satisfy, is the interoperability of models developed in the new framework with legacy models. By interoperability we mean, that any component of the of the model can be interchanged by a legacy software and be used for the integrated simulation. This would allow disparate research groups working on modeling different aspects of a transportation model to plugnplay their models into the framework and test those as a part of an integrated model of an entire system, providing a benefit to researchers, modelers and institutional users of such models.

Keywords: integrated models, agent-based simulation, modelling framework

1. Introduction

There are several confluent factors in the transportation community which have influenced the development of a new kind of tool to answer questions about the transportation system. Historically, existing transportation-related models have looked at different aspects of the transportation system (travel demand, traffic flows, emissions, etc.) independently from each other. When the realization developed with the transportation community that these phenomena needed to be modeled in an integrated manner, attempts were made to link these unrelated models into a unified system. The integrated solutions produced have frequently been either inflexible, non-modular, or low performance. There was a need for the different models to inter-operate with one another to answer these questions about a transportation system, but transportation models generally lacked a common framework to do so in a straightforward manner.

[*]Corresponding author. email: vsokolov@anl.gov, phone: +1 630 252 6224, fax: +1 630 252 5394

The framework being developed to address these needs is called: POLARIS (Planning and Operations Language for Agent-based Regional Integrated Simulation). It has two major conceptual components. First, a high level agent-based modeling language targeted specifically at transportation - a minimalistic set of concepts, building blocks, and symbols which can be utilized to succinctly describe all possible elements of a transportation system. Second, a framework and SDK (software development kit) which facilitates the development, execution, and review of a model written in such a language.

The user community envisioned for POLARIS is broad, including those from a wide variety of transportation research disciplines and those involved in all phases of the transportation model development process from design to application. Transportation researchers will use POLARIS to test and validate theories in an integrated environment quickly and easily as well as refine and expand the transportation ontology model. Integrated transportation model developers would utilize POLARIS to weave together disparate model components developed by researchers and bring in new technologies in addition to connecting existing models of interest. Transportation modelers will apply models created by the integrated model developers and solve real world problems using POLARIS. Software developers working in the transportation arena will work to improve performance and increase usability as well as utilize the low level libraries to aid in functional programming or memory management tasks.

The design philosophy of the POLARIS modeling framework is founded on several tenets which serve to maximize ease of use and flexibility. First, give developers tools which will ease their model development efforts, however do not force them to be used. This is a common philosophy in many programming languages, advanced features are offered, but so are the low level concepts needed to assemble competitive features from scratch if the user is dissatisfied with the default implementation. Next, suggest to developers standardized symbols and structure which can be used to facilitate communication within their model components as well as among other models. A simple example is a term such as *location* (intersection, transit stop, point on a street, parking lot, etc.), nearly every software has its own version (likely with nearly identical implementation) of the same concept; in POLARIS, this concept is defined once and any applications written using it would use this same definition. This means that any developers wishing to interface with any POLARIS application need only to look at one central place in POLARIS to understand what location means in software terms. Next, using a fixed API, abstract low level details from high level ones such that core developers optimizing low level operations can work independently of researchers implementing high level concepts. This is a common thread in many well-engineered software APIs, it allows a model developed in POLARIS to be able to achieve high performance as well as re-usability, code clarity, and short development times. Finally, and perhaps most importantly POLARIS is founded on an ontological base which allows for extreme modularity. Within the POLARIS ontology, extremely complex components of the transportation system (such as human travelers) are broken down to their atomic parts and re-organized in a hierarchy. This is done to a degree such that a customized component created by one developer can understand another customized component created by another developer through understanding the more primitive (and more standardized) ancestry of the other component. As an example, consider a router: regardless of which algorithm is used internally, a user should be able to count on the fact that it will deliver, as a minimum, the path from a start location to an end location on a given type of network.

2. Using an ABM Approach to Model Transportation Systems

The conventional approach to modeling transportation systems is to split the system into distinct components (land use, network, travel demand) and model each of the components by fixing parameters and passing results between components. A typical example of this approach is the four-step travel demand model. The biggest disadvantage of those models is lack of behavioural realism [1], as they model travel as flows between aggregate points rather than the outcome of many individual decision to move from place to place. The methodologies developed within the agent-based (sometimes called multi-agent) modeling community provide a good basis for developing an interoperable disaggregated simulation framework for a transportation system to remedy these deficiencies.

The essence of an agent-based simulation is the concept of agents, which have a set of behaviours that govern the interactions among the other agents and the environment. This is a relatively new field of research, however, agent-based methodologies have proved to be a powerful tool which allows users to model a vast array of phenomena, such as political and social processes [2], software systems [3], manufacturing systems [4], urban dynamics [5], business applications [6], geographical systems [7] and economics [8]. The agent-based methodology allows the construction of a model of transportation systems in which travelers can have behavioral responses and sensitivity to the actions of other agents and the overall system performance. In contrast with a traditional four-step model, an agent-based approach leads to an out-of-equilibrium solution [9]. The issue of calculating an equilibrium solution when travelers are modeled as heterogeneous agents with individual behaviours is not very well explored. An attempt to develop several equilibrium strategies while developing the New York City transportation model is presented in [1]. However, the biggest advantage of an agent based approach is ability of understanding how patterns are formed and what parameters effect the formation of those patterns. There was a shift in the transportation community with in last 20 years towards disaggregated multi-agent simulation models, specifically in the area of activity-based models.

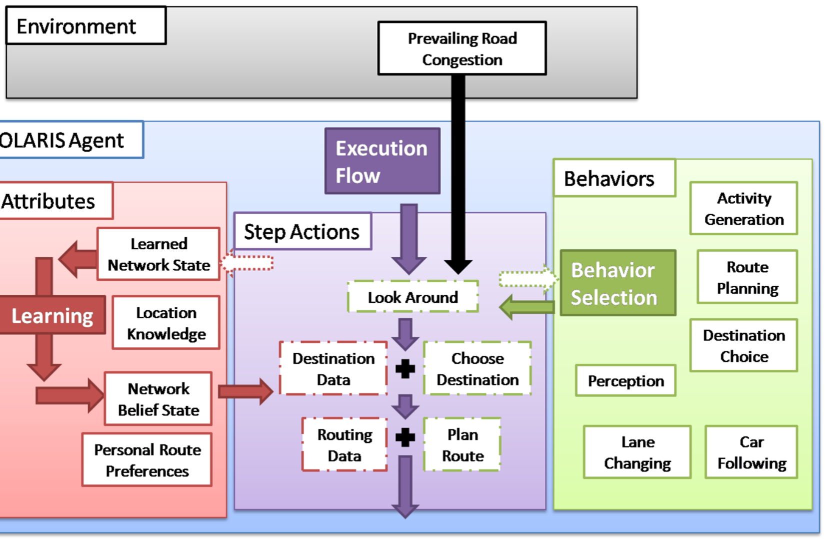

Fig. 1. Traveller Agent Design

Figure 1 shows an example of a traveller agent design. This is a typical skeleton used to design an intelligent agent [10]. The agent has several behavioural aspects (green box), which are associated with the agents state. For example, while in driving state a traveller agent might have the following behavioural aspects: (i) Car Following, (ii) Lane Changing, (iii) Intersection Passing, (iv) Traffic Jam Detection, etc.. Each behavioural aspect is a set of actions which is available to the agent at a given state. At each time step (tic), the agent decides on actions (performs planning) based on the current percepts, which are part of the step function (purple box) as well as the belief state (red box) and executes the scheduled actions. The belief state is the model of the agent's knowledge about the environment (network travel times, transit schedules, etc.). The belief state can either be learned as a result of previous percepts or can be prescribed by a modeller. The state of the agent, the belief state, the planning routines as well as behaviours available to an agent are spread over several components. The reason it is not designed as a single object is that the interoperability requirement discussed in the 4 section can be met. The percepts of the agent are modelled using a publish subscribe pattern [11] and to be handled by the POLARIS communication system.

In addition to the artificial intelligence and agent-based modeling communities, POLARIS also pulls heavily from the computational science fields. Communication in POLARIS was inspired by standards in computer networking. Concepts such as TCP/IP demonstrates how it is possible for two extremely different machines can talk with each other quickly with a minimum of common understanding and BSD sockets demonstrate how a relatively simple user API can be used to implement a specific communication. In many ways POLARIS is designed partially as its own operating system: it manages memory, schedules execution processes, protects data regions, performs inter-process communication, and does so while providing a stable API to the developer. Therefore, many techniques used for creating fast, secure, and effective operating systems are being used in the low level core library. Finally, a strong knowledge of processor design, automated parallelization, and advanced generic programming techniques are utilized in concert to provide an API to the low level functionality which is extremely high performance as well as very generic and simple to use.

3. Architecture Overview

The prototype POLARIS framework addresses the major requirements of a flexible, extensible, agent-based transportation simulation environment. POLARIS is organized as a series of six components, which address different aspects of the framework. The components include the following, as can be seen in Figure 2: (i) TOM - Ontological representation of components in a transportation system, (ii) TOL - Basic set of pre-configured simulation objects, (iii) TSF - Workspace for user-defined simulation objects (extension to TOL), (iv) EXM - Simulation initialization and execution, (v) IPC - Low-level communication operations; contains ontological representation of interoperable objects and a user API for communication and (vi) COR - Low-level data operations; contains utility functions and generic data containers, memory management, execution control.

The Figure 2 shows how the components relate to each other, and how they would be used by different levels of POLARIS users. A POLARIS core developer, for example, would have access to all levels of the code-base, and at the deepest level would likely including making changes to the Core Library (COR) to optimize low-level details, or handling some aspects of the IPC module, such as management networked communications, adding new communication routines, etc. At a slightly higher level of abstraction, a core developer could also make changes to the object model (TOM) and add new classs to the object library (TOL). Areas where only a core developer would have access are highlighted in blue. The next class of users would be model developers, who are able to create new objects in the TSF and also are generally required to write specific interchange routines in the IPC for whatever external program is being hooked in. Model developers are also likely to write the execution module for a specific model type. Model developers would access framework areas highlighted in red. Finally, the highest level of user would be the model user, such as a modeler at a state planning agency, or similar. These individuals would have access only to the specific model implementation developed by a model developer, in effect using one instantiation of a POLARIS model.

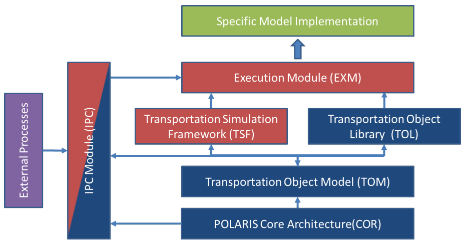
Specific Model Implementation
Execution Module (EXM)
Transportation Simulation Framework (TSF)
Transportation Object Library (TOL)
Transportation Object Model (TOM)
POLARIS Core Architecture(COR)
IPC Module (IPC)
External Processes

Fig. 2. POLARIS Framework Design

The fundamental concepts underlying all POLARIS transportation simulation models are defined in the Transportation Object Model (TOM). This is a highly abstract set of interfaces which determine the object inheritance hierarchies and capabilities which exist in general transportation models, and how these objects/concepts relate to each other. The TOM is therefore organized as a high-level, non-exhaustive ontology, which attempts to define the base objects and characteristics from which any element which would exist in a simulation could be defined. All objects in the TOM derive from the base Element class, which represents any object or concept in the TOM. The TOM is organized in such a way that the requirements fundamental to an object are passed to inherited object, while the capabilities of the derived objects are passed up to the base element. This is slightly different from the usual object-oriented class structure, where derived objects inherit the requirements of base object only. In other words, in OOP, a base class generally has no information about the behavior of a derived class, but in the TOM a base object actually gains access to the functionality of derived objects. An illustrative example is shown in Figure 3 below. The Figure 3

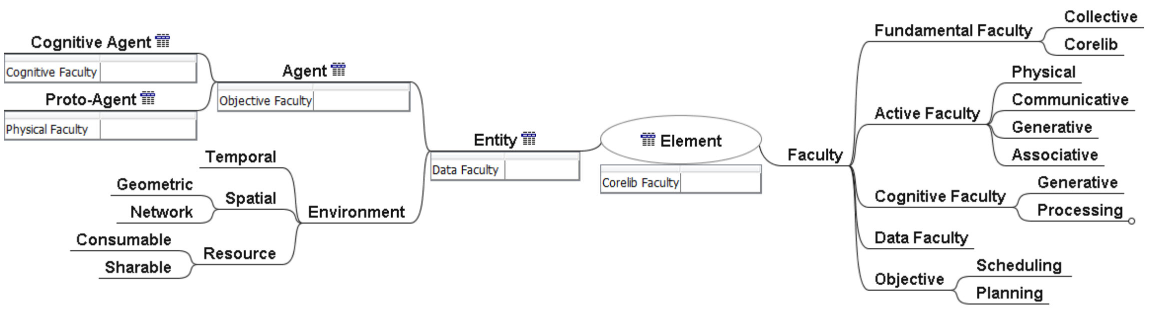

Fig. 3. Illustrative Example of Transportation Object Model (TOM)

shows the *Element* class as the base of all other objects in the TOM. Therefore, as stated above, while all the objects in the model derive from Element, declaring a simulation object of type Element in the simulation framework actually gives that object access to all functionality of every object in the TOM. The highest level split in time is between an Entity and a Faculty. By POLARIS convention, an Entity is simply something that can exist outside of another element while a faculty can only exist as a member variable within another element. From the example above, an object derived from the Agent class can be created within the EXM, while an object of type Objective Faculty could not, and is in fact specified as a member of an Agent or Agent-derived object. For this reason faculties in the TOM tend to represent either Entity capabilities, concepts or complex data members, while the Entities tend to be the actual things in a transportation simulation, something which has an id.

The power of this approach is that the TOM defines a set of core, useful concepts in common across most transportation simulations, which attempt to define the objects and behaviors at a fairly abstract level. This allows developers to create specific instances of these fundamental objects and use them in different ways depending on the needs of the specific model instance. So, one modeler may decide to allow traveler agents to have limited cognitive ability and derive their class as Traveller: Proto-Agent, while another may want to allow travelers to learn from simulated experiences and therefore create a Traveller : Agent, which gains the abilities of the Proto-Agent, but also has access to the Cognitive Faculty of the Cognitive Agent. This flexible, extensible structure is allowed due to the unique design of the TOM.

4. Interoperability in Transportation Simulation

One of the core motivations for POLARIS is to have a centralized standard and communication framework which can connect independently developed modules or processes. This is accomplished by using the TOM in concert with two standardized communication spheres. The first encompasses all intra-agent communication (that is, the communication of an agent with other parts of itself). The second sphere covers all inter-agent communication (that is between an agent and the environment or another agent).

Having a system to handle communication within an agent may seem unusual as it seems intuitively true that an agent should at a given time have access to the totality of itself. However, the reality of modeling an agent using a programming language means that there will be artefacts not encountered in the real world such as the presence of parallel processing, importance of data concurrency, and necessity of intermediate attributes without a real world equivalent. POLARIS is designed first and format as a modeling language, therefore it is important to aid the modeler with these programming-related artefacts. The intra-agent communication interface includes concepts such as exposing simple consistent interfaces to other developers, ensuring that member data access is rationally designed and strongly enforced, and finally (as POLARIS is designed as a parallel application) that data is thread safe where necessary.

The interoperability, modularity, and agent based design of POLARIS are all enhanced by the robust inter-agent communication system. The primary goal with this API is to ensure that an agent can talk to any other type of environment or agent even if that agent might only be a virtual wrapper around an external process. This is accomplished by requiring all agents to maintain a locally customized (yet heavily standardized) version of the same communication module and ensure that agents are never able to see more than the interface dictated by the other agent's communication module. Furthermore, in any communication, the requestee dictates the terms of the communication using only POLARIS components defined in the TOM and the requester merely initiates the communication. Requests need not be fulfilled immediately or even at all, though it behooves the developer to provide this kind of feedback to the other developer.

The nuts and bolts of this inter-communication are written in terms of direct memory access (for intra-process), memory mapped files or conventional files (for inter-process, intra-machine), and socket to socket communication (for inter-process, inter-machine). Depending on the specific external application which will be communicated with, a more or less extensive translation routine from that application to POLARIS terminology will need to be written. It is the goal of the POLARIS project to be able to easily communicate with other major software from every corner of the transportation modeling realm, possible candidates include DYNASMART-P, TransCAD, CORSIM, UrbanSim, Synchro, MOVES, and others.

5. Conclusion

After a rigorous conceptualization and design process, POLARIS is initially being built as an open source C++ API and collection of libraries to meet the needs of current and future integrated transportation model developers and users. This is being accomplished by combining concepts from the general agent-based community with a new transportation-specific object model and implementing them using a highly performance optimized library of core operations and advanced concepts from the software engineering and design community. Users will create their models by utilizing the API in a IDE such as Microsoft Visual Studio and linking with these libraries. The code documentation will be extensive and produced in an easy-to-understand format by a tool such as Doxygen. As the framework evolves from its initial release, usability will continue to increase through the wrapping of many of the core concepts within a specially designed domain-specific high level language and graphical user interface. These improvements will provide a strong interactive help system, gentle enforcement of program structural concerns, additional tools to aid in the debugging of the model, and sub-programs usable for simulation data visualization. In addition, the concepts in the TOM will continue to evolve as the transportation research community determines what components are the strongest and which might be removed in favour of new categories. It will be an ongoing goal to promote a centralized repository where users can submit, critique, and pick up model fragments written in POLARIS in an effort to provide the best possible default user libraries as well as share ideas about how they might be improved or utilized in currently unknown ways.

References

[1] P. Vovsha, R. Donnelly, S. Gupta, Network Equilibrium with Activity-Based Microsimulation Models: The New York Experience , Transportation Research Record: Journal of the Transportation Research Board (2008) 102–109.

[2] R. Axelrod, The Complexity of Cooperation: Agent-Based Models of Competition and Collaboration, Princeton University Press, Princeton, New Jersey, 1997.

[3] N. R. Jennings, On agent-based software engineering, Artificial Intelligence 177 (2) (2000) 277–296.

[4] W. Shen, D. Norrie, 'Agent-Based Systems for Intelligent Manufacturing: A State-of-the-Art Survey, Knowledge and Information Systems: An International Journal 1 (2) (1999) 129–156.

[5] M. Batty, Cities and Complexity: Understanding Cities with Cellular Automata, Agent-Based Models, and Fractals, The MIT Press, 2007.

[6] M. J. North, C. M. Macal, Managing Business Complexity: Discovering Strategic Solutions with Agent-Based Modeling and Simulation, Oxford University Press, USA, 2007.

[7] A. Heppenstall, A. Crooks, L. See, M. Batty (Eds.), Agent-Based Models of Geographical Systems, Springer, 2011.

[8] L. Tesfatsion, Agent-Based Computational Economics: Growing Economies from the Bottom Up, Staff General Research Papers 5075, Iowa State University, Department of Economics (Mar. 2002).

[9] W. B. Arthur, Out-of-Equilibrium Economics and Agent-Based Modeling, Vol. 2, 2006, Ch. 32, pp. 1551–1564. doi:10.1016/S1574-0021(05)02032-0.

[10] S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, 3rd Edition, Prentice-Hall, Englewood Cliffs, NJ, 2009.

[11] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design patterns: elements of reusable object-oriented software, Addison-Wesley Professional, 1995.