UCI-ITS-AS-WP-99-2

A Simulation Framework and Environment for Activity Based Transportation Modeling

UCI-ITS-AS-WP-99-2

James E. Marca [1]
Craig R. Rindt [1]
Michael G. McNally [1]

[1] Department of Civil & Environmental Engineering and Institute of Transportation Studies
University of California, Irvine; Irvine, CA 92697-3600, U.S.A.
jmarca@uci.edu
crindt@uci.edu
mmcnally@uci.edu

November 1999

Institute of Transportation Studies
University of California, Irvine
Irvine, CA 92697-3600, U.S.A.
http://www.its.uci.edu

A Simulation Framework and Environment for Activity Based Transportation Modeling

James E. Marca Craig R. Rindt Michael G. M^{c}Nally [*]

November 19, 1999

Abstract

This paper presents an event-based simulation framework designed specifically for applying activity-based transportation models to a variety of problems. General concepts of activity-based travel modeling are discussed followed by a specification of abstract concepts common to most activity modeling approaches. A simulation framework is developed and implemented in two examples to demonstrate the feasibility of the concepts presented.
Keywords: Activity-based modeling, travel behavior, computer simulation 5600

1 Introduction

This paper presents an event-based simulation framework designed specifically for applying activity-based transportation models to a variety of problems. The result is a simulation environment which is focused towards research. This environment is a starting point for development of a tool which will address real-world problems. Therefore, although this paper is directed towards researchers, it will be of interest to practitioners.

By way of introduction, we briefly discuss in section 2 some basic ideas of activity-based transportation analysis. Next, section 3 introduces three different approaches to activity simulation. These three approaches illustrate the breadth of variation in the possible details of activity based simulations. These differences in the details have so far hindered the development of a unified simulation framework. We circumvent this problem by identifying the abstract concepts that are common to any kind of activity-based simulation.

Section 4 describes the basic ideas of event-based simulation. This section also describes the necessary capabilities of the abstract activity simulation environment. Section 5 describes how the abstract framework can be applied in practice. Two simple activity simulations are presented, one implemented in C++, and the other using the Swarm simulation libraries (Langton, Burkhart, Daniels, Jojic and Lancaster, 1999) in Objective C.

This paper concludes with a summary of the major points, a discussion of how this work can be applied in the field of transportation engineering, and the directions it may take in the years ahead.

2 Applying activity-based transportation modeling concepts

The fundamental premise behind all activity-based transportation modeling is that people travel in order to do something at a place other than where they are, and that understanding the

[*]Institute of Transportation Studies; University of California, Irvine; Irvine, CA 92075; 949-824-6571; jmarca@translab.its.uci.edu

motivations behind various activities should improve our understanding of travel behavior. From (Hägerstrand, 1970) to the present, the focus of activity-based models has been on the people who are performing the activities which create the need for travel.

Stepping beyond the trip-based travel model paradigm into the activity-based one opens up layer after layer of potential abstraction. In order to gain insight into the fundamental properties and relationships of activities and activity sequences, one can study the factors motivating and limiting those activities. And those factors have further layers of abstraction, right down to the basic human needs for food, shelter, and companionship. The one trait all of these layers have in common is the focus on the individual, rather than aggregated populations.

It is our belief that any comprehensive application of activity-based concepts to travel modeling will require a simulation approach. The focus on the individual inherent in activity analysis naturally leads to a simulation. A more detailed discussion of this point is given in section 4.1.

3 Alternate approaches to activity-based simulations

This section describes three distinctly different activity-based simulation models. Each modeling approach is geared towards different kinds of questions that one might wish to answer. The reason we do this is to ensure that we develop a sufficiently general environment, so that it will be widely useful to the activity analysis research community. The focus of each of the following subsections will be on the generalities of each implementation, rather than worrying about the details or the correctness of the underlying activity paradigm. The commonalities between the different simulations are summarized in section 3.4, and form the basis of the general simulation environment presented in section 4.

3.1 Representative activity patterns

The first simulation model presented is the well documented representative activity pattern concept that has its roots in Recker, M^{c}Nally and Root (1986) and is most recently described in M^{c}Nally (1990) and M^{c}Nally (1998). This approach first classifies activity patterns of surveyed individuals into representative patterns. A synthetic population is generated, and then the characteristics of the population are matched with the characteristics of surveyed individuals, which in turn are matched to the classified activity patterns. This matching is probabilistic rather than one-to-one, as explained in M^{c}Nally (1990). This matching enables each person to be assigned a daily set of activities.

Since the activities on any given day are randomly assigned, and since the locations for the activities are also randomly assigned, the execution of those activities creates random travel movements. This randomness is grounded to observed activity patterns, and so the resulting travel patterns are likely to be much closer to real movements than those deriving from a uniform distribution of activities and locations.

The elements required for this kind of an activity simulation are shown in table 1. The contents of the *object* column shown in table 1 are also those required by most other typical activity based simulations published in the literature. Most simulations have *actors* and *locations*. However, the various simulations differ in the contents of the *behavior* and *source* columns.

3.2 Intelligent actors

A long-standing hypothesis in travel demand analysis is that people choose the travel modes that maximize utility. Analyzing activity sequences allows one to examine what it might mean for people to maximize the utility of their activities. Travel choices then become just another part of the choices the person makes in building a day's activities. Even more interesting, one

Table 1: The elements required for an activity simulation using the activity pattern approach

object	behavior	source
actors	create activities	activity survey data
	choose locations	activity survey data, land use data
locations	provide site for acts	land use data
	provide travel times	land use data, travel survey data

Table 2: The elements required for a simulation of the activities of "intelligent" agents

object	behavior	source
actors	create activities	activity survey data
	evaluate activities	survey data, multi-attribute decision theory, conjecture
locations	provide site for acts	land use data
	provide travel times	land use data, travel survey data
	capacity limits	land use data, survey data, conjecture
	provide act quality info	survey data, conjecture

can explore in a simulation what happens when different kinds of value functions are given to individual actors.

A simulation of intelligent actors would require the following elements. First, there would need to be actors, and they would need to be assigned an initial set of activities to perform, using some accepted method. Second, those activities would have to take place somewhere in the simulated city. To add realism (without losing generality), suppose further that some locations, such as home and work site, are fixed for the duration of the simulation, while others are flexible.

Initially, one would expect that the first simulated day would produce lots of patterns of movement that we would consider strange, since the locations and activities are drawn randomly. The next step is to hypothesize a mechanism for evaluating a day's activities—in other words, hypothesize intelligence using techniques such as multiattribute decision theory (Saaty, 1994; Goicoechea, Hansen and Duckstein, 1982). One decision criteria is to trade off the quality of the activity performed at a location with the level of congestion and delay experienced. The actors try to maximize the quality of their activities, while minimizing the delays that they face.

At the end of the day, each person would be randomly assigned a new sequence of activities for the next day. However, unlike the first day, the random assignment of characteristics (such as location) to each activity would be tempered by the "knowledge" that the actors collected by performing past activity sequences. The way the collected knowledge is incorporated is through the application of some hypothesized decision algorithms.

The elements of this simulation are summarized in table 2.

3.3 The interaction of cooperating actors

The third and most complicated example activity simulation presented examines cooperation and interpersonal constraints. Instead of performing activities that satisfy individual constraints, each individual interacts with others, and belongs to one or more groups which add additional constraints to the activity process.

Perhaps due to the difficulty of the problem, no one has yet developed a general cooperative actor simulation. Most research to date has focused on household groups (Borgers, Hofman, Ponjé and Timmermans, 1998; Recker, 1995), but this research has not been extended to tasks and constraints required by other groups, such as work colleagues or social circles. Since there are no accepted models of general interactions between actors, we propose a new simulation

Table 3: The elements required for a simulation of the activities of cooperating and interacting agents

object	behavior	source
actors	create activities	activity survey data
	join groups	survey data, sociology literature, conjecture
	communicate with other actors	survey data, conjecture
	arbitrate between the dictates of different groups	sociology literature, conjecture
groups	dictate activities to members	survey data, sociology literature, conjecture
	interact with locations	conjecture
locations	provide site for acts	land use data
	provide travel times	land use data, travel survey data
	capacity limits	land use data, survey data, conjecture
	provide seeds for formation of groups	conjecture, sociology literature

model structure. If the simulation framework we develop is to be widely useful, it must also be applicable to new and experimental models such as this one.

A simulation incorporating interaction and cooperation between actors is as follows. The first step is to generate the actors and the locations. Then the actors need to be assigned to groups. At a minimum, each actor should be assigned to a household, with each set of household members forming a single group. Other groups might be formed as the simulation progresses. Next the group "rules" should be specified. For example, one household might place constraints over the time and place of meals.

The next step for the simulation is to generate activities for each individual which conform to the requirements of the person's member groups. But instead of forcing actors to check with all of their groups when scheduling activities, a modeling structure that seems to work well on paper is to transfer control of scheduling activities to the group construct, which becomes a type of super-actor. For example, a household group can schedule in-home meals for each household member. The actor would then schedule his or her other activities, with or without knowledge of the scheduled constraint. At some point the "eat at home" event would pop up, forcing the actor to respond to the constraint. An ephemeral group might schedule activities that are ignored by actors who are serving the constraints of a more important group.

All that is left is to devise different constraint structures for the groups. Since this is an experimental simulation, the results of any study are likely to be statements about likely *kinds* of group interactions. Therefore it is important that the simulation environment allow for different types of groups to be specified, with different types of rules and levels of enforcement for those rules.

Once each person's activity sequence is set, the activities can be executed. As the day progresses, circumstances such as unexpected queuing may force changes in what the actor intends to do. The level of the conflict depends on the strength of the constraints. The features of the conflicting activities may have to be revised based on the level of conformity to the imposed group constraints. For example, if there is a traffic delay, then it may happen that a social group requirement for entertainment activities after work is modified to eliminate a conflict with the household group's dinner activity.

The elements required to build a simulation of interacting and cooperating actors are shown in table 3.

3.4 The common elements from each of the preceding activity-based simulations

The goal of this project is to develop a common, abstract simulation interface. This will serve as the foundation for a simulation environment that can handle a wide variety of activity-based travel simulations. This subsection will reexamine the needs of each of the example simulations to extract a common set of features. These features will then be used to express an abstract basis for generic activity-based simulations.

In terms of *objects*, it is clear that activity simulations require *actors* and *locations*. In what follows, these concepts are generalized as *agents* and *locales*, respectively.

On the functional side, activity simulations require the *generation* of activities, and *communication* between the simulation objects. These functions can be wrapped up into a third abstraction called an *activity manager*.

3.4.1 Agent

An actor and a group are different implementations of the generic concept of an *agent*. An agent does things, communicates with other agents, and communicates with locations. The details of these actions are left to the analyst who applies the agent concept.

An actor, meaning the abstraction of a person, extends an agent by adding the feature that the actor is a single person, which implies different things to different simulations. A person in one simulation might have a fixed home and work location, and a preferred travel mode. Another simulation might not care about fixing locations. The actor inherits from the agent the ability to communicate with other actors, with other agents, and with locations.

The group concept described in section 3.3 also extends the capabilities of an agent, this time to include the ability to exert control over other agents. The full set of abilities of the agent are passed on to the group, with the addition of the ability to dictate activities for the group's members. This which allows the group to schedule activities for its members as if it were a bunch of simple agents acting in concert.

3.4.2 Locale

The location concept presented in each of the different activity simulation applications is roughly consistent. However, the location sometimes needs to communicate more than just (x, y) coordinates to the agent who is performing an activity there. We have chosen to call the abstraction of the location concept a *locale*. A locale at a minimum provides a site where an activity may take place. As such, a locale must communicate with agents in order to negotiate and properly schedule the use of resources. Other details are left to specific extensions of the locale concept.

A location is an extension of a locale that implements the details of resource allocation, communication, and so on, and adds other details such as latitude and longitude or operating hours. One might implement transportation as a specialized locale extending over space, providing the ability to change coordinates (see the example described in section 5.2). As another example, one might decide to connect aggregate data to disaggregate data by defining land-use-zone objects containing many specific locales, just as groups contain specific agents.

3.4.3 Activity manager

The third abstract concept that is common to all activity simulations is the *activity*. While the details are always different, an activity requires a series of events and messages in a simulation. We have decided to encapsulate these events and messages, whatever they may be, into an *activity manager* class. For simple simulations, it is probably sufficient to ignore the activity manager and just send messages between agents and locales directly (see section 5.1). However, the negotiations between even a single agent and a single locale can become difficult to follow. It is more convenient to describe a third party that manages the communications necessary for the instantiation of an activity. The benefits of using the activity manager object become critical when the simulation allows agents to cooperate and coordinate with other agents and with groups. The activity manager class must contain the ability to communicate with an agent and a locale. The activity manager can also be extended to provide the specific details necessary for the performance of a specific kind of activity.

4 Simulation modeling as applied to activity analysis

4.1 Event-based simulation

Simulation is appropriate for examining systems whose complex interactions resist direct analysis. The system under study here is the urban transportation system, but with the focus placed on the activities behind the trips. A closed-form expression describing the interactions of the many kinds of locations, people, and activities would be very difficult to develop. Much easier is the problem of describing parts of the system, then describing how the parts interact, and finally running a simulation to examine the system. Depending on the purpose of the research, one can examine the statistical impact of either variations in the specification of the interactions, variations in the descriptions of the individual elements, or both.

In an *event-based simulation*, discrete events happen at discrete times, with nothing of interest occurring in the interim. For example, the real world activity of eating lunch is modeled as two successive events: engaging in a meal activity at 12:00 noon, and completing the meal activity at 1:00 p.m. In between the two events, if no other event occurs, all system variables remain constant. When the first event occurs, some variables are changed to represent the commencement of the lunch in question. When the concluding event occurs, some variables are changed to reflect the fact that lunch has ended.

In an object oriented environment, each event contains its own time of execution. The simulation schedule is a list of events sorted in increasing order. To add an event, it must be inserted into the appropriate spot on the list. The execution of an event defines the current time for the entire simulation. Again using the lunch example, if nothing happens between the beginning and the end of lunch, then the global simulation clock will jump from 12:00 noon to 1:00 p.m.

The alternative way to view a simulation is to take a continuous time approach. Returning to the lunch example, one might define a continuous variable for the desire to end lunch. This desire would start out very low at 12:00 noon when lunch begins. The simulation would tick off very small clock increments, and there would be a growing desire to end lunch. Finally, at some point in time, the desire to leave lunch would rise high enough to fire off a change in state and end lunch.

The continuous approach has many strengths. The most attractive feature of the continuous simulation model of activities is the ability to describe events and motivations in terms of competing demands that rise and fall with the passage of time and with the influence of other system variables. For example, with each delectable mouthful of seared tuna with rice and grilled vegetables, a person gets less hungry, but at the same time less food is left on the plate. If the plate is empty before the person is satisfied, then the waiter must be called to order crème brûlée.

The weakness of the continuous approach is the need to update every variable with each small increment of a simulation clock. An event-based simulation can be seen as a way to reduce the number of calculations in a continuous simulation. Instead of "continuously" updating variables, the event-based simulation assumes that all variables remain constant until the next relevant event, and so the rate of change of all variables is also constant over the interval. Even as more and more details are added to an event-based simulation, it still tends to be more efficient than a continuous simulation since only the variables affected by a particular event need to be checked and updated.

4.2 The specification of an event-based simulation environment for activity modeling

A starting point for designing a general system for activity simulations is to examine existing simulation packages. One such program, Arena (Kelton, Sadowski and Sadowski, 1998), provides an excellent environment for designing industrial simulations. However it is not so easy to program an activity simulator. In testing this product, we concluded that in order to build a satisfactory activity simulation, it is necessary to write custom code, using C, C++ or SIMAN (Arena's underlying simulation language (Kelton et al., 1998)), tasks which defeat the purpose of using a high-level tool like Arena. Although we did not do extensive testing, we expect that other integrated simulation packages also face the same constraints.

The alternative to using an integrated environment is to build a related library of routines. This paper explores the use of one such library, called Swarm (Langton et al., 1999), and the potential for programming a completely new environment in C++. The details of both of these approaches are covered in section 5.

The next step in designing a general simulation environment is to specify a set of related abstractions. This was done in section 3. That section concluded with a discussion of the common elements of the *agent*, the *locale*, and the *activity manager* objects, as well as the need to communicate between these objects.

An *activity manager* can be viewed as an object that governs all of the individual, discrete events that constitute an activity in the usual sense. One possible breakdown of the events comprising an activity is shown in figure 1. Note that the necessary storage details for the activity manager object are not shown in the figure.

An *agent* is something or someone who performs activities. In order to get the ball rolling, one (temporary) axiom in this simulation environment is that the motivation for activities comes from the agent. This implies that the code that implements the analyst's model of how activities are generated should be built into the various implementations of the agent class.

A *locale* is primarily a provider of the resources necessary for an agent to engage in an activity. Resource limitations are implemented through different types of locales being available for different types of activities at different times of the day.

Binding together the interaction of the activity manager, the agent, and the locale are *events* and *messages*. Events are used to properly sequence everything that happens in a simulation. Events are characterized by a timestamp, and are loaded onto an external schedule for execution in order of their timestamp. Messages are instantaneous communications between objects.

The details governing all of these abstractions are the responsibility of the analyst creating the simulation. The events must be relevant to the implementations of the activity managers, agents, and locales that are in use, and the messages must be logically consistent. For example, if one an agent's methods, say `agent.EndActivity( timestamp )`, is expecting to receive only a timestamp code as a message from an event, say `event::beginActivity`, then `event::endActivity` should send *only* a message containing a timestamp to `agent.EndActivity( timestamp )`.

One example of an activity simulation (based in part on the implementation of section 5.1) is shown in figure 2. The logic diagrammed in this figure makes some basic *implementation* assumptions about the behavior of agents and locations. First an activity manager is created by an agent engaging in a new activity (message 2). The activity is started when and if the locale has the resources to serve the request for service (messages 3 and 5). This will happen when another event ends (message 11), or if the locale has capacity when the request is received (message 4). Another assumption is that both the locale (message 5) and the agent (messages 6 and 7) have some input into the determination of the duration of the activity in question. An

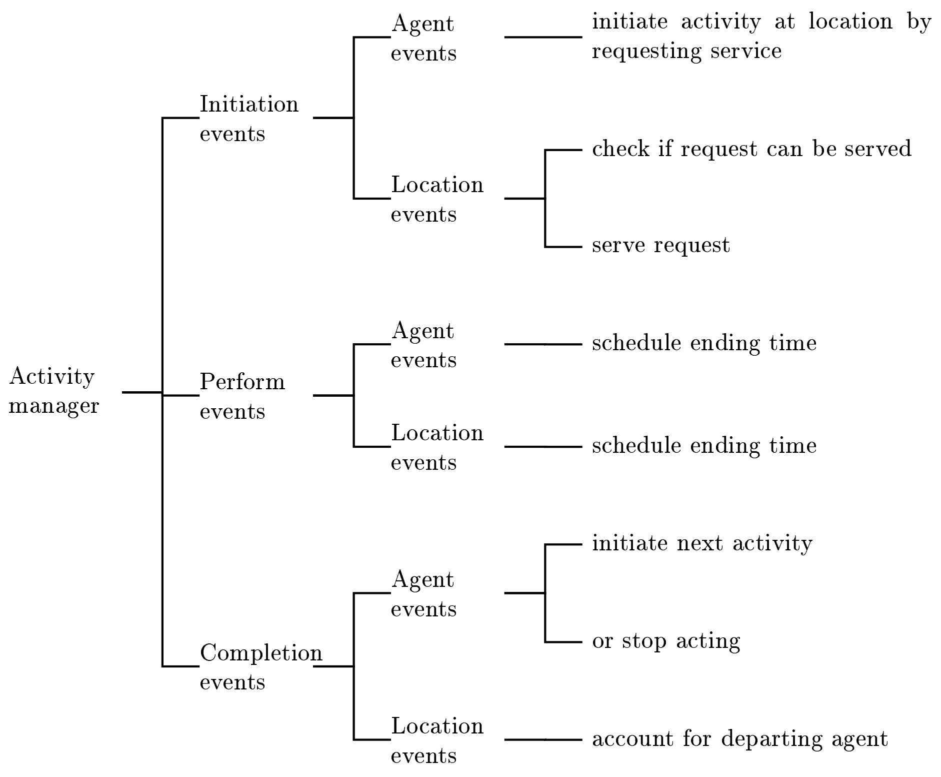

Figure 1: One set of discrete events governed by a generic activity manager

alert event is placed on the simulation schedule (message 8), which will end the activity at the correct time (messages 9, 10a, and 10b). Other message and event logic will work equally well within this framework.

So far we have not mentioned travel. Under the abstract paradigm presented, travel can be represented as a type of activity. The locale providing the resource is the transportation system that exists for the mode chosen by the agent. One straightforward implementation of this is given in the example described in section 5.2.

One final point to make is that there are events that may happen outside of the simulated world. The analyst obviously is interested in examining the progress of the simulation. One way to do that is to schedule events that probe particular features of the agents, activities, and locales that are of interest. The results of these probes can then be stored to cumulative statistics objects or dumped to some graphical interface. For example, if one is interested in the average queuing delay at a particular locale, then one would implement an average queue-length method for the locale, and schedule an event that sampled that method at the end of each day.

5 Two examples

This section demonstrates how one can use the abstract simulation framework to implement an activity simulation. As we've tried to stress repeatedly, simply declaring the existence of events, activities, agents, and locations does not produce a simulation. Instead, the analyst is responsible for implementing what he or she *means* by these terms. This is where the object-oriented features of C++ and Swarm come in handy. We have constructed an abstract framework that specifies the minimum interfaces that events, agents, activities, and locales must have in order to function. In addition, we have created basic simulation tools that expect these minimum sets of functions for each of these objects.

The following two subsections present specific applications of the simulation environment. Section 5.1 is an early implementation designed more to exercise the simulation concepts than to prove any profound result. This simulation does not implement the activity manager class. Rather it relies on a more complicated system of messages passed between agents and locales. This simulation is programmed in C++. Section 5.2 is a more evolved simulation, one which builds upon the lessons learned in early efforts to design a standard simulation interface. This simulation does implement the activity manager class, and is written using the Swarm simulation library.

5.1 Simulating the travel patterns to and from a restaurant area

The first example application is to simulate the impact of indirectly improving access to a restaurant district by submerging an elevated expressway that separates the neighborhood from the downtown area. This example is entirely fictional, and any results obtained should not be applied to any real world problems. The intent of this simulation was to exercise the abstract simulation concepts only.

In the base case, access to the neighborhood is difficult. There is a large area available for parking underneath the expressway. Parking patrons access the restaurants in the neighborhood by walking underneath the expressway. Patrons who drive directly into the neighborhood are required to use valet parking at the destination restaurant. In the after case, driving access to the neighborhood will improve, while parking availability will drop as a result of the loss of the parking lot under the expressway. Some very simple assumptions were made about the shifts in mode, requiring that the delays on each access method in the after case should be roughly equal.

The restaurants in the neighborhood are busy, with queues forming on Friday night in the before case. To simplify matters, the restaurants were lumped into two different locales. The

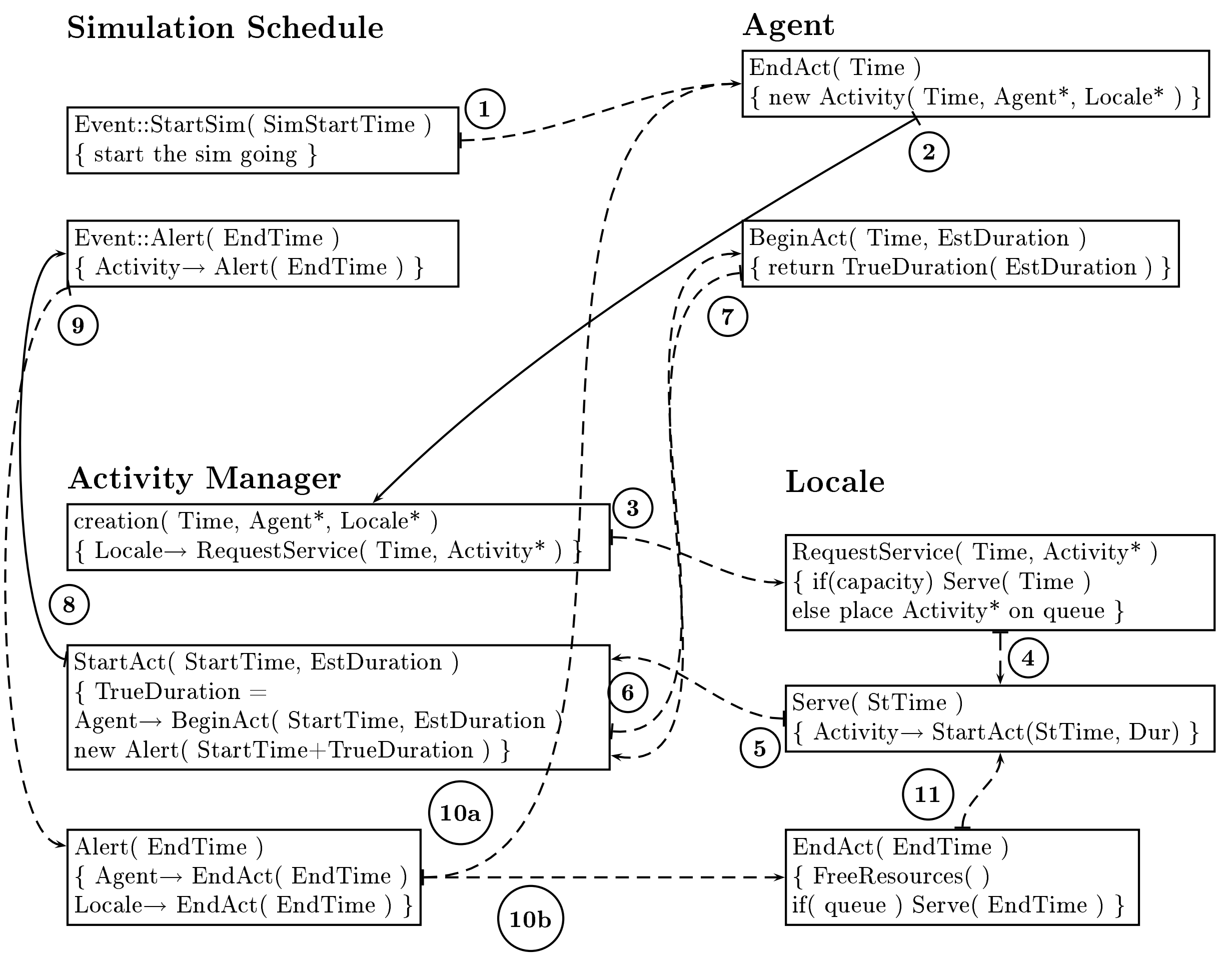
Simulation Schedule
Agent
Event::StartSim(SimStartTime)
{ start the sim going }
EndAct(Time)
{ new Activity(Time, Agent*, Locale*) }
Event::Alert(EndTime)
{ Activity→ Alert(EndTime) }
BeginAct(Time, EstDuration)
{ return TrueDuration(EstDuration) }
Activity Manager
Locale
creation(Time, Agent*, Locale*)
{ Locale→ RequestService(Time, Activity*) }
RequestService(Time, Activity*)
{ if(capacity) Serve(Time)
else place Activity* on queue }
StartAct(StartTime, EstDuration)
{ TrueDuration =
Agent→ BeginAct(StartTime, EstDuration)
new Alert(StartTime+TrueDuration) }
Serve(StTime)
{ Activity→ StartAct(StTime, Dur) }
Alert(EndTime)
{ Agent→ EndAct(EndTime)
Locale→ EndAct(EndTime) }
EndAct(EndTime)
{ FreeResources( )
if(queue) Serve(EndTime) }
1
2
3
4
5
6
7
8
9
10a
10b
11

Figure 2: An example set of messages and events to initiate, perform, and complete an activity

first represented cheap restaurants, and the second represented fancy restaurants. The two types of restaurants differ in their capacity, with the cheap restaurants having almost twice the total capacity of the fancy restaurants.

The simulation was implemented in a manner very similar to figure 2, although without an explicit activity manager class. The base case simulated 1,800 customers on Thursday, and 2,100 customers on Friday. The after case assumed that the improved access would increase both days by 200 customers. All of these customers were randomly assigned a start time, and then randomly assigned to one of the four activity sequences: (ParkIn, FancyMeal, ParkOut), (ParkIn, CheapMeal, ParkOut), (ValetIn, FancyMeal, ValetOut), or (ValetIn, CheapMeal, ValetOut). Each activity was served by a unique locale. Queues resulted if the customers exceeded the capacity of the locale. Customers were not allowed to balk from a queue. Finally, the duration of each activity was generated randomly for each simulated customer.

Figure 3 and figure 4 show the progression of patrons through the simulated neighborhood. In the base case (figure 3), the restaurants are operating at full capacity on Fridays. For the after case, shown in figure 4, the Thursday peaks look like the Friday peaks from figure 3, while the Friday after crowds are even worse. The waiting time for a table climbs from 12 minutes maximum in the base case, to almost 30 minutes after. This leads to a spread of the peak period, even though the starting times for the for each of the four different kinds of activity sequences are drawn from the same set of distributions both before and after.

The purpose of this implementation was to test the ideas about specifying an abstract simulation framework. Coding up this simulation in C++ showed that it was possible to create random durations for activities which required some back-and-forth communication between agents and locales. Further, it showed that transportation facilities could be included within the concept of a locale, and that doing so resulted in a fairly natural simulation.

The primary lesson learned was that the logic coordinating the communication between objects could be simplified tremendously by defining an activity manager class. This capability is indispensable in any sort of simulation of group cooperation or constraints, since the coordination of messages between the many parties involved in an activity becomes very complex. The other lesson learned was that the graphical interfaces and statistical manipulation tools provided by published tools are difficult to reproduce. This is why we decided to explore using the Swarm simulation library.

5.2 Simulating evolutionary adaptation of activity schedules

This example explores the use of an activity manager class to implement a generalized model of activity engagement using the negotiation paradigm discussed in section 3.4. The purpose of this simulation is to lay the groundwork for the implementation of more advanced simulations including the representative activity pattern and intelligent actor approaches discussed in sections 3.1–3.3.

This example application was programmed using the Swarm simulation libraries (Langton et al., 1999). Swarm was developed by researchers at the Santa Fe Institute to facilitate programming concurrent simulations in an explicitly repeatable manner. A Swarm application requires the programmer to embed the objects of the simulation within a swarm. The objects of the swarm simultaneously interact with each other and with their environment in a controlled, repeatable way. Aside from the convenience of using pre-programmed elements such as plotters and data collectors, the primary advantage of using the Swarm libraries is that the resulting simulation makes correct assumptions about how to handle time.

For this example, we have implemented our generic activity simulation framework using the Swarm libraries. The specific activity-based details fleshing out the generic simulation are that a person generally follows a schedule of activities each day, but must also be able to adapt to an unpredictable environment. The schedule specifies the starting times, expected durations,

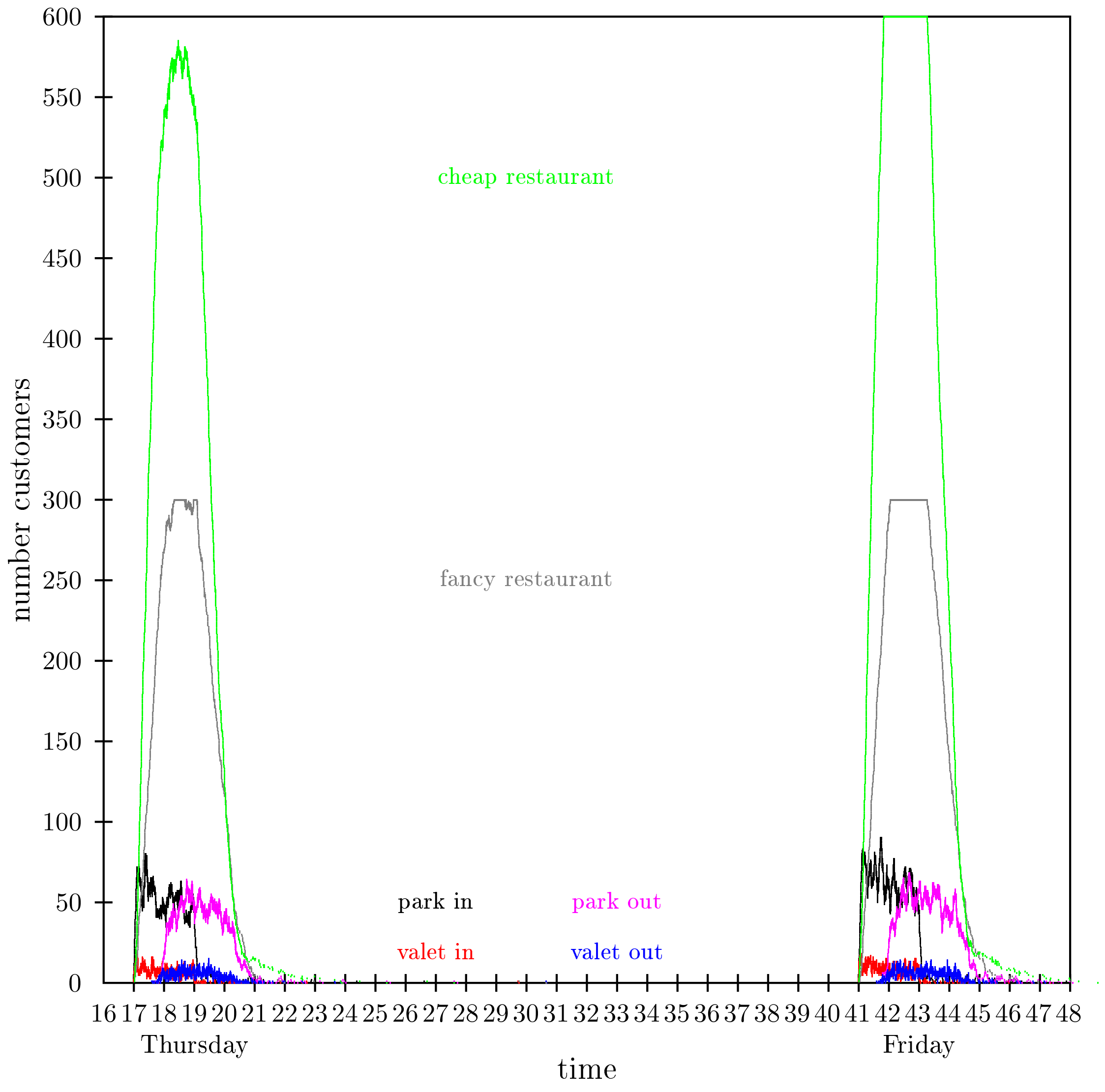

Figure 3: Customer loadings at each of the simulation locations, base case.

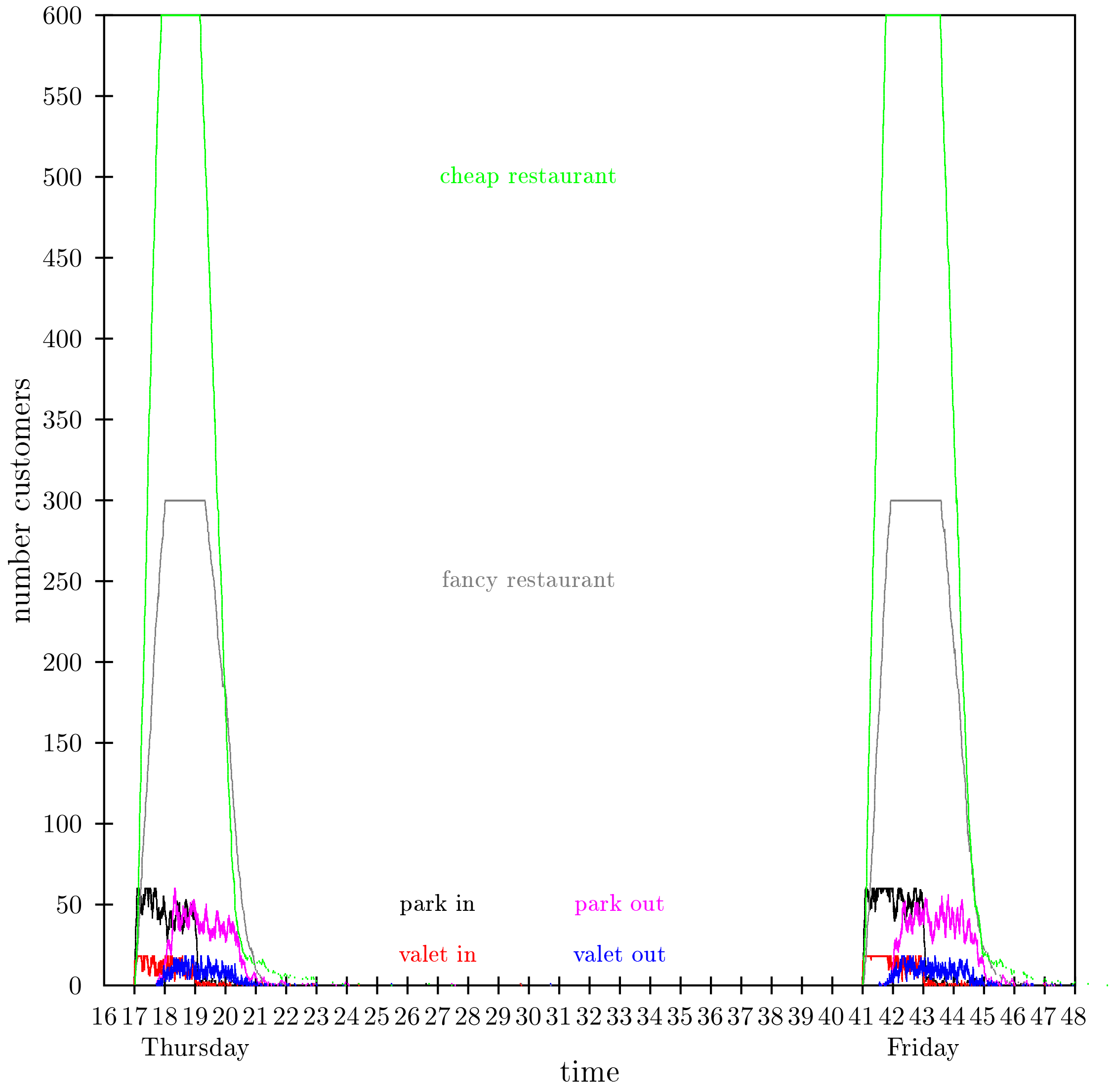

Figure 4: Customer loadings at each of the simulation locations after removal of the expressway, with parking delays and a shift to valet parking services.

minimum durations, and locale requirements for each activity the person should ideally perform throughout the day. Details such as travel times and locale capacity limitations which might cause queuing delays are not known to the person in advance, and therefore the scheduled times are only ideal times, not the actual times that will happen as the simulation unfolds. The locale requirements are sometimes very specific, e.g., specifying a home or work locale that may be used several times in the schedule. Other activities in the schedule require a locale that meets only generic restrictions, such as a any restaurant or grocery store.

The person-agents are randomly assigned one of three generic schedules weakly representing workers, primary care givers, and students, and drive the simulation by selecting activities from their schedule and attempting to perform them. The activity selection rule is straightforward. The person simply looks to see what activity on the schedule has the next starting time. The agent then tries to find a locale in the environment that can service the activity. Locales are chosen myopically, without explicitly considering how the choices will effect later activities. This shortcoming is partially overcome by an adaptation procedure we will discuss in a moment.

To speed the implementation, we generated a random locale environment, based on a uniform potential distribution of locations over a square simulation city. Each locale is given a time-dependent resource set that specifies its serving capacity for five types of activities: in-home, work, meal, maintenance, and discretionary. Travel is a special type of activity that is only serviced by a capacity-sensitive "transportation system" locale. The randomly generated city is stored in a standard database format, and is accessed via interfaces that can easily be implemented by a real land-use database or geographic information system.

The randomness of the environment means that an agent's abstract schedule may not be implementable. A person will omit an activity when the present locale does not service the activity, and there is no locale close enough to travel to and still meet the minimum duration requirement. In this case, the agent will start planning travel for the next activity in the schedule.

We implemented the simple activity-behavior hypothesis that missing scheduled activities is undesirable. In this example, agents adapt their schedules using simple evolutionary rules to adjust the desired start time and duration of the activities in their schedule after each day. The rules maintained the sequencing of the schedule. However, the locations and timings were allowed to vary. Agents revised their daily schedules based on selecting the past scheduled that produced the highest "activity success ratio," the ratio of time actually spent in each activity to the time in the schedule. This is a proxy for goodness of fit between the behavior and the schedule.

Figure 5 shows the evolution of schedule fit versus time in the simulation using this simple evolutionary behavior. The agents were rapidly successful in adapting their schedules to fit their particular environmental constraints. The example, however, imposed only location-based constraints on behavior. The addition of more complex interpersonal coupling and resource constraints will require a more complex evolution strategy.

This implementation benefited greatly from the knowledge gained from the earlier example discussed in section 5.1. In particular, an activity class was used to negotiate activity engagement between individuals and locales. This simplified inter-object communication by centralizing management of the activity. In addition, travel was simulated explicitly as activities negotiated with a transportation system locale. This advance, coupled with the activity class will lead to easier implementation of travel resource sharing in future simulations. It is easy to envision an activity that negotiates not only between a single agent on the transportation system locale, but also with other agents who may share resources to engage in the travel activity.

The Swarm simulation library proved to be a flexible tool for implementing the simulation. In addition to neatly encapsulating event-based simulation concepts, the library provides a significant set of utilities for supplemental analysis of simulation data both during and after the simulation. When combined with the important advantage of temporal correctness, the Swarm simulation library was deemed far superior to implementing the simulation environment from scratch in C++.

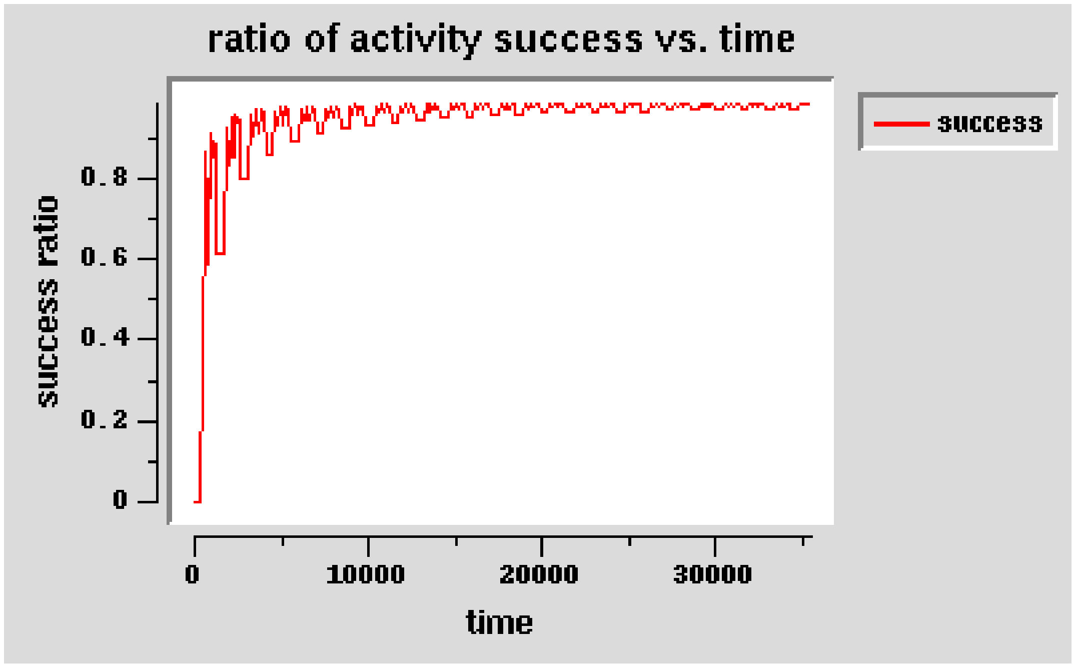

Figure 5: Evolution of population schedule completion success

6 Conclusion, applications, and directions for future research

The goal of this research has been to define general concepts for operationalizing activity-based transportation models. The complexity inherent in modeling human behavior makes comprehensive analytical analysis difficult. As a result, we see simulation strategies as central to further research efforts. Toward this end, we have outlined an event-based simulation abstraction centered around the concept that activities are negotiated between agents and locales. We have shown this to be a flexible concept that can accommodate several distinct types of activity analysis, and have operationalized it in two separate domains.

The work presented here has many possible practical and research applications. The most important point is that we have shown that our abstraction is flexible enough to handle a wide variety of different approaches to activity simulation. In other words, the details of a particular researcher's or practitioner's approach, while very important in and of themselves, do not place a significant strain on the core abstractions presented here. This means that software tools may be developed which support the abstract activity modeling concepts, and which can be applied to the specific details of any particular approach. The availability of a high quality set of simulation and modeling tools will speed up advances in the state of the art, and make activity based travel analysis more accessible to practitioners.

References

Borgers, A., Hofman, F., Ponjé, M. and Timmermans, H. J. P. (1998). Towards a conjoint-based, context-dependent model of task allocation in activity settings: some numerical experiments, 77th *annual meeting of the Transportation Research Board*, TRB, Washington, D. C.

Goicoechea, A., Hansen, D. R. and Duckstein, L. (1982). *Multiobjective Decision Analysis with Engineering and Business Applications*, John Wiley.

Hägerstrand, T. (1970). What about people in regional science?, *Papers of the Regional Science Association* **24**: 7–21.

Kelton, W. D., Sadowski, R. P. and Sadowski, D. A. (1998). *Simulation with Arena*, WCB/McGraw-hill, Boston, Massachusetts.

Langton, C., Burkhart, R., Daniels, M., Jojic, V. and Lancaster, A. (1999). The Swarm simulation system. http://www.santafe.edu/projects/swarm/.

M^{c}Nally, M. G. (1990). An activity-based microsimulation model for travel demand forecasting, *in* D. F. Ettema and H. J. P. Timmermans (eds), *Activity-based approaches to travel analysis*, Pergamon, Elsevier Science, Oxford, U.K., chapter 2.

M^{c}Nally, M. G. (1998). Activity-based forecasting models integrating gis, *Geographical Systems* **5**: 163–187.

Recker, W. W. (1995). The household activity pattern problem: General formulation and solution, *Transportation Research B* **29B**(1): 61–77.

Recker, W. W., M^{c}Nally, M. G. and Root, G. S. (1986). A model of complex travel behavior: Part I—Theoretical development, *Transportation Research A* **20A**(4): 307–318.

Saaty, T. L. (1994). Highlights and critical points in the theory and application of the Analytic Hierarchy Process, *European Journal of Operational Research* **74**: 426–447.