Evaluating JACK Sim for agent-based modelling of pedestrians

Nicole Ronald, Leon Sterling, and Michael Kirley
Department of Computer Science and Software Engineering
The University of Melbourne, Victoria 3010, Australia
E-mail: {naron, leon, mkirley}@csse.unimelb.edu.au

Abstract

In this paper, we describe the use of an agent-based simulation language, JACK Sim. JACK Sim is one of several add-ons for the agent language JACK, which is based on the belief-desire-intention architecture. The implementation of a prototype model of pedestrian wayfinding behaviour is described and the prototype is then evaluated using software engineering quality principles. A recent addition to the simulation language world, JACK Sim solves several problems encountered with using JACK alone for certain simulations and provides a solid framework for creating discrete-event simulations with BDI agent technology.

1. Introduction

Agent-based simulation is increasingly popular for modelling complex systems in diverse fields such as social studies, management, economics, logistics, and transport. These models can have a variety of purposes from providing forecasts to exploring behaviour.

Models can be implemented in three ways [14]. A simulator – a package designed to simulate a particular system requiring no programming knowledge – may be used. Alternatively, the model may be written from scratch using a general purpose language such as Java or .NET. Another method is to use a simulation language like JACK Sim: a language that has been designed for writing models.

The JACK Intelligent Agents package allows the user to create belief-desire-intention (BDI) agents using syntactic additions to the Java programming language. The JACK Sim add-on, which is one of several add-ons included in the package, provides more functionality to the JACK language for creating and running repeatable discrete event simulations. BDI agents have been used to model human behaviour and decision-making, which are a large focus of many agent-based simulations.

This paper presents the implementation and evaluation of an agent-based model of pedestrian behaviour using JACK Sim. An overview of the BDI architecture and JACK is provided. The design and implementation of a prototype is described and an evaluation of the model development using software engineering quality principles is discussed. Finally, some ideas for further work are presented.

2. Background

Two of the most well-known agent-based simulation languages are Repast and Swarm. Repast [17] is based on discrete-event modelling and is mostly used for explorative models. It has three “concrete” representations in Java, .NET and Python. Swarm [16] is similar and consists of libraries for Objective C and Java. This paper investigates another simulation language, JACK, which is based on the belief-desire-intention agent architecture.

2.1. Belief, Desire and Intention

The belief-desire-intention (BDI) architecture is based on philosophy, in particular the work of Bratman [9]. The philosophical component of BDI is based upon practical reasoning. Practical reasoning is defined as reasoning toward actions, as opposed to theoretical reasoning, which is reasoning about beliefs. Practical reasoning can be broken down further into two activities: deliberation (deciding what goals to achieve) and means-end reasoning (how to achieve a goal) [24]. Another nice feature of the BDI architecture is the ability to act in both a reactive and proactive manner, however there is a danger of being too reactive or too proactive.

The key concepts in the BDI architecture are:

- beliefs: what I know or don’t know about the world;
- desires: what I want to do;
- intentions: how I plan to do what I want to do.

Desires are defined as a “potential influencer of conduct”, and other desires need to be taken into account before anything is committed to. That is, once I have decided to try and achieve something, it becomes an intention and I
make a reasonable attempt to achieve that intention. However, the semantics of the concepts within the BDI architecture vary between philosophers and also between computer scientists.

2.2. JACK

JACK is based on the BDI architecture and was purpose-built for simulations, in particular defence simulations. The aim of the package was to develop a stable, lightweight and practical agent-based programming language that would not be superseded quickly and would facilitate further research. It is based on Java with a few syntactic extensions, and when compiled compiles to Java code [10]. Java was chosen due to its widespread availability and acceptance. As the JACK files are compiled into Java before execution, normal Java statements can be embedded in JACK files.

JACK has been used for several applications, mainly within defence, however it has a strong reputation worldwide both in research and industry [1]. The intended users of JACK are people with knowledge of agent-based applications, concurrent object-oriented programming and software engineering [10].

2.3. JACK Sim

JACK supplies several add-on packages, such as JACK Teams which is popular for developing teams of agents. Another package is JACK Sim [2], which adds more simulation functionality to JACK.

Discrete-event simulation is used for models where the events in the system drive the model. Queues are used to keep track of the events that will happen in the near future. There are commonly three world views of a simulation [5]:

- **event**, which focuses on the events and their effects on a system;
- **process**, which models the entities/objects in a system and their lifecycles; and
- **activity**, which focuses on the activities in a system and when they begin and end.

JACK Sim provides a new world view: the BDI world view [2], where behaviours are defined within agents and the simulation is driven by the JACK model. It appears to be a combination of the process model (agents have a defined lifecycle) and the event model (the agents perceive internal and external events, which cause them to execute plans).

Applications built using JACK Sim use a single clock for the entire application, which is controlled by a TimeSource agent. The agents in the application are allowed to execute until they are all blocked. Each agent is managed by a TimeDispatcher agent. The TimeDispatcher’s role is to let each agent execute and then let the TimeSource know the next time one of its agents is ready to execute again. The TimeSource advances the clock to the earliest time it receives from the TimeDispatchers and another round of execution begins. The clock stops when there is no more activities to be scheduled. This package effectively adds discrete-event simulation behaviour to JACK. Not all agents in the simulation have to be controlled by a TimeDispatcher.

There are three phases to a JACK Sim program. During Creation, the agents are created, initialised, and registered with a TimeSource agent. Setup allows for developer-designed setup to take place. Finally, the Execution phase steps through time until there are no events left to execute. The start of the Setup phase and the start and the end of the execution phase are signalled with an event, which is received by all agents controlled by a TimeDispatcher. The process is shown in figure 1.

3. Prototype development of a pedestrian model

The purpose of this prototype, developed with JACK Sim, is to explore the development of an agent-based model of pedestrian behaviour, using agent-oriented design methodologies and an agent-based/simulation lan-
guage. The focus of this model will be more on performance and usability rather than behaviours exhibited by the model, however a brief discussion of wayfinding behaviour follows.

3.1. Scenario

Wayfinding is the process of finding a path. This differs from navigation, which is following a predefined path [23]. It is difficult to define a generic wayfinding process as it depends on the purpose of the trip. Trips that are repeated frequently, such as to work or to school, are refined quickly. These trips usually follow the shortest path and minimise the amount of en-route decision making. Weston and Handy [23] refers to these as commuting trips: regular travel between two points. The wayfinding for leisure or recreation trips consists of search and exploration.

There are three methods of forming a route [8]. Simultaneous planning involves selecting the whole path from the choice set of all complete paths from origin to destination. In this situation the traveller does not change their path during the trip. Sequential and hierarchical choice involve decision points during the trip. Sequential involves selecting the next link independently of the last link followed, but hierarchical takes into account the past. These are the extreme choice situations, however they are common in reality.

If a traveller experiences inferior conditions during the trip than what they expected, they may change their route. This is called adaptive route choice and is usually experienced when the traveller is under time pressure.

3.2. Model development

3.2.1. Design Although there are several agent-oriented design methodologies that could be used for this system (e.g. Tropos, Gaia/ROADMAP) [7], Prometheus [18] was chosen because of its maturity and because the concepts used in the methodology tie in with the concepts used in JACK Intelligent Agents, our chosen implementation language.

Prometheus consists of three design phases. The system specification phase contains activities relating to identifying functionalities, inputs, outputs and shared data sources. The agents required and their interaction is determined in the architectural design phase. The internals of agents are designed in the detailed design phase. The resulting design is a combination of forms and diagrams, which clearly describe the percepts, action, environment, agents, capabilities and plans in the system.

Although not tied to any implementation, the Prometheus methodology aligns neatly with JACK Intelligent Agents. By reusing elements of OO methodologies, it is expected that those familiar with UML will not have difficulty in adapting to agent-oriented design [12].

Prometheus is “intended to be useful and usable by industry developers” [18]. The creators quantify this as being well-described and complete so it can be used effectively. Providing and supporting effective tools – such as the Prometheus Design Tool – is also a criteria.

3.2.2. Implementation The first attempt at developing an agent-based model of pedestrian behaviour was undertaken using Prometheus and JACK [20]. A prototype model was constructed using JACK Intelligent Agents which involved agents entering a sports precinct and moving towards a stadium. Several “distractions” were located on the way to the stadium, such as food stands and street performers. In the first version of the prototype, the entire model architecture (user-vehicle-environment) was implemented in JACK to avoid complex interfacing.

Several issues arose from this prototype:

- It is straightforward to implement goal-directed behaviour, such as moving towards the stadium. The belief system, however, is similar to facts in a logic programming language such as Prolog and does not handle complex beliefs well. For example, it is difficult to represent an environment in detail using JACK belief sets. Ideally an interface to the environment should be developed and then any environment format (e.g. graph, cells, shapes) could be used behind that.

- The decision-making used in BDI and in JACK cannot elegantly handle continuous events, such as stepping. It is also difficult to define the subconscious decisions behind walking. Therefore the vehicle model in our architecture would be better suited to an object representation rather than an agent one.

- An advantage of agent-based technology is that agents are capable of doing several things concurrently without trouble e.g. walking and looking about. So, if you are walking to the post office and you see a shop that has a sale, you continue walking as you make a decision whether to detour or not. If you decide to keep going, you continue with your existing walking plan. However, if you decide to detour, you stop your current walking plan and construct another to get to the sale. JACK had problems interrupting one plan before starting another and in our model, agents sometimes found themselves in two environmental locations at once. A solution is to increase the lookahead of the pedestrian agent, so that they make decisions and construct new plans earlier.

- The main issue with using an agent-oriented methodology is that it designs the simulation only i.e., what the agents are doing. It cannot design the core of the
simulation i.e., how the clock will tick over, the graphical user interfaces required to set up the simulation, the methods to collect outputs. Therefore Prometheus needs to be combined with another methodology to design the whole of the simulation.

### 3.3. The JACK Sim model

The JACK Sim packages solves the plan interrupting problem from using JACK alone. In JACK Sim the execution time of the plan is irrelevant – the clock is stopped until a block is reached in the plan.

For the second version using JACK Sim, a prototype was constructed of pedestrians moving around a network, represented as a directed graph implemented in pure Java. Entry and exit points to the environment and the number of people using them was specified, as well as locations within the model that a certain proportion of the model population needed to visit. No time constraints on the pedestrians’ planning were implemented.

Two pedestrian flavours were implemented based on route choice planning theory described by [8]. **Familiar pedestrians** knew their route through the environment and are based on the simultaneous planning theory described in 3.1. It was most likely the shortest and they were unlikely to deviate from it. For this model, each pedestrian retrieved their route from a central database containing the shortest routes between two points. This could be enhanced by randomly altering the path returned. Some *adaptive planning* ability was also added, so if the pedestrian found they were going too slowly, they would replan their route based on real-time time estimates between their current location and their destination.

**Unfamiliar pedestrians** have no prior knowledge of the environment and are based on the sequential route planning described in section 3.1. They are given an origin, a destination, and possibly locations in-between to visit. They select a link to travel on, and at the end of that link select the next link. The next link is selected either randomly or as provided by the central database.

Reaching a node is an event for the TimeSource agent. Every time an agent reaches a node, it calculates its next route and informs the Environment agent. The Environment agent calculates how long the agent will take to travel along the link based on the speed and the current density of the link. The calculation is based on the speed-density curves in [3]. The curve is based on the space per pedestrian or inverse density, as opposed to the number of pedestrians in an area. The equation for this curve was estimated in equation 1.

\[
v = 1.4 - 0.35 \frac{\text{space}}{\text{m/s}} \quad (1)
\]

Some runs of the prototype were made in order to explore the behaviours. The familiar pedestrians, as expected, all took the same paths through the environment. The unfamiliar pedestrians introduced some variability to the model. The adaptive behaviour was raised when the agent dropped below a certain speed as expected, however due to the number and heterogeneity of the familiar pedestrians, the real-time path was rarely different to the original path based on distance. An example of the graphical output is shown in figure 2.

An issue with the current implementation is that the time spent on the link cannot be changed mid-link, for example, if crowds begin to clear and the pedestrian speeds up. A solution to this is to use smaller links.

### 4. Prototype evaluation and discussion

Software quality standards measure functionality, reliability, usability, efficiency, maintainability, and portability characteristics to determine the quality of a finished product [22].

![Figure 2. A snapshot of the environment. The square nodes are entry/exit nodes. The width and colour of the paths show a higher density of traffic.](image-url)
For this project, only a prototype was developed. Little research has been undertaken into evaluating tools, especially prototypes. A series of articles in the late 1990s in *Software Engineering Notes* looked at different methods for qualitative evaluations. The case study methodology [13] will be adapted for this evaluation. This involves testing a method or tool on a real project and developing a lightweight assessment of the method or tool. However, only one evaluator will be used, therefore removing the variable of different evaluators.

Several lists of desirable features for models have been published. Axelrod [4] described validity, usability, and extensibility as goals of a simulation model. Law and Kelton [14] describes a wider range of features. Finally, Tobias and Hofmann [21] evaluated several libraries for ABSS and some of their criteria will be incorporated.

Reliability is excluded from this set of criteria as it relates to product performance, which is inappropriate for prototypes. However, some performance criteria was included in the efficiency criteria.

### 4.1. Functionality

Functionality is “the degree to which the software satisfies stated needs” [19]. The following criteria were investigate:

- reasonable number of agents generated ([14] prefers maximum model size);
- presence of an appropriate agent communication process;
- ease of generating agents;
- ease of generating networks;
- whether spatial location of agents is catered for;
- ease of input;
- provision for statistical output [14];
- support provided for modelling and simulation, such as timing and repeatable experiment functionalities.

Functionality usually refers to the end product, not prototypes. However, some assumptions about functionality of the end product will be included here.

The model appeared to handle 12000 agents in a simple environment, however the model was running at a fraction of real-time. Approximately 7000 agents could be created for the model to run close to real-time.

The JACK platform provides communication functionality. However, in order to use it, an agent needs to know the name of the agent they want to communicate with. Unlike other toolkits such as JADE, no WhitePages or YellowPages agent is provided. However, the developer could implement one if required. No broadcast capability is provided either, requiring a dedicated agent to be developed. This would be useful to communicate with a group of pedestrians on a particular link.

In JACK, agents are easy to generate – the agent is constructed and then a user-defined start method is called. In JACK Sim, the agents are created from an input file. The only parameters allowed are the name and type of the agent. In order to read in other initialisation values, the developer must use JACOB, an object modelling tool provided with JACK. The number of agents is also explicit. For this simulation, a program was required that created a file with the required number of agents to be read in as input.

JACK does not provide explicit functionality for keeping track of an agent’s spatial location.

The input and output are developed by the user. For this prototype, the input was created in XML format. As Java provides XML support as part of its API, it is easy to set up. An additional feature of this support is the ability to validate a file against a DTD. From our experience, missing fields are often a problem with complex input files. The XML file itself can be edited with a text editor or an XML editor. The text-formatted output is used by a separate Java application to visualise the environment over time. If a standard DTD was introduced for transport output, then the model could output in the correct format for a dedicated graphical program to visualise.

A GUI was created that showed a static picture of the network, but did not allow editing of the network. The latter functionality would be a requirement for a final system. The ability to read from a CAD drawing or GIS database would also be a requirement.

Timing and clocks are explicit in JACK and JACK Sim. In JACK Sim, the timing is more transparent once the Time agents are defined. Both event-based timing and real-time timing are available, however due to the temporal duration experienced in transport models, real-time clocks are inappropriate and were not used. JACK Sim creates repeatable simulations and does not use seeds, so the developer must construct seeds explicitly for variable simulations.

### 4.2. Usability

Understandability, learnability, and operability are the subattributes of the usability criteria [22].

Understandability looks at the support provided and required in order to use the product, in particular the presence of appropriate documentation and other forms of support (eg. mailing lists) for the intended user base [14]. This involves identifying the target audience.

Learnability looks at the time required to understand principles and to learn the product, using the provided resources and other materials.
Operability looks at the programming/software engineering knowledge required and the software required to operate the product. This will involve a hypothetical about the appearance of the final product for the prototypes.

To use JACK, the developer needs to know Java and have a good understanding of agent concepts. Agent-Oriented Software provides documentation for JACK and JACK Sim and simple tutorials for JACK. A mailing list for JACK is also hosted and regularly used by AOS staff. Their intended audience is accurate, as this application would be difficult to use for non-software engineers with little knowledge of agents. Some Javadocs for the API are provided.

For Prometheus, a basic understanding of agent concepts and and understanding of software design is required. Padgham and Winikoff [18] is a valuable resource and the Prometheus Design Tool is useful. It is regularly maintained and updated.

To use Prometheus, a JRE is required which is common on most systems. For JACK, a JDE and the JACK API is required. JACK is available free for trial and academic use, however for other uses a commercial licence is required.

For the BDI prototype, two user interfaces were provided. The first displayed an overview of the environment. This would be enhanced so that the environment could be edited. The second displayed the output of the simulation, as shown in figure 2. For ease of use, another user interface that assists in setting up and running the model would be required.

For ease of development, a graphical user interface can be used to create JACK programs. However, this needs to be used from the start of the project, as it is not possible to import existing code.

4.4. Maintainability

Axelrod [4] mentioned extensibility as a goal for simulation models, because this allows other researchers to adapt the model to explore new features and variations. Modularity and extensibility will be the main focus of this criteria.

Two features of object-oriented programming is that it is modular and extensible [19]. In Java, classes can be organised into packages that contain all the objects and methods required for a particular task. The package can then be extended as necessary, as long as the public operation contract is not broken. Java interfaces are useful for defining the public operations required from an object.

JACK inherits these properties from Java. It also adds the concept of capabilities, which define a set of events, plans and beliefsets that are required to perform a task.

4.5. Portability

For our evaluation, portability will cover the ability to use the product on different platforms with no change in results. This is important for repeatable simulations. L’Ecuyer [15] defines repeatability as “the ability to replicate exactly the same sequence of random numbers”. This assists in the verification step and also it ensures that different scenarios can be run with the same parameters.

A key feature of Java is its portability [11]. The JACK framework is also available for several platforms.

The random number generator provided in the Java API uses a linear congruential pseudorandom number generator and will return the same series of numbers given the same seed. This is important for repeatable experiments.

The JACK Sim framework is guaranteed to be repeatable for one agent only. For more than one agent, multiple threads are required and the order of events cannot be guaranteed. However, from the runs undertaken for the efficiency discussion, it appeared that the system was producing the same results for the same seeds.

5. Conclusions and further work

In this paper, we presented JACK Sim, a recently-developed add-on to the JACK Intelligent Agents language. We presented a prototype of a pedestrian model using JACK Sim and evaluated the prototype using criteria based on software engineering quality principles.

The JACK Sim add-on solves several issues with the standard JACK platform. In particular, the discrete-event capability provides much more control over time to the developer. The ability to quickly change the model parameters and agents by changing the input file (rather than source code) is also useful.
Future work involves extending the model using more of the theories from [8]. Validation against the real world of these behaviours is also required.

References