Dhirendra Singh, Lin Padgham, Kai Nagel

Using MATSim as a Component in Dynamic Agent-Based Micro-Simulations

Conference paper | Accepted manuscript (Postprint)
This version is available at https://doi.org/10.14279/depositonce-9447

Singh, Dhirendra; Padgham, Lin; Nagel, Kai (2019): Using MATSim as a Component in Dynamic Agent-Based Micro-Simulations. Presented at 7th International Workshop on Engineering Multi-Agent Systems (EMAS 2019).

Terms of Use
Copyright applies. A non-exclusive, non-transferable and limited right to use is granted. This document is intended solely for personal, non-commercial use.

WISSEN IM ZENTRUM UNIVERSITÄTSBIBLIOTHEK
Technische Universität Berlin

Rafael H. Bordini, Louise A. Dennis and Yves Lespérance (eds)
Pre-proceedings of the 7th International Workshop on Engineering Multi-Agent Systems (EMAS 2019)
13th-14th May 2019, Montreal, Canada
cgi.csc.liv.ac.uk/~lad/emas2019/accepted/EMAS2019_paper_23.pdf

Using MATSim as a Component in Dynamic Agent-Based Micro-Simulations

Dhirendra Singh[1] and Lin Padgham[1] Kai Nagel[2]

[1] RMIT University, Melbourne, Australia
{dhirendra.singh,lin.padgham}@rmit.edu.au
[2] Technical University, Berlin, Germany kai.nagel@tu-berlin.de

Abstract. This paper discusses use of the widely used transport simulator, MATSim, as one component in a large complex agent based microsimulation where dynamic changes in the environment require the agents to be reactive as well as goal directed. We describe a number of refinements to MATSim that have been made to facilitate its use within our deployed wildfire evacuation applications, as well as some tools that have been developed which complement MATSim. All code is freely available under open source licenses. As applications increasingly require complex microsimulations, with many aspects, it is important to use existing software where possible. However most simulation systems, like MATSim, have been developed as standalone systems. We identify ways that MATSim has needed to be extended or modified in order for it to be used as a component in a larger whole. The paper provides details that will be useful for anyone wanting to use MATSim within their specific application.

Keywords: MATSim · Belief-Desire-Intention · BDI · Agent-Based Simulation

1 Introduction

This paper focusses on the use of the widely used MATSim (Multi-Agent Transport Simulation) [9] traffic simulator as a component in large scale agent based micro-simulations, where, as is increasingly relevant, it is often important to make use of detailed real world data (e.g. [6, 17]. We use examples and motivations from deployed applications in the emergency evacuation domain. The specific contributions are extensions to MATSim and some additional tools for use with MATSim. The specific aspects are: the ability to control MATSim externally, making it suitable for inclusion as a component; a standardised API to edit MATSim plans, routes and trips; a mechanism for dynamically controlling routing in a variety of ways; population initialisation support; and a discussion of principles for designing the BDI component of the agent and its interaction with the MATSim component. The unifying aspect of these contributions was their need in a family of applications in the evacuation domain, although they are also more widely applicable. One of these evacuation planning applications can be viewed at tiny.cc/bushfire-sim. The others are not publicly available.

Originally MATSim was developed for finding traffic equilibrium as individual agents adapt their travel behaviour to a specified transport infrastructure, based on their individual activity patterns. The system is initialised with a set of agents, having various attributes, each having an "activity plan" which specifies the location and duration of various activities throughout the day. The system then determines the best route between activities at suitable times, embellishing the plans with specific detailed routes for each trip.[3] The execution simply steps through these plans. If as a result of congestion or other issues travel between destinations takes a longer or shorter time than expected, this is recorded and plans are scored accordingly. At the end of each one day simulation plans are reviewed and some poorly rated ones are modified using genetic algorithm style techniques, until eventually after some number of iterations a stable state is reached. This approach is very successful for assessing the impact of proposed new infrastructure in a city or area where there is data concerning the current behaviours. However it is not suitable for applications where decisions need to be made reactively, based directly on a dynamic situation. Two examples of such situations are evacuation simulations and simulations involving taxis which must respond to the evolving environment. In recent years there has been a focus on modifying the "Mobsim" component of MATSim to accommodate this using what is called "within-day replanning" or "en-route replanning" [8, 4]. It is this aspect of MATSim which is considered in the current paper, considering only a single iteration of the agents over some time period

The BDI-MATSim system [11] is one approach to supporting the ability of MATSim agents to be reactive to a dynamic situation. It builds on the infrastructure developed for integrating any existing cognitive system (as long as it relies on percepts and actions) with any agent-based model that fills certain requirements [16]. The integration facilitates "within-day replanning" in MATSim by allowing agents to proactively make decisions to change their original plan, depending on both environmental situations and agent goals. Conceptually, the "brain" of a MATSim agent is modelled in the BDI system (as a BDI agent) while the "body" remains inside MATSim. The communication between these agent counterparts is defined based on standard agent concepts, *percepts* and *actions*. A MATSim agent sends percepts to the BDI counterpart, which conducts high-level reasoning and issues a (BDI) *action* for the MATSim agent to execute. *Percepts* from the MATSim counterpart agent can be either information about its own state (e.g. location), or an observation from the MATSim environment. Basically, a BDI action modifies the travel plan of a MATSim agent using low-level MATSim functions. MATSim controls the simulation, integrating the event-based BDI system, by passing control to it at the end of one or more MATSim time step(s). Control is returned when reasoning about actions for the next step concludes. The evacuation applications which have motivated and used

[3] A *plan* encompasses *activities*, *trips* which contain the (possibly multi-modal) movement between activities, and *routes* which are the detailed road/path segments to be traversed by a vehicle/person.

the extensions and tools described in this paper have all used the BDI-MATSim system.

2 MATSim as a component

In developing large and complex simulations it is essential to be able to incorporate components which are themselves large and complex pieces of software. These must all work together – and preferably continue to work together as components are further modified and developed.

We use the architecture shown in Figure 1 where a controller pauses and continues the execution of components as well as providing a mechanism for data sharing. This is conceptually similar to the HLA [7] standard, although, unlike HLA, multiple models can represent aspects of the same conceptual agents at the same time, as long as aspects are managed to ensure consistency, as they are in the BDI-ABM integrated framework described in [16].

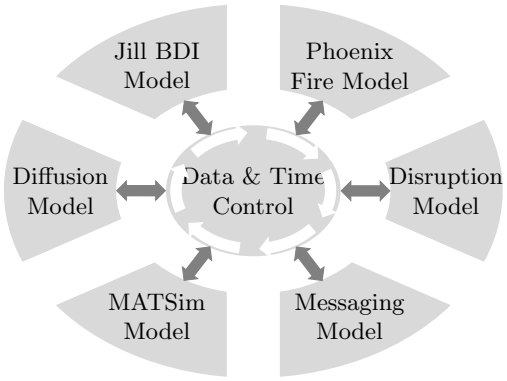

Fig. 1: Architecture of component based simulation

If there is a producer consumer relationship with respect to data produced and used within the same simulation time step, then the controller must sequence the component executions appropriately. The model cannot deal with circular relationships between components within a single timestep, only with pipeline relationships. The issues of shared resource management as handled by frameworks like OpenSim [15] that were built for integrating existing models do not apply here. Our framework supports models that run on different size timesteps as well as variable time steps such as discrete event models. Data exchange is based on a publish/subcribe scheme whereby a model is called on one of two events: to handle incoming data from other models that it is subscribed to, or to publish its own data at a frequency (fixed or variable) under its full control.

Figure 2 shows in more detail the MATSim and BDI components of figure 1, showing the original MATSim modules, the pre-existing extensions which we build on, and the new additions that are described in this paper.

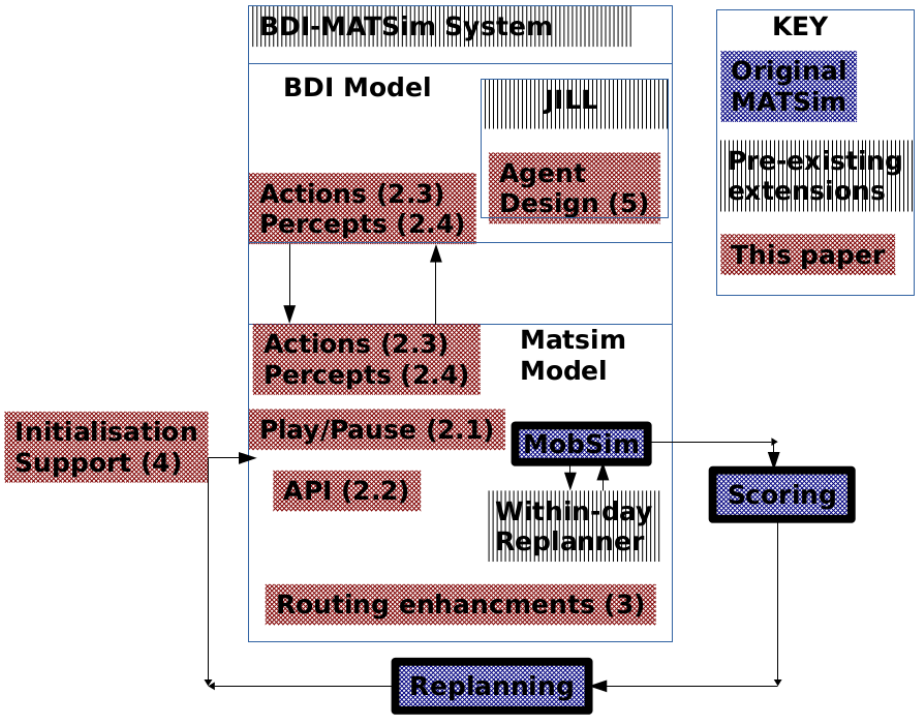

Fig. 2: Details of Architecture

For MATSim to operate as a component within this framework, rather than as a standalone application, it is necessary to allow stopping and starting from an external controller (unlike [11] where all control was with MATSim), as well as options for MATSim to receive and provide data. To support additional functionality likely to be needed for new applications it is also desirable to have a principled API providing access to internal MATSim functionality.

2.1 External control of MATSim steps

A new facility called `PlayPauseSimulationControl` provides a `doStep(time)` function to continue the MATSim simulation forward up until `time` and return. This new play/pause API also ensures that the simulation clock of the controller is not tied to MATSim's simulation clock.

Algorithm 1 shows the high level simulation loop where MATSim is a component. Each model is first initialised (line 1), and registers with the central

controller of Figure 1 all data types it wishes to publish or subscribe to (line 2). We do this upfront, but models are free to register types also during the simulation conditional on some event. Then on every simulation loop iteration (line 3), the BDI model is called first (line 4) followed by MATSim so that new or dropped BDI actions passed via container `dataBDI` are handled by MATSim immediately in the same time step (line 7). BDI actions status' and percepts coming back from MATSim in `datABM` are handled by the BDI model in the next iteration of the loop. Other models are called as needed (line 8) and the entire simulation terminates (lines 5–6) when MATSim itself reaches the end of its own simulation.

```
   Result: Simulation completed
1  // initialise all models
2  // register ordered models with controller
3  while true do
       // invoke BDI model with incoming data from ABM
4      controller.publish(BDI_CONTROL, dataABM);
5      if MATSim has reached end of simulation then
6          break; // exit the loop
       // invoke MATSim with incoming data from BDI
7      controller.publish(ABM_CONTROL, dataBDI);
       // progress time & advance other models
8      controller.stepTime();
9  // terminate all models
```

Algorithm 1: Simulation loop for MATSim as a component

Data flow between models happens in several places in the simulation loop. Direct data passing from publisher models to subscriber models occurs in the controller's `stepTime()` function (line 8). We use this to directly feed, for instance, time-stamped fire shape data from the Phoenix Fire Model into MATSim to dynamically increase link penalties on road segments impacted by fire. Data flow between the BDI-MATSim coupling is managed more precisely by the controller as mentioned already, by routing the data from the models (`dataBDI` and `dataABM`) through the controller.

2.2 API to modify agent behaviour

Previously, while it was possible to modify MATSim agent behaviour in all sorts of ways, this involved editing into the internals of MATSim in functions that even if they were public, possibly should not have been, and were in danger of changing as MATSim developed. There are now three clearly specified classes allowing for

editing of plans (**EditPlans**[4]), routes (**EditRoutes**[5]) and trips (**EditTrips**[6]). These classes provide a clean interface for programming functions to change agent behaviour, as well as for querying MATSim regarding aspects of an agent's current plan. Plans are the highest level of abstraction and consist of a sequence of activities at specific locations, interleaved with trips between locations. A trip may have a variety of different modes, including car, public transport, and walking. A route is the specific set of links to be traversed within a trip. Table 1 shows the API functions for plan editing. Trip and route editing follow a similar pattern. Basically these are the typical insert/add/remove/modify methods that one knows from the Java List class, plus some helper methods that have to do with the data model.

Table 1: High level API for plan editing.

```
addActivityAtEnd (agent, activity, routingMode)
createFinalActivity (type, newLinkID)
findIndexOfRealActAfter (agent, index)
findRealActAfter (agent, index)
findRealActBefore (agent, index)
flushEverythingBeyondCurrent (agent)
getModeOfCurrentOrNextTrip (agent)
insertActivity (agent, index, activity)
isAtRealActivity (agent)
isRealActivity (agent, planElement)
removeActivity (agent, index, mode)
replaceActivity (agent, index, newAct)
rescheduleActivityEndtime (agent, index, newEndTime)
```

2.3 Adding BDI actions to MATSim

When a BDI action is sent to MATSim it performs the necessary changes to the agent's plan elements using suitable API functions (of section 2.2) and registers the BDI percept handlers that watch for MATSim event(s) relevant to that action. In particular there must always be some event(s) which indicate that the action has terminated – normally successfully, but possibly that it has failed. Information which needs to be provided to the BDI agent is then packaged up for transmission to the relevant BDI agent as a percept, possibly along with information that the BDI action has succeeded or failed.

An example reusable BDI action is `driveTo(args)` for which we provide a default action handler on the MATSim side, which on reception of the BDI action

[4] http://matsim.org/javadoc → matsim main → `EditPlans`
[5] http://matsim.org/javadoc → matsim main → `EditRoutes`
[6] http://matsim.org/javadoc → matsim main → `EditTrips`

(i) inserts a new activity immediately following the agent's current activity/leg, on the network link nearest to the coordinates given in `args`; (ii) optionally sets the end time of the current activity, if the agent is currently performing an activity, to some future time given in `args`; and (iii) registers an event handler for the MATSim `PersonArrivalEvent` event, which when triggered on the link of the newly inserted activity indicates the end of the `drive-to` action. All event monitors registered with the BDI action for an agent are removed when the action is removed upon success, failure, or abortion.

A particular application may introduce both new BDI actions and new percepts to be used by the BDI agents in their reasoning. This necessitates new application code to be added to MATSim to implement the percept handlers, and also to implement the MATSim realisation of the BDI action. This is done by registering an application specific action handler at initialisation. Existing BDI actions may also be extended by the relevance of new percepts.

2.4 Generating BDI percepts from MATSim

MATSim's mobility simulation, or mobsim, that is responsible for moving the agents around according to their plans, generates a stream of events that capture the movement of people between activities. This stream gives MATSim extensions a mechanism to plug in and listen to events of interest and perform their own computations as needed. The events that are of most interest to us are those that describe the start(end) of an activity (`ActivityStart(End)Event`), a person arriving at(departing from) the location of an activity (`PersonArrival(Departure)Event`), after(before) exiting(entering) their vehicle (`PersonLeaves(Enters)VehicleEvent`), a vehicle entering(exiting) traffic (`VehicleEnters(Leaves)TrafficEvent`), and a vehicle entering(leaving) a link in the MATSim road network (`LinkEnter(Exit)Event`).

Like [12], we use mobsim events to build percepts for BDI counterpart agents. As mentioned, when a BDI agent sends a new `driveTo(args)` BDI action, the action handler in MATSim registers an event listener for `PersonArrivalEvent`. As the simulation progresses, and the `PersonArrivalEvent` event is eventually triggered for that agent arriving on the destination link, the event handler code flags the BDI action as `passed` and also generates an `arrived` percept for the BDI agent.

It is also possible to add custom events based on the application's requirements. For the evacuation application, we defined two new events that are both relevant when the agent is engaged in a `drive-to` BDI-action. The event `AgentInCongestionEvent` flags the condition that the vehicle is stuck in traffic congestion, while the event `NextLinkBlockedEvent` is triggered if the vehicle is about to enter a link that is blocked, due to a road closure for instance. Consequently, code for responding to these custom events is registered as additional event handlers for the `drive-to` action which previously had only registered an arrival handler. These custom handlers generate the appropriate BDI percept information for passing back to the BDI agent as well as deciding if the BDI-action should potentially be deemed failed.

The custom `AgentInCongestionEvent` is computed on `LinkLeaveEvent` as

$$c_{k,n} = \begin{cases} 1 & \frac{t_{k,i} - t^*_{k,i}}{t^*_{k,i}} > w \\ 0 & otherwise \end{cases}$$

where $c_{k,i}$ is the congestion evaluation between some traversed link k and current link i, time $t_{k,i}$ is the recorded travel time for the route taken from k to i, and $t^*_{k,i}$ is the expected travel time if travelling at freespeed on that route. The constant w is the congestion tolerance threshold. Practically, we set a time period T for congestion evaluation and take the maximum permissible $t_{k,i}$ such that $t_{k,i} \leq T$. This makes congestion parameterisation somewhat more intuitive. For instance, $T = 300, w = 0.4$ means that an agent will consider itself to be stuck in congestion if over the last 5 minutes, the time delay in travelling the route from k to i was greater than 40% of the expected travel time for that route.

The `nextLinkBlocked` event is generated when the following link has freespeed close to zero as the intent is to prevent the agent from entering a blocked link where it might get stuck forever.

The architecture of Figure 1 supports data flow directly between models, as described in Section 2.1. Such incoming "data events" can therefore be used by the MATSim model to generate BDI percepts. For instance, on reception of updated fire(smoke) shape information from the Phoenix Fire model, we first query MATSim for a list of all agents that are within the polygon shape (plus some configurable buffer around the shape) at that timestep, and then generate `fire(smoke)-visual` BDI percepts for all those agents.

As different applications are developed with MATSim as one component, it is anticipated that a range of application specific percepts and percept handlers will be developed, some of which will be reusable across multiple applications. The `in-congestion` percept is one such addition which could be expected to be reused across applications. The smoke and fire percept and percept handler on the other hand is likely to be relevant only to applications in the bushfire domain. The mechansim of percept handlers and the way they can be linked to specific high level actions (BDI-actions) is a new/refined facility which supports the integration of MATSim into new application areas, combined with other components.

3 Flexible route planning

In many simulation applications the environment itself is dynamic. For example a bushfire progression creates smoke and fire while a traffic accident causes a road blockage. Often these environmental changes will require modifications to a simulation environment, even when they do not originate there (such as fire and smoke). In MATSim the key environmental component is the road network. Some dynamics of the environment already result in MATSim effects, such as congestion influencing the speed at which traffic moves through a link. There are

also existing options to make some dynamic modifications to the road network, such as modifying the speed and capacity of links. However, as we developed our evacuation applications we found that these aspects were insufficient for some situations. In particular we needed to have residents who were evacuating avoid driving into the fire, while still allowing emergency service vehicles to do so (to some extent).

3.1 Route planning in MATSim

Whenever an agent needs to plan its route between destinations it calls a router. The router uses a Djikstra algorithm to find a close to optimal path to the destination, based on the cost of a link. Link cost is based on a travel time parameter multiplied by a penalty. Travel time can be either a current travel time based on traffic conditions, or a travel time based on maximum speed on the link. Link times and penalties are associated with a particular router (e.g. `freespeed`, using max link speed, or `globalspeed` using current actual speed) and the router is used for a specific combination of vehicle type and network mode such as car, bicycle, public transport, walking, etc. set during configuration.

However, in the evacuation application we needed to use different route planning in different situations, for the same vehicle type: sometimes a car would require using the `freespeed` router, sometimes `globalspeed`, and then emergency vehicles required different costs again. The refinement to MATSim that was introduced was to allow a specific router to be specified dynamically as a parameter with the destination node for a trip.

3.2 The evacuation system routers

Initially when we developed our evacuation applications we found that as roads became congested (with reduced speeds) we had a problem preventing agents from driving directly into the fire in order to find a faster route to the evacuation destination. We tried a number of ways of controlling this, including: 1. requiring the routing algorithm to disregard current speeds and capacities based on congestion when routing, and instead use the maximum speed of links. 2. routing via specific waypoints at the BDI level to enforce particular routes. 3. modifying maximum speed to zero or very low on links affected by smoke and fire to influence the routing away from using those links.

However, none of these were really satisfactory. Inability to use congestion information meant we could not adequately model the fact that while this information may not immediately be known to an agent, it is likely to be transmitted via various mechanisms and used in planning egress routes. Setting the speed to zero meant that vehicles currently on the link remained stuck there. Also using very low values did not properly reflect the speed should an agent actually decide to go into an affected area based on some cognitive goal such as reaching family members or, in the case of emergency services, dealing with the fire.

Routing via specific waypoints at the cognitive level required far too much MATSim specific information within the BDI system and was not robust to changing circumstances.

As a result three specific routers were developed for the evacuation domain using the key idea of risk reduction routing as described in [10]. Links within the danger zone (the fire area) were given high penalties, while those in an area around the danger zone (for us the smoke zone) were given penalties that decreased based on distance from the fire front. The travel time could then be either maximum link speed or actual current link speed. This successfully controlled the behaviour of the agents such that they didn't "mindlessly" drive into the fire, but if they had a goal to reach a destination within the danger zone (such as rescuing family members), then they were not prevented from doing so. We currently have three different routers for our evacuation applications: `carFreespeed`, `carGlobalInformation` and `emergencyVehicle`, with the ability to switch between them depending on context. The `emergencyVehicle` router is similar to the `carGlobalInformation` router but imposes lower penalties for coming close to the fire – it allows taking of greater risks.

Within our system the road link penalties used by the routers are updated periodically based on information from the fire model. This information is also used to provide percepts to the (BDI) agents to be potentially used in their reasoning process. The sequence of execution is as follows: (i) updated smoke and fire shape information is published by the Phoenix Fire model; (ii) the MATSim model has subscribed to this information and therefore receives it immediately; (iii) at the next MATSim step MATSim places penalties on links and also produces smoke/fire percepts for relevant agents in the areas (as explained in Section 2.4); and (iv) at the next BDI step the smoke percepts are passed to the specified BDI agents where it affects their decision making.

The ability to dynamically choose which router to use, combined with the ability to set penalties dynamically based on a changing situation provides great flexibility which can be relatively easily extended and modified for different application needs.

4 Initialisation of MATSim

Like most microsimulations MATSim typically includes data from real environments, and creates agents with attributes and activities taken from data. The easiest way to create a road network is to use OpenStreetMap[7] and then convert to MATSim representation using the MATSim utility class `OsmNetworkReader`. The population of agents is given by an input file in XML format as shown in Figure 3.

Typically initial agents with attributes are created by writing a script that processes census data in some format. Activities are then added by generalising from activity diaries from a population sample. If census data provides place of

[7] https://www.openstreetmap.org/

```
<population>
 <person id="">
  <attributes>
   <attribute name="" class="">...</attribute>
    ...
  </attributes>
  <plan selected="yes">
   <activity type="home" x="" y="" end_time="" />
   <leg mode="car" />
   ...
  </plan>
 </person>
 ...
</population>
```

Fig. 3: Structure of MATSim's input population XML file

work and mode of transport to work for sufficiently small geographical areas, reasonably realistic work travel plans can be created.[8]

4.1 Creating the agent population

It is relatively straightforward to create a set of individuals that match census data with regard to attributes such as gender, age, etc. These attributes can be directly encoded as attributes of a person in the MATSim population using the `attributes` set as shown in Figure 3. Grouping individuals into family and household structures that also match census data is more complex. Various approaches have been used in the literature, and for Australian census data there is software available that can create a population, assigned to households with address coordinates.[9] There are scripts which take this information and create an initial population in MATSim format. Activities can then be assigned based on census data and on surveys.

We further provide a new convenient way of connecting BDI behaviour classes to individual MATSim agents directly via the input population file using the special `BDIAgentType` attribute (of the form shown in Figure 3) whose value is a fully qualified Java class name that captures the BDI behaviours for that agent type.

4.2 Creating the activities

There are a number of approaches that have been used for creating the activity schedules of the agents. These include activity based demand generation models

[8] Cf. https://github.com/matsim-org/matsim-code-examples/find/0.11.x → `RunDemandGenerationFromShapefileExample.java` for intuition.

[9] Software is available at https://github.com/agentsoz/synthetic-population.

(e.g. [14]), smart card or mobile phone data (e.g. [1, 5]) hourly origin destination matrices [3], or commuting matrices [2]. One can additionally calibrate against emergent properties such as traffic counts (e.g. [18]). The challenge is to combine the data that is locally available, and which is typically different in each location, to come up with a good approximation.

For the bushfire applications we have a user requirement to simulate the agents going about their daily business, prior to the evacuation request. Consequently we have developed some tools to assist with this. If census data includes information on whether individuals work, where they work (at a suitably fine geographical granularity), and mode of transport to work, then reasonable initial activity schedules for work travel can be created by assigning travel to and from a work activity of appropriate length, for a suitable number of people in each geographical area. Work locations can be assigned randomly within the relevant geographical area, or probabilistically according to knowledge of centres of activity. Timings must also be assigned based on some assumptions (or data) about usual length and time of work activities. Code is available which allocates these work activity schedules to the population described in section 4.1, based on census data of individuals and households.

In order to simulate other activity we have developed a tool that allows us to use expert knowledge about the approximate proportions of different agent types doing various activities at certain times of day, in order to generate representative activity schedules. This tool can possibly also be useful for creating scenarios involving specialised situations, such as an influx of tourists during holiday season, or scenarios involving a special event. The aim is not to build calibrated populations, but instead build representative populations that capture sufficient richness of activities while being relatively easy for domain experts to specify. However this tool is potentially suitable only for relatively small geographical areas in its current state, as with large scale scenarios, a random destination choice with a uniform distribution leads to distances that are too large (in the average half the scenario diameter). Nevertheless it has been useful for the current applications and is an area of ongoing work.

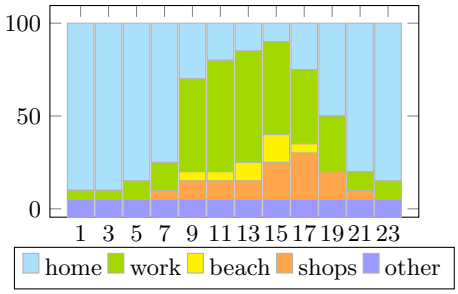

Fig. 4: Example input weekday activities for `Residents`.

The tool takes as input, for each population subgroup, a table of activities distributions for the day, a list of GIS shapes associated with each activity signifying places where those activities can be performed, and the number of persons of the subgroup to generate. For instance, given the example activities distributions for the `Resident` subgroup in Figure 4 – that tells us what activities the subgroup is doing at different times of the day and in what proportion – the algorithm constructs a population whose activites taken together resemble these distributions. The output is a MATSim population file in GZipped-XML (`.xml.gz`) format.[10]

5 Designing agents and their behaviours

Designing the agents and their behaviours in a complex simulation involving multiple components will typically involve some level of interaction between these components. Decisions must be made about which components receive and are affected by, which information. In some cases agents may be represented in different components to take advantage of specialised representations and modelling. This is the case in BDI-ABM integration as described in [16], where the cognitive reasoning of the agents is in a different component than the interaction of the agent with the environment. This raises questions regarding the design of the agents and what aspects should be in which component. This will always depend on the particular application and the specifics of the components. The principle with BDI-ABM agents has always been that reasoning decisions should be made by the cognitive system, with actions carried out by the ABM. In practice however this is a fuzzy boundary. Route planning is a cognitive process. However it is tightly coupled with the representation of the road network, and given that MATSim has route planning as an integral part of the system it would be a lot of additional work to reproduce this in the BDI system. It is also the case that as it is MATSim which has detailed location information regarding agents, it is sometimes MATSim that must receive information and then channel it to the relevant (BDI) agents. In consequence, an active design decision needs to be made at which level of abstraction the BDI model operates. For our present applications, we assume that the BDI model knows about certain fixed location coordinates (e.g. activity locations), but not about the road network, or dynamic co-ordinates such as agent location.

When a BDI-action is sent to MATSim, then the BDI goal of which it is a part is suspended until MATSim returns from that action with either a PASS or a FAIL. We note that failing a BDI action is not the same as failing a BDI plan, as an action is a plan step (similar to a sub-goal), and therefore should, like a sub-goal, raise a consideration of alternative options for achieving success for that action, rather than automatic failure, leading to failure of the containing plan. We have developed a mechanism to specify (possibly with some analysis or query regarding the current situation) together with an action, what should

[10] This software can be accessed at https://github.com/agentsoz/ees-synthetic-population/tree/master/plan-algorithm.

be done if it fails. Sometimes a percept relevant to a BDI-action may be better provided to the BDI system as a trigger for reasoning in a separate intention, rather than associated with immediate SUCCESS or FAIL of the BDI-action. This allows the BDI system to reason about the situation using standard goals and plans with context conditions, in order to determine what should be done. The BDI action can then be aborted/replaced if that is considered appropriate.

A simplified example of this situation is shown in figure 5 using the detailed design diagram of the Prometheus agent system design methodology [13]. This

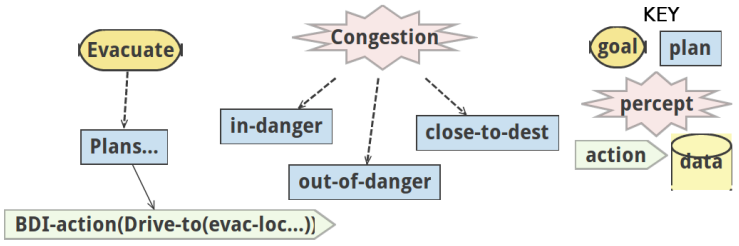

Fig. 5: Basic design of handling of congestion percept

figure shows an evacuate goal which through some series of plans and subgoals has led to a BDI-action to drive to the evacuation destination. The design specifies that if a congestion percept arrives (which will happen only while the agent is engaged in some `drive-to`), then an appropriate plan is chosen to assess the situation. Here we show 3 different plans depending on whether the agent is 1. still in the danger zone, 2. out of the danger zone but far from the destination, 3. close to the destination.

If as a result of the reasoning that happens when one of these plans is chosen, there is a decision that the agent should modify its destination, then the decision code within the congestion intention[11] will abort the `drive-to` action within the Evacuate intention. The code for handling this situation will then instantiate a new `drive-to` action with the new destination.

There may be a number of subgoals and further plans associated with the plans for handling the congestion percept. Figure 6 shows a possible design. Plan in-danger results in the agent querying MATSim to find a location outside the danger zone to which there is a faster route (given congestion) than the point outside the danger zone on the current route. If such exists the agent will register the new destination and abort the current `drive-to` action allowing it to be replaced with a new `drive-to` action. The new destination will be accessed and provided as a parameter to the new `drive-to`. MATSim will then add the route in standard fashion. The code for failing/aborting the `drive-to` action for

[11] An intention is simply the code stack resulting from a top-level instantiated goal.

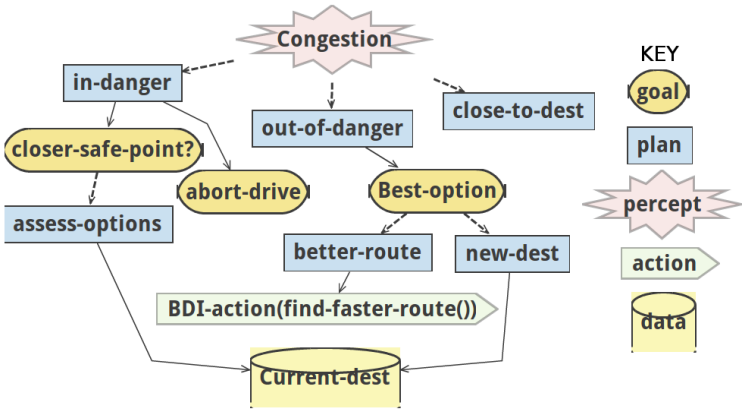

Fig. 6: Detailed design of handling of congestion percept

this case will, in addition to instantiating a new `drive-to`, need to ensure this is followed by a choice of final destination once out of the danger zone.

If the out-of-danger plan is chosen the agent may consider either looking for a new route to the current evacuation destination, or looking for an alternative destination which is faster to get to, given congestion. Let us assume that the agent first looks for a better route, by choosing the better-route plan, resulting in instantiating a new BDI-action to "find-faster-route". If this has been properly set up in MATSim as described in section 2.4, then the action will be generated in MATSim to replace the current route with a faster one if such exists and return SUCCESS. At this point the intention triggered by the congestion percept would complete. If no faster route was found then a FAIL would be returned, in which case the plan can just be allowed to fail and the standard BDI execution will lead to the plan new-dest which can consider alternative evacuation destinations. Depending on the outcome of that reasoning, either a new destination may be chosen or it may be determined that there is nothing better and no change is made. The former will result in aborting the current `drive-to` and instantiating a `drive-to` with new destination (as in case (1)), while the latter would simply terminate with no change. Case (3) may simply be a no-op where the agent does no further reasoning, as they are anyway close to the destination.

The key aspects that we have identified for design are:

- percepts should always be handled by a new separate intention. This may simply alter a belief that affects a current intention, it may generate a substantial reasoning process regarding what to do with regards to a current intention, or it may generate new behaviour unrelated to other current intentions.
- any reasoning, other than that specifically related to route planning or simple locational reasoning, should be done by the BDI system.

- action failures must be handled differently to plan failures - they are more like goal failures. (Future work should investigate infrastructure support for high level specification and management of such, in the same style as is done for goal failure with a search for alternative plans based on context).
- there is a need for aborting an action, which arises from BDI reasoning, as well as failing an action which arises from the environment.
- querying of MATSim may be needed in order to do the BDI reasoning. This is supported by the BDI-ABM infrastructure of [16].

Agent intentions may also be trigered by messages from another agent. This happens in the evacuation domain when a policeman "sees" (via a percept from MATSim) an approaching agent, and directs them to take a particular turn. A similar approach to that shown with the congestion percept is appropriate. First generate an intention that reasons about the message, and any effect on current intention(s). Then modify current intentions as needed.

6 Discussion and conclusion

In this paper we have described some of the important aspects of MATSim which facilitate its use as a component in large complex simulations, including some new extensions and some supporting tools. MATSim is itself a large and complex system. However, where transport simulation is an important aspect of a larger micro-simulation it does not make sense to implement a simpler (and likely less efficient and accurate) alternative. Rather effort should be made to facilitate the re-use and incorporation of this existing and highly flexible software. The work described here contributes to this effort.

A number of the aspects described have been motivated by our use of MATSim as a component in evacuation applications and in an urban planning application. We believe that this description will assist others in using MATSim in similar ways. The key aspects we have described are:

1. Architecture that enables external control of multiple simulation components. Components must be able to be started and paused externally.
2. A well specified API to support addition of new percepts and actions, as well as the structure within which to do this.
3. An ability to modify the environment dynamically, in this case using dynamic penalties and flexible routers.
4. Tools to assist in creating a suitable representative initial scenario for the simulation.
5. Design of cognitive agents within the BDI-MATSim system.

We also gave examples of how we have used these facilities within evacuation applications, where requirements were user driven. One of these applications is currently deployed and the other is expected to be deployed within coming months.

References

1. Anda, C., Erath, A., Fourie, P.J.: Transport modelling in the age of big data. International Journal of Urban Sciences **21**(sup1), 19–42 (Feb 2017)
2. Balmer, M., Axhausen, K., Nagel, K.: A demand generation framework for large scale micro simulations. Transportation Research Record **1985**, 125–134 (2006)
3. Balmer, M., Rieser, M., Vogel, A., Axhausen, K., Nagel, K.: Generating day plans using hourly origin-destination matrices. In: Bieger, T., Laesser, C., Maggi, R. (eds.) Jahrbuch 2004/05 Schweizerische Verkehrswirtschaft, pp. 5–36. Schweizer Verkehrswissenschaftliche Gesellschaft (2005)
4. Bazzan, A., Amarante, M.d.B.d., Sommer, T., Benavides, A.J.: ITSUMO: An agent-based simulator for ITS applications. In: Rossetti, R., Liu, H., Tang, S. (eds.) Proceedings of the 4th Workshop on Artificial Transportation Systems and Simulation. IEEE (Sep 2010)
5. Bouman, P., Lovric, M.: Rotterdam: Revenue management in public transportation with smart-card data enabled agent-based simulations. In: Horni, A., Nagel, K., Axhausen, K.W. (eds.) The Multi-Agent Transport Simulation MATSim, chap. 81. Ubiquity, London (2016). https://doi.org/10.5334/baw
6. Bruch, E., Atwell, J.: Agent-based models in empirical social research. Sociological Methods & Research **44**(2), 186–221 (2015)
7. Dahmann, J.S., Kuhl, F., Weatherly, R.: Standards for simulation: as simple as possible but not simpler the High Level Architecture for simulation. Simulation **71**(6), 378–387 (1998)
8. Dobler, C., Nagel, K.: Within-day replanning. In: Horni, A., Nagel, K., Axhausen, K.W. (eds.) The Multi-Agent Transport Simulation MATSim, chap. 30, pp. 187–200. Ubiquity Press, London (2016)
9. Horni, A., Nagel, K., Axhausen, K.W.: The Multi-Agent Transport Simulation MATSim. Ubiquity Press, London (2016)
10. Lämmel, G., Klüpfel, H., Nagel, K.: Risk minimizing evacuation strategies under uncertainty. In: Peacock, R., Kuligowski, E., Averill, J. (eds.) Pedestrian and Evacuation Dynamics. pp. 287–296. Springer, Berlin (2011)
11. Padgham, L., Nagel, K., Singh, D., Chen, Q.: Integrating BDI agents into a MATSim simulation. In: ECAI 2014. pp. 681–686. IOS Press, Prague, Czech Republic (2014)
12. Padgham, L., Singh, D.: Making MATSim agents smarter with the Belief-Desire-Intention framework. In: Horni, A., Nagel, K., Axhausen, K.W. (eds.) The Multi-Agent Transport Simulation MATSim, chap. 31, pp. 201–210. Ubiquity Press, London (2016)
13. Padgham, L., Winikoff, M.: Developing Intelligent Agent Systems: A practical guide. Wiley Series in Agent Technology, John Wiley and Sons (2004)
14. Rieser, M., Nagel, K., Beuck, U., Balmer, M., Rümenapp, J.: Truly agent-oriented coupling of an activity-based demand generation with a multi-agent traffic simulation. Transportation Research Record **2021**, 10–17 (2007)
15. Singh, D., Padgham, L.: OpenSim: A framework for integrating agent-based models and simulation components. In: ECAI'14. pp. 837–842. IOS Press, Prague, Czech Republic (2014)
16. Singh, D., Padgham, L., Logan, B.: Integrating BDI agents with agent-based simulation platforms. Autonomous Agents and Multi-Agent Systems **30**(6), 1050–1071 (2016)

17. Waddell, P.: Urbansim: Modeling urban development for land use, transportation, and environmental planning. Journal of the American Planning Association **68**(3), 297–314 (2002)
18. Zilske, M., Nagel, K.: A simulation-based approach for constructing all-day travel chains from mobile phone data. Procedia Computer Science **52**, 468–475 (2015)