CONFIDENTIAL. Limited circulation. For review only.

Modular Framework for Simulation Modelling of Interaction-Rich Transport Systems

Michal Jakob[1] and Zbyněk Moler[1]

Abstract— **The increasing penetration of information and communication technology (ICT) in transport systems changes the requirements on techniques and tools for transport simulation modelling. Novel ICT-powered mobility services, such as real-time on-demand transport, rely on complex, ad hoc interactions between different entities of the transport system. These interactions have to be captured in the model if it is to provide accurate representation of the modelled transport system. Unfortunately, existing modelling tools are not well suited for modelling such interaction-rich transport systems. We address this problem and provide a modular simulation framework suitable for modelling transport systems in which ad hoc interactions and decision making play an important role. By fully employing the agent-based modelling paradigm, our framework provides flexibility and extensibility that cannot be achieved by traditional approaches. More specifically, the framework provides an extensible library of model elements based on a unifying ontology of modelling abstractions, a high-performance discrete-event simulation engine and suite of tools supporting real-world deployment and utilization of implemented models. We demonstrate the flexibility and effectiveness of the framework on several specific models of interaction-rich transport systems.**

I. INTRODUCTION

The increasing deployment of ubiquitous GPS-enabled and internet-connected devices is changing the way transport is organized and managed. Novel ICT-powered mobility services, such as real-time on-demand transport, peer-to-peer car sharing or dynamically priced taxis, are on the rise. A common feature of these services is the intensive use of (semi-)automated, electronic communication for coordinating the use of transport resources, in order to improve the efficiency and convenience and to reduce the financial and environmental costs of the service. In the case of shared collective taxi services, for example, the explicit, real-time coordination between the riders and the provider of the service allows using fewer vehicles and, consequently, road space compared to when the same travel demand was served in an uncoordinated fashion. At the same time, however, the newly introduced coordination interactions increase the complexity of the transport system and, consequently, make its operation more difficult to analyze and foresee.

Simulation modelling is a well-established approach for analyzing the behaviour of complex socio-technical systems and should therefore be also applicable for analyzing transport systems employing ICT-powered services. Unfortunately, existing simulation toolkits, partly due to their inherent design constraints, do not support the simulation of ICT-powered transport systems well – in particular, they lack the support for modelling ad hoc interactions among the entities of the transport system and the just-in-time decision making required for participating in such interactions, which are both crucial for accurately capturing the behaviour of ICT-powered systems (in the following, we use the general term *interaction-rich transport systems* for systems in which ad hoc interactions among entities in the system strongly affect system behaviour). Consequently, purpose-specific simulators have to be developed for each simulation study of ICT-enabled, interaction-rich transport systems, which is costly.

In our work, we aim to remedy this situation by providing a simulation modelling framework, termed *AgentPolis*, designed from its inception to support the modelling of interaction-rich transport systems. Key to achieving this objective is the use of the concept of *multiagent systems* as the basis of the framework's design. Multiagent systems capture the interaction-centricity of ICT-powered transport systems very well – putting them in the core of the modelling framework therefore minimizes the structural and behavioural gap between the target interaction-rich system and its model.

In this paper, we present the main results of our research, describing the four pillars of the AgentPolis framework – ontology of modelling abstractions, library of ready-to-use model elements, discrete-event simulation engine and simulation tools – along with the experience of employing the framework to implement models of five different types of interaction-rich transport systems.

II. RELATED WORK

In the last decade, simulation modelling has become an indispensable tool for studying the behaviour of ICT-powered, interaction-rich transport systems. Horn et al [8] employed an agent-based simulation, completely developed in-house, to study operational characteristics of a multimodal transport system integrating scheduled and flexible on-demand services. Demand-responsive transport systems were also studied by Quadrifoglio et al. [12].

Taxi operations were also evaluated using simulations, both in their standard form (e.g. [4]) or employing a real-time taxi sharing scheme (e.g. [9], [7]). In all three cases, model-specific simulation tools had to be developed and used, with [4] explicitly stating that existing simulation toolkits, including MATSim and SUMO, were not suitable for the task. Another type of transport systems evaluated using simulations are car sharing services. Barth et al. [3]

[1]{jakob, moler}@agents.fel.cvut.cz, Agent Technology Center, Faculty of Electrical Engineering, Dep. of Computer Science and Engineering, Czech Technical University, 121 35 Prague, Czech Republic

Preprint submitted to 16th International IEEE Annual Conference on Intelligent Transportation Systems. Received April 15, 2013.

evaluated a car sharing scheme under real-world conditions of a Californian resort community, again employing an in-house developed simulation tool.

A very interesting approach is presented by Wainer [13]. The author developed a general language for describing simulation models that allows decoupling model description from the simulation engine used to execute the model. The objectives of Wainer's work – flexibility and the ability to rapidly develop simulation models – are similar to our goals. His approach is, however, based on discrete-event cellular automata and directed towards vehicle-centric low-level traffic simulations.

A common attribute of the majority of simulations of ICT-powered transport systems is that these simulations were developed from scratch using general-purpose programming languages (most often C++ or Java). There are exceptions – [5] and [6] used the MATSim simulation framework [1] for evaluating car sharing and collective taxi schemes, respectively. Furthermore, Martinez [10] used the general-purpose AnyLogic simulation toolkit to model a taxi sharing scheme in Lisbon. In all of these cases, however, model developers faced considerable difficulties expressing and implementing required model behaviour using the chosen general toolkit; this resulted in long development times and/or reduced fidelity of implemented models.

III. Background and Motivation

Although there are many differences between systems such as collective taxis and car sharing, there are also many elements (e.g. the concept of road networks, vehicles, passenger demand, or coordination protocols) that are similar and can be shared between the models of such transport systems. Judging from the minimum use of general toolkits for the simulation modelling of interaction-rich transport systems, it appears that the above similarities have not been sufficiently exploited. We believe – and, as we shall see, this belief has been confirmed by our results so far – that the difficulties in employing general simulation toolkits, and the consequent lack of reuse in modelling interaction-rich transport systems, stems from the fact that existing toolkits do not sufficiently take into account the multiagent nature of the ICT-powered transport systems and, consequently, fail to provide abstractions for modelling such systems in a direct, natural way.

Before explaining how we have solved the problem, let us briefly introduce the very concept of multiagent systems (see e.g. [11] for an in-depth discussion). With an acceptable level of simplification, the *multiagent system* can be defined as a system composed of multiple autonomous entities, termed *agents*, situated in a shared environment. The *environment* represents the physical space surrounding the agents and the agents can interact with the environment in two ways. First, agents perform *actions* that modify the state of the environment; second, in the opposite direction, agents are informed about the state of the environment through *perceptions*. We assume that the agents are endowed with intelligence that allows them to select and execute actions that lead to their goals. However, as the environment is one and the agents are many, the actions of individual agents can mutually interact and produce results that, for better or worse, cannot be achieved by single agents alone. In addition to such implicit interaction, agents can also interact directly, i.e., bypassing the environment, through message-based *communication*. See Figure 1 for a scheme relating the above concepts in a high-level conceptual model of a multiagent system.

Let us now turn attention to transport systems. In transport systems, a large number of autonomous entities, such as passengers, drivers or transport operators, pursue their transport-related objectives within the context of a shared and capacity-constrained transport infrastructure. The individual entities interact among themselves and with the transport infrastructure (e.g. queuing on junctions), and produce complex, emergent global behaviours (e.g. congestion). In traditional transport systems, interactions among entities are mostly implicit, mediated by the transport environment. In ICT-powered transport systems, implicit interactions are complemented by explicit ICT-mediated interactions that can also strongly affect the overall system behaviour.

The purpose of the previous two paragraphs was to show that ICT-powered, interaction-rich transport systems are essentially multiagent systems. Consequently, to model interaction-rich transport systems, the (multi)agent-based modelling paradigm should be employed because it offers the most direct conceptual match between the model and the system. Unfortunately, existing transport modelling toolkits support the agent-based modelling paradigm only to a limited extent. Although MATSim [1], for example, uses individual-level modelling, it treats individuals as passive data structures whose state can only be updated synchronously by central modules at infrequent, predefined points in time. Despite some practical advantages, such a centralized approach contradicts the nature of multiagent systems and consequently introduces a significant modelling gap – in reality, agents in a transport system make just-in-time decisions asynchronously at different occasions during a day, often in reaction to external observations or messages.

To avoid these issues, the proposed AgentPolis framework employs the agent-based modelling approach in its full extent. AgentPolis does not impose constraints on when and how decision making, activities and interactions can occur in the model, and it is therefore also suitable for modelling ICT-powered transport systems with ad hoc interactions and just-in-time decision making.

IV. Framework Overview

The proposed AgentPolis framework provides abstractions, code libraries and software tools for building and experimenting with agent-based models of interaction-rich transport systems. More specifically, the framework consists of the following four components:

1) *Modelling abstraction ontology* which provides a unifying set of concepts for expressing agent-based simulation models. The abstractions refine the more general

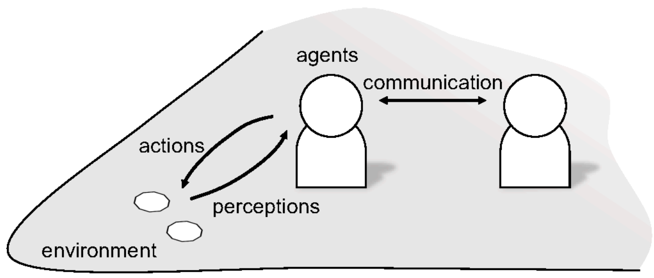

Fig. 1: High-level conceptual model of a multiagent system.

multiagent systems concepts and make them expressible in object-oriented programming languages.

2) *Model element library* which contains concrete implementations of the modelling abstractions chosen so as to represent the elements frequently used in real-world transport models.
3) *Simulation engine*, based on the discrete event simulation approach, which provides the runtime functionality for simulating AgentPolis models.
4) *Simulation tools* which support the deployment and use of AgentPolis models in real-world conditions by providing data import, scenario configuration and simulation result analysis and visualization capabilities.

In the following two sections, we describe the components of the framework in more detail.

V. Modelling Abstractions and Elements

In designing the AgentPolis framework, our aim was to provide a framework that is not overly complex and that provides maximum ready-to-use functionality out of the box while offering enough flexibility to adapt to initially unforeseen requirements. A key tool for achieving this objective was the explicit separation between well-defined modelling abstractions, based on the multiagent conceptual model (see Section III), and concrete model elements for building specific application models. By requiring that any model element is an instance of some modelling abstraction, we enforce design and implementation decisions that ensure interoperability among different elements and facilitate addition of new application-specific model elements.

The AgentPolis framework currently has eight modelling abstractions (see Figure 2) and several tens of model elements – these evolved through several iterations during which the abstractions were used to define concrete model elements that were, in turn, used to build specific simulation models.

In the rest of the section, we describe individual modelling abstractions along with the corresponding model elements. Due to limited space, we omit some technical details and focus on the features that best convey the overall idea of the framework. Also note that due to circular dependencies among concepts and elements, sometimes references are made to concepts or elements that are only defined later.

A. Agents

Agents are the central entities of agent-based models and are the main drivers of model dynamics. Somewhat surprisingly, the concept of the agent is only loosely defined in the AgentPolis framework. This is primarily because of the large variation between the behaviour of agents in different models, which makes standardization of agent behaviour difficult and, in fact, counterproductive. Each agent in the AgentPolis framework is therefore only required to have defined its *lifecycle*, which is a top-level activity governing the agent's behaviour.

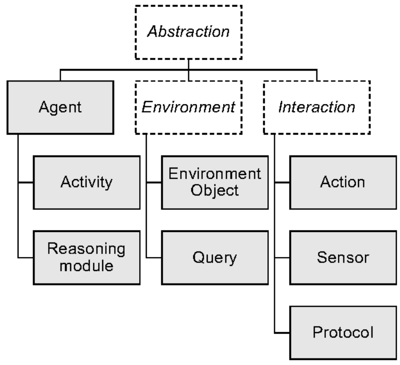

Fig. 2: Modelling abstractions of the AgentPolis framework. The concepts in the white, dashed-outline boxes only provide grouping and are not used as modelling abstractions.

Two predefined lifecycles are nevertheless provided in the framework and can be utilized for defining new agents. The `PTDriver` lifecycle represents the top-level behavioural loop of the agent serving as a public transport vehicle driver; the `UrbanTraveller` lifecycle can be used to implement an agent generating and executing basic activity-driven travel patterns[1].

B. Activities

Activities provide the abstraction for defining agent behaviour. Technically, activities are reactive control structures implementing the logic determining which actions or nested activities the agent executes at a certain point in time or in response to sensor information or messages received from other agents.

For example, the `DriveVehicle` activity moves a vehicle along a predefined route. The route to follow, expressed as a sequence of nodes of an underlying transport network, is given as an input parameter of the activity. The `DriveVehicle` activity then sequentially, for each edge of the transport network, invokes the `MoveVehicle` action to change the location of the vehicle (as well the driver and any passenger inside the vehicle) on the network. After the vehicle reaches the final waypoint, the activity notifies the caller about its successful conclusion and finishes. The list of activities currently provided by the AgentPolis framework is given in Table I.

[1]Because of their defining role in specifying agent behaviour, we sometimes refer to agents by the name of their assigned lifecycle, e.g., calling an agent employing the `PTDriver` lifecycle as a `PTDriver` agent.

Activity	Description
`Walk`	The agent walks between locations according to a specified journey plan.
`RideInVehicle`	The agent travels as a passenger of an individual transport vehicle according to a journey plan.
`RideOnPT`	The agent travels by public transport according to a journey plan.
`DriveVehicle`	The agent drives a vehicle according to a journey plan.
`ParkVehicle`	The agent parks a vehicle at or near a specified location.
`Wait`	The agent spends a specified time waiting.

TABLE I: Core activities in the AgentPolis framework.

Action	Description
`MoveVehicle`	Moves a vehicle across an edge of the road network, taking possible congestion in the account.
`MoveAgent`	Moves an agents across an edge of the road network.
`TeleportAgent`	Moves an agent instantly to a specified location.
`GetInVehicle`	Moves a passenger into a vehicle (the passenger will be linked with the vehicle and move automatically whenever the vehicle moves).
`GetOffVehicle`	Removes a passenger from a vehicle (unlinks the passenger from the vehicle).
`WaitForVehicle`	Waits until a specified vehicle arrives.

TABLE II: Core actions in the AgentPolis framework.

C. Actions

Actions provide the abstraction for modelling how agents manipulate the environment. Each action defines the logic determining action duration and the logic defining which state attributes of which environment objects should be modified as the effect of executing the action.

For example, the `MoveVehicle` action moves a vehicle along a transport network edge by changing the vehicle's location from a transport network node (corresponding to a junction) to another, adjacent network node. The `MoveVehicle` action interacts with the queuing logic implemented by the `TransportNetwork` environment object. The state of the `TransportNetwork` object can affect the duration of the `MoveVehicle` action and can even make the action fail if the queue associated with the traversed network edge is full. The list of actions currently provided by the framework is given in Table II.

D. Sensors

Sensors process percepts from the environment and allow agents (and their activities) to be informed about events in simulation execution, in particular about the changes of the environment state and the execution of action and activities.. Together with messages received from other agents, sensor notifications provide the main trigger for starting, terminating or changing activities executed by agents.

For example, the `PositionUpdate` sensor notification is sent to the `DriveVehicle` activity after the vehicle has reached a new position; after receiving the notification, the `DriveVehicle` activity decides where to move the vehicle next and invokes the next `MoveVehicle` action accordingly. The list of all sensors implemented in the framework is given in Table III.

Sensor	Description
`PositionUpdated`	Informs about a new position of a specific agent or an environment object.
`NextVehicleLoc.`	Informs about the upcoming next location of a vehicle.
`DrivingFinished`	Informs that a vehicle driver has reached the destination specified by the plan.
`WaitingFinished`	Informs that a specified waiting time has elapsed.
`VehicleArrived`	Informs that a vehicle arrived to a given node.

TABLE III: Core sensors in the AgentPolis framework.

Environ. Object	Description
`TransportNetwork`	A network of roads, railways, cycle paths and/or pedestrian pathways with the associated queuing logic.
`PTStops`	A list of public transport stops or stations.
`Attractor`	A location acting as a destination for trips with specific purpose (i.e. schools, offices, shops etc.).
`Vehicle`	A vehicle that can move along a transport network (car, bus, tram, train etc.).

TABLE IV: Core environment objects in the AgentPolis framework.

E. Environment Objects

The environment models the physical context in which agents are situated and perform their activities. In the AgentPolis framework, the environment is decomposed into and, consequently, represented as a collection of *environment objects*, where each environment object represents a fragment of the modelled physical reality and its associated state. The state of an environment object is represented by its attributes and it can only be changed by actions or by the object's internal update logic. Environment objects notify agents through sensors about changes in their state.

For example, the `TransportNetwork` environment object represents a transport network (road, cyclepath, footpath or railway). It consists of a graph of junctions and connecting network segments with associated queues and update logic for modelling congestion. The queue is used by the `MoveVehicle` action to determine how much time a vehicle needs to move along the respective network segment. The list of the environment objects provided by the AgentPolis framework is given in Table IV.

F. Queries

Queries are used by agents to obtain information about the state of the environment. Queries read, filter or aggregate but do not change the state of any environment objects. In contrast to sensors, queries are invoked by the agents (or, typically, by activities)[2]. Although not strictly necessary – calls to queries could be replaced with direct calls to respective environment objects – queries improve encapsulation by providing a layer that hides environment's internal implementation from agents.

For example, given an agent identifier, the `AgentPosition` query returns the position of the

[2]Queries can therefore be viewed as information *pull* requests, while sensors correspond to information *push* requests.

Query	Description
AgentPosition	Returns the current position of an agent or an environment object.
PTStopPosition	Returns the position of a (public transport) stop or station.

TABLE V: Core queries in the AgentPolis framework

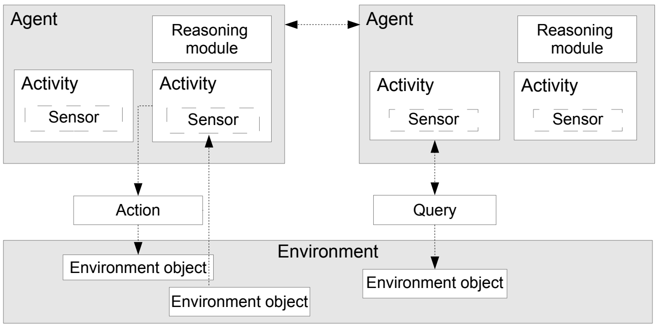

Fig. 3: Simplified architecture of AgentPolis models.

agent as the identifier of the transport network node on which the agent is located. The list of queries implemented in the framework is given in Table V.

G. Communication Protocols

Communication protocols are the abstraction for modelling inter-agent communication by means of message passing. At the moment, the framework core only provides simple protocols: `1-to-1 messaging` and `1-to-many messaging`. Additional, more complex protocols (e.g., tendering and auctions) have, however, been implemented as part of application-specific models (see SectionVII).

H. Reasoning Modules

As part of their behaviour, agents may need to make decisions that require executing complex algorithms. In the AgentPolis framework, such algorithms can be encapsulated into *reasoning modules* and reused in different activities.

At the moment, the only reasoning module provided in the framework core is the *JourneyPlanner* module. The module, given an origin and destination location and time constraints, finds a shortest-duration journey plan that can subsequently be executed by framework activities. Additional reasoning modules have been implemented as part of application-specific models (see Section VII).

Figure 3 shows how all modelling abstraction relate to each other in AgentPolis simulation models.

VI. Simulation Engine and Tools

The library of model elements and the underlying ontology of modelling abstractions form the fundamental part of the AgentPolis framework. Additional functionality is, however, required for practically using developed models as part of simulation-based evaluation and decision making processes. To this end, the AgentPolis framework provides software components supporting the whole modelling lifecycle from importing real-world data, executing simulation models and analyzing and visualizing simulation results.

A. Data Import Tools

To facilitate the incorporation of real-world data into AgentPolis models, the framework provides data importers for converting external datasets into framework's internal data models. At the moment, the framework supports importing data in the *OpenStreetMap (OSM)*[3] and *General Transit Feed Specification (GTFS)*[4] formats, including the automated cross-referencing between both formats (e.g., mapping of public transport stops that appear both in OSM and GTFS files). This way information about road, cyclepath and footpath networks, public transport routes and timetables and (basic) land use can easily be incorporated in AgentPolis models. Files imported by the framework tools are checked for consistency in order prevent the hard-to-trace errors caused by invalid data during simulation execution.

AgentPolis models can incorporate additional categories of data, such as socio-demographic data or origin-destination matrices representing travel flows. However, as no established standards exist for these data categories, importers for such datasets are scenario-specific and need to be developed or customized for each model.

B. Simulation Engine

Simulation engine for executing AgentPolis simulation models is an essential part of the framework. The AgentPolis framework employs the *discrete event simulation (DES)* approach [2] in which the operation of the target system is modelled as a discrete sequence of events in time. Each event occurs at a particular instant in time and marks a change of state of the system. Between consecutive events, no change in the system is assumed to occur; thus the simulation can directly jump in time from one event to the next, which makes it computationally more efficient than the time-stepped approach that is mostly used in transport models.

In AgentPolis models, events provide the low-level causal link between actions, model updates and sensor invocations. Whenever an agent executes an action, the action inserts an event into the event queue; the event has a state update logic attached specifying which environment objects should be updated as the effect of action execution. The state update logic is executed only after the simulation time corresponding to the duration of the action has elapsed. The modification of the environment state caused by the update logic triggers sensor notifications which are received by agents (activities); the agents (activities) can consequently react by invoking further actions, thus closing the model update loop.

The AgentPolis uses the discrete event-queue implementation provided by the *Alite*[5], a general purpose lightweight toolkit for building multiagent systems. A screenshot of a running AgentPolis simulation is given in Figure 4.

[3] `http://openstreetmap.org`
[4] `https://developers.google.com/transit/gtfs/reference`
[5] `http://alite.agents.cz`

Fig. 4: High-level view of a running AgentPolis simulation model. Road (black), pedestrian (grey), tram (yellow) and metro (red) networks and UrbanCitizen (green) and PTDriver (yellow) agents are shown. Simulation events are depicted in the overlay window.

C. Result Reporting, Analysis and Visualization Tools

Recording simulation progress and results is a necessary part of simulation execution. To this end, AgentPolis provides a customizable logging mechanism employing the Java event bus programming concept that allows detailed recording of low-level simulation events (e.g. the start and end of the execution of activities and actions). From the recorded events, higher-level, aggregate performance metrics can be calculated and visualized using a customizable reporting pipeline. The pipeline is based on the open-source GIS software stack employing the PostGIS[6] spatially enabled database and the OpenGeo[7] interactive geovisualization framework. Powerful aggregation and filtering functions can easily be specified using the spatial extension of the SQL language supported by PostGIS. Together, these tools allow analysing and browsing simulation results at different spatial and temporal resolution.

The current version of the AgentPolis framework can be obtained from `http://agentpolis.org`.

VII. Example Models

We have successfully used the AgentPolis framework to implement several simulation models. The models cover a wide range of interaction-rich transport systems differing in a number of important characteristics, including the type and number of agents, the complexity of agent decision making, the type and number of transport modes present and the complexity of agent-to-agent interactions. The basic information about the implemented models is given in VI – below we describe each model in more detail.

[6]`http://postgis.net`
[7]`http://opengeo.org`

Model	# agents	Types of agents
Multimodal mobility	$10^5 - 10^7$	Urban citizen, PT driver, Driver
Ridesharing	$10^2 - 10^3$	Passenger, Driver, Dispatcher
Dynamic pricing	10^2	Passenger, Driver
Fare inspection	$10^4 - 10^5$	Passenger, Inspector
Parcel logistics	10^2	Dispatcher, Van driver

TABLE VI: List of implemented AgentPolis models with the overall number and the types of agents used.

A. Multimodal Urban Mobility

The multimodal urban mobility model is the most comprehensive and the largest model built using the AgentPolis framework, covering areas up to thousands of square kilometres and simulating populations of up to millions of inhabitants. Employing the activity-centric approach, the model aims to reproduce traffic flows in a multimodal urban transport system. The model is similar in purpose and scope to other activity-based mobility models but it is internally implemented in the fully agent-based way – this gives it the benefits associated with the agent-based approach, in particular the ability to model within-the-day decision making and to include ICT-powered mobility services relying on ad hoc inter-agent interactions in activity models.

Technically, the model utilizes most of the core AgentPolis model elements with the UrbanTraveller lifecycle being the basis of the agents representing the population of the modelled region. An overview of the main model elements used in the multimodal mobility model as well as in the other example models is given in Table VII.

B. Real-time Ridesharing

The real-time ridesharing model has been implemented for studying the performance of ridesharing services under different deployment conditions. The model comprises three types of agents: vehicle drivers (corresponding to drivers of collective taxis, flexible buses or shared private vehicles), passengers of the ridesharing service, and the dispatcher, who matches passengers with drivers and vehicles. While the dispatcher agent is completely new, the driver and the passenger agents largely reuse the core AgentPolis activities. New, model-specific logic consists of the negotiation protocol used to arrange shared trips and the associated decision logic on the side of participating agents. Extension on lower-level of the model, i.e. actions and sensors, were not required.

In its basic configuration, the ridesharing model only employs hundreds of agents directly participating in the modelled ridesharing service. Thanks to its fully agent-based design, it is, however, possible to combine the ridesharing model with the multimodal urban mobility model and to study interactions between ridesharing services and other mobility modes and services. See Table VII for the list of model elements used in the model.

C. Auction-based Dynamic Taxi Pricing

The dynamic taxi pricing model has been implemented for studying the effect of auction-based dynamic pricing of taxi

Abstraction	Element	Multimodal mob.	Ridesharing	Dynamic pricing	Parcel logistics	Fare inspection
Activities	`Walk`	●	●	●		●
	`RideInVehicle`	●	●	◇		
	`RideOnPT`	●				●
	`DriveVehicle`	●	●		●	●
	`ParkVehicle`	●				
	`Wait`	●	●	●	●	●
	`DriveTaxi`			+		
	`PatrolInStation`					+
	`PatrolInVehicle`					+
Env. Objects	`TransportNetwork`	●	●	●	●	●
	`PTStops`	●				●
	`Attractor`	●				
	`Vechile`	●	●	●	●	●
	`Warehouse`				+	
	`DeliveryPoint`				+	
	`VehicleInspectArea`					+
	`StationInspectArea`					+
Actions	`MoveVehicle`	●	●	●	●	●
	`MoveAgent`	●	●	●	●	●
	`TeleportAgent`	●				
	`GetInVehicle`	●	●	●		●
	`GetOffVehicle`	●	●	●		●
	`WaitForVehicle`	●	●	●		●
	`RideInTaxi`			+		
	`TaxiWaitForJob`			+		
	`LoadParcel`				+	
	`UnLoadParcel`				+	
	`UnLoadParcel`				+	
	`InspectPassengers`					+
	`ExistInspectArea`					+
	`EnterInspectArea`					+
Sensors	`PositionUpdated`	●	●	●	●	●
	`NextVehicleLoc.`	●	●	●	●	●
	`DrivingFinished`	●				●
	`WaitingFinished`	●	●	●	●	●
	`VehicleArrived`	●	●	●		●
	`PassengerInSight`					+
	`InspectorInSight`					+
Queries	`GetAgentPosition`	●	●	●	●	
	`GetPTStopPosition`	●				
Protocols	`1-to-1 Messaging`		●	●	●	
	`Auction`		+			
Reasoning modules	`JourneyPlanner`	●	●	●		●
	`EuclideanAStar`				+	
	`DistanceTripFinder`				+	

TABLE VII: The use of model elements in the example AgentPolis models. Core elements printed using normal font; newly added in italics. (● reused core element, ◇ modified core element, + newly added model element).

services. In contrast to the previous model, the modelled dynamic taxi pricing scheme relies on peer-to-peer interactions and only contains two types of agents: passengers and taxi drivers. Similarly to the ridesharing model, the taxi pricing model reuses a large part of framework's core model elements, with the majority of newly developed code concerning the auction protocol and the associated decision logic. In contrast to the ridesharing model, new activities related to travelling by taxi were added. Again, the taxi pricing model can be combined with the multimodal urban mobility model to study mutual interactions. See Table VII for the list of model elements used in the model.

D. Urban Parcel Logistics

The urban parcel logistics model has been implemented for studying the performance of parcel delivery services. The model comprises two types of agents: van drivers and dispatchers. Because of its focus on the transport of goods rather than people, the model lies outside the main focus of the AgentPolis framework and, consequently, provided an interesting test of the flexibility of the framework's design. The framework has passed the test successfully – although the model required the implementation of several model-specific elements at the environment level, these elements could be expressed using the AgentPolis abstractions. Specifically, we added depots and delivery locations as new types of environment objects together with actions and sensors related to parcel loading and unloading. See Table VII for the list of model elements used in the model.

E. Public Transport Fare Inspection

Finally, the fare inspection model has been implemented for studying the effectiveness of different strategies for conducting ticket inspection patrols in public transport networks. The model takes travel demand, ticket options and inspector patrol schedules as the input and produces inspection and fare evasion statistics as the output. Different passenger and fare evasion strategies, including the ability of passengers to avoid inspection through learning and communication, are modelled. The model uses two types of agents: passengers and ticket inspectors. The implementation of the model reused a significant portion of the core AgentPolis elements but also required the addition of a number of elements related to performing ticket inspections. See Table VII for the list of model elements used in the model.

Because of their strong reliance on modelling ad hoc interactions and just-in-time decision making, security models, such as this one, are another important category of interaction-rich transport systems that can benefit from the fully agent-based modelling supported by the AgentPolis framework.

F. Additional Models

We are currently considering the implementation of models of other ICT-powered transport systems, including demand-responsive fleets of driverless cars, peer-to-peer car sharing and smart parking schemes. We are confident that in their implementation, similarly to the models already implemented, it would be possible to reuse a large number of AgentPolis core model elements and that the extensions and additions required would be expressible using the abstractions of the modelling ontology.

VIII. DISCUSSION

The positive experience with the development of several models confirmed the viability of the fully agent-based approach, and the AgentPolis framework in particular, to modelling interaction-rich transport systems. The five models implemented represent a diverse set of models, each testing the flexibility of the framework in a different way. The framework proved capable of supporting models with a low number of computationally intensive agents (e.g. ridesharing or parcel logistics) as well as models with millions of lightweight agents (multimodal urban mobility). The latter is important because it shows that the higher flexibility of the fully agent-based approach does not come on the expense of degraded runtime performance of fully agent-based models. Furthermore, despite the diversity of the implemented models, the ratio between the reused and the newly developed code remained good, with the newly developed code mostly focusing on the logic specific to each model. Although is some cases significant extensions were necessary (in particular for parcel logistics and fare inspection models), they were easily accommodated by the framework.

There are still a number of open issues, though. The development of AgentPolis models remains a non-trivial task and requires model developers with good software design and implementation skills. In some cases, there are multiple ways in which a certain behaviour can be expressed in the framework but only some of them allow the model to fully leverage the strengths of the framework and its tools. At the moment, the modeller can refer to the example models for guidance on which abstractions should be employed for which purposes; in the future, we plan to make such guidance explicit in a set of model design patterns.

The above issue is also related to the fact that the simulation logic concerning a certain fragment of the modelled phenomena typically cuts across several modelling abstractions (in particular activities, actions, sensors and environment objects); the implementations of these abstractions thus need to be kept consistent, which is not easy. Although such a mutual dependency problem cannot be fully solved and affects all extensible simulation platforms, there are ways in which the burden on the modeller can be reduced and which we consider for the future versions of the framework. A more usual way to address the dependency problem is to provide a set of well-defined and encapsulated extensions points, which would reduce the need to modify core model elements and consequently shield the developer from having to understand their exact interdependencies. This approach would be particularly efficient if the scope of the framework is narrowed. Focusing, e.g., solely on modelling on-demand mobility services (such as ridesharing) would allow fixing the majority of lower-level model elements; the model developer would then only implement higher-level model logic governing the arrangement of rides but not their actual execution. In a longer run, the maintainability and extensibility of the framework could be improved by employing more modular programming abstractions – such as traits or lambda expressions – available in some progressive programming languages now and coming to Java in a near future.

At the moment, the AgentPolis framework provides the strongest support and guidance for modelling the environment and agent-to-environment interactions. The support for modelling agent behaviour, on the other hand, remains relatively basic, with activities and reasoning modules as the only supporting abstractions. This is partly intentional because of the large diversity of agent behaviours and the notorious difficulty to provide a flexible set of abstractions for programming general agent behaviour. That said, we aim to improve the support for behaviour modelling by providing simple yet proven behaviour programming abstractions such as (hierarchical) finite state machines.

IX. Conclusions

We have developed a modular framework for the implementation of simulation models of interaction-rich transport systems. The framework fully adopts the agent-based modelling paradigm, which makes it very versatile and capable of modelling systems with complex ad hoc interactions and just-in-time decision making. We have used the framework to implement models of five different transport systems. The positive experience obtained has confirmed the effectiveness of the fully agent-based approach in general and the AgentPolis framework in particular in quickly building models of different kinds of interaction-rich transport systems.

References

[1] M. Balmer, K. Meister, M. Rieser, K. Nagel, and K. W. Axhausen. Agent-based simulation of travel demand: Structure and computational performance of MATSim-T. In *TRB Conference on Innovations in Travel Modeling*, 2008.

[2] J. Banks, J. S. Carson, B. L. Nelson, D. M. Nicol, et al. *Discrete-event system simulation*. Pearson Prentice Hall Upper Saddle River, NJ, 2005.

[3] M. Barth and M. Todd. Simulation model performance analysis of a multiple station shared vehicle system. *Transportation Research Part C: Emerging Technologies*, 7(4):237–259, 1999.

[4] S.-F. Cheng and T. D. Nguyen. Taxisim: A multiagent simulation platform for evaluating taxi fleet operations. In *Proceedings of the 2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology-Volume 02*, pages 14–21, 2011.

[5] F. Ciari, M. Balmer, and K. W. Axhausen. Concepts for large-scale carsharing system: Modeling and evaluation with agent-based approach. In *Transportation Research Board 88th Annual Meeting*, number 09-1888, 2009.

[6] F. Ciari, M. Balmer, and K. W. Axhausen. Large scale use of collective taxis. Technical report, ETH, Eidgenössische Technische Hochschule Zürich, IVT, Institut für Verkehrsplanung und Transportsysteme, 2009.

[7] P. M. d'Orey, R. Fernandes, and M. Ferreira. Empirical evaluation of a dynamic and distributed taxi-sharing system. In *Proceedings of the 15th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 140–146, 2012.

[8] M. Horn. Multi-modal and demand-responsive passenger transport systems: a modelling framework with embedded control systems. *Transportation Research Part A: Policy and Practice*, 36(2):167–188, 2002.

[9] E. Lioris, G. Cohen, and A. de La Fortelle. Overview of a dynamic evaluation of collective taxi systems providing an optimal performance. In *Intelligent Vehicles Symposium (IV), 2010 IEEE*, pages 1110–1115. IEEE, 2010.

[10] L. M. Martinez, G. Correia, and J. Viegas. An agent-based model to assess the impacts of introducing a shared-taxi system in lisbon (portugal). In *Proceedings of the 7th International Workshop on Agents in Traffic and Transportation*, 2012.

[11] F. Michel, J. Ferber, A. Drogoul, et al. Multi-agent systems and simulation: a survey from the agents community's perspective. *Multi-Agent Systems: Simulation and Applications*, 2009.

[12] L. Quadrifoglio, M. M. Dessouky, and F. Ordóñez. A simulation study of demand responsive transit system design. *Transportation Research Part A: Policy and Practice*, 42(4):718–737, 2008.

[13] G. Wainer. Developing a software toolkit for urban traffic modeling. *Software: Practice and Experience*, 37(13):1377–1404, 2007.