Athos: An Extensible DSL for Model Driven Traffic and Transport Simulation

Benjamin Hoffmann[1] Neil Urquhart[2] Kevin Chalmers[3] Michael Guckert[4]

Abstract: Multi-agent systems may be considered appropriate tools for simulating complex systems such as those based around traffic and transportation networks. Modelling traffic participants as agents can reveal relevant patterns of traffic flow. Upsurging traffic in urban areas increases the relevance of such simulations and the insight they provide into reducing congestion and pollution. Developing multi-agent traffic simulations is a challenging task even for professional software developers. In contrast, domain experts need tools that can be quickly adapted to new questions emerging in their research without potentially error-prone communication with software developers. There is a need for simulation tools that are intuitive to domain experts yet flexible and adaptable by software developers as required. A model driven approach with an extensible domain specific language delivers an answer for both of these opposing requirements. The modeller is relieved from implementing time consuming programming details and can focus on the application itself. We present the domain specific language Athos that allows to create simulations of traffic and transport related problems declaratively. The models are platform independent and executable code can be generated for appropriate multi-agent platforms. The language is flexible and can be easily extended by exploiting the structure of the problem domain itself. In this paper, we present Athos and focus on how it can be extended by arbitrary traffic and routing algorithms through an annotation-based extension mechanism.

Keywords: Domain Specific Language; Traffic Simulation; Multi Agent System;

1 Introduction

Human cognition alone can no longer comprehend the growing complexity observed in many world systems. Instead, such systems must be analysed using experimentation and simulations that reveal some of the system's internal mechanisms. Transport networks and the associated traffic flows in urban areas are examples of such complex systems. Increasing traffic, congestion, and a rising number of last-mile deliveries call for new ideas that must be assessed before implementation because real-world testing would be too expensive. Tools that allow realistic simulation of alternative traffic scenarios offer insight into the feasibility and effects of the options.

[1] Kompetenzzentrum für Informationstechnologie – Technische Hochschule Mittelhessen benjamin.hoffmann@mnd.thm.de
[2] School of Computing – Edinburgh Napier University N.Urquhart@napier.ac.uk
[3] Department of Media, Culture and Language – University of Roehampton Kevin.Chalmers@roehampton.ac.uk
[4] Kompetenzzentrum für Informationstechnologie – Technische Hochschule Mittelhessen michael.guckert@mnd.thm.de

Rapidly changing requirements and the potential variety of alternative scenarios require software solutions that can be easily adapted and extended. When using general-purpose languages (GPLs), traffic domain experts must rely on software engineers to modify the system. Communication of experts of different domains is a known source of misunderstanding and may lead to weak models and error-prone systems [DC12]. The typically low-level abstraction in GPLs prevents reuse of larger building blocks so that implementations have to start from scratch each time a new problem has to be solved. Conversely, using a proprietary simulation platform often does not provide sufficient flexibility to deliver answers to specific questions and may create undesirable dependencies.

Model-driven approaches offer potential to overcome this dilemma (see [Ho18a]). Domain-Specific Languages (DSLs) can overcome the problems of platform dependency and miscommunication. Due to their higher abstraction level, and a notation specific to the underlying domain, they more expressiveness [va00, do12]. With our DSL, named Athos, domain experts can specify traffic simulations and related optimisation problems declaratively without lower-level programming concerns. Athos models follow a multi-agent paradigm and are accessible to humans and can be transformed into executable programs for given target platforms (e.g., NetLogo[5] or Repast Simphony[6]).

As a language for modelling simulations and optimisation problems, Athos targets domain experts as language users. Algorithms are considered to be part of the supporting infrastructure that runs the simulations. Therefore they are not implemented in the Athos language itself. This approach keeps models as computationally-independent as possible. Nevertheless, Athos has an elaborate mechanism that allows developers access and integrate external algorithms into the development and runtime environment. This is an important feature, as, in dynamic simulations of traffic scenarios, optimisation problems have to be solved to measure relevant indicators of the subject of investigation. For example, when simulating a multi-vehicle delivery problem, we must first solve the underlying routing problem and then test against dynamic traffic patterns (see [Ho18b, Ho19b]). According to the philosophy of Athos, implementing basic algorithms is not within scope and must be provided externally through problem solvers that must integrate transparently. This way, Athos is an interface between domain experts and algorithm developers from academia. Domain experts do not only put academic algorithms to real-world application, but also define requirements and discover potential for improvement. Algorithm scientists possess the expertise required for the creation of efficient algorithms that fulfill the defined requirements.

The rest of this paper is organised as follows: Section 2 introduces Athos, points out the need for an extension mechanism, and outlines how such a mechanism was implemented for Athos. Section 3 provides an example that illustrates how to extend Athos by integration of an external algorithm. In Section 4, other languages in the domain of agent-based traffic modelling are discussed. Finally, section 5 concludes the paper and identifies important future work.

[5] `https://ccl.northwestern.edu/netlogo/`
[6] `https://repast.github.io/`

2 Athos

Athos is a DSL for the specification of traffic and transport simulations that comprise vehicle routing problems (VRPs). Written with the Xtext framework[7], it features a full set of tools for developing models (e.g. a textual editor, model checking, and a generator). Our approach is platform independent, though we currently use NetLogo as our main target platform. This section introduces the reader to the language before it discusses the challenges of the domain. These challenges are then shown to be the catalyst that led to the development of the extension mechanism presented in the final part of this section.

2.1 A Domain-Specific Language for Vehicle Routing Problems

Athos is a declarative language designed for domain experts with little to no programming experience. In a typical traffic and transport scenario, vehicles, agents (the terms are used interchangeably in this paper) aim to perform given tasks or solve predefined problems, such as following a predefined route or delivering goods from depots to a list of targets. Athos is designed for tasks ranging from simple routing problems – e.g., the travelling salesman problem (TSP) or the vehicle-routing problem with time-windows (VRPTW) – to models of complex urban scenarios that respect the dynamics of traffic flow. In Athos, agents mutually influence each other. Congestion effects that occur due to parts of the road network being frequented by too many agents can thus be observed and examined. Due to congestion, agents may reconsider their original plan and adapt their behaviour according to the present situation; e.g. by avoiding congestion. Athos can thus be used to specify scenarios in which emergent phenomena may occur and new insights into the flow of traffic gained.

```
1  model VRPTW_Example
2   world xmin 0 xmax 75 ymin 0 ymax 75
3   products product soap weight 1.0
4   agentTypes
5   agentType staticDelivery congestionFactor 60.0 maxWeight 200.0
6      behaviour awt awaitTour when finished do die;
7      behaviour die vanish;
8   functions
9   durationFunction normal  length default
10  complete network
11  nodes
12  node n1 (35.0, 35.0)
13  ...
14  node n51 (47.00, 47.00)
15 edges
```

List. 1: General structure of an Athos model

Listing 1 demonstrates the structure of an Athos program. It defines a network, agent behaviour and details of the delivery problem. The network or graph defines the roads (edges) that vehicles can traverse. A complete graph is generated and the Euclidean distance

[7] https://www.eclipse.org/Xtext/

is used as length. Alternatively, a graph can be specified with individual attributes for its edges. Listing 2 demonstrates how sources and demands can be specified using nodes of the network. The keyword `ea` indicates that an evolutionary algorithm is to be used to compute the tours for the vehicles. Additional parameters for the algorithm are provided. For each demand node, quantities, time windows, and service time are defined.

Implementing such simulations with a GPL and using problem solver libraries leads to code in which the original problem is difficult to manipulate. By contrast, Athos is a declarative language that focuses on the specification of the actual problem rather than its solution. The solution is determined by the behaviour of the agents. In Athos, behaviour specifications are encapsulated in Agent-Behaviour Blocks (ABBs) that are associated with appropriate algorithms (see Listing 2). These algorithms provide solutions for the respective problem which are finally re-translated into observable agent behaviour.

```
1 sources
2     n1 isDepot soap sprouts (staticDelivery)   agentsStart route
3      ea (n2, n3, n4, ..., n48, n49, n50, n51) popSize 30
4 demands
5      n1 hasDemand soap absQuantity 0.00 earliestTime  0
6         latestTime 230 serviceTime 0
7      ...
8      n51 hasDemand  soap absQuantity 13.00 earliestTime 124
9         latestTime 134 serviceTime 10
```

List. 2: Sources and demand specification in Athos

Athos models are processed by a generator that creates code for an appropriate target platform. In order to create this code, the generator applies a set of transformations to the model and generates code that can be executed on the target platform. Currently, the generator features a complete set of transformations for the NetLogo platform. NetLogo is a DSL for multi-agent programming that is supported by a suitable simulation environment [TW04]. Though NetLogo specialises on the *technical domain* of multi-agent simulations, it is not tailored towards a specific *application domain*. Therefore, a language like Athos, which is specifically designed for the domain of traffic and transport simulation, further facilitates the creation of appropriate models. Since Athos models are platform independent, the creation of additional transformation sets designed for other target platforms is straightforward and has already been described in our previous work [Ho18a].

2.2 Modelling Vehicle Routing Problems

Since Athos is tailored towards the domain of traffic simulation and transport optimisation, the language must offer means to concisely formulate VRPs. Thus, the meta model of the language provides elements to define various types of routing problems (see [Ho18a, Ho18b, Ho19a]). Surveying the literature reveals that there is a range of problems that can be subsumed under the general term VRP.

From a mathematical point of view, the process of adding additional requirements to a problem is referred to as the *generalisation* of the problem. For example, a TSP can be generalised to a problem that defines time windows for each node that must be visited. The TSP may then be considered as a special case of the more general problem, in which the time windows are maximised. From the perspective of modelling this relation can be inverted as the TSP with time windows owns potentially more attributes and each TSP with time windows certainly is a TSP. From a syntactical object-oriented perspective, adding attributes and behaviour specialises a more general concept. In our hierarchy we consider a TSP with time windows as a *specialisation* of TSP, i.e., TSP is the more general problem.

The family of VRPs can be arranged in a taxonomy. At the top, there is the most general problem that offers the fewest definable attributes. The deeper the problem is located within this hierarchy, the more variables are required by the problem type. The most general problem type is the TSP (see Dantzig et al. [DFJ54] for its original formulation). The TSP can be modelled on a complete graph with n *vertices* or *customers* $c_i (i = 1, \ldots, n)$ are connected by *edges* (or arcs, depending on whether the problem is *symmetric* or *asymmetric*, respectively). Beginning at a given vertex, referred to as the *base* or *depot*, we find a minimal-cost path of nodes (costs occur upon usage of an edge/arc that connects the two respective nodes) that starts and ends at the depot and visits every other vertex exactly once.

By adding the requirement that the touring *vehicle* must return to the depot after m nodes have been visited, we get the *Clover Leaf Problem (CLP)* [DFJ54].The TSP can be considered a special instance of the CLP in which $m \geq n - 1$ holds. Nevertheless, we see the CLP as a more specific problem as the TSP and classify it as a successor of the TSP in our taxonomy.

In the *Capacitated Vehicle Routing Problem (CVRP)* [BTV10], each vertex is assigned a predefined *demand* d_i $(i = 1, \ldots, n)$, while the vehicle has a limited *capacity* q. Since the vehicle may not perform partial deliveries, it is assumed that none of the demands exceed the capacity of the vehicle.

The *Heterogeneous Vehicle Routing Problem (HVRP)* [BBV08] accepts different vehicle types with different capacities, unlike the one vehicle type allowed in the CVRP. This problem can be further extended by introducing:

- *fixed costs* for each vehicle type that occur upon deployment of a vehicle of the respective type (the problem is then referred to as the HVRP with Fixed-costs (HVRPF)).
- vehicle-type dependent routing costs, i.e., the costs that occur for travelling a given edge depend on the type of the respective vehicle (this is known as the HVRP with vehicle-Depending routing costs (HVRPD))

The HVRP is a special case in which the fixed costs are set to zero and the costs that occur are the same for all vehicle types (c.f. [BBV08]).

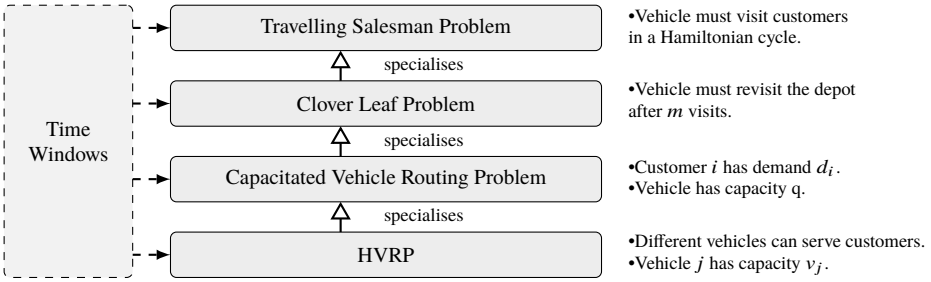

Fig. 1: Excerpt of Vehicle Routing Problem hierarchy

All the problem types defined above can further be extended by assigning time-windows to each customer. A time-window is a time-interval in which a visit must begin. If the vehicle arrives early, it is required to wait until the customer is ready (start of the interval). Conversely a vehicle that arrives after the end of the time window will be unable to service that customer. The depot is assigned a time-window which defines the time by which all vehicles have to return. Figure 1 illustrates the described hierarchy with the *generalises* relationships and the possible adoption of time windows.

This problem hierarchy represents a small excerpt of the taxonomy of VRP types found in the literature. Algorithms for solving such problems have been published and, as most of the problems are NP-hard, for larger problem instances meta-heuristics are applied to find feasible solutions of sufficient quality in a reasonable time. Algorithms and heuristics have elaborate sets of parameters controlling the operation of the computations for which appropriate values have to be provided. Athos transparently integrates such algorithmic solutions for problems underlying the simulations with an open mechanism for extended problems and algorithms. The language design must therefor address three major issues:

1. It must allow for an integration of various algorithmic approaches (e.g., ant colony-based or evolutionary algorithms) for each specific type of problem.
2. It must act as an interface that allows users to pass parameters together with their respective values to the chosen algorithms.
3. The language must be designed in a way so to support modelling the entire VRP hierarchy as far as reasonably possible.

2.3 Extensibility of the Language

Athos features a built-in extension mechanism that allows *algorithm developers* to integrate their implementations into the Athos environment. There is no need for a *language developer*

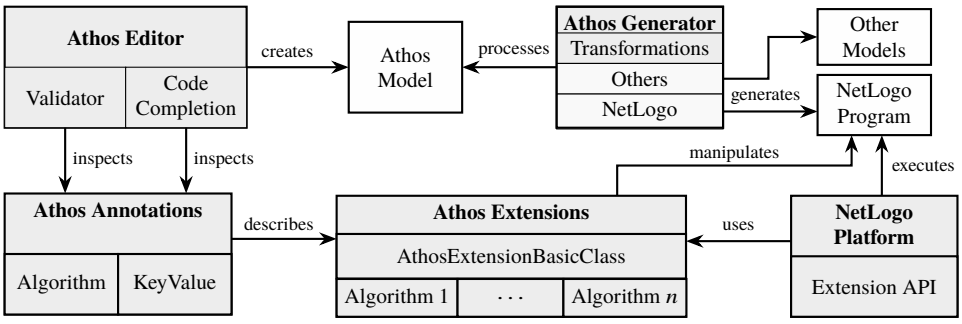

Fig. 2: Architecture of the Athos Development Environment

to change definition of the DSL; i.e., its abstract or concrete syntax, or its static semantics. There is also no need for a language developer to adapt the transformations to the respective target platform. Instead, all an algorithm provider needs to do is to create a Java class that extends a given base class, using two Java annotations provided by the Athos framework. The domain expert will then be able to use the newly integrated algorithm almost as if it was natively supported by the environment.

Figure 2 illustrates the general architecture of the DSL and its development framework. In the described scenario, NetLogo is used as a target platform. However, this approach works in principle with any Java-based multi-agent-simulation platform. Inside the Athos editor, an Athos model is developed and then further processed by the generator. The generator uses a subset of its transformations in order to generate a program that can be run on the NetLogo multi-agent simulation platform. The generated NetLogo program requires the provision of a special Athos extension package referred to as the *generic optimisation algorithm library*. Note that different target platforms might require to adapt this library to the respective target platform. The package encapsulates all routing-related algorithms that can be used inside the NetLogo program. In order to integrate an algorithm into the DSL framework, algorithm providers have to extend the base class `ExtendedOptimisationBehaviourImplementation` and use two distinct annotations provided by Athos. These annotations provide information to the extension algorithms. Moreover, the Athos Editor can use the information provided with these annotations to check certain constraints and provide code completion proposals. It is also important to note that an algorithm in the extension package can also manipulate the specifications of the NetLogo program. In the case study in Section 3, this will be used to assign routes of nodes to the vehicles.

Figure 3 provides an overview of the annotations and their respective attributes required for the integration of an algorithm into the Athos framework. The first of these two annotations is `@Algorithm` that marks a class as an algorithm that will be accessible in an Athos model. It features a `name` attribute that is used to assign a name to the algorithm for invocation inside an Athos model. The second annotation is `KeyTypeSet`. It is used to

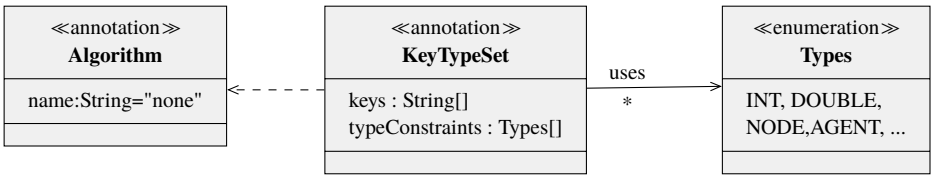

Fig. 3: Annotations Used to Integrate Additional Algorithms into Athos

Type	Explanation
`INT`,`DOUBLE`, `STRING`	Primitive types must meet the lexical requirements of the respective type.
`AGENT`	Parameter must refer to an existing agent type declared in the agents section of the model.
`NODE`	Parameter must refer to an existing node declared in the network.
`START_POS.`	Parameter must refer to a node in the network. This node should also sprout (create) the correct number of vehicles.
`LINK`	Parameter must refer to an edge declared in the network.

Tab. 1: Type System Applied in the Athos Extension Annotations

provide additional information that cannot be modelled with the native language elements provided by Athos. As an example, consider the implementation of a new heuristic that features a probabilistic parameter. It can also be used to model vehicle routing problems for which Athos does not natively feature the required modelling elements. The `KeyTypeSet` annotation has a `String` and a `Type` array as its attributes. Algorithm developers can assign a name to each parameter required by their algorithms and at the same time ensure type-safety by providing a type for each parameter. The parameter names and their types simply have to be at corresponding positions inside the respective array, i.e., the parameter with the name provided at `keys[0]` is of type `typeConstraints[0]` and so on. Parameters can have the primitive types string, integer, double. They can also have the complex types node, `START_POSITION`, or `AGENT`. Each of these types can also be defined as a one- or two-dimensional array. Table 1 gives an overview of the allowed parameter types together with a brief explanation of their semantics (an example will also be discussed in the case study in Section 3).

```
1 @Algorithm (name="EAForVRPTW")
2 @KeyTypeSet(
3     keys= { "startCity", "customers", "vehicleCapacity",
4             "populationSize","probabilityForGreedyCreation"},
5     typeConstraints={Types.NODE, Types.NODE_ARRAY, Types.INTEGER,
6         Types.INTEGER, Types.DOUBLE}
7 )
8 public class EvolutionaryAlgorithmForVRPTW
9     extends ExtendedOptimisationBehaviourImplementation {
```

List. 3: Annotation of an Algorithm for VRPTW

In [Ho19a], we describe the implementation of an evolutionary algorithm for the VRPTW based on that presented by Ombuki et al. [Om06]. Since this algorithm was used in the process of bootstrapping Athos, it is natively supported by the language. However, Athos is also open for any other algorithm that solves VRPTWs. Listing 3 provides an example that shows how the information required by Ombuki's algorithm could have been provided using Athos' extension mechanism. The `@Algorithm` annotation in line 1 marks the class as an algorithm that can be invoked from an Athos model with the name `EAForVRPTW`. This evolutionary algorithm requires some additional parameters (some omitted for brevity): The first parameter, `startcity`, is required so that the algorithm knows the location of the depot. The `customers` node array represents the customers of the problem – the nodes that have to be visited. The integer parameter `vehicleCapacity` sets the maximum capacity of the homogeneous fleet of vehicles assumed in the algorithm. The `populationSize` parameter of type integer is a common parameter for evolutionary algorithms that determines the number of chromosomes per generation. The parameter `propabilityForGreedyCreation` of type double is more special to the chosen algorithm. In the initialisation phase of the algorithm, it determines the probability for a new chromosome to be created by the application of a greedy heuristic. With the complementary probability the customers represented by the chromosome are simply ordered randomly.

```
1 sources
2  n1 isDepot soap sprouts (delivery1) agentsStart extended EAForVRPTW
3     startCity (n1)
4     customers (n2, n3, n5, n9, n10, n12, n15, n22)
5     vehicleCapacity 200
6     populationSize 20
7     probabilityForGreedyCreation 0.7
8     at 2
```

List. 4: Example Model With Behaviour Specification

The annotated algorithms in the package are inspected by the Athos framework. The validator uses the information provided via the annotations to determine new keywords that are valid when defining depots in the network. Listing 4 defines node `n1` as a depot from which a homogeneous fleet of vehicles with a capacity of 200 start their tour. In addition to the problem-related parameter values, the model also specifies parameter values that affect the execution of the algorithm. In this example, the population size is set to 20 and the probability that a chromosome is created with a greedy approach is set to 70 per cent.

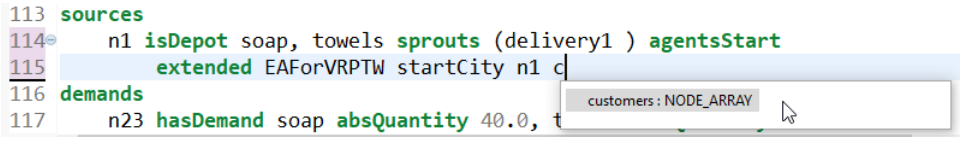

Fig. 4: Proposal Provision in the Athos Editor

Additionally, the proposal provider inspects the annotations in order to provide valid proposals. This is especially important given that the targeted users of the language are

domain experts rather than professional software engineers. In Figure 4, it is illustrated how the Athos IDE provides support in the invocation of the `EAForVRPTW` algorithm. After the specification of the name of the algorithm, the Athos framework inspects the associated `KeyValue`-Annotation in order to help the user specify the parameters in the correct order and with the correct types. In the illustrated example, the framework can deduce from the position of the cursor and the previous information, that the next parameter to be defined must be the `customers` parameter. A corresponding proposal is thus provided to the user who runs little risk of specifying the parameters incorrectly. The framework can also check that the provided information matches the types expected by the algorithm.

3 Example Use Case

In this section, we will provide a detailed examination on how to use the extension mechanism of Athos in order to integrate and apply new routing algorithms. We will show that this does not only allow integration of the most efficient algorithms, but also to extend the language and its infrastructure in a way that new problem types can be modelled and solved.

3.1 The HVRPD and an Instance Model

In section 2.2, the HVRPD was defined as a generalisation of the TSP from a mathematical point of view and as a specialisation of the TSP from an object-oriented point of view. In this example use case, we will model an even more specialised variant of the HVRPD. Here, there is not only one but two different products that are to be delivered. Hence, for each customer, two different demands have to considered, and for each vehicle type of the heterogeneous fleet two different capacities have to be modelled. Moreover, in this variant of the VRP, the vehicles do not start their tour from a central depot but from different nodes of the network. However, for the sake of modelling, the vehicles are regarded as being coordinated by one depot inside the network, even though they are not parked there.

Listing 5 shows how the HVRPD with multiple products and non-central vehicle starting locations can be modelled with Athos. The first two lines set the formal name of the model and the boundaries of the environment. Line 3 defines the product types that are to be delivered. In line 4, the required agent attributes are declared. As each vehicle has routing costs and a maximum quantity for each product, there are three declared attributes. Lines 5 to 15 define the agents that form the heterogeneous vehicle fleet. In this example, the fleet consists of three different agent types. The declaration of the first vehicle type spans from line 6 to 11. Lines 7 to 9 assign an actual value to each of the declared agent attributes. The behaviour exhibited by the first vehicle type is modelled in lines 10 and 11. The modelled behaviour has the vehicle waiting at its starting location (in a parking position) until it is assigned a route by the coordinating depot. After the route was performed, the agent is to disappear from the simulation.

```
1  model id001
2  world xmin 0 xmax 40 ymin 0 ymax 22
3  products product soap, product towels
4  agentAttributes soapCapacity, towelsCapacity, costPerDistance
5  agentTypes
6      agentType delivery1 congestionFactor 0.0
7          attr soapCapacity  120.0
8          attr towelsCapacity 240.0
9          attr costPerDistance  1.0
10         behaviour awt awaitTourExternal when finished do die;
11         behaviour die vanish;
12     agentType delivery2 congestionFactor 0.0
13         // omitted for reasons of brevity
14     agentType delivery3 congestionFactor 0.0
15         // omitted for reasons of brevity
16
17         // network specification omitted for reasons of brevity
18
19 sources
20     n1 isDepot soap, towels sprouts (delivery1 )
21     agentsStart extended JSpritHVRPSolver
22         customers (n1, n2, n3, n4) demands ("soap", "towels")
23         vehicleTypes ("delivery1", "delivery2", "delivery3")
24         capacities ("soapCapacity", "towelsCapacity")
25         costsPerDistance (1.0, 1.1, 1.2) instancesPerType (2.0, 1.0, 1.0)
26         startLocations ((n1,n1), (n2), (n4)) at 2
27     n1 sprouts quantity 2 (delivery1) at 1
28     n2 sprouts quantity 1 (delivery2) at 1
29     n4 sprouts quantity 1  (delivery3) at 1
30 demands
31     n23 hasDemand soap absQuantity 40.0, towels absQuantity 50.0
32     // additional demands omitted
```

List. 5: General Structure of an Athos Model

The declaration of the other types is done analogously and was omitted for reasons of brevity. The same goes for the specification of the network which was already explained in section 2.1. Line 19 starts the definition of sources – nodes inside the network where vehicles enter the simulation. Line 20 marks node `n1` as a depot. The `agentsStart` keyword followed by the `extended` keyword allows the invocation of an algorithm that possesses the aforementioned extension annotations required by Athos.

Lines 21 to 26 represent the code that invokes an externally provided algorithm. Line 21 specifies that the `JSpritHVRPSover` provided in the generic optimisation algorithm library is to be invoked. Line 22 specifies that `n1, n2, n3, n4` are the `customers` that have to be supplied with `soap` and `towels`. The actual demands of the customers are specified beginning in line 31. However, it would also be possible to use the extension mechanism to specify the demands of each customer[8]. Line 23 specifies the vehicle types that comprise the heterogeneous fleet controlled by the depot. Line 24 is required so that the algorithm can infer how to query for the actual capacities of a vehicle type[9]. In line 25, the

[8] This could actually be achieved in more than one way. It is possible to define two arrays, one containing the soap and the other the towel demand for each customer. Alternatively, a two-dimensional array could be specified that has both demands for each customer inside a tuple.

[9] Not every agent attribute is necessarily a capacity attribute. In lines 7 to 9 only two of the defined attributes are actually capacity attributes. To allow the algorithm to infer this fact, this information has to be provided

distance-dependent routing costs for each vehicle are defined, e.g. one distance unit travelled by a vehicle of type `delivery1` adds 1.0 cost units while the same distance adds 1.2 cost units when a vehicle of type `delivery2` is used. In the same way, the number of actual instances of each vehicle type is defined in this line. Finally, in line 26 the starting positions of the vehicles inside the network are defined. The two instances of type `delivery1` both start at node `n1`. The instance of `delivery2` starts at node `n2`, while the vehicle of type `delivery3` begins its tour at node `n4`. The keyword `at` defines at what point in time (which is measured in ticks) the coordinating depot has the vehicles start their tour.

It is important to note, that in this scenario the `startLocations` parameter is used to inform the algorithm on where the vehicles are to start their tour, i.e., at what node they wait for an order from the coordinating depot. However, the creation of these vehicles is initialised by the code provided in lines 27 to 28. The nodes defined as start locations in line 26 have to correspond to the actual start locations defined in lines 27 to 28. To enable the validator to check the model for this correspondence, the `startLocations` parameter was assigned the type `Types.START_POSITION_ARRAY_ARRAY`. This way, the validator automatically checks that each node declared as a starting location creates a sufficient number of vehicles prior to the execution of the algorithm. However, if the constraint fails, the validator only issues a message instead of an error. The reason for this is that the algorithm creator can perform the creation of the cars inside the algorithm.

3.2 Implementation of the Algorithm

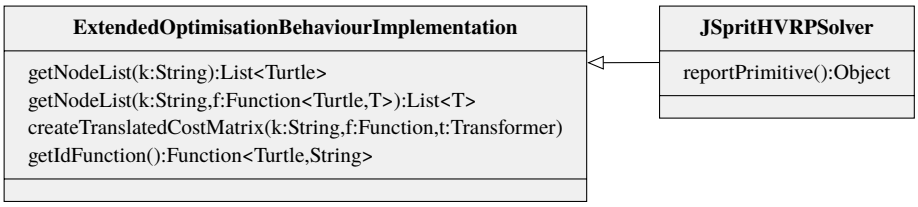

Fig. 5: Convenience Methods Provided by the Extension Framework to Extension Algorithms

The annotation of the algorithm functions as discussed in section 2.3 and exemplified in Listing 3. An example implementation is provided online[10]. As is depicted in Figure 5, classes that implement an algorithm must implement the `reportPrimitive()`-Method. The obtainment of the required extension parameter values is facilitated by convenience methods (e.g. `getNodeList()`) provided by the base class. Though the extension mechanism works for all Java-based simulation platforms, in this example the extensions for the NetLogo platform are presented. Since the algorithms that solve the problems are independent of the NetLogo platform, there must be a conversion mechanism, that allows a mapping from NetLogo nodes (known within NetLogo as `Turtles`) to the corresponding node data structures used in the algorithm (e.g. `Location` that can be identified by a

[10] `https://athos.mnd.thm.de/public/JSpritHVRPSolver.txt`

`String` id). For this, a default mapping `Function` is provided by a base class that can be changed by overwriting the `getIdFunction` method. The base class also provides facilitated access to the agents of the simulation together with their data structures. This way, the algorithm can easily obtain and manipulate data from entities of the simulation. This mechanism is used to equip each agent with a tour from the solution of the algorithm.

4 Related Work

Literature on DSLs specifically designed for the domain of transportation and traffic scenarios incorporating multi-agent environments and VRPs is scarce. Research has been published that examines general DSLs with a domain in the context of multi-agent systems. This section compares Athos with DSL-like approaches as well as with simulation platforms.

Steil et al. [St11] present a model for the expression, execution, evaluation and engagement of routing plans for patrols (e.g. police patrols) which they call the 4Es model. A DSL named *Turn* is used to describe algorithms that allow an agent to successively determine the next node on a route in a given graph network. Turn lets the user specify algorithms for the target selection as a composition of *set reduce functions (SRFs).* An SRF takes a set of possible target nodes and reduces this set according to certain criteria. Turn is used to chain and configure an arbitrary number of SRFs. An SRF chain is executed until only one node is left which then becomes the next target node. If the subset comprises multiple nodes, one of these nodes is chosen randomly. Algorithms written in Turn are executed by the *PatrolSim* environment which also evaluates routes produced by the algorithms according to a predefined set of metrics. A geographic information system (GIS) engages users through provision of patrol routes upon request.

Similar to Athos, the 4Es model can be used to model specific routing problems. The creation of satisfactory patrol routes is a problem related to the VRPTW. Though Athos and Turn can both be used to direct agents through a network of nodes, they differ in the way target nodes are specified. While Turn is mainly a composition of rules that successively reduce a set of potential target nodes, Athos allows an explicit specification of which node (or set of nodes) to visit next or requires that a set of nodes must be visited in an optimal order according to a given cost function. Another difference is the way behaviour changes are defined in both languages. In Turn, events that lead to a change of behaviour are transparently “hidden” inside the SRF and controlled by changing the SRF. Whenever an SRF returns an empty set of nodes it is skipped and replaced by another SRF which leads to a change of behaviour. In Athos, it is possible to explicitly state when a change of behaviour has to occur.

While in the 4Es model the cost to travel between any two nodes is defined as the minimum number of vertices between these nodes (i.e., the *minimum hop-count*), Athos allows the definition of arbitrary cost functions on the edges of the network. Turn does not reflect dynamic changes in the network, but uses the same distances throughout the entire simulation.

Athos cost functions can contain factors that dynamically change depending on the current traffic situation and allow for richer scenarios.

The GAMA framework [Gr13] supports the creation of spatialised, multi-scale agent-based models. It features a DSL named GAML with which agent-based simulation experiments can be designed. Hence, it seeks to raise the abstraction and allow for easy parameterisation and flexible visualisation of the models. GAML allows description of arbitrary details of the model. Multiple levels of abstraction can be used inside a simulation. So it is possible to look at a coarse-granular model for a road network with multiple detailed models that represent the inside of buildings adjacent to the roads. GAML can be considered a language that increases the abstraction level of models compared to those created in other simulation platforms like Repast Simphony or Netlogo. Since the GAMA framework does not focus on one specific application domain, models written in GAML can get complex. Data types and the definition of aspects to visualise agents in the simulation are hard to understand for domain experts with only limited experience in software engineering.

MATSim [HNA16] is a multi-agent microsimulation system based on a principle of co-evolution, in which agents are equipped with a set of plans which they can try and evaluate. In each cycle a plan is selected, applied and evaluated. With a given probability, agents modify different dimensions of their plans. For example, agents can vary the time they leave a given location, choose a different route or switch to a different mode of transport. Each agent seeks to optimise its individual outcome. Macijewski and Nagel present a MATSim-based approach to evaluate algorithms for the DVRP [MN12]. Their work intends to plan demand-responsive transport services (DRT severcies) using the MATSim framework. Using the MATSIm framework again requires deep programming and does not have a declarative perspective which we find more suitable for domain experts.

All of the approaches mentioned above can be used to model real-world VRPs and simulations. However, none of them can be used to generate traffic simulations from a pure specification in a DSL without considerable additional effort. This effort would be required for the development of either an appropriate simulation platform that processes the specification or a generator which would transform the specification to an executable traffic simulation. In this aspect, Athos is a consequent implementation of an instrument with which models of dynamic routing problems can be formulated and solved thus making use of the capabilities of the agent-based philosophy.

5 Conclusion and Future Work

We have demonstrated how Athos was opened for integration of additional VRP algorithms by the introduction of an annotation-based extension mechanism. We have also shown how this extension mechanism can be used to control arbitrary parameters of additional algorithms and how it allows for modelling additional problem properties not natively supported by the DSL without changing core elements of the DSL. Moreover, we discussed

how this extension mechanism can feature constraint checks. We see two perspectives that have to be further elaborated next. One is the flexibility of the language for both the language developer and the domain expert and the other is a systematic evaluation of the language.

Flexibility of Language The generic nature in which problem solvers can be accessed in the generated code allows for future extension of the system. In section 2.3 we presented two important roles: the *domain-expert* uses the language for their research into traffic and transportation, the *algorithm developer* can extend the language. Domain-experts identify specific requirements and formulate them so that software engineers can then implement appropriate Athos extensions. Such individual changes do not affect the kernel of the language and must not be integrated into the language itself.

Evaluation of the Language Initial experiments and interviews with domain experts have shown that Athos code is easily comprehensible for a traffic expert even without training. Writing models is obviously more difficult and requires at least a minimal amount of training. Future work will evaluate the language in field experiments and results will be reflected back to the language and its design in order to further improve its usability. Besides Challenger's multi agent systems specific SEA_ML framework [CKT15] for evaluating DSLs we plan to also blend the Cognitive Dimension of Notations framework (see [BG03]) and additional approaches (e.g., [RdBZ17]) into our evaluation methodology.

References

[BBV08] Baldacci, Roberto; Battarra, Maria; Vigo, Daniele: Routing a Heterogeneous Fleet of Vehicles. In (Golden, Bruce; Raghavan, S.; Wasil, Edward, eds): The Vehicle Routing Problem: Latest Advances and New Challenges, volume 43 of Operations Research/Computer Science Interfaces, pp. 3–27. Springer US, Boston, MA, 2008.

[BG03] Blackwell, Alan; Green, Thomas: Notational systems–the cognitive dimensions of notations framework. HCI Models, Theories, and Frameworks: Toward an Interdisciplinary Science. Morgan Kaufmann, 2003.

[BTV10] Baldacci, Roberto; Toth, Paolo; Vigo, Daniele: Exact algorithms for routing problems under vehicle capacity constraints. Annals of Operations Research, 175(1):213–245, 2010.

[CKT15] Challenger, Moharram; Kardas, Geylani; Tekinerdogan, Bedir: A systematic approach to evaluating domain-specific modeling language environments for multi-agent systems. Software Quality Journal, pp. 1–41, 2015.

[DC12] Dalal, Sandeep; Chhillar, Rajender Singh: Case Studies of Most Common and Severe Types of Software System Failure. International Journal of Advanced Research in Computer Science and Software Engineering, 2(8):341–347, 2012.

[DFJ54] Dantzig, George; Fulkerson, Ray; Johnson, Selmer: Solution of a large-scale traveling-salesman problem. Journal of the operations research society of America, 2(4):393–410, 1954.

[do12] do Nascimento, Leandro Marques; Viana, Daniel Leite; Am Neto, Paulo Silveira; Martins, Dhiego A. O.; Garcia, Vinicius Cardoso; Meira, Silvio R. L.: A systematic mapping study on domain-specific languages. In: Proceedings of the 7th International Conference on Software Engineering Advances (ICSEA'12). pp. 179–187, 2012.

[Gr13] Grignard, Arnaud; Taillandier, Patrick; Gaudou, Benoit; Vo, Duc An; Huynh, Nghi Quang; Drogoul, Alexis: GAMA 1.6: Advancing the art of complex agent-based modeling and simulation. In: International Conference on Principles and Practice of Multi-Agent Systems. pp. 117–131, 2013.

[HNA16] Horni, A.; Nagel, K.; Axhausen, K.W.: The Multi-Agent Transport Simulation MATSim. London: Ubiquity Press., 2016. DOI: http://dx.doi.org/10.5334/baw.

[Ho18a] Hoffmann, Benjamin; Chalmers, Kevin; Urquhart, Neil; Farrenkopf, Thomas; Guckert, Michael: Towards Reducing Complexity of Multi-agent Simulations by Applying Model-Driven Techniques. In: Advances in Practical Applications of Agents, Multi-Agent Systems, and Complexity: The PAAMS Collection - 16th International Conference, PAAMS 2018, Toledo, Spain, June 20-22, 2018, Proceedings. pp. 187–199, 2018.

[Ho18b] Hoffmann, Benjamin; Guckert, Michael; Farrenkopf, Thomas; Chalmers, Kevin; Urquhart, Neil: A Domain-Specific Language For Routing Problems. In: ECMS 2018 Proceedings edited by Nolle L. et.al. pp. 262–268, 05252018.

[Ho19a] Hoffmann, Benjamin; Chalmers, Kevin; Urquhart, Neil; Guckert, Michael: Athos - A Model Driven Approach to Describe and Solve Optimisation Problems: An Application to the Vehicle Routing Problem with Time Windows. In: Proceedings of the 4th ACM International Workshop on Real World Domain Specific Languages. RWDSL '19, ACM, New York, NY, USA, pp. 3:1–3:10, 2019.

[Ho19b] Hoffmann, Benjamin; Guckert, Michael; Chalmers, Kevin; Urquhart, Neil: Simulating Dynamic Vehicle Routing Problems With Athos. In: ECMS 2019 Proceedings edited by Mauro Iacono, Francesco Palmieri, Marco Gribaudo, Massimo Ficco. ECMS, pp. 296–302, 06142019.

[MN12] Maciejewski, Michał; Nagel, Kai: Towards Multi-Agent Simulation of the Dynamic Vehicle Routing Problem in MATSim. In (Wyrzykowski, Roman; Dongarra, Jack; Karczewski, Konrad; Waśniewski, Jerzy, eds): Parallel Processing and Applied Mathematics. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 551–560, 2012.

[Om06] Ombuki, Beatrice M.: Multi-Objective Genetic Algorithms for Vehicle Routing Problem with Time Windows. Applied Intelligence, 24(1):17–30, 2006.

[RdBZ17] Rodrigues, Ildevana Poltronieri; de Borba Campos, Márcia; Zorzo, Avelino F.: Usability Evaluation of Domain-Specific Languages: a Systematic Literature Review. In: International Conference on Human-Computer Interaction. pp. 522–534, 2017.

[St11] Steil, Dana A.; Pate, Jeremy R.; Kraft, Nicholas A.; Smith, Randy K.; Dixon, Brandon; Ding, Li; Parrish, Allen: Patrol Routing Expression, Execution, Evaluation, and Engagement. IEEE Transactions on Intelligent Transportation Systems, 12(1):58–72, 2011.

[TW04] Tisue, Seth; Wilensky, Uri: NetLogo: Design and implementation of a multi-agent modeling environment. In: Proceedings of agent. volume 2004, pp. 7–9, 2004.

[va00] van Deursen, Arie; Klint, Paul; Visser, Joost et al.: Domain-specific languages: An annotated bibliography. Sigplan Notices, 35(6):26–36, 2000.

6 Reviewer's remarks

6.1 Section 0 – The Abstract

- [Ben]: ~~Abstract should mention that this paper focusses on the extension mechanism of the language (first sentence of section 5 could be used for that). (review 2)~~ Changed last sentence of abstract.
- [Ben]: ~~Abstract consists mainly of a justification and short description of the approach, while the actual focus of the paper, i.e. the extensibility of the language, is not pointed out.~~ Changed last sentence of abstract.

6.2 Section 1 – Introduction

- [Ben]: ~~improve explanation on necessity for extension algorithms with regard to the usefulness of algorithm designers and domain experts working in the same domain (ben: one is the experts in implementing correct and efficient algorithms, while the other knows real-world scenarios to proficiently apply these algorithms) (review 1)~~ Added an explanation to the third paragraph on page 2.
- [Ben]: ~~issues like increased traffic and congestion are stated to necessitate a language like athos, however it is not discussed how Athos helps to remedy/alleviate these problems -> reference to one of our other papers required (perhaps the first ECMS paper) (review 3)~~ Added 'last mile deliverie' in the introduction as something that calls for new ideas. It is already explained, how Athos can help to alleviate the problem. The fourth paragraph explains, how routing problems are solved and tested against dynamic traffic patterns.

6.3 Section 2 – Athos

- [Ben]: ~~Expand the acronym for the title of section 2.1 (review 2)~~ Done!
- [Ben]: ~~subsection 2.2 does not seem to fit overall section title (review 1)~~ Added some sentences at the beginning of section 2 that explain the structure of the subsections.
- [Ben]: ~~final paragraphs in section 2.2 have little connection to the first paragraphs of this section (review 1)~~ Added some sentences at the beginning of section 2 that explain the structure of the subsections.
- [Ben]: a list of requirements could have been presented earlier in the motivation (review 1)
- [Ben]: ~~Discussion about the meaning of generalisation is claimed to be misleading (there is a quote provided: Ä is a generalization of B if and only if every instance of concept B is also an instance of concept A; and there are instances of concept A which are not instances of concept B. [Wikipedia]"-> the reviewer demands~~

~~that the generalizes arrows in figure 1 be changed to «specializes» or «refines». (review 4)~~ Discussion is clear and comprehensible. Reviewer was right in that the labelling of the arrow had to be changed, which was done.

- ●[Ben]: ~~Hierarchy in Figure 1 is claimed to reduce to a hierarchy of class interfaces [honestly, I don't see the problem in that].~~ Taxonomy (Figure 1) was revised.
- ●[Ben]: ~~It is criticised, that NetLogo is not introduced. (review 4)~~Added URL of NetLogo project to the position where NetLogo is mentioned for the first time.
- ●[Ben]: It is argued that it is not evident that the extension mechanism can be applied to arbitrary multi-agent platforms. (review 4)
- ●[Ben]: ~~We are asked to explain why NetLogo does not provide the means to solve the described problem. [We should include something along the lines of "NetLogo is a DSL for the domain of multi-agent simulations, i.e. it is domain specific with regard to the technical domain, however it is not tailored toward a specific application domain.]~~ In section 2.1. - 3rd paragraph on pages 4/5, an explanation why Athos models are even more concise was added.
- ●[Ben]: ~~Listing above should be changed to Listing 1 (number instead of position) (review 4)~~ Done!
- ●[Ben]: ~~page 8, l.5, below: _startcity must be changed to startcity~~ Done!
- ●[Ben]: a concrete description of how the simulation of a model like listing 1 looks like in the end is deemed desirable.
- ●[Ben]: ~~section 2.1 mentions the generation of a graph, however it is not clear where and why the graph is generated and what the actual purpose of the graph is. (review 3)~~ Added an additional sentence to the beginning of section 2.1 that explains the purpose of the graph.
- ●[Ben]: ~~the purpose of section 2.1 is not made clear (review 3)~~Added some sentences at the beginning of section 2 that explain the structure of the subsections.
- ●[Ben]: ~~the value [which actually is to show the different parameters/restriciton of the different problems] of section 2.2 is not conveyed clearly / is doubted (review 3)~~Added some sentences at the beginning of section 2 that explain the structure of the subsections.
- ●[Ben]: the example algorithm uses libraries of NetLogo -> are the algorithm implementations always specific to a specific simulation platform [currently: indeed - they are! perhaps this is someting that could be added to the future work section?] (review 3)

6.4 Section 3 – Case study

- ●[Ben]: Mention that there is currently no data on user acceptance but that we are working on that.
- ●[Ben]: First show the implementation of the algorithm, then show how it is used in the model. some snippets from the java file would be nice as a complement to the model [I guess we won't be able to fit that into 10 pages](review 3)

- ●[Ben]: ~~Case study is not motivated as being realistic or useful. (review 1)~~ Changed name of section from 'case study' to 'exemplary use case'. Added to the beginning of the section to better convey the purpose of the exemplary use case.
- ●[Ben]: ~~The implementation of the case is described but there is no ßtudy"(make the eample ore specific, give a better motivation, add a short evaluation) (review 1)~~ Renamed section and changed the introduction of the section to motivate it better.
- ●[Ben]: UML-diagram (Fig. 5) should be moved to previous section, since it describes genral facts (main capabilities of the base class and the `reportPrimitive()` method. (review 1)

6.5 Section 4 – Related work

- ●[Ben]: more works in the area of dsls and vrps to get a broader overview (review 1)
- ●[Ben]: topic of incorporation of external algorithms into a dsl is missing completely (review 1)
- ●[Ben]: DVRP in related work is not explained

6.6 Section 5 – Conclusion and Future Work

- ●[Ben]: a more clear transition from conclusion to future work is required (review 1)

6.7 Genral problems

- ●[Ben]: A section or a paragraph on common problems in traffic simulation before section 2 is asked for (review 2)
- ●[Ben]: ~~There are terms for "language developer"(Section 5) and äpplication programmer"(section 1) who both have the intent to extent Athos -> harmonize these terms (and use the term "language developer")~~ The term 'application developer' is no longer used in the text.
- ●[Ben]: ~~An outline of the paper structure is missing (review 3)~~Changed the last paragraph of the introduction.
- ●[Ben]: ~~The term network based traffic simulation is never explained nor used (review 1)~~ Removed the term "network based traffic simulation".
- ●[Ben]: ~~Fig 1, 2, 3 and 5 have the same caption. (review 1)~~ Took care of that.
- ●[Ben]: ~~The introduction as well as the conclusion state that two roles are identified in subsection 2.3. However, in this subsection, there are no roles specified, nor is the word 'role' even used. (review 1)~~ The word 'role' still isn't used in section 2.3, however, it should be clear now, that the first paragraph of said section deals with the different roles that develop/extend/use Athos.

- ●[Ben]: ~~On page 9, it is stated that the terms "vehicle"and ägent"are used interchangeably. However, both terms are used multiple times before. (review 1)~~ This is now mentioned at the begining of Section 2.1 on page 3.
- ●[Ben]: ~~H019a] and [Ho19b] are actually the same paper.~~ Fixed.
- ●[Ben]: "The language is called extensible. This property only holds, if changes in Xtext are restricted by rules to "conservativeönes"[I quoted this remark directly, since I do not get what is meant here] (review 4)
- ●[Ben]: ~~For references, we use the complete words (e.g. Listing, Figure) while the referenced labels are abbreviated (e.g. List., Fig.)~~ Since the abbreviated Form is used by the template, the references in the text are now also abbreviated.
- ●[Ben]: ~~References to the mentioned platforms are missing [I'm sure, we provided these references via footnotes]~~ References were added.