A Model-Driven Solution to support Smart Mobility Planning

Antonio Bucchiarone
Fondazione Bruno Kessler
Trento, Italy
bucchiarone@fbk.eu

Antonio Cicchetti
Mälardalen University
Västerås, Sweden
antonio.cicchetti@mdh.se

ABSTRACT

Multimodal journey planners have been introduced with the goal to provide travellers with itineraries involving two or more means of transportation to go from one location to another within a city. Most of them take into account user preferences, their habits and are able to notify travellers with real time traffic information, delays, schedules update, etc.. To make urban mobility more sustainable, the journey planners of the future must include: (1) techniques to generate journey alternatives that take into account not only user preferences and needs but also specific city challenges and local mobility operators resources; (2) agile development approaches to make the update of the models and information used by the journey planners a self-adaptive task; (3) techniques for the continuous journeys monitoring able to understand when a current journey is no longer valid and to propose alternatives. In this paper we present the experiences matured during the development of a complete solution for mobility planning based on model-driven engineering techniques. Mobility challenges, resources and remarks are modelled by corresponding languages, which in turn support the automated derivation of a smart journey planner. By means of the introduced automation, it has been possible to reduce the complexity of encoding journey planning policies and to make journey planners more flexible and responsive with respect to adaptation needs.

CCS CONCEPTS

• **Software and its engineering** ▪ **Software design engineering**; **Software evolution**; • **Applied computing**;

KEYWORDS

Model-Driven Engineering, Model Transformation, Smart Mobility, Journey Planning

ACM Reference Format:
Antonio Bucchiarone and Antonio Cicchetti. 2018. A Model-Driven Solution to support Smart Mobility Planning. In *Proceedings of ACM/IEEE 21th International Conference on Model Driven Engineering Languages and Systems (MODELS 2018)*. ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Organizing and managing the mobility services within a city, meeting travelers expectations and properly exploiting the available transport resources, is becoming a more and more complex task [6]. The inadequacy of traditional transportation models is proven by the proliferation of alternative, social and grassroots initiatives aiming at a more flexible, customized and collective way of organizing transport (e.g., car pooling, ride and park sharing services) [1, 7]. Some of these attempts have been very successful (e.g., Uber[1]), even if in most cases these are isolated solutions targeting specific mobility customer groups and are not part of the city mobility eco-system, mainly based on traditional public and private transport facilities. An attempt of re-thinking the way mobility is managed and offered is represented by the Mobility as a Service (MaaS) model [11, 14].

MaaS techniques (e.g., MaaS Global[2]) aim at arranging the most suitable transport solution for their customers thanks to cost effective integrated offer of different multi-modal means of transportation. MaaS also foresees radical changes in the business landscape, with a new generation of mobility operators emerging as key actors to manage the increased flexibility and dynamism offered by this new concept of mobility.

Although there are many journey planners (JP)s already running in the market, there is none yet that allows users to be supported during the whole duration of the journey, from the initial request to the end of the journey itself. Planning for a trip is still inefficient as people typically receive alerts and notifications and have to perform several newer attempts to solve different issues (i.e., delays, suppressed trips, weather conditions, etc.). At the same time, the mobility operators and also the local government/municipalities are not really an active part of the city's mobility. They can provide their own updated information or promote their own challenges to make the city mobility more sustainable,

[1]https://www.uber.com
[2]http://maas.global

but they cannot be technically integrated into the city JPs yet. These make the actual JPs not prepared to consider more flexible mobility resources and sustainable mobility policies [10].

Model-Driven Engineering (MDE) allows coping with the complexity of reality by abstracting the relevant aspects for a certain application into corresponding models [13]. In this respect, this work discusses the provision of an MDE based solution for smarter JPs. In particular, domain-specific languages (DSLs) allow the definition of travel parameters from the different stakeholders' points-of-view, while model transformations are exploited to generate the code of the planner application as well as to manage runtime contextual information to possibly adapt travel routes. The definition of the DSL is made challenging due to the nature of the domain itself, which can be reduced to modelling a constraint-solving problem where the modellers have to both specify the variables of each scenario and how they should be combined. As a consequence, the accidental complexity [3] introduced by the DSL tends to grow and in the worst cases could be comparable to writing source code. On the contrary, the proposed solution keeps policy specification simple, and by exploiting the links across the views and their embedded semantics is capable of supporting the required level of planning flexibility and adaptation. All the proposed mechanisms have been concretized in a prototypical implementation that played the role of a feasibility study and workbench before the knowledge transfer into real JPs.

The remainder of the paper is organized as follows. Section 2 describes the motivating scenario, the challenges it poses and a summary of our contribution. Section 3 illustrates the *Code-Driven Mobility Configuration* part of our approach, while Section 4 its *Model-Driven Mobility Configuration* counterpart. The prototype implementation of a Smart Journey Planner (SmartJP) is introduced in Section 5. In Section 6 lessons learned and future investigations are discussed. Section 7 surveys related works and Section 8 concludes the paper.

2 MOTIVATING SCENARIO, CHALLENGES AND CONTRIBUTION

In the Smart Mobility domain, a SmartJP is expected to go beyond standard journey planning, notably by managing in a collective way all the interests/objectives of the actors that are involved in a journey (during the planning and execution phases). The goal of the SmartJP is to provide optimal itineraries taking into account the single interests of the different involved stakeholders (i.e., Government/-Municipality, public/private Mobility Operators, Individual Citizens) and all the contextual information (e.g., weather conditions, urban environment conditions, available mobility resources, etc..) that make the itineraries proposed more contextualised and reliable. In particular, a SmartJP should be:

- **Intermodal:** travellers must be provided with optimal itineraries, possibly involving two or more means of transportation.
- **Personalized:** personalisation means to make personal mobility recommendations out of routing proposals. Routing proposals represent the output of a classical intermodal journey planner (i.e., OTP[3]). Mobility recommendations can, for example, be generated by the filtering and reordering of routing proposals. The user shall be able to specify its preferences and the journey planner will provide specific itineraries taking into account these preferences and the user profile information.
- **Proactive:** all the needed (contextual) information regarding the selected itineraries of the users must be monitored. Information such as real-time data (traffic, accidents, delays, weather conditions, blocked roads, no parking spaces available, etc..) are matched with the selected itineraries. If a problem is detected, a re-planning can be initiated with the objective of proposing alternative itineraries to the involved users.
- **Sustainable:** the itineraries proposed by the journey planner must take into account not only the preferences and profile of the single user (i.e., personalisation), but also the sustainable mobility challenges of the city administration. These challenges are used to filter and sort the resulting itineraries respect to their environmental impact (i.e., carbon emissions estimate) or, health factors (e.g., steps taken, calories consumed).
- **Realtime:** the journey planner must use real-time information on the available resources to provide reliable and safe itineraries.
- **Portable:** the journey planner must be easily configured and instantiated in different cities exploiting the various available mobility services and the specific policies defined by the different involved stakeholders.

The remainder of the section first presents a scenario that serves as running example for this work, and then explains the reasons why the presented scenario is challenging for the current planners, and hence motivate better our contribution.

2.1 Smart Journey Planner

The following illustrates a simple but complete scenario coming from the daily life of common users that need to move around in a smart city. We will specifically refer to the city of Trento, Italy, since we already have realized a concrete journey planning app to travel on this territory [4], and we can access it for validation purposes.

[3]http://www.opentripplanner.org/
[4]http://www.smartcommunitylab.it/apps/viaggia-trento/

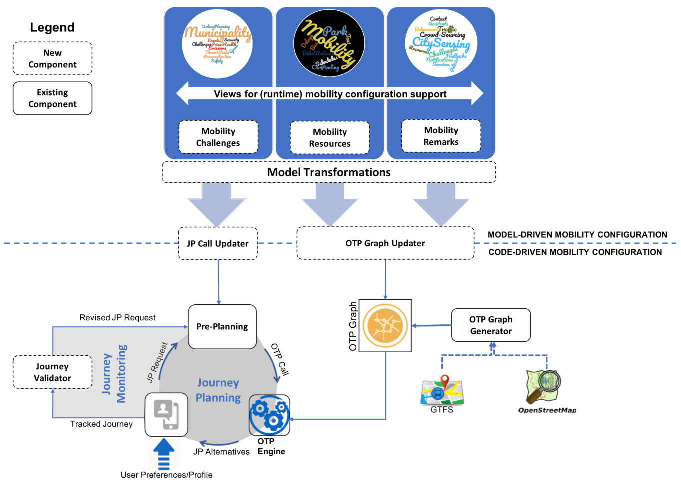

Figure 1: Smart Mobility Framework.

- A big event, with thousands of people, is organised in Trento. Most of the participants arrive by car from different locations and use the SmartJP application to plan in advance how to reach the city center.
- For the whole duration of the event, the local municipality wants to promote the usage of two main mobility services: *Park&Ride* and *Bike Sharing*. Through the former participants leave the car at specific parking spaces and use the organised shuttle buses to reach the city center. By means of the latter, participants simply borrow a bike from a dock and return it at another dock in the city. To promote greater use of bicycles rather than shuttle buses (i.e., sustainable mobility), the municipality defines a game where users who do more Kms in a sustainable way (e.g., by bike, by walk) during the event will receive discounts for purchases in the city's shops.
- During the entire duration of the event, a municipality ordinance prohibits public and private buses to go through some streets of the city. For this reason, public and private mobility operators decide to re-schedule their buses paths accordingly, disabling some bus stops.
- The night before the event begins, a heavy rain drops a set of trees in some streets of the city, making some of the routes for the organized shuttle buses not suitable anymore. At the same time, some bicycle paths become unusable due to the amount of water on the road.
- For this unexpected event, the municipality decides to organize different stacking points for the park&ride service and at the same time to revise the game challenges. In particular it decides to reward more walked distance than Kms ridden on bikes. Moreover, some public bus service providers decide to reactivate some bus stops that were cancelled in order to allow the visitors to reach the city center directly by public buses.

2.2 Challenges

By considering the desired features listed at the beginning of this section and the scenario illustrated so far, in the general case it is possible to assess that intermodal, personalized, and sustainable requirements are achieved for a *static* use of planners, i.e. that does not include the proactive and realtime features. Indeed, in most of the cases planning challenges are hard-coded in text-based configurations and/or code chunks used to run the planner. In this respect, any of the last minute changes mentioned in the scenario due to the heavy rainfall would require a manual reconfiguration/recoding of the planner. Since in most of the cases such a rework is unfeasible, users can be only notified about the issues affecting their travel plans, without any appropriate replanning support. Due to the same flexibility issues, the planners are not meant to proactively react to changing traffic conditions. At this point it is worth noting that while some planners provide recalculation facilities for specific means of transportation (notably cars), they are usually neither intermodal, nor knowledgeable of city specific travel challenges.

2.3 Contribution

The goal of this paper is to show how MDE is a strategic piece of a framework to realise advanced journey planning solutions taking into account different aspects and stakeholders involved in the smart mobility domain. In this respect, our contribution is visualized in Figure 1, which depicts an overview of the proposed solution and highlights what has been introduced, and what has been re-used from the existing journey planning techniques. In particular, components shaped with dashed line borders are contributions of our work, while solid line borders identify existing components, possibly slightly adapted to be integrated with the rest. Technically, the solution is composed by two fundamental layers: (1) a code-driven mobility configuration layer and (b) a model-driven mobility configuration layer. Each of the layers are discussed in details in the following sections.

3 CODE-DRIVEN MOBILITY CONFIGURATION

A journey planner has the goal to provide travellers itineraries involving two or more means of transportation to go from one location to another within a city. An *advanced journey planning* not only is devoted to answer with journey alternatives (i.e., *Journey Planning* cycle of Figure 1) but it concerns the entire planning process, ranging from the specification of travel preferences to the arrival at the final destination. Additionally it includes a *Journey Monitoring* cycle (see Figure 1) to continuously check the validity of a *tracked journey* and proactively re-plan in case of contextual changes and failure conditions (i.e., traffic, weather conditions, accidents, etc..). The *Journey Planning* phase starts when the user specifies the *JP Request*. It is done specifying a set of parameters like: (i) the origin and the destination of the trip, (ii) the desired departure time, (iii) the desired departure date, (iv) the transport modes to consider (e.g., walk, car, bus, train, etc..), and (v) the maximum distance in meters that the traveller is willing to walk.

The *JP Request* is elaborated in the *pre-planning* phase where extra parameters are added to derive the specific *OTP Call*. These parameters are defined with the help of the *JP Call Updater* component, devoted to the analysis of the *Mobility Challenges* defined by the local Government/-Municipality and to the generation of specific parameters related to the sustainable mobility campaigns launched by the local government.

When a city wants to promote sustainable behaviours of their citizens, through the mobility, the *JP Call Updater* component extends the call to the *OTP Engine* with extra parameters that aim at such sustainability. This is the situation when the city may want to promote itineraries that suggest the use of a specific mobility service, or that encourage using a public transport, as opposed to another, to reach the same destination at certain critical times during the day. At the same time the city may want to avoid not-safe itineraries and itineraries that consider blocked-roads or where manifestations are in place.

With this pre-planning phase, the *OTP Call* includes *personal preferences*: common among those are the desired means of transportation, prioritisation criteria such as shortest, fastest or cheapest routes, or special information for users who are bicyclists, and *Mobility Challenges* devoted to the enforcement of mobility policies to be taken in to consideration during planning for e.g. green policy, safety factors, etc.. The *OTP Call* generated in the pre-planning step becomes an input of the *OTP Engine* where the OpenTripPlanner (OTP) module is running. OTP is an open source platform for multi-modal and multi-agency journey planning. OTP relies on open data standards including *GTFS* [5] for transit and *OpenStreetMap* [6] for street networks. The multimodal journey planning facility provided by *OTP Engine* operates on an object called *OTP Graph*, which specifies all of the locations in a specific region, and how to get from one to another. The *OTP Graph Generator* has the objective to configure and build the trip planner graph and any other artifacts, included in the *OTP Graph*, necessary for the journey planning.

While in the majority of the released journey planners, the graph used for generate journey alternatives is deployed and modified in a occasional manner (at each new application release), the goal of our framework is to make it adaptable. This is done thanks to the *OTP Graph Updater*. It is composed by a set of routines able to receive contextual changes and update the current version of the *OTP Graph*.

[5]http://gtfs.org/

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

One source of contextual changes is represented by the *Mobility Remarks*: a set of declared contextual changes devoted to represent changes in the mobility resource of a city. As we will describe in Section 4.1, any public/private mobility operators can decide at any time to change the status of a specific mobility resource (i.e., route changes, closed parking spaces, bus Time table updates, strikes, etc..). At the same time, the *OTP Graph* can be updated reflecting changes in the real context of the city. This is done exploiting data generated by *sensors* installed around the city and that are part of a *City Sensing* infrastructure capable to collect and compute heterogeneous and distributed data.

4 MODEL-DRIVEN MOBILITY CONFIGURATION

4.1 Abstract syntax for the Mobility View languages

The SmartJP solution is made-up of three different mobility views, namely the *Mobility Challenges* view, the *Mobility Resources* view, and the *Mobility Remarks* view. The first is used to describe the journey planning preferences coming from the different mobility stakeholders, notably citizens, city administrations, and so forth; the second collects the mobility resources available for a certain city; the third represents mobility remarks that affect certain resources. Each view is realized by means of a corresponding metamodel, each of which is illustrated in the remaining of this section[7].

4.1.1 The Mobility Challenges View. In order to grasp challenges modeling approach, it has to be considered that when citizens look for journey alternatives they select the possible types of transportation (e.g. bus and walk), route characteristics (e.g. fastest), and set some travel preferences (e.g. the maximum distance they are willing to walk or the maximum number of changes). In a similar way, city administrations set some basic transportation rules, notably when transportation types are (in)/compatible and hence the selection of one (excludes)/includes the other, when transportation types entail specific route characteristics (e.g. bicycle or walk shall entail safest route alternatives), and default travel preferences related to transportation and route characteristics (e.g. travelling by bus should not require walking more than 500 meters, while the fastest alternative should not include more than 2 changes).

Figure 2 shows an excerpt of the metamodel defining the abstract syntax of the Mobility Challenges View. In particular, a `TravelPlanner` comprises of `Policy`, `TransportType`, `RouteType` and `SmartPlannerParameters`. Policies are distinguished in `PreCallPolicy` and `PostCallPolicy`, to distinguish

[7]For the sake of space, we necessarily show excerpts of the view metamodels. The interested reader is referred to the project GitHub for the complete version of the artifacts.

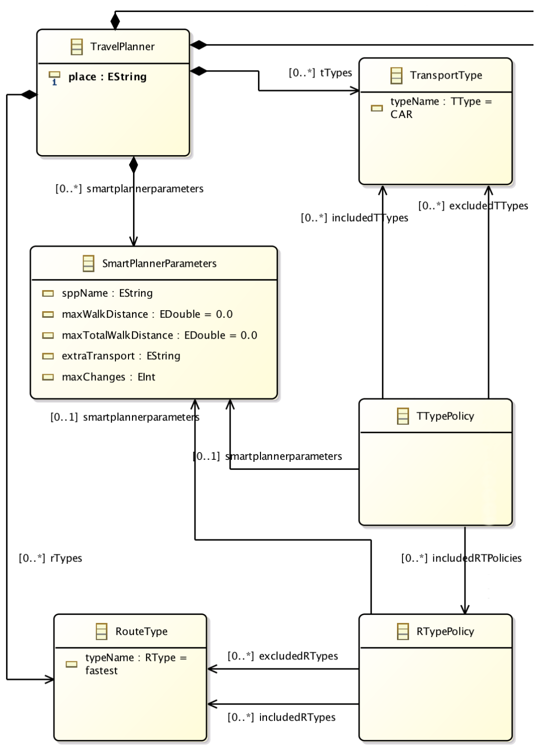

Figure 2: An excerpt of the Mobility Challenges View metamodel.

whether planning rules are used as pre-condition to compute an alternative or if as a refinement of the available solutions, respectively. In turn, challenges are specialized further into `TTypePolicy` and `RTypePolicy`. Consequently, challenges are defined by means of associations defining (in-) compatibility relationships. Notably, two `TransportTypes` are compatible they are linked by the `includedTTypes` association, while they are mutually exclusive if they are related through `excludedTTypes`. Moreover `smartplannerparameters` associations link travel preferences with certain transportation type policies.

Analogously, it possible to define (in-) compatibilities between route types and corresponding planner parameters.

4.1.2 The Mobility Resources View. Every city owns a constellation of mobility services, like busses, trains, bike/-car sharing, parkings, and so on. This view collects those services by providing corresponding concepts and properties. Figure 3 shows an excerpt of the metamodel that defines the abstract syntax of the Mobility Resources View. In particular, the `MobilitySupport` is composed by a set of `MobilityResources`, which in turn are specialized as `Parking`, `BikeSharing`, transit `Stop`, and so forth. The knowledgeable reader will recognize the concepts coming from the Google Transit Feed Specification (GTFS)[8], a de-facto standard

[8]https://developers.google.com/transit/gtfs/reference/

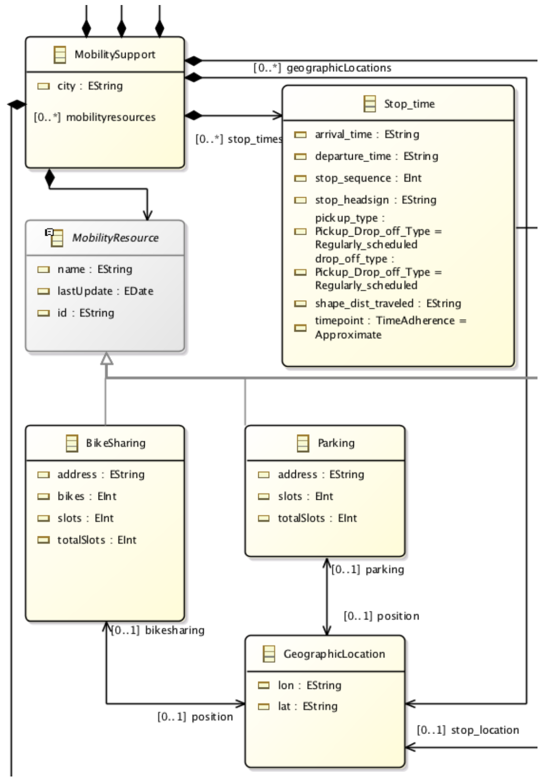

Figure 3: An excerpt of the Mobility Resources View metamodel.

used to store transportation information to be used for travel planning. Indeed, `Stop` and `Stop_time` are two of the available concepts in GTFS to define stops distributed over a certain geographical area together with serving times. On the contrary, GTFS does not provide support for bike sharing and parking, hence the corresponding bike sharing and parking concepts have been abstracted from OTP. In total, the Mobility Support view contains 17 metaclasses and 10 domain-specific datatypes.

4.1.3 The Mobility Remarks View. As discussed at the beginning of this work, one of the goals of SmartJP is to be adaptable to the inherent floating conditions of traffic and hence the corresponding adequacy of travel alternatives. In order to support such a feature, the solution presented in this work provides an appropriate view for city sensing inputs. An excerpt of the mobility remarks view metamodel, called `MapNotes`[9], is depicted in Figure 4. The map notes are a collection of different kinds of timestamped `StatusRemarks`, notably `BikeSharingRemark`, `ParkingRemark`, `TrafficRemark` and `TransitRemark`. The first two report the availability status of bike sharing and parking resources, respectively; the third is used to note traffic conditions for selected route segments (identified through map nodes and geographic

[9]The naming is due to the intended long-term exploitation of this metamodel, i.e. storing sensing information representable on a corresponding city map.

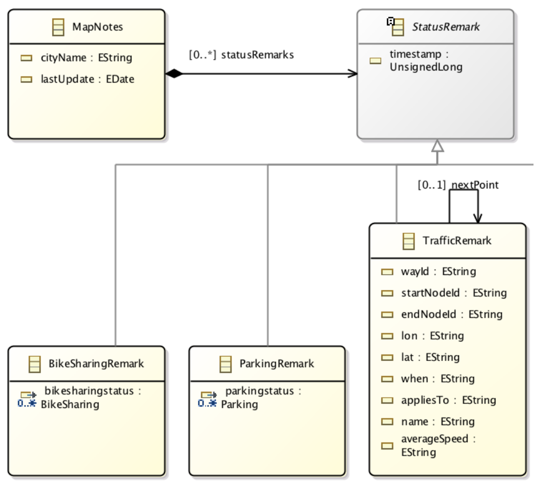

Figure 4: An excerpt of the Mobility Remarks View metamodel.

coordinates), e.g. by updating the current average speed (see `averageSpeed` property in the metaclass `TrafficRemark`); the fourth is, analogously to the mobility resources, aligned with the GTFS realtime format devoted to storing updates for transits[10].

It is worth noting that there exist several links across the mobility views: visible in Figure 4, the `bikesharingstatus` property refers to `BikeSharing` elements in the mobility resource view (see Figure 3) as well as `parkingstatus` to `Parking`. In the same way, there exist interconnections for other data as trips, routes, stops, and so forth. These links keep the consistency between the involved views by construction, since it will not be possible to update the status of mobility resources (in the city sensing view) that do not exist (in the mobility resource view). On the contrary, in the current text-based approaches this constraint is only a requirement expressed in natural language in the format specification, prone to errors like providing erroneous identifiers for resource status updates.

4.2 Semantics for the Mobility View languages

As mentioned so far, the three different views proposed in this paper can be used to describe different concerns related to mobility and its planning. In the current state of practice, travel planning is performed at the text level of abstraction. More precisely, pre-/post- planning policies are hard-coded in the configuration and calls for the planning algorithms, while planning monitoring and adaptations are performed by exploiting well defined text formats. In this respect, the proposed solution supports both the continuous deployment of planning strategies, based on mobility stakeholders' requirements, and the monitoring and adaptation of journey plans. These goals are achieved through translational semantics, that is by mapping the information carried by each view into corresponding artefacts in the mobility domain (configuration files), where the semantics is known.

For all the views, the Acceleo transformation language[11] has been used to provide corresponding semantics. Acceleo is a model-to-text transformation language that adheres to the OMG standard for model-to-text transformation [12]. The following sections provide further details about providing semantics to each of the mobility views.

4.2.1 Semantics for the Mobility Challenges View. Section 4.1.1 introduced the view that enables the specification of mobility needs by different stakeholders, like citizens, city administrations, and so forth. In general the challenges affect mobility planning *a priori*, by setting the boundaries within which the planner will compute the alternatives. As a consequence, challenge information is typically translated into corresponding code and/or configurations for travel planners, notably what transportation alternatives to consider and how to rank the computed solutions. Figure 5 shows an excerpt of the Acceleo transformation that provides semantics to the Mobility Challenges View. Going into more details, the picture shows the management of compatible transportation types: a travel planning request comes from the user as including several transportation types (e.g. bus and walk), or the city administration has established that whenever the bus is selected as transportation, then also tram should be included in the planner as transportation alternative. These cases are modelled as `TransportationType` entities linked through the `includedTypes` relationship in the policy model (see Figure 2). Correspondingly, the Acceleo transformation creates a call to the planner that composes of all the modeled compatible transportation types: the template `generateTTypeConstraints` receives as input a `TTYpePolicy` entity, that is checked for the existence of `includedTTypes` relations. If existing, the transformation iterates over them and generates the corresponding code to check whether the related transportation types are part of the current planner request configuration, and if not to add them (see the generated if statement `allTypes.contains(TType.[itt.typeName/])`).

4.2.2 Semantics for the Mobility Resources View. Mobility resources are typically listed by means of text-based formats. Notably, GTFS is made-up of a set of files adopting a well defined comma-delimited text. Moreover, OTP leverages JSON files to exchange relevant journey planning information. Therefore, our solution "serializes" the information stored in the Mobility Resources View towards appropriate textual representations to be consumed throughout the

[10]https://developers.google.com/transit/gtfs-realtime/reference/

[11]https://www.eclipse.org/acceleo/
[12]https://www.omg.org/spec/MOFM2T/About-MOFM2T/

```
[template public generateTTypeConstraints(aTTypePolicy: TTypePolicy)]
[if (aTTypePolicy.includedTTypes <> null)]
if ([for (itt : TransportType | aTTypePolicy.includedTTypes) separator(' ||')] type.equals(TType.[itt.typeName/]) [/for]) {
    [for (itt : TransportType | aTTypePolicy.includedTTypes)]
    if (!allTypes.contains(TType.[itt.typeName/])) {
        PlanningRequest npr = PlanningPolicyHelper.buildDefaultDerivedRequest(journeyRequest, pr, TType.[itt.typeName/], null, null, null, pr.isWheelChair(), true, prg);
        result.add(npr);
        allTypes.add(TType.[itt.typeName/]);
    }
    [/for]
}
[/if]
[/template]
```

Figure 5: An excerpt of the Acceleo transformation giving semantics to the Mobility Challenges View.

```
[template public generateElement(aMobilitySupport : MobilitySupport)]
[comment @main/]
[file (aMobilitySupport.city + 'BikeSharing.json', false)]
['['/][for (bsr : BikeSharing | aMobilitySupport.mobilityresources->filter(BikeSharing)) separator(',')][generateBikeSharingEntries(bsr)/][/for][']'/]
[/file]
[/template]

[template public generateBikeSharingEntries(aBikeSharingResource : BikeSharing)]
{"name":"[aBikeSharingResource.name/]","address":"[aBikeSharingResource.address/]","id":"[aBikeSharingResource.id/]","bikes":[aBikeSharingResource.bikes/],
"slots":[aBikeSharingResource.slots/],"totalSlots":[aBikeSharingResource.totalSlots/],"position":['['+generatePosition(aBikeSharingResource.position)+']'/]}
[/template]

[template public generatePosition(aPosition : GeographicLocation)]
[aPosition.lat+','+aPosition.lon/]
[/template]
```

Figure 6: An excerpt of the Acceleo transformation giving semantics to the Mobility Resources View.

journey planning and its management. Figure 6 depicts a portion of the Acceleo code to manage the semantics of the Mobility Resources View. In particular, the template `generateBikeSharingEntries` takes as input model elements of type `BikeSharing` (see the corresponding metamodel element in Figure 3) and writes out a corresponding JSON file in which the characteristics of the resource are appropriately encoded. By going into more details, the text lists the name and position of the resource, together with the total number of bike slots and the current number of bikes and slots available.

4.2.3 Semantics for the Mobility Remarks View. In a very similar way to what done for the Mobility Resources View semantics, the Mobility Remarks View contents are serialized towards appropriate textual formats that can be exploited from available journey planners. In particular, the remarks dealing with bike sharing, parking, and traffic conditions, will be encoded into JSON files that are used to update the graph on which the planner computes the travel alternatives. Moreover, the remarks devoted to the transits are translated towards the corresponding GFTS realtime feeds.

5 PROTOTYPE IMPLEMENTATION

To evaluate the approach proposed in this paper, we realized the activity diagram depicted in Figure 7. It represents the internal logic of the implemented Smart JP prototype, made by three components: (1) the SmartJP User Interface, (2) the Model-Driven Mobility Configurator, and the (3) Code-Driven Mobility Configurator. For the interested reader, the prototype is available in its entirety on a GitHub repository [13].

[13]https://github.com/das-fbk/SmartJP

The *SmartJP User Interface* is a map-based interface used by sending `HTTP GET` requests to the server running our prototype on a local machine (i.e., http://localhost:8080/). It presents a web client that interacts with the local SmartJP instance to generate journey alternatives answering the *Journey Planning Request* formulated by the user (i.e., *Citizen/Tourist*).

The *Model-driven Mobility Configurator* has been realized using the Eclipse Modeling Framework (EMF)[14]. EMF supports MDE techniques with a set of tools for creating modeling languages and their ecosystems of utility plug-ins. The three different languages, introduced in the previous sections, are defined as a model in the Ecore format (a meta-modeling language), and starting from the meta-model definition the tool automatically generates model editing facilities as plug-ins for Eclipse. Moreover, there exist many EMF-based MDE tools for defining custom editors, model transformations, and so on. Notably, the generators devoted to the code-driven mobility configurator are realized by means of Acceleo, that is a model-to-text transformation language available in EMF.

The *Code-Driven Mobility Configurator* is a Java-based component that provides multi-modal journey planning. It runs on any platform with a Java virtual machine and it is made by two main sub-components: (1) the *Journey Planner*, and (2) the *Journey Monitor*. The *Journey Planner* is built on top of the OTP project (i.e., *OTP Engine* in Figure 7), by enhancing its basic functionality with the following delta features:

[14]https://www.eclipse.org/modeling/emf/

- *Mobility Challenges Enforcement*: This feature aims at augmenting mobility recommendations not only by responding to the preferences and profile of the single user, but also by taking into account the sustainable mobility policies of the local city government. It allows the configuration and enforcement of *Mobility Challenges* to be taken into consideration during the planning. The *Mobility Challenges*, defined using the *Model-Driven Mobility Configurator*, and transformed as a set of *Journey Planning Parameters*, are injected in the *Pre-Planning Configuration* activity and can include factors such as the following: the city might promote itineraries that suggest the use of a specific mobility service, or that encourage using a public transport route, as opposed to another, etc.. The effect of being able to inject city-specific challenges in the process of journey planning is that this provides the ability to induce a high degree of engagement in the citizens using the SmartJP, with respect to citywide objectives.
- *Real-Time Mobility Information Exploitation*: This feature allows the SmartJP to provide journey alternatives taking into account updated city mobility information. These include the information coming from the *Update a Mobility Resource* and the *Analysis of Mobility Sensors* activities. Executing the first activity, each *Mobility Operator* can change the status of a *Mobility Resource* (i.e., bike-sharing station or shared bikes not available, bus stops canceled or entire bus line removed, etc.) and trigger the execution of the *OTP Graph Update* activity. This is done bu using *Model Transformation* techniques able to generate a *New OTP Graph* (in form of `JSON` files used by OTP during the *OTP Engine Call* activity). The SmartJP exploits these real-time data using a pull system configured by a `JSON` file that includes all the `Updaters` used by the *OTP Engine*. For each `Updater`, the SmartJP fetches a file from web servers every few seconds and updates its *OTP Graph* immediately.
  In addition to mobility resources updates, the SmartJP is able to use information coming from the *City Sensing* operation. Sensors, installed around the city, can be used to retrieve relevant information for the journey planning. As in the case of mobility resources, *JSON* files are stored and used by the *OTP Engine* to plan with real-time information.

The *Journey Monitoring* component has been realized to track specific journey alternatives. When a citizen selects her favorite journey alternative, it is continuously validated by this component and, if the *New OTP Graph* compromises its execution, the *Update JP Parameters* activity is triggered. The effect is to derive a revised version of the *Journey Planning Call Parameters* to be considered in the *Pre-Planning Configuration* activity. With this component each citizen is not simply notified with mobility information (i.e., delays, cancellations, accidents, etc..) but she receives an alternative journey plan valid for the new contextual situation.

6 LESSONS LEARNED AND FUTURE INVESTIGATIONS

This paper describes the efforts devoted to the evolution of current journey planners towards smarter solutions. The proposed approach relies on MDE: three different views allow for the separation of concerns, that together to a higher level of abstraction reduce the complexity of dealing with the mobility planning specification. The current solution can be thought as a continuous deployment and adaptation by means of MDE: in fact, whenever mobility challenges change new policies are modelled in the mobility challenges view and corresponding planning configurations are generated. Similarly, mobility related updates trigger recalculations of the existing travel plans. Since the models pertaining to the three mobility views are not exploited directly by the planner, it is not appropriate to consider this solution as models@runtime [15]. In this respect, while the exchange of mobility information is expected to remain text-based, it would be desirable a tighter integration between text formats and the views. Notably, a future development goal is the back-propagation of text-based information to the views, in order to complete the round-trip process from the views, to the planning configuration, implementation of the journey planning and its monitoring, and back to the views with the values coming from city sensing.

The relationships between the views, their corresponding semantics, and the configuration of the journey planner, constitute a megamodel [8]. Despite our efforts in looking for a suitable support for megamodeling, we did not find any ready to use solution for the EMF. The ATLAS Model Weaver (AMW)[15] has been abandoned in 2013, while other tools like EMFViews [4] appear to be not ready yet. More precisely, the accidental complexity introduced by those techniques is comparable to implementing your own solution for view management, making the effort not worth the case. Luckily, the mobility views considered in our approach can be considered as *orthogonal* [2], thus putting less pressures in terms of consistency preservation and conflict management. In fact, in all the cases the consistency can be managed by exploiting links between resources provided by EMF (specifically Ecore). However, if we would have liked to implement more advanced semantics, notably self-adaptive journey planning, we would have needed more complex interconnections requiring, e.g., model weaving mechanisms [5]. Self-adaptive journey planning is indeed another item in our future investigations agenda.

The definition of user-friendly DSLs has been not straightforward, especially when abstracting the text-based formats. We tended to realize a one to one mapping between text

[15]https://projects.eclipse.org/projects/modeling.gmt.amw

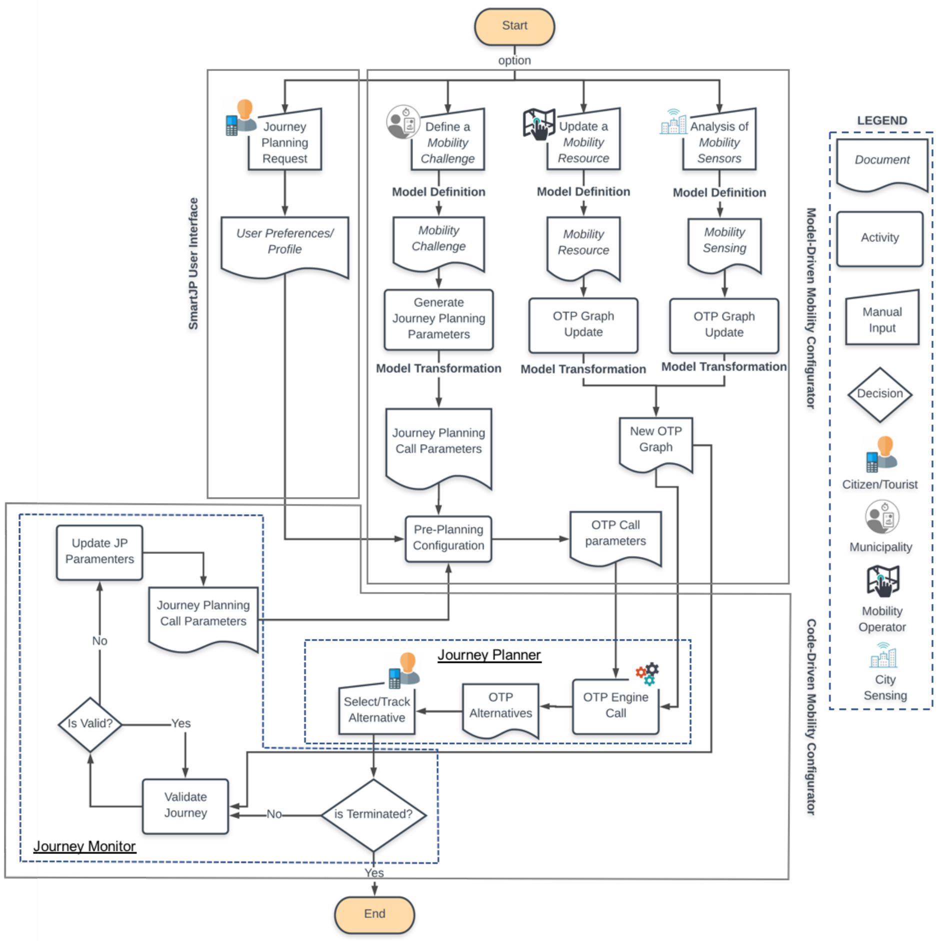

Figure 7: SmartJP Prototype Logic.

and modeling concepts, which however made modeling complex and in some cases even counterintuitive. The views proposed in our approach are the result of several months of studies and refinements of multiple language drafts. In this respect, after multiple failures we eventually decided to restart from scratch by conducting a preliminary study trying to elicit domain needs from the text artefacts, and then from those needs we proceeded with the definition of appropriate DSLs. As future improvements, we are leading ongoing developments for the integration of map-based interfaces with the presented views. In this way, the map of a city area and a palette for marking the map with notes would be the concrete syntax for our view metamodels.

7 RELATED WORKS

7.1 Multimodal Journey Planning

Multimodal journey planners can involve both public or private transportation services and can cover various areas ranging from local to global. For instance *Google Transit* [16] and *Rome2Rio*[17] are global planners, since they can be used for planning all around the world. *Viaggia Trento*[18], instead, is a local planner, since it combines all the current public transports specifically for the city of Trento in Italy. In this paper we are interested in journey planners able to consider different transport modes and to use *real-time information*. A Journey planner can use *real-time information* for public transport including arrival or departure times and any delays and line closures. At the same time, using this information, it can automatically re-route to alternative routes or inform users about forthcoming departures. Crowdsourcing is an example of techniques used to collect data from users that report specific events at their location (i.e., delays, accidents, etc..). These data are used by journey planners to understand the current situation and react accordingly sending related notifications. *Moovit*[19] makes

[16]https://maps.google.com/landing/transit/index.html
[17]https://www.rome2rio.com
[18]http://www.smartcommunitylab.it/apps/viaggia-trento
[19]https://moovit.com/

Name	Coverage	Modes	Configuration Support	Journey Monitoring
Citymapper	World	2	Mobility Resources	Traffic Info
FromAtoB	World	4	NO	NO
Google Transit	World	8+	Mobility Resources, Mobility Remarks	Alerts/Notification
HERE WeGo	World	4+	NO	NO
My TfGM	Manchester/UK	3	NO	NO
MyWay	World	5+	Mobility Resources, Mobility Remarks	Traffic and Services Status
Moovit	75 countries/700 cities	5+	Mobility Resources, Mobility Remarks	Alerts/Notification
RATP	Paris	7	Mobility Resources	Traffic Info
Rome2Rio	World	10+	NO	NO
Transport For London	London/UK	11+	Mobility Resources	Alerts/Notification
TripGo	World	4+	Mobility Resources	Alerts/Notification
Viaggia Trento	Trento/Italy	8	Mobility Resources	Alerts/Notifications
Voyager Route Planner	World	3	NO	NO

Figure 8: Overview of Multimodal Journey Planners

use of user reports (e.g., line didn't stop, platform change, etc..) to help other users in planning with real-time updated information. Other planners , as for example MyWay [20], Google Transit or Citymapper [21], use these information to send alerts or specific notifications to user that have their journeys tracked. Figure 8 shows a summary of a representative selection in the ecosystem of multimodal journey planners, evaluated respect to the following features: (1) Area Coverage, (2) Transportation Modes supported, (3) Configuration Support, and (4) Journey Monitoring.

Following this classification, we can make several observations about the actual multimodal journey planners. In particular, planners dealing with a few transportation modes, as well as planners having a local coverage for one or a few cities/countries are characterized by a high accuracy of the provided data. To the contrary, the more global are the planners, the more they tend renounce accuracy. This can be due to different aspects: for instance, while private companies own and manage their data, other open mobility services can rely on available open data and open API, which can be incomplete or not up to date. For what concerns the *configuration support*, most of the actual planners are based on code-driven support that means that all the city mobility policies are hard-coded while we want planners able to be configured at runtime when: (a) we have new *Mobility Resources* and *Mobility Remarks* requirements or when (b) the mobility managers of a city want to push some citizens' behaviors to have a more sustainable mobility through *Mobility Challenges*.

7.2 Model-driven support

To the best of our knowledge, there is no MDE approach addressing the support of smart journey planning in the sense addressed in this paper. There exist modelling approaches to deal with traffic predictions and intelligent transportation systems (ITS)s behavior simulations. The interested reader is referred to [9] for an MDE approach supporting simulations of ITSs and as a starting point for related literature on (model-based) simulation of transportation systems. The work presented in this paper instead deals with an orthogonal issue, that is providing travellers with the most suitable journey plans in terms of mobility policies in the municipality, user's preferences, and external conditions (e.g. traffic, weather, etc.). These systems are called personal travel advisors (PTAs) [12]: while modeling efforts exist to support smart travel path choices, they mainly address the analysis of big data for realizing self-learning mechanisms. On the contrary, we exploit well-defined mobility policies and user preferences to compute and possibly update travel plans.

From a broader perspective, the approach described in this paper adopts well-established MDE mechanisms, like separation-of-concerns through multi-view based modelling and model-to-text transformations to generate the artifacts needed in the code-driven mobility configuration layer. As mentioned in Section 6 our aim was to exploit more appropriate megamodeling solutions [8], but we did not succeed in finding a ready to use implementation of megamodeling mechanisms.

8 CONCLUSIONS

In this paper, we presented a model-driven solution to support the continuous deployment and adaptation of a Smart Journey Planner. We have presented the different modeling views, part of a *Smart Mobility Framework*, with their appropriate DSLs. Additionally, we presented the developed prototype for validation purposes. In the future, we plan work in a number of directions including: extending the approach to deal with the more complex interconnections between the views; integrating map-based interfaces with the presented views; realizing a new version of the existing journey planner mobile application that includes the new features illustrated in this work.

[20]http://app.myway-project.eu/
[21]https://citymapper.com/

REFERENCES

[1] Giorgio Ambrosino, John D. Nelson, Marco Boero, and Irene Pettinelli. 2016. Enabling intermodal urban transport through complementary services: From Flexible Mobility Services to the Shared Use Mobility Agency: Workshop 4. Developing inter-modal transport systems. *Research in Transportation Economics* 59 (2016), 179 – 184.

[2] Colin Atkinson, Christian Tunjic, Dietmar Stoll, and Jacques Robin. 2013. A Prototype Implementation of an Orthographic Software Modeling Environment. In *Proceedings of the 1st Workshop on View-Based, Aspect-Oriented and Orthographic Software Modelling (VAO '13)*. ACM, 3:1–3:10.

[3] Frederick P. Brooks, Jr. 1987. No Silver Bullet Essence and Accidents of Software Engineering. *Computer* 20, 4 (April 1987), 10–19. https://doi.org/10.1109/MC.1987.1663532

[4] Hugo Brunelière, Jokin Garcia Perez, Manuel Wimmer, and Jordi Cabot. 2015. EMF Views: A View Mechanism for Integrating Heterogeneous Models. In *34th International Conference on Conceptual Modeling (ER 2015)*. Stockholm, Sweden.

[5] Marcos Didonet Del Fabro and Patrick Valduriez. 2007. Semi-automatic Model Integration Using Matching Transformations and Weaving Models. In *Proceedings of the 2007 ACM Symposium on Applied Computing (SAC '07)*. ACM, New York, NY, USA, 963–970.

[6] Iain Docherty, Greg Marsden, and Jillian Anable. 2017. The governance of smart mobility. *Transportation Research Part A: Policy and Practice* (2017).

[7] Marion Drut. 2018. Spatial issues revisited: The role of shared transportation modes. *Transport Policy* 66 (2018), 85 – 95.

[8] Jean-Marie Favre and Tam NGuyen. 2005. Towards a Megamodel to Model Software Evolution Through Transformations. *Electron. Notes Theor. Comput. Sci.* 127, 3 (April 2005), 59–74.

[9] Alberto FernÃąndez-Isabel and RubÃľn Fuentes-FernÃąndez. 2015. Analysis of Intelligent Transportation Systems Using Model-Driven Simulations. *Sensors* 15, 6 (2015), 14116–14141.

[10] Hans Jeekel. 2017. Social Sustainability and Smart Mobility : Exploring the relationship. *Transportation Research Procedia* 25 (2017), 4296 – 4310.

[11] Maria Kamargianni, Weibo Li, Melinda Matyas, and Andreas Schafer. 2016. A Critical Review of New Mobility Services for Urban Transport. *Transportation Research Procedia* 14 (2016), 3294 – 3303.

[12] Agostino Nuzzolo and Antonio Comi. 2016. Advanced public transport and intelligent transport systems: new modelling challenges. *Transportmetrica A: Transport Science* 12, 8 (2016), 674–699.

[13] Douglas C. Schmidt. 2006. Guest Editor's Introduction: Model-Driven Engineering. *Computer* 39, 2 (Feb. 2006), 25–31. https://doi.org/10.1109/MC.2006.58

[14] Goran Smith, Jana Sochor, and I.C. MariAnne Karlsson. 2018. Mobility as a Service: Development scenarios and implications for public transport. *Research in Transportation Economics* (2018).

[15] Thomas Vogel, Andreas Seibel, and Holger Giese. 2011. The Role of Models and Megamodels at Runtime. In *Proceedings of the 2010 International Conference on Models in Software Engineering (MODELS'10)*. Springer-Verlag, 224–238.