Mobility services data models for open and inclusive MaaS infrastructures

Athanasios I. Salamanis[1a], Theodoros Ioakeimidis[a], Maria Gkemou[b], Dionysios Kehagias[a] Dimitrios Tzovaras[a]

[a]*Information Technologies Institute, Centre for Research & Technology Hellas, P.O Box 60361, GR 57001 Thermi-Thessaloniki, Greece*

[b]*Hellenic Institute of Transport, Centre for Research & Technology Hellas, P.O Box 60361, GR 57001, Thermi-Thessaloniki, Greece*

Abstract

Over the recent years, the vast variety of widely accessible cloud computing services along with the need to combine transportation services either from public or private providers, have led to the rise of the Mobility as a Service (MaaS) concept. The main feature of MaaS is that it gives users access to a set of heterogeneous transportation services from a single access point (i.e., an app). The ever-increasing adoption of MaaS by service providers introduces a variety of new business models and technologies that can successfully support the design and deployment of MaaS services. However, the outcome of this process depends on the definition of a data model for the transportation services, and its proper implementation that will ensure a seamless service provision within the MaaS platform. Towards this direction, this paper presents a definition of the transportation service data model suitable for a MaaS platform, as well as two different implementation approaches. In particular, a custom-design approach and an ontology-based approach are presented and compared with each other based on a set of key performance indicators (KPIs) such as code complexity, code maintainability, and performance. The two approaches were quantitatively compared using both artificially generated and real data derived from a

[1] Corresponding author

real-world MaaS platform, namely the one developed in the context of the H2020 European research project MyCorridor. The preliminary results present the advantages and disadvantages of the two approaches for the implementation of the defined transportation service data model in the context of a real-world MaaS platform.

Keywords: MaaS, JSON schema, ontology, OWL, KPI, cyclomatic complexity, Halstead metrics, maintainability index, technical debt

1. Introduction

Nowadays, many people are concentrating in the big urban centers mainly in search of work. Consequently, a number of issues related to increased transport needs (e.g., traffic congestion) has emerged, leading to the need for a shift towards mobility solutions that provide sustainable and environmental friendly alternatives. Such an alternative is the shared mobility schemes.

Shared mobility schemes include a variety of green mobility solutions (e.g., bike sharing, car sharing, and ride sharing) that decrease the need of vehicle ownership. For example, platforms like Motivate ("Motivate," 2019) and Car2Go ("Car2Go," 2019) offer access to bicycles and cars on-demand. A combination of such mobility options with the available public transport serves as a convenient option compared to using private vehicles. However, these mobility schemes require searching, booking and paying for each service separately. This is a time consuming process, which includes accessing many different applications and using different booking and payment methods.

The idea of Mobility as a Service (MaaS) has emerged as a way of tackling the above difficulties. MaaS provides users the ability to access a set of heterogeneous transportation services, choose those that satisfy their needs, and book and pay for them from a single access point (i.e., an app). All the required processes (e.g., services search, booking, and payment) are handled by the MaaS platform without the need for user intervention. However, at a technical level, the seamless provision of these mobility

solutions requires the integration of several different application programming interfaces (APIs) into the MaaS platform. This integration process depends on the definition of a data model for the transportation services and its proper implementation.

To this end, this paper presents a definition of the transportation service data model suitable for MaaS platforms, as well as two different implementation approaches. In particular, a custom-design approach and a ontology-based approach are presented and compared with each other based on a set of key performance indicators (KPIs) such as code complexity, code maintainability, and performance. The two approaches were quantitatively compared using both artificially generated and real services derived from a real-world MaaS platform, namely the one developed in the context of the H2020 European research project MyCorridor. The experimental results present the pros and cons of each approach for the implementation of the defined data model.

The rest of the paper is organized as follows. Section 2 reviews the current MaaS literature focusing mainly on real-world MaaS implementations. Section 3 presents the definition of the transportation service data model and describes the two different implementation approaches. Section 4 describes the KPIs used to compare the two implementations. Section 5 presents the results of the comparison and highlights the most important findings. Finally, Section 6 concludes the paper by reviewing its main contributions and proposes future research directions.

2. Literature Review

Although MaaS is a relatively new transport paradigm, there has been a great interest in the various aspects of its successful implementation and overall adoption. However, the vast majority of research studies focuses on theoretical rather than practical aspects of MaaS. For example, Kamargianni and Matyas (M Kamargianni & Matyas, 2017) provided a preliminary definition of MaaS and described the different levels, actors and roles that compose the MaaS business ecosystem. Following a similar

approach, Jittrapirom et al. (Jittrapirom et al., 2017) provided a review of existing MaaS definitions and schemes and identified a set of MaaS unique characteristics. Similarly, efficient business model design for mobility platforms was the subject of the research works presented by Aapaoja et al. (Aapaoja, Eckhardt, & Nykänen, 2017) and Eckhardt et al. (Eckhardt, Aapaoja, Nykänen, & Sochor, 2017). Additionally, Ebrahimi et al. (Ebrahimi, Sharmeen, & Meurs, 2017) presented a classification of MaaS business architectures based on the integration of mobility services, while Sarasini et al. (Sarasini, Sochor, & Arby, 2017) identified the different ways in which MaaS business models can generate sustainable value for the several stakeholders.

Assuming that a higher level of mobility integration is more appealing to travellers, Kamargianni et al. (Maria Kamargianni, Li, Matyas, & Schäfer, 2016) provided a comprehensive review of existing MaaS schemes and presented an index for evaluating the level of mobility service integration for each scheme. Furthermore, Li and Voege (Li & Voege, 2017) presented a summarization of the required conditions for MaaS operation and provided a checklist for potential MaaS developers to assess whether MaaS can be implemented in a city. The factors that determine a city's readiness for MaaS implementation were also examined in the work of Goulding and Kamargianni (Goulding & Kamargianni, 2018). Additionally, Matyas and Kamargianni (Matyas & Kamargianni, 2018b) investigated the potential of using subscription plans as a management tool towards the promotion of shared mobility solutions. Furthermore, Matyas and Kamargianni (Matyas & Kamargianni, 2018a) presented a survey that aimed to capture the decision-making process of purchasing MaaS products. In a similar attempt to model the potential demand for MaaS packages, Ho et al. (Ho, Hensher, Mulley, & Wong, 2018) examined the potential uptake of mobility packages as well as the travellers' willingness to pay for different MaaS plans. From a different perspective, Smith et al. (Smith, Sochor, & Karlsson, 2018) examined the potential development of MaaS scenarios (i.e., market-driven, public-controlled, and public-private) and portrayed their implications for

public transport. Finally, Hensher (Hensher, 2017) examined the potential future contexts of bus services provision under the MaaS paradigm.

One of the most important objectives of MaaS platforms is to integrate a variety of different transportation services seamlessly. Towards this direction, Melis et al. (Melis et al., 2018) introduced Smart Mobility for All (SMAll), a platform designed based on the microservices architectural style. The platform's capability of integrating different services was demonstrated by a successful use case presented by Callegati et al. (Callegati et al., 2017). Regarding security within MaaS platforms, Callegati et al. (Callegati, Giallorenzo, Melis, & Prandini, 2018) proposed a gossip-based overlay network architecture in order to constrain the information a malicious user could potentially acquire from the users. Moreover, Thai et al. (Thai, Yuan, & Bayen, 2018) presented a framework for quantifying the vulnerability of MaaS systems to Denial-of-Service (DoS) attacks. Finally, it is important to mention that in addition to theoretical approaches to the MaaS concept, real-world MaaS applications have already been designed, developed, deployed, and used by thousands of users across different countries. Some examples of these applications are Whim ("Whim," 2019), Ubigo ("Ubigo," 2019), Moovit ("Moovit," 2019), and SkedGo ("SkedGo," 2019).

3. Transportation Service Data Model

The term "transportation service", within a MaaS ecosystem, is usually referred to the digital representation of a *mobility product*, which is defined as a real-world physical transportation service provided by a public, private or public-private transport company/authority. For example, a trip with a coach bus from one city to another can be considered as a mobility product, and when it is presented as an option to the traveller through a MaaS app, it can be referred to as transportation service. In the MaaS ecosystem designed and developed in the context of MyCorridor project, the transportation services are

classified into *clusters* and *subclusters*. The definitions of these categories and subcategories are presented in deliverable D1.1 of MyCorridor (Gkemou, 2018).

An important step towards the definition of a concrete transportation service data model is the identification of the minimum information required for describing a service. In MyCorridor MaaS platform, we identified the following attributes that can define a transportation service data model:

- **Name***: the name of the service
- **Cluster***: the cluster of the service
- **Subcluster***: the subcluster of the service
- **Mobility product***: the mobility product represented by the service
- **Operating areas***: a list of service areas (i.e., cities or countries)
- **Operating time periods***: a list of service time periods (i.e., days and/or hours within days)
- **Service provider***: the official name of the service provider
- **URL**: the URL of the official site of the service provider
- **API***: a boolean variable denoting whether service operation information (e.g., itineraries information) are provided through an API
- **API URL***: the base URL of the service's API
- **API response***: the format of the API's responses (e.g., JSON)
- **Booking API***: a boolean variable denoting whether the booking and ticketing information of the service are provided through an API
- **Booking API URL***: the base URL of the service's booking API
- **Booking API response***: the format of the booking API's responses (e.g., JSON)
- **Business rules***: a list of rules describing the business policy of the service (e.g., special discounts for trips on weekends)

- **Mode**: the transportation mode of the service (e.g., bus)
- **Paid***: a boolean variable denoting whether a service is paid or free of charge
- **Currency**: the type of currency in which the service is paid (if the service is paid)
- **Cost**: the cost per trip of the service (if the service is paid)
- **Registration Status**: a tag denoting the status of the service within MyCorridor MaaS platform. This tag can take one of the following values:
    - *Submitted*: The status tag the service receives when it is first submitted to the MyCorridor MaaS platform
    - *Registered*: The status tag of the service after it was successfully evaluated. This status means that the service is provided to the travellers through the MaaS app.
    - *Under Evaluation*: The status tag of the service during the period evaluated for its suitability for the MyCorridor MaaS platform
    - *Under Update*: The status tag of the service, if the provider agrees to proceed with the proposed changes to the service in order to successfully register it to the MyCorridor MaaS platform
    - *Rejected*: The status tag of the service when it has not passed the evaluation process
- **Weight**: a value assigned to the service by the MaaS operator indicating its importance for the overall MaaS ecosystem
- **Average Rating**: the average rating value of the service based on feedback provided by the travellers
- **Comments**: miscellaneous information of the service not described by the other attributes

Attributes denoted by the (*) symbol are mandatory for the complete description of a service, and hence for integration to the MyCorridor MaaS platform; the rest of them are not required (but nice to have) for

successfully registering a service into MyCorridor MaaS platform and take predefined values in case no user input is provided.

The above transportation service data model is implemented in two ways; (a) a custom-design approach and (b) an ontology-based approach, which are described in the following subsections. The actual instances of services derived by both approaches are accessible through an appropriate API. In particular, in the context of MyCorridor MaaS platform a RESTful API was designed and implemented, through which (among other functions) the service instances are accessed. This API was developed using the EVE RESTful API framework ("Python Eve," 2019), which is an open source Python framework that allows effortlessly building and deploying highly customizable, fully featured RESTful Web Services (RWS). It is powered by Flask ("Flask," 2019) and Cerberus ("Cerberus," 2019), and offers native support for MongoDB data stores, as well as rapid prototyping in both development and production level.

Custom-design approach

The first approach followed for the implementation of the above transportation service data model was the custom-design approach. In this, a JSON schema that includes all the attributes of the transportation service data model was defined (i.e., **Service JSON schema**), and based on this schema JSON documents representing different transportation services were generated and stored in a NoSQL database for JSON-like documents, namely MongoDB. In the rest of this subsection, all the key concepts of this approach are briefly described.

JavaScript Object Notation (JSON) is a human-readable data exchange format, which represents data in the form of key-value pairs for the most part (i.e., it also utilizes several other serializable data types, such as arrays). Although JSON was initially derived from JavaScript, it is a language-independent data format and many modern programming languages include code to generate and parse JSON documents. JSON is mostly used for exchanging data in the context of web applications, and it can be considered as an

alternative to XML data format. The main advantages of the JSON format is the comprehensibility of the data representation, its flexibility in representing the data entity in multiple ways, and its lightweight structure. The JSON format has been standardized by both ECMA ("RFC 8259," 2017) and ISO ("ISO/IEC 21778:2017," 2017).

To design a specific data structure in JSON format, a JSON schema should be defined. A JSON schema defines the structure of specific JSON data for validation, documentation, and interaction control. It is essentially a data structure declaration, which is written as a JSON itself. The JSON schema provides a specification for the data required by an application, and how that data can be modified. The JSON schema is a specification similar to the XML schema (i.e., the XSD).

As mentioned above, the instances of the Service JSON schema (i.e., JSON documents) are stored in a document-oriented database called MongoDB ("MongoDB," 2019). MongoDB is a free open-source cross-platform database that belongs to the category of the NoSQL databases, and it stores data in the form of JSON-like documents, using a format called binary JSON (BSON) ("BSON," 2019). MongoDB is the most popular document-oriented NoSQL database as of November 2019 according to DB-Engines Ranking ("DB-Engines Ranking," 2019). MongoDB has very big and active developer community, it contains lots of supporting tools, and it yields better performance, scalability, consistency and data integrity compared to other document-oriented NoSQL databases (e.g. Apache CouchDB ("Apache CouchDB," 2019) and Amazon DynamoDB ("Amazon DynamoDB," 2019)).

The defined Service JSON schema is presented in Figure 1 and Figure 2. Figure 1 presents the Service JSON schema as it was extracted from the Eve framework. Figure 2 presents the Service JSON schema as extracted from Swagger ("Swagger," 2019), which is the documentation tool of the MyCorridor RESTful API. The fields of the Service JSON schema are the attributes of the defined transportation service data

model presented above. These fields are described in detail in deliverable D3.1 of MyCorridor (Salamanis, 2019).

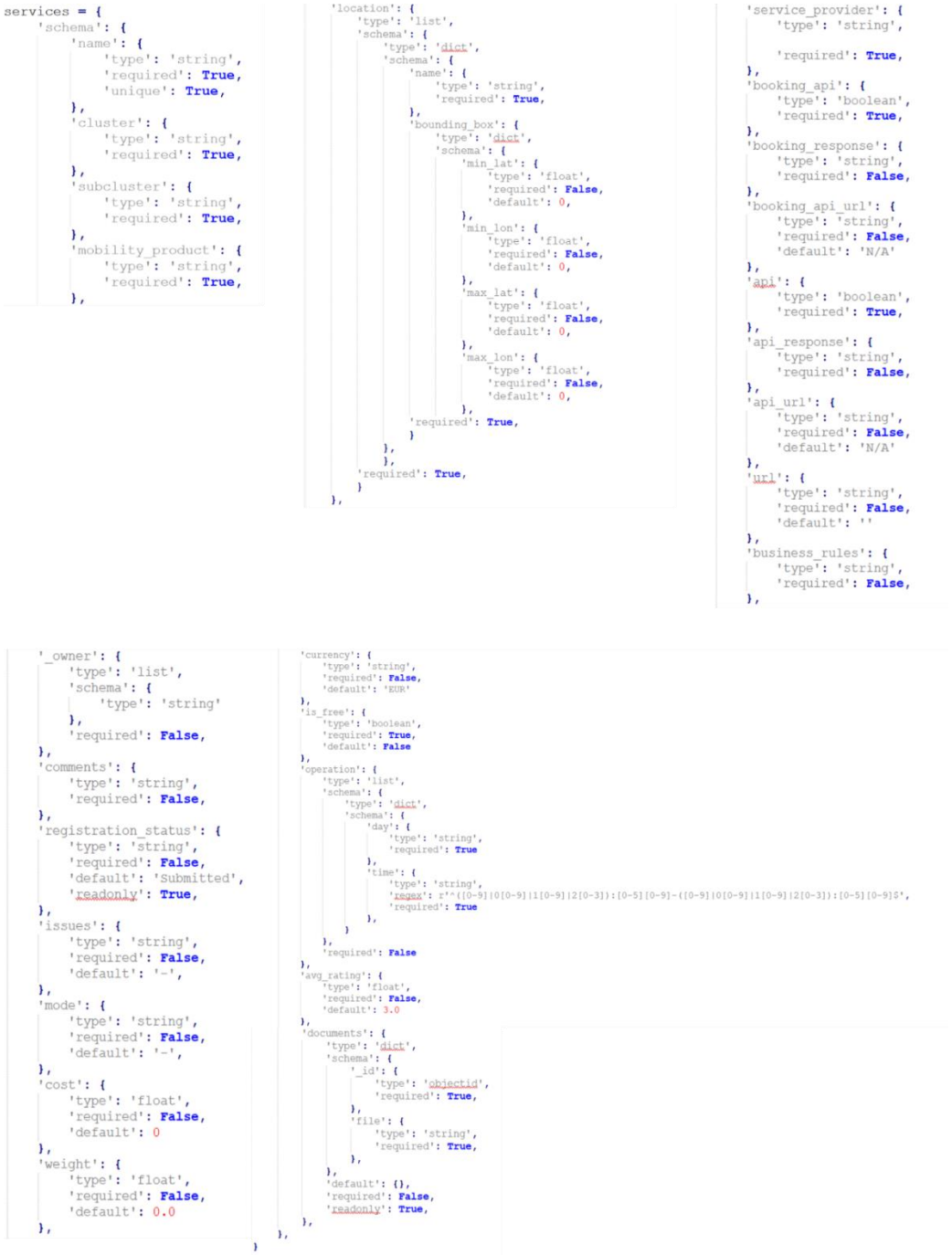

Figure 1: Service JSON schema as extracted from the Eve framework

Figure 2: Service JSON schema as extracted from the Swagger API documentation tool

Ontology-based approach

The second approach for the implementation of the defined transportation service data model is based on *ontologies*. An ontology is a representation of concepts, data and entities within a domain, along with the relationships between them, using formal semantics. In essence, an ontology is a formal way to define all key aspects of a concept (e.g. attributes, categories/subcategories, and relationships with other concepts), and provides a common language for all stakeholders working with this concept. For example, in the context of movie recommendation systems, an ontology can be defined to represent the concept

of a movie. This ontology will include the formal definition of the movie (i.e., the attibutes of a movie), the several subcategories (e.g., action movies, or comedy movies), as well as its relationships with other concepts (e.g., actors, film producers, and audience). An ontology may also be considered similar to the concept of a class, in the context of object-oriented programming.

An ontology is implemented using a formal ontology language. There are several formal ontology languages, both proprietary and standards-based. For the implementation of the ontology that represents the transportation service data model, we utilized the *OWL* ontology language. OWL is a family of knowledge representation languages for authoring ontologies, which are built upon the World Wide Web Consortium's (W3C) XML standard for objects in the Resource Description Framework (RDF) (Group, 2012). In OWL, a *class* is the main representation of the ontology, and it represents a real-life concept. Then, an *instance* is a realization of a class. A class may contain many instances, and an instance may belong to none, one or more classes. Moreover, a class may have *subclasses*, which inherit attibutes from their parent *superclass*. In OWL, a class is represented by a noun. Additionally, the classes have characteristics, i.e. directed binary relations that specify some attributes which are true for instances of the classes. These characteristics are called *properties*, and they are represented as verbs.

In the field of transportation, there have been some efforts for creating ontologies that represent transport related data (Sobral, Galvão, & Borges, 2017)(Corsar, Markovic, Edwards, & Nelson, 2015)(Berdier, 2011)(Gunay, Akcay, & Altan, 2014)(Koç, Lantow, & Sandkuhl, 2014), but in most cases the generated ontologies were application-specific and could not be easily generalized and extended to other related applications. Additionally, in the context of the MobiVoc (“MobiVoc,” 2019) and the Oasis (“OASIS,” 2017) projects, there has been an effort to design a generic ontology that could potentially support the operation of MaaS platforms. In particular, the objective of the MobiVoc project was to improve the data transferability between all stakeholders involved in the transport industry, by providing a standardized vocabulary, i.e. the Open Mobility Vocabulary. Similarly, the Oasis project aimed to

increase the accessibility of public transport through linked open data, utilizing ontology-based architecture.

Following the work of the aforementioned projects, we implemented the **Service OWL ontology** that represents the transportation service data model. In particular, the following classes were implemented:

- **Service**: it represents the concept of a service
- **Mobility**: it represents the concept of a mobility service
- **Infomobility**: it represents the concept of an infomobility service
- **Traffic Management**: it represents the concept of a traffic management service
- **Added Value**: it represents the concept of an added value service
- **Operation**: it represents the working hours of a service
- **Bbox**: it represents an operating area of a service, in the form of a bounding box

The Mobility, Infomobility, Traffic Management and Added Value classes are subclasses of the Service class. The instances of the Service class are related with the instances of the Operation class via the *hasOperation* property and with the instances of the Bbox class via the *hasBbox* property.

The Service class has the following datatype properties:

- *hasName* (string): it represents the name of a service
- *hasMobilityProduct* (string): it represents the mobility product of a service
- *hasProvider* (string): it represents the service provider of a service
- *hasUrl* (string): it represents the URL of the official website of a service
- *hasApi* (boolean): it represents the existence of an API for a service
- *hasApiUrl* (string): it represents the base URL of the API of a service

- *hasApiResponse* (string): it represents the data format (e.g., JSON) of the responses from the API of a service
- *hasBusinessRules* (string): it represents the business rules of a service
- *hasCurrency* (string): it represents the currency used to make payments for the purchase of a service
- *hasRegistrationStatus* (string): it represents the registration status of a service
- *hasWeight* (float): it represents the weight of a service
- *hasAvgRating* (float): it represents the average rating of a service

Additionally, the Service class has the following object properties:

- *hasOperation*: it represents a specific time period of operation of an instance of the Service class
- *hasBbox*: it represents an operating area of an instance of the Service class

The Mobility, Infomobility, Traffic Management and Added Value classes inherit all the datatype properties of the Service superclass. Moreover, the Mobility class has the following additional datatype properties:

- *hasBookingUrl* (string): it represents the base URL of the booking API of a mobility service
- *hasBookingResponse* (string): it represents the data format (e.g., JSON) of the responses from the booking API of a mobility service
- *hasMode* (string): it represents the transportation mode (e.g., bus) of a mobility service

It should be noted that by default we assume that a mobility service has a booking API, and therefore no *hasBookingApi* property is required. The Operation class has the following datatype properties:

- *hasDay* (string): it represents the day of the operation period of a service
- *hasTime* (string): it represents a specific time period within a day for the operation period of a service

Finally, the Bbox class has the following datatype properties:

- *hasBboxName* (string): it represents the name of the operating area of a service
- *hasMinLat* (float): it represents the latitude of the down left point of the bounding box that corresponds to an operating area of a service
- *hasMinLon* (float): it represents the longitude of the down left point of the bounding box that corresponds to an operating area of a service
- *hasMaxLat* (float): it represents the latitude of the upper right point of the bounding box that corresponds to an operating area of a service
- *hasMaxLon* (float): it represents the longitude of the upper right point of the bounding box that corresponds to an operating area of a service

Figure 3 depicts the classes, along with their properties, that belong to the Service OWL ontology. The Service OWL ontology was created using the semantic editor Protégé ("Protégé," 2019). Moreover, the Owlready2 ("Owlready2," 2019) library has been used for parsing and updating the ontology. Owlready2 is a package for ontology-oriented programming in Python that allows for loading, modifying and saving ontologies as Python objects.

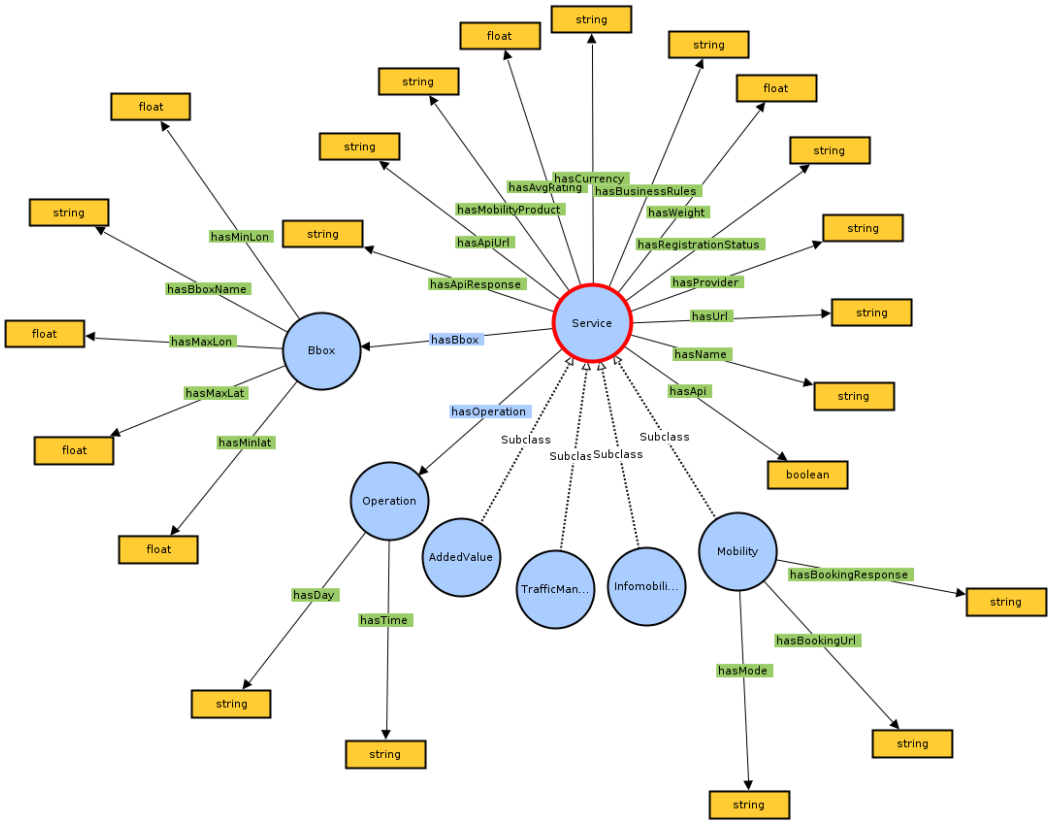

Figure 3: The Service OWL ontology

4. Key Performance Indicators

In order to quantitatively compare the two implementation approaches, we had to select some specific KPIs. Several aspects of a MaaS platform's functionality are affected by the appropriate definition and implementation of the data model representing a transportation service, and hence the selected KPIs should reflect as many of these aspects as possible. It should also be noted that most of the selected KPIs are evaluated *indirectly* on the two service data model implementations, through the corresponding software components that are responsible for handling them (e.g., generating new instances of the data models, and parsing the instances). This means that when the name of a KPI starts with the term "code" it refers to the software components responsible for handling the two service data model implementations. Based on these clarifications, the following KPIs were selected: code complexity, code

explainability and speed of code development, code maintainability, performance, and size. The “code” KPIs were calculated using two static code analysis tools, namely Radon (“Radon,” 2019) and SonarQube (“SonarQube,” 2019). In particular, all the utilized code metrics were calculated using Radon, except of the technical-debt-related metrics that were calculated using SonarQube. Finally, the performance and size KPIs were calculated through experimental processes defined by the authors. In the rest of this section, the selected KPIs are described.

Code Complexity

The term code complexity in software engineering reflects the interconnections between the number of various software entities. As this number tends to grow, measuring the overall complexity is important for assuring a high code quality. In order to quantify the complexity of the software modules responsible for handling the two service data model implementations, we mainly utilized the *Cyclomatic Complexity (CC)* metric, which is a measure of the number of linearly independent paths through the code. Additionally, we utilized the following raw metrics:

- *LOC*: the total number of lines of code
- *LLOC*: the number of logical lines of code
- *SLOC*: the number of source lines of code (not necessarily corresponding to the LLOC)
- *Comments*: the number of comment lines
- *Multi*: the number of lines, which represent multi-line strings
- *Blanks*: the number of blank lines (or whitespace-only ones)

The following equation should always hold:

$$SLOC + Multi + Single\ Comments + Blank = LOC \quad (1)$$

Code Explainability/Speed of Code Development

In order to enhance the identification of the measurable properties of software and their relations, Maurice Howard Halstead introduced a set of metrics known as the Halstead complexity measures (Halstead, 1977). More specifically, the following numbers can be computed by statically analyzing source code:

- n_1: number of distinct operators
- n_2: number of distinct operands
- N_1: the total number of operators
- N_2: the total number of operands

Based on these numbers, the following measures can be calculated:

- *Program Vocabulary*: $n = n_1 + n_2$
- *Program Length*: $N = N_1 + N_2$
- *Calculated Estimated Program Length*: $\widehat{N} = n_1 \log_2 n_1 + n_2 \log_2 n_2$
- *Volume*: $V = N \times \log_2 n$
- *Difficulty*: $D = \frac{n_1}{2} \times \frac{N_2}{n_2}$
- *Effort*: $E = D \times V$
- *Time Required To Program*: $T = \frac{E}{12}$ (sec.)
- *Number of delivered bugs*: $B = \frac{E^{\frac{2}{3}}}{3000}$

In order to quantify the explainability of the software components responsible for handling the two service data model implementations, we employed the Halstead Difficulty metric, as it is connected with someone's ability to understand a piece of code. Additionally, in order to measure the speed of development we utilized the Halstead Time Required To Program (ttp) metric.

Code Maintainability

The concept of code maintainability in software engineering refers to the ease of software maintenance in terms of source code support and modification. In this paper, the comparison of the two service data model implementations in terms of maintainability was based on the *Maintainability Index (MI)* and the *Technical Debt (TD)* metrics. More specifically, the MI is defined by the following equation:

$$MI = \max\left[0, 100\ \frac{171 - 5.2\ln V - 0.23G - 16.2\ln L + 50\sin\left(\sqrt{2.4C}\right)}{171}\right] \quad (2)$$

where V is the Halstead Volume, G is the total Cyclomatic Complexity, L is SLOC and C is the percent of comment lines converted to radians. Additionally, the TD reflects the effort required for addressing the several maintainability issues (i.e., fixing all code smells), and it is measured in time units (e.g., minutes, hours, and 8-hour days).

Performance

Another KPI that we wanted to estimate regarding the two service data model implementations was the performance of the platform (i.e., of the MyCorridor RESTful API) against multiple requests. This performance is affected by the way the transportation service is represented, or in other words, by the service data model implementation used. In order to measure this performance, we set up the following process. Given N_s services registered (i.e., stored) in the platform, we made N_c calls to the API using an external client. The calls were performed sequentially, and we measured the overall *cumulative response time* (crt - in seconds). The process was executed for both the custom-design approach and the ontology-based approach. Additionally, the process was performed for different values of N_s and N_c, it was repeated five times for each set of (N_s, N_c) values (in order to have stable results), and the *average cumulative response time (*$acrt$*)* was calculated for each case. Finally, it should be noted that both real (from the MyCorridor platform) and artificially generated services were utilized in the above process.

Size

Finally, we compared the two service data model implementations in terms of size requirements for data storage. For this purpose, we utilized N_s instances of transportation services (both real and artificially generated), and we measured the size (in kilobytes -KB) of the required space for their storage, using both the custom-design and the ontology-based approaches. This process was performed for different values of N_s. The size of the required space for data storage was extracted from MongoDB statistics.

5. Results and Discussion

In this section, we present the results of the implementation of the KPIs on the two proposed service data model implementations and their corresponding software components. Based on these results, we try to highlight some of the advantages and disadvantages of both approaches. The results of the implementation of the code complexity KPI are presented in Table 1.

Table 1: Code Complexity results

	Service JSON schema	**Service OWL ontology**
CC	B (8.0)	B (10.0)
LOC	248	319
LLOC	50	179
SLOC	227	255
Comments	16	62
Multi	0	0
Blank	6	10

As shown in the table, the code complexity metrics values derived by the implementation of the metrics on the software component responsible for handling the Service OWL ontology are, in all cases, higher than the corresponding metrics derived by the implementation of the metrics on the software component responsible for handling the Service JSON schema. This means that the Service OWL ontology is a much

more complex representation compared to the Service JSON schema, and thus it requires more and more complicated code in order to be handled.

The results of the implementation of the code explainability and the speed of code development KPIs are presented in Table 2 and

Table 3, respectively. As shown in the tables, the two proposed approaches have the same value on the Halstead Difficulty (3.3), but they have considerably different values on the Halstead ttp (i.e., 65.6 seconds for the Service OWL ontology versus 29.3 seconds for the Service JSON schema). These results may seem contradictory at first glance, but they can be explained if they are combined with the results of the code complexity metrics. In particular, as already mentioned, the code complexity of the Service OWL ontology implementation is very much higher than the complexity of the Service JSON schema. This means that the time required for a developer to write the code of a software component that will handle the Service OWL ontology, should be much higher than the corresponding time for the Service JSON schema. This is verified by the ttp results. However, this does not *necessarily* mean that the code of the Service OWL ontology's software component would be much more difficult to understand compared to the one of the Service JSON schema. For example, the code of the Service OWL ontology's software component may include lots of if-else statements that, on one hand, increase the program vocabulary and length and consequently the volume and ttp, but on the other, they do not (always) result in a code that is much more difficult to understand.

Table 2: Code Explainability results

	Service JSON schema	**Service OWL ontology**
Halstead Difficulty	3.3	3.3

Table 3: Speed of Code Development results

ttp	**Service JSON schema**	**Service OWL ontology**

	29.3 sec.	65.6 sec.

The results of the implementation of the code maintainability KPI are presented in Table 4. As shown in the table, both implementations present very high maintainability (i.e., MI 70.11 for the Service JSON schema and 54.12 for the Service OWL ontology), which can be considered as an indication that both implementations can be utilized for maintaining a large repository of transportation services in a MaaS platform. On the contrary, the TD metric indicates that the Service OWL ontology may require much more effort for fixing possible code smells compared to the Service JSON schema. However, this result is directly connected with the code complexity of the Service OWL ontology's software components as presented above.

Table 4: Code Maintainability results

	Service JSON schema	**Service OWL ontology**
MI	A (70.11)	A (54.12)
TD	10 min.	1 h. 53 min.

The results of the implementation of the performance KPI are presented in Table 5, Table 6, and Table 7. As shown in the tables, in all cases, the performance of the MaaS platform (or of the MyCorridor RESTful API that orchestrates the operation of the platform) in terms of serving multiple external requests is approximately the same for both of the proposed implementation approaches.

Table 5: Performance results for $N_s = 30$

Service JSON schema						
N_c	crt_1	crt_2	crt_3	crt_4	crt_5	$acrt$
10	0.092935	0.060275	0.060431	0.058606	0.059404	0.066330
100	0.810309	0.833953	0.857855	0.831901	0.838257	0.834455
1000	8.756840	8.518682	8.475384	8.399318	8.548431	8.539731

Service OWL ontology						
N_c	crt_1	crt_2	crt_3	crt_4	crt_5	$acrt$
10	0.074939	0.058827	0.097976	0.058725	0.057742	0.069642
100	0.896580	0.879611	0.879010	0.869142	0.851595	0.875188
1000	8.448229	8.784771	8.326078	8.513504	8.504259	8.515368

Table 6: Performance results for $N_s = 100$

Service JSON schema						
N_c	crt_1	crt_2	crt_3	crt_4	crt_5	$acrt$
10	0.073349	0.070097	0.089286	0.070832	0.060039	0.072721
100	0.887065	0.882068	0.828549	0.793458	0.875982	0.853424
1000	8.612768	8.538605	8.677484	8.511654	8.398195	8.547741
Service OWL ontology						
N_c	crt_1	crt_2	crt_3	crt_4	crt_5	$acrt$
10	0.097689	0.063266	0.076968	0.069008	0.072439	0.075874
100	0.828521	0.800629	0.821406	0.797074	0.856451	0.820816
1000	8.463793	8.414838	8.507498	8.711283	8.667731	8.553028

Table 7: Performance results for $N_s = 1000$

Service JSON schema						
N_c	crt_1	crt_2	crt_3	crt_4	crt_5	$acrt$
10	0.071040	0.071846	0.074694	0.080554	0.085402	0.076707
100	0.967715	0.987158	0.975828	1.000964	0.966392	0.979610
1000	9.802965	9.841989	9.596794	9.873627	10.013564	9.825787
Service OWL ontology						
N_c	crt_1	crt_2	crt_3	crt_4	crt_5	$acrt$
10	0.088310	0.103115	0.091013	0.088604	0.126367	0.099481
100	1.215834	1.185732	1.123688	1.260968	1.213071	1.199858
1000	11.892021	10.431661	11.378388	11.466034	11.348439	11.303308

Finally, the results of the implementation of the size KPI are presented in Table 8. As shown in the table, the Service OWL ontology results in instances that require much more space for storage than the instances of the Service JSON schema.

Table 8: Size results

N_s	Service JSON schema	Service OWL ontology
30	21 KB	104 KB
100	70 KB	267.8 KB
1000	707 KB	2000 KB

6. Conclusions

In this paper, we provided a definition for the concept of the transportation service in a MaaS ecosystem, and we proposed two different approaches for the implementation of this definition, namely the custom-design approach and the ontology-based approach. Then, we defined a set of KPIs in order to quantitatively compare the two proposed approaches. We run several experiments based on these KPIs and we acquired preliminary results. Based on these results, we identified that the proposed Service JSON schema (i.e., the software component that handles it) presents less code complexity, higher maintainability, it requires less time to program, and its instances require less space to be stored compared to the Service OWL ontology. In the terms of code explainability and performance, the two approaches yield similar behavior. We want to stress out here that these results are not enough to be able to draw a safe conclusion about which approach is most appropriate for the implementation of the proposed service data model. For this reason, our future work includes the extension of the comparisons on other KPIs such as extensibility, ability to create hierarchies of services, and transferability from one MaaS platform to another, in order to be able to safely (i.e., based on concrete quantitative results) propose one of the two approaches for the implementation of the defined transportation service data model in a generic MaaS ecosystem.

Abbreviations

MaaS: Mobility as a Service; KPI: Key Performance Indicator, API: Application Programming Interface, SMAll: Smart Mobility For All, DoS: Denial of Service, URL: Uniform Resource Locator, RWS: RESTful Web Services, JSON: JavaScript Object Notation, XML: Extensible Markup Language, XSD: XML Schema Definition, ECMA: European Computer Manufacturers Association, ISO: International Organization for Standardization, OWL: Web Ontology Language, W3C: World Wide Web Consortium, RDF: Resource Description Framework, CC: Cyclomatic Complexity, LOC: Lines of Code, LLOC: Logical Lines of Code, SLOC: Source Lines of Code, MI: Maintainability Index, TD: Technical Debt

Acknowledgements

This work was supported by the European Union's Research and Innovation Collaborative Project "MyCorridor: Mobility as a Service in a multimodal European cross-border corridor" under Grant Agreement No. 723384.

References

Aapaoja, A., Eckhardt, J., & Nykänen, L. (2017). Business models for MaaS. *1st International Conference on Mobility as a Service*.

Amazon DynamoDB. (2019). Retrieved October 10, 2019, from https://aws.amazon.com/dynamodb/

Apache CouchDB. (2019). Retrieved October 10, 2019, from https://couchdb.apache.org/

Berdier, C. (2011). An ontology for urban mobility. In *Advanced Information and Knowledge Processing*. https://doi.org/10.1007/978-0-85729-724-2_14

BSON. (2019). Retrieved October 17, 2019, from http://bsonspec.org/

Callegati, F., Delnevo, G., Melis, A., Mirri, S., Prandini, M., & Salomoni, P. (2017). I want to ride my bicycle: A microservice-based use case for a MaaS architecture. *IEEE Symposium on Computers and Communications*. https://doi.org/10.1109/ISCC.2017.8024498

Callegati, F., Giallorenzo, S., Melis, A., & Prandini, M. (2018). Cloud-of-Things meets Mobility-as-a-Service: An insider threat perspective. *Computers and Security*, *74*, 277–295. https://doi.org/10.1016/j.cose.2017.10.006

Car2Go. (2019). Retrieved September 24, 2019, from https://www.car2go.com/US/en/

Cerberus. (2019). Retrieved October 18, 2019, from https://docs.python-cerberus.org/en/stable/

Corsar, D., Markovic, M., Edwards, P., & Nelson, J. D. (2015). The transport disruption ontology. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. https://doi.org/10.1007/978-3-319-25010-6_22

DB-Engines Ranking. (2019). Retrieved October 12, 2019, from https://db-engines.com/en/ranking

Ebrahimi, S., Sharmeen, F., & Meurs, H. (2017). Innovative Business Architectures (BAs) for Mobility as a Service (MaaS) – exploration, assessment, and categorization using operational MaaS cases. *Transportation Research Board 97th Annual Meeting*.

Eckhardt, J., Aapaoja, A., Nykänen, L., & Sochor, J. (2017). Mobility as a Service business and operator models. *12th ITS European Congress*. Strasbourg.

Flask. (2019). Retrieved October 18, 2019, from https://palletsprojects.com/p/flask/

Gkemou, M. (2018). *MyCorridor Use Cases, Deliverable 1.1 MyCorridor (Mobility as a Service in a multimodal European cross-border Corridor) project (G.A.: 723384), http://mycorridor.eu/*.

Goulding, R., & Kamargianni, M. (2018). *The Mobility as a Service Maturity Index: Preparing the Cities for the Mobility as a Service Era*. https://doi.org/10.5281/ZENODO.1485002

Group, W. O. W. (2012). *OWL 2 Web Ontology Language Document Overview (Second Edition)*. Retrieved from https://www.w3.org/TR/owl2-overview/

Gunay, A., Akcay, O., & Altan, M. O. (2014). Building a semantic based public transportation geoportal compliant with the INSPIRE transport network data theme. *Earth Science Informatics*. https://doi.org/10.1007/s12145-013-0129-z

Halstead, M. H. (1977). *Elements of Software Science (Operating and Programming Systems Series)*. New

York, NY, USA: Elsevier Science Inc.

Hensher, D. A. (2017). Future bus transport contracts under a mobility as a service (MaaS) regime in the digital age: Are they likely to change? *Transportation Research Part A: Policy and Practice*, *98*, 86–96. https://doi.org/10.1016/j.tra.2017.02.006

Ho, C. Q., Hensher, D. A., Mulley, C., & Wong, Y. Z. (2018). Potential uptake and willingness-to-pay for Mobility as a Service (MaaS): A stated choice study. *Transportation Research Part A: Policy and Practice*, *117*, 302–318. https://doi.org/10.1016/j.tra.2018.08.025

ISO/IEC 21778:2017. (2017). Retrieved October 15, 2019, from https://www.iso.org/standard/71616.html

Jittrapirom, P., Caiati, V., Feneri, A.-M., Ebrahimigharehbaghi, S., González, M. J. A., & Narayan, J. (2017). Mobility as a Service: A Critical Review of Definitions, Assessments of Schemes, and Key Challenges. *Urban Planning*. https://doi.org/10.17645/up.v2i2.931

Kamargianni, M, & Matyas, M. (2017). The Business Ecosystem of Mobility-as-a-Service. *96th Transportation Research Board (TRB) Annual Meeting*. Washington.

Kamargianni, Maria, Li, W., Matyas, M., & Schäfer, A. (2016). A Critical Review of New Mobility Services for Urban Transport. *Transportation Research Procedia*. https://doi.org/10.1016/j.trpro.2016.05.277

Koç, H., Lantow, B., & Sandkuhl, K. (2014). Ontology development for intelligent information logistics in transportation. *CEUR Workshop Proceedings*.

Li, Y., & Voege, T. (2017). Mobility as a Service (MaaS): Challenges of Implementation and Policy Required. *Journal of Transportation Technologies*, *07*, 95–106. https://doi.org/10.4236/jtts.2017.72007

Matyas, M., & Kamargianni, M. (2018a). Survey design for exploring demand for Mobility as a Service plans. *Transportation*.

Matyas, M., & Kamargianni, M. (2018b). The potential of mobility as a service bundles as a mobility management tool. *Transportation*. https://doi.org/10.1007/s11116-018-9913-4

Melis, A., Mirri, S., Prandi, C., Prandini, M., Salomoni, P., & Callegati, F. (2018). Integrating Personalized and Accessible Itineraries in MaaS Ecosystems Through Microservices. *Mobile Networks and Applications*, *23*, 167–176. https://doi.org/10.1007/s11036-017-0831-z

MobiVoc. (2019). Retrieved October 19, 2019, from https://www.mobivoc.org/

MongoDB. (2019). Retrieved October 10, 2019, from https://www.mongodb.com/

Moovit. (2019). Retrieved September 25, 2019, from https://moovitapp.com/

Motivate. (2019). Retrieved September 24, 2019, from https://www.motivateco.com/

OASIS. (2017). Retrieved October 19, 2019, from https://oasis.team/

Owlready2. (2019). Retrieved October 22, 2019, from https://pypi.org/project/Owlready2/

Protégé. (2019). Retrieved October 20, 2019, from https://protege.stanford.edu/

Python Eve. (2019). Retrieved October 16, 2019, from http://docs.python-eve.org/en/stable/

Radon. (2019). Retrieved November 5, 2019, from https://radon.readthedocs.io/en/latest/

RFC 8259. (2017). Retrieved October 15, 2019, from https://tools.ietf.org/html/rfc8259

Salamanis, A. (2019). *MyCorridor cloud service delivery platform, service gateway, big data management module and business rules implementer module, Deliverable 3.1 MyCorridor (Mobility as a Service in a multimodal European cross-border Corridor) project (G.A.: 723384), http:/*.

Sarasini, S., Sochor, J., & Arby, H. (2017). What characterises a sustainable MaaS business model? *1st International Conference on Mobility as a Service (ICOMaaS)*.

SkedGo. (2019). Retrieved September 25, 2019, from https://skedgo.com/

Smith, G., Sochor, J., & Karlsson, M. (2018). Mobility as a Service: Development scenarios and implications for public transport. *Research in Transportation Economics*, *69*, 592–599. https://doi.org/10.1016/j.retrec.2018.04.001

Sobral, T., Galvão, T., & Borges, J. (2017). Semantic integration of urban mobility data for supporting visualization. *Transportation Research Procedia*. https://doi.org/10.1016/j.trpro.2017.05.106

SonarQube. (2019). Retrieved October 10, 2019, from https://www.sonarqube.org/

Swagger. (2019). Retrieved October 15, 2019, from https://swagger.io/

Thai, J., Yuan, C., & Bayen, A. M. (2018). Resiliency of Mobility-as-a-Service Systems to Denial-of-Service Attacks. *IEEE Transactions on Control of Network Systems*, *5*, 370–382. https://doi.org/10.1109/TCNS.2016.2612828

Ubigo. (2019). Retrieved September 25, 2019, from https://www.ubigo.me/

Whim. (2019). Retrieved September 25, 2019, from https://whimapp.com/