ATLAS: A World-wide Travel Assistant exploiting Service-based Adaptive Technologies

Antonio Bucchiarone, Martina De Sanctis, Annapaola Marconi

Fondazione Bruno Kessler, Via Sommarive, 18, Trento, Italy
{bucchiarone,msanctis,marconi}@fbk.eu

Abstract. Nowadays, users can count on a large amount of mobility services offering disparate functionalities and providing all needed information. Yet, from a user perspective, properly exploiting the available mobility services to organize journeys meeting personal expectations, is becoming a complex task. Indeed, discover and select the appropriate services in an open and constantly expanding domain, is a challenging and time-consuming task. We claim that a uniform and easy way for exploiting these services while moving around, getting accurate and personalized information is still missing. In this paper we propose a platform for the definition of value-added mobility services by (i) enhancing interoperability among the existing services, (ii) supporting their execution via run-time adaptation, (iii) through the definition of multi-channel front-end applications. On top of the platform, we have implemented and evaluated a world-wide travel assistant.

1 Introduction

Today, a multitude of applications offering flexible, dynamic and personalized mobility services to users are available in the *mobility domain*. These services are designed independently from each other and made available through a large variety of different technologies (e.g., web pages, mobile apps). They provide solutions that are fragmented, limited, and that have a partial coverage (e.g., only planning, only booking) of the overall journey. For instance, *Rome2Rio* [1] is a world-wide multi-modal journey planner, that offers traveling solutions between two given locations, but it is not consistently integrated in the (local) mobility offer of a city (i.e., local bus schedules). *Viaggia Trento* [2], instead, is an accurate local multi-modal planner for the city of Trento. In this context, often the users must interact with different applications to accomplish a journey. This makes the benefits of having multiple and accurate mobility services a drawback instead of an added value for the users. To overcome these limits and to leverage on the potentialities of the available services, we need a systemic and general approach dealing in a uniform way with services of an open and heterogeneous context. In this way, we can facilitate services integration and interoperability.

In this paper, we present a service delivery platform providing methods and techniques to design and release adaptive service-based applications. The platform capitalizes on the achievements and findings of our research in the last years. In particular, this work represents the combination of the following results: (1) a *design for adaptation* approach supporting the development, deployment and execution of service-based systems operating in dynamic environments [5, 6]; (2) a comprehensive framework for

[1] https://www.rome2rio.com/ [2] http://www.smartcommunitylab.it/apps/viaggia-trento/

automated service composition [4] that allows for context-aware service adaptation, and (3) a set of implemented software components and prototypes [1, 9]. To show the potentiality offered by the platform, we implemented a world-wide travel assistant (ATLAS) able to provide accurate and context-aware traveling solutions.

In the rest of the paper we discuss the challenges behind this work, we presents all the details of our service delivery platform and its usage, and we report the experimental validation of the platform.

2 Challenges and Application Scenario

Nowadays, users can count on a large amount of mobility services. They may differ depending on the offered functionalities, the targeted users, or the provider. In particular, there are *journey planners* (e.g., Rome2Rio, Google Transit [3]) for finding traveling solutions between two or more given locations. Then, *specific mobility services* are those referring to specific transport modes (e.g., CityBikes [4] focuses exclusively on bike sharing data) or provided by transport companies (e.g., Flixbus [5], Trenitalia [6]). Moreover, an emerging trend is that of *shared mobility services* that are based on the shared use of vehicles, bicycles, or other means (e.g., Bla Bla Car [7]). Mobility services also differ in their *geographic coverage*. For instance, while Google Transit is a *global* planner, since it can be used for planning all around the world, ViaggiaTrento is a *local* planner for the city of Trento. The *transport modes coverage*, instead, measures the number of different transport means handled by mobility services (i.e., *single mode* and *multiple mode*). For instance, Flixbus and Trenitalia refer to a single transport mode, namely bus and train. To the contrary, journey planners usually consider different transport modes. Furthermore, both services dealing with one or a few transport modes and services having a local coverage are characterized by a high accuracy of the provided data. To the contrary, the more global are the services, the more they tend renounce accuracy. Focusing on cities, we can observe that there is a lot of disparate local services, which are specific for a few transport means and very accurate. However, this implies that, while moving around and changing their context, users need to discover and exploit the respective services (and applications) in each city. To sum up, besides the huge amount of mobility services available up today, it is still missing for the users the possibility of getting **context-aware**, **accurate** and **personalized** travel solutions while moving around, without using different applications. In this context, there is no need for yet-another-mobility-app. Our goal, instead, is to provide a solution for enhancing mobility services interoperability through their runtime and context-aware discovery and composition, to exploit their potentialities and fill their gaps.

Application scenario. Sara is living in Trento. She wants to visit Vienna, in Austria. ViaggiaTrento does not give her any results, being Vienna out of region, so she opens the Trenitalia mobile app and she starts planning. Unlikely, the founded solutions implies at least two changes, and she does not like the idea. Sara thinks that a rideshare or a bus solution would be also less expensive, if available. So she checks for both a BlaBlaCar ride and a Flixbus travel. Finally, she founds a cheap and direct solution among the ones given by Flixbus, and she books it. For organizing her travel, Sara has used four different mobility apps, by relying on her knowledge of services, without any support.

[3] https://maps.google.com/landing/transit/index.html [4] https://www.citybik.es/
[5] https://www.flixbus.com/ [6] http://www.trenitalia.com/ [7] https://www.blablacar.com

3 System Implementation

In this Section, we present our service delivery platform and a world-wide *personAlized TraveL AssiStant – ATLAS* developed on top of it. ATLAS consists in (i) a demonstrator showing the system's models and its execution and evolution through automatic runtime adaptation, and (ii) a Telegram [8] chat-bot, for the interaction with the users. We remark that ATLAS exploits real-world mobility services exposed as open APIs, which are wrapped as domain objects to be effectively part of the system.

3.1 Adaptive Service-based Systems through Domain Objects

The *Domain Object Model* [5, 6] has been built to satisfy the need for service-based applications adaptable by-design. A domain object represents a uniform way to model independent, heterogeneous, and open services such that they can be easily interconnected thus enhancing *services interoperability*. Each domain object defines the behavior of the service it models–*core process* (e.g., BlaBlaCar ride-sharing), and the functionalities it provides–*fragments* (e.g., offer/require a car ride). Unlike traditional services, domain objects allow the partial specification of the expected behavior of a service by defining *abstract activities*. These activities are defined in terms of the *goal* they need to achieve (e.g., organize a journey). When, at runtime, abstract activities need to be executed, they can be *refined* according to the fragments offered by other domain objects, thus allowing the goal to be reached. Indeed, fragments represent executable processes that can be dynamically discovered, received and executed by a domain object.

While abstract activities goals are defined at design time at a conceptual level, the need for refining them arises at runtime, triggering real services interoperability. Indeed, only during the execution the system can discover and select the services effectively implementing the functionalities it needs, in the current context (i.e., a specific city). For example, only for users planning journeys starting from Trento, it makes sense to provide them the functionalities of the ViaggiaTrento app. Also fragments can be partially specified. In this way, their execution relies also on fragments provided by other domain objects, thus enabling a *chain of refinements* (as in Figure 2). The refinement is performed through the application of advanced techniques for dynamic and incremental service composition [3] based on AI planning. We refer to [6] for details on the execution of adaptive systems via dynamic interactions among domain objects.

3.2 Domain Object-Based Platform

The platform is organized in three main layers, as shown in Figure 1. The **Enablers** leverage on our previous results on the adaptive by-design wrapping of (mobility) services [5, 6]. Developers can exploit and wrap up as domain objects the available services in the mobility domain. Besides the design of mobility services, enablers allow also for their runtime operation, as we will see further on. The **Mobility Services** layer exposes the functionalities implemented or facilitated by the Enablers. These services can exploit and/or combine into value-added services the functionalities of the services previously wrapped up and made available by the Enablers (i.e., services for user profiling, planning, booking, monitoring of journeys, etc.). The key idea is that the platform

[8] https://telegram.org/

is open to continuous extensions with new services as domain objects. Their functionalities can thus be exploited in a transparent way to provide value-added services. All the platform mobility services can be eventually provided to final users through a range of multi-channels front-end applications that constitute the **Front-end** layer. These can be mobile or desktop applications, and they can be independent or rely on existing services (e.g., chat bots). The runtime operation of the services relies on different enablers. Domain objects processes are executed by the *Process Engine*. It manages service re-

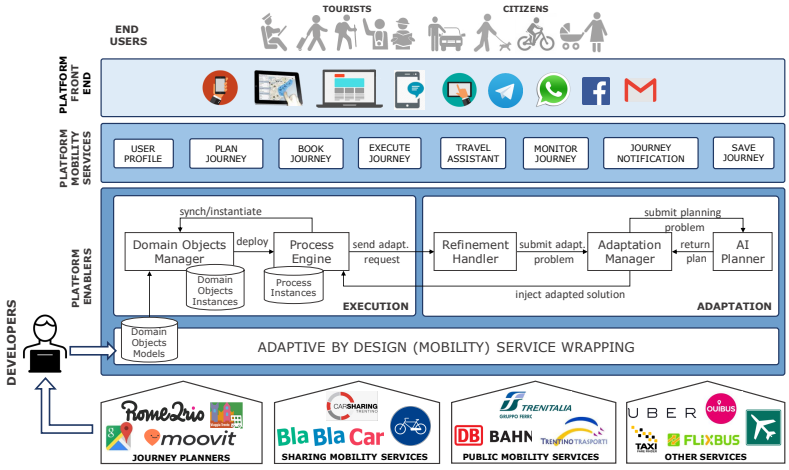


Fig. 1: Domain Object-based Platform.

quests among communicating processes and, when needed, it sends requests for domain objects instantiation to the *Domain Objects Manager*. In this way, correlations among processes are defined. During the normal execution, abstract activities can be met. They need to be refined with one or a composition of fragments modeling services functionalities. To this aim, the process engine sends a request for abstract activity refinement to the *Refinement Handler* component that is in charge of defining the corresponding *adaptation problem*. It defines the *problem domain* by selecting the proper fragments driven by the abstract activity's goal. The adaptation problem is submitted to the *Adaptation Manager* that translates it into a *planning problem* for the *AI Planner* component, which will send back a plan that can be injected into the abstract activity being refined.

3.3 Travel Assistant Implementation

In this Section, with the platform in mind, we detail ATLAS travel assistant [9], and how applications can be realized on top of our platform. To realize a world-wide travel assistant able to provide to the users the proper mobility services in the specific context(s) of their journeys, we selected real-world mobility services exposed as open APIs. We identified their behavior and functionalities and their input and output data. Finally, we wrapped them up as domain objects to be stored in the platform knowledge base. For instance, we wrapped *Rome2Rio* and *Google Transit* as global journey planners and *ViaggiaTrento* as local planner, for the city of Trento. Combining the geographical coverage of global planners with the accuracy of local planners is a concrete example

[9] ATLAS is available here: https://github.com/das-fbk/ATLAS-Personalized-Travel-Assistant.

of services interoperability promoted by our platform. Other examples are *Travel for London* [10] as local planner, *BlaBlaCar* as ridesharing service, *CityBikes* as bike sharing services applying to about 400 cities, *Trentino Trasporti* [11] for the public transportation in the Trentino region. Being defined as domain objects, all these services can now be executed, automatically composed and adapted by the Enablers of the platform.

At the Mobility Services platform level, instead, we can find the *Travel Assistant* defined as a value-added service leveraging on the services available in the system. Its main features are the following: (i) given a user planning request, it is able to decide between a *local* or a *global* planning solution; (ii) given the planners responses, it defines the better way to show this responses to the user (e.g., a list of travel alternatives, a message); (iii) given the user selection, the travel assistant is able to identify the transport means in the legs making the entire solution. In this way, it can incrementally provide to the user specific functionalities and context-aware information for her journey. We emphasize here that the more (mobility) services are wrapped up and stored in the system's knowledge base, the more responsive and accurate the travel assistant will be. Finally, among the multi-channel front-ends that can be defined on top of the platform, we realized ATLAS as a Telegram chat-bot, exploiting the Telegram's open API.

Executing ATLAS. In Figure 2, we report examples of *chains of incremental refinements*, from the execution of the scenario in Section 2. The execution starts from the core process of ATLAS, modeling the chat-bot started by Sara. We focus on the refinement of the `Plan Journey` abstract activity, whose goal consists in finding a travel plan. The refinement generates the following steps.

Step 1. The fragment `PlanJourney` of the Travel Assistant is selected and injected in the process of ATLAS. It allows Sara to insert the source and destination locations and to send a journey plan request. The activities `Plan Request` and `Plan Response` of this fragment model the communication between it and its core process, where the request is handled. In our scenario, being the destination Vienna, the Travel Assistant will go for a global plan, by executing a fragment from the Rome2Rio domain object.

Step 2. To properly show the travel alternatives to the user, an appropriate data visualization pattern must be selected, based on the data format (e.g., a list, a message). This is defined at runtime, by the Data Viewer domain object providing the `DefineDataViewerPattern` fragment for this purpose. Thus, Sara receives the list of the found travel alternatives satisfying her requirements.

Step 3. Sara selects her preferred alternative (suppose the first one, a multi-modal solution made by a train and a bus travels). Based on her choice, the `Define Journey Legs` abstract activity is refined with the `HandleJourneyLegs` fragment, which defines the goal for the `Specialize Journey` abstract activity, whose refinement allows the Travel Assistant to find the proper fragments for each journey leg.

Step 4. The last step shows a *composition* of fragments provided by the transport companies involved in the legs of the user selection (e.g., Sudtirol Alto Adige and Hello). Their execution provides to Sara the proper solutions, from the two companies.

In conclusion, this execution example exhibits the *bottom-up* nature of the approach, from grounding services till the user process. This happens in a completely transparent way for the user that interacts with only one application, ATLAS.

[10] https://api.tfl.gov.uk/ [11] http://www.ttesercizio.it/

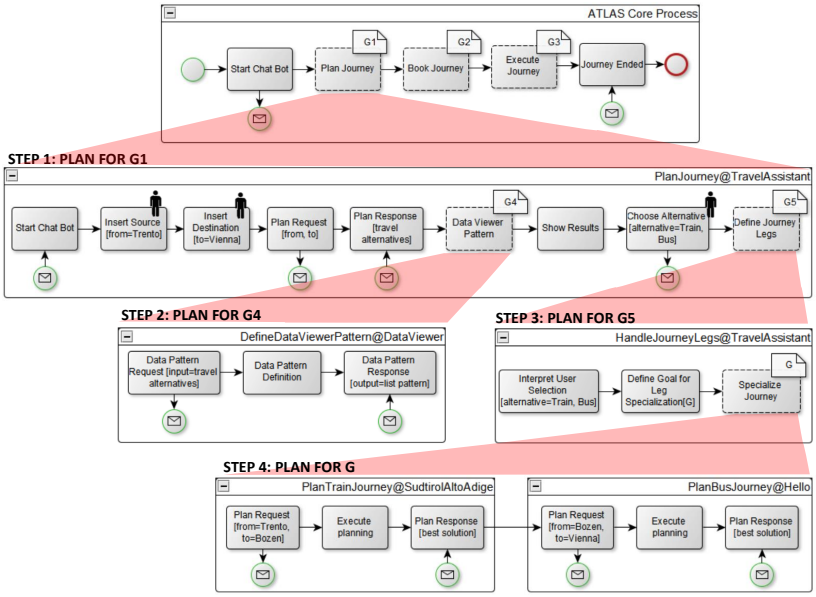


Fig. 2: ATLAS: an example of the system execution via incremental and dynamic refinements. For each fragment, we specify its name and the domain object which it belongs to.

4 Evaluation

To evaluate the *effectiveness* and *efficiency* of our platform, we have run a set of experiments based on real-world problems [12]. We ran ATLAS using a dual-core CPU running at 2.7GHz, with 8Gb memory. To show its *feasibility*, we evaluate: **(1)** How long it takes to wrap up real services as domain objects; **(2)** How much automatic refinement (service selection and composition) affects the execution of ATLAS. Based on our experience, we can argue that to wrap a real service as a domain object, the developer needs (i) to master the domain objects modeling notation and (ii) to understand the service behavior, its functionalities, its input/output data format and how to query it. Wrapping time clearly changes between experienced and non-expert developers. From our analysis, it ranges from 4 to 6 hours, considering average complex services. Moreover, it is also relevant to claim that this activity is done *una tantum*: after wrapping, the service is seamlessly part of the platform. To evaluate the automatic refinement, we collected both the adaptation and mobility services execution statistics, to understand how long they take, on average, to be executed. We carried out an experiment considering 10 runs of ATLAS handling various end-users' requests. For each run, more than 150 refinement cases were generated. As shown in Figure 3, the majority of the problems have a complexity in-between 0 and 19 transitions, while the most complex problems range from 80 to 100 transitions. Notice that the occurrence of complex problems is relatively rare. For all the runs, only 3% of the problems require more than 0.5 secs to

[12] The specification of ATLAS used for the evaluation contains 14 domain object models, 17 fragment models and 12 types of domain properties Domain properties are high-level representations of the domain concepts, and they are used to evaluate the conditions under which each fragment can be exploited (for details refer to [5, 6]).

be solved, and the worst case is anyhow below 1.5 secs. To measure how much automatic refinement influences the execution of ATLAS, we compared the data about the time required for adaptation with the response time of real-world services wrapped in ATLAS. As expected, Figure 4 shows that problems with the most complex planning domain take more planning time than problem with less complexity. In the worst case,

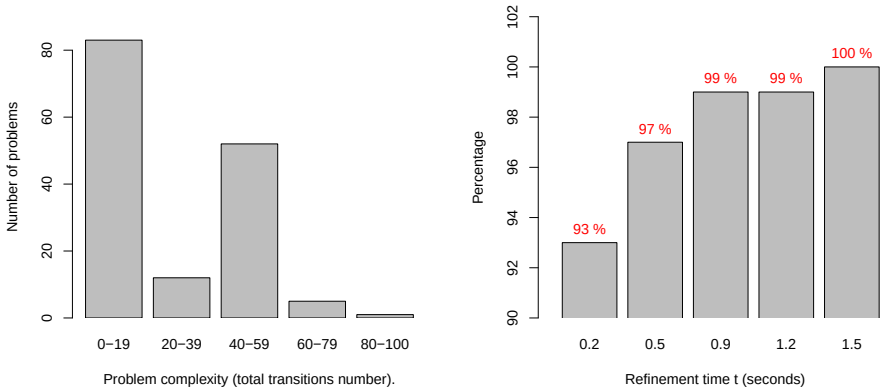
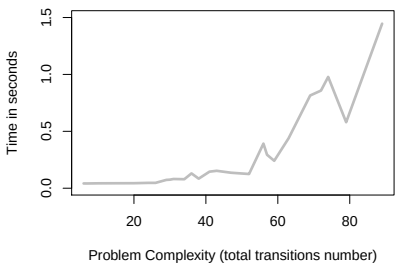


Fig. 3: (Left) **Distribution of Problems Complexity:** it shows the distribution of problem complexity of adaptation problems calculated as the *total amount of transitions* in the state transition systems representations of the domain properties and fragments present in each problem.
(Right) **Percentage of refinement problems solved within time t.**



Services Response Time	
Service Name	**Avg. Response Time**
Bla Bla Car	0.78 secs
City Bikes	0.23 secs
Google Transit	0.65 secs
Rome 2Rio	1.20 secs
Travel for London	3.20 secs
Viaggia Trento	0.77 secs

Fig. 4: (Left) **Trend of the Adaptation Time:** it relates the (average) time required to solve a composition problem to the problem complexity. It is computed considering in the 10 runs all the refinement problems having the same complexity.
(Right) **Services Response Time:** it refers to (a subset of) ATLAS real mobility services.

the adaptation requires a time close to 1.5 secs, while the services response time ranges from 0.23 to 3.20 secs. Moreover, the adaptation takes more time for the most complex problems that are the less frequent to occur. We argue that the automatic refinement responsiveness is equivalent to that of mobility services.

5 Related Work and Conclusion

Open services are easy to understand and to access services and can be exploited to develop applications or new value-added services. Web APIs are the most common way to specify them. To overcome the limitations of semantic web services (i.e., the use of non-standard languages for description) a model for Linked Open Services has been

introduced in [7] in which services are viewed as RDF "prosumers". With the rise in popularity of web APIs, platforms for their management and customization, called *API management platform*, have been provided. However, while advances in web services and their composition enable automation and reuse, new challenges have emerged in the case of APIs. The service developer requires sound understanding of the different service types, access-methods, and input/output data formats [8] (e.g., XML, JSON, SOAP, HTTP). ServiceBase [2] proposes a Unified Services Representation Model where common service-related low-level logic can be abstracted and reused by other applications developer. With it a set of APIs have been implemented that expose a common and high-level interface for integrating heterogeneous services in a simplified manner. Organizations like Mashery [13] and Apigee [14] are building on these trends to provide platforms for the management of APIs. For instance, ProgrammableWeb [15] now has more than 10,000 API in its directory. However, despite advances in SOA, complete solutions for open services management are yet required. There is still a need to make services easy to understand and to access. Our service delivery platform is an attempt to solve the previous open issues and to provide a complete solution for open services management and exploitation. The core idea is to factorize the capabilities offered by service providers as a set of building blocks (i.e., domain-objects), which can be easily combined to give place to composite services that can be published and exploited.

In conclusion, we have presented a service delivery platform providing engineering methods and techniques to design and release adaptive service-based applications. We have shown how applications can be realized on top of it exploiting the functionalities provided by real-world services. As stated, our platform requires that services are previously wrapped as domain-objects to be used. Although this may seem a limitation, we can argue that the service wrapping activity can be performed as a collective co-development process, in a crowd-sourcing style. Furthermore, *open data* can help to overcome the limitations imposed by services that are not open. Extensions of our platform refers to the inclusion of functionalities provided by smart things, in the IoT sense, and the support for other forms of run-time adaptation.

References

[1] DeMOCAS: Domain objects for service-based collective adaptive systems. https://github.com/das-fbk/DeMOCAS/.

[2] M. Chai Barukh and B. Benatallah. Servicebase: A programming knowledge-base for service oriented development. In *Database Systems for Advanced Applications, 18th International Conference, DASFAA 2013. Proceedings, Part II*, pages 123–138, 2013.

[3] A. Bucchiarone, A. Marconi, C. A. Mezzina, M. Pistore, and H. Raik. On-the-fly adaptation of dynamic service-based systems: Incrementality, reduction and reuse. In *Service-Oriented Computing - 11th International Conference, ICSOC 2013*, pages 146–161, 2013.

[4] A. Bucchiarone, A. Marconi, M. Pistore, and H. Raik. A context-aware framework for dynamic composition of process fragments in the internet of services. *Journal of Internet Services and Applications*, 8(1):6, 2017.

[5] A. Bucchiarone, M. De Sanctis, A. Marconi, M. Pistore, and P. Traverso. Design for adaptation of distributed service-based systems. In *Service-Oriented Computing - 13th International Conference, ICSOC 2015, Proceedings*, pages 383–393, 2015.

[6] A. Bucchiarone, M. De Sanctis, A. Marconi, M. Pistore, and P. Traverso. Incremental composition for adaptive by-design service based systems. In *IEEE 23rd International Conference on Web Services*, 2016.

[7] R. Krummenacher, B. Norton, and A. Marte. Towards linked open services and processes. In *Proceedings of the Third Future Internet Conference on Future Internet*, FIS'10, pages 68–77. Springer-Verlag, 2010.

[8] M. Nesa Lucky, M. Cremaschi, B. Lodigiani, A. Menolascina, and F. De Paoli. Enriching API descriptions by adding API profiles through semantic annotation. In *Service-Oriented Computing - 14th International Conference, ICSOC 2016, Proceedings*, pages 780–794.

[9] H. Raik, A. Bucchiarone, N. Khurshid, A. Marconi, and M. Pistore. Astro-captevo: Dynamic context-aware adaptation for service-based systems. In *Eighth IEEE World Congress on Services, SERVICES 2012*, pages 385–392.

[13] http://www.mashery.com [14] http://apigee.com [15] http://www.programmableweb.com