Preprint: Bühring P., Kuryazov D., Sandau A., Winter A. (2021) A Sustainable Software Architecture for Mobility Platforms. In: Marx Gómez J. et al. (eds) Progress in Sustainable Mobility Research. Progress in IS. Springer, pp 115-136, https://doi.org/10.1007/978-3-030-70841-2_7

A Sustainable Software Architecture for Mobility Platforms

Phillip Bühring, Dilshodbek Kuryazov, Alexander Sandau, Andreas Winter

Abstract: The NEMo project aims at providing new mobility means to facilitate *sustainable fulfillment of mobility needs in rural areas*. The collection of mobility services provided by NEMo serve to develop mobility platforms fulfilling inter-modal needs. In order to meet the core objectives of NEMo, the envisioned mobility platforms must be sustainable itself by providing flexibility in co-evolution with changing and novel mobility needs, services, and business models. With the overall objective of NEMo being *sustainable*, it is only appropriate to strive for it in terms of sustainable software design and architecture. Thus, in order to be technically sustainable, the architectural provision behind a mobility platform has to be flexible and adaptable.
SENSEI and DORI provide architectural support to enable sustainability on software engineering level. They are applied to NEMo in order to achieve the aforementioned goals to create, extend and adapt a NEMo inter-modal mobility scenario. SENSEI provides flexibility, adaptability and long-term sustainability by utilizing model-driven, service-oriented and component-based concepts to provide flexible orchestration of NEMo's functionality. The users of a mobility platform further need support for interacting with the mobility platform. To this end, the DORI approach is applied to design user interactions in NEMo. DORI intends to provide flexible interaction modeling support for designing state-based interactivity models to describe the overall interaction by GUI states and transitions between these states. It defines abstract GUI widgets and their underlying implementations, separately.
This paper summarizes the sustainable architecture of NEMo and shows the extensibility and adaptability of the NEMo mobility platform.

Andreas Winter, Phillip Bühring, Alexander Sandau, University of Oldenburg,
e-mail: winter@se.uol.de {phillip.buehring,alexander.sandau}uol.de
Dilshodbek Kuryazov, Urgench branch of Tashkent University of Information Technologies named after Muhammad al-Khwarizmi, e-mail: kuryazov@se.uol.de

1 Motivation

The interdisciplinary research project NEMo (Jelschen et al., 2016) aims at the sustainable fulfillment of mobility needs in rural areas considering social, demographic, accessibility, legal, economic, and ecological conditions and objectives. It intends to facilitate the provision of *smart mobility services* based on social self-organization. NEMo develops novel business models (Akyol et al., 2017) that increase utilization of public and private transport, while reducing the overall stream of vehicles on the streets (Kuryazov et al., 2019). In smarter cities, which also include rural areas, information and communication technologies (ICT) are viewed as the key enabler to support these objectives implementing a mobility platform that is accessible through various devices and media. This paper strictly focuses on designing a flexible and adaptable, i.e., a *sustainable software architecture* to provide appropriate ICT support.

Like any software system, the NEMo mobility platform needs to evolve to remain up to date with new or modified requirements, e.g., new business models, mobility services and their implementations. Continuously adapting the mobility platform leads to more complex and less maintainable software systems (Lehman 1996). Due to the innovative nature of the NEMo mobility project, a sustainable software architecture plays an essential role in simple and fast development, integration and maintenance of new mobility services (Jelschen et al. 2016). Even during the course of the project, developing the NEMo system required various revisions and adaptations.

The application domain of smart mobility services also requires highly flexible software support. The NEMo mobility platform should be able to support all kinds of mobility needs and scenarios, modes of transportation, and business models. According to (Combemale et al. 2016), the evolution of NEMo has to facilitate the recombination of existing mobility services to provide enhanced services, as well as completely new, unanticipated usage scenarios.

Finally, with the overall objective of NEMo aiming at sustainability, it is only appropriate to strive for it in terms of software design. A rigid, monolithic software system would lead to high maintenance costs, and ultimately to its phaseout, close down, and forced replacement (Rajlich and Bennett 2000). To be sustainable, the NEMo mobility platform must make architectural provisions for sustainability, flexibility and adaptability (Kateule and Winter 2018). In this way, a smart system can be continuously adapted and made smarter and smarter.

A major use case, that is expected to play an essential role for the NEMo mobility platform, is inter-modal routing, combining the different modes of transportation, e.g., walk, bike, bus, train, carpooling, etc. The existing infrastructure and functionality of the ICT Platform and Services (ICTS) project (Wagner vom Berg 2015) is already able to support the use case of inter-modal routing to a large extent. On the top of this platform, the ICT Services project build another software system to combine its basic software services and offer value-added services to support the designated business processes. These mobility services are reused in developing the sustainable prototypical NEMo mobility platform in Section 3.

1.1 Challenges

Due to the fast development of early prototypes of ICTS to enable applied research on rural mobility at early stages, the existing infrastructure was designed and developed without a particular focus on flexibility, sustainability and adaptability. In order to achieve sustainable development and maintenance of the mobility platform, the NEMO project addresses to several challenges considering sustainability, innovation and evolution:

- The software architecture of the mobility platform has to be developed focusing on flexibility, adaptability, extensibility and long-term sustainability.
- The existing functionality of the existing mobility platforms has to be easily reused, enhanced and modified.
- The realization of the mobility platform has to focus on consistent separation of functionality (services) and implementation (components), following the principle of *separation of concerns* (Dijkstra, 1982).
- Development of novel, flexible interaction user interfaces has to be automated so that changes and new user requirements can easily be adapted in user interactions of the mobility platforms.

These challenges are considered as the main engineering and technical-conceptual challenges that can be resolved by the novel software engineering trends, which are addressed in this contribution.

1.2 Objectives

In order to solve the scientific challenges described in Section 1.1, this section describes several objectives that are addressed throughout this paper. The sustainability objectives of the NEMO project, from a software engineering point of view, are manifold:

- *Sustainable Software Architecture.* First of all, there is a need for a flexible, adaptable and extensible software architecture that incorporates the existing functionality, but highlights flexibility, adaptability, and long-term sustainability. A sustainable software architecture serves as a common blueprint in reusing existing features of mobility platforms and developing and adapting new features.
- *Reusable Mobility Services.* In case of existing mobility platforms, the existing functionality of the mobility platforms should be enhanced, modified and reused. Future changes (i.e., extensions, optimization and corrections) based on the research findings within the NEMO project have to be adaptable and reusable.
- *Separation of Concerns.* Sustainable software architecture and reusable mobility services should support novel business models, model-driven and service-oriented mobility services and component-based functionality enabling consistent separation of functionality (services) and implementation (components), This allows for eased maintenance of business models, mobility services and components.

- *Interaction Modeling.* As long as users interact with mobility platforms by using various user interfaces and media devices, there is a need for a model-driven, flexible interaction modeling feature by separation of user interaction and user interface designs. Utilization of interaction modeling provides developing user interaction independent from various platforms and devices.

These objectives remain on the central focus throughout this contribution. So far, some prototypical results are achieved by providing the existing mobility platform with the flexible, sustainable and adaptable software support based on component-based, service-oriented software development and maintenance. This feature supports including the achieved research results in an interdisciplinary research project lite NEMO to its software support from a software engineering's perspective.

The remainder of this paper is outlined as follows: Section 2 introduces a reference architecture for sustainable mobility platform development. The same section describes the central concepts of the SENSEI and DORI approaches. Section 3 sketches the application of SENSEI and DORI approaches to the NEMO mobility platform based on the reference architecture using an example. Section 4 defines several user requirements that have to be adapted in the running NEMO example throughout this paper. Section 5 presents the complete solution combining all requirements defined in Section 4. Section 6 explains the core results and contributions of this research and draws conclusions about sustainable architecture and SENSEI in NEMO.

2 Conceptual Idea

This section sketches the theoretical foundations of the engineering technologies used in development of the NEMO mobility platform. These foundations consist of a sustainable reference architecture (explained in Section 2.1), an introduction of the SENSEI (Section 2.2) and DORI approach (Section 2.3). A sustainable reference architecture helps to develop an evolvable mobility platform for providing mobility services.

2.1 Sustainable Reference Architecture for Mobility Platforms

One of the main objectives in NEMO is the development and application of a sustainable reference software architecture. Figure 2.1 depicts the four-layer NEMO taxonomy (Akyol et al., 2017). This taxonomy serves as a common blueprint and foundation in developing the sustainable NEMO mobility platform. The taxonomy provides clear separation of concerns by distinguishing between the *mobility services*, *business models*, *information-technology-(IT)-services* and *IT-components*.

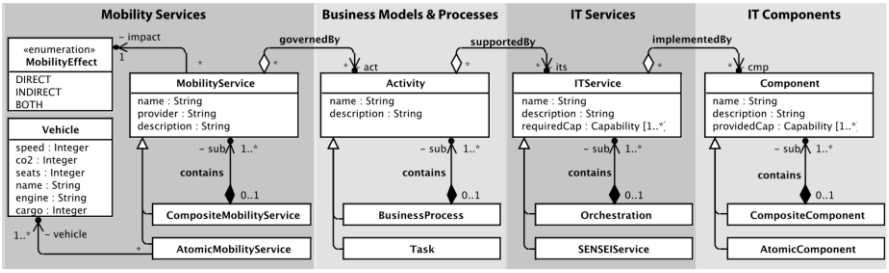

Figure 2.1. Four Layer NEMo-Taxonomy for Mobility Platform (Akyol et al., 2017)

As the NEMo taxonomy depicted in Figure 2.1 enables separation of concerns, mobility platforms developed based on this taxonomy can achieve higher level of sustainability, adaptability and flexibility. Each level of the taxonomy can be sustained separately, independent from the rest. It allows for eased adaptation of changes in user requirements, business models, IT-services and IT-components.

Mobility Services. In general, the mobility platform offers mobility services by means of transportation (vehicle) and offered by providers. A mobility service can also be comprised of more fine-grained mobility services which is referred to as a composite mobility service. This is due to the inter-modal nature of mobility services. Any mobility service can be performed directly by transporting people, indirectly by transporting things, or both by transporting people and things at once. Each mobility service may be associated to several *business models* and *processes*.

Business Models and Processes. In the second column, the taxonomy describes business models and processes that might be related to many mobility services. The business models and processes consist of activities performed by the users and providers of the mobility platform. The activities can further be defined as tasks that the users should perform before, while and/or after using the mobility service. The activities are supported by *IT-services*.

IT-Services. An IT-service defines a piece of functionality. It adds appropriate functionality to the activities focusing on human behavior (Jelschen, 2015). An IT-service is a description of what a software component should do. In case of the mobility platform, each mobility service is provided by several IT-services, e.g., each inter-model mobility service combines several IT-services supporting the transportation mode. In the same vein, each IT-service provides a functionality, e.g. finding the nearest stations to an origin or destination, finding sub-routes with different transport modes, etc. These IT-services are usually implemented by *IT-components*.

IT-Components. IT-components are the concrete implementations of the functionality defined by IT- services (Jelschen, 2015). A service can be implemented by several combined components. For instance, a find route service might use several components for each transportation means, e.g., bus, walk, etc.

Figure 2.2 depicts a sustainable reference architecture based on the NEMo Taxonomy in Figure 2.1. In this architecture, the mobility services on the top level are described by business models and processes in the second level. These business models are then defined as IT-services, whereas each

activity (i.e., functionality) is defined as one abstract service in a service catalog. In the same vein, these services are implemented by IT-components enabling reuse and sustainability of IT-services and IT-components.

Each level of the architecture in Figure 2.2 can be sustained separately, independent from the rest of the taxonomy. It allows for eased adaptation of changes in user requirements, business models, IT-services and IT-components.

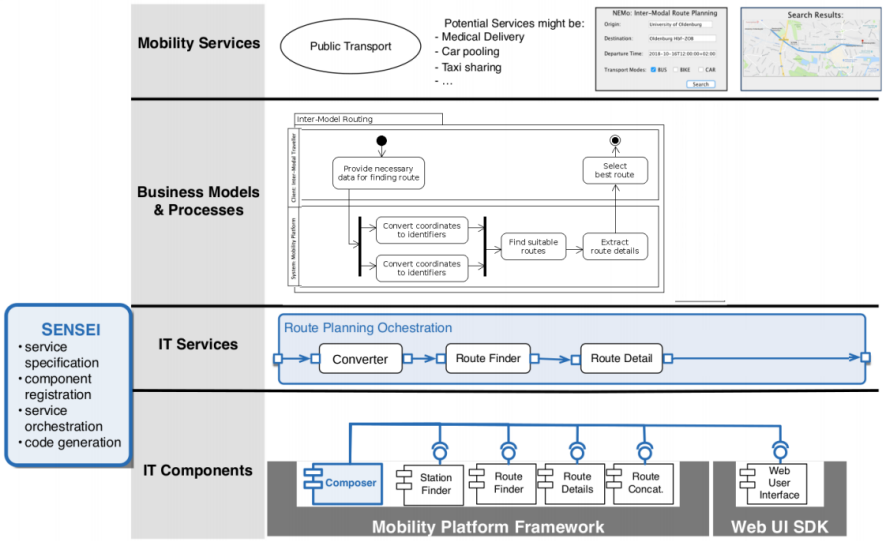

Figure 2.2. Sustainable Reference Architecture for Mobility Platforms (Jelschen, 2015)

This sustainable reference architecture in Figure 2.2. is utilized in Section 3.1 to develop a concrete mobility platform. As there is a need for technical support for realizing this reference architecture, the SENSEI approach explained in Section 2.2 is utilized as realization technology.

2.2 SENSEI Approach

The sustainable reference architecture explained in Section 2.1 serves as a common blueprint for model-driven (Kleppe et al., 2003), service-oriented and component-based development and maintenance of the mobility platforms (Breivold and Larsson, 2007). This section explains the SENSEI approach making the sustainable reference architecture as the central architectural provisions for developing model-driven, service-oriented and component-based mobility platform. Eventually, the SENSEI approach serves as main technical grounds for realizing sustainable reference architecture depicted in Figure 2.2.

SENSEI (Software EvolutioN SErvice Integration) (Jelschen 2015, Jelschen 2020) provides service-oriented software design facilities (service orchestration) on an abstraction level close to the application domain of mobility needs. Strictly separated from this layer, concrete implementations of these services are realized in component-based terms. An automated mapping from services to components bridges the gap between service-oriented specification and component-based realization.

SENSEI provides a toolchain-building support framework providing flexibility, reusability, and productivity. It combines service-oriented, component-based, and model-driven techniques to automatically map high-level, process models (*service orchestrations*) onto reusable and interoperable components possessing the required capabilities and generate code that combines and coordinates them in the required manner (Jelschen 2015). Using the SENSEI framework, applications are built sustainably by combining (and reusing) components providing clearly specified functionality (services). For this purpose, services are kept in a *service catalog*. A *component registry* maps this functionality to potential implementing components (*Service-Component-Matching*). Based on *orchestrations of the services*, suitable components are automatically linked to the desired application by the *SENSEI generator*. SENSEI consists of the following core concepts which can easily be mapped to the four levels of the reference architecture in Figure 2.2.

Service Catalog. A service catalog serves as a central repository containing service definitions that are described in a standardized way. The mobility services are defined in the SENSEI service catalog. All services defined in the service catalog have names and descriptions, along with input and output parameters, and associated data types. The implementations (*IT-components* in the reference architecture) of these services usually provide traveling information for various imaginable modes of transport, and the users of the service might only need a subset of them. At the same time, the service catalog would become extremely cluttered if a service were to be defined for every possible variant. SENSEI solves this issue by introducing *capabilities* to describe service variants concisely. Services define capability classes to represent aspects that can vary independently. This service catalog is utilized to collect mobility services in the NEMo project.

Component Registry. The component registry establishes relations between services defined in the service catalog, and *IT-components* that provide the functionality. Services are implemented by components offered by various providers. Each entry in the component registry refers to a component and lists one or more services it implements. With each service, the provided capabilities are specified in the same way as required capabilities for *orchestrations*. Existing mobility services, provided by the NEMo mobility platform, are wrapped for SENSEI compatibility and interpreted as implementations behind mobility services in orchestrations.

Service-Component-Matching. Considering *required* capabilities of orchestrated services, *provided* capabilities of registered components, as well as constraints resulting, e.g., from data type compatibility concerns, a suitable composition of components will be searched for to realize the orchestrations.

Service Orchestrations. Service orchestrations (e.g., depicted in Figure 3.3 and Figure 5.3) allow to instantiate and combine abstract services from the service

catalog to create more complex functionality, using a process-oriented, graphical modeling language. In order to fulfill business models and processes (in the reference architecture), IT-services are orchestrated using one or many IT-services from the service catalog. For instance, different, single IT-services provide different routing services by different transport modes and capabilities, whereas they are orchestrated to provide the complete inter-modal routing scenarios for developing a mobility platform in the NEMo project.

Application Generation. To conclude the process, the SENSEI orchestration model is mapped to a particular target platform providing a runtime environment, e.g. WSO2, the middleware used by the ICT Platform project. This step is performed by a model-driven code generator. It results in a fully auto generated component that depends on the components found in the previous step to perform the work specified by the orchestrated services. The result is an executable software application, ready to be deployed, e.g., to the WSO2 application server. After all, the deployed application can be called by different means of media such as mobile, web, desktop and other frontend clients.

Based on the sustainable reference architecture depicted in Figure 2.2, the SENSEI approach is applied to the concrete NEMo mobility applications in Section 3.

2.3 DORI Approach

The mobility platform is bound to offer a wider range of functionality (e.g. route planning, carpooling, etc.), and is supposed to be used in an everyday context by people with limited technical proficiency, which creates a need for *user interaction*. The SENSEI orchestrations are based on the input-process-output principle, and thus not capable of interacting with the user at runtime. Thus, there is a need for another, complementary approach to enable users to interact with the mobility platform and utilize its services in their daily routing planning activities.

In order to provide such flexible, model-driven means for the development of *user interactions* with the mobility platform, the students' project group DORI (Do Your Own Reactive Interface) has developed a concept and tool support for modeling user interactions through the use of the DORI Domain Specific Language, which was based on UML state charts and the IFML (Interaction Flow Modeling Language) (Brambilla and Fraternali 2014). *Interaction diagrams* representing the interactive behavior of applications, are executed by a dedicated interpreter. Following the SENSEI structure given in Section 2.2, abstract behavior is represented in the *abstract widget catalog*. Its *platform*-specific implementation is provided by the concrete widget *catalog.*

Abstract Widget Catalog. This contains abstract Widgets and Functions. Abstract Widgets serve as "blueprints" for the actual interaction states (called Abstract Widget Instances - AWI), defining the set of data fields in their possession, as well as the events which they may react to. Conceptually, they are similar to SENSEI services. They may also contain Sockets, enabling composition or parallel

processing, albeit only on instance level. The second kind of abstract elements, Abstract Functions, define a return type and a set of parameters.

Widget Catalog. The concrete catalog consists of a list of concrete widgets for each of the pre-defined abstract widgets. While abstract widgets merely describe a signature, their concrete counterparts contain implementation-dependent information. Concrete Widgets contain a path to the implementation of the UI element which is supposed to represent it and Concrete Functions a path to the implementation of their logic (e.g., a REST endpoint). Their roles are similar to those of SENSEI components.

Platform Catalog. This catalog contains a list of platforms. In the context of DORI, a platform is a set of elements whose implementations target a common platform. This allows interpreters targeting different platforms to choose the appropriate concrete implementations from the catalog before executing a model. So far, there are two interpreters available; one based on Java Server Faces to enable desktop applications and the other one supporting the android platform.

Interaction Diagram. The flow of user interactions through states (AWIs) and transitions is described by interaction diagrams. The states are instantiated from the predefined abstract widgets; transitions may be supplemented with guards and Parameter Binding Groups (see Figure 3.2). The Parameter Binding Groups (1) define which events may trigger their host-transition, (2) which (abstract) functions are to be called once its host-transition fires, and (3) how data is transferred between the involved variables. Since the interaction diagrams themselves are based on abstract elements, they may be realized on different platforms, which makes this approach quite useful for multi-platform or multi-device applications, as it would be the case for the NEMo Mobility Platform. DORI is applied to development of the NEMo application in Section 3.

3 NEMo application

The sustainable reference architecture for the mobility platform in Section 2.2 is used as blueprint for model-driven, service-oriented and component-based development and maintenance of the NEMo mobility platform. As the proof of the concept, this section applies the SENSEI and DORI approaches to the NEMo mobility platform. This section depicts a simplified scenario of the mobility platform. Section 4 defines several additional user requirements (i.e., change requests) raising a need for changing, adapting and sustaining the existing mobility platform. According to these user requirements, Section 5 describes what changes and adaptations are required in the user interface, orchestration and interaction model in order to adapt these requirements to the mobility platform. It is shown that using the SENSEI/DORI-approach results in a technical sustainable software evolution.

The inter-modal routing finds routes to connect a point of origin and a destination (*Brake* and *Vechta* in this example). Combining different modes of transport, e.g., walking, riding a bike, taking a bus or train, driving a private car, or joining a carpool, makes this inter-modal routing.

Figure 3.1. depicts the user interface of this simple inter-modal routing use case representing two different states of user interaction with the mobility platform. In the first window, origin and destination of a route, and time of departure are entered for searching possible routes between these places.

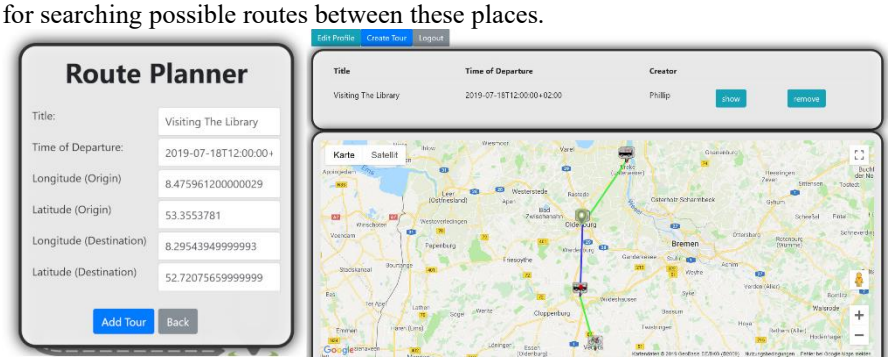

Figure 3.1. Mobility Service UI. mapdata ©2019 GeoBasis-DE/BKG (©2009), Google

Once the *Add Tour* button is clicked (left side of Figure 3.1), all possible routes, regardless of the transport mode, are calculated and added to the list of routes visible on the dashboard (right side of Figure 3.1). Upon selection of a route, it is drawn on the map below the list. In this example, the user interface is based on Java Server Faces, using a slightly modified dialect to access the data of the `DORI` interaction states from the .xhtml sources describing their corresponding UI counterparts.

Interaction Model

The `DORI`-Interaction Diagram behind this example makes use of seven abstract widget instanced (AWI) in total (see Figure 3.2), although only four of them are of immediate interest for the use case. The *Dashboard-AWI* (centermost in Figure 3.2) serves as a hub for the application. Depending on the incoming events it allows for switching to other widgets (e.g., *Login*, *Profile* and *RouteFinder*). It also grants access to a list of pre-calculated routes (or tours). A map is used to depict the selected route. For this purpose, both *RouteList*- and *RouteMap* widgets are nested within the *Dashboard* through the use of sockets. The *RouteList* widget possesses three data fields: a list of route-objects, an ID determining which of the routes is to be removed once the event *removeRoute* is received, and finally the navigation data of the route which supposed to be sent to the *RouteMap* widget for depiction once the user triggered the *showRoute* event. The *RouteMap* widget itself does nothing but contains a field to store the route-data received by the *RouteList* widget. Via the *gotoRouteFinder* event on the *Dashboard*, a user can switch to the *RouteFinder* widget, which is coupled to the route planning dialog as shown in Figure 3.1. The *RouteFinder* widget allows a user to define new routes to be added to the Dashboard's *RouteList*, and possesses one data field for each of the input fields visible in the form of Figure 3.1. The event *findRoute* starts the process meant to calculate routes, afterwards stores the calculated routes within the user's list, and finally returns to the *Dashboard*. This is done with the help of two subsequent transitions with a pseudo-state between them: the first transition calls the function meant to

calculate the new routes (navigational) data, ultimately referencing to a WSO2 service, which in turns executes the SENSEI orchestration (Figure 3.3) used in this example. Necessary parameters are prepared beforehand using *Parameter Bindings* (longitude, latitude, etc.). *Parameter Binding* is also used to store the resulting route-data within the intermediate pseudo-state.

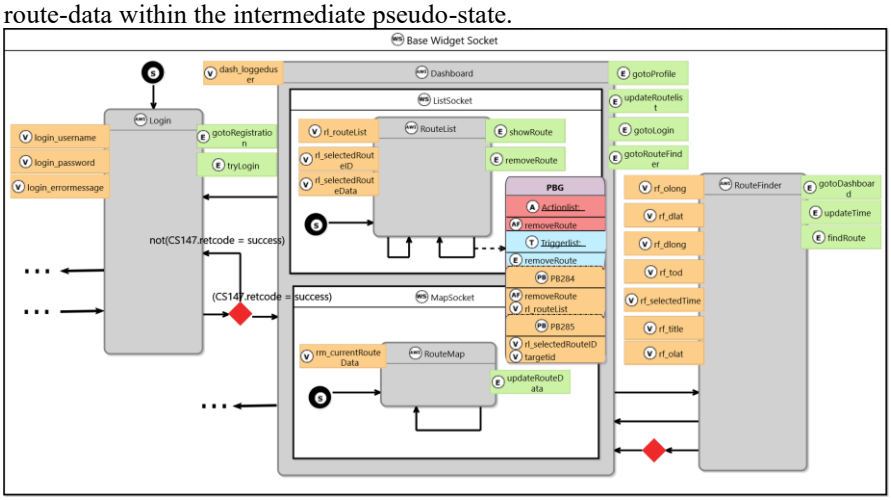

Figure 3.2. Simplified Interaction Model for Calculating Inter-modal Routes

IT-Services

According to the sample mobility scenario explained above, the following IT-services are needed:

- *Converter*: As shown in Figure 3.1, origin and destination locations are initially given by coordinates. But all locations and routes are processed by their identifiers within the already existing implementation. Thus, a converter service is needed to convert coordinates to identifiers of these locations.
- *Route Finder*: This service is utilized for finding all possible routes between origin and destination. It finds all routes by combining different transport modes.
- *Route Details*: In the existing ICT mobility platform, mobility services return the sub-set of route data, i.e., the identifiers of locations connecting subroutes. Detailed information including stops (i.e., coordinates, names, etc.) between the given two locations is then extracted by this additional service.

These three services are defined in the SENSEI service catalog. All services defined in the service catalog have names and descriptions, along with input and output parameters, and associated data types, also modeled in the catalog as data structures. These pre-defined services are implemented by IT-components already existing in the previous ICT-implementation (Wagner vom Berg et al., 2010). Within the generation step, these components are linked to manifest the SENSEI orchestration.

IT-Components

IT-components are the concrete implementations of the functionality defined by IT-services. As long as some mobility functionality is already developed as the outcome of the Electric Mobility Showcase program (Wagner vom Berg et al., 2010), these existing functionality of ICT are invoked as components in the context of NEMo. Existing mobility services provided by the ICTS are wrapped for SENSEI compatibility and interpreted as implementations behind mobility services in the SENSEI orchestrations. These components are published as REST Web services and made available to reuse in the framework of NEMo.

Various route planning algorithms are used in the NEMo project. For instance, new route planning services come with their own planer algorithms in the form of further IT-components. To include those, a service implementation can firstly be embedded behind its sole routing service, which has to be combined with a global one. This allows for early adaptation of the new mobility services in an unoptimized way. Later, the routing services can be implemented in a global routing component to provide an optimized routing in Section 5.

Service Orchestrations

In order to fulfill this particular mobility services, IT-services defined above are orchestrated selecting from the service catalog. Figure 3.3 depicts a SENSEI orchestration model for the initially simple mobility platform above. Services are instantiated from the catalog by selecting the required capabilities. In the orchestrations, the invocation order of services defined by the control flows (gray arrows) and the flow of data among these services is defined by the data flows (green arrows) connecting the inputs and outputs (green boxes) of the services. Services are marked with an encircled "S" ahead their names. The input parameters of the overall orchestration are defined by bigger green boxes, e.g., three boxes on the most left side of Figure 3.3 with names *origin*, *destination* and *tripRequest*, and one green box named *route* on the most right side.

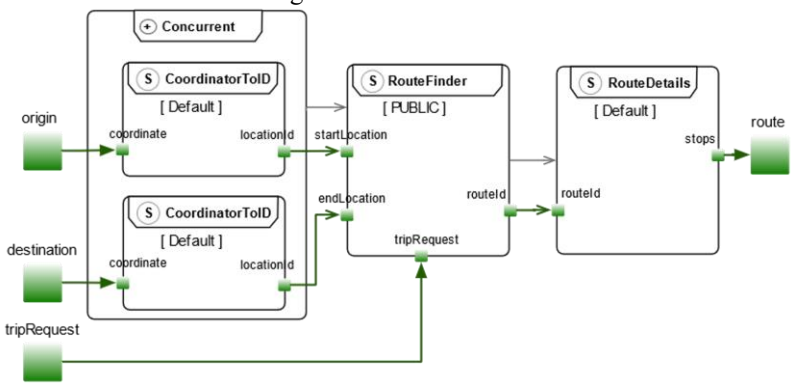

Figure 3.3. Service Orchestration before changes

The orchestration consists of applying two instances of the service *CoordinatorToID*. As the origin and destination locations of the searched route are given in the

form of the coordinates, these services convert the coordinates to identifiers. Then, the converted identifiers are sent to the *RouteFinder* service as *startLocation* and *endLocation*, whereas it also receives the third parameter *tripRequest.* The latter consists of the time of departure and the modes of transportation. In this case, the *RouteFinder* service provides the capability *PUBLIC*, meaning the service searches for all possible combination of transport modes including bus, train, walk, bike, etc. After finding an optimal route, the route finder service sends its identifier to the *RouteDetails* service, where further details about the found route are extracted. Eventually, the result of this orchestration is displayed in the second user interface depicted in Figure 3.1.

4 User Stories (Change Requests)

Section 3 has explained a simplified mobility service example. This mobility service is the subject to various changes such as extensions, improvements and optimization because of evolving user requirements over time. In this sense, the mobility platform must be sustainable, easy to adapt and flexible to meet new and changed user requirements. In order to demonstrate *sustainability of the reference architecture* and associated technical support explained in Section 2, this section introduces several user requirements (i.e., change requests) for the mobility platform explained in Section 3.

- **Text-based Location Information.** While route planning, the user wants to be able to give the names of the origin and destination locations instead of their geographic coordinates.
- **Points of Interest.** The users want to spend their spare time (waiting time between the changes of transport modes) meaningfully. For instance, if a traveler should change transport from train to train, from train to bus, or vice versa, there might be waiting time more than one hour. Then, travelers like to travel to *points of interest (PoI)*, i.e., coffee shops, restaurants, ATMs, museums, gardens, fast food chains, etc. Thus, they want to see recommended points of their interest on the map in the second UI of the mobility platform.
- **Biking.** The user wants to travel any subsection of a given route by bike if that subsection is less than five kilometers. For example, if there are subsections of the given route which is less than five kilometers and using bus, all of such subsections should be replaced by the *bike* transport mode. Suppose, in these cases rental bikes are available as a new mobility service. Travelers use these rental bikes instead of any other transport means.

These user requirements are defined as extensions, optimization and improvements for the simple mobility platform. They must be adapted in the existing mobility platform. The following subsections explain what to do in the user interface, service orchestration and interaction model of the mobility platform in order to extend the simple initial NEMo platform. The compete graphical descriptions for all adaptations are given in Section 5.

4.1 Text-based Location information

This section depicts what changes have to be made to change location coordinates to location names.

User Interface. On the first graphical user interface, four input fields (*Longitude (Origin)*, *Latitude (Origin)*, *Longitude (Destination*), *Latitude (Destination)*) are removed from the user interface. New input fields (*Origin* and *Destination*) are added in order to allow the user to specify origin and destination locations by their names instead of their geo-coordinates. The *RouteList* is also extended by the columns *Origin* and *Destination*. The changes made to the graphical user interface are reflected in Figure 5.1.

Interaction Model. Analogously, the *RouteFinder* widget's data fields for *longitude* and *latitude* are removed and two new ones, *origin* and *destination*, are added. The same changes are also reflected in the parameter sets of the abstract functions *calculateRoute* and *addRoute*, as well as in the parameter bindings. The last change is to actualize the (one) parameter binding used to initialize the data fields of *RouteFinder* with default values just before it becomes the active state by adding the origin and destination fields as targets. The changes related to the first user story are depicted in Figure 5.2.

Orchestration. In order to support textual location names, a new IT-service called *Geocoder* is entered to the service catalog and its implementation is registered in the component registry. In this case, the implementation of the *Geocoder* service invokes a ready-to-use geocoding service of Google Inc. This service provides conversion of location names to location coordinates. The rest of the process remains unchanged These changes are depicted in the left part of Figure 5.3.

4.2 Points of Interest

This section explains what changes have to be made on the UI, orchestration and interaction model in order to add support for bridging waiting time.

User Interface. NEMo supports bridging waiting time by finding the points of interest (e.g., ATMs, coffee shops, etc.), if a traveler has to wait more than one hour while changing the means of transportation. In order to adapt this user story, several types of point of interest are added to the graphical user interface, so that the user can choose where she/he wants to visit during that spare time. The changes made on the graphic user interface can be seen in Figure 5.1. Here, the user can tick different types of PoIs, he is interested in.

Interaction Model. A new data field stores the list of types of PoI which a user wants to spend spare time and thus to be included in the resulting route data. A new Parameter Binding prepares the parameter accordingly. Due to the types of data fields (List of Strings) and the current technical limitations of the `DORI`-Editor, a new Function *initPOIModes* is used to initialize the (empty) list when transitioning from the *Dashboard* to the *RouteFinder* widget. The changes related to the second user story are depicted in Figure 5.2.

Orchestration. The chosen types of PoI are then given to the overall orchestration as *poiModes*. To find the PoI locations, a new service called *PoIFinder* is added to

the service catalog. This service receives the route identifier and the PoI types as input and returns the PoI locations. This service is instantiated on the service orchestration to find point of interest locations based on the user request. This service can be invoked together with the *SubrouteExtractor* service in parallel which is explained below. This parallel invocation of services helps to improve the runtime performance of the overall orchestration. The changes in the service orchestration are displayed in Figure 5.3.

4.3 Biking

This section presents the changes made in the user interface, interaction model and service orchestration in order to fulfill adding biking as an new mobility service for short distances.

User Interface. *Bike* transport mode has to be used to travel sub-routes less than five kilometers. This does not request changing the graphical user interface and only requires adapting the orchestration.

Interaction Model. In the same vein neither the user interface nor the signature of the WSO2 endpoint are modified to adapt this request. No changes have to be made to the `DORI` interaction model, as well.

Orchestration. The orchestration has to be extended with several changes to add the additional mobility service. A new service called *SubrouteExtractor* extends the service catalog. This service receives a route identifier and extracts all sub-sections within that route which are less than five kilometers. These sub-sections are then processed to a loop as a map, where the route finder service is invoked for each with the *bike* transport mode. There, the route finder service has to be associated with the capability *BIKE*. In the same loop in the orchestration in Figure 5.3 the route details are also extracted for each sub-section. After finding all sub-routes that can be traveled by bike, these results are sent to the *RouteCombiner* service to combine all *bike sections*, *public sections* and *point of interest locations*. Finally, the result is assigned to the variable *route* and returned to the second graphical user interface (Figure 5.1) of the mobility scenario. These changes are depicted in Figure 5.3.

5 Complete Solution

This section presents the complete extension of the mobility platform explained in Section 3 combining all extensions depicted in Section 4.

User Interface after Changes. Figure 5.1. depicts the screenshot of the extended graphical user interface of the mobility scenario. The graphical user interface displays two states; the left window to search routes and the right window to show results.

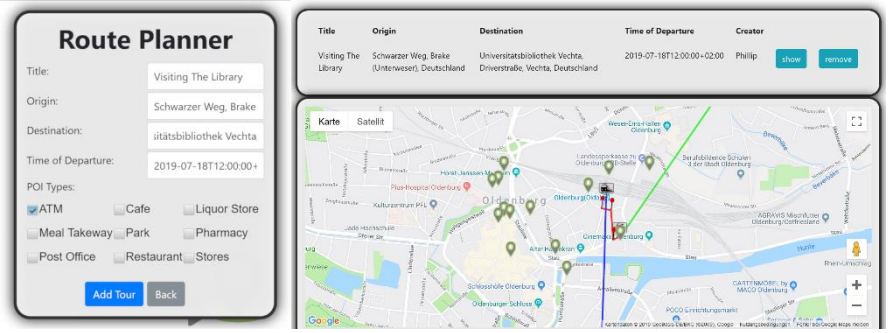

Figure 5.1. UI after Changes. Mapdata ©2019 GeoBasis-DE/BKG (© 2019), Google

The routes may be searched based on the names of their respective origin and destination location, which initially required input of their geocoordinates. Additionally, the route planner user interface provides a list of PoI Types in order to enable travelers to visit their favorite points of interest, if they are expecting longer waiting time. In the search results, the map depicts several indicators to show points of interest based on the PoI requests of travelers. Finally, there is no change in both graphic interfaces to provide biking for short distances.

Interaction Model. Figure 5.2 shows the relevant parts of the interaction diagram in its original state (left) and the final version (right) incorporating all the necessary changes to fulfill the three change requests.

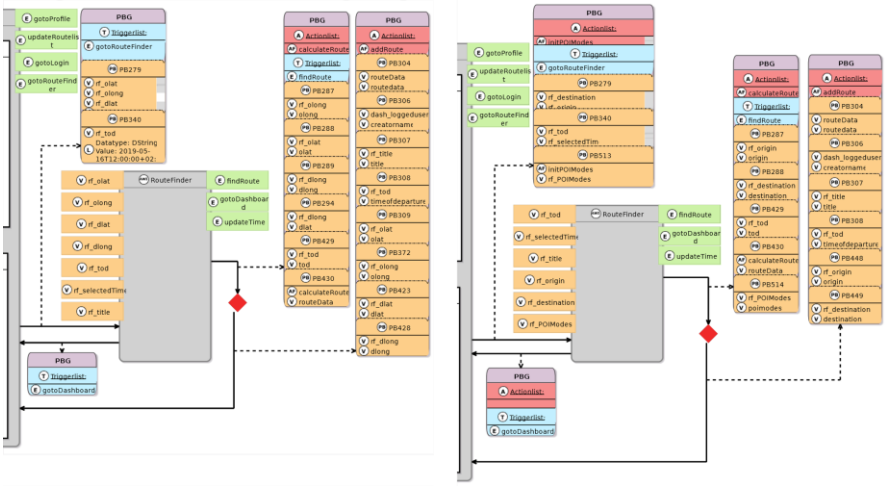

Figure 5.2. Interaction model before (left) and after (right) the changes

The changes are not very complex. Adding location names requires the deletion (or transformation) of some simple elements (four variables of the *RouteFinder*, the four parameters for each of the functions *calculateRoute* and *addRoute* and the eight bindings used to prepare their parameters) and subsequent addition of further elements (inserting the fields for origin and destination across widget, functions and bindings). Adding *PoIs* is handled by adding one function for initializing the list of

PoI types, as well as adding the list itself to the *RouteFinder* widget, the *calculateRoute* function (as a parameter) and finally a new Parameter Binding in order to transfer it from the widget to the function itself. Adding the bike mobility service does not require any changes to the interaction model.

IT-Services. To adapt the user stories, several new IT-services are added to the SENSEI service catalog. These services are: *Geocoder* which converts location names to location coordinates, *SubrouteExtractor* which finds sub-routes less than five kilometers, and extension of the *RouteFinder* service with the *BIKE* capability, and *PoIFinder* which finds the points of interest for the given types and on the given route. As long as different services deliver different outputs, these results are combined by the *RouteCombiner* service, eventually.

IT-Components. The implementations of these IT-services are provided by the IT-components registered in the SENSEI component registry. For providing the implementations of the newly defined IT-services, the component registry is also extended with respective components. A new component for the *Geocoder* service is added to the component registry, whereas it invokes the *geocode* service of Google API. A new component for the *SubrouteExtractor* service is locally implemented, and a new component implementing the existing *RouteFinder* service is extended which invokes the route finder service of ICTS with the *BIKE* transport mode. The latter requires to add a new capability to the *RouteFinder* service.

Service Orchestrations. All IT-services in the service catalog are defined as abstract services. These abstract services are then instantiated as concrete services whenever they are utilized in the orchestration model. Figure 5.3 shows the service orchestration model for the mobility scenario, which takes into account all change requests.

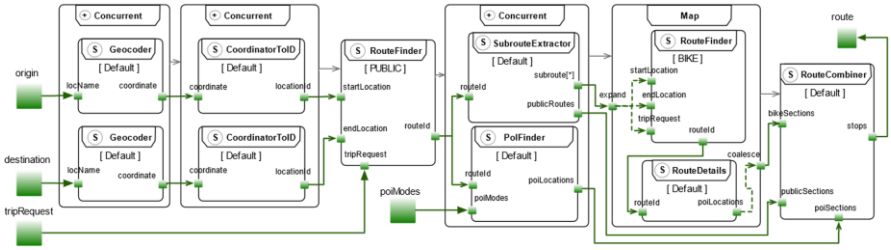

Figure 5.3. Service Orchestration after Changes

The orchestration is extended by the new services *Geocoder, SubrouteExtractor, PoIFinder, RouteCombiner* and *RouteFinder* with new capability to adapt the aforementioned change requests. The two instances of the G*eocoder* service, two instances of the *CoordinatorToID* service, *SubrouteFinder* and *PoIFinder* are placed in the *concurrency* container to accomplish higher runtime performance of the overall orchestration. As the *SubrouteExtractor* service instance returns the map of subroutes, this value is processed in the map to find the *bike* routes for each sub-section by the *RouteFinder* service (with *bike* capability) and their details by the *RouteDetails* service. Finally, all *bike sections*, *public sections* and *PoI sections* are combined into a single result *route* by the *RouteCombiner* service instance.

These changes and extensions are made in the orchestration model without affecting the underlying implementations and any other artifacts. This allows for sustainable evolution of the mobility platform. Modeling service orchestrations allows to remain abstract from technical implementation and does not require programming skills or expert knowledge of the diverse technologies used by components.

6 Evaluation and Contributions

This section evaluates the objectives on sustainable software development and evolution, validates the reference architecture and discusses the application of SENSEI and DORI in the context of the NEMo mobility platform.

The NEMo mobility platform is subject to software evolution and has to remain up to date with new or modified mobility requirements, e.g., new business models, mobility services and implementations. Continually adapting the mobility platform leads to more complex and less maintainable software systems. Due to the innovative nature of the NEMo project, a sustainable and adaptable software architecture plays an essential role in providing simple and fast development, integration and maintenance of new features, i.e., sustainable software development.

The reference architecture explained in Section 2.1 serves as architectural provisions for developing the NEMo-like mobility platforms focusing on sustainability, flexibility and adaptability. The architectural provision covers the major use case; inter-modal routing combining the different modes of transportation, e.g., walk, bike, bus, train, carpooling, etc. The ICT mobility infrastructure, which was developed in advance to NEMo, is already able to support the use case of inter-modal routing. However, it has not been designed with a focus on software sustainability, which makes it hard to evolve.

Sustainability objectives defined in Section 1.2 are entirely achieved by the reference architecture (Section 2.1), and the application of the SENSEI (Section 2.2) and the DORI approach (Section 2.3).

- *Sustainable Mobility Platform by Sustainable Reference Architecture*: A flexible, adaptable and extensible reference software architecture serves as a common underlying blueprint for developing mobility platforms highlighting flexibility, adaptability, and long-term sustainability. This architecture provides clear extension points for services and components to enable software adaptations easily.
- *Reusability*: The existing mobility services and their implementaions (components) can be reused, enhanced and modified, and future changes (i.e., extensions, optimization and corrections) based on the research findings within the NEMo project can be incorporated within the mobility platform without much technical knowledge and effort.
- *Separation of Concerns*: The sustainable reference architecture and its associated technical support (SENSEI and DORI) provides consistent separation of novel business models, model-driven, service-oriented mobility services, and component-based functionality enabling separation of functionality (services) and implementation (components).

- *Interaction Modeling:* It provides a model-driven, flexible interaction modeling feature for separation of user interaction and user interface designs.

This paper has demonstrated the application of the SENSEI and DORI approaches to develop a sustainable and flexible mobility platform based on the sustainable reference architecture in the framework of the NEMo project. The clear separation of concerns, i.e., services and components in SENSEI allows to specify application behavior on a non-technical level, close to the application domain. Service orchestrations are comparatively easy to adapt or extend, and the corresponding software application can be re-generated, allowing for fast turnarounds, and resulting in a high degree of flexibility. The use of SENSEI reduces the effort required to develop and maintain the mobility platform, particularly when sustainability raises.

The only prerequisite of applying the proposed approach, is the provision of basic functionality, as in the NEMo case already available in the form of components provided by the existing ICT Platform. The component-based structure supported by SENSEI promotes building up the catalog of both services and components, so that over time existing functionality can be readily reused, adapted, extended or new ones can be added. Both aspects potentially increase productivity and serve as the basis for sustainable mobility platforms.

References

Jelschen J, Küpker C, Sandau A, Wagner vom Berg B, Gómez J M, Winter A: *Towards a Sustainable Software Architecture for the NEMo Mobility Platform.* In: EnviroInfo (2), 2016. pp 41-47

Akyol A, Halberstadt J, Hebig K, Kuryazov D, Jelschen J, Winter A, Sandau A: Marx G J: *Flexible Software Support of Imovated Mobility Business Models*, In: no.:31, Adjuct Proceedings of the 31st EnviroInfo Conference, pp. 27-34, Luxembourg, Shaker Verlag, September 2017.

Kuryazov D, Winter A, Sandau A: *Sustainable Software Architecture for NEMo Mobility Platform.* In, Wiesbaden, 2019. Smart Cities/Smart Regions – Technische, wirtschaftliche und gesellschaftliche Innovationen. Springer Fachmedien Wiesbaden, pp 229-239

Lehman M M: *Laws of software evolution revisited*. In, Berlin, Heidelberg, 1996. Software Process Technology. Springer Berlin Heidelberg, pp 108-124

Combemale B, Cheng B H, Moreira A, Bruel J-M, Gray J: *Modeling for sustainability*. In: 2016 IEEE/ACM 8th International Workshop on Modeling in Software Engineering (MiSE), 2016. IEEE, pp 62-66.

Rajlich V T, Bennett K H (2000): *A staged model for the software life cycle*. Computer 33 (7):66-71. doi:10.1109/2.869374

Kateule R, Winter A (2018): *Architectural Design of Sensor based Environmental Information Systems for Maintainability*. In: Arndt H-K, Marx Gómez J, Wohlgemuth V, Lehmann S, Pleshkanovska R (eds) Nachhaltige Betriebliche Umweltinformationssysteme: Konferenzband zu den 9. BUIS-Tagen.

Springer Fachmedien Wiesbaden, Wiesbaden, pp 87-96. doi:10.1007/978-3-658-20380-1_9

Wagner vom Berg B (2015): *Konzeption eines Sustainability Customer Relationship Managements (SusCRM) für Anbieter nachhaltiger Mobilität*. Shaker Verlag.

Dijkstra E W (1982): *On the role of scientific thought*. In *Selected writings on computing: a personal perspective* (pp. 60-66). Springer, New York, NY.

Akyol A, Halberstadt J, Hebig K, Jelschen J, Winter A, Sandau A, Gómez J M (2017): *Flexible Software Support for Mobility Services*. INFORMATIK

Jelschen J: *Service-oriented toolchains for software evolution*. In: 2015 IEEE 9th International Symposium on the Maintenance and Evolution of Service-Oriented and Cloud-Based Environments (MESOCA), 2-2 Oct. 2015 2015. pp 51-58. doi:10.1109/MESOCA.2015.7328127

Kleppe A, Warmer J, Bast W (2003): *MDA explained: the model driven architecture: practice and promise*. Addison-Wesley Professional.

Breivold H P and Larsson M (2007): *Component-based and service-oriented software engineering: Key concepts and principles*. In EUROMICRO 2007-Proceedings of the 33rd EUROMICRO Conference on Software Engineering and Advanced Applications, SEAA 2007, pages 13–20. IEEE, aug 2007. ISBN 0769529771.

Jelschen J: *Software Evolution Services, A Framework for the Integration and Development of Flexible and Reusable Toolchains*, PhD thesis, University of Oldenburg, to appear, 2020.

Brambilla M and Fraternali P (2014). *Interaction flow modeling language: Model-driven UI engineering of web and mobile apps with IFML*. Morgan Kaufmann.