Towards a Sustainable Software Architecture for the NEMo Mobility Platform

Jan Jelschen, Christoph Küpker, Andreas Winter,
Alexander Sandau, Benjamin Wagner vom Berg, Jorge Marx Gómez

{jelschen, kuepker, winter}@se.uni-oldenburg.de,
{alexander.sandau, benjamin.wagnervomberg, jorge.marx.gomez}@uni-oldenburg.de
Carl von Ossietzky University, Oldenburg, Germany

The NEMo project aims at building a software system for the *sustainable fulfillment of mobility needs in rural areas*. The resulting *mobility platform* should facilitate communication, interaction, and business transactions between mobility service consumers and providers. It has to be flexible enough to co-evolve with changing and novel mobility needs, services, and business models – the software must therefore be sustainable, too.

The SENSEI approach proposes a framework to define (business) processes in terms of technology-independent services that are mapped automatically to suitable, reusable software components, from which the final software system is generated. This paper outlines the planned application of SENSEI to the NEMo mobility platform, and describes the expected benefits in terms of flexibility, longevity, and sustainability.

1 Introduction

Over 60% of Germany's population lives in rural areas [1], yet medical facilities, shopping centers, cultural and recreational offerings, as well as job, training, and educational opportunities are primarily found in urban centers. This rises diverse mobility needs not (sufficiently) addressed by public transport.

The interdisciplinary research project NEMo[1] aims at the sustainable fulfillment of mobility needs in rural areas, and opts for a holistic view, considering social, demographic, accessibility, legal, economic, and ecological conditions and objectives. NEMo wants to facilitate the provision of novel mobility services based, for example, on social self-organization, and to develop business models that increase utilization of private transport, while reducing the overall number of vehicles on the streets. Information technology is viewed as key enabler to support these objectives by implementing a mobility platform. A prototype of such a mobility platform will be a major outcome of the overall project.

Like any software system, the NEMo mobility platform will need to evolve to keep up with new or modified requirements, e.g. changing legal regulations, or new business models for the provision of

[1] This work is part of the project "NEMo - Sustainable satisfaction of mobility demands in rural regions". The project is funded by the Ministry for Science and Culture of Lower Saxony and the Volkswagen Foundation (Volkswagen-Stiftung) through the "Niedersächsisches Vorab" grant program (grant number VWZN3122).

mobility services. Unless specifically counteracted, continuously adapting the mobility platform to support unanticipated use cases leads to an ever more complex and less maintainable software system [2].

The NEMo mobility platform should be able to support all kinds of mobility needs and scenarios, modes of transportation, and accompanying business models. It should facilitate the recombination of existing mobility services to provide enhanced services, as well as completely new, unanticipated usage scenarios, and the development of corresponding business models. Besides the technical implementation issues that may arise due to a required software change, the semantic gap between process-oriented business needs on the one hand, and mostly object-oriented software architecture views on the other hand, remains a major software engineering challenge [3, 4].

Finally, with the overall goal of NEMo being to foster *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 phase-out, closedown [5], and forced replacement. To be sustainable, the NEMo mobility platform must make architectural provisions for flexibility, evolvability, and longevity.

This paper proposes to apply the SENSEI [6] approach to the design and development of the NEMo mobility platform. SENSEI provides *service-oriented* software design facilities (service orchestration) on an abstraction level close to the application domain of mobility needs and services. Strictly separated from this layer, concrete implementations of these services are realized in *component-based* terms, to promote reuse. An *automated mapping* from services to components, and *model-driven* code generation, bridges the gap between service-oriented specification and component-based realization.

This paper is outlined as follows: Section 2 introduces an exemplary usage scenario for the mobility platform, and preexisting infrastructure. Section 3 describes the central concepts of the SENSEI approach, and sketches its application in NEMo. Section 4 summarizes expected benefits, and outlines future work.

2 The NEMo Mobility Platform

NEMo[2] commenced in March 2016 to answer the research question of how to satisfy mobility needs of rural areas, based on existing social structures, in a purposeful, sustainable manner. To do this, an inter- and transdisciplinary project consortium has gathered to investigate the NEMo challenges holistically and comprehensively, covering social sciences, economics, law, business informatics, and computer science.

The focus of this paper is firmly on the computer science perspective. Of course, the software support depends heavily on the scientific results and outcomes of the other disciplines. However, since the project was only started recently, there are no final results to build upon, yet.

Some functional requirements are already established, largely due to previous activities and outcomes from the *ICT Services* and *ICT Platform* projects in the context of the *Electric Mobility Showcase Lower Saxony* program [7, 8]. The projects created ICT infrastructure which will serve as a basis for NEMo.

A first, simple, exemplary use case for the NEMo mobility platform is to find routes: given a point of origin and a destination, it should provide possible routes. Combining different modes of transport, e.g. walking, riding a bike, take a bus or a train, driving a private car, or joining a car pool, makes this *inter-modal routing*. The existing ICT infrastructure already supports this use case to a large extent.

[2] http://www.nemo-mobilitaet.de

However, the current infrastructure has not been designed with a particular focus on flexibility and evolvability. The business processes underlying the use cases, such as for inter-modal routing, are following fixed procedures with a small set of dynamic variables. The system is therefore hard to adapt and extend. This is not a good fit for NEMo, given its sustainability objective, and the need for the mobility platform to sustain, facilitate, and mirror the project's progress, innovative nature, and evolving requirements regarding business models and software support. In summary, the goals for the NEMo mobility platform are therefore as follows:

1. Enhance and extend existing functionality based on research findings, to support, e.g. community-driven, self-organized mobility of prosumers[3], novel business models, and legal constraints.
2. Facilitate these, and future changes, by providing an architectural framework that incorporates the existing functionality, but highlights flexibility, adaptability, and long-term sustainability.

The exact nature of the first goal will only be known once the project has progressed further. The solution proposed to tackle the second goal is adopting the SENSEI approach for building the mobility platform.

3 The SENSEI Approach

SENSEI (*Software EvolutioN SErvice Integration*) [6] was originally conceived to address the lack of interoperability of software evolution tools. It aims at providing a toolchain-building support framework, increasing flexibility, reusability, and productivity. SENSEI combines service-oriented, component-based, and model-driven techniques to map high-level process models (*orchestrations*) onto reusable components, possessing the required capabilities, and generate code that combines and coordinates them in the required manner. The approach is expected to be applicable to general application development, too, providing the same benefits: a high degree of flexibility and evolvability, and therefore sustainability.

3.1 Overview

SENSEI emphasizes clear separation of concerns, distinguishing **services** that abstract from technologies and interoperability issues on one hand, and concrete implementations as **components** on the other hand.

SENSEI is largely defined by its *metamodel*, which allows to model software support for (business) processes on a high level, close to the problem domain (e.g. rural mobility), using *services*. It consists of the following core concepts: The **service catalog** serves as a central repository containing service definitions described in a standardized manner. **Service orchestrations** allow combining services from the catalog to create more complex functionality, using a process-oriented, graphical modeling language. The **component registry** establishes relations between services defined in the catalog, and components that provide the functionality defined by them. Orthogonal to these three layers are **capabilities**, which help organize the catalog and describe its services concisely, and precisely specify required and provided functionality of orchestrated services and registered components, respectively.

Modeling application behavior using this notion of services allows to abstract from technical implementation and interoperability issues, does not require programming skills or expert knowledge of the diverse technologies used by components, and thus makes for a simpler and faster integration process.

[3] Consumers who also take on the role of mobility service providers, such as in carpooling.

Of course, the specified functionality has to be implemented, eventually. SENSEI prescribes the use of software *components*, which ensures unified structures and eases reusability. The specification of provided functionality through services further promotes reusability and interchangeability. These measures help to ensure no need for redundant implementation efforts: with the adoption of SENSEI, as the number of available services and components rises over time, the need to leave the service level to create or modify components decreases. For NEMo, a basic set of mobility services has already been implemented within the ICT Services project, and will be reused by wrapping the functionality in SENSEI components.

Service specifications and component implementations are tied together by automatically mapping orchestrated services to appropriate, registered components, and by generating code gluing components together and implementing orchestration logic, i.e. routing data and coordinating component invocation.

3.2 SENSEI Applied

Using SENSEI encompasses the following steps: 1. Defining services and their capabilities, 2. modeling orchestrations with their required capabilities, 3. implementing and registering components with their provided capabilities, 4. *automatically* matching orchestrated services to registered components, and 5. *automatically* generating composer code implementing orchestrations.

Steps 1 and 3 can be considered bootstrapping: Step 1 results in a *service catalog*, and Step 3 in a *component registry*. Once established, these steps are skipped, although both, new services and components will occasionally be added. The central activity is Step 2, creating *orchestrations* to model complex functionality combining more basic services. Using these three artifacts as input, the remaining steps are performed automatically: Step 4 compares orchestrated services and their *required capabilities* with registered components and their *provided capabilities*, and tries to find appropriate matches. If successful, Step 5 generates the code required to realize the behavior modeled in the orchestration and "glues" the matched components together into a composition, resulting in a complete, executable software system.

The manual steps are supported by the SENSEI *Editor*. A screenshot is shown in Figure 1, with a model of the inter-modal routing example, which was briefly introduced in Section 2. The steps of the corresponding business process are mapped to three services: *Find Stops* decomposes a routing request into requests for sub-routes, *Find Routes* provides traveling information (e.g. itineraries or driving instructions) for a single route, and *Combine Routes* integrates them into coherent instructions for the whole trip.

Service Catalog: In the left-hand side tree view of Figure 1, the service catalog can be seen. Services have names and descriptions, along with input and output parameters, and associated types, also modeled in the catalog as data structures. The *Find Route* service, for example, takes a routing request and the desired mode of transport, and returns traveling information. In addition, services have *capabilities*.

Capabilities in SENSEI are a means to describe service variants concisely. For example, implementations of the *Find Route* service cannot usually be expected to be able to provide traveling information for *all* imaginable modes of transport, and 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 was to be defined for every possible variant. For the *Find Route* service, this is used to model variabilities in terms of supported transport modes, with capabilities being *car*, *train*, *bus*, etc. Another class of capabilities represents supported optimization goals, such as *shortest route*, *fastest route*, *cheapest connection*, or *lowest* CO_2 *footprint*.

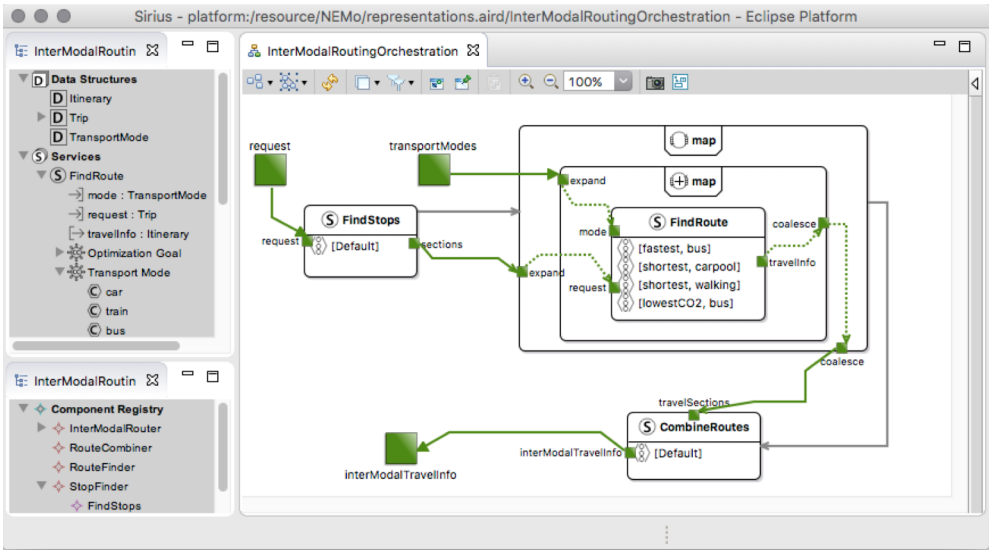

Figure 1: Inter-modal routing use case modeled in the SENSEI editor.

Service Orchestrations: Using services defined in the catalog, the intended behavior of the software to support the inter-modal routing use case is modeled as a *service orchestration*. This is done graphically, and the resulting orchestration is shown in the center of the editor screenshot. Data flows are drawn as green arrows, connecting inputs and outputs (green boxes) of services (rounded rectangles with their name next to an encircled "S" at the top). Gray arrows indicate control flow. Structured control flow constructs are available to model branching, concurrency, and loops (the two nested *map* constructs iterate data fields).

Required capabilities are shown for the *Find Route* service, in the lower compartment of its shape. The capabilities specified demand support for finding either the fastest bus route, or the one with the lowest carbon footprint, as well as the shortest routes by carpool or by walking.

Component Registry: Services are *implemented by* components. This relationship is persisted in the component registry, which can be thought of as a table. Each entry refers to a component and lists one or more services it implements. With each service, the *provided capabilities* are specified in exactly the same way as required capabilities for orchestrations. The specification of implemented services and provided capabilities can also be used to generate the boilerplate code, as an aid to developing a component for a particular target platform, or wrapping existing software systems for SENSEI compatibility.

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 orchestration.

This does not have to be a one-to-one mapping, e.g. the *required capabilities* specified in the orchestration for *Find Route* might be implemented by a single component that provides them all, or each may be provided by an individual component.

Composer Generation: To conclude the process, the SENSEI model is mapped to a particular target platform that defines the component model, and provides a runtime environment, such as WSO2 [9], 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, new component called *composer*, that depends on the components found in the previous step to perform the work specified by the orchestrated services.

The composer itself implements the orchestration logic, i.e. routes data according to the data flows, and invokes components in the order dictated by control flow. If a single service is mapped to multiple components, it will automatically choose the right one based on runtime data and provided capabilities. The result is an executable software application, ready to be deployed, e.g. to a WSO2 application server.

4 Summary and Outlook

This paper has sketched the planned application of the SENSEI approach to create a sustainable mobility platform for the NEMo project. The clear separation of 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 regenerated, allowing for fast turnarounds, and resulting in a high degree of flexibility.

The prerequisite is, of course, that the basic functionality is already available in the form of components. While this is initially not the case, the component-based structure enforced by SENSEI promotes building up a library of both services and components, so that over time an increasing amount of required functionality can be readily reused.

Both aspects potentially increase productivity in the long term, and are the basis for software systems that are long-living and sustainable. While the approach has already been applied successfully to toolchain building, showing its applicability for generic application development is an objective of NEMo.

As a highly relevant topic, both for NEMo and in general, the introduction of *quality of service* (QoS) attributes within the SENSEI framework is planned to be investigated. The existing capability model currently supports the expression of functional properties, and is thought to provide an excellent basis for such an extension. This would allow to automatically find components for services with specific quality requirements, e.g. in terms of correctness, performance, and responsiveness. If evaluated at runtime, it could also enable highly context-sensitive applications, whose orchestrated services are mapped to different components based on the quality needs and provisions at that moment.

References

1. Statistisches Bundesamt (ed.): Statistisches Jahrbuch Deutschland 2015. Wiesbaden (2015).
2. Lehman, M.M.: Laws of software evolution revisited. Lecture Notes in Computer Science. 1149, 108–124 (1996).
3. Combemale, B., Cheng, B.H., Moreira, A., Bruel, J.-M., Gray, J.: Modeling for Sustainability. Modeling in Software Engineering 2016 (MiSE'16). ACM (2016).
4. Schmidt, D.D.: Guest Editor's Introduction: Model-Driven Engineering. Computer. 39, 25–31 (2006).
5. Rajlich, V., Bennett, K.: A Staged Model for the Software Life Cycle. Computer. 33, 66–71 (2000).
6. Jelschen, J.: Service-Oriented Toolchains for Software Evolution. 9th International Symposium on the Maintenance and Evolution of Service-Oriented and Cloud-Based Environments (MESOCA). pp. 51–58. IEEE (2015).
7. ICT Services, http://www.ikts-niedersachsen.de/en (2016).
8. Wagner vom Berg, B.: Konzeption eines Sustainability Customer Relationship Managements (SusCRM) für Anbieter nachhaltiger Mobilität. Shaker, Aachen (2015).
9. WSO2, http://wso2.com (2016).