Generating User Interface Documentation Based on Imported Service Models

Hendrik Bünder
buender@itemis.de
itemis AG

Abstract

Modernization of software systems often starts with replacing the graphical user interface (GUI), while the existing services are reused. Besides specifying the new user interface, supporting material such as system and user documentation have to be created. The paper introduces a model-driven approach to create new user interfaces based on existing service definitions. Additionally, it will be shown how the user interface model can be utilized to automatically generate system and user documentation based on the input and output parameters of the existing services and their documentation.

1 Introduction

Software modernization is driven by the need for modern graphical user interfaces or additional frontends such as web or mobile. Business logic and persistence, in contrast, should be reused. Instead of reimplementing the user interface in the latest technology, such as JavaScript [5] or Angular [1], a model-driven approach is proposed. In order to efficiently create user interfaces for different target technologies, abstract user interface models are specified. Based on these models not only the application source code for different UI frameworks but also system and user documentation can be generated automatically. Additionally, future technologies can be adopted faster by implementing a new transformation and generation process.

The user interface models are built on top of service and data type models as introduced by Buender [2] that represent legacy services. Based on the used data types *Layouts* are specified that arrange the attributes on the GUI. Further, the user interface models reference the services required to load and manipulate the data shown in the GUI. Since the documentation of data types and attributes is imported into the anti-corruption layer model, they can be leveraged for generating the system and user documentation.

2 Specifying the Graphical User Interface

As shown by Figure 1 the user interface metamodel is built on top of the metamodel for modeling anti-corruption layer for legacy services (shadowed in light grey) [2].

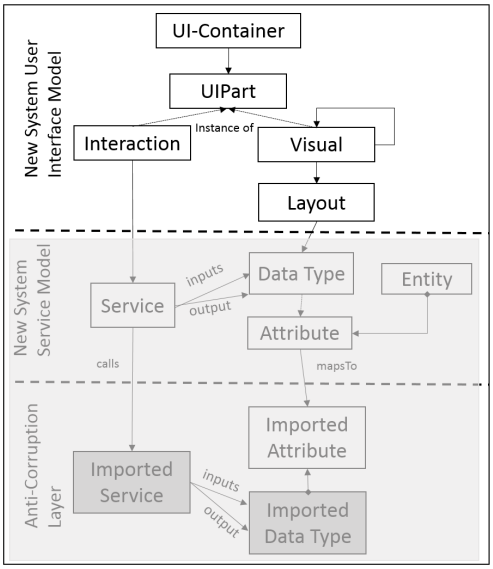

Figure 1: Metamodel for Legacy Services and User Interface

The *Layouts* are built upon existing data types that are also used within the services. *Layouts* define how the data type attributes should be represented by specific widgets on the user interface. Besides specifying their position it is also possible to alter the widgets labels or their types such as *ComboBox* or *RadioButtonGroup* for enumeration attributes. Additionally, the purpose of the *Layout* is stored in a multi-language documentation attribute.

Layouts are wrapped by *Visuals* that represent either panels or whole pages. While a panel *Visual* holds a direct reference to a *Layout*, a page *Visual* references an arbitrary number of *Visuals*. Thereby, multi-page applications can be specified by a combination of *UI-Containers* and *UI-Parts*. In that case, the *UI-Container* represents the whole application, while the page *Visuals* represent pages that reference panel *Visuals* and eventually *Layouts*. In addition, there are *Interactions* that encapsulate service calls and make their input and output data types available in the user interface model. Both share the parent

class *UI-Part*. The *UI-Container* bundles an arbitrary number of *Visuals* and *Interactions* that form one graphical user interface. Thereby, the model holds information about the static structure of the user interface as well as the dynamic parts of reading and processing the data through existing services.

There are different parts of the model containing documentation information. First, *UI-Container, UI-Part, Interaction and Visual* all have a documentation attribute. Second, *Imported Services*, *Imported Data Types*, and *Imported Attributes* are enriched by documentation extracted from the existing implementation.

3 Generating Graphical User Interface Documentation

A transformation and generation process is implemented that converts instances of the metamodel presented in Figure 1 into system and user documentation.

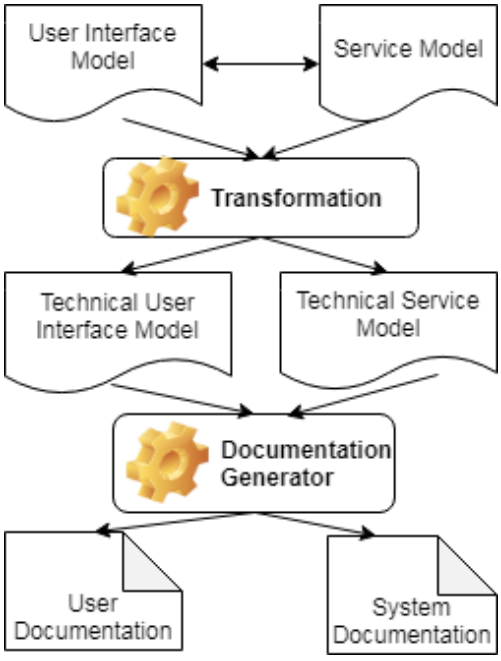

Figure 2: Documentation Transformation and Generation Process

Figure 2 shows that the user interface and service model are jointly transformed into a technical user interface and service model. For every technology that should be supported by the application, e.g. JavaScript or Angular, a specific technical model is created. The models are not only specific for the used technology but also for the target device. Thereby, the varying screen sizes e.g. for desktop and web application can be mapped properly. Consequently, user documentation is generated accordingly.

As shown by Figure 2 a generator eventually converts the technical models into the system and user documentation. Existing documentation was used as a blueprint for implementing the generator templates [4]. By replacing the dynamic parts with place holders the templates were created. The Jinja2 generator engine [3] combines templates and technical models to create documentation files.

The development process guideline dictates that for each graphical user interface system documentation must be provided. The system documentation must include all screens of the application, the widgets, and their properties as well as the services called by the GUI. Based on the technical user interface and service models the system documentation could be generated completely.

In addition, the software itself offers online help to provide in-place information in the user interface of the application. The most important part of the user documentation is the description of each widget and the expected input in terms of upper and lower bounds, regular expressions, or documentation strings. Since the documentation of the attributes specified by the existing system is available within the model it can be generated into the user documentation of the application. Additionally, since documentation might change based on the type of widget or the usage scenario, the generator interweaves legacy with newly specified documentation. Consequently, the user documentation can be generated completely.

4 Conclusion

It was shown how the introduced model-driven approach for user interface modeling is built upon imported legacy services. Further, the paper elaborated on how user interfaces are modeled and eventually be transformed into user and system documentation. By relying on existing documentation and the automatic creation of the required files the efficiency of creating this documentation was increased. Further, the user interface models enable an easy switch of target technologies in future by not relying on a specific technology.

References

[1] Angular. *Angular - One framework. Mobile and desktop.* URL: `https://angular.io/`.

[2] Hendrik Bünder. "Anti-Corruption Layer Modeling Introducing a Model-Driven Approach to Integrate Legacy Software". In: *Softwaretechnik-Trends* 38.2 (2018), pp. 65–66.

[3] Jinja. *Jinja2 - Welcome.* 2018. URL: `http://jinja.pocoo.org/`.

[4] Thomas Stahl. *Modellgetriebene Softwareentwicklung: Techniken, Engeneering, Management.* 2., aktual. und erw. Aufl. Heidelberg: Dpunkt.verlag, 2007. ISBN: 978-3898644488.

[5] W3C. *JavaScript Tutorial.* URL: `https://www.w3schools.com/js/`.