A pipe-based approach for service-oriented UIs

Mathias Kühn and Peter Forbrig
University of Rostock
Albert-Einstein-Str. 22
18051 Rostock, Germany
{mathias.kuehn, peter.forbrig}@uni-rostock.de

ABSTRACT
Service-oriented architectures as instruments for distributed applications can be used in contexts that are grounded on networked devices. Specifications of web service interfaces can be instantiated across different platforms. Combining this method with specifications of platform-independent user interface models would allow to access applications from various interactive devices. According to this, application states and events need to be considered for individual consumers. Additionally, interpreters and generators need to be available for each consumer.

The paper proposes an approach for consumer-specific configurations of mentioned tools that are based on models derived from the Cameleon Reference Framework. In the end, pipes are configured that allow using web services by generated user interfaces. Configuring these pipes with models for layouts and dialogs is focus of the paper.

Author Keywords
Model-based user interfaces, multi-path development, interactive system design

ACM Classification Keywords
H.5.m. Information interfaces and presentation (e.g., HCI): Miscellaneous.

INTRODUCTION
Software that is based on service-oriented architectures can be used together with various platforms. Web services can be addressed from consumers nearly with any platform. Related interfaces can be specified with Web Service Definition Language (WSDL) and need to be generated at runtime. Also user interfaces (UIs) can be specified with model-based languages that can be generated using methods which are grounded on the Cameleon Reference Framework. Combining both methods for generating interfaces would allow using functional cores together with UIs.

Generated user interfaces need to be flexible due to the fact that user abilities can vary a lot. UIs for users that are able to see differ from UIs for users that are blind. UIs can for instance be graphical or vocal. However, needed tool support could apply methods for generating UIs on different levels of abstraction. UI events need to be reified and abstracted at runtime. Additionally, domain data need to be communicated for any application. Especially web services are good to achieve this goal, because serialization & marshalling are supported by default.

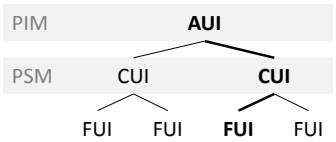

Figure 1. Transformation paths for generated FUIs.

Figure 1 shows a visualization of paths that generated final user interfaces (FUIs) can be transformed to when applying the Cameleon Reference Framework. Many FUIs can be generated from a specific concrete user interface (CUI), for instance for graphical UIs. Additionally, many CUIs can be transformed from specific abstract user interfaces (AUIs), for instance for graphical or vocal UIs. However, single FUIs have specific paths that they can be transformed to. These paths could be represented by pipes that are configured by models which describe UIs at different levels of abstraction. Applying pipes would allow transforming AUI to FUI and FUI to AUI whenever platform-independent models are needed. These models allow a broader range of applications for interactive systems.

Service-oriented architectures [9] are well-known in industry these days. Implementations are independent of specific platforms, what makes them more usable for a variety of different consumer platforms. Providing services allows using functions of software systems by any consumer. Different services can be combined to achieve new functionality. However, services also can be used to provide functional cores of interactive systems. One application core can be used by many different consumers. According to this, software architectures based on services would allow to use interactive systems together with any kind of UI.

The Cameleon Reference Framework [3] considers UIs on different levels of abstraction. Specifications can be made at the level of tasks, abstract, and concrete UIs. Additionally, UIs can be transformed to other levels of abstraction as well as to different contexts of use at the same level. Following that, the transformation types are translation, abstraction, and reification. In the end, final user interfaces can be generated from concrete UI specifications. These FUIs result from specific transformations along the different levels of abstraction. Configuring pipes for performing transformations could help in applying the framework.

The Arch model [1] separates interactive systems into different kinds of components that are depending on each other. The approach can also be seen as pipe, but it does not consider a parallel processing of data and events. Instead of this, it considers a sequence of components that assume certain relations between data and events. However, UI (presentation & interaction components) and data (domain adaptor & domain-specific component) also are considered on different levels of abstraction. Both kinds of components are related to each other on the level of dialog structure (dialogue component) that forces a sequential dependency.

The context of use [8] for interactive systems is an information space spanned by entities user (U), platform (P), and environment (E). It is defined as triple composed by U, P, and E. Relevant factors for users (U) are their physical abilities to see, hear, feel, smell, and taste. Relevant factors for platforms (P) are their potentials to externalize information visual (e.g. with displays), acoustical (e.g. with speakers), haptic, olfactory, and gustatory. For instance, displays can only be used by users that (temporary) are able to see. Additionally, relevant factors for platforms can be to internalize information with different kinds of sensors that are related to environmental factors. Relevant factors for environments (E) are light, sound, texture, aroma, and flavor. All factors are depending on information of space and time.

	Visual	**Acoustical**	**Haptic**	**Olfactory**	**Gustatory**
User	See	Hear	Feel	Smell	Taste
Platform (Ext.)	Display	Speakers	*Actuator*	*Actuator*	*Actuator*
Environment	Light	Sound	Texture	Aroma	Flavor
Platform (Int.)	Camera	Microphone	*Sensor*	*Sensor*	*Sensor*

Table 1. Relevant factors for the context of use.

Table 1 summarizes factors that are considered for the context of use. According to user-centered designs and as mentioned before, all stimuli that users physically can perceive with senses are listed here. Targeted platform components to internalize & externalize information as well as environmental effects, which surely can be other users, also are listed here.

RELATED WORK

UIs based on models are beneficial for targeting the generation of individual user interfaces that depend on specific contexts of use [6]. Model-based user interfaces (MBUIs) are independent of concrete platforms, but can be transformed so that they can be applied to specific exemplars. Widgets of programming languages can be addressed from UI specifications that are based on these models. For instance, containers within MBUI specifications can be transformed to JFrame objects in Java Swing or to Window objects in C#. However, using model-based UIs allows addressing different platforms that can fit the variety of different end-user groups.

In [7] Paternò et al. propose an extension of WSDL that uses annotated specifications which can be combined with models for UIs. These annotations can be related to tasks of task model specifications in the end. By using this approach designers need to be aware of task model semantics and relations between tasks and web services. However, functionality of web services also can be used directly together with UIs that were generated from abstract models. These UIs are strongly coupled to specific functional cores.

In [2] Akiki et al. introduce the Cedar Architecture that uses web services for adapting UIs. Specifications of UIs also are communicated for making them context-specific then. Bindings to the application core are specified for individual clients and are not part of the UI model. Each consumer of functionality that is provided by web services need to adapt mappings to the application core by itself. However, UIs are context-specific and can be used by a variety of different groups of end-user clients.

The User Interface XML (UsiXML) [5] is one exemplar of model-based languages for UIs which is used for the proposed approach. UIs can be specified at the level of tasks, AUIs, and CUIs. Additionally, models for mappings, transformations, and contexts can be used for specifying dependencies to their application in general. Models for specifying behavior with state machine semantic is not covered by the language. However, UI layouts can be described in UsiXML and can be used for generating FUIs on any platform at runtime.

PROPOSED APPROACH

The approach in general considers data and events for communication based on web services at runtime. Domain-specific data need to be shared between service providers as well as consumers. Data continuously will be marshalled and de-marshalled for this reason. UI-specific data also need to be shared. UI-related events will consumer-sided be reified and abstracted. This is needed because only abstract events will be communicated between providers and consumers. Events additionally need to be mapped to functions that are implemented on provider side and also need to be mapped e.g. to widgets of generated FUIs.

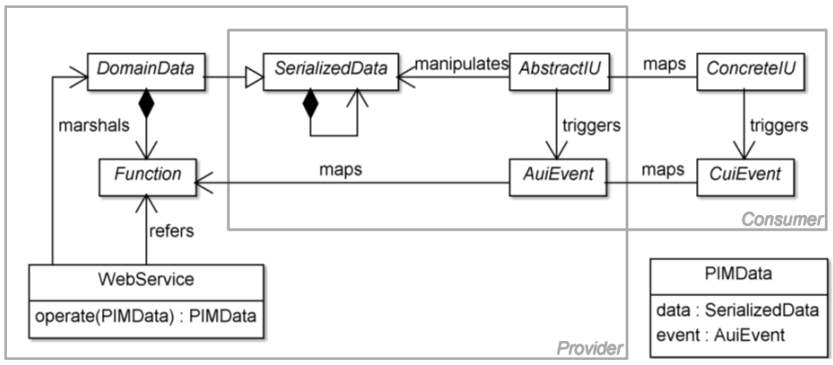

Figure 2. Architectural structure of the proposed approach.

Figure 2 shows a class diagram for the architectural structure of the proposed approach. The structure is separated into provider- and consumer-parts. Their intersection represents the structure which is used with web services. Class `WebService` uses instances of `PIMData` that are shared between consumers and providers. `SerializedData` only contains data and `DomainData` additionally contains functions. All instances of `Function` represent functional cores of applications. These functions are referred by web service instances. CUI models which consist of `ConcreteIU` instances are generated from AUI models which consist of `AbstractIU` instances. Instances of both are mapped to each other after generating UIs. Events that can be triggered later also are mapped to functions of the core.

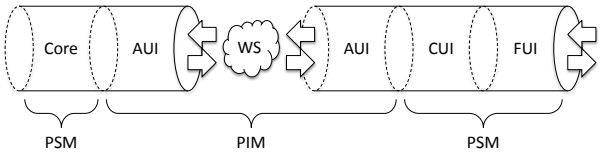

Figure 3. Visualization of the pipe for the proposed approach.

Figure 3 shows a visualization of the proposed approach for context-specific UI instances. Events of generated FUIs will be abstracted, communicated via web service interfaces (WS) together with relating domain data, and computed on provider side (Core), afterwards. Resulting events of the Core also will be communicated via the same WS interface together with domain data, reified on consumer side, and their effects will be presented on FUIs at last. Platform-specific models (PSMs) are used on provider and consumer side, but communication of events and domain data is made via platform-independent models (PIMs), what benefits the approach in general. Additionally, communication based on web services as well as serialized data are platform-independent, what makes the approach usable for any application of context-specific UIs with interactive systems. However, transformations of AUIs to FUI instances also need to be specified depending on corresponding applications.

Simple Example

This section introduces the approach at the example of a simple specification. The pipe will be configured by values that match the consumer-specific context of use for the UI.

```
<uiModel id="0">
    <auiModel id="01">
        <abstractIU id="011">
            <trigger id="0111"/>
            <!-- etc. -->
        </abstractIU>
    </auiModel>
    <cuiModel id="02">
        <concreteIU id="021">
            <button id="0211"/>
            <!-- etc. -->
        </concreteIU>
    </cuiModel>
    <cuiModel id="03">
        <concreteIU id="031">
            <voiceCommand id="0311"/>
            <!-- etc. -->
        </concreteIU>
    </cuiModel>
    <!-- etc. -->
</uiModel>
```

Figure 4. Example of a UsiXML specification.

Figure 4 shows a specification of a model-based UI that can be used for configuring pipes which allow reifications and abstractions of UI-specific events at runtime. A single AUI model specification (`id="011"`) can be transformed to CUI model specifications either for graphical (`id="021"`) or vocal (`id="031"`) UIs depending on the consumer-specific context of use. The FUI is generated from the CUI afterwards. However, events that are triggered on context-specific FUIs are transformed to their abstract representations in AUI for use with web services.

CONCLUSION AND FUTURE WORK

The paper proposes an approach that allows using web services with context-specific UIs which individually will be generated for consumers. A pipe allows abstracting and reifying events that are relating to generated UIs. The pipe also supports serializing and marshalling data that need to be communicated between consumers and providers. It is assumed that functional cores of applications are provided by web services. These can be used by any UI that is generated. The pipe prepares events and data especially for consumer- or provider-side. Functions of the functional core only take events at the AUI level into account. However, the approach in general can only be used if sensors are available on consumer-side. It is assumed that end-user abilities can be detected by those sensors.

Next investigations need to be made on giving tool support for making specifications. The web service interface acts as façade [4] to the application core. It would be nice to generate this façade from an existing core. However, contexts of use are only considered when starting the application. It would be nice to react on changes at runtime so that generated UIs can always depend on the current context of use. For instance, a graphical user interface can be shown if end-users look at a display. If not, a vocal UI can be used for interacting with the application.

REFERENCES

1. A metamodel for the runtime architecture of an interactive system: the UIMS tool developers workshop. In *ACM SIGCHI Bulletin*, vol. 24 no. 1 (1992), 32-37.
2. Akiki, P. A., Bandara, A. K., and Yu, Y. Engineering Adaptive Model-Driven User Interfaces. *IEEE Transactions on Software Engineering*, vol. 42 (2016), 1118-1147.
3. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., and Vanderdonckt, J. A unifying reference framework for multi-target interfaces. *Interacting with computers*, vol. 15 (2003), 289-308.
4. Gamma, E., Helm, R., Johnson, R., and Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Prentice Hall, 1st ed. (1994).
5. Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., and López-Jaquero, V. UsiXML: a language supporting multi-path development of user interfaces. In *Proc. of EHCI-DSVIS 2004*, Springer (2005), 200-220.
6. Paternò, F and Santoro, C. One model, many interfaces. In *Proc. of CADUI*, Springer (2002), 143-154.
7. Paternò, F., Santoro, C., and Spano, L. D. Engineering the authoring of usable service front ends. *Journal of Systems and Software*, vol. 84 (2011), 1806-1822.
8. The Context of Use: http://www.w3.org/2005/Incubator/model-based-ui/XGR-mbui/#the-context-of-use
9. Yuan, Y., Li, B., and Kreger, H. SOA Reference Architecture: Standards and Analysis. In *Proc. of SmartCom 2016*, Springer (2017), 469-476.