An Internet-based Architecture Supporting Ubiquitous Application User Interfaces

Heiko Desruelle[1], Simon Isenberg[2], Dieter Blomme[1], Krishna Bangalore[3], and Frank Gielen[1]

[1] Ghent University – iMinds,
Dept. of Infomation Technology – IBCN, Ghent, Belgium
{heiko.desruelle, dieter.blomme, frank.gielen}@intec.ugent.be
[2] BMW Forschung und Technik GmbH, Berlin, Germany
simon.isenberg@bmw.de
[3] Technische Universität München, Garnich, Germany
krishna.bangalore@in.tum.de

Abstract. Maintaining a viable balance between development costs and market coverage has turned out to be a challenging issue when developing mobile software applications. The diversity of devices running third-party developed software applications is rapidly expanding from PC, to mobile, home entertainment systems, and even the automotive industry. With the help of Web technology and the Internet infrastructure, ubiquitous applications have become a reality. Nevertheless, the variety of presentation and interaction modalities still limit the number of targetable devices. In this chapter we present webinos, a multi-device application platform founded on the Future Internet infrastructure. Hereto we describe webinos' model-based user interface framework as a means to support context-aware adaptiveness for applications that are executed in such ubiquitous computing environments.

Keywords: ubiquitous web; model-driven user interfaces; adaptation

1 Introduction

The diversity of personal computing devices is increasing at an incredible pace. In result, people are often using a multitude of consumer electronic devices that have the ability to run third-party developed applications. Such devices can range from desktop PC, to mobile and tablet devices, to home entertainment and even in-car units. However, the fragmentation of devices and usage contexts makes it for applications particularly difficult to target a broad segment of devices and end-users. From a development perspective, the greatest common denominator amongst the available multi-device approaches is the Web. By adopting the Web as an application platform, applications can be made available whenever and wherever the user wants, regardless of the device type that is being used.

Nevertheless these clear advantages, existing Web application platforms are generally founded on the principles of porting traditional API support and operating system aspects to the Web. The evolution towards large-scale distributed service access and sensor usage is often not supported [1]. In result, the true immersive nature of ubiquitous web applications is mostly left behind. To enable developers to set up Web applications and services that fade out the physical boundaries of a device, the webinos platform has been proposed. Webinos is a virtualized application platform that spans across the various Web-enabled devices owned by an end-user. Webinos integrates the capabilities of these devices by seamlessly enabling the distribution of API requests.

In this chapter, we elaborate on the webinos platform's innovation and in particular its ability to dynamically adapt application user interfaces to the current delivery context. The remainder of this chapter is structured as follows. Related work and background on adaptive user interfaces are covered in Section 2. Section 3 provides a high-level introduction to the webinos platform, as well as a more in-depth discussion of its adaptive user interface approach. Section 4 highlights the proposed approach via a use case on dynamic adaptation of an application's navigation structure. Finally, our conclusions and future work are presented in Section 5.

2 Model Based User Interfaces

Model driven engineering (MDE) aims to accommodate with high-variability aspects of software systems. This development methodology is characterized by the separation of concerns, as it embodies a well accepted technique to reduce the engineering complexity of a software system [2]. A vast number of Web engineering approaches incorporate partial support for model-based development (e.g., UWE, WSDM, HERA, WebML, etc.). With a model driven engineering approach, software development is started with an abstract platform independent model (PIM) specification of the system [3]. A transformation model is in turn applied to compile the PIM to a platform-specific model (PSM). The transformation process is at the heart of the methodology's flexibility. For this purpose, MDE can use transformation languages such as the Query-View-Transformation standard (QVT) or the ATLAS Transformation Language (ATL) for specifying model-to-model transition rules [4].

Recent research on model driven engineering has been particularly active in the domain of user interface (UI) engineering. The CAMELEON Reference Framework (CRF) defines an important foundation for this type of approaches [5]. The framework specifies a context-sensitive user interface development process, driven by an intrinsic notion of the current user context, the environment context, as well as the platform context. According to the CRF approach, an application's user interface development consists of multiple levels of abstraction. Starting from an abstract representation of the interface's task and domain model, a PSM of the user interface is subsequently generated by means of a chained model transformations based on contextual knowledge. A number of major UI languages have adopted CRF, e.g., UsiXML [6], and MARIA [7]. Moreover, the World Wide Web Consortium (W3C) charted the Model-Based UI Working Group (MBUI-WG) as part of its Ubiquitous Web Activity (UWA) to investigate the standardization of context-aware user interface authoring [8]. Its goal is to work on standards that enable the authoring of context-aware user interfaces for web applications. The MBUI-WG aims to achieve this type of adaptivity by means of a model driven design approach. In this context, the semantically structured aspects of HTML5 will be used as key delivery platform for the applications' adaptive user interface.

The CAMELEON Reference Framework, more specifically, relies on a model driven approach and structures the development of a user interface into four subsequent levels of abstraction:

– Specification of the task and domain model, defining a user's required activities in order to reach his goals.
– Definition of an abstract user interface (AUI) model. The AUI model defines a platform-independent model (PIM), which expresses the application's interface independently from any interactors or modalities within the delivery context's attributes.
– Definition of a concrete user interface (CUI) model, a platform-specific model (PSM) which generates a more concrete description of the AUI by including specific dependencies and interactor types based on the delivery context.
– Specification of the final user interface (FUI), covering the code that corresponds with the user interface in its runtime environment (e.g., HTML, Java, etc.).

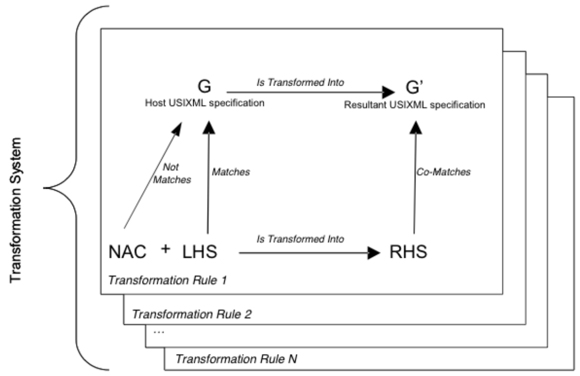

Fig. 1: Model-to-model transformation approach for the adaptation of a model-based user interface [6]

As documented by Schaefer, various approaches can be used to express the adaptation of a model-based user interface [9]. In essence, three types of adaptation approaches can be distinguished: model-to-model transformations, transformations on the XML representation of models, and code transformations. The model-to-model approach relies on the fact that most MBUI models can be designed based on a directed graph structure. In result, adaptations between two models are specified with model mappings by means of graph transformation rules. As depicted in Fig. 1, transformation rules consist of a Left Hand Side (LHS) condition matching the current UI model represented by graph G [6]. To add expressiveness, one or more Negative Application Condition (NAC), which should not match G, can be defined. Based on the matching of these conditions a Right Hand Side (RHS) defines the transformation result by replacing LHS occurrence in G with RHS. This substitution operation results in an adapted UI model represented by graph G'.

Furthermore, for UI models represented with XML, XSLT transformations can be used as a more declarative way to define adaptations [10]. The transformation process takes a XML based document as input together with an XSLT stylesheet module containing the transformation rules. Each transformation rule consists of a matching pattern and an output template. Patterns to be matched in the input XML document are defined by a subset of the XPath language [11]. The output after applying the appropriate transformations can be standard XML, but also other formats such as (X)HTML, XSL-FO, plain text, etc.

3 Multi-device Adaptive User Interfaces

3.1 The webinos Platform

To enable application developers to set up services that fade out the physical boundaries of a device, we propose the webinos platform. Webinos defines a federated Web application platform and its runtime components are distributed over the devices, as well as the cloud. Fig. 2 depicts a high-level overview of the platform's structure and deployment. The system's seamless interconnection principle is cornered around the notion of a so called Personal Zone.

The Personal Zone represents a secure overlay network, virtually grouping a user's personal devices and services. To enable external access to and from the devices in this zone, the webinos platform defines a centralized Personal Zone Hub (PZH) component. Each user has his own PZH instance running in the cloud. The PZH is a key element in this architecture, as it contains a centralized repository of all devices and contextual data in the Personal Zone. The PZH keeps track of all services in the zone and provides functionality to enable their discovery and mutual communication. This way, the PZH facilitates cross-device interaction with personal services over the Internet. Moreover, PZHs are federated, allowing applications to easily discover and share data and services residing on other people's devices. Webinos achieves this structure by incorporating two service discovery abstraction mechanisms. On a local level, webinos supports various fine-grained discovery techniques to maximize its capability to detect devices and services (e.g., through multicast DNS, UPnP, Bluetooth discovery, USB discovery, RFID/NFC, etc.). Secondly, on a remote level, the local discovery data are propagated within the Personal Zone and with authorized external

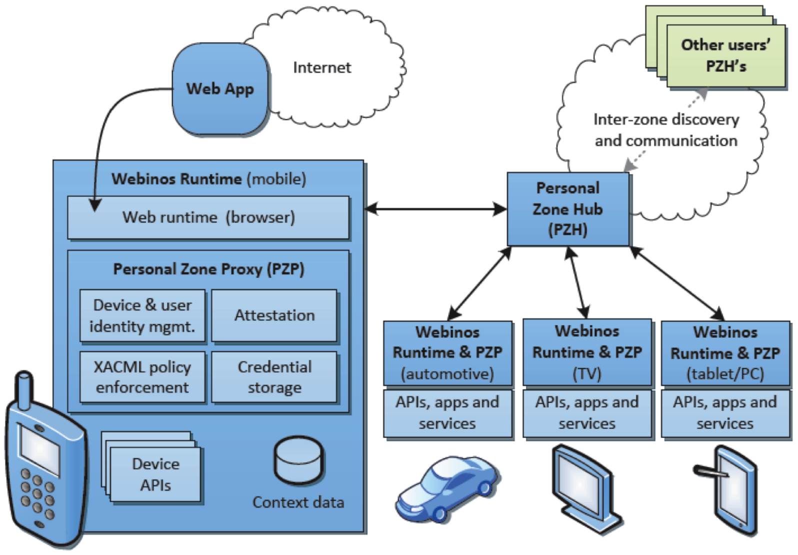

Fig. 2: High-level overview of webinos' ubiquitous application platform

PZHs. Based on webinos' aim for flexible Personal Zones in terms of scalability and modifiability, the overlay network is designed in line with the callback broker system pattern [12]. With this pattern, the availability of locally exposed services is communicated throughout the Personal Zone via a service broker in the PZH. Moreover, the platform's high-level communication infrastructure is founded on a direct handle tactic via JSON-RPC (JavaScript object notation - remote procedure call), which is invoked over HTTP and WebSockets.

On the device-side, a Personal Zone Proxy (PZP) component is deployed. The PZP abstracts the local service discovery and handles the direct communication with the zone's PZH. As all external communication goes through the PZP, this component is responsible for acting as a policy enforcement point and managing the access to the device's exposed resources. In addition, the PZP is a fundamental component in upholding the webinos platform's offline usage support. Although the proposed platform is designed with a strong focus on taking benefit from online usage, all devices in the Personal Zone have access to a locally synchronized subset of the data maintained by the PZH. The PZP can thus temporarily act in place of the PZH in case no reliable Internet connection can be established. This allows users to still operate the basic functionality of their applications even while being offline and unable to access the Internet. Through communication queuing, all data to and from the PZP are again synchronized with the PZH as soon as the device's Internet access gets restored.

The Web Runtime (WRT) represents the last main component in the webinos architecture. The WRT can be considered as the extension of a traditional Web render engine (e.g., WebKit, Mozilla Gecko). The WRT contains all necessary components for running and rendering Web applications designed with standardized Web technologies: a HTML parser, JavaScript engine, CSS processor, rendering engine, etc. Furthermore, the WRT maintains a tight binding with the local PZP. The WRT-PZP binding exposes JavaScript interfaces, allowing the WRT to be much more powerful than traditional browser-based application environments. Through this binding, applications running in the WRT are able to securely interface with local device APIs and services. In addition, the PZP also enables the runtime to connect and synchronize with other devices in the Personal Zone through its binding with the PZH.

3.2 Adaptive User Interface Framework

Within webinos, the process for adapting user interfaces to the active delivery context is regulated by the local PZP. For this particular purpose, the PZP contains an Adaptation Manager component (see Fig. 3). The Adaptation Manager aggregates all available adaptation rules, analyzes them, and feeds them to a Rule Engine for evaluation. In turn, the Rule Engine aims to match the applicability of each rule by comparing its conditions with the context data exposed by the PZP's internal services. Once an applicable rule is identified, the adaptation process is fired by sending its transformation instruction to the Web runtime. Moreover, by applying the RETE algorithm for executing the reasoning within the Rule Engine, the worst case computational complexity of this process remains linear

$$O\left(R \cdot F^{C}\right) , \tag{1}$$

with R the average number of rules, F the number of facts in the knowledgebase that need to be evaluated, and C the average number of conditions in each rule [13].

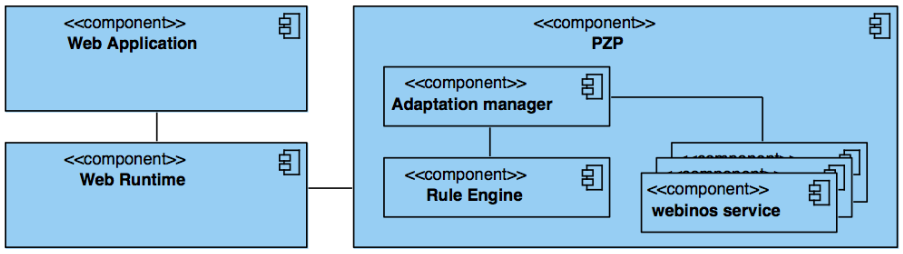

Fig. 3: Component diagram of webinos' UI adaptation framework

In order to accommodate webinos with support for dynamically triggered adaptations based on at runtime contextual changes, the used rule syntax complies with the Event Condition Action (ECA) format. The structure of an ECA rule consists of three main parts.

$$\text{on } [\textit{event}] \text{ if } [\textit{conditions}] \text{ do } [\textit{action}] \qquad (2)$$

The event part specifies the system signal or event that triggers the invocation of this particular rule. The conditions part is a logical test that, if evaluated to true, causes the rule to be carried out. Lastly, the action part consists of invocable JavaScript instructions on the resource that needs adaptation. In result, adaptation requirements can be expressed in terms of events or other context changes which might occur during the application's life cycle (see the case study in Section 4 for elaborating examples on adaptation rules).

For each ECA rule, the Adaptation Manager analyzes the rule's trigger event. Based on the event type, it subsequently feeds the rule to a dedicated instance of the Rule Engine, which is registered with the appropriate webinos services for a callback so it can launch the condition evaluation when the event occurs. For rules triggered by the *application.launch* event, the Rule Engine instance is initiated right away. The active instance matches the rules' conditions based on context properties fetched from webinos services such as the Device Status API [14].

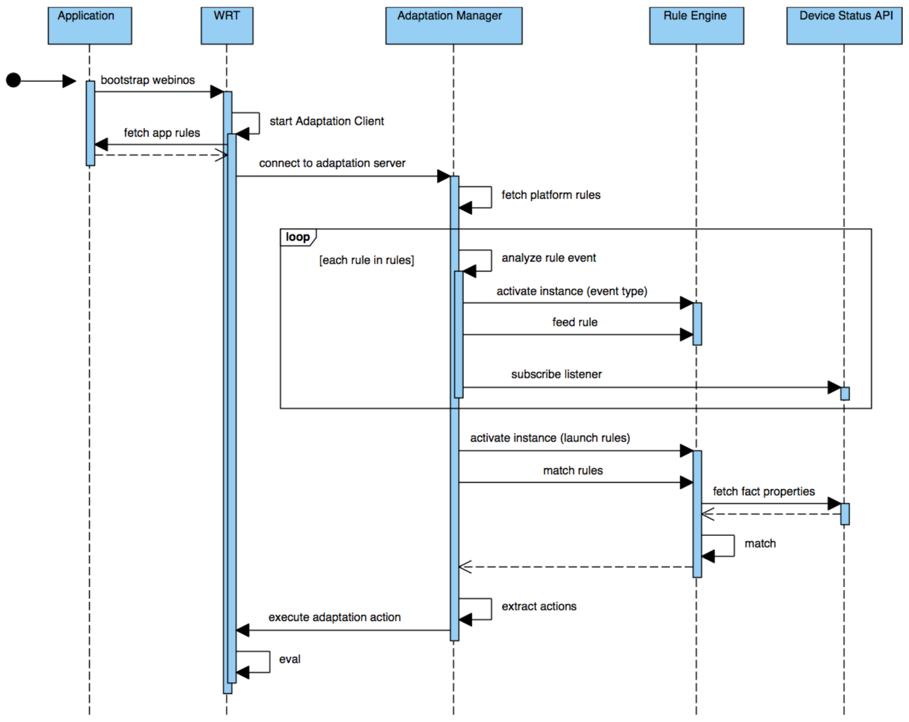

Fig. 4: Sequence diagram for the lookup of applicable UI adaptation rules at application launch

The sequence diagram in Fig. 4 provides a more detailed overview of how the adaptation process is handled. By bootstrapping webinos at the launch of an application, a communication interface is established between the Web runtime environment and the local PZP. This interface allows for the injection of an Adaptation Client component in the WRT. The Adaptation Client executes all the UI adaptation instructions it receives from the PZP's Adaptation Manager. As the Adaptation Client runs within the WRT, it has access to the application's DOM (Document Object Model) [15]. Moreover, this component is thus able to access and adapt the application's content, structure and style via the manipulation of DOM structures and properties.

4 Case Study: Adaptive Navigation Bar

This section elaborates on a case study for using webinos' UI framework to dynamically adapt the presentation of an application's navigation structure. For this adaptation case study, the HTML skeleton code in Listing 1.1 will serve as a sample application. This basic application is semantically enhanced with HTML element attributes to guide the adaptation process. The application skeleton contains a navigation menu component and a number of application specific subviews. As shown in Fig. 5 and Fig. 6, the presentation of this application's navigation component can be optimized based on various parameters such as the device's operating system, input modalities screen size, screen orientation, etc. Taking these contextual characteristics into account is necessary in order to ensure the adaptive usability requirements of a multi-device ubiquitous application, but also, e.g., for meeting existing safety regulations regarding user distraction in-vehicle applications [16].

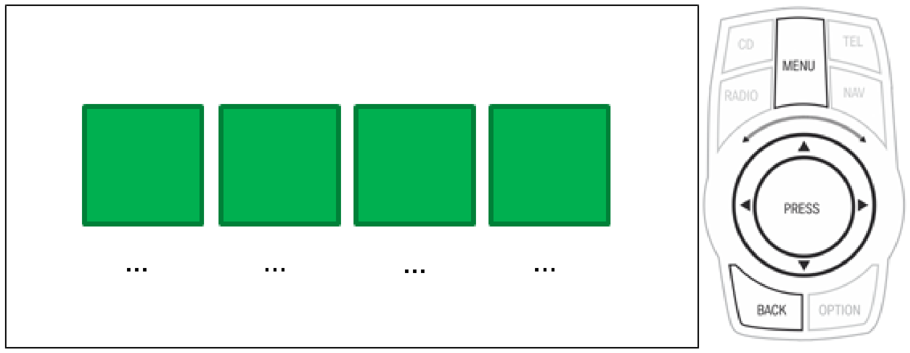

Fig. 5: Application navigation bar adaptation for an in-vehicle infotainment setup

For such an in-vehicle infotainment (IVI) system, adaptation rules can be set to display the application's navigation bar fullscreen (see rule in Listing 1.2). This can be done on application startup (i.e., *application.launch* event trigger combined with an IVI-based system as rule condition). All other UI elements are hidden to further decrease the risk for user distraction. Moreover, based on the specific interaction modalities provided by an IVI system, displaying the application's navigation bar can also be triggered by pressing the MENU button on its controller module. The interaction controller depicted in Fig. 5 is BMW's iDrive controller, which internally maps to the combination of a jog dial and four-way scroller device.

Listing 1.1: Sample HTML application skeleton

```
<body>
   <div class="menu">
      <!-- list menu items -->
   </div>
   <div class="page" id="home">
      <!-- home screen content -->
   </div>
   <div class="page" id="settings">
      <!-- settings screen content-->
   </div>
   ...
</body>
```

Listing 1.2: Vehicular adaptation rule

```
<rule description="vehicular menu">
   <event>application.launch</event>
   <condition>device.type == "ivi"</condition>
   <action>
      <!-- spread menu items over the headunit's screen -->
      <!-- map and link iDrive controller buttons -->
      <!-- hide all page elements -->
   </action>
</rule>
```

However, when accessing the same application from a mobile or tablet device, other presentation and interaction requirements come into play. The case depicted in Fig. 6 provides an adaptation example based on changes a device's screen orientation (i.e., landscape or portrait mode). In the event of a touchscreen device that is being rotated to landscape mode, adaptation rules are set to transform the navigation bar in a vertically organized list that is moved to the lefthand side of the display. Moreover, on the right side of the screen only one page element is shown. All other page elements can be accessed via its appropriate link in the navigation bar (see rule in Listing 1.3). In case the device is rotated to portrait mode, the navigation bar is reduced to a collapsible UI element located on the top of the screen. Finally, a running multi-device prototype of this case is depicted in Fig. 7.

Listing 1.3: Touch-based adaptation rule

```
<rule description="touch−based menu landscape">
    <event>device.orientationchange</event>
        <condition>
            device.inputtype == "touchScreen" &&
            screen.orientation == "landscape"
        </condition>
    <action>
        <!−− resize menu items to fit the screen's height −−>
        <!−− move menu to the lefthand side −−>
        <!−− hide all page elements but the active −−>
    </action>
</rule>
```

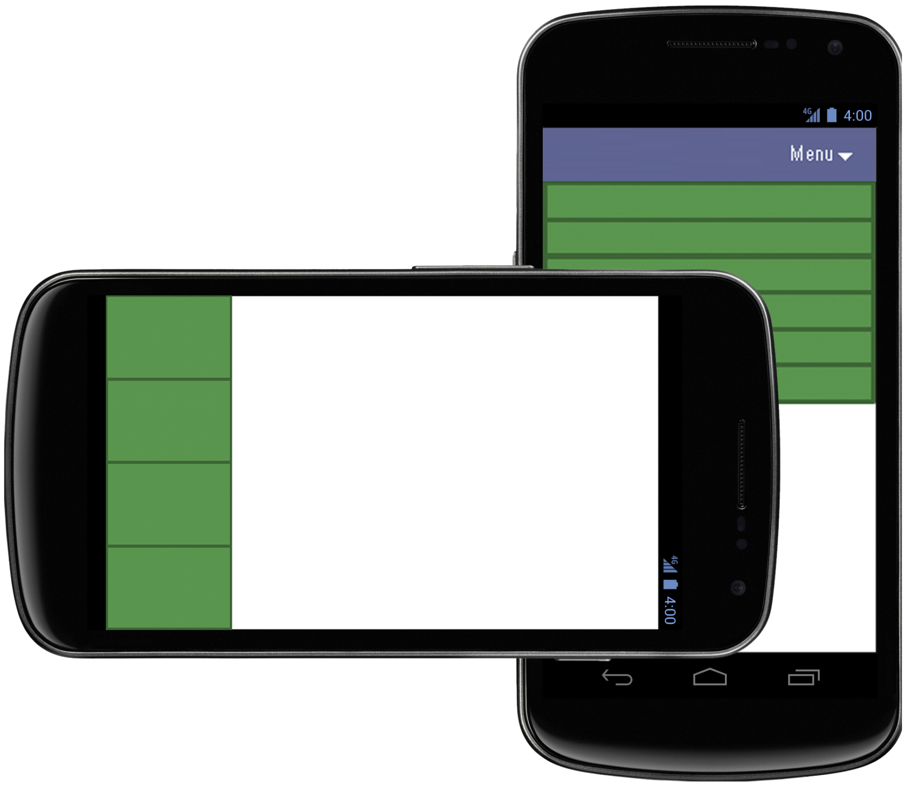

Fig. 6: Application navigation bar adaptation for mobile and tablet devices based on screen orientation

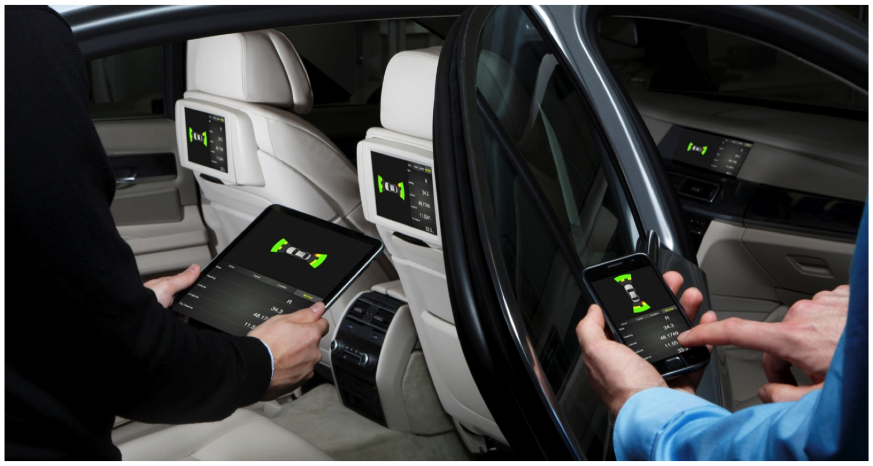

Fig. 7: Running multi-device prototype of the application navigation case

5 Conclusion

In this chapter we presented the webinos application platform and its aim to enable immersive ubiquitous software applications through adaptive user interfaces. Webinos does so by leveraging the fundamental cross-platform opportunities offered by the Web as well as the Future Internet infrastructure. With the introduced Personal Zone concept, applications developers are enabled to create software that transcends the executing device's physical boundaries by simultaneously accessing the capabilities of multiple devices. In order to ensure users a comparable and intuitive quality in use throughout all their devices, the presentation and interaction modalities of the applications' user interface can be adapted accordingly. Based on the contextual knowledge available in the webinos Personal Zone, rule-based adaptation decisions can be made as a means to dynamically optimize the applications user interfaces to the executing device's characteristics.

The developed webinos technology aims to influence the Future Internet architecture and its related frameworks. Collaboration has hereto been established with various Future Internet projects such as FI-WARE and FI-CONTENT. Future work for the webinos project includes further exploring the possibility to use the webinos platform as a generic enabler for these initiatives and to seamlessly connect ubiquitous devices on an application level [17].

Acknowledgments. The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7-ICT-2009-5, Objective 1.2) under grant agreement number 257103 (webinos project). The authors thank all Webinos project partners, as this chapter draws upon their work.

References

1. Desruelle, H., Lyle, J., Isenberg, S., Gielen, F.: On the challenges of building a Web-based ubiquitous application platform. In: 14th ACM International Conference on Ubiquitous Computing, pp. 733–736. ACM, New York (2012)
2. Dijkstra, E.W.: A Discipline of Programming. Prentice Hall, Englewood Cliffs (1976)
3. Moreno, N., Romero, J.R., Vallecillo, A.: An overview of model-driven Web engineering and the MDA. In: Rossi, G., Pastor, O., Schwabe, D., Olsina, L. (eds.) Web Engineering: Modelling and Implementing Web Applications. Human-Computer Interaction Series, pp. 353–382. Springer, London (2008)
4. Koch, N.: Classification of Model Transformation Techniques used in UML-based Web Engineering. Software, IET. 1:3, 98–111 (2007)
5. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., Vanderdonckt, J.: A Unifying Reference Framework for Multi-Target User Interfaces. Interacting with Computers. 15, 289–308 (2003)
6. Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., Lopez-Jaquero, V.: USIXML: A language supporting multi-path development of user interfaces. In: Bastide, R., Palanque, P., Roth, J. (eds.) EHCI-DSVIS 2005. LNCS, vol. 3425, pp. 200–220. Springer, Heidelberg (2005)
7. Paterno, F., Santoro, C., Spano, L.D.: MARIA: A universal, declarative, multiple abstraction-level language for service-oriented applications in ubiquitous environments. ACM TOCHI. 16:4, 19 (2009)
8. Cantera, J.M. (ed.): Model-Based UI XG Final Report. W3C Incubator Group Report (2010), `http://www.w3.org/2005/Incubator/model-based-ui/XGR-mbui`
9. Schaefer, R.: A Survey on Transformation Tools for Model Based User Interface Development. In: Jacko, J.A. (ed.) HCII 2007. LNCS, vol. 4550, pp. 1178–1187. Springer, Heidelberg (2007)
10. Kay, M. (ed.): XSL Transformations (XSLT) Version 2.0. W3C Recommendation (2007)
11. Berglund, A. Boag, S., Chamberlin, D., Fernandez, M.F., Kay, M., Robie, J., Simeon, J. (eds.): XML Path Language (XPath) 2.0 (Second Edition). W3C Recommendation (2010)
12. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M.: Pattern-oriented software architecture: A system of patterns. John Wiley & Sons, West Sussex (2001)
13. Forgy, F.: On the efficient implementation of production systems. PhD thesis, Carnegie-Mellon University (1979)
14. Webinos Device Status API, `http://dev.webinos.org/specifications/new/devicestatus.html`
15. Le Hors, A., Le Hegaret, P., Wood, L., Nicol, G., Robie, J., Champion, M., Byrne, S. (eds.): Document Object Model (DOM) Level 3 Core Specification. W3C Recommendation (2004)
16. Faily, S., Lyle, J., Paul, A., Atzeni, A., Blomme, D., Desruelle, H., Bangelore, K.: Requirements sensemaking using concept maps. In: Winckler, M., Forbig, P., Bernhaupt, R. (eds.) HCSE 2012. LNCS, vol. 7623, pp. 217–232. Springer, Heidelberg (2012)
17. Allott, N.: Collaboration Opportunities: FIWARE and webinos, `http://www.webinos.org/blog/2012/11/14/collaboration-opportunities-fiware-and-webinos/`