Univ Access Inf Soc (2016) 15:5–19
DOI 10.1007/s10209-014-0373-0

LONG PAPER

Accessible user interface support for multi-device ubiquitous applications: architectural modifiability considerations

Heiko Desruelle · Simon Isenberg · Andreas Botsikas · Paolo Vergori · Frank Gielen

Published online: 17 August 2014

Abstract The market for personal computing devices is rapidly expanding from PC, to mobile, home entertainment systems, and even the automotive industry. When developing software targeting such ubiquitous devices, the balance between development costs and market coverage has turned out to be a challenging issue. With the rise of Web technology and the Internet of things, ubiquitous applications have become a reality. Nonetheless, the diversity of presentation and interaction modalities still drastically limit the number of targetable devices and the accessibility toward end users. This paper presents webinos, a multi-device application middleware platform founded on the Future Internet infrastructure. Hereto, the platform's architectural modifiability considerations are described and evaluated as a generic enabler for supporting applications, which are executed in ubiquitous computing environments.

Keywords Ubiquitous web · Multi-device applications · Model-driven user interfaces · Dynamic adaptation

H. Desruelle (✉) · F. Gielen
Department of Information Technology (INTEC), Ghent University – iMinds, Ghent, Belgium
e-mail: heiko.desruelle@intec.ugent.be

S. Isenberg
BMW Forschung und Technik GmbH, Munich, Germany

A. Botsikas
Department of Electrical and Computer Engineering, National Technical University of Athens (NTUA), Athens, Greece

P. Vergori
MultiLayer Wireless solutions (MAIN), Istituto Superiore Mario Boella (ISMB), Turin, Italy

1 Introduction

A series of new generation human–computer interaction paradigms such as mobile and ubiquitous computing are enabling software applications and services, which execute on a wide variety of consumer electronic devices. These devices currently range from desktop and laptop computers, to mobile and tablet devices, to TV and home entertainment systems, and to in-car devices. Nevertheless, the fragmentation of devices and usage contexts makes it particularly difficult to target a broad segment of devices and end users. In this context, the use of web technology can provide a standardized abstraction layer for applications to execute device independently. 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.

Despite 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 toward large-scale distributed service access and sensor usage is often not supported [10]. 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 service requests.

This paper elaborates on the webinos platform's innovation and in particular its ability to dynamically adapt application user interfaces to the current delivery context and thus optimize the end user's accessibility. The remainder of this

article is structured as follows. Related work and background on adaptive software engineering and user interfaces are covered in Sect. 2. Section 3 provides a high-level introduction to the webinos platform and elaborates on the platform's architectural decisions for meeting its adaptability requirements. Section 4 covers an in-depth discussion of webinos' adaptive user interface support and highlights the proposed approach via a case study on dynamic adaptation of an application's navigation structure. Section 5 quantitatively measures and evaluates the platform's adaptability qualities. Moreover, the qualitative evaluation of existing webinos prototype applications aiming for multi-device accessibility is discussed. Finally, conclusions and future work are presented in Sect. 6.

2 Background and related work

2.1 Modifiability in software architecture

Modifiability has always been an important concept in software engineering. By supporting this quality, software architects aim to prepare a system for change requirements after its initial release [8]. Software constantly tends to evolve, from the addition of features, to the support for new technology platforms. As a result, modifiability is about minimizing the technical risks and cost impact of such changes. In order to achieve modifiability as a system quality, software architects need to envision and incorporate modifiability support in the system's design cycle.

Through the years, a considerable number of best practices on architectural approaches have been designed to support the modifiability requirements of a system. In general, the modifiability quality of a system can be expressed in terms of cohesion and coupling [25]. Coupling measures the mutual association strength between the system's software components. Cohesion, on the other hand, is a measure for the number of internal relationships between the responsibilities of a software component. Based on the notion of cohesion and coupling, Bass et al. structured a set of architectural modifiability tactics. This set aims to guide software architects toward achieving the required modifiability qualities for their system [2]. As depicted in Fig. 1, the proposed architectural design decisions can be devised in three high-level categories, i.e., increasing cohesion, reducing coupling, and deferring binding.

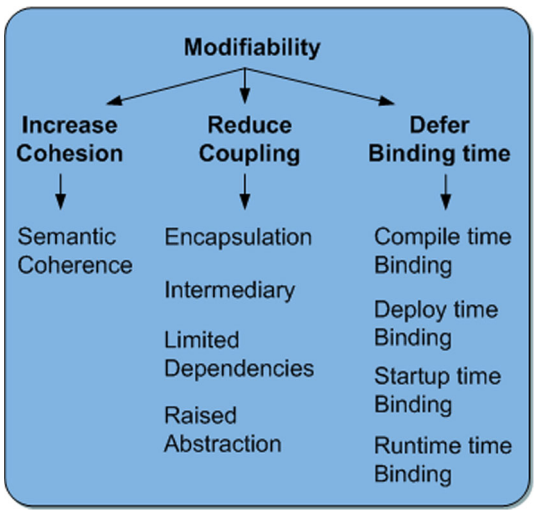

Fig. 1 Organizing architectural tactics for modifiability (derived from Bass et al. [2])

Increasing cohesion tactics aims to deal with the number of internal responsibilities within each of the system's components. This is in order to prevent changes to one responsibility affecting the other responsibilities within the same component. As a tactic, increasing the semantic coherence is intended to stimulate a software architect to relocate one or more component responsibilities in case the internal responsibilities of that component do not serve the same purpose.

Tactics regarding the reduction of coupling aim to reduce the number of mutual relationships among the various components that shape the system. High coupling might result in changes to one component impacting one or more of its associated components as well. Reducing the coupling intends to prevent such change propagation by means of the following architectural decisions.

- *Encapsulation:* Each system component is to interact with other components through a well-defined yet abstract interface. With this kind of encapsulation, the coupling between associated components is limited to their exposed interfaces rather than the entire components.
- *Intermediary:* The use of an intermediary can be opted to break dependencies between system components. Depending on the type of dependency (i.e., location, identity, behavior, and creation), the intermediate can remove the explicit knowledge requirements from those components.
- *Raised abstraction:* In case multiple similar responsibilities exist within the system, abstraction can help to extract the generic part of the responsibility. This way, any change to the common part of the responsibility will only need to be handled in one component.

Finally, the possibility to defer the binding of components is mainly a result of applying and combining the above-mentioned tactics on coupling and cohesion. Depending on the system's exact modifiability requirement, binding can be designed to initiate at various points in the software life cycle. Ranging from compile time (build configurations

and parameterization, and aspect-oriented programming, etc.), at deploy and start-up time (configuration binding, resource files, etc.), or at runtime (runtime registration and binding, dynamic lookup, parameter interpretation, and polymorphism, etc.).

2.2 Model-based user interfaces

Model-driven engineering (MDE) aims to accommodate with high-variability aspects of software systems. This development methodology is characterized by a raised abstraction tactic via the separation of concerns throughout all the phases of software engineering (i.e., analysis, design, and implementation). This approach embodies a well-accepted technique to reduce the engineering complexity of a software system [11]. A vast number of Web engineering methods 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 [20]. 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 [16].

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 definition languages have adopted CRF, e.g., UsiXML [18], and MARIA [22]. 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 [6]. 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.

More specifically, the CAMELEON Reference Framework 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. At the lowest abstraction level, these models define 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.).

Figure 2 shows the interconnection principles and transformations between the above-mentioned abstraction levels. The downward arrows depict reification processes. Reification is the transformation from a higher-level abstraction to a lower-level abstraction phase, hence inferring a more concrete user interface description. The upward arrows, on the other hand, specify the abstraction processes. An abstraction operation is the inverse transformation of reification. The third transformation type is the translation, depicted by the horizontal arrows. The translation deals with adapting the UI description to changes in one of the contextual parameters (i.e., user, device, physical environment). In this case, the UI description is optimized to the context change, but its

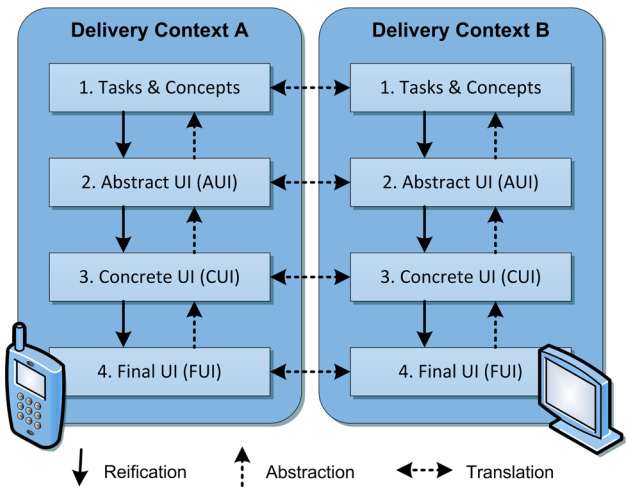

Fig. 2 Model-based user interface abstraction levels and transformations

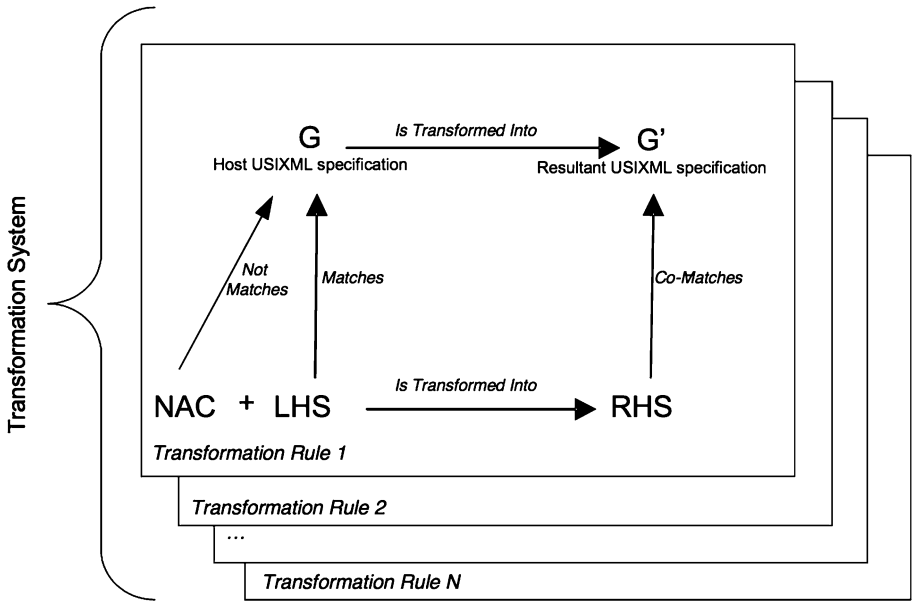

Fig. 3 Model-to-model transformation approach for the adaptation of a model-based user interface [18]

abstraction level remains the same when performing a translation.

As documented by Schaefer, various approaches can be used to express the adaptation of a model-based user interface [24]. 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. 3, transformation rules consist of a left-hand side (LHS) condition matching the current UI model represented by graph G [18]. To add expressiveness, one or more Negative Application Conditions (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 [15]. The transformation process takes an 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 [3]. The output after applying the appropriate transformations can be standard XML, but also other formats such as (X)HTML, XSL-FO, plain text, etc.

2.3 Ubiquitous application middleware

Various cross-device middleware platforms have previously been developed, aiming to create a platform-independent layer for running generic applications. The Java Virtual Machine (JVM) and the .NET framework are part of the most well-known and widespread solutions in this category, providing a common runtime and set of APIs in support of their "write once, run everywhere" philosophy. These solutions, however, are closed proprietary systems and mainly confined to PC operating systems [1].

With the increasing maturity of Web technology and the rise of mobile platforms, Web-based application middleware solutions have started to emerge. This type of middleware aims to leverage the popularity and market coverage of devices with built-in support for HTML, CSS, and JavaScript. Web widget runtimes such as Qt and full-featured mobile browsers such as Chrome for Android, Firefox mobile, and Mobile Safari have enabled Rich Internet Applications (RIAs), which need little to no modification to run on a wide variety of target devices [19]. Despite their clear benefits, these runtimes still focus on supporting localized application execution rather than enabling cross-device user experiences (e.g., multi-screen applications, remote service invocation.) [10].

The Global Public Inclusive Infrastructure (GPII) initiative has the goal to build an infrastructure for automated personalization of services based on a user's personal preferences and capabilities [27]. Based on this globally available profile, the presentation and interaction modalities of every accessed service and application are automatically adapted. The Cloud4All project, co-funded by the EU's FP7 programme, aims to support the creation of

such an infrastructure. The webinos middleware platform described in this paper has set a very similar goal. The main difference lies with the approach for storing profile data and accessing services. The current Cloud4All vision is based on a cloud-centric approach for profile storage and services access. Webinos, on the other hand, focuses on a distributed application platform, which allows the user to maintain control over personal data and services. With the general public's raising awareness regarding privacy and security, webinos only stores a reference in the cloud, while the actual data remain on the user's device.

3 The webinos platform

The webinos project aims to design and deliver an open source application platform that enables Web applications and services to be executed consistently over a broad range of Web-enabled devices. These connected devices include PC, mobile and tablet, home entertainment, and in-car units [9]. Moreover, webinos' "one application for every device" vision is not just limited to portability by enabling a single application to be executed on each of the targeted device groups. Webinos particularly aims to also simultaneously leverage all the specific capabilities of one's owned devices within that application. For example, in an in-car setup this could include accessing your vehicle's sensor data for a parking assistance application running on a smartphone or tablet device.

These modifiability aspects lay out a considerable number of dynamic change requirements for the webinos application platform to adapt to. This section presents the modifiability tactics that were applied to webinos' architectural design for coping with these requirements and constraints. The interested reader can refer to [13] [23] for a more elaborate background discussion on the exact requirement scenarios as well as an overview of the platform's complete architectural structure.

3.1 Platform portability

An important driver for designing the webinos platform is its device independence support for running applications. A webinos application should be executable on each of the targeted device domains (i.e., desktop, mobile, home entertainment, in-car, and embedded devices), without requiring any modifications to the actual application. On an architectural level, webinos addresses this portability requirement by deferring binding time through an instruction set intermediary. With this virtual machine approach, application instructions are translated at runtime into instructions for the underlying technology platform. The webinos applications' code is thus only interpreted and

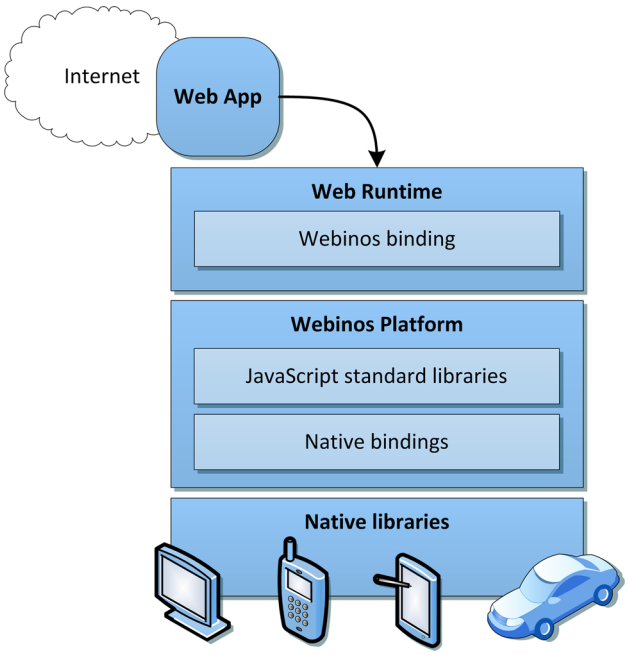

Fig. 4 Virtual machine approach for meeting webinos' portability requirements

bound at runtime. The application platform does so by leveraging broadly accepted and standardized Web technology including HTML, CSS, and JavaScript. As depicted in Fig. 4, various modifiability tactics have been incorporated at this level. An encapsulation tactic is applied to reduce the number of exposed interfaces to a set of well-defined JavaScript-based Web APIs (Application Programming Interfaces). Existing Web runtime (WRT) engines with HTML5 support can in turn hook into these APIs to allow their Web applications to interact with the webinos platform and access its functionality. In turn, webinos can remain independent from the WRT used to run the Web application. The WRT can thus be a browser (e.g., Mozilla Gecko for Firefox, WebKit, Google blink for Chrome.), as well as a hybrid WRT solution, which packages the Web application as a native app or widget (e.g., PhoneGap).

Moreover, an intermediary tactic provides the at-runtime binding between the webinos applications' instructions in JavaScript and the associated native instructions for the devices' underlying operating systems. In result, the device-dependent platform code is clearly separated from webinos' device-independent standard libraries and APIs.

Webinos aims to support a wide range of devices. Each of these devices will have its particular set of APIs and services (e.g., based on the available sensors and actuators). In order to enable webinos to leverage the full potential of its supported devices, it needs to enable external developers to dynamically expose additional services as Web APIs. Webinos does so by applying an encapsulation tactic to package APIs into modules and by

deferred the binding time of these packages. In result, external developers can implement and deploy additional webinos-enabled APIs.

3.2 Dynamic device and service binding

In addition to supporting portable applications, webinos aims to facilitate the development of applications for multi-device interaction and service usage. For webinos to seamlessly dispatch service requests to the most suited physical device, the platform needs to keep track of all devices owned by each individual end user. To do so, webinos relies on two abstraction mechanisms for service discovery. The design decisions reflecting this approach are based on semantic cohesion, a service intermediary tactic, encapsulation, deferred binding, and raised abstraction.

On a local level, webinos encapsulates the various fine-grained discovery techniques offered by the underlying devices' operating systems and exposes them via an abstract discovery API. This includes service discovery through, e.g., multicast DNS, UPnP, Bluetooth discovery, USB discovery, RFID/NFC, etc. Secondly, the local discovery data are propagated to a central repository residing in the cloud (see Fig. 5). This intermediary acts as a service broker, aiming to dissolve the strong binding between webinos applications and their executing device. The virtual overlay network created by such a service broker enables webinos applications to transparently call upon device services without requiring any explicit knowledge regarding to which physical device the request will be delegated. From the perspective of an application developer, webinos completely abstracts remote procedure calls (RPC) as if the functions are discovered and executed locally. This virtual overlay concept is internally referred to as the user's Personal Zone.

Within the platform, all available services and APIs are uniquely identified through a service-type URI (Unified Resource Identifier) with the following prefix for core APIs:

http://webinos.org/api/<webinos-api-name>

and the following prefix for APIs provided by external developers, respectively:

http://<dev-domain>/api/<external-api-name>

In addition to its own API set, webinos also supports the APIs defined by W3C's Device APIs Working Group [29]. These APIs are identified via the URI prefix.

http://webinos.org/api/w3c/<w3c-api-name>

The code snippet in Listing 1 demonstrates the deferred service binding for an access requests to webinos' core vehicular API. The vehicle API offers car-specific sensor data regarding the vehicle's engine, its climate control, the media system, etc. Access is requested via the API's associated URI. Webinos' discovery mechanism will in turn trigger the service broker to dynamically lookup the most suited registered device to handle such request. In turn, the broker returns the application a JavaScript callback function, which provides the at-runtime binding between the requested service-type and the selected device.

Although the webinos platform is designed with a primary focus on taking benefit from online usage, the highly

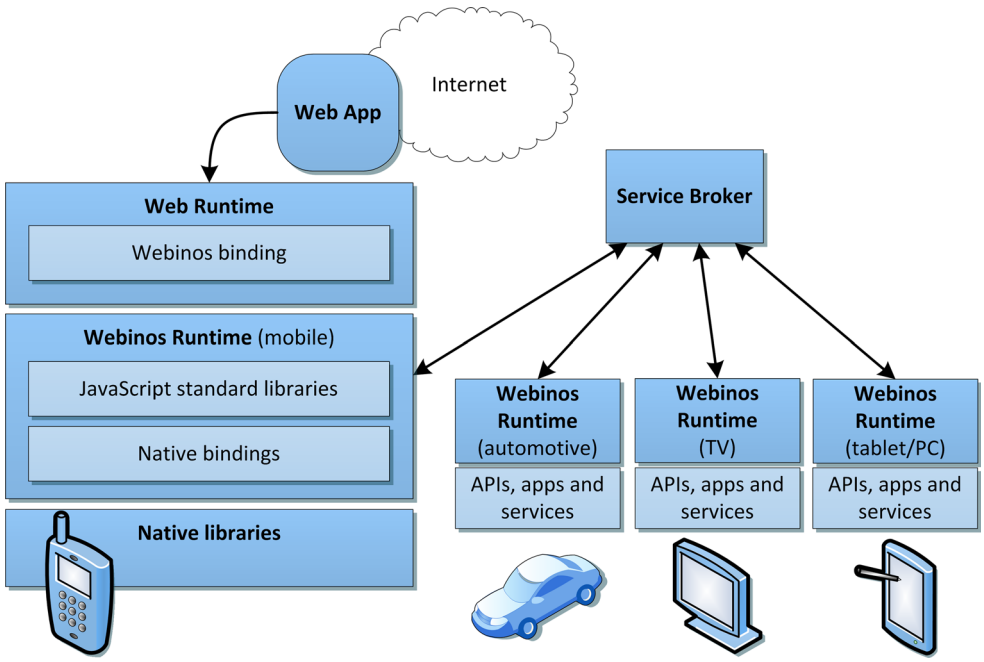

Fig. 5 Service broker approach for webinos' dynamic service binding

mobile nature of ubiquitous computing requires the platform to dynamically cope with temporary offline devices as well. This should allow users to still operate the basic functionality of their webinos applications even while being offline and unable to access the Internet. For this purpose, webinos' architectural design incorporates encapsulation and raised abstraction tactics. Each device running the webinos runtime can temporarily act in place of the service broker in case no reliable Internet connection can be established. The local webinos runtime does so by maintaining a synchronized copy of the service broker's repository, encapsulated as a cache within their communication interface. Through communication queuing, all data shared with the service broker is again synchronized as soon as the device's Internet access is restored.

Listing 1 Webinos service discovery

```
if ("webinos" in window)
{
   // get the vehicle sensor service
   webinos.findServices(
      "http://webinos.org/api/vehicle"
         ,
      function(service) {
           // execute actions
           ...
      }, null);
}
```

4 Multi-device adaptive user interfaces

For webinos to facilitate the development of accessible multi-device applications, the platform needs to accommodate developers with adaptive user interface support. This includes dynamic adaptability support for both the application's presentation, as well as its interaction modalities. Webinos does so by incorporating a model-driven engineering approach for its user interfaces (see Sect. 2.2). In order to minimize the learning curve for application developers, the platform uses standardized HTML as user interface definition language (UIDL). Based on a rule-driven mechanism, model-to-model transformations are dynamically executed to generate an optimal platform-specific model (PSM) of the user interface. The rules are a means for developers to express the contextual conditions in which certain action to the user interface should be taken. As a result, the supported process translates the developer's concrete user interface (CUI) definition based on the end user's active delivery context.

4.1 Webinos user interface framework

Within webinos, the user interface adaptation is regulated by each of the local webinos runtimes. For this particular purpose, the webinos runtime contains an adaptation manager component. The adaptation manager aggregates all available adaptation rules, analyzes them, and feeds them to a forward-chaining 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 runtime's internal services. Once an applicable rule is identified, the adaptation process is fired by sending the rule's transformation instruction to the Web runtime. In order to accommodate webinos with support for dynamically triggered adaptations based on at-runtime contextual changes, the implemented rule syntax complies with the Event Condition Action (ECA) format. The structure of an ECA rule consists of three main parts:

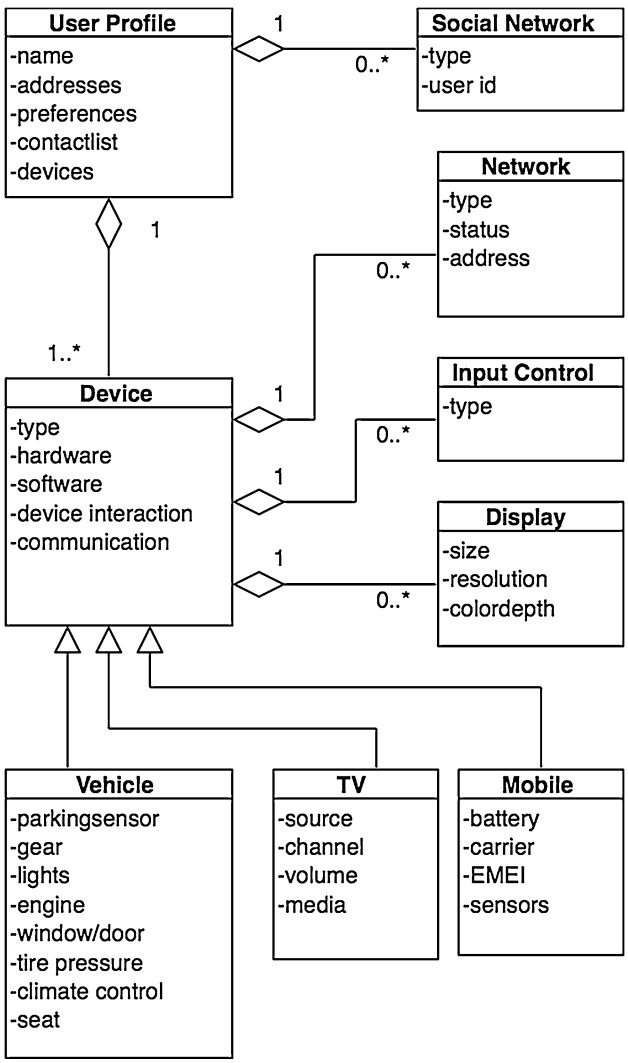

Fig. 6 Simplified representation of webinos' device and user context model

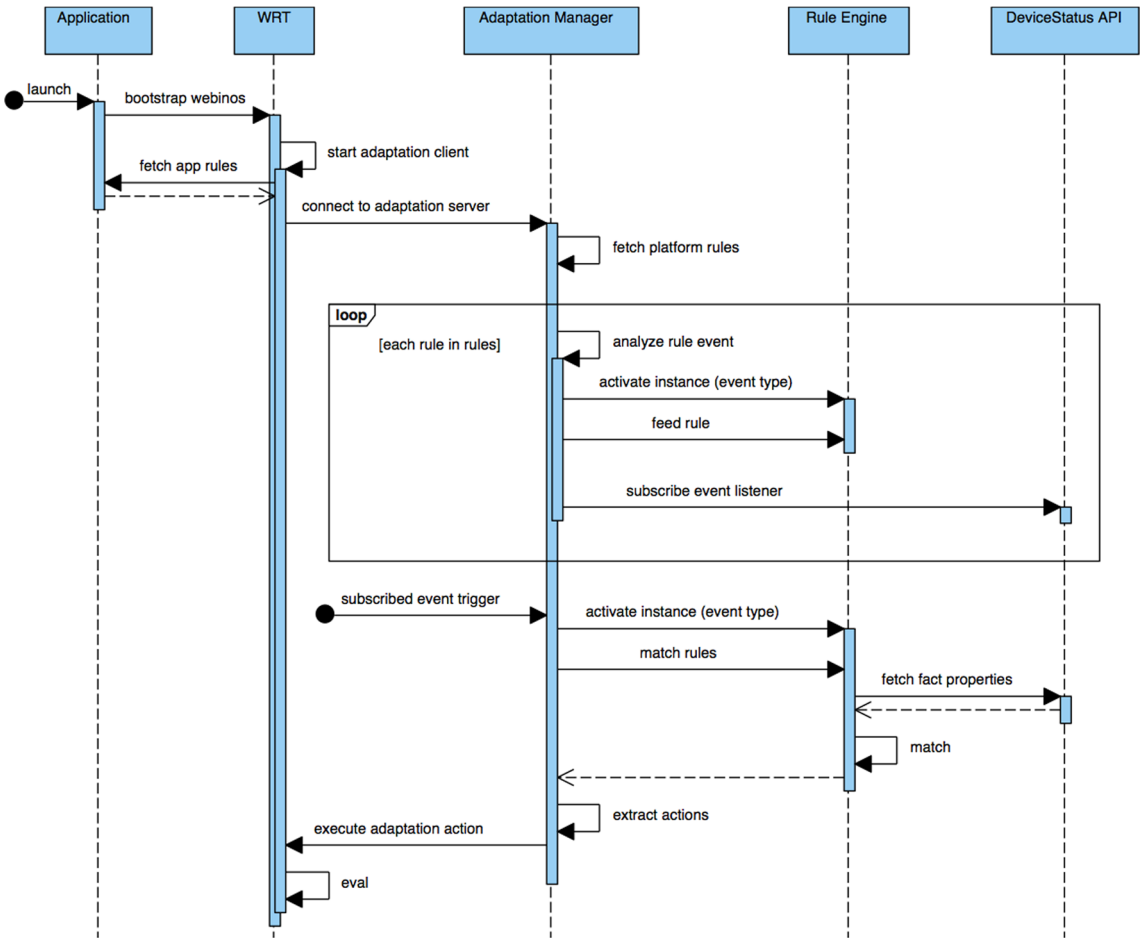

Fig. 7 Sequence diagram for the lookup of applicable UI adaptation rules at application launch

on [event] if [conditions] do [action] (1)

The event part specifies an internal webinos 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. This code is able to programmatically access and manipulate the Web application's user interface via the Document Object Model (DOM) [17]. The DOM is a W3C standard for representing and interacting with the objects in a HTML-based Web application. The model is maintained by the WRT and can be altered at runtime through scripting.

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. The reasoning within this instance is only triggered in case its associated system event occurs. As an instance is activated by its registered event, the engine starts matching its allocated rules' conditions. The evaluation is performed based on the meta-data fetched from webinos services such as the Device Status and Interaction API, Context API, Vehicle and TV API, and Contacts API [31]. These APIs provide a rich contextual at-runtime representation of the user and his or her devices. The standard context model aggregated from the available meta-data is depicted in Fig. 6. As described in Sect. 3, however, webinos provides an easily extensible API structure. In order to extend the available context model with more specific knowledge dimensions, developers are only required to implement and deploy additional APIs. In the context of accessibility, this can include API support for particular assistive hardware or software, or more elaborate user profiles, etc.

The sequence diagram in Fig. 7 provides a detailed overview of how the adaptation process is handled by the platform. By bootstrapping webinos at the launch of an application, a communication interface is established between the WRT environment and the local webinos platform. 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 webinos' adaptation manager. As the adaptation client runs within the WRT, it has access to the application's DOM. Hence, this component is able to access and adapt

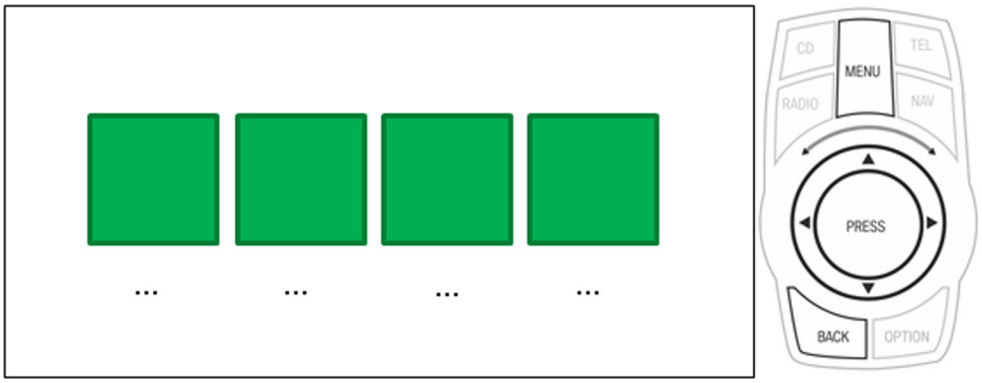

Fig. 8 Application navigation bar adaptation for an in-vehicle infotainment setup with BMW iDrive controller

the application's content, structure and style via the manipulation of DOM structures and properties. In result, webinos enables developers to express dynamic adaptation requirements for their Web-based applications in terms of runtime events and contextual conditions as they occur during the application's life cycle.

4.2 Case study: adaptive navigation bar

This section elaborates on a simple 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 2 will serve as a sample application. This basic application is semantically enhanced with HTML element attributes to guide the adaptation process. Developers can use any semantical structure for the annotation of their user interface objects and widgets. However, the use of the "role" attribute is recommended as specified by many accessibility guidelines such as the W3C WAI-ARIA candidate standard (Accessible Rich Internet Applications) [7]. The role attribute declares what a UI object does, rather than how it should be represented or how it should be interacted with. Hence, role-based semantics will be used for this case study, as it provides a good foundation for at-runtime interpretation and adaptation.

The presented application skeleton contains a menu component (navigation role) and a number of application-specific subviews (page role). As shown in Figs. 8 and 9, 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, available sensors, also based on the user's profile and preferences. Taking these contextual characteristics into account is necessary in order to ensure the adaptive usability requirements of a multi-device ubiquitous application, but, e.g., for meeting existing safety recommendations and regulations regarding user distraction by vehicular applications [10].

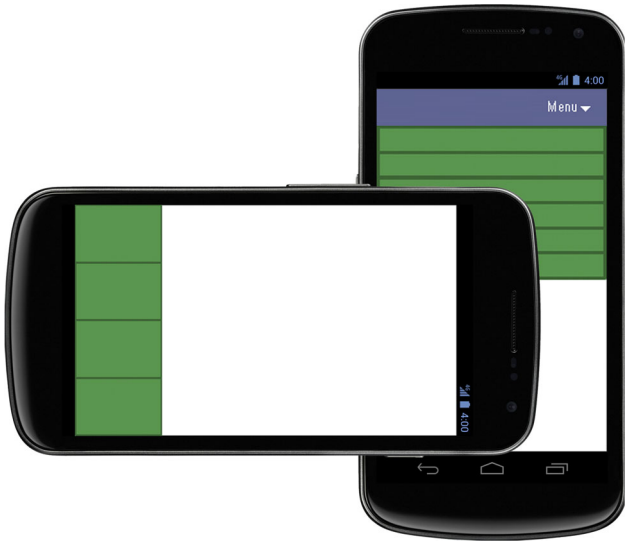

Fig. 9 Application navigation bar adaptation for mobile and tablet devices based on screen orientation

Listing 2 Sample HTML application skeleton

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

In-car systems are increasingly used to run various applications. Setups in this domain range from mountable navigation systems, to built-in dashboard units. Overall, the application user interface for such in-vehicle infotainment (IVI) systems needs to be clear and easy to use. To do so for the application's navigation bar, adaptation rules can be set to display the menu in

fullscreen mode with large buttons (see rule in Listing 3). Webinos can be instructed to execute this rule at the application's start-up (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 the 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. 8 is BMW's iDrive controller [4], which internally maps to the combination of a jog dial and four-way scroller device. Once a specific navigation item is selected, either via the touchscreen or with the hardware controller, the associated page item is unhidden and displayed as a dialog on top of the navigation bar.

Listing 3 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>
```

Listing 4 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 lefthand side
       -->
    <!-- hide all page elements but
       the active -->
  </action>
</rule>
```

On the other hand, when accessing the exact same application from a smartphone or tablet device, completely different presentation and interaction requirements can come into play. The case depicted in Fig. 9 provides an alternative adaptation example of the navigation bar based on the changes in 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 left-hand 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 the appropriate link in the navigation bar (see rule in Listing 4). 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.

5 Evaluation

In this section, an evaluation of the proposed platform and its flagship applications is presented. First, the platform's prototype implementation is discussed and its ability to meet the set of key modifiability requirements regarding platform independence and portability is quantitatively evaluated (as laid out in Sect. 3).

The second part of the analysis focuses on a qualitative impact evaluation of webinos' proof-of-concept applications. All selected applications focus on accessible multi-device functionality. The process starts by elaborating the applications' main use case scenarios. Moreover, a conducted impact evaluation for each of these webinos-enabled applications is discussed.

5.1 Platform modifiability evaluation

A prototype of the webinos platform is currently under development. All sources and documentation are available as open source resources [30]. The development is part of a research project supported by the European Union's 7th Framework Programme (FP7-ICT). The project consortium involves over 30 partner companies and organizations, ranging from device manufacturers, service providers, universities, and research organizations. Various teams distributed across Europe have been working on the requirements, design, and development of webinos since September 2010.

Based on the project's extensive background analysis of the current ubiquitous ecosystem [28], the following prototype platforms were selected for implementation: PC (Linux, Windows, Mac OS X), mobile and tablet (Android), in-vehicle systems (Linux on Pandaboard), and home entertainment systems (Linux on ARM). For rapid

Table 1 Distribution of platform-independent versus platform-dependent code, expressed in lines of code (LOC)

Core platform components	Platform implementation (lines of code, %)				
	Generic code	Android	Linux	Windows	Mac OSX
Client runtime	3146 (92.07 %)	263 (7.70 %)	2 (0.06 %)	4 (0.12 %)	2 (0.06 %)
Service broker	2548 (100 %)	N/A	0 (0.00 %)	0 (0.00 %)	0 (0.00 %)
Communication library	2770 (95.85 %)	99 (3.43 %)	3 (0.10 %)	15 (0.52 %)	3 (0.10 %)

prototyping purposes, the webinos client runtime as well as the service broker component are both implemented on top of Node.js. Node.js is an event-driven JavaScript runtime for Google's V8 engine [26]. The runtime provides a JavaScript virtual machine, enabling webinos to implement most of its core functionality based on device-independent JavaScript code.

This approach enabled the implementation of a working platform prototype for each of the targeted operating systems. Table 1 lists the distribution of platform-dependent and platform-independent code for these implementations, expressed in lines of code (LOC). The code analysis covers three core system components: the client-side web runtime, the cloud-based service broker, and the shared communication and synchronization library (see Fig. 5). For the client web runtime, nearly 92.1 % of the codebase consists of generic JavaScript instructions shared across all platforms. The number of platform-specific modifications needed for Linux, Windows and Mac OSX was remarkably low (0.06 %, 0.12 %, 0.06 % respectively). Only bootstrap code was required for hooking up their underlying filesystems. The Android implementation, on the other hand, required most device-dependent code (7.7 %), due to the need for custom components with regards to WebSocket support, certification handling, and key storage.

Similar results can be observed when analyzing the shared communication library, which handles the communication and synchronization of entities within the webinos Personal Zones. Over 95.8 % of the codebase consists of device-independent JavaScript instructions. The largest number of platform-dependent code can be found in the Android build (3.43 %), which needed custom modifications to its RPC communication stack for handling chunked messages. As with the client-side web runtime, the required modifications for Linux, Windows, and Mac OSX were minimal (0.1 %, 0.52 %, 0.1 % respectively). These modifications are limited to hooking the library into the device's filesystem.

The cloud-based service broker, finally, is completely platform-independent. The Linux, Windows, and Mac OSX builds all share an identical code base. Note, however, that based on webinos' requirements and design Android support has been dismissed for this component. By design, the service broker is a server-side component, which needs to be accessible at all times and across networks. Enabling Android support would not outweigh the costs and constraints in terms of battery consumption, service availability due to sleep mode, network coverage and traversal issues, etc.

In a second stage, webinos' portability and modifiability capabilities were put to an additional test by extending the set of supported platforms and operating systems. Two new device categories were added to the list, i.e., netbook (Chrome OS), and machine-to-machine setups (Arduino, Raspberry Pi). In addition, the mobile device category was extended with Firefox OS. Support for each new platform was implemented by a dedicated webinos developer with prior knowledge of the system's internals. An analysis of the required resources was performed based on the developers' timesheets, Git source code management (SCM) statistics, and Jira issue tracking activities. The extracted results were encouraging. On average, each additional platform category required the assigned engineer to develop for 0.3 to 0.5 PM (Person Months). For more information and detailed instruction on webinos platform ports, the interested reader can refer to [30].

5.2 Application evaluation

Five proof-of-concept application scenarios were selected to be built on top of the webinos platform. These flagship applications aim to demonstrate the potential impact of webinos' built-in platform support for accessible and ubiquitous human–computer interaction (HCI). As for the webinos platform design and implementation, these proof-of-concept applications were developed as part of the webinos FP7-ICT research project. The application prototype development process currently covers a 14-month timespan. All applications are made open source and can be accessed online [30].

5.2.1 Proof-of-concept applications

Travel.[1] The travel application is a multi-device webinos application. The prototype enables a user to manage his or her points-of-interest (POIs) while traveling. Based on

[1] https://github.com/webinos/app-travel-manager.

Fig. 10 Running multi-device prototype of the webinos Travel application

webinos' service brokerage POIs and status information are automatically synchronized between all of a user's devices. The application enables a user to enter travel plans via a desktop or laptop computer. Next, the user can access the travel application from his or her in-vehicle infotainment system. Moreover, a smartphone or tablet can be used for guidance once the vehicle is parked. A running multi-device prototype of this application is depicted in Fig. 10.

Zap and Shake[2]. This media consumption prototype application allows multiple devices of various users to share and render media contents. This application offers both an accessible control interface and an adaptive media rendering interface. The prototype applications also stimulate social interaction by allowing users to share videos and pictures over social media. The application can be executed from any webinos-enabled device. To increase usability, mobile devices can be used to remotely control the media rendering on television screens.

Creative Notes.[3] Creative Notes is a multi-device note editor. With these applications, notes are synchronized between all devices owned by a particular user. Whenever a note is created with one device, all other devices are automatically notified. The webinos platform handles all communication, no intermediate or third server is required. The application also aims to benefit from webinos' cross-device functionality. For example, instructions for capturing images can dynamically be redirected to the user's mobile device in order to take a picture with its built-in camera.

File Manager.[4] The File Manager encompasses a proof-of-concept enhanced document management application. The application adds to the commonly available file management operations (i.e., local file operation including rename, copy, move). Moreover, the application includes selectively sharing of data among personal devices and trusted users. With this application, all personal documents and files are made available and accessible throughout the user's devices. Yet, control over this data remains with the user. The webinos platform allows for seamless service dispatching, regardless of the physical device it is residing on.

Katwarn integration.[5] This application integrates the existing Katwarn service with the webinos platform. Katwarn is a service that informs citizens about nearby emergencies. The webinos Katwarn integration application aims to enhance the interaction with end users as well as the service's accuracy. With webinos' multi-device application platform, the Katwarn service can rely on more detailed knowledge regarding the user's location. Moreover, the webinos service broker allows incoming notifications to be dispatched to the most recently used device.

5.2.2 *Impact evaluation*

The qualitative impact of each prototype application was evaluated during a face-to-face focus group meeting. All applications were presented to the participants in a pitch style presentation, followed by a live 20 min demonstrator and the option for participants to ask questions. The conducted evaluation included three 5-scale Likert questions (1. strongly disagree; 2. disagree; 3. neutral; 4. agree; and 5. strongly agree) [21] regarding the participants' perception of each application's usability, impact, and added value:

- Q1: Does the application support your multi-device accessibility needs?

[2] https://developer.webinos.org/webinostv.

[3] https://developer.webinos.org/creativenotes.

[4] https://github.com/webinos/app-file-manager.

[5] https://developer.webinos.org/inrush.

- Q2: Does the application solve a customer problem by adding value?
- Q3: Does the application clarify webinos' built-in platform support for accessibility?

The second part of the evaluation digs deeper into these key questions. This evaluation part consisted of three open questions regarding the prototype applications' target audience, main functionality, and direct competitors.

- Q4: Describe the key audience for the presented application?
- Q5: How does the application compare to its competition?
- Q6: Which functionality would be key for this application?

During the focus group, the input of 41 participants from across Europe was collected (BE: 2.4%; DE: 19.5%; ES: 2.4%; FR: 7.3%; GR: 7.3%; IT: 17.1%; NL: 4.9%; PO: 2.4%; SE: 4.9%; UK: 31.7%). In order to maximize the coverage of webinos stakeholders, participants were selected based on various parameters (age, technical background, and occupation, etc.). Table 2 and Fig. 11 aggregate the evaluation results for the first three questions. The listed results reflect each application's mean score $\bar{x}$ over the N submitted evaluation scores x_i (with $N = 41$):

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{2}$$

Moreover, the results include the standard deviation σ, depicting the answers' typical variation from $\bar{x}$:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2} \tag{3}$$

The majority of the participants showed a particular interest in the multi-device capabilities of the presented webinos prototypes. The focus group's face-to-face discussions confirmed people's need for accessible human–computer interaction within the ubiquitous computing domain. From this perspective, the webinos prototype applications received good evaluation results with regards to dynamic multi-device support (PC, mobile, TV, car), as well as support for multiple presentation paradigms (screen size, resolution, reading distance) and interaction modalities (touchscreen, mouse and keyboard, stylus). Furthermore, for the Travel, ZapShake, and FileManager applications, the audience clearly understands the benefits of using webinos as a key enabler for the presented ubiquitous accessibility scenarios.

However, evaluation results also uncover the need for a better market positioning of some of the prototype applications (i.e., KatWarn, CreativeNotes). For these particular applications, participants suggest to line out a clearer target audience, as the applications' message would be passed more effectively and thus have a significantly higher impact on the audience's understanding of its accessibility goals.

Table 2 Prototype impact evaluation results

	Mean ($\bar{x}$)	SD (σ)
KatWarn		
Q1	3.22	0.95
Q2	3.43	1.08
Q3	2.67	1.2
Travel		
Q1	3.8	0.71
Q2	3.84	0.62
Q3	3.8	1.0
ZapShake		
Q1	3.77	0.81
Q2	4.24	0.7
Q3	3.62	1.2
CreativeNotes		
Q1	2.89	0.88
Q2	3.0	0.82
Q3	2.95	1.28
FileManager		
Q1	3.68	1.11
Q2	4.16	0.9
Q3	4.47	0.51

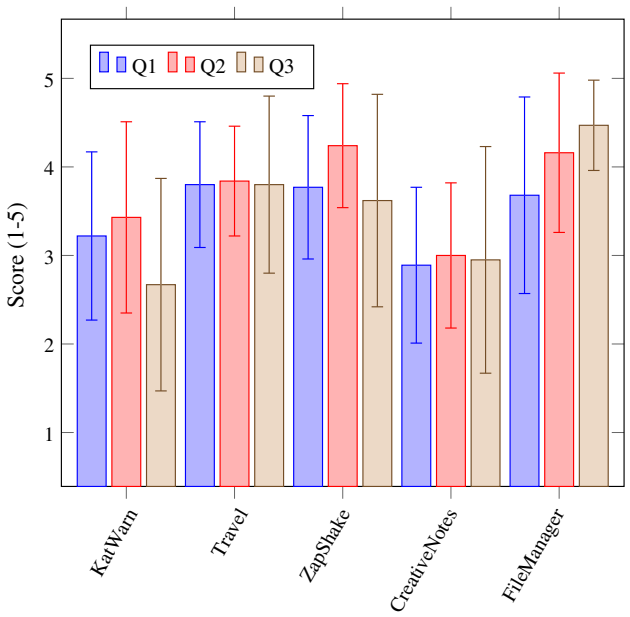

Fig. 11 Focus group impact evaluation results of the webinos prototype applications

6 Conclusion

Designing flexible mobile applications has turned out to be a major challenge to software developers. In this paper, the core architectural modifiability considerations for designing a multi-device ubiquitous platform for accessible Web-based applications have been presented. The proposed architectural structure was applied to the design process of the webinos application platform, aiming for applications available for everyone, at any time, and on any device. With webinos, applications developers are enabled to create software that transcends the executing device's physical boundaries by simultaneously accessing the capabilities of multiple devices. Moreover, webinos aims to enable immersive ubiquitous software applications through adaptive user interfaces. 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 within the webinos platform, 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 [12], as well as the GPII/Cloud4All initiative. Future work for the proposed platform thus includes exploring the possibility to use the webinos platform as a generic enabler for these initiatives to seamlessly connect ubiquitous devices on a global scale. Moreover, future work includes a further evaluation of both the platform's and the applications' implementation results. At first with regards to the webinos platform meeting its modifiability requirements, but also based on the tradeoffs and sensitivity points implied by these architectural decisions. Moreover, the analysis of architectural patterns and tactics should be expanded to a broader range of key quality attributes. These tactics should include architectural considerations on important qualities such as scalability, security, performance, etc.

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 article draws upon their work.

References

1. Amatya, S., Kurti, A.: Cross-platform mobile development: challenges and opportunities. In: Trajkovik, V., Mishev, A. (eds.) ICT Innovations 2013. Advances in Intelligent Systems and Computing, vol. 231, pp. 219–229. Springer, Berlin (2013)
2. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice. Addison Wesley, Reading (2012)
3. Berglund, A., Boag, S., Chamberlin, D., Fernandez, M.F., Kay, M., Robie, J., Simeon, J. (eds.): XML Path Language (XPath) 2.0 (2nd Edn.). W3C Recommendation (2010)
4. BMW: BMW technology guide: controller, http://www.bmw.com/com/en/insights/technology/technology_guide/articles/controller.html
5. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., Vanderdonckt, J.: A unifying reference framework for multi-target user interfaces. Interact. Comput. **15**, 289–308 (2003)
6. Cantera, J.M. (ed.): Model-Based UI XG Final Report. W3C Incubator Group Report, http://www.w3.org/2005/Incubator/model-based-ui/XGR-mbui (2010)
7. Craig, J., Cooper, M. (eds.): Accessible Rich Internet Applications (WAI-ARIA) 1.0. W3C Candidate Recommendation (2011)
8. Chung, L., do Prado Leite, J.: On non-functional requirements in software engineering. In: Conceptual Modeling: Foundations and Applications, pp. 363–379, Springer, Heidelberg (2009)
9. Desruelle, H., Lyle, J., Gielen F.: Leveraging the ubiquitous web as a secure context-aware platform for adaptive applications. In: Proceedings of the 4th International conference on Adaptive and Self-Adaptive Systems and Applications, pp. 57–62 (2012)
10. 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)
11. Dijkstra, E.W.: A Discipline of Programming. Prentice Hall, Englewood Cliffs (1976)
12. FI-WARE: Core platform of the Future Internet, http://www.fi-ware.eu/
13. Fuhrhop C (ed.): Webinos platform architecture and components, Tech. rep. D3.1, Webinos consortium (2012)
14. Isberg, A., Andre, P. (eds.): Webinos discovery API, http://dev.webinos.org/specifications/api/servicediscovery.html (2012)
15. Kay, M. (ed.): XSL Transformations (XSLT) Version 2.0, W3C Recommendation (2007)
16. Koch, N.: Classification of model transformation techniques used in UML-based web engineering. Software **1**(3), 98–111 (2007)
17. 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)
18. 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)
19. Mikkonen, T., Taivalsaari, A.: Apps vs. Open Web: the battle of the decade. In: Proceedings of the 2nd Workshop on Software Engineering for Mobile Application Development (2011)
20. 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)
21. O'Donnell, P.J., Scobie, G., Baxter, I.: The use of focus groups as an evaluation technique in HCI. People Comput. **5**(1), 211–224 (1991)
22. 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)
23. Paul A (ed.): Updates on Scenarios and Use Cases, Tech. rep. D2.4, Webinos consortium (2012)

24. 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)
25. Stevens, W.P., Myers, G.J., Constantine, L.L.: Structured design. IBM Syst. J. **13**(2), 115–139 (1974)
26. Tilkov, S., Vinoski, S.: Node.js: Using JavaScript to build high-performance network programs. Int. Comput. **14**(6), 80–83 (2011)
27. Vanderheiden, G.C., Treviranus, J., Gemou, M., Bekiaris, E., Markus, K., Clark, C., Basman, A.: The evolving global public inclusive infrastructure (GPII). In: Stephanidis, C., Antona, M. (eds.) UAHCI/HCII 2013, Part I. LNCS, vol. 8009, pp. 107–116. Springer, Heidelberg (2013)
28. Voulgaris, G. (ed.). Webinos Target Platforms, Target Requirements and Platform IPRs, Tech. rep. D2.3, Webinos consortium (2011)
29. W3C, Device APIs Working Group, http://www.w3.org/2009/dap/
30. Webinos project consortium: Developer portal, http://developer.webinos.org
31. Webinos project consortium: Device APIs, http://dev.webinos.org/specifications/new/