Architectural modifiability considerations for designing a multi-device web application platform

Heiko Desruelle*, Frank Gielen

Ghent University – iMinds, Department of Information Technology, Ghent, Belgium

Abstract

The mobile web has enabled applications to become available anywhere, anytime and on any device. Numerous emerging web applications have the ability to execute and collaborate across a wide range of web-enabled devices. However, due to the increasing variety of target delivery contexts, the development of such mobile applications has led to a strong need for adaptive software engineering. To address this significant issue, webinos was designed. Webinos is a multi-device distributed platform for ubiquitous web-based applications. In this paper we discuss the architectural modifiability tactics and patterns that were considered for the design of the webinos platform. Moreover, we reflect on the implementation details of realizing such a modifiable architectural design.

Keywords: ubiquitous web, mobile applications, modifiability tactics, software engineering, webinos

1. Introduction

Next generation human-computer interaction paradigms such as mobile and ubiquitous computing have enabled software applications and services to execute on a wide variety of consumer electronic devices. These devices currently range from desktop and laptop computers, to mobile and tablet devices, to television and home entertainment systems, to automotive devices. In this context, the use of web technology provides a standardized abstraction layer for applications to execute device-independently. First of all, the Web already possesses and exposes an inherently ubiquitous nature. The general availability of Internet connections, combined with stable browser technology support on various device types has led to a market coverage that easily surpasses any other application platform [1]. In result, web developers are able to reach significantly more customers with a single application and thus run a more cost-effective business. Moreover, the maturity of web technology has considerably evolved in recent years. The evolution towards HTML5, CSS3, and Just-In-Time (JIT) compilation of JavaScript has greatly increased the basic capabilities of the Web as an application platform. Hence, even enabling web applications to take up the challenge with device-specific implementations of an application. Both in terms of performance metrics, as well as the richness and flexibility of user interfaces (UI).

However, existing web application solutions only partially succeed in enabling a ubiquitous user experience [2]. The lack of runtime modifiability support has resulted in virtual application silos, limiting

*Corresponding author. Tel.: +32 9 33 14941.
Email addresses: heiko.desruelle@intec.ugent.be (Heiko Desruelle), frank.gielen@intec.ugent.be (Frank Gielen)

cross-device execution and collaboration to a specific set of predefined devices and vendors (e.g., Connected TV platforms supporting second screen applications via smartphone devices) [3] [4]. To address this type of challenges webinos was designed. Webinos is an open and distributed platform for web-based applications, aiming to seamlessly interconnect a user's devices through web technology.

In this paper we briefly introduce the webinos platform and elaborate on the architectural modifiability tactics that were applied in order to achieve a solid foundation for such a ubiquitous application platform. The remainder of this paper is structured as follows. Section 2 covers background and related work with respect to modifiability as part of a software architecture. Section 3 provides a high-level introduction to the webinos application platform and its concepts. Moreover, this section elaborates on webinos' modifiability requirements and the incorporated architectural tactics for meeting these constraints. Section 4 covers implementation results for the proposed architectural design. Finally, the conclusion and future work are drawn in Section 5.

2. 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 [5]. Software constantly tends to evolve. From the addition of features, to the support for new technology platforms. In 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 [6]. Coupling measures the mutual association strength between the system's software components. Where 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 towards achieving the required modifiability qualities for their system [7]. 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.

Increasing cohesion tactics aim to deal with the number of internal responsibilities within each of the system's components. This 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 responsibility 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 amongst 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, 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.

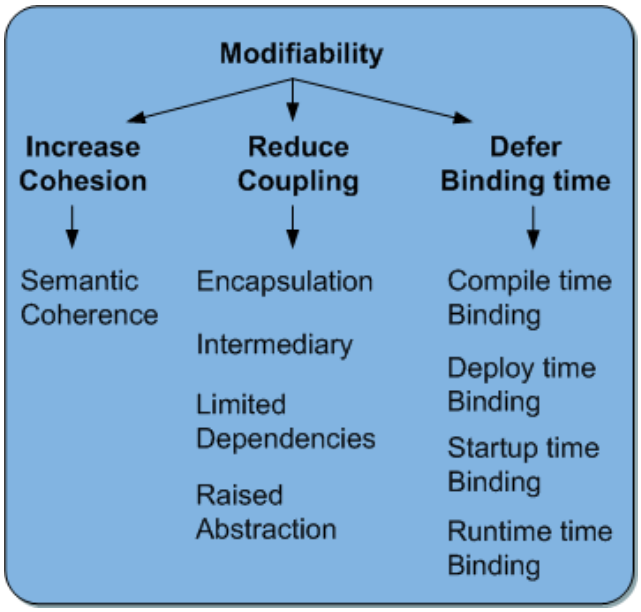

Fig. 1. Organizing architectural tactics for modifiability (derived from [7])

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

3. Modifiability of a multi-device web application 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 [8]. 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. E.g., in an in-car setup this could include accessing your vehicle's sensor data for a parking assistance application running on your smartphone or tablet device.

These modifiability aspects lay out a considerable number of dynamic change requirements for the webinos application platform to adapt to. In this section we present the modifiability tactics that were applied to webinos' architectural design for coping with these requirements. We refer the interested reader to [9] [10] for a more elaborate 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

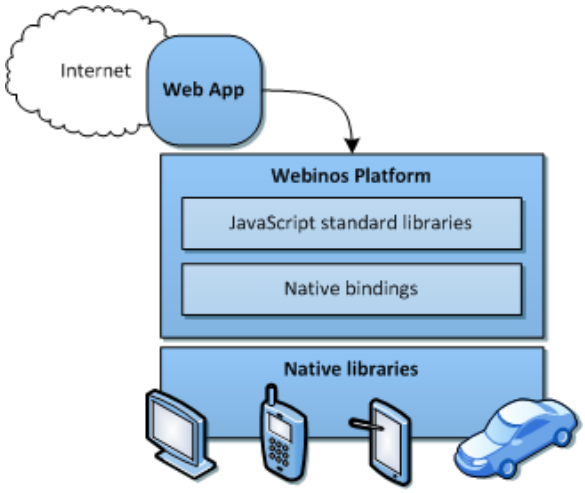

Fig. 2. Virtual machine approach for meeting webinos' portability requirements

instructions are translated at runtime into instructions for the underlying technology platform. The webinos applications' code is thus only interpreted and 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. 2, 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 web APIs (Application Programming Interfaces). Moreover, an intermediary-based tactic allows for the at runtime binding between the webinos applications' instructions in JavaScript and their associated native instructions, interpretable by the devices' underlying operating systems. In result, the device-dependent platform code is clearly separated from webinos' device-independent APIs and standard libraries.

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, 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. 3). 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 on which device the request will be executed.

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

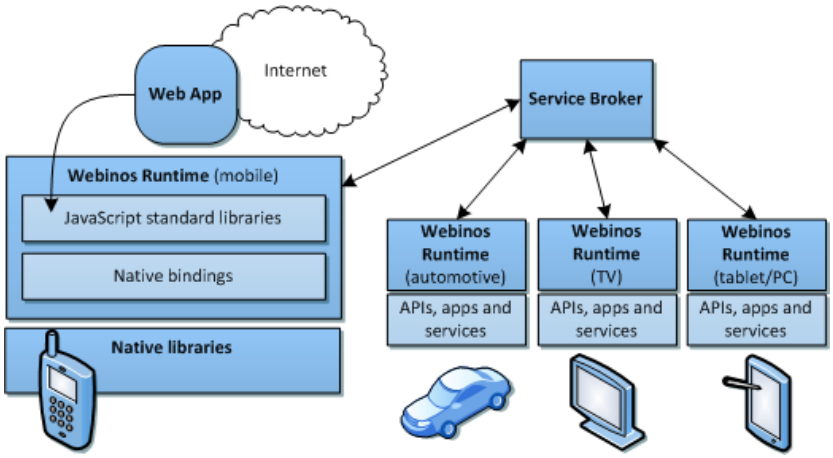

Fig. 3. Service broker approach for webinos' dynamic service binding

encapsulated as a cache within their communication interface. Through communication queuing, all data shared with the service broker are again synchronized as soon as the device's Internet access is restored.

4. Implementation details

A prototype of the webinos platform is currently being implemented. All sources are available as part of the webinos open source project [11]. The development is part of a research project co-funded by the European Union's 7th Framework Programme. The project involves over 30 partner companies, ranging from device manufacturers, service providers, universities, and research organizations. Various teams across Europe have been working on the design and development of webinos since September 2010. Based on the project's extensive background analysis of the current ubiquitous ecosystem [12], the following prototype platforms have been selected for implementation: PC (Linux, Windows, OS X), mobile and tablet (Android, Firefox OS), netbook (Chrome OS), vehicular devices (Pandaboard with Linux), home entertainment (Linux), machine-to-machine (Arduino, Raspberry Pi).

The webinos runtime prototypes as well as the service broker component both extend Node.js, an event-driven runtime for Google's V8 JavaScript engine [13]. Node.js provides a foundation for the webinos platform through its JavaScript virtual machine. Moreover, it implements webinos' portability tactics via its JavaScript-to-native bindings (see Section 3.1). In result, each webinos application can communicate with the platform through generic JavaScript APIs, regardless of the executing device's native programming language. Fig. 4 depicts a prototype application running on multiple device categories, without requiring any modifications to the application's code base.

As discussed in Section 3.2, webinos' service binding mechanism is abstracted for modifiability purposes using a centralized service broker with dedicated discovery API [14]. Within the platform, all services and APIs are identified through a service-type URI (Unified Resource Identifier). E.g., an application developer can pass the URI "http://webinos.org/api/vehicle" to the discovery API in order to get access to vehicular sensor data. The API call triggers 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.

5. Conclusion

Designing flexible mobile applications has turned out to be a major challenge to software developers. In this paper we presented the core architectural modifiability considerations for designing a multi-device

Fig. 4. Running prototype of the webinos multi-device application platform

mobile application platform. The discussed tactics were applied to the design process of the webinos application platform, aiming for maximum platform independence and deferred service binding.

Future work includes a further evaluation of the platform's 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 modifiability decisions. Moreover, the analysis of architectural 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.

Acknowledgements

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 paper draws upon their work.

References

[1] Cozza R, Zimmermann A, Milanesi C, De La Vergne HJ, Sato A, Lu CL, Glenn D, Nguyen TH, Shen S, Gupt A. Market Share: Mobile Communication Devices by Region and Country, Tech. rep., Gartner inc.; 2011.

[2] Taivalsaari A, Mikkonen T. The Web as an Application Platform: The Saga Continues. In: *Proceedings of the 37th EUROMICRO Conference on Software Engineering and Advanced Applications*, New York: IEEE Press; 2011, p. 170-174.

[3] Desruelle H, Lyle J, Isenberg S, Gielen F. On the Challenges of Building a Web-based Ubiquitous Application Platform. In: Proceedings of the 2012 ACM International Conference on Ubiquitous Computing, New York: ACM; 2012, p. 733-736.

[4] Desruelle H, Isenberg S, Lyle J, Gielen F. Multi-device application middleware: leveraging the ubiquity of the Web with webinos. *Journal of Supercomputing*, 2013, DOI: 10.1007/s11227-013-0901-3.

[5] Chung L, do Prado Leite J. On Non-Functional Requirements in Software Engineering. In: *Conceptual modeling: Foundations and applications*, Heidelberg: Springer; 2009, p. 363-379.

[6] Stevens WP, Myers G J, Constantine LL. Structured Design. *IBM Systems Journal*, 1974, 13(2):115-139.

[7] Bass L, Clements P, Kazman R. *Software Architecture in Practice*. 3rd ed. Reading: Addison Wesley, 2012.

[8] 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*, 2012, p. 57-62.

[9] Faily S, Lyle J, Andre P, Atzeni A, Blomme D, Desruelle H, Bangalore K. Requirements sensemaking using concept maps. In: *Proceedings of the 4th International Conference on Human-Centred Software Engineering (HCSE 2012)*, Heidelberg: Springer; 2012, p. 217-232.

[10] Fuhrhop C (ed.). Webinos platform architecture and components, Tech. rep. D3.1, Webinos consortium, 2011.

[11] Webinos developer portal. [Online] http://developer.webinos.org [Acccessed: 1 February 2013].

[12] Voulgaris G (ed.). Webinos target platforms, target requirements and platform IPRs, Tech. rep. D2.3, Webinos consortium, 2011.

[13] Tilkov S, Vinoski S. Node.js: Using JavaScript to build high-performance network programs. *Internet Computing*, 2011, 14(6):80-83.

[14] Isberg A, Andre P (eds.). Webinos discovery API. [Online] http://dev.webinos.org/specifications/api/servicediscovery.html [Accessed: 1 February 2013].