Towards an Architecture for Future Internet Applications

Jacek Chmielewski

Poznan University of Economics, Poznan, Poland
chmielewski@kti.ue.poznan.pl

Abstract. A growing number of connected devices and solutions, related to the concept of Internet of Things, makes our environment increasingly smart and capable. However, existing application development processes and tools, designed for single device applications, do not allow to fully address this opportunity and to efficiently create applications that employ multiple devices and use the context information provided by ubiquitous sensors. To address this situation we propose a concept of Device Independent Architecture, which can be used to separate applications from devices and to provide a uniform platform for building smart multi-device applications. The main ideas of the Device Independent Architecture is to move processing from end-devices to a server side (backend) and to introduce a middleware layer that separates applications from devices and supports development of multi-device applications.

Keywords: Internet Architecture, Multi-Device Applications, Device-Independent Applications, Cross-Platform Applications, UI Adaptation, Internet of Things, Internet of Services, Internet Middleware.

1 Introduction

According to current predictions [1], the number of connected devices will reach 30-50 billion in 2020. This is 4 to 6 times more than the estimated world population at that time. On average, every person in the world will be surrounded by 4 to 6 different devices that will be continuously connected to the Internet. This will be not only smartphones and tablets, but also a smart TV [2], a smart fridge and cooker [3], a home automation system with a number of various sensors and actuators, etc. Technology savvy users use more than just one device even today. Usually the set of used devices include a stationary PC or a laptop (at least one at work and sometimes additional at home), a smartphone, a tablet shared with relatives, a game console, a smart TV, etc. Users are more and more surrounded by smart and connected devices. Today these devices are mostly used independently forcing a user to switch contexts and maintain a number of devices with overlapping functionality. In most cases, it is due to applications designed for a single device (platform), which cannot be reused on a different device (platform). Thus, a user is forced to buy and use a similar application on each of his devices (platforms). To fully exploit the smart environment that grows around us, we need new kind of applications: smart multi-device applications.

A. Galis and A. Gavras (Eds.): FIA 2013, LNCS 7858, pp. 214–219, 2013.

A smart multi-device application is an application that uses context information to intelligently adapt its behavior and that can be used on a subset of devices available for a user at a particular moment. The context information describes the situation in which the application is used, so the application may behave accordingly and appear to the user as a 'smart' entity. Usually, the 'multi-device' term means that an application can be used on different devices: on a smartphone, on a tablet, or even on a TV, which is the case of many web applications. However, such application is better described by a term 'cross-platform' application. In our approach, a 'multi-device' application is an application which is capable of employing multiple devices at the same time – for example, a smartphone used as a remote controller for a TV set and a PVR, and as an EPG browser.

This trend poses new challenges to the traditional application development process. Native application development forces developers to target each device specifically, making the development process difficult (different skillset required for each new platform; variable device capabilities) and time consuming (multiple versions of an application targeting different device classes). Therefore, applications shift from end device (native) to the backend (servers, cloud, etc.) using the client-server architecture and sharing resources to support multi-device usage scenarios.

However, it is not the end of application architecture evolution – the future brings new challenges. With new developments in the area of Internet of Things a massive number of new smart and connected devices will appear in the digital ecosystem surrounding a user: a car, a home automation system, a cooker and fridge, a weather station, public displays, various sensors and actuators, etc. It will make application development even more complicated and will open a way for truly multi-device use cases – where a user not only switches from one device to another, but uses multiple devices at the same time for one purpose (the same application). Existing application architectures are not capable of fully supporting the new breed of applications – smart multi-device applications.

2 Device Independent Architecture

To be able to provide applications capable of addressing the large, diverse and fast growing set of end-devices, it is required to build these applications in a way that will make them independent of a particular device and will enable multi-device usage scenarios. This is the goal of the Device Independent Architecture (DIA).

2.1 Device Independence

The idea behind our Device Independent Architecture originates from the Service-Oriented Architecture, where systems are decomposed into atomic services and processes use necessary services without knowing much about systems that provide these services. Similar approach can be used to decompose devices. Each device, be it a laptop, a smartphone, or a smart temperature sensor, provides: resources, services and user interaction channels.

Resources are: processing power (CPU), memory, and storage. Services are providers of context information, such as location, temperature, light intensity and data from other sensors. Input and output user interaction channels are means of communication with a user and include: screen, speakers, vibration, keyboard, mouse or other pointing method, touchscreen, microphone, camera, etc.

One of the main problems of device-independent application development is the diversity of ways in which these resources, services and interaction channels can be utilized by an application. Each operating system or a software platform has its own programing languages and APIs. Even devices of the same class (e.g. Android smartphones and tablets) provide different sets and characteristics of available services and user interaction channels.

Our approach to device independence is to separate the application from a device by using only backend resources (OS and software platform independence), building abstract user interfaces (user interaction channel independence), and providing a middleware layer that wraps all services with a standardized API and adapts application user interfaces to capabilities of available user interaction channels.

Resources can be provided as cloud-based services in the form of a Platform as a Service (PaaS). To maintain device-independence of applications it is not necessary to specify a particular PaaS infrastructure. Each PaaS implementation has its own requirements and API but the selection of a particular PaaS will influence only the choice of a programming language and development patterns. It will not hinder the device-independence of an application developed on the selected platform.

Services cannot exist without a device, but can be exposed using REST GET method (for one time access) or a protocol based on a subscriber/provider pattern (for continuous access). Such a generic protocol ensures proper application—device separation, while maintaining the ability to use data provided by device sensors. Types of data provided by information sources may be described using syntactic and semantic service description languages such as WSDL or USDL [4].

User interaction channels can be divided into output interaction channels provided to applications as data-sinks accepting specific data formats (e.g. visual data for screen, audio data for speakers, etc.) and input interaction channels exposed as services accessible according to a protocol based on the Observer pattern. Again, types of interaction events provided by these services can be described using service description languages.

Such a generic approach to application—device communication enables an application to employ any set of devices that provide services and user interaction channels required by the application. The DIA itself does not impose any requirements on capabilities of devices, suitability of a device depends solely on application requirements. With proper semantic service descriptions a device could expose any services and user interaction channels, including those that will be introduced in the future.

2.2 Multi-device Support

The fact that applications operate on sets of services and user interaction channels, instead of monolithic devices, enables implementation of multi-device scenarios:

- complementary services scenario – application using various services provided by different devices;
- redundant services scenario – application using the same services provided by different devices to get the best possible results (e.g. usually a geolocation service on a smartphone is better than the one on a laptop);
- multi-device UI scenario – application distributing fragments of its UI to separate devices choosing the best composition of available interaction channels.

In our approach, the middleware maintains a register of available services and user interaction channels provided by devices. All actions related to device discovery, authorization and inter-device communication are hidden from an application. What the application sees is a set of services and user interaction channels, not separate devices. When a device becomes available or unavailable the middleware modifies the register accordingly and notifies applications. Therefore, the application may adapt to the new situation.

Also, the middleware layer acts as a user interface proxy, which enables the multi-device application to maintain a single abstract UI description and target multiple end-devices. With additional UI cues embedded in an abstract UI description, it is possible to decide on UI fragmentation and distribute different parts of the application user interface to different devices. Each of such UI fragments is then adapted by the middleware to match capabilities of a target end-device and a final UI is generated and sent to the device. In the reverse direction, user interactions performed at all used end-devices are mapped back by the middleware to appropriate controller actions of the multi-device application. This way, the application developer does not have to care about knowing how many and what devices could be used by a user.

The middleware may also support additional functions such as security or billing. With the Device Independent Architecture, the middleware may act as a device authorization center and device usage policy enforcement point. The security and usage information may influence the set of services and user interaction channels provided to different applications. The DIA architecture may also support or limit the usage of public, shared, and private devices. For example, a usage of public devices may require additional authorization or micropayment and a usage of a shared device may require confirmation by the owner or the current user of the device.

2.3 The Architecture

With DIA, any application may use any subset of the available devices in a way that is transparent for the application development process. The communication between applications and the middleware, and between devices and the middleware, is based on standard protocols, making it easy to support new devices and new applications.

Devices available to a user, that form a personal zone [5] of the user, register with the DIA middleware providing information about the offered services, the available user interaction channels, and – optionally – other information such as usage policy, billing settings, etc. When an application is started, the middleware provides it with information about available services and user interaction channels. The middleware

also notifies the application about any changes in the set of services and user interaction channels. Consequently, the application is always aware of the situation and may act accordingly. The application may request a selected service with a standardized API call and respond with an abstract description of a user interface that should be presented to the user. The middleware will forward the service call to a device that provides the service and will use the abstract UI description and embedded UI cues to properly fragment, adapt and distribute the UI to available end-devices. User interactions registered on devices presenting the UI are caught by the middleware, mapped back to appropriate application controller actions, and dispatched to the application closing the user interaction loop.

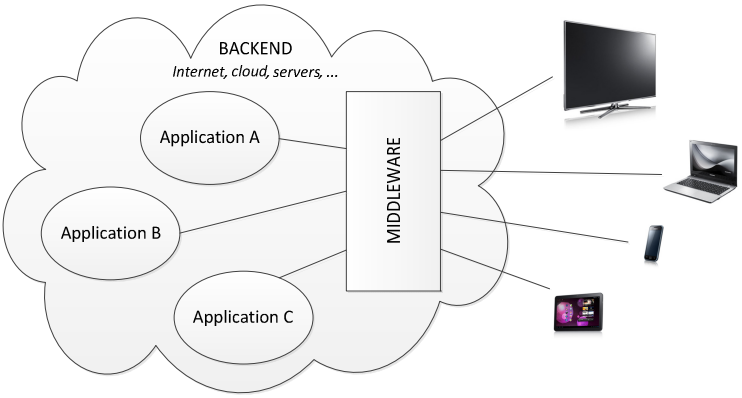

Fig. 1. Device Independent Architecture concept

An early implementation of ideas introduced by the Device Independent Architecture is the ASIS framework [6], which was developed for the IT-SOA project [7] and was also used in e-learning applications [8-9]. Another implementation of DIA concepts related to UI adaptation is being prepared for the UbiqUI project [10].

3 Future Directions

Despite ongoing efforts on defining device-independent or platform-independent cloud services there is no complete framework that could support implementation of smart multi-device applications. The main aim of our current research on the DIA is implementation of a prototype of such a framework. It will be used as a proof-of-concept and will allow analyzing various application scenarios and multi-device issues – including, but not limited, to the following:

- Device discovery and authentication
- Device usage policy
- Services and user interaction channels registration
- Context information modeling and access protocols
- UI abstraction, adaptation and distribution (not only for graphical UI)

- Application catalogs, markets and private application repositories
- User preferences gathering and provisioning methods
- Device usage billing and micropayments

Implementation of such a framework will help to move the focus of IoT research from communication and interoperability issues to applications issues, bringing it closer to final users of IoT solutions. Each of the topics mentioned above is a challenging task on its own. To succeed in this broad field, a coordinated research effort is required. We believe that the concept presented in this chapter provides a solid basis for future research in the field of smart multi-device applications and will eventually ease the burden of developing truly smart applications that efficiently use capabilities of multiple devices.

References

1. Zappa, M.: Envisioning emerging technology for 2012 and beyond, Envisioning Technology (2012), http://envisioningtech.com/envisioning2012/
2. SAMSUNG Smart LED and Plasma TVs Usher in a New Era of Connectivity and Control, Samsung US News Center, http://goo.gl/3pmlb
3. LG smart appliances for 2012 deliver connectivity, efficiency through Smart ThinQ™ technologies, LG Newsroom (2012), http://goo.gl/lsz68
4. Simon, L., Mallya, A., Bansal, A., Gupta, G., Hite, T.D.: A Universal Service Description Language. In: Proceedings of the IEEE International Conference on Web Services, ICWS 2005, pp. 823–824. IEEE Computer Society (2005)
5. Lyle, J., Monteleone, S., Faily, S., Patti, D., Ricciato, F.: Cross-platform access control for mobile web applications. In: Proceedings of the IEEE International Symposium on Policies for Distributed Systems & Networks, pp. 37–44. IEEE Computer Society (2012)
6. Chmielewski, J., Walczak, K., Wiza, W.: Mobile interfaces for building control surveyors. In: Cellary, W., Estevez, E. (eds.) 13E 2010. IFIP AICT, vol. 341, pp. 29–39. Springer, Heidelberg (2010)
7. Chmielewski, J., Walczak, K., Wiza, W., Wójtowicz, A.: Adaptable User Interfaces for SOA Applications in the Construction Sector. In: Ambroszkiewicz, S., Brzeziński, J., Cellary, W., Grzech, A., Zieliński, K. (eds.) SOA Infrastructure Tools - Concepts and Methods, pp. 439–469. Poznań University of Economics Press, Poznań (2010)
8. Walczak, K., Wiza, W., Rumiński, D., Chmielewski, J., Wójtowicz, A.: Adaptable User Interfaces for Web 2.0 Educational Resources. In: Kiełtyka, L. (ed.) IT Tools in Management and Education - Selected Problems, pp. 104–124. Publishing Office of Czestochowa University of Technology, Częstochowa (2011)
9. Walczak, K., Chmielewski, J., Wiza, W., Rumiński, D., Skibiński, G.: Adaptable Mobile User Interfaces for e-Learning Repositories. In: Sánchez, I.A., Isaías, P. (eds.) IADIS International Conference on Mobile Learning, pp. 52–61. IADIS Press (2011)
10. UbiqUI - Ubiquitous mobile user interfaces for industrial monitoring and control applications (2012-2013), http://www.kti.ue.poznan.pl/en/node/3230