Dynamic User Interface Architecture for Mobile Applications Based on Mobile Agents

Nikola Mitrović[1], Carlos Bobed[1,2], and Eduardo Mena[1,2]

[1] Dept. of Computer Science & Systems Engineering
University of Zaragoza, Spain
[2] Aragon Institute of Engineering Research (I3A), Spain
mitrovic@prometeo.cps.unizar.es, {cbobed,emena}@unizar.es

Abstract. Developing Graphical User Interfaces (GUIs) for mobile applications is a difficult task. Modern applications frequently need to interact with humans that use several devices with different characteristics (such as screen size or operating system). Any application that creates a specific GUI (surely designed for a certain device family) is likely to be rendered and/or behave incorrectly on many other user devices. Moreover, user interactions also need to be adapted to the preferences of each specific user and be learned from user context. The above challenges delegate every single application to have multiple versions of its GUI to be correctly executed on every possible device and operating system combination, in addition to consider user preferences and context.
In this paper, we propose an architecture based on mobile agents for developing adaptive user interfaces for multiple devices and applications. This architecture makes it possible to further separate GUIs from their underlying logic, allowing GUIs to be specified once and automatically be adapted to different platforms and user preferences without further development. Moreover, our architecture enables GUIs to be composed in a collaborative way by multiple agents and across different devices by automatically adapting them to each device capabilities and user preferences. Thus every application developer is relieved of considering these issues.

Keywords: Dynamic GUIs, Mobile Cooperative Agents, Mobile Computing

1 Introduction

Adoption of mobile devices over the last decade has been significant: Just the number of mobile phones has exceeded world population in 2014[3]. Nowadays people rely on devices such as smartphones, tablets, laptops, smart watches, smart TVs, and PCs not only for practical and productivity tasks, but also to enhance social aspects of their lives. As the user adoption of these devices is increasing, the number of new platforms and devices is on the increase too.

With the rising use of mobile devices, users' expectations of new applications and GUIs have risen too: Users frequently expect applications to be available on any device

[3] Definitive data and analysis for the mobile industry, https://www.gsmaintelligence.com/, last accessed 12th September 2016.

they own, regardless of the operating system or device features. Usability and flexibility of GUIs are more important than ever as interaction with the computer can happen anywhere and users dislike spending unnecessary time performing a task. Moreover, we are increasingly surrounded by sensors, sensing devices, computers which brings new dynamic and complexities. Users expect that all these devices and sensors will collaborate to learn about the user, predict users' needs, understand the context and help them complete tasks in the most efficient way. In this context, GUIs are required not only to be adapted to the specific device and platform combinations, but also to take into account users contexts.

Therefore, creating GUIs for collaborative and mobile applications that can work on multiple platforms and provide users, with experiences adapted to their context and preferences, is not an easy task. Different platforms or interaction modes (e.g., Android or Windows OS, smart watch, or smartphone applications) require different and separate GUI code. On the other hand, GUI adaptation to users preferences and context is typically custom developed for each GUI or application, which reduces the re-usability of the solutions. Every single application must have multiple versions of its GUI to be correctly executed on every possible device and operating system combination, in addition to consider user preferences and context, if its goal is to execute correctly in a wide spectrum of user devices with different capabilities and features. Even more, applications developed today are likely expected to be prepared, not adapted, to new incoming devices and operating systems. To solve all these problems we need 1) a way in which applications specify GUIs without compromising its flexibility to allow them to adapt to different devices and preferences, 2) an intermediate module that plays the role of interface between applications and user devices, alleviating the former from efforts to adapt their GUIs to user devices, preferences, and context.

Some approaches provide a certain level of GUI portability across one or two specific platforms (e.g., Android [1] or Xamarin [2]). However, such approaches do not automatically adapt the GUI to the device, require development of platform-specific GUI code, and do not offer automatic ability to adapt to the user preferences or context. Many approaches advocate using a *User Interface Definition Language (UIDL)* such as XUL [3], UIML [4], or UsiXML [5]. Such approaches allow re-using a single GUI design on multiple platforms but the GUI code is typically created and pre-compiled separately for different platforms [5]. As the code is specifically crafted for individual devices or platforms, it is hard to change and adapt to user or context once the application has been deployed to a target device. Some approaches use a client-server architecture [6] to adapt UIDL at run-time; these approaches can adapt GUI more easily across multiple devices. However, these approaches do not offer adapting to user preferences and context, and require the existence of client and server software (a fixed point in the network). Finally, it is important to note that GUIs in the above approaches are usually pre-defined and may take input from other devices or software; they are however seldom generated in a collaborative way, with multiple independent entities contributing to the GUI.

In this paper, we propose ADUS[4], an architecture based on mobile agents [7] for developing adaptive user interfaces for multiple devices and applications. This is achieved

[4] ADUS stands for **AD**aptive **U**ser interface for mobile device**S**.

by adopting the use of UIDL specifications [4], which are processed by a network of agents who collaborate to both adapt the GUI to the local platform, as well as to learn from user interactions. By using the proposed architecture, application developers only have to specify their GUIs according to their functionality, which will be automatically adapted to each device and user at run-time.
The contributions of this paper can be summarized as follows:

1. We review the state of the art of GUI abstract definitions with specific focus on applicability to current mainstream mobile environments.
2. We introduce the ADUS architecture for GUI adaptation and its application in mainstream mobile computing environment.

The remainder of this paper is structured as follows. Section 2 introduces different existing GUI adaptation approaches, including different architectures and proposals based on mobile agents. In Section 3 we summarize the ADUS architecture and show how ADUS uses mainstream UIDLs. Finally, Section 4 concludes the paper giving a summary and an outlook.

2 Related Work

The development of GUIs has been and continues being a subject of intensive research, as they provide users with the means to interact with the computers (and applications). Up to this point, the different proposals to develop and generate GUIs can be broadly classified as follows:

1. *Design-time GUIs*: A tool is used to design the application GUI, the tool generates some skeleton to write the interaction code, and finally, the code is then compiled for a specific platform. This approach is presented in Section 2.1.
2. *Client-server GUIs*: In this approach, the client component renders the GUI to the specific device and the server component provides the business logic, GUI contents, and any adaptation. This approach is discussed in Section 2.2.
3. *Dynamic GUIs*: In this approach, GUIs can be defined at run-time and depend less on specific operating systems. Dynamic approaches are described in Section 2.3.
4. *Mobile agent-based GUIs*: Last but not least, the approaches in this category use mobile agent technology to provide mobility and enable collaboration and adaptation of GUIs to devices and user's context in run-time.

In the remainder of this section we will describe details of each of these broad categories of GUIs.

2.1 Design-time

GUIs defined statically during the application design phase are one of the most common and popular approaches to GUI development. Using this approach, an *abstract* GUI is first defined using the abstract User Interface Description Language (UIDL) [4], and it is then compiled to a *concrete* user interface using a compiler or similar tool. The GUI is specifically designed for the target operating system or device.

The Design-time approaches are able to tailor and fine-tune GUIs to the specific of the target devices or operating systems. However, they require multiple versions of user interaction code to be developed (e.g., Xamarin [2]) or are able to serve only proprietary platforms (e.g., Apple Storyboards [8]). The GUI interaction code needs to be developed multiple times, and if the GUI was to be changed, the application needs to be re-developed and re-deployed to all devices. Software developers need to have significant expertise, know how to code for different devices, and how to work with their quirks. The resulting GUI has limited adaptability to contexts, events, or environments that have not been explicitly considered and developed by the software developer during the application design phase. For example, the software development would need to pre-program GUI code so to adapt to meet user's preferences whilst in a car or whilst at office. Besides, all GUI interaction is typically defined within the single application: GUIs are pre-defined and are not typically modified by other applications (i.e. GUIs are not generated in a collaborative way).

2.2 Client-Server GUIs

In this category of approaches, the client component usually renders the GUI to the specific device and the server component provides business logic, GUI contents, and any adaptation. This approach is very popular on the Internet; in fact, all web pages are delivered this way. In the case of web pages, the GUI is defined by the software developer using HTML [9] as the UIDL.

The client-server approach is very complex. HTML is not sufficient to describe and handle GUIs on its own and is almost always modified using Cascading Style Sheets (CSS) [9] and multiple JavaScript libraries [9] to achieve desired GUI. Client software (Web Browser) need to be developed for each device and existing Web Browser implementations interpret HTML, CSS, and JavaScript differently. Getting run-time insight for troubleshooting in client-server situations is very difficult too [9]. All of this makes client-server GUI difficult to develop and troubleshoot. Finally, the client-server approach relies on the availability of both the server and client component. If the server is 'off line' the GUI cannot be presented. For example, a Smart Watch does not have a Web Browser (the client) and so the HTML GUI cannot be displayed. More importantly, for complex interactions and applications, the GUI requires native code to be developed which, in turn, requires again multiple versions of code to be developed (specific to target operating systems or devices).

2.3 Dynamic GUIs

Some approaches allow GUIs to be defined and adapted as they are executed on the device. Examples of such approach include QML/QT [10]. Using this approach a developer can define GUI using QML notation and the code is bound to the GUI specification at the run time. However, as in the case of QML/QT, availability of the middleware script runtime for QT will be required for different devices or operating systems.

GUI specification languages in this category sometimes do not offer sufficient GUI abstraction level; for example, QML requires stronger widget positioning than other

UIDLs. Although QML and QT are available for a number of different operating systems, it is restricted by some mainstream mobile devices (e.g., Apple) in such a way that all code must be statically packaged together, effectively making these approaches equivalent to design-time approach as described in Section 2.1. Dynamic GUI generation approach suffers from similar problems as the design-time approach when it comes to user context, device GUI adaptation, and goal orientation (see Section 2.4).

2.4 Mobile Agents and Agent-Based GUI Approaches

Mobile agents are autonomous software entities [7] that are capable of migrating from one device to another autonomously and continue their execution on the destination device. Mobile agents are goal oriented and social, i.e., in order to achieve their goal, they collaborate with other mobile agents. They have been used in the context of distributed computing [11] and are gaining popularity in fields such the Internet of Things (IoT) applications [12].

Because of their properties, mobile agents can provide good solutions for adaptive GUIs that are created in a collaborative way. Mobile agents can arrive at the users' device, check the device capabilities (i.e., its display size, the available interaction modes and GUI elements, etc.), and show their GUIs adapting it to the application goals. Thus, they can exploit different models of user interfaces on different and heterogeneous platforms. Due to their autonomy, mobile agents can handle communication errors (unreachable hosts, etc.) by themselves. Also, in contrast to the client-server model, they can move to the target device instead of accessing target devices remotely. For example, agents can be sent to a Windows desktop computer or they can play the role of a proxy server for a wireless device such as a Smart Watch that has limited processing capabilities. A number of researchers have adopted mobile agents technology for GUIs in order to address specific problems [13] [14] [15].

In our previous and preliminary work [16], we presented a previous version of our multi-agent system for generating GUIs. The system was using XUL as UIDL and mobile agents to deliver GUIs to different devices. However, the mainstream devices, platforms, and ecosystems have developed significantly over the last few years, and none of the approaches described in this section would be able to work on modern devices. The approach presented in this paper devises a new architecture and implementation that takes into consideration architectural challenges of the current mainstream devices and platforms.

3 ADUS Architecture Overview

ADaptive User interface for mobile deviceS (ADUS) is a system and architecture for generating adaptive GUIs. The ADUS is based on mobile agent technology (see Section 2.4) and the use of User Interface Description Languages (UIDLs) in order to achieve its flexibility. Its main goal is to adapt GUIs to devices with very different characteristics (e.g., fixed vs mobile devices, display sizes, input modes, ...).

In this section, firstly, due to their importance in our approach, we present a briefly mainstream UIDLs which we have analysed, and our choice for ADUS. Then, we present our mobile agent architecture which enables the development of adaptive GUIs.

3.1 User Interface Definition Languages

One of the basic aspects of GUI adaptation is the adoption of a User Interface Definition Language (UIDL) [4, 5, 17] to specify the desired GUI. Using this approach, an *abstract* GUI is first defined using the abstract UIDL, and it is then compiled to a *concrete* user interface. The abstract definitions can be pre-compiled as described in Section 2.1, or processed in a dynamic way at run-time (see Section 2.3).

We have compared several commercially available mainstream UIDLs [18] to establish their fitness for mobile application development. For space reasons, we focus here on three UIDLs based on their high level of adoption for mobile application development, namely, Android XML, Windows XAML, and Apple's Storyboards:

- **AndroidXML** Android is by far the most widely used operating systems when it comes to mobile devices (i.e., smartphones and tablets) and the respective UIDL is Android XML [19]. Given the heterogeneity of Android devices, support for GUI adaptation to different devices is provided using layout elements and visual behaviour policies. It is, however, the developer who is in charge to develop and modify GUI so it is correctly adapted to different Android devices.
- **Storyboards (iOS)** Similarly to Android, Apple's iOS also adopts a XML-driven interface via *Storyboards* [8]. Apple's approach goes further than Android: Storyboards use *views* (similar to Android's *activities*), but they can also explicitly include navigation aspects of user interaction in the GUI specification. Thus, a storyboard provides a comprehensive model of the whole application and the workflow of interactions. Storyboards XML is clearly designed to be machine-generated and is not developer-friendly.
- **XAML** (eXtensible Application Markup Language) [17] is the UIDL developed by Microsoft. XAML is at the core of Microsoft's effort to unify application development on different Microsoft devices and platforms (Unified Windows Platform, UWP) [17]. Such unified applications share a basic API and GUI elements which are then extended and specialized for specific device families. Although XAML can be used on any Windows-based device, there is still a need to adapt GUIs to specific devices characteristics[5]; also adaptation to non-Windows devices is still required.

For developing the ADUS system, we decided to adopt Microsoft XAML as UIDL as it is most suitable for our use. Comparing with other mainstream and commercially available UIDLs, XAML offers very high abstraction level and is one of the less vendor-specific languages. XAML also offers good visual tools and most importantly native support across Microsoft's extensive ecosystem. XAML can be used as a native tool to develop for both Windows mobile phones, Windows desktop and laptop devices, and other Microsoft devices such as tablets or surface computers. It is important to note that our architecture (as we will see in Section 3.2) can use any abstract UIDL to generate adaptive GUIs.

[5] Note that XAML is thus not used only for mobile devices, but for a broad family of heterogeneous devices (e.g., desktop computers, surfaces, consoles, ...).

3.2 ADUS Mobile Agent Architecture

In our system, mobile agents collaborate [7] in order to achieve their goal of adapting GUIs. In particular, as we can see in Figure 1, the ADUS system contains several agents:

1. *Visitor Agent*: This mobile agent contains the core of the application functionality and the business logic used in the application. Apart from the application logic, this agent carries along with him the descriptions of all the different GUI elements (using a UIDL) that it needs to interact with the user.
2. *User Agent*: This static agent is in charge of storing and managing information about the user, her preferences, and the device at which this agent resides. For example, it provides profile and context information such as device type, location, or user's notification and font size preferences.
3. *ADUS Agent*: This static agent is in responsible for orchestrating all the required UIDLs adaptations and transformations, as well as for handling the user-computer interaction by creating the appropriate GUIs.
4. *(Optional) Knowledge agents*: These agents are specialised agents that can learn from collected information (e.g., GUI interaction events) and create knowledge that can be then either 1) passed to ADUS Agent to further adapt/transform the GUI or 2) passed to another agent for their use.

Using these agents and a UIDL to define the GUI, ADUS adopts an indirect generation architecture for generating and managing GUIs[6]. The detailed interaction among agents is as follows (see Figure 1):

1. The Visitor Agent arrives (or is created) at the target device, carrying both the application code and a basic GUI specification using UIDL.
2. The resident User Agent detects the Visitor Agent arrival, and creates a new ADUS Agent, which is specifically assigned to the recently arrived Visitor Agent.
3. The Visitor Agent sends the UIDL specification to the newly created ADUS Agent.
4. The ADUS Agent analyses the UIDL specification and passes it to the User Agent, which adapts such a specification to taking into consideration the user preferences and device features (i.e., it applies basic customisations such as font size and background color).
5. The ADUS Agent receives the adapted specification along with other user and device related information from the User Agent which might be relevant for the GUI (e.g., global properties such as when not to disturb the user with notifications).
6. The ADUS Agent generates a GUI for the specific device, reconciling it with the local device interface model and GUI available elements. Moreover, it handles all interaction events forwarding them appropriately to the Visitor Agent and calling its handling callbacks.
7. Each time the Visitor Agent processes a forwarded event, it responds accordingly presenting an updated GUI specification for ADUS' consumption (back to point 3).

[6] We refer the interested reader to our previous work [20], where a discussion about the benefits and drawbacks of other possible alternative agent architectures are presented.

8. As ADUS Agent processes interaction data, these data are collected and can be analysed. Such an information can be made available to any agent, e.g., User or Knowledge Agents. These agents can then analyse past interactions, learn from them, and create knowledge that can be applied to improve GUI or application usability. For example, input data could be saved and pre-populated on the next launch of the application. ADUS can use any tool set or learning techniques to enhance the GUI.

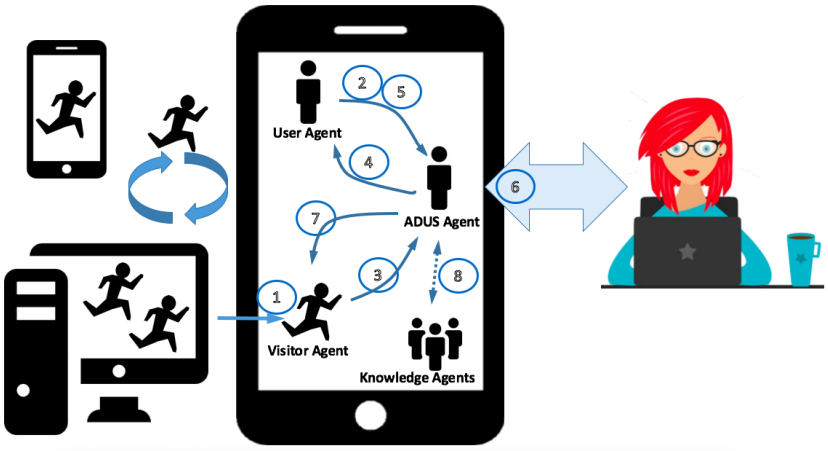

Fig. 1. ADUS: Indirect GUI generation architecture

By using the above method, the different elements of the GUI are created via an ADUS Agent. The GUI can be adapted to target device and to user preferences. In this way, the Visitor Agent does not have to know how to generate GUI or handle interactions with the specific device or user. Besides, note how the ADUS Agent is instantiated by the target platform, so the trust between the User and ADUS Agents is inherent. The trust relationship is important as agents that are not inherently trusted (e.g., the Visitor Agent) could generate GUI that is not appropriate or respecting users' preferences.

The user interactions are completely delegated to the ADUS Agent. The reason is twofold: 1) if the GUI was to be created by the User or Visitor agent, they would need to know how to interpret any GUI specification and create it on any device (this programming effort is put on the development of ADUS versions); and 2) if the User Agent was creating GUIs locally, it would quickly be overloaded by the possible multitude of applications and different tasks (ranging from learning about user and device to creating GUIs).

Finally, it is important to note that our approach allows collection of interaction data independently of the underlying platform. Collected data can be analysed at run time by learning algorithms and specialist agents so that GUI can be improved at the

time of execution. The cooperation of agents when creating GUI in our approach is significant: the specification of a GUI is built collaboratively by agents with specialist roles allowing GUI to become truly dynamic and adapted to context and task at hand. For example, a Knowledge Agent can modify GUI specification so to make GUI more usable, and User Agent can help GUI be more adapted to users' preferences.

We have chosen this approach as it enables the creation of a functional, flexible, and trusted user interface. In addition, as multiple Agents collaborate to create a GUI, human computer interaction can be easily monitored and system load can be easily distributed.

4 Conclusions and Future Work

Adapting GUIs to devices and users automatically is a difficult task. In this paper we proposed ADUS, an architecture based on mobile agents for developing adaptive user interfaces for multiple devices and applications. Previous GUI adaptation approaches reviewed in this paper would not be able to operate using mainstream platforms on current mobile devices. Our focus here has been not only to create an architecture and approach that adapts GUIs but also that the approach can be implemented using mainstream technologies and on current mobile and fixed devices. The main contributions of this paper are:

1. Ability to adapt GUIs on today's devices and platforms. Majority of related work reviewed in this paper would not be able to work with the current mainstream platforms without significant modification. We presented an up-to-date architecture (based on mobile agents) for adapting GUIs that is usable for modern mainstream platforms and devices. Using our architecture GUIs are specified once and automatically adapted to devices and user preferences at run-time.
2. Collaborative GUI. ADUS architecture allows GUIs to be created collaboratively by multiple agents (as opposed by a single entity or agent). Moreover, in ADUS architecture, agents collaborate in order to enhance GUIs at run-time. Our architecture allows monitoring of user behaviour and application of learning techniques to improve the GUI.

As future research we intend to improve content transformation and capturing of GUI events. Finally, we envisage including additional learning and re-targeting algorithms as well as more complex widgets in the prototype.

Acknowledgments

This work was supported by the CICYT project TIN2013-46238-C4-4-R and DGA-FSE.

References

1. Rogers, R., Lombardo, J., Blake, M.: Android Application Development. O'Reilly (2009)

2. Xamarin: http://www.xamarin.com, last accessed 12th Sep 2016.
3. Bullard, V., Smith, K.T., Daconta, M.C.: Essential XUL Programming. Wiley (2001)
4. Abrams, M., Phanouriou, C., Batongbacal, A.L., Williams, S.M., Shuster, J.E.: UIML: an appliance-independent XML user interface language. Computer Networks **31**(11) (1999) 1695–1708
5. Michotte, B., Vanderdonckt, J.: GrafiXML, a multi-target user interface builder based on UsiXML. In: Proc. of 4th International Conference on Autonomic and Autonomous Systems (ICAS'08), IEEE Computer Society (March 2008) 15–22
6. Coninx, K., Luyten, K., Vandervelpen, C., Van den Bergh, J., Creemers, B.: Dygimes: Dynamically generating interfaces for mobile computing devices and embedded systems. In: Proc. of 5th International Symposium, Mobile HCI 2003, Udine, Italy, September 2003, Springer (September 2003) 256 – 270
7. Gray, R.S., Kotz, D., Nog, S., Rus, D., Cybenko, G.: Mobile agents for mobile computing. Technical Report TR96-285, Dartmouth College (1996)
8. Neuburg, M.: Progamming iOS 9. O'Reilly (2015)
9. Weyl, E.: Mobile HTML5. O'Reilly (2013)
10. Rischpater, R.: Application Development with Qt Creator (2nd Edition). Packt Publishing (2014)
11. Bobed, C., Ilarri, S., Mena, E.: Distributed mobile computing: Development of distributed applications using mobile agents. In: Proc. of the 16th International Conference on Parallel and Distributed Computing (PDPTA'10), CSREA Press (July 2010) 562–568
12. Leppänen, T., Liu, M., Harjula, E., Ramalingam, A., Ylioja, J., Närhi, P., Riekki, J., Ojala, T.: Mobile agents for integration of internet of things and wireless sensor networks. In: Proc. of 2013 IEEE International Conference on Systems, Man, and Cybernetics (SMC'13), IEEE Computer Society (October 2013) 14–21
13. Liu, H., Lieberman, H., Selker, T.: A model of textual affect sensing using real-world knowledge. In: Proc. of the 8th International Conference on Intelligent User Interfaces (IUI'03), ACM (January 2003) 125–132
14. Su, C.J., Chu, T.W.: A mobile multi-agent information system for ubiquitous fetal monitoring. International Journal of Environmental Research and Public Health **11**(1) (2014) 600–625
15. Vassileva, J., Mccalla, G., Greer, J.: Multi-agent multi-user modeling in I-Help. Journal of User Modeling and User-Adapted Interaction **13**(1-2) (2003) 179–210
16. Mitrovic, N., Mena, E.: Adaptive user interface for mobile devices. In: Proc. of the 9th International Workshop on Design, Specification, and Verification - Interactive Systems (DSV-IS'02), Springer (June 2002) 47–61
17. Nathan, A.: Building Windows 10 Applications with XAML and C# Unleashed (2nd Edition). Sams (2016)
18. Mitrovic, N., Bobed, C., Mena, E.: A review of user interface description languages for mobile applications. In: Proceedings of 10th International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM'16), ARIA XPS (October 2016)
19. Morris, J.: Android User Interface Development. Packt Publishing (2011)
20. Mitrovic, N., Royo, J., Mena, E.: Adus: Indirect generation of user interfaces on wireless devices. In: Proc. of 7th International Workshop Mobility in Databases and Distributed Systems (MDDS'04), within 15th International Workshop on Database and Expert Systems Applications (DEXA'04), Springer (September 2004) 1–5