MVIC – An MVC Extension for Interactive, Multimodal Applications

Marc Hesenius and Volker Gruhn

paluno - The Ruhr Institute for Software Technology
University of Duisburg-Essen
Gerlingstr. 16, 45127 Essen, Germany
{marc.hesenius,volker.gruhn}@paluno.uni-due.de
http://www.paluno.de

Abstract. MVC is considered an important architectural patterns when it comes to interactive applications since its invention in the days of Smalltalk. However, interaction with computers has changed. Touch-screens are as natural to users nowadays as mouse and keyboard have been for the past decades of computing and HCI-researchers keep on developing more interaction modalities. Multimodal applications pose major challenges to software engineers who have to deal with different ways for users to express the same intention. MVC does not incorporate the flexibility needed to cope with multimodal applications as it makes the controller component responsible for interaction interpretation and managing the application flow. We propose MVIC, an extension to MVC dedicated to provide a solid software architecture for multimodal, interactive applications by introducing a dedicated interaction component.

1 Introduction

Software engineers have been optimising and refining the incorporation of classic interaction modalities like mouse and keyboard into software products for decades. Model-View-Controller (MVC), *the* architectural pattern for interactive applications, was introduced with Smalltalk-80 [1,5] and has ever since been used by different applications in different contexts. But with the appearance of smartphones and tablets, interaction with computers has changed. Surface gestures have become a common interaction modality and HCI-Researchers keep on adding new technologies to the existing portfolio, increasing and improving interaction possibilities.

Often, the same functionality can be accessed in different ways depending on the current situation – for example, a ringing phone can be turned off by either pressing a button or turning the device upside down. The reasons to incorporate different interaction modalities are manifold, from situational reasons to personal preferences and cultural background. Different schemes of the same interaction modality, like novice and expert gestures, may improve usability and applications could be easily adapted to the customs of different cultures.

But what maybe a wonderful new world full of possibilities to users poses major challenges to software engineers. The new interaction modalities are often highly ambiguous and not easily interpreted – interaction has evolved from basic concepts to a

K. Drira (Ed.): ECSA 2013, LNCS 7957, pp. 324–327, 2013.

highly complex matter in the last years. Recent mobile devices equipped with different sensors allow developers to implement a variety of interaction modalities. Operating systems like Android typically give support by providing recognition functions for a basic set of gestures. However, the implementation of custom gestures requires developers to work on raw sensor data. In a typical MVC setup, the controller is in charge of tracking user actions and inferring the meaning for the application, which requires developers to implement recognition and maintenance code for several input sources at one place. As applications may change often and quickly, maintenance is made difficult. Developers aim for using agile development methodologies, short release cycles and be able to extend and adapt applications to current needs. The foundation for this goal is always a sound and solid software architecture. MVC has provided this base for many years but is reaching limits when it comes to multimodal technologies.

We describe an extension to the well-known MVC pattern named Model-View-Interaction-Controller (MVIC) focused on separating *controller* and *interaction* concerns. The software's architecture should reflect the flexibility that results from the advances in HCI and allow developers to add new interaction modalities without the risk of breaking existing ways of interaction. In standard MVC, the controller is responsible for interpreting *any* kind of input and mapping it to the application's functionality, hence breaking *separation of concerns*.

Our contribution to the field of software architecture for interactive applications is twofold. One the one hand, MVIC slims down code written in the controller and introduces a dedicated interaction component, thus increasing separation of concerns. On the other hand, MVIC adapts the reliable and familiar MVC pattern to recent needs, making it easier to learn and implement in existing frameworks. This paper is structured as follows: Section 2 will introduce MVIC, its concepts and ideas and how the media player implementation can be optimized. Related work is discussed in section 3. The paper is concluded in section 4.

2 Model-View-Interaction-Controller

MVIC consists of the basic MVC elements and adds a dedicated interaction component. It is oriented towards more recent variants of MVC, removing any relationship between view and model. The components and their tasks are:

Model. The model in MVIC remains unchanged from its role in MVC. Its main task is to provide the core functionality, map real world objects into the application and provide a place to store and manipulate their current state.

View. The view has the same function in MVIC as in MVC as well – it presents data from the model to the user and is therefore in charge of the application's *output*. Different descriptions of MVC also interpret the view role in a distinct way, adding or removing *input* responsibilites (cf. [4] for a narrow and in contrast [2,5] for a wide interpretation). In MVIC this makes no difference as any input task should be delegated to the interaction component, no matter where the input information comes from.

Interaction. Here processing of all interaction takes place. In MVC, the controller solely is responsible for this task, which involves identification as well as interpretation of input. Whatever the user did is transferred to the controller and it decides how

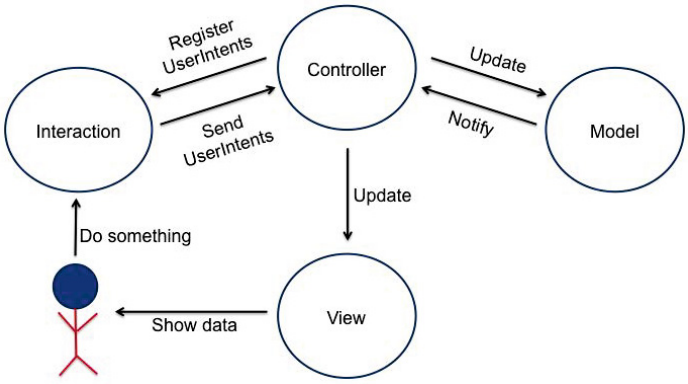

Fig. 1. MVIC Structure

the application should react. In MVIC, the interaction component is in charge of interpreting the user input: It receives input data from any device sensor, tries to figure out what the user wants to do, and sends a message to the controller containing the user's intention; hence we call this message a *UserIntent*. UserIntents should be designed to carry additional information needed for the controller to perform actions (e. g. touch coordinates). The interaction component will usually consist of several smaller components called *InteractionHandlers*, each of them dedicated to processing a certain input modality. All code previously crammed up in the controller is now distributed over various dedicated components, enforcing separation of concerns and leading to a more structured architecture. Although the InteractionHandlers map a certain kind of input to a specific UserIntent, they are oblivious to its meaning. Mapping the UserIntent to functionality and calling the model is still the controller's task.

Controller. The main purpose of the controller remains unchanged. As in MVC, the controller is in charge of the whole application flow. However, it does not receive detailed input information from sensors anymore and is completely oblivious to how interaction processing is implemented. Instead, it waits for the arrival of *UserIntents*. As a consequence, the application needs a precise *interaction concept*, which maps user input to UserIntents to specific functionality.

Summarized, MVIC strips the controller of any detailed user input recognition, sensor data is transferred to the interaction component. This component is in charge of identifying the user's intention and inform the controller about it. Whatever actions within the application are necessary to satisfy the user's wishes is up to the controller, who will call the appropiate functions on the model.

3 Related Work

Another architecture for multimodal applications strictly following the MVC pattern is the W3C recommendation *Multimodal Architecture and Interfaces*[1] (MMI). MMI is

[1] `http://www.w3.org/TR/mmi-arch/`, Feb. 2013

divided into three components, each representing one part of MVC, and all components are interconnected via a *Runtime Framework* providing infrastructure which is not defined by the W3C recommendation and left to platform specific implementations.

MMI and MVIC share common goals and target the same class of applications but differ in details. MVIC is more oriented towards recent technology like Android and iOS, while MMI is more loosely coupled and takes e.g. distribution of components into account. MMI emphasises the use of markup and scripting, which is not specifically defined in MVIC. However, using markup for configuration of InteractionHandlers is an interesting option to define an interaction concept and is part of our future research.

How to use MMI in a mobile environment is demonstrated by Cutugno et al. in [3]. They present a framework for multimodal mobile applications with focus on the possibility to configure MMI's controller via an XML file, but they extend MMI by adding *Input Recognizers* and *Semantic Interpretation Components* for each interaction modality, providing an interesting alternative to MVIC.

The idea of UserIntents is closely related to the typical way of calling other applications in Android. When an Android application needs to trigger e.g. a phone call it will do this by invoking an `Intent` without actually knowing what application will answer. Android will answer to the user's intention by bringing up its phone app. UserIntents bring this concepts deeper into applications on a much finer level.

4 Conclusion

We presented MVIC, an extension to the well-known MVC pattern targeting multimodal applications. We expect MVIC to make interactive applications more flexible and easier to extend and maintain. MVIC is build around the idea that dedicated interaction components for the different interaction modalities are in charge of interpreting the user input and identifying his or her intentions – a task left solely to the controller in MVC. Identified *UserIntents* are then send to the controller for further processing, so in MVIC the controller is still in charge of managing the application flow. Taking the interaction matters out of the controller will decrease the chance of breaking the UI when making changes to existing or adding new interaction modalities.

References

1. Burbeck, S.: Applications programming in smalltalk-80 (tm): How to use model-view-controller (mvc). Smalltalk-80 v2. 5. ParcPlace (1992)
2. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M.: Pattern-Oriented Software Architecture, vol.1 - A system of patterns. Wiley, Chichester (1996)
3. Cutugno, F., Leano, V.A., Rinaldi, R., Mignini, G.: Multimodal framework for mobile interaction. In: Proceedings of the International Working Conference on Advanced Visual Interfaces, AVI 2012, pp. 197–203 (2012)
4. Freeman, E., Robson, E., Sierra, K., Bates, B.: Head First Design Patterns. O'Reilly, Sebastopol (2004)
5. Krasner, G.E., Pope, S.T.: A description of the model-view-controller user interface paradigm in the smalltalk-80 system. Journal of object oriented programming 1(3), 26–49 (1988)