MIMIC: Leveraging Sensor-based Interactions in Multimodal Mobile Applications

Nadia Elouali
LIFL Laboratory
University of Lille1
Villeneuve d'Ascq, France
nadia.elouali@ed.univ-lille1.fr

José Rouillard
LIFL Laboratory
University of Lille1
Villeneuve d'Ascq, France
jose.rouillard@univ-lille1.fr

Xavier Le Pallec
LIFL Laboratory
University of Lille1
Villeneuve d'Ascq, France
xavier.lepallec@univ-lille1.fr

Jean-Claude Tarby
LIFL Laboratory
University of Lille1
Villeneuve d'Ascq, France
jean-claude.tarby@univ-lille1.fr

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the Owner/Author. Copyright is held by the owner/author(s).
CHI 2014, Apr 26 - May 01 2014, Toronto, ON, Canada
ACM 978-1-4503-2474-8/14/04.
http://dx.doi.org/10.1145/2559206.2581222

Abstract

In recent years, there has been an increasing interest in the presence of sensors on mobile devices. Emergence of multiple modalities based on these sensors greatly enriches the human-mobile interaction. However, mobile applications slightly involve sensors and rarely combine them simultaneously. In this paper, we seek to remedy this problem by detailing the key challenges that face developers who want to integrate several sensor-based modalities and combine them. We then present our model-based approach solution. We introduce M4L modeling language and MIMIC framework that aim to produce easily sensor-based multimodal mobile applications by generating up to 100% of their interfaces.

Author Keywords

Sensor-based interactions; Multimodal mobile application; Model-based approach; Domain-Specific Modeling Languages (DSML).

ACM Classification Keywords

H.5.2 [Information Interfaces and Presentation]: User Interfaces - Graphical user interfaces (GUI), Input devices and strategies, Interaction styles, Prototyping, Screen design, User interface management systems (UIMS); D.2.2 [Software Engineering]: Design Tools and Techniques - User interfaces.

Input sensors	Top applications
GPS	26%
Camera	13%
Microphone (speech)	12%
Accelerometer	10%
NFC	5%
Gyroscope	2%
Compass	1%
Light sensor	0%
Proximity	0%

Table 1: Use of input sensors in top 100 Android Applications

Output sensors	Top applications
Vibrator	43%
Speech synthesis	3%

Table 2: Use of output sensors in top 100 Android Applications

Introduction

For most of us, smartphone belongs to the daily life. A current smartphone includes hundreds of applications and comes with a growing set of embedded sensors. These sensors are enabling new interaction modalities (interaction by inclining the phone or changing its orientation) while user interactions are often limited to tactile. Yet, the other sensor-based interactions could be useful in many situations. For example, taking a phone call while wearing gloves could be easier by shaking the smartphone (accelerometer) rather than removing gloves and performing the usual touch command. In the same way, user may prefer text-to-speech synthesis rather than screen for reading a text in bright sun. Exploiting more sensors for interactions enables users not only to easily interact with applications, but also to benefit from several of their senses. Moreover, using these interactions in a multimodal way allows users to select the modality (or combination of modalities) they find more appropriate to their preferences and needs. However, the mobile developers slightly involve sensors and rarely combine them. In our discussions with mobile developers, they say using sensors is practically a very challenging task. And it is a challenging task, mainly because sensors require the handling of noisy and meaningless low-level measurements without proper interpretations.

We aim to produce easily mobile applications with more sensor-based interactions. We apply a model-based approach that increases the abstraction level and address the development issues of such applications. In this paper, we identify the key challenges that developers face when dealing with sensor-based interactions. We introduce and discuss our model-based approach solution. Problems of existing approaches are also discussed.

Challenges and related work

Our analysis of top 100 Android applications shows that the mobile input/output sensors and their associated modalities are still barely used (see tables 1 and 2). Developers are facing the following challenges:

Rapid advance of mobile devices and difficulty of sensor-based interaction development: Several sensors are now provided by mobile devices: accelerometer, linear acceleration sensor, gravity sensor, magnetic field sensor, orientation sensor, gyroscope, light sensor, proximity sensor, GPS, pressure sensor, temperature sensor, etc. New sensors are increasingly integrated to the mobile devices like gesture sensor, hall sensor, humidity sensor, step detector and step counter sensors, etc. Other ones are external and can be connected to the mobile through Bluetooth for example like connected watches and glasses, or specific devices like IT Sensor Tag or Arduino board. The set of sensors becomes larger and gains importance. However, developing application that uses sensors freshly integrated is generally not easy. For most of them, developer has to transform noisy and low-level data (x,y,z for accelerometer and magnetometer, distance in centimeters for proximity sensor...) to high-level interaction (shaking the smartphone, making a gesture in front of it). It also requires extensive user testing in order to figure out an efficient and accessible ways of interacting.

Inputs and outputs interactions are strongly related: In mobile applications, feedbacks are important. Input interactions have direct effects on the application outputs in order to provide feedbacks: a change on the GUI, a vibration or a sound. When a user employs an unusual modality, it is crucial to notify her/him that s/he has (correctly) performed a command. For non-tactile interactions, one cannot be sure that user will be able to see the screen. Developer has to select the adequate

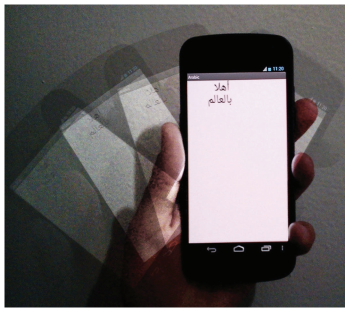

Figure 1: Shaking accelerometer-based interaction

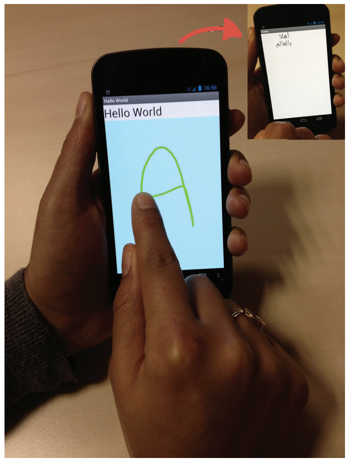

Figure 2: Gestural tactile interaction

output modality to provide feedback. So, dealing with multimodal interactions is a concern that includes both input and output modalities.
Conflicts between sensors-based interactions: Relations between mobile sensors are pretty significant. The gravity sensor is a virtual device based on the accelerometer, the magnetic field sensor provides a location-based services, the proximity sensor is based on the light sensor, etc. Some interaction forms are detected by several sensors with varying accuracy (the phone rotation can be detected with the orientation sensor or gyroscope). Developer has then to choose the most accurate sensor to detect an interaction. Moreover, this interaction should not come into conflict with other interactions detected by the same sensor at the same time. Conflicts can be detected when developer programs manually sensor-based interactions, but this refers to the problems of low-level programming. In high-level, an overview of the application interactions may help developer to avoid conflicts. However, developer should have more details about sensors (s/he needs to know if a sensor is virtual or real, if it is deprecated or not,...).
Modalities synchronisation: The development of synchronization mechanisms to manage combinations between interaction modalities (fusion and fission) takes a lot of effort and requires a significant amount of low-level programming. This challenge is not specific to mobile applications development even if synchronization of sensor-based events is more difficult if one considers the noise of sensors.

During the last decade, several model-based approaches have been proposed to address issue about multimodal interactions (ACICARE [8], OpenInterface [9], SMUIML/HephaisTK [5], IMBuilder [1]...). However, they are limited concerning the context of current smartphones. They provide a very limited set of sensors-based interaction modalities and do not keep the rapid growth of mobile sensors. Few of these approaches (SMUIML/HephaisTK [5]) provide modalities based on RFID and GPS while others propose to extend their frameworks in order to integrate new ones (OpenInterface[9]). However, it is not always possible to integrate new modalities in existing tools. For example, there are some interaction modalities based on virtual sensors (proximity sensor, gravity sensor...), which are not compatible with the modality definition adopted on the most modeling languages: Modality=<physical device, interaction language> [2].
The existing approaches rarely address both input and output interactions. So if a designer/developer wants to build a multimodal mobile application, s/he must use at least two model-based tools. The cooperation between these tools is difficult; if not impossible. They are not only technically different, but also based on different modeling paradigms (State machine, Petri net, Hierarchy and Component paradigm [6]). Models are, therefore, not compatible with each other. In addition, they cannot give overviews of the whole multimodal interactions (inputs and outputs) while this is the first interest of modeling.

In the next section, we present a model-based approach that we have developed in order to address the previous challenges and limitations of existing model-based solutions. It includes a modeling language and its related framework, respectively called M4L and MIMIC.

Our MDE-based approach

As many propositions related to multi-modality, we have adopted a model-based approach. In this approach, we address previous challenges as following. First, we propose a modeling language based on one paradigm for input and

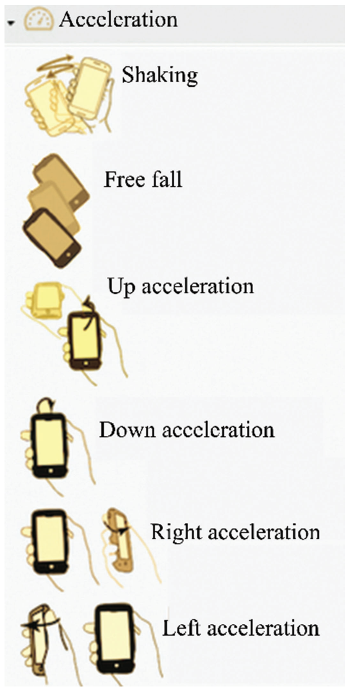

Figure 3: The accelerometer-based interactions menu in MIMIC

output multimodal interactions. Second, we provide a collection of ready-to-use sensor-based interaction modalities with the possibility to add new ones in order to address the rapid advance of mobile sensors. We also provide a set of Tycoon (TYpes of COOperatioN) [7] and CARE properties (Complementarity, Assignment, Redundancy, and Equivalence) [2] to manage combinations between modalities (fusion and fission mechanisms) and facilitate their synchronization. Finally, we give great attention to conflicts between sensors-based interactions by providing some modeling guidances in addition to the overviews of interactions.

M4L modeling language

Based on the state-machine paradigm and inspired from existing modeling languages like SMUIML [5], we defined our DSML M4L (Mobile MultiModality Modeling Language) to model input and output multimodal mobile interfaces. We chose the state-machine paradigm since it fits well with specification of multimodal interactions. It is very useful for modeling multimodal commands as shown in [3], allows modeling complex interaction dialog as shown in [4], very useful to specify the cooperation properties of multimodal commands and provides a powerful way to describe dynamic behavior of applications. The main concepts of M4L are the following: **Interaction state.** We model the mobile application as a state-machine. States correspond to the application screens while transitions refer to the interaction events exchanged between the user or her/his environment, and the multimodal application. **Interaction event.** We define an interaction event as actions issued by the user, the environment or the system to trigger a process (a treatment). From a system viewpoint, there are two types of events. Input events: provided by the user or the environment and received by the application, such as shaking the phone or providing GPS coordinates. These interaction events serve as transitions between the interaction states. Output events: provided by the application and received by the user, such as displaying a widget or read a text with speech synthesis. The only difference between input and output events is that output ones can be permanent or transient. A transient event is volatile such as a short vibration or quick message while permanent one is more durable such as an image displayed until the state changes. **Input/output modalities.** Each interaction event is typed by its modality. For instance, a user can orient the phone to the left (orientation modality) or touch a button (tactile modality) to trigger music. We can also illustrate this with an example of output events: an application can notify the user by displaying a message (display modality) or by vibrating the phone (vibration modality). **Combination**. We have integrated in M4L the support of the different combinations of interaction events defined by CARE/TYCOON classifications: complementarity, assignment, redundancy, equivalence and concurrency. **Effect.** The designer usually specify an event in order to associate an effect with it. Input events have effect on the multimodal system such as changing the interaction state, triggering a vibration, changing the size of an image or even access to a database. Output events have effect on the user which allow her/him to take decision about the next interaction.

Software support for M4L : MIMIC (MobIle MultImodality Creator)

MIMIC (MobIle MultImodality Creator) is our modeling environment. It allows direct and graphical manipulation of the M4L's concepts to model multimodal mobile interfaces. We used Obeo Designer (`http://www.obeodesigner.com/`) for the complete specification of our language and the development of the

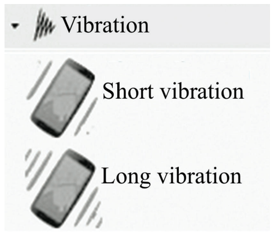

Figure 4: The vibrator-based interactions menu in MIMIC

model editor. Obeo Designer is a meta-case tool dedicated to MDA (Model Driven Architecture). It permits to create Eclipse-integrated graphical editor for any metamodel. A strong feature of MIMIC is the automatic code generation for any model defined with its model editor. We implemented the generator with Acceleo. We have chosen to generate code for the Android platform because it is currently the most widespread operating system. However, we are currently implementing code generators for iOS and HTML5. We took great care to define our metamodel as platform-independent as possible for this multi-platform perspective.

Modeling. To model a multimodal mobile interface using MIMIC, the designer can start by defining the applications screens (states). For example, to model the example in figure 5, s/he starts by modeling the "Example" state. Then, s/he associates the different input ("shaking", "close" and "falling") and output interactions events ("MIMIC demonstration", "snoring", "shout" and "shake") typed by their interaction modalities. S/he can also add cooperation operators between these events ("Complementarity"). The combination operators are tightly linked to the C, R, E and CC (Complementarity, Redundancy, Equivalence and Concurrency) properties. The toolbar contains the basic concepts that can be added to the model with a simple drag and drop. In addition, a control panel visualizes and allows modification, if necessary, of the concepts attributes (describes the interface elements like background color, font, width, etc.). In order to help designer/developer and guide her/him while modeling we currently provide tutorial documentation and some guidances through comments that appear when selecting a modeling concept. We define also a set of OCL (Object Constraint Language) constraints and integrate them in our editor in order that designers can check their models.

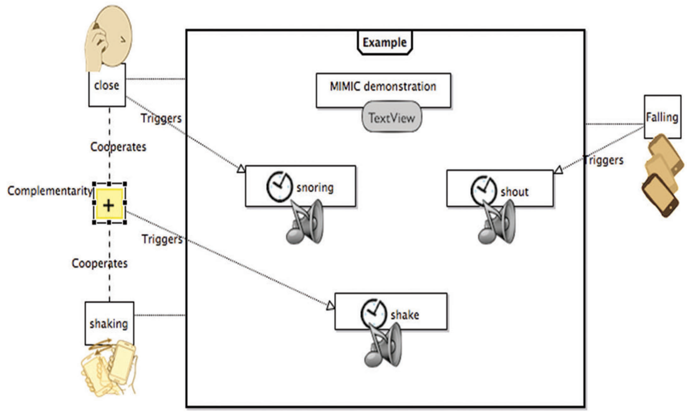

Figure 5: A modeling example created using MIMIC (The smartphone screams if it is falling, snores if it is covered and if one covers it while shaking, it emits a sound of shaken matchbox)

Library of interaction events. We address the challenges corresponding to the rapid advance of mobile devices and difficulty of sensor-based interaction development with our interaction events library. This library unifies various high-level sensor-based interaction events. It thereby hides the complexity of the technical implementation (x, y and z for accelerometer, distance for proximity sensor, etc) from the user. These events are already tested with end-users in order to figure out accessible ways of interaction (figure 3 and 4). At the moment, the library contains a total of 33 ready-to-use input and 55 output events (49 of which are widgets). They do not encapsulate code, but create already filled model elements that will be used to generate the corresponding code. For example, if a designer needs to define a phone shake as an input event to her/his application, s/he can drag the event called "Shaking" from the toolbar and drop it in the model (see figure 5). After completing the model, the shake corresponding code will be generated automatically. Some interaction events are defined based on different sensors in order to allow developers choosing the appropriate sensor. For example, up, down, left and right phone orientations are defined (calculated) using the accelerometer and the orientation sensor. If the development is on a platform that neglects the orientation sensor, the same events based on the accelerometer can be used. To resolve conflicts that arise between incompatible events, MIMIC provides guidances through comments that appear when selecting events from the library. In addition, as designer uses diagrams to define models, s/he gets an overview of the whole interaction. This overview as well as the modeling guidelines simplify the detection of conflicts.

Conclusion and Future Work

In this paper, we presented our model-based approach that aims to facilitate the penetration of multimodal mobile applications with more sensor-based interaction modalities. Its main contributions are that it combines various sensor-based interaction events in a common library and covers both input and output mobile multimodality. We continuously evaluate our environment with a set of multimodal mobile applications modeled and generated using MIMIC (http://www.lifl.fr/~eloualin/examples.html). Some of these applications are already present on Google play. In order to validate our work, we are currently evaluating the effect of our model-based approach on the development of multimodal mobile applications that leverage sensor-based interactions.

References

[1] Bourguet, M.-L. A toolkit for creating and testing multimodal interface designs. In *UIST'02* (2002).

[2] Coutaz, J., Nigay, L., Salber, D., Blandford, A., May, J., and Young, R. M. Four easy pieces for assessing the usability of multimodal interaction: The care properties, 1995.

[3] Cutugno, F., Leano, V. A., Rinaldi, R., and Mignini, G. Multimodal framework for mobile interaction. AVI '12 (2012), 197–203.

[4] Dumas, B., Lalanne, D., Guinard, D., Koenig, R., and Ingold, R. Strengths and weaknesses of software architectures for the rapid creation of tangible and multimodal interfaces. TEI '08 (2008), 47–54.

[5] Dumas, B., Signer, B., and Lalanne, D. A graphical uidl editor for multimodal interaction design based on smuiml. In *Proc. of Int. Workshop on User Interface Description Language UIDL 2011* (2011).

[6] Elouali, N., Rouillard, J., Le Pallec, X., and Tarby, J.-C. Multimodal interaction: a survey from model driven engineering and mobile perspectives. *Journal on Multimodal User Interfaces* (2013), 1–20.

[7] Martin, J.-C. Tycoon, six primitive types of cooperation for observing, estimating, and specifying cooperations. In *Working Papers of the AAAI Fall Symposium on Psychological Models of Communication in Collaborative Systems*, American Association for Artificial Intelligence (1999), 61–66.

[8] Serrano, M., Nigay, L., Demumieux, R., Descos, J., and Losquin, P. Multimodal interaction on mobile phones: development and evaluation using acicare. MobileHCI '06 (2006), 129–136.

[9] Serrano, M., Nigay, L., Lawson, J.-Y. L., Ramsay, A., Murray-Smith, R., and Denef, S. The openinterface framework: a tool for multimodal interaction. CHI EA '08 (2008).