

UNIVERSIDADE DE LISBOA
Faculdade de Ciências
Departamento de Informática



**PHASED - PHYSIOLOGIC ADVANCED SENSING
DEPLOYED**

André Filipe Feiteiro Justo

DISSERTAÇÃO

MESTRADO EM ENGENHARIA INFORMÁTICA
Especialização em Engenharia de Software

2014

UNIVERSIDADE DE LISBOA
Faculdade de Ciências
Departamento de Informática



**PHASED - PHYSIOLOGIC ADVANCED SENSING
DEPLOYED**

André Filipe Feiteiro Justo

DISSERTAÇÃO

MESTRADO EM ENGENHARIA INFORMÁTICA
Especialização em Engenharia de Software

Dissertação orientada pelo Prof. Doutor Luís Manuel Pinto da Rocha Afonso Carriço
e co-orientado por Luís Miguel Santos Duarte

2014

Acknowledgments

First and foremost, I would like to express my gratitude to my supervisor, Professor Luís Carriço and to my co-supervisor Luís Duarte, not only for accepting me as their student and introducing me to the world of research, but also for their guidance, advice and for being patient with me during the elaboration of this master thesis.

A huge “thank you” to my family. I am immensely grateful for their support, guidance and specially for providing me with this opportunity. I thank my father, António Justo, that inspired me to learn and be curious. To my mother, Helena Justo, for the understanding and for being always there for me. To my brother, Daniel, with whom i share some of the best moments of my life.

I would like to thank all the participants in my work’s evaluation, who have willingly shared their precious time to help me.

Last but not least, I want to thank my friends and colleagues for sharing all of the good and the bad moments and making my day better when I needed it the most.

Aos meus pais.

Resumo

Atualmente os dispositivos móveis têm uma enorme importância em vários aspectos das nossas vidas, pois a sua evolução ao longo dos últimos anos fez com que estes dispositivos agregassem inúmeras funcionalidades que anteriormente apenas existiam em equipamentos mais especializados. Os *smartphones* de última geração destacam-se pelo facto de possuírem um grande leque de tecnologias e sensores, tais como o Wi-Fi, GPS, acelerómetro, entre outros. Apesar de grande parte destes mecanismos terem sido integrados nos *smartphones* para melhorar as funcionalidades do mesmo, hoje em dia tornaram-se fundamentais para, entre outras coisas, obter informações sobre o contexto do utilizador e do ambiente que o rodeia.

Com a evolução das capacidades dos dispositivos e com a facilidade de difusão de aplicações nos principais sistemas operativos móveis, têm surgido imensas aplicações móveis sensíveis ao contexto. Estas aplicações utilizam as informações recolhidas por sensores para adaptar os seus serviços face a um determinado contexto ou atividade do utilizador. Com a possibilidade de integração de sensores nas aplicações móveis, têm surgido também diferentes cenários em diferentes domínios onde esta integração se revela uma mais-valia. Por exemplo, no domínio da saúde surgiram aplicações para monitorização remota de pacientes através de sensores fisiológicos, permitindo uma abordagem proativa em caso de emergência. Outro cenário comum é o desenvolvimento de jogos com recurso a sensores e atuadores, proporcionando uma experiência de jogo mais imersiva. No entanto, um problema pertinente é o facto de que para criar este tipo de aplicações é muitas das vezes necessário trabalhar com vários sensores de diferentes fabricantes e com diferentes especificações, tornando o processo mais difícil e moroso. Outro problema é o facto destas aplicações serem criadas com um propósito específico, invalidando a hipótese de serem adaptadas para funcionarem em distintos cenários.

Existem domínios em que a necessidade de criar e adaptar as aplicações é uma atividade recorrente. Muitas das vezes, os requisitos impostos pelo domínio variam rapidamente o que torna inviável o desenvolvimento de novas soluções num curto espaço de tempo. Por exemplo no domínio da saúde, é necessário que as aplicações sejam facilmente adaptáveis para acompanhar a evolução dos pacientes. Uma potencial solução para ultrapassar estes obstáculos é a criação de ferramentas de autoria que juntem programa-

dores e não programadores, em que os primeiros fornecem os mecanismos necessários para que os últimos possam rapidamente criar soluções que satisfaçam as suas reais necessidades. Apesar de algumas ferramentas deste tipo se terem revelado bem-sucedidas, atualmente nenhuma permite a criação de aplicações Android com recurso a sensores, de uma forma prática e acessível a utilizadores sem experiência em programação.

O principal objetivo deste trabalho é aproveitar o potencial dos *smartphones* e das informações de contexto recolhidas através de sensores para tornar as aplicações móveis reativas e proativas. Estas informações podem ser utilizadas para, entre outras coisas, iniciar interações adicionais com o utilizador. As principais contribuições desta dissertação são:

1. Um ambiente de execução de aplicações móveis que permite que as aplicações criadas através de uma ferramenta de autoria possam ser executadas nos dispositivos móveis. Para além disso, este ambiente de execução contém ainda os mecanismos necessários para recolher informações dos sensores existentes nos *smartphones* ou sensores externos. Esta informação é depois processada e utilizada para despoletar eventos que são utilizados para iniciar interações entre os utilizadores e as aplicações. Esta abordagem permite que as aplicações se tornem reativas (reagindo com uma determinada ação face a um contexto específico) e/ou proativas (as aplicações podem, por exemplo, iniciar-se automaticamente).
2. A integração de um conjunto de componentes e melhorias numa ferramenta de autoria já existente, com vista a permitir que utilizadores sem experiência na área da programação possam rapidamente criar aplicações móveis que utilizam sensores para despoletar ações nas aplicações em função de um determinado contexto.

O trabalho desenvolvido nesta dissertação assentou sobre uma plataforma de autoria já existente denominada DETACH (DEsign Tool for smartphone Application Composition - Ferramenta de Desenho para a Composição de Aplicações para Smartphones). Embora esta plataforma já possibilitasse a criação de aplicações móveis, estas não suportavam o uso de sensores e, portanto, numa primeira fase foi necessário identificar as limitações deste sistema face à integração de sensores. Uma vez que o sistema DETACH é composto por um ambiente de autoria e um ambiente de execução de aplicações móveis, foi necessário avaliar até que ponto estes ambientes suportavam os mecanismos necessários para a inclusão de sensores nas aplicações. As limitações encontradas permitiram perceber as alterações necessárias e identificar um conjunto de requisitos que serviram de base para o desenvolvimento deste trabalho.

O maior foco deste trabalho foi o desenvolvimento do ambiente de execução das aplicações móveis. Este ambiente foi desenvolvido em Android e é responsável tanto pela execução das aplicações como pela recolha de dados através de sensores. Para su-

portar aplicações reativas e proativas, foi necessário identificar uma abordagem que permitisse recolher informações de contexto, avaliá-la e posteriormente despoletar determinadas ações nas aplicações. Como tal, foi necessário escolher uma arquitetura baseada em eventos, por forma a dar o comportamento reativo e proativo às aplicações. Foram também tidos em conta outros aspetos como a modularidade, uma vez que a capacidade de adicionar novos componentes à ferramenta de autoria é um requisito fundamental para o sucesso da mesma.

Após o desenvolvimento do ambiente de execução das aplicações móveis, foi necessário melhorar o ambiente de autoria. Neste ambiente, foram introduzidos os mecanismos que permitem a criação de aplicações que utilizam sensores. Os utilizadores podem, através de programação visual, utilizar um conjunto de eventos oferecidos pelos sensores para definir o comportamento das suas aplicações com base nesses eventos. Com base em *feedback* obtido em avaliações do DETACH (prévias a este trabalho), foram ainda feitas algumas alterações na *interface* da ferramenta por forma a melhorar a usabilidade da mesma.

Na última fase deste trabalho, e com o objetivo de validar o mesmo, pedimos a um conjunto de programadores que realizassem uma avaliação em que tinham de adicionar um novo sensor ao sistema. Este processo envolveu os dois ambientes do ecossistema e permitiu-nos perceber se no futuro os programadores conseguem garantir a implementação de novos sensores. Posteriormente, os mesmos utilizadores tiveram de executar todo o processo de criação de uma nova aplicação para testar se o sensor implementado estava a funcionar corretamente. Este processo consistiu na criação de uma aplicação através da ferramenta de autoria, sincronização da aplicação para o *smartphone* e execução da mesma. Embora este grupo de utilizadores não fosse o principal público alvo desta ferramenta de autoria, esta tarefa permitiu-nos observar e tirar algumas conclusões sobre os novos mecanismos introduzidos e que suportam a criação de aplicações com sensores. Todos os participantes conseguiram terminar com sucesso as tarefas propostas, validando a nossa abordagem com vista a garantir a modularidade e capacidade de adicionar novos sensores à plataforma de autoria.

Palavras-chave: Sensores, Sistemas sensíveis ao contexto, Sistemas baseados em eventos, Ferramentas de autoria, Plataformas de execução

Abstract

The latest technological innovations contributed to the evolution of smartphones and the availability of embedded sensors, such as the accelerometer, digital compass, gyroscope, GPS, microphone, and camera are creating new application scenarios. Nowadays, mobile devices also come with a large set of resources that offer third-party programmers the tools to develop sensing applications.

Even though several domains capitalized this information to improve their applications capabilities, there are others where applications' requirements change quickly and it becomes important to have an easy and flexible development environment in such a way that the applications deployed can be rapidly tailored to each situation. Authoring tools proved to be a successful approach to overcome those problems; however, currently there is a lack of tools that support the creation of mobile sensing applications. We also believe that we can go one step further and combine the potential of smartphones and sensors' context-data to create reactive and proactive mobile applications.

This work aims at addressing the previous problems with the introduction of an ecosystem that comprises: a) an Android runtime environment that runs the applications created and uses a set of sensors to collect informations about the context of the user; b) a web authoring-tool that enables non-expert users to create mobile applications that rely on sensors. Supported sensors encompass the chronometer, GPS and a set of third-party physiological sensors (electromyography and electrocardiography).

To validate our work we conducted an evaluation encompassing developers in order to assess the complexity of adding new sensors to the platform. All participants were able to complete the proposed tasks, validating our approach and thus ensuring that in future developers are capable of expanding the authoring environment with additional sensors.

Keywords: Sensors, Context-aware systems, Event-based systems, Authoring tools, Runtime environments

Contents

Figures List	xix
Tables List	xxi
Listings	xxiii
1 Introduction	1
1.1 Motivation	2
1.2 Goals	3
1.3 Contributions	4
1.4 Planning	4
1.5 Document organization	5
2 Related work	7
2.1 Context-aware computing	7
2.1.1 Sensors	8
2.2 Sensor Runtime environments	10
2.2.1 Context in event processing modelling	11
2.2.2 Event filtering approaches	12
2.3 Sensor Programming environments	14
2.3.1 Context-aware prototyping environments	15
2.3.2 DETACH	17
2.4 Summary	17
3 DETACH Analysis	19
3.1 System architecture	19
3.2 DETACH Authoring Tool	20
3.2.1 Applications' anatomy	20
3.2.2 Interface	21
3.2.2.1 Screen templates	22
3.2.2.2 Screen triggers	23
3.2.2.3 Runtime Emulator	24

3.3	DETACH Mobile	24
3.4	Constraints in sensor integration	25
3.4.1	Runtime environment	25
3.4.2	Authoring environment	27
3.5	Summary	28
4	Adding Sensors	29
4.1	Architecture	29
4.2	DETACH Mobile	34
4.3	Adding sensors to the runtime environment	37
4.3.1	Adding a new sensor	38
4.3.2	Implemented Sensors	43
4.4	Adding sensors to the authoring environment	46
4.5	Summary	50
5	Authoring Sensors	51
5.1	DETACH interface	51
5.2	Quality of life improvements	53
5.2.1	Creating connections	54
5.2.2	Editing connection details	56
5.2.3	Deleting screens	58
5.3	Using Sensors	58
5.3.1	UI characteristics	59
5.3.2	Connection types	59
5.3.3	Available Sensors	61
5.3.4	Scenario	63
5.4	Run-Time Emulator	66
5.4.1	Scenario	68
5.5	Summary	70
6	Evaluation	73
6.1	Participants	73
6.2	Equipment and Tools	74
6.3	Metrics	74
6.4	Procedure	75
6.5	Results	76
6.6	Discussion	80
7	Conclusions	85
7.1	Future work	86

List of Figures

1	DETACH's system architecture	20
2	Anatomy of a DETACH application	21
3	DETACH interface	22
4	DETACH mobile screen templates	22
5	Connection rules specification	23
6	Example screen configuration and respective run-time result	24
7	Application running in DETACH Mobile	25
8	Runtime environment updated architecture	30
9	PHASED architecture	31
10	Sensor's workflow	33
11	Development and installation process of a new Sensor Processing Unit, and related stakeholders	34
12	DETACH Mobile status screen (left), DETACH Mobile menu options (right)	35
13	List of available sensors (left), Sensor status updated (right)	36
14	DETACH Mobile authentication screen	36
15	UML Class diagram	38
16	Shimmer EMG sensor	44
17	Sample output of Shimmer EMG	44
18	Shimmer ECG sensor	45
19	ECG waveform of a heart beat.	45
20	Sample electrocardiogram output of Shimmer ECG	45
21	Emulator <i>tabs</i>	49
22	Alternative interfaces to simulate user's current location	49
23	Comparison between initial and improved DETACH's interface	52
24	Panel which allows the configuration of a new transition between screens	53
25	Button to create connections (old interface)	54
26	Visual feedback when creating connections	55
27	AND/OR evaluation mode	55
28	Alternative ways to transit from one screen to another	56
29	Information displayed in transition's label	56

30	Screen configuration panel	57
31	Transition details displayed in the right side panel	57
32	Button to delete selected screens	58
33	Trash can area, with visual feedback when deleting a screen	58
34	Available sensors templates	59
35	Transition between screens (without the use of sensors) and corresponding triggers	60
36	Transition between two screens using a sensor and corresponding triggers	60
37	External transition using an ECG sensor and corresponding trigger	61
38	Time sensor representation	61
39	Time sensor condition trigger example	61
40	Location sensor representation	62
41	Location sensor condition trigger example	62
42	ECG sensor representation	63
43	ECG sensor condition trigger example	63
44	EMG sensor representation	63
45	EMG sensor condition trigger examples	63
46	Scenario application - phase 1	64
47	Scenario application - phase 2	65
48	Specifying a location to trigger an event when the user goes inside it . . .	65
49	Scenario final application	66
50	DETACH application running in the DETACH Run-Time emulator	67
51	DETACH run-time emulator with an extra section to simulate sensor's events	67
52	DETACH run-time emulator after pressing the "Simulate values" button .	68
53	Scenario application emulation (left), new screen shown after answering the question (right)	69
54	Screen activated (right) when João arrives at the airport (left)	69
55	Screen activated (right) when João's heart rate goes above 140bpm (left) .	70
56	EMG electrodes placement	74
57	Average time per task (task 1 to 4, respectively)	77
58	Frequency of unsuccessful attempts in template creation	78
59	Feedback about adding a new template in DETACH	78
60	Frequency of unsuccessful attempts when developing the Android module	79
61	Feedback about adding a new sensor module in Android	80
62	External trigger and a transition between screens based on a sensor trigger, respectively	82
63	Sketch of an alternative way to use sensors in transitions	82
64	Context menu after right-clicking in a screen/sensor template	83

List of Tables

1	Map between events and actions	32
---	--	----

Listings

4.1	Android Manifest XML file example	39
4.2	Sensor Service skeleton	40
4.3	Location service implementation	41
4.4	Sensor XML file structure	47
4.5	Example of an Event representation in XML	48
4.6	Location sensor JavaScript file	50

Chapter 1

Introduction

Sensors are becoming increasingly important in interaction design and sensing technologies are becoming pervasive. A significant effort in human-computer interaction has been dedicated to create user-interfaces that rely on sensorial mechanisms.

Nowadays we have a lot of different sensor types such as environmental sensors (e.g. light, temperature, noise), motion sensors, physiological sensors, among others. In addition, most recent smartphones offer numerous technologies, such as the mobile network, Wi-Fi and internal sensors (e.g. GPS¹, accelerometer, gyroscope). This proliferation of sensors has created many usage scenarios, which can be enabled with their proper integration into devices of all kinds.

With the evolution and dissemination of smartphones, the integration of sensors in mobile applications becomes more and more a reality. One of the promising fields is in the domain of healthcare and wellness management. New application scenarios have emerged, such as mobile and remote healthcare services for elderly people, self-monitoring services on physical exercise or physical rehabilitation, remote health monitoring services in emergency situations, among others [1]. Over the years, healthcare evolved to a more proactive approach where it is essential to have the ability to detect and prevent certain conditions in patients, and manage their well-being over time [2]. Another scenario, with fewer clinic concerns, is the development of games which use sensors and actuators (e.g. vibration) to interact with the environment of the game, providing an immersive game experience.

Despite the numerous existing scenarios, the need to create and adapt applications for mobile devices is in some areas a recurring activity. In addition, it is also necessary to take into account the complexity to integrate the different sensors available in a practical and fast way, in order to match the needs of each application. This issue can be addressed by creating an environment encompassing programmers and non-programmers, in which

¹Global Positioning System

the former provide the necessary mechanisms so that the latter can create applications that fulfill their needs, even without programming experience.

To achieve our goals, we developed a set of components responsible for collecting data from sensors. We also extended an existing authoring tool by developing a set of features which enable non-expert users to access sensor's data and use it in their applications.

1.1 Motivation

The evolution of smartphones and proliferation of all kinds of sensors enabled new applications across a wide variety of domains, such as healthcare [3], social networks [4], environmental monitoring [5] and transportation [6]. Until recently, research in mobile sensing activities, such as user's activity recognition (e.g. walking, driving, sitting) required specialized mobile devices or platforms to be fabricated. However, the availability of cheap embedded sensors initially included in smartphones (e.g. accelerometer used to change the display orientation) are creating possible new applications [7]. In addition, smartphones come with a large set of resources that offer third-party programmers the tools to develop sensing applications, such as monitoring a user's well being, tracking sport's performance, among others [7]. In the health field, several cardiac monitoring systems emerged, such as iRhythm [8], Corventis [9] and Toumaz [10]. In general they have sensors for vital sign monitoring (e.g., ECG², blood pressure) and motion monitoring through accelerometers, gyroscopes and similar sensors.

Despite using new technologies, the main shortcoming of these systems is the fact that they continue to be dedicated systems, with a very well defined purpose. While this may not be an issue in some domains, there are some cases where the requirements imposed by the domain, change so quickly that it becomes important to create an easy and flexible development environment in such a way that the applications deployed can be rapidly adapted to each situation, without expensive development costs. We also believe that we can combine the potential of smartphones and sensors to create richer and proactive applications. We intend to use smartphones as a personal activity coordination device with many capabilities such as scheduling (e.g. calendaring), personal and generic context-awareness (e.g. GPS location), and others. This context information may be used to adapt the process of sensor data collection and also to initiate additional interactions with the user [11]. Some challenges for context-aware sensing are the diversity of context environments, the range of physiological conditions and issues related to sensors, such as overcoming sensor noise, sensor failure and smooth context recognition [12].

Our motivation is to create a tool that allows users to easily and rapidly prototype

²Electrocardiography

mobile applications, addressing the lack of authoring tools that supports the creation of mobile sensing applications. We believe that there are a lot of applications that can be supported by context-information provided by sensors. That information can either be used for simple analysis or for more complex situations. A few examples are:

- Continuous heart monitoring, using sensors like ECG and store the information for later access - typical passive use of sensor data.
- Monitor a user's chronic disease and in case of failure, automatically call for help - typical active use of sensor data.
- Use motion sensors to create custom movements and match them with specific actions.

We want to collect and transform that information, and provide users with a set of events that can trigger specific behaviours in an application.

1.2 Goals

The main goal of this work is the development of an ecosystem that provides the necessary tools to let users compose mobile applications and afterwards, run them in Android devices. Moreover, our work focuses in integrating sensors in the applications created, to collect user's context-information and create context-aware mobile applications.

To achieve this, our system must provide two different tools:

- **A web authoring-tool** which allows users without programming skills to create mobile applications. This tool let users customize the applications' look and feel. Users can also customize the applications' behavior and use sensors to create applications that adapt to the context of the user.
- **An Android run-time environment** that interprets and runs the applications created with the web authoring-tool. In addition, the run-time environment is responsible to provide a set of sensors that collect informations about the context of the user. Therefore, it should be able to:
 - **Collect data** from external sensors that are connected to the smartphone via Bluetooth or internal sensors that exist in smartphones (e.g. GPS, accelerometer).
 - **Work with several types of sensors** and be flexible in order to easily allow the addition of new sensors to the system.

- After the collection of data, optionally, it should be possible to **apply some types of processing**. This means that for example, we can use the ECG raw data to draw a graph of the user’s heart rhythm or process it to get user’s heart rate.
- **Work like an event-based system**: Each sensor has a user-defined set of rules that define the system events. This will allow the applications to react differently when facing different contextual information.

1.3 Contributions

This work’s main contributions are:

- Development of an Android runtime environment capable of collecting context data from sensors (internal or external to the smartphone). Additionally, this data can be used to trigger events within mobile applications, making them not only reactive but also proactive.
- Improvements in DETACH’s authoring environment³ to support the creation of sensor-based mobile applications. We included the required mechanisms to allow non-expert users to design applications that can have different flows according to conditional transitions based on sensor’s data.

The following publication emerged from this work:

- André Justo, Luís Duarte, Luís Carriço (2014). *Integrating Sensors in a Mobile Application Authoring Environment*. HCI 2014 - 28th International British Computer Society Human Computer Interaction Conference, SouthPort, UK, September, 2014.

1.4 Planning

The development of this thesis work was divided in the following phases:

- **Phase 1 - September and October 2013: learning process of the functioning of the sensors and definition of PHASED system**

The purpose of this phase was to learn how to work with the sensors available and how to work with some of the technologies that were used in the development process. Additionally, it was also necessary to start working on some aspects related to PHASED, such as the system requirements and the system architecture.

³<http://accessible-serv.lasige.di.fc.ul.pt/~afjusto/detach/webtool>

- **Phase 2 - November 2013 until January 2014: development of PHASED**

During this phase, we started the development of the system. It also included the writing of a preliminary report describing the work made until then and how the work should evolve until the end.

- **Phase 3 - February and March 2014: development of DETACH Mobile**

This phase consisted on the improvement of several DETACH Mobile functionalities.

- **Phase 4 - April until July 2014: DETACH's interface improvements and integration of PHASED and DETACH**

During this phase, several interface improvements were made. Some of those improvements were the result of feedback obtained in previous DETACH trials, while others were related with the inclusion of sensors in the tool.

- **Phase 5 - August to October 2014: System evaluation and writing of the Master's thesis**

Establishment of an experiment to evaluate the work done. Writing of this document with all the developed work and results achieved.

1.5 Document organization

This document is organized as follows:

- **Chapter 2 - Related work**

In this chapter we focus in some topics and definitions related to the work domain.

- **Chapter 3 - DETACH Analysis**

This chapter gives an overview of the DETACH System.

- **Chapter 4 - Adding sensors**

This chapter describes PHASED - an Android runtime environment to collect user's context data from sensors and run applications created with DETACH. Here, we also explain the architecture used to tackle the requirements of this work. In the end, this chapter explains how new sensors can be added to the system, from developers' point of view.

- **Chapter 5 - Authoring Sensors**

In this chapter we present several DETACH's interface improvements, mostly due to the integration of sensors in the authoring environment. We also present the process of creating applications with sensors.

- **Chapter 6 - Evaluation**

This chapter presents the results of an experiment done with developers, to evaluate the process of adding new sensors to the system.

- **Chapter 7 - Conclusions**

This chapter presents a discussion of the full work accomplished and point out possible future work.

Chapter 2

Related work

This chapter presents some aspects related to this project domain. We start by defining context-aware computing and the importance of context in this field of research. After, we present an overview of a wide range of sensors available today, either embedded in modern smartphones or through dedicated sensor platforms. Since one of the goals of our work is to use context-data to create an event-based system, we also describe the importance of context in these type of systems and explore some approaches used to create systems of this kind. At the end, a description about programming environments is given.

2.1 Context-aware computing

Context is an important factor in human behaviour because people act based on that. The context may relate to external conditions, location, weather conditions and time of the day, among others [13].

In the last years, several context-aware applications emerged, incorporating various kinds of relevant context information in order to improve their usability. Context awareness describes the ability of a system to sense user's state and the environment, and modify its behavior based on this information [12]. Gartner [14] defines "context-aware computing" as the concept of taking advantage of information about the end-user to improve the quality of the interactions. Most of the software only uses direct relevant information like direct inputs, while context-aware systems also acts upon indirect relevant information. The utilization of context in processing is a big step towards modelling the real world behaviour. Current technology can provide us with information such as user location, user activity and light and noise levels [2]. Location context coupled with user's identity can be very useful for various service applications [13] and for example a few location-aware guides have been designed for city tours [15].

An architecture that is aware of the end user's context and delivers relevant information was first presented by Gartner as Context Delivery Architecture (CoDA) [14]. CoDA is aimed for event-driven applications and in summary, the functioning of the software elements (e.g., services or event handlers) is determined by the input to the element and by secondary sources of information – the context offered by external sources such as weather or traffic services. That means that two invocation of the same service with the same parameters may yield different results in different circumstances. It also introduces the concept of creating applications through reusable components and aims to enhance user's experience using the knowledge of context and adapt the application behaviour to it.

2.1.1 Sensors

Context can be acquired by means of different sensors that gather pieces of context information and supply the information for interpretation and utilization in applications. With the deployment of sensor networks it is not only possible to obtain real-time information about the physical world but also act upon that information. The accuracy and the timeliness of this information is important since actions are usually taken based upon these sensed values. In general, the quality and the reliability of this data are important issues that have received attention [16, 17]. Acquiring context information can mainly be done in two ways:

1. **Explicit acquisition** - user is aware that context information is being collected for use by an application and usually needs to give consent for the information to be collected. For example, location sensors in web browsers require authorization to obtain information about the user's current location.
2. **Implicit acquisition** - user is normally unaware that context information is being collected either for use by an application or for storage. For example, when the application uses an accelerometer to implicitly determine the speed at which the user is travelling while accessing the application. Another example of implicit context information acquisition include the recording of a user's previous interaction with the system [18, 19].

In 2004, Michael Beigl et. al [20] conducted a survey to identify the most typical sensors needed for applications in Ubiquitous and Pervasive Computing. Through a survey of 12 typical existing and implemented applications, they identified 7 general types of sensors - namely movement, light, force, temperature, audio, humidity and proximity. The distribution of sensors among the analyzed applications showed a preference to sensors for movement, followed by light and force/pressure. These sensors are especially suited for activity recognition of the object to which the sensor is attached (e.g. to derive

the *general internal context of an object*). However, two more sensors must be added to the list of most used sensors: temperature and audio. Both sensors are commonly used to derive information about the environment of the object. These sensors are therefore useful for deriving the *situational context of an object*.

Since then, mobile phones evolved as a computing platform and acquired richer functionalities and these advancements often have been paired with the introduction of new sensors. For example, accelerometers have become common after being initially introduced to enhance the user interface and use of the camera. Now, among other things, they are used to automatically determine the orientation in which the user is holding the phone and use that information to automatically re-orient the display between a landscape and portrait view or correctly orient captured photos.

Most modern smartphones include several dedicated sensors, such as gyroscope, compass, accelerometer, proximity sensor and ambient light sensor. They also include other more conventional devices that can be used to sense, such as front and back facing cameras, microphones, GPS, WiFi, and Bluetooth radios. Many of the newer sensors are added to support the user interface (e.g. the accelerometer) or augment location-based services (e.g. the digital compass).

Several domains capitalized the potential of sensors to improve their application's capabilities. One of the most common type of context-aware applications are based on user's location. Applications such as Foursquare¹ and museum or tourist guides heavily rely on this information to adapt the information that is presented to the user and are very popular nowadays. GPS is also commonly used in applications to track sport's performance, such as Runstatic² and Endomondo³. Non-phone-based mobile sensing devices such as the Intel/University of Washington Mobile Sensing Platform (MSP) [21] have shown value from using other sensors not found in phones today (e.g., barometer, temperature, humidity sensors) for activity recognition; for example, the accelerometer and barometer make it easy to identify not only when someone is walking, but when they are climbing stairs and in which direction. Other researchers have embedded sensors in standard mobile phone earphones to read a person's blood pressure [22] or used neural signals from cheap off-the-shelf wireless electroencephalography (EEG) headsets to control mobile phones for hands-free human-mobile phone interaction [23].

Even though it's evident that sensor capabilities heavily contributed to improve applications used in our daily lives, there is also a problem associated with such applications targeted to the general population: most of them are not flexible in order to meet some domain's requirements. If a user requires a specific functionality, it would be necessary

¹<http://foursquare.com>

²<https://www.runtastic.com>

³<http://www.endomondo.com>

to deploy a whole new version of the system, which is not practical. Instead of creating custom made applications from scratch, a possible solution may be the development of sensor runtime environments where multiple sensors can be integrated to provide an easier way of creating mobile sensing applications.

2.2 Sensor Runtime environments

Event driven architectures have evolved in the last years, changing the traditional computing architectures that are based on synchronous request-response interactions between clients and servers. This paradigm shift brings two critical changes:

- Event-based systems support applications that are reactive in nature. Being reactive means that the processing is triggered in response to events, instead of the traditional responsive applications in which processing is done in response to a specific request [24];
- Event-driven architectures adopt the decoupling principle, in which there are event producers, event consumers and event processing agents [24].

The next phase of the evolution of those systems introduces a conceptual architecture known as proactive event-driven computing [24]. Proactivity refers to the ability to identify desired future events and take advantage of future situations by applying prediction and automated decision making technologies.

The growing availability of cheap and pervasive sensor technology, the spreading of broadband connectivity and the developments in predictive analytics technology are some factors that make those systems possible [24]. Analytics has evolved from being descriptive (understanding of historical data), to being predictive (providing forecasts of future behaviour). The next step is prescriptive analytics which means the use of data to prescribe the best course of action to realize the best outcome [25]. This way, reactive computing coupled with predictive analytics brings the ability to react to events before they occur, which is the main objective of proactive event-driven computing.

The conceptual model of proactive event-driven applications is built over the existing work in event processing architectures. The core of this model is the notion of event processing network (EPN). This notion has been defined by Luckham [26] and refined by Etzion and Niblett [27] as a collection of event processing agents (EPA), producers and consumers. Those agents are in an intermediary layer that stands between the event producers and consumers. The main types of event processing agents are:

- Filter - decides whether an event continues to flow in the system, based on satisfaction of an assertion;

- Transformation - derives events as function of input in various ways (for example, aggregation);
- Pattern detection - derives events based on detection of patterns.

This work follows the guidelines of an event-based system and will focus mainly on filter and transformation agents. Filter agents will be used to process events triggered by rules with one condition only, such as the current location of the user. Transformation agents are useful to process more complex events and will be essentially used to aggregate various rules (e.g., current location of the user and time of the day). The agents will use the data collected by the sensors and analyze each sensor's rules in order to trigger, or not, the action attached to the event.

2.2.1 Context in event processing modelling

In event processing, context plays the same role that it plays in real life: a particular event is processed in a different way depending on the context in which it occurs and may be ignored in some contexts. In context aware computing it might be useful to take a set of events and group them into context partitions so that they can be processed in a related way. The way events are grouped is defined by context dimensions that tell us what aspect of the event is used to do the grouping. There are various context dimensions, such as temporal, spatial, state oriented and segmentation oriented [13].

Temporal context

This type of context is aimed to partition the temporal space into time intervals. The typical use of temporal context is to group events to process them together based on the fact they have occurred during the same time interval.

Spatial context

Spatial context groups events according to their geospatial characteristics. There are three types of spatial contexts:

- Fixed location - context partitions are associated with the location of a reference entity called geofence. An event is classified into a context partition if its location is inside the spatial entity.
- Entity distance location - context partition is based on the distance between the event's location and some other entity.

- Event distance location - this type specifies an event type and a matching expression predicate. If an event occurrence that matches the predicate is detected, then a new partition is created.

Segmentation oriented context

State-oriented context differs from the previous dimensions because the context is determined by the state of some entity that is external to the event processing system.

Our work will mainly use temporal and segmentation oriented context dimensions. When an event is triggered by a rule with one condition only (e.g. heart rate above 80bpm) it is not required to use any context dimension presented above. However, when using rules with several conditions (e.g. heart rate above 80bpm and current location of the user) it is required to group those events in a temporal context (to ensure that both events occurred in the same time span) and in a segmentation oriented context (to ensure that both events belong to the same rule).

2.2.2 Event filtering approaches

The approach chosen by each system is very important because it defines how things work and therefore it defines key aspects of the system such as the importance of the mobile device or the amount of data retrieved from the sensors (and consequently the amount of processing needed to compute it all).

Many health monitoring systems use a 3-tier remote monitoring architecture [28, 29] where a personal device such as a smartphone acts as a hub that collects data from body-worn biomedical sensors and later transmit the data (with or without additional processing) to a backend server. Some prototypes that rely on the 3-tier architecture are geared towards low-intensity sensors (e.g., weight scales or glucose readings) [30]. In these cases the mobile device doesn't do any local processing and the data is sent to the remote server for either real-time or offline analysis.

In [31], a middleware known as HARMONI (Healthcare-Oriented Remote Monitoring) uses a rule engine on the mobile device to support context-aware distributed event processing where the mobile device dynamically changes its processing of the sensor streams based on changes in both local and external context. Processing refers to operations that may be performed on the sensor data, such as filtering (e.g., heart rate between 70-90), statistical summarization (e.g., average of heart rate readings over a period of

time) or feature extraction (e.g., obtain the QRS⁴ components of an ECG signal). HARMONI also supports the concept of adaptive sensor event processing. As monitoring applications are typically interested only in higher-layer events (e.g., arrhythmia from ECG readings) or statistical summarization, HARMONI uses a spatio-temporal processing on the mobile device to filter relevant data. In [11] there's also another approach denoted as activity-triggered deep monitoring (ATDM) that focus on applications that need to transmit potentially high-rate data streams of data, but only when a specific contextual activity of the patient is detected.

The activity-triggered deep monitoring and HARMONI approaches have the advantage of reducing the amount of data collected by each sensor. This is very helpful to reduce the amount of processing that has to be done and avoids wasting unnecessary resources. However, in order to dynamically change the process of collection of data, other techniques are required to understand the activity or the context of the user (e.g. if user is walking, running or sleeping). This can be a limitation if we don't have the resources to achieve that.

Other approaches focus on the continuous transmission of data from various 'always-on' sensors. To reduce the higher volumes of sensor data that will arise from the use of continually-transmitting sensors (such as ECG or EMG), Mohamed et al. [31] proposes the use of data management middleware that distributes the processing across both the mobile device and the backend. The principal focus is the exploration of the user's context (both physiological and activity) to adapt the stream processing logic on the client device. This focus is motivated by the fact that both the medical events being observed and the expected values for medical parameters (e.g., heart rate) are often a function of an individual's activity (e.g., walking or running) and her medical context (e.g., prescribed medication).

Continuous sensing raises considerable challenges in comparison to sensing applications that require a short time window of data. There is an energy tax associated with continuously sensing and potentially uploading in real time to the cloud for further processing. Solutions that limit the cost of continuous sensing and reduce the communication overhead are necessary. If the interpretation of the data can tolerate delays of an entire day, it might be acceptable if the phone can collect and store the sensor data until the end of the day and upload when the phone is being charged. However, this delay-tolerant model of sensor sampling and processing severely limits the ability of the phone to react and be aware of its context. Successful sensing applications will have to be smart enough to adapt to situations. There is a need to study the trade-off of continuous sensing with the goal of minimizing the energy cost while offering sufficient accuracy and real-time

⁴The QRS complex is a name for the combination of three of the graphical deflections seen on a typical electrocardiogram (ECG)

responsiveness to make the application useful.

In our work, users have the ability to manually start and stop sensors. When a sensor is started, it uses a 'always-on' approach to constantly collect and process data, and eventually fire events. However, our system also supports the ability to use context information from "low-cost" (in terms of resources consumption) sensors (e.g. Time sensor) to activate "heavier" sensors (e.g. GPS) that usually drain considerable amounts of battery when used. As an example, users can personalize the activation of sensors based on his/her daily activities. The main benefit of this approach is to reduce sensors' uptime and consequently reduce the amount of processing that needs to be done and minimize the energy problem associated with continuous sensing.

2.3 Sensor Programming environments

End-user programming allows people who are not professional developers to create and modify software. The programming environments used by end-users include spreadsheet systems, authoring tools and visual programming languages. With these tools, users are able to write and edit formulas, drag and drop objects to create their programs [32]. Visual programming consists in moving graphical elements instead of typing code to program something. This type of programming is very attractive to young people who have the goal to create simple applications or simple interactive games and is very popular in educational technology. Scratch [33] is a visual programming environment that allows users to use event driven programming to create interactive projects. Users have at their disposal a set of blocks that can be snapped together to create a sequence of commands and the blocks visually fit together like puzzle pieces. There are four kinds of blocks: command blocks, function blocks, trigger blocks and control structure blocks. Scratch is a powerful tool and allows to create a wide range of projects, including animated stories, games and educational projects.

Inspired by Scratch programming language and environment, Catroid [34] is a similar tool intended for the use by kids and runs on smartphones and tablets. Programs are constructed using visual Lego-style pieces where commands are stuck together by arranging them visually. Catroid focus on devices with multi-touch screens and take advantages of many sensors built into smartphones/tablets such as acceleration or gyroscope sensors, or GPS for location based programs. Catroid also allow to wirelessly control external hardware such as Lego robots and other devices.

In [35], a programming environment to create physics-based games (such as Angry Birds) called Fizz is presented. This system allows children to produce games and simulations using events and drag and drop programming. The main idea of Fizz is that scenes of

games, stories or simulations are created from toys which have properties and behaviours. The properties include colour and the 2D physics simulation characteristics shape, mass, and elasticity. There are default properties so that children do not need to modify them unless they wish to. Behaviours are lists of actions the toys perform when certain events occur. A careful combination of these foundations enables children to produce interesting and engaging programs.

Although none of the presented tools is related to this project main goal, this research shows that visual programming tools can be useful to domains where stakeholders are non-expert programmers and therefore can create personalized applications if they are supported by the necessary tools. Having said that, we believe that applications can take advantage of using sensors in many ways if non-expert users have the ability to control those using only visual elements, making the production of applications much simpler and richer.

2.3.1 Context-aware prototyping environments

Over the years, several systems and infrastructures have been developed to provide developers with toolkits that enables the creation of context-aware applications. Even though those tools and technological advances for acquiring contextual information contributed to the development of numerous context-aware applications, there was still a lack of authoring environments supporting both programmers and end-users. This lack of support closes off the context-aware application design space to a larger group of users. Without a proper authoring environment, developing a context-aware application requires developers and end-users to work together to either build an application from scratch (involving direct interaction with hardware sensors and devices) or to use low-level sensor toolkits. Either way, in order to acquire context information, large amounts of code must be written to develop simple sensor-based applications. Those limitations narrow the design of interesting applications because end-users have less control over how the applications behave. Since they have more knowledge about their activities and surrounding environments than anyone else, they need the ability to create and modify applications as those activities and environments change. In summary, end-users without technical expertise must be able to rapidly prototype applications and have control over such applications, or else they might fail to meet users' needs.

To address this problem, some systems were created to empower end-users in building context-aware applications by lowering barriers and allowing the development of applications that match their needs without requiring them to write code [36, 37, 38]. iCAP (Interactive Context-aware Application Prototyper) [36] is a system that allows end-users to visually design a variety of context-aware applications based on if-then rules, tem-

poral and spatial relationships and environment personalization. iCAP allows users to prototype applications by describing situations and associate specific actions to it which results in exerting control over sensing systems and dictate application behavior. Topiary [37] was created to enable the rapid prototyping of location-enhanced applications. Topiary builds storyboards from scenarios that represent local contexts. These scenarios are demonstrated by the end-user and the constructed storyboards describe interaction sequences that can run on mobile devices. “a CAPella” [38] is a context-aware prototyping environment that gives the end-user the ability to program by demonstration. Its main components are: a recording system, an event detection engine, a user interface and a machine learning system. More recently, Realinho et. al [39, 40] developed the IVO (Integrated Virtual Operator) platform which also enables end-users to build and deploy context-aware applications. This platform is composed by a visual programming application builder available on the web (IVO Builder) where users can define a set of context conditions (spatial and temporal) and workflows of activities, which are later triggered when the user is in the presence of those contexts. There is also a smartphone runtime layer (IVO Client) that loads the developed applications and provides a Workflow Engine to enable the coordination of the flow of activities.

Most of the systems stated above were developed when the research in context-aware computing started to become more important. Even though they provided important contributions in empowering end-users to create context-aware applications, all the systems (except IVO) are outdated and don't support the newest mobile OS. In addition, the majority have a big focus in location-based scenarios which narrows the domains where such applications can be used. Another important difference when compared with our work is that it seems the applications created with those systems are just a set of context scenarios (without any relationship) which triggers actions when facing those contexts. With our work, we want to create an authoring environment where users can create and customize the screens of the application and create transitions between them. In addition, the context information collected will be used to either trigger those transitions or automatically start additional interactions with the user (e.g. automatically open the application and prompt the user to fill a form).

As far as we are concerned, there are few authoring tools that enable end-users without programming expertise to create mobile applications that: a) support sensor based interactions; b) use context-information to adapt application's behavior based on different scenarios. Even though sensors are becoming important in interaction design, specifying the relationship between sensor values and application logic still remains an exercise of trial-and-error [41]. Hartmann et al. [41] introduces techniques for authoring sensor-based interactions by demonstration. A combination of direct manipulation and pattern recognition techniques enables designers to control how demonstrated examples are gen-

eralized to create interaction rules. In our work, we aim at providing a set of sensor events that can be easily used by non-expert users. Even though we try to use a higher level of abstraction, some events may require additional configuration (e.g. define a range of values in which the event may occur).

2.3.2 DETACH

DETACH⁵ [42, 43] is an authoring tool which was created to target therapists that need to design and deploy mobile applications for Cognitive Behavioral Therapy (CBT) procedures. This type of therapy is usually divided in two different kinds of sessions: there are sessions where the therapist and the patient are both inside an office; and sessions where the patient needs to follow some homework tasks. In this domain, the ability for those homework tasks to adapt to the patient's evolution is crucial to ensure the success of the treatment. The use of technology solutions proved to improve patient engagement as well as improving registry organization, since paper artefacts can be replaced with digital ones. However, the diversity of pathologies and the fast change of application's requirements undermines the common approach of creating dedicated solutions to each patient. To address these limitations, DETACH allows therapists without programming experience to quickly create applications tailored to each patient's needs.

The development of DETACH started with a series of participatory design and thinking aloud trials with non-programmer users aiming to understand how they conceptualized programming. The results of interacting with low and high fidelity prototypes provided a set of interaction patterns and behaviors used to design the final DETACH tool. Final evaluations showed that the tool fulfilled all the initial requirements by allowing health professionals, and people with no programming experience, to create applications and run them in a mobile runtime environment.

Even though DETACH was developed to provide solutions in the health domain, there are other domains where DETACH's applications can be useful. In this work we are going to extend DETACH's capabilities by integrating sensors in the environment. A detailed explanation about DETACH is given in Chapter 3.

2.4 Summary

In this chapter we started by presenting the importance of context in ubiquitous applications that are developed nowadays. With the evolution of smartphones and the availability of sensors (either embedded or external) obtaining context information about user's ac-

⁵DEsign Tool for smartphone Application Composition

tivity or user's surrounding environments becomes easier. Even though numerous context-aware applications have been developed, most of them are dedicated applications with a well-defined purpose and don't provide any expandability (in terms of adding new sensors to the application) or customization to fit several scenarios. There are domains where applications' requirements are often changing and developing a context-aware application from scratch is expensive and time consuming. In addition, creating such applications involve working with several sensors from different manufacturers and with different hardware specifications which requires some expertise. Another problem is the fact that end-users have a deeper knowledge of their activities and surrounding environments than everyone else; however, they may fail to translate this knowledge to applications requirements.

In order to overcome the problems stated above, there is a need to develop a visual prototyping environment that enables end-users to build context-aware applications without writing any code. The goal of this environment is to empower end-users and let them have an active role in application's composition. The literature presented in this chapter shown us that existing visual programming environments produced positive results when used by non-expert users. This motivate us to create an environment where users can use sensor's context information to create reactive and proactive applications. This environment also provides two benefits: expand the design space of context-aware application design to a larger group of users and give end-users the ability to control what should happen in those applications. Some challenges to create an ecosystem that allows the development and execution of applications with sensors are:

- Providing an easy and flexible way to let users customize application's behavior by using events provided by sensors.
- Collecting context informations from sensors and transform it in higher-level events that can be easily understood by non-expert users.
- Creating a rule engine that sits between the visual authoring environment and the mobile runtime environment. Most context-aware applications can be described as a collection of rule-based conditions where `if` some condition is matched `then` some behavior is triggered in the application.
- Creating a versatile runtime environment that works with all kinds of sensors, in order to match the requirements of different domains.

Chapter 3

DETACH Analysis

In this chapter we present a global overview of the DETACH system. This overview provides the necessary background about the current state of DETACH, presenting some important concepts which allows the reader to understand how the system works.

In the end, we discuss several constraints related to the integration of sensors in the system. Overcoming these constraints by providing the required mechanisms to compose sensor-based applications and collect context-data from sensors are our goals in later chapters.

3.1 System architecture

This system comprises three different components:

- a web tool to create mobile applications, denominated DETACH.
- a mobile runtime environment capable of running the created applications called DETACH Mobile.
- a database in which applications and user data is stored.

DETACH and DETACH Mobile use a thin client/fat server architecture. This architecture is shown in Figure 1.

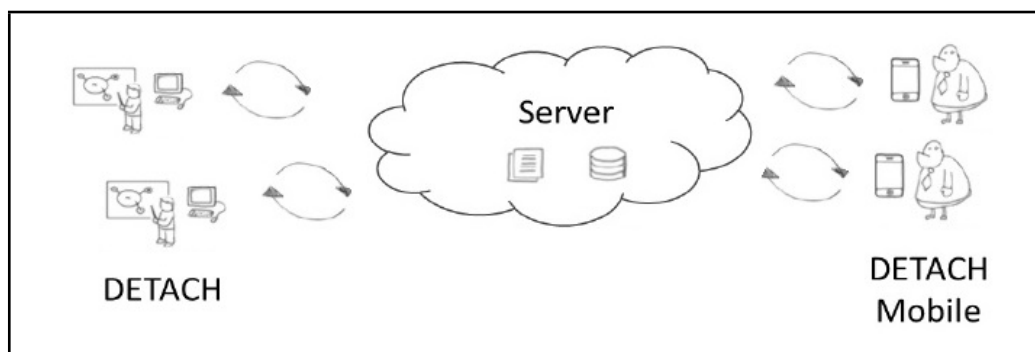


Figure 1: DETACH's system architecture

This architecture covers three types of stakeholders:

- **Non-Expert Programmers:** should be able to easily create mobile applications using the available screen templates, personalizing their contents and behavior as they want in order to satisfy their needs.
- **IT Professionals:** responsible for the creation and maintenance of DETACH's components.
- **Mobile Application End-Users:** should be able to use the mobile application that was assigned to them.

3.2 DETACH Authoring Tool

Having in mind that DETACH is aimed at people without programming experience, the tool was developed as a web application to ensure that no installations or configurations were necessary prior to using the tool to create applications. This choice also ensures that all users working on different operating systems and different devices can use the authoring tool without any constraint.

DETACH's interface was built with HTML5, CSS3 and Javascript. DETACH server also uses PHP and MySQL to handle some functionalities such as user and application management.

3.2.1 Applications' anatomy

Like any mobile application, each DETACH application is comprised by a set of screens. To ease and guide the authoring process, users have at their disposal a set of template screens, each one with different content elements and a purpose (e.g. display a

message, display images, ask a question). The content elements of each screen are customizable by users; however, users can't add additional content elements to the screen. Screen templates are developed and maintained by developers. To control the behavior of an application, users must specify transitions between screens. Transitions may depend on simple navigation elements (e.g. navigation buttons) or content elements (e.g. radio buttons) and are represented as arrows. The anatomy of a DETACH application is summarized in Figure 2.

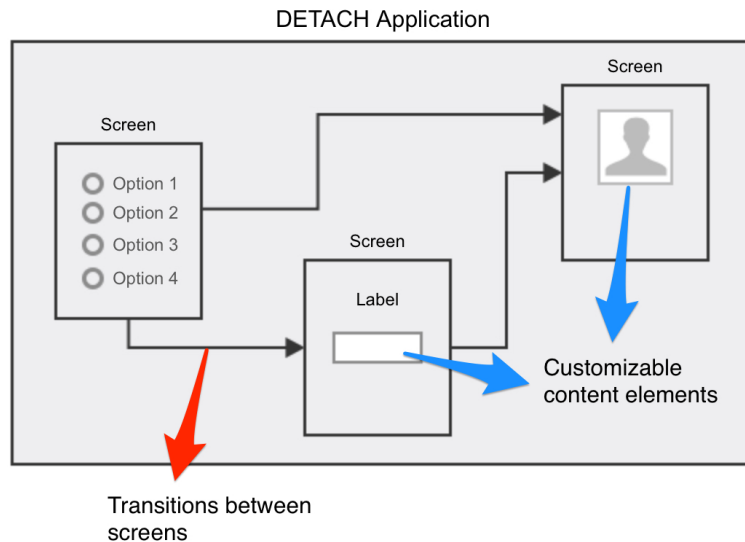


Figure 2: Anatomy of a DETACH application

3.2.2 Interface

The interface of the web application (Figure 3) is divided in three distinct areas:

- A top section containing the available screen templates which can be used to compose applications (red layer in Figure 3).
- A central canvas to which screen templates can be dragged and organized (green layer).
- A configuration panel on the right side that enables the customization of the selected screen(s) in the canvas (blue layer).

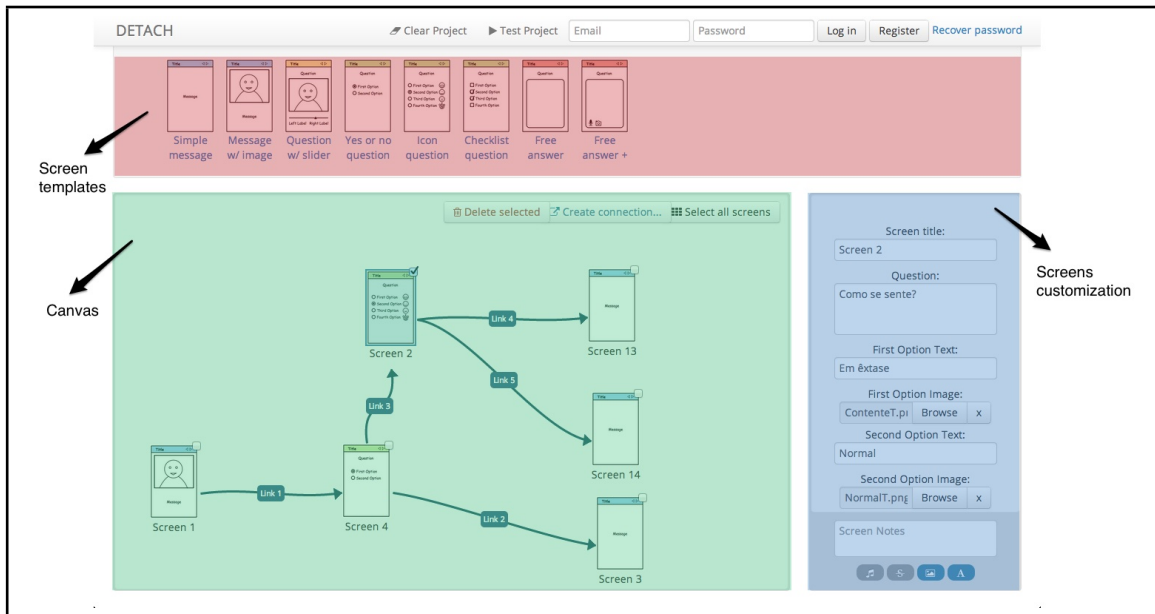


Figure 3: DETACH interface

3.2.2.1 Screen templates

To create applications, users can choose from a set of available screen templates (Figure 4):

- Screens containing only messages (blue color).
- Screens containing dynamic content that changes according to user answers (yellow color).
- Screens containing questions with possible choices (green color).
- Screens containing free answer questions (red color).

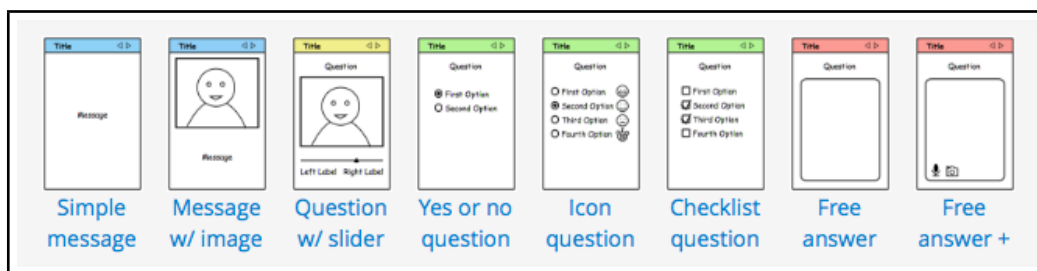


Figure 4: DETACH mobile screen templates

When a screen template is clicked or dragged into the canvas, that screen is added to the current application. Each screen template contains different elements that can be

customized, using the configuration panel located on the right side of the interface. Common to all screens are the screen title, screen notes (small appointments, only visible to the user which is creating the application) and screen add-ons (audio, subliminal content, background and text customization). Each screen template may have other fields that can be personalized by the user.

3.2.2.2 Screen triggers

In addition to dragging screens to the canvas, users also need to specify transitions between screens (represented in the canvas as blue arrows, linking one screen to another). Even though each screen template offers different triggers, we can group them in two categories:

- **Triggers based on navigation elements:** These triggers allow actions based on navigation elements, such as clicking the "next button" available in all screen templates. They are particularly used with screen templates that only display static content (such as text and images) that don't provide additional user interactions.
- **Triggers based on screen content:** Some screen templates have content elements (e.g. radio buttons, check boxes, sliders) that provide additional user interactions, and therefore give users the possibility to control the transitions based on the current state of the contents (e.g. if radio button 1 is selected goes to screen 2, otherwise goes to screen 3).

Transition can be customized through a connection definition panel that appears after a connection is created (Figure 5).

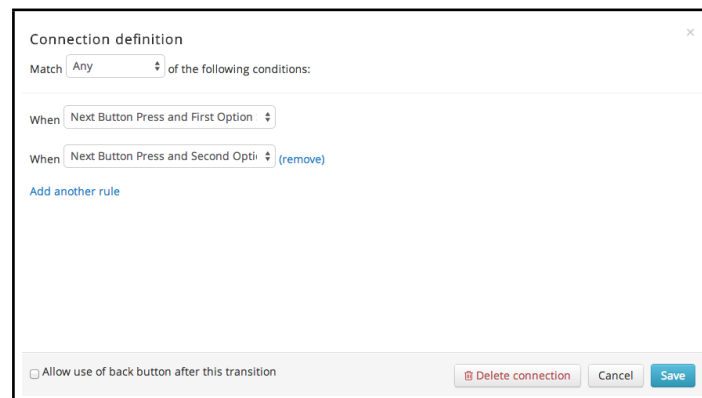


Figure 5: Connection rules specification

3.2.2.3 Runtime Emulator

DETACH also provides a runtime emulator that let users quickly preview how a mobile application created with DETACH would look in a smartphone. This emulator allows a person to test the application's interface, content, sound, styling and behavior without the hassle of deploying the application to a smartphone, which is very handy when prototyping new applications.

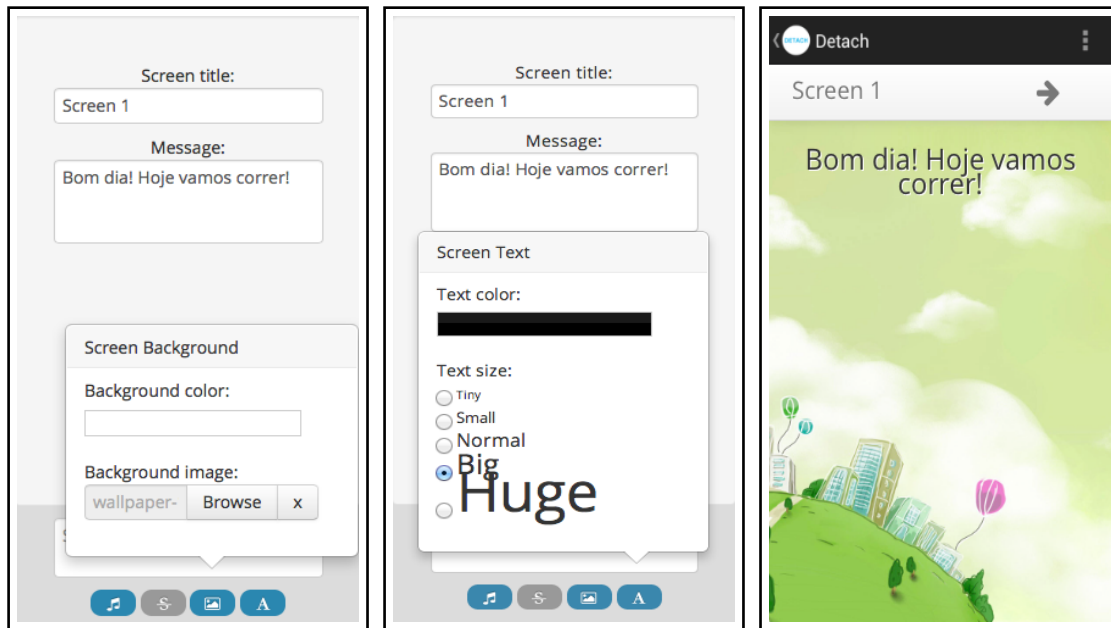


Figure 6: Example screen configuration and respective run-time result

3.3 DETACH Mobile

DETACH Mobile is a runtime environment, built on top of Android OS, capable of running applications created with DETACH. It uses Java with an Android WebView that handles HTML5, CSS3 and Javascript content the same way as DETACH Runtime Emulator. This solution requires an active internet connection to authenticate users and load application's contents from server in real-time.

Users can navigate through the application screens (Figure 7) back and forward and this information is appended to a log file which can be viewed later by the application's author.



Figure 7: Application running in DETACH Mobile

3.4 Constraints in sensor integration

In order to better perceive most of the requirements of our work, first we must understand current DETACH limitations regarding the integration of sensors in the system. Including sensor-based interactions requires improvements and modifications in both DETACH's environments: authoring and mobile runtime. In this section we individually discuss the constraints of each environment, which are addressed in later chapters.

3.4.1 Runtime environment

Context awareness is widely used in ubiquitous computing in order to perceive user's activity or surrounding environment, and react accordingly. Being 'context aware' means that raw data from sensors have to be analyzed and converted into a description that makes sense for both applications and users. Our goal in this work goes beyond collecting context data from users; we want to take advantage of such information to create more dynamic and proactive applications.

Currently DETACH Mobile, the runtime environment, merely contains a way to run DETACH applications. Until now, the capabilities of this environment were enough to address the existing requirements; however, integrating sensors to collect context data and control the behavior of an application arises several problems:

- (a) How does the system communicates with sensors in order to use the data collected?

- (b) How does the system transform data into events?
- (c) How does the system consume fired events and trigger specific actions within an application?

Besides these problems, we must also take into consideration that DETACH follows a principle of modularity, where developers are important stakeholders that are responsible to add new components (screen templates and sensors) to the system. In order to achieve this, the runtime must provide a high abstraction level and support a way to easily work with several types of sensors. Our objective is to create a unified programming interface that supports multiple sensors and remove barriers that may exist when using different types of sensors (e.g. data formats, programming languages).

As we presented in Chapter 2, event-based systems are well suited to tackle these type of requirements. They support applications that are reactive, meaning that we can use processed data to take actions in response to specific events. With this approach, we may have several producers of events that must be able to:

- Communicate with sensors and collect data from them.
- Apply some processing to the collected data (e.g. process raw data from an ECG sensor to calculate user's heart rate). It's important to convert meaningless data into something that can be easily understood by non-experts users.
- Trigger events: each sensor has a set of rules that define the system events. This will allow the system to react differently when facing different contextual information.
- Support the configuration of the sensors (e.g. setting the sensors' operation threshold values).

In addition to producers, it is also required to have a consumer of events. This component must be able to:

- Understand events triggered by sensors. These events will be associated with a specific behavior in a DETACH application (e.g. change screen when user's heart rate is above 80 bpm).
- Support the aggregation of events from different sensors (e.g. heart rate value and user location at the same time). DETACH supports transitions with multiple conditions, and therefore we must ensure that asynchronous events are grouped and correctly processed.
- Take actions in response to events.

3.4.2 Authoring environment

In order to integrate sensors in the authoring environment, we must take into consideration two different perspectives:

1. On the one hand, developers need to create representative sensor elements which can be used by non-expert users when composing applications.
2. On the other hand, the process of composing applications requires some modifications to introduce new sensor elements and allow the configuration of screen transitions based on sensor triggers.

At this point DETACH lacks the support of sensor elements, and therefore we must include them in the authoring environment. The specification of those elements are responsibility of developers. Most importantly, developers must specify a list of triggers that are provided by each sensor. An important note is that sensors are not available in the authoring environment and therefore, this requires particular attention when it comes to application testing, since the current runtime emulator is not prepared to support applications that use context data.

From the point of view of non-expert users, they must be able to easily compose applications with sensors. A key difference to the current tool is the possibility to use sensors to control the behavior of an application. While screens are used to define the interface of an application, sensors will be used to control the transitions between screens. The challenge here is to provide an easy method to specify transitions which rely on sensors while avoiding the disruption of the current application composition procedures. At this point, the runtime emulator does not successfully support sensor-based applications. To do so, this emulator must be updated in order to support the input of fake sensor data. Otherwise, triggers that require context data won't work.

In summary, the current authoring environment must be improved in order to:

- Support the addition of sensor elements in the tool's interface.
- Use sensor elements to specify sensor-based triggers that are used to control transitions between application's screens.
- Use the runtime emulator to preview and test the behavior of applications, including a way to input sensor data and simulate sensor-based transitions.

3.5 Summary

In this chapter we reviewed DETACH - an existing authoring environment that enables users without programming experience to design and deploy mobile applications. It comprises three components: i) a server with a database in which applications and user data is stored; ii) a web tool to author applications denominated DETACH; iii) an Android runtime environment capable of interpreting the created applications called DETACH Mobile.

In DETACH's web tool, non-expert programmers can compose mobile applications by selecting screen templates and configure its contents. They can also give the application some type of behaviour, by creating transitions from each screen to another based on different triggers. This tool also features an emulator that let users quickly preview how a application would look in a smartphone.

Since our work focuses in extending the functionalities of DETACH to enable the development of sensor-based applications, we analyzed the current state of both environments (authoring and runtime) in order to perceive their limitations regarding the integration of sensors. Those limitations provided us with a set of requirements and problems to tackle in the following chapters.

Chapter 4

Adding Sensors

This chapter covers the design of an Android runtime environment which is used to execute the applications created with DETACH. This work focuses on providing the necessary mechanisms to collect data from sensors.

A key point for the proper functioning of DETACH is related to the fact that this tool is the result of joint efforts between the various stakeholders. IT experts have the necessary skills and expertise to create and maintain content that supports the requirements imposed by the domain specialists, and therefore we must ensure that in the future they can also add new sensors to the system. In the last sections we describe, from the developers' point of view, how to add new sensors in the system. First, we explain how to code and add a sensor to the mobile runtime environment, in order to collect data and control events associated with the sensor. Then we focus in explaining the necessary steps to add a sensor in the DETACH authoring environment which enables the composition of sensor-based applications.

4.1 Architecture

In order to properly integrate sensors in DETACH, several changes in the architecture were required. As pointed out in Chapter 3, the current architecture of the authoring tool is enough to support the addition of sensors. Most changes in the authoring tool are related to the process of application composition and will be described in Chapter 5. However, the runtime environment, which is the focus in this section, lacks the mechanisms to collect data, and use it to control the behavior of the applications.

To support the requirements presented in the previous chapter, we propose an updated architecture to the runtime environment (Figure 8). This architecture is now comprised by:

- **Application View:** used to render mobile application's contents. This component already existed in the previous architecture.
- **Event Manager:** used to interpret and coordinate events triggered by sensors and consequently, trigger behaviors within a DETACH application.
- **Sensor Processing Units:** used to collect and interpret sensor data. Also used to trigger events within the system.

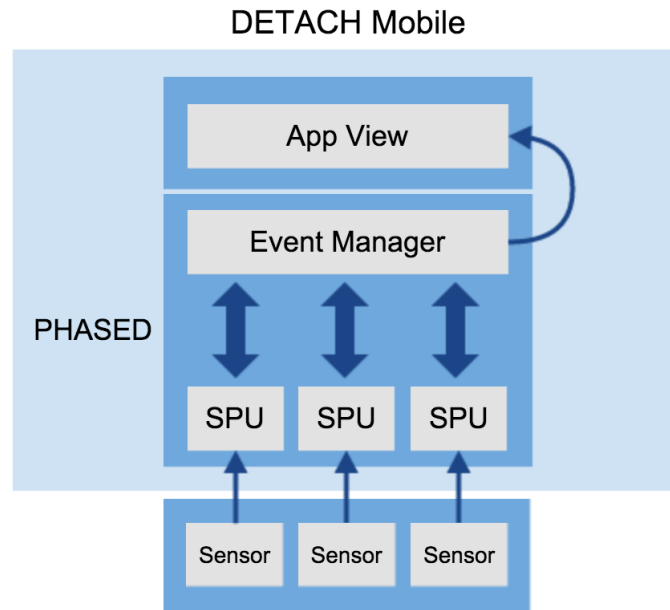


Figure 8: Runtime environment updated architecture

As shown in the previous figure, the Event Manager and Sensor Processing Units are part of a sub-system called PHASED. This sub-system will run in smartphones along with DETACH Mobile, providing a bridge between sensor devices and the Application View. In this architecture we ensure that each component has a well-defined function targeting a specific concern. This approach provides modularity and separation of concerns, facilitating the maintenance and addition of components. Even if the Application View is modified, these changes won't have impact on the system operation since the Event Manager just knows which screen should show or hide at a given time. The presentation layer is responsibility of the Application View. The Sensor Processing Units encapsulate the logic applied to the data collected via the sensors. The operation of these units is independent and the events triggered must contain the relevant information needed by the Event Manager in order for them to be correctly interpreted. The communication with external or internal sensors is also independent from sensor to sensor, thereby ensuring that different types of sensors can be used.

With this approach, we ensure that PHASED is capable of:

- (a) collecting and interpreting sensor data.
- (b) using sensor data to trigger certain behaviours within applications created with DETACH.

The proposed architecture uses an event-driven approach. This approach works around the production, detection and consumption of events and allows the transmission of events among loosely coupled components. In our system, the Event Manager is responsible to detect and consume events that are produced by Sensor Processing Units. PHASED's architecture is shown in Figure 9.

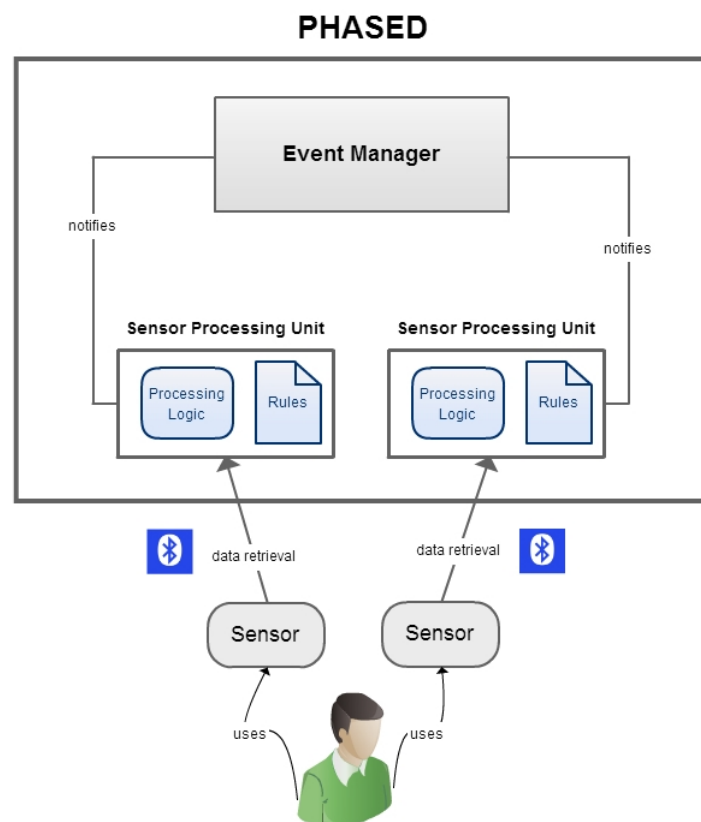


Figure 9: PHASED architecture

Event Manager

The Event Manager is the system's primary engine and it plays an important role in our system because it is the only component that consumes events. This component ensures that all events are processed in correct order and it also provides the ability to aggregate asynchronous events that are fired by different sensors. Consequently, it allows users to specify transitions between screens that rely on different sensors.

This component is executed when a user launches DETACH Mobile in his/her smart-phone. It's responsible for:

- parsing an input XML file that contains information about all the events of a DETACH application. When users compose applications whose screen's behavior depend on sensors (covered with more detail in Section 5.5), a XML file with all the events of the application is generated. This file also contains information about the actions to take when specific events occur. Each event is defined as a triplet $\langle identifier, source, target \rangle$ where the *identifier* is unique and *source* and *target* are used by the Event Manager to perform the action associated with the event.
- detecting and consuming events fired by Sensor Processing Units. It maintains an internal state table of all events and actions to take when events are fired (Table 1). This table is created with the information provided by the XML file.
- performing actions when necessary. In this case, the action is the activation of a specific screen of a DETACH application when an event (or a combination of events) occur.

Event(s)	Source screen	Screen to activate
[1:3:0] and [1:3:1]	1	3
[2:3:0]	2	3

Table 1: Map between events and actions

When several events share the same source and target screen, it means that there is a transition between two screens that encompasses several conditions (e.g. heart rate above 120bpm and user is at home). Each time an event is triggered by a Sensor Processing Unit, the Event Manager receives a notification with the *identifier* of the event. Then, it updates the state table and checks if the event received must trigger an action. For instance, if the Event Manager received a notification with the *identifier* [1:3:0] nothing would happen until the Event Manager also receive an event with the *identifier* [1:3:1] (Table 1).

The Event Manager is particularly important to combine events from different sensors and prioritize certain events, if necessary. In addition to maintaining a state table, the Event Manager is also responsible to communicate with the Application View module, in order to let it know that a specific screen must be displayed to the user. The Event Manager acts as a bridge between a low level layer (Sensors) and a higher layer (Application View) thus ensuring a better integration between both layers.

Sensor Processing Units

Each Sensor Processing Unit is comprised by a Processing Logic module and a XML file containing a list of events that should be controlled by a sensor. They are responsible for performing the required communication with a sensor, whether the sensor belongs to the smartphone or the sensor is attached via Bluetooth. After the communication is established, this component must collect data from the sensor and process it. This is done by the Processing Logic module. Each Processing Logic module must transform raw data into data types that can be used to evaluate specific actions offered by a sensor. When a specific action is detected, the Sensor Processing Unit triggers an event that will be received by the Event Manager. To evaluate these actions, each Processing Logic module contains one or more drivers (each driver controls a specific action). Drivers are functions that use context data provided by the sensor to evaluate if any event must be triggered.

All Sensor Processing Units share the same workflow during their execution. They are always retrieving data, processing it and finally they evaluate conditions to check if any event should be fired, notifying the Event Manager when necessary. This workflow is shown in Figure 10.

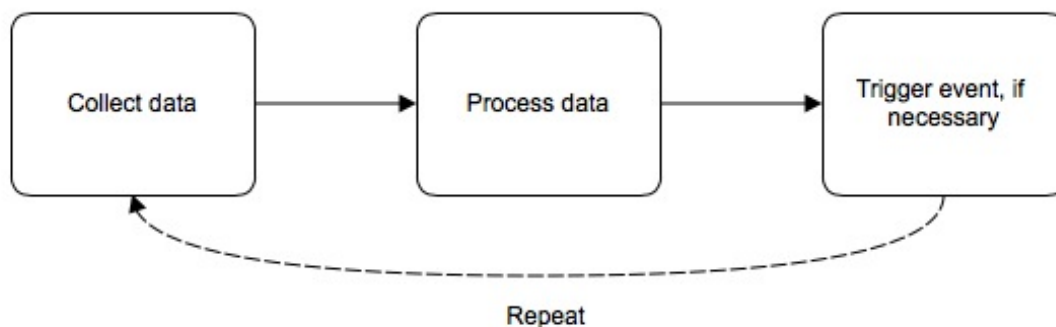


Figure 10: Sensor's workflow

As mentioned before, developers play a important role in DETACH since they are responsible for creating new components requested by the users who have the role of composing applications. Having this in mind, each Sensor Processing Unit is an independent module and is deployed as a standalone APK¹. This way, users can individually install and remove Sensor Processing Units in the same way they do with common Android applications, and future programmers can add new sensors without the need to deploy a new version of the whole system. When an APK is installed in the smartphone, it is

¹Android Application Package File

automatically recognized by the runtime environment and can be activated/deactivated by end-users through DETACH Mobile interface. This procedure is shown in Figure 11.

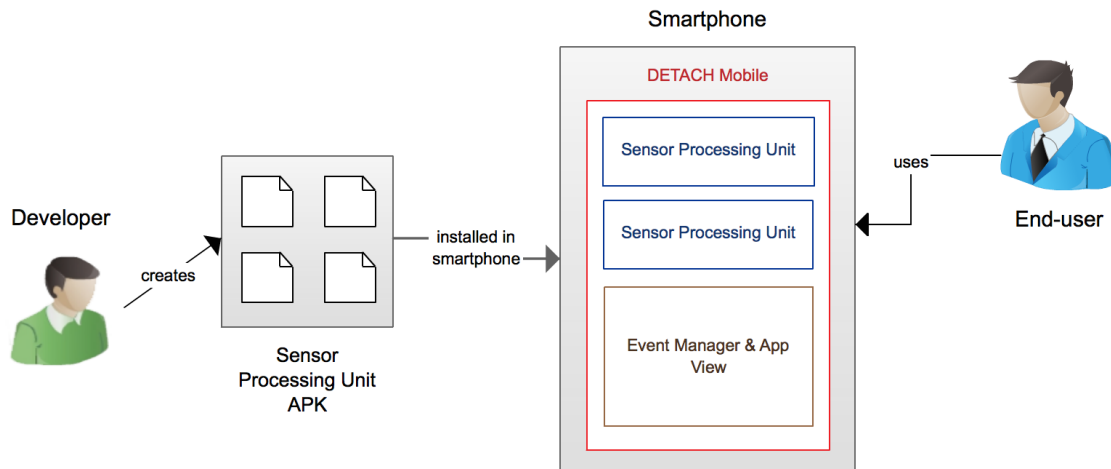


Figure 11: Development and installation process of a new Sensor Processing Unit, and related stakeholders

4.2 DETACH Mobile

DETACH Mobile is an Android application whose purpose is to run applications created with DETACH, and it is the implementation of the architecture explained before. It uses an Android WebView that handles HTML5, CSS3 and JavaScript to display the application's screens. Internet connection is not required except to synchronize the necessary application files from the server.

When a user starts DETACH Mobile, a status screen is shown (Figure 12 - left). This screen displays the name of the last project downloaded and a list with each sensor's state (either running or not running). Again, all the Sensor Processing Units previously installed in the smartphone will be automatically recognized by the runtime environment, and are available to be used to collect context information. From this screen, users can also intentionally start a DETACH application by using the button "Launch application". Despite providing a way to manually start an application, as will be explained in more detail in Chapter 5, the integration of sensors in the applications also allows users to specify that an application automatically start when facing a specific context (e.g alert the user to take his medication at 10am).

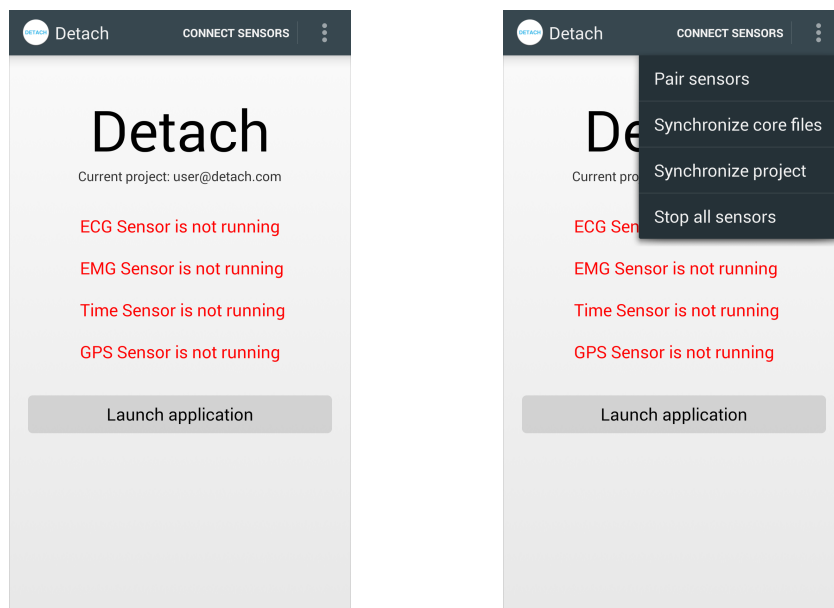


Figure 12: DETACH Mobile status screen (left), DETACH Mobile menu options (right)

Accessing the menu (Figure 12 - right), users can also:

- **Connect sensors:** used to start sensors. When DETACH Mobile is launched it automatically detects all the sensors installed in the smartphone (Figure 13 - left) and include them in the list presented to the user when this menu item is pressed. When a sensor is started, DETACH Mobile invokes the service corresponding to the sensor selected. If the sensor is successfully connected, the label indicating the sensor status in the main screen is updated (Figure 13 - right).
- **Pair sensors:** List all the Bluetooth devices that are paired with the smartphone. It allows the user to associate an external device with a specific sensor (e.g. select the *mac address* of the EMG sensor).
- **Stop all sensors:** Stop the execution of all sensors.
- **Synchronize core files:** Downloads *core* files required to run a DETACH application. Without these files, users can't run applications. An active Internet connection is required to download the files from the server.

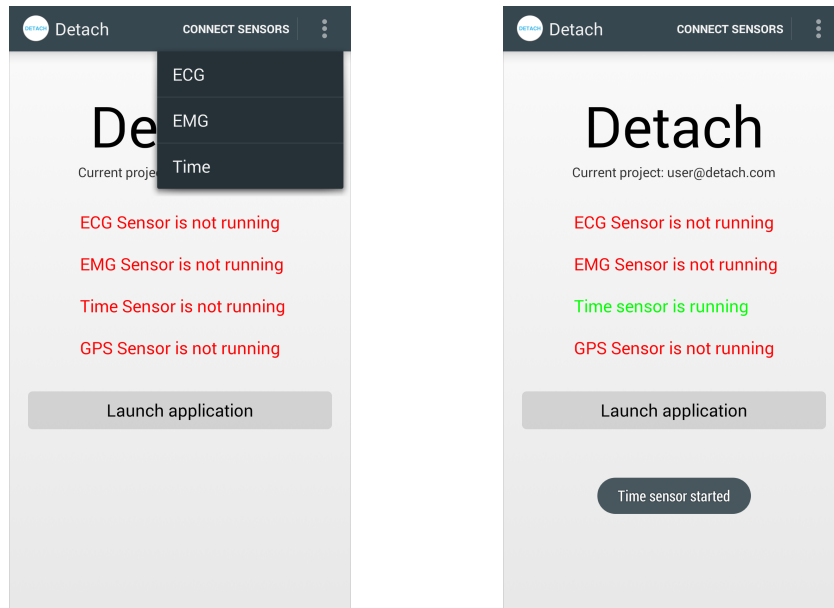


Figure 13: List of available sensors (left), Sensor status updated (right)

- **Synchronize project:** Download an application that was previously assigned to a user. When the user select this option, an authentication screen is shown (Figure 14). If the input credentials are valid, an application previously assigned to the user is downloaded. This action requires an active Internet connection.

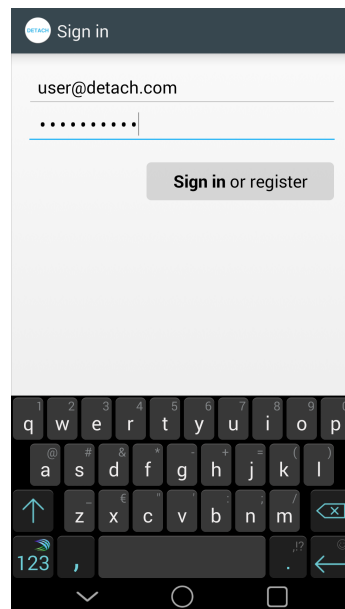


Figure 14: DETACH Mobile authentication screen

After downloading the application, the internet connection is no longer necessary. From this point, a user can run the downloaded application. If this application requires

sensors, these must first be enabled to allow the application to work correctly.

4.3 Adding sensors to the runtime environment

The Event Manager and all Sensor Processing Units are implemented as Android Services. An Android Service² is an application component that allows long-running operations in the background (even if the user switches to another application) and does not provide any user interface. Services can also be started by another application which is very useful and allow us to start and stop sensors from DETACH Mobile interface. Since the components need to communicate with each other, a Messenger³ component was used which allows the implementation of message-based communication between our services.

In order to add new sensors to the runtime environment, developers must code the necessary mechanisms to retrieve data from the sensor and code the necessary drivers to control the events featured by a sensor. To ease the development of new sensors, developers must use a library that contains a set of base classes that already offer some utilities (e.g. parse XML files, communication with the Event Manager). Figure 15 shows the UML Class Diagram of this library.

The most generic class is called `SensorBase` and inherits from the class `Service` provided by Android OS (Figure 15). This class is responsible for:

- initializing generic data structures and register a sensor within the system (which allows the communication between the Sensor Processing Unit and the Event Manager).
- parsing the XML file which contains information about the events related with the sensor, after the initialization.
- notifying the Event Manager when an event is fired.
- unregistering the sensor when a sensor is stopped.

Since external sensors require a way to connect and communicate with a smartphone, developers can extend and code custom sensor classes that inherit from the base class. Specifically, in our work we used Shimmer's physiological sensors (ECG and EMG). Shimmer⁴ is a provider of wearable sensor products and solutions. To properly operate with these sensors, we also developed the `SensorShimmer` class which inherits from the class `SensorBase`. `SensorShimmer` provides the utilities to establish a Bluetooth

²<http://developer.android.com/guide/components/services.html>

³<http://developer.android.com/reference/android/os/Messenger.html>

⁴<http://www.shimmersensing.com/>

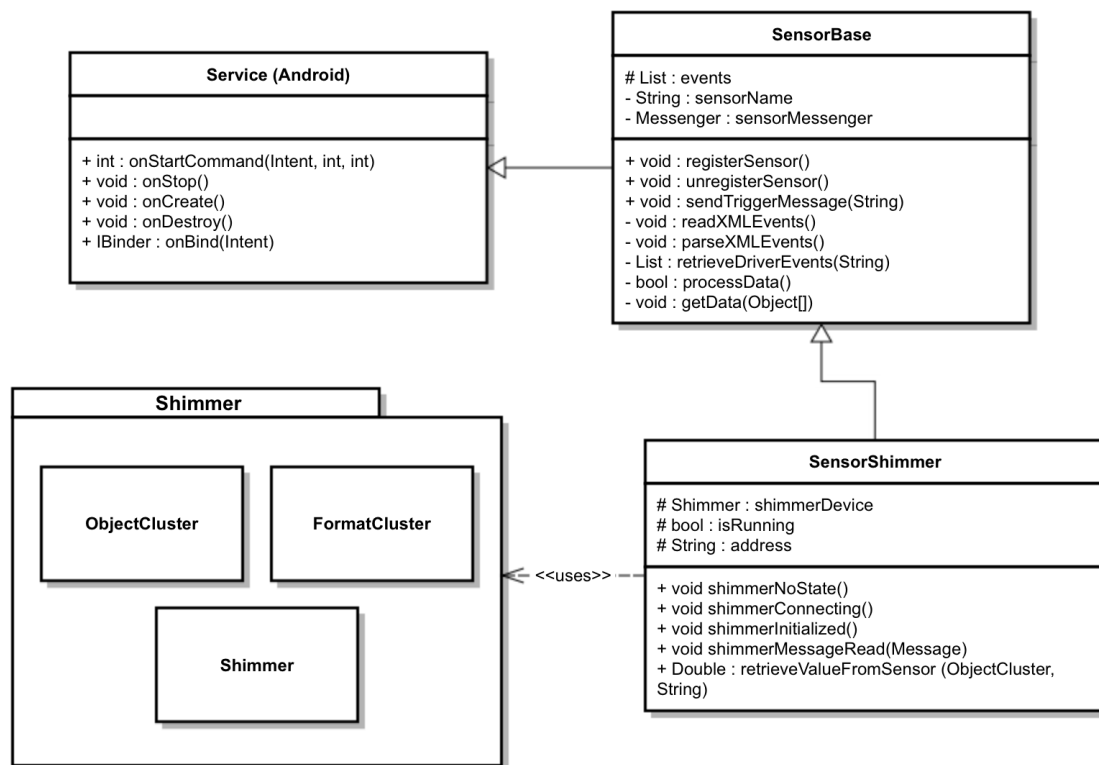


Figure 15: UML Class diagram

connection between the sensor and the smartphone and collect data from it. To perform the connection and collect data, Shimmer's manufacturers provide a library which offers a set of utilities to interact with a Shimmer device.

4.3.1 Adding a new sensor

First, developers need to create a new Android project with an Android Service and import the custom library explained before. The project package name must be `com.phased.SensorName` (e.g. `com.phased.ECG`) because DETACH Mobile automatically loads all the sensors that share this package name.

Before coding the Sensor's Service, developers must setup the Android Manifest file. First, it's required to include a schema that allows exportable services. Afterwards, developers may need to list some permissions used by the sensor (e.g. accessing Bluetooth, accessing GPS location) and finally, they must export the Service (which allows DETACH Mobile to recognize it). More details about how to setup the Android Manifest can be found in the developer's guide annexed to this document. Listing 4.1 shows an example of a full Android Manifest file.


```
1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3   xmlns:tools="http://schemas.android.com/tools"
4   package="com.phased.ecg"
5   android:versionCode="1"
6   android:versionName="1.0" >
7
8   <uses-sdk
9     android:minSdkVersion="8"
10    android:targetSdkVersion="18" />
11
12   <uses-permission android:name="android.permission.INTERNET"/>
13   <uses-permission
14     android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
15   <uses-permission
16     android:name="android.permission.BLUETOOTH_ADMIN"/>
17   <uses-permission android:name="android.permission.BLUETOOTH"/>
18   <uses-permission
19     android:name="android.permission.ACCESS_FINE_LOCATION"/>
20   <uses-permission
21     android:name="android.permission.ACCESS_COARSE_LOCATION"/>
22   <uses-permission
23     android:name="android.permission.ACCESS_NETWORK_STATE"/>
24
25   <application
26     android:allowBackup="true"
27     android:icon="@drawable/ic_launcher"
28     android:label="ECG">
29     <activity
30       android:name="com.phased.mysensor.StartActivity"
31       android:label="ECG">
32       <intent-filter>
33         <action android:name="android.intent.action.MAIN" />
34         <category android:name="android.intent.category.LAUNCHER" />
35       </intent-filter>
36     </activity>
37     <service
38       android:name="com.phased.ecg.ECG"
39       android:exported="true"
40       android:label="ECG"
41       tools:ignore="ExportedService">
42     </service>
43   </application>
```

40 </manifest>

Listing 4.1: Android Manifest XML file example

Moving to the Android Service programming, it must extend the class `SensorBase` or the class `SensorShimmer` (if it's a Shimmer device) provided in the custom library. A skeleton of the service is shown in Listing 4.2.

```

1 public class MySensor extends SensorBase {
2
3     @Override
4     public int onStartCommand(Intent intent, int flags, int startId) {
5         super.onStartCommand(intent, flags, startId);
6         // TODO: more initializations here, if necessary
7     }
8
9     @Override
10    public void getData(Object[] objs) {
11        // processData is a boolean function which allows the
12        // programmer to check if data can be processed.
13        if (processData()) {
14
15            // iterate each rule associated with this sensor
16            // the parameter of retrieveDriverRules() should be equal
17            // to the value(s) in the tag <driver>
18            // specified in the XML
19            for (Rule rule : retrieveDriverRules("driver-name")) {
20                // TODO: evaluate rules and trigger events
21            }
22    }
23 }

```

Listing 4.2: Sensor Service skeleton

onStartCommand() function

This function is called every time DETACH Mobile starts the service. It's responsible to perform initializations (e.g. configure some sensor parameters, connect the sensor). If the connection fails, the running service is destroyed.

getData() function

Each time the sensor receives new data, this function should be executed. The argument `objs` provides an array of objects (giving the developer the flexibility to define the type of data passed in the argument) with the data collected.

processData() function

It's a boolean function which allows the programmer to check if data can be processed. By default it always returns true but it can be overridden, for example to check if it makes sense to evaluate data (e.g. is there any data yet? is the user doing any activity?).

sendTriggerMessage() function

Fires an event, notifying the Event Manager.

Developers also need to implement a method responsible for retrieving data from the sensor and process it if necessary (e.g. transform raw data into data that has some meaning and can be used to evaluate event's conditions). This function isn't included in Listing 4.2 because it varies from sensor to sensor.

Lastly, the developer must program the drivers that use the data collected and fire events when necessary. Those drivers are responsible for evaluating the set of events associated with each sensor and are executed each time the sensor collects new data. Listing 4.3 shows the implementation of the Location service.

```
1 public class GPS extends SensorBase {
2
3     private static final long MINIMUM_DISTANCE_CHANGE_FOR_UPDATES = 10;
4         // in Meters
5     private static final long MINIMUM_TIME_BETWEEN_UPDATES = 1000; //
6         in Milliseconds
7
8     private LocationManager locationManager;
9
10    @Override
11    public int onStartCommand(Intent intent, int flags, int startId) {
12        super.onStartCommand(intent, flags, startId);
13        // Create a handler to access user's location
14        locationManager = (LocationManager) getSystemService(Context.
15            LOCATION_SERVICE);
16        locationManager.requestLocationUpdates(
17            locationManager.GPS_PROVIDER,
18            MINIMUM_TIME_BETWEEN_UPDATES,
19            MINIMUM_DISTANCE_CHANGE_FOR_UPDATES,
20            new MyLocationListener()
21        );
22        return START_STICKY;
23    }
24
25    @Override
26    public void getData(Object[] obj) {
```

```

25     if (processData()) {
26         for (Rule rule : retrieveDriverRules("user-location")) {
27             Location event_location = new Location("");
28             String[] event_coords_split = rule.getCondition_values
29                 ().split("_");
30             event_location.setLatitude(Double.parseDouble(
31                 event_coords_split[0]));
32             event_location.setLongitude(Double.parseDouble(
33                 event_coords_split[1]));
34             Double radius = Double.parseDouble(event_coords_split
35                 [2]);
36
37             float distance = event_location.distanceTo((Location)
38                 obj[0]);
39             if (distance <= radius) {
40                 sendTriggerMessage(rule.getId());
41             }
42             else
43                 markAsFalse(rule.getId());
44         }
45     }
46
47     /**
48     * Handler that retrieves the user location from the GPS sensor.
49     * Each time the user position changes, the function getData() is
50     * called.
51     */
52     private class MyLocationListener implements LocationListener {
53         public void onLocationChanged(Location location) {
54             getData(new Object[] { location });
55         }
56
57         public void onStatusChanged(String s, int i, Bundle b) {
58             Toast.makeText(GPS.this, "GPS status changed",
59                 Toast.LENGTH_LONG).show();
60         }
61
62         public void onProviderDisabled(String s) {
63             Toast.makeText(GPS.this, "GPS turned off",
64                 Toast.LENGTH_LONG).show();
65         }
66
67         public void onProviderEnabled(String s) {
68             Toast.makeText(GPS.this, "GPS turned on",
69                 Toast.LENGTH_LONG).show();
70         }
71     }
72 }

```

Listing 4.3: Location service implementation

Once all the coding is finished, developers must deploy the sensor's APK in order to be downloaded and installed in Android devices. At this point, DETACH Mobile will automatically detect the new sensor and it can now be used by end-users in their applica-

tions.

4.3.2 Implemented Sensors

In the following section, we briefly describe the sensors available in our system.

Time

This sensor provides events based on time conditions. The events available are:

- **Elapsed time since an application started:** To control these events, we use a `Chronometer` that starts when a `DETACH` application is launched. This event let users define triggers based on the elapsed time since the application was started (e.g. display `Screen 3` when the elapsed time since the application started is 2 minutes).
- **Elapsed time on current screen:** Similarly to the previous type of events, we use a `Chronometer` that starts each time a new screen is displayed to the user.
- **Time of the day:** These events use the `Alarm Manager` offered by Android OS to create alarms at specific times of the day.

Location

This sensor is responsible for controlling the user's geolocation using the internal GPS sensor available in smartphones. It allows the specification of a set of activity zones to control if a user enters or leaves one of these zones. Each zone is defined as a pair of coordinates in the form of (*latitude*, *longitude*) and a *radius*. Events are triggered when a user goes inside or leaves one of the locations defined.

To obtain the user's position, we use a `Location Provider` which is a provider offered by Android OS that periodically reports on the device's geographical location.

EMG

The Shimmer EMG sensor (Figure 16) measures and records the electrical activity associated with muscle contractions and can be used to analyse and measure the movements' biomechanics.



Figure 16: Shimmer EMG sensor

This sensor is non-invasive (surface EMG) and therefore the activity it measures is a representation of the activity of the whole muscle or group of muscles whose electrical activity is detectable at the electrode site. The maximum signal ranges from ± 2.2 mV. A sample output is shown in Figure 17.

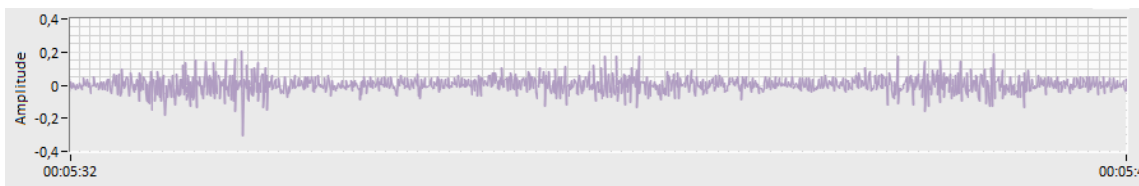


Figure 17: Sample output of Shimmer EMG

At this point, our EMG sensor detects two gestures:

1. **Muscle contraction:** to detect a contraction we establish an **activation threshold value** (e.g. values > 1.0). When the signal obtained goes above this threshold we consider that the muscle was contracted. The threshold value can be adjusted because the activity measured varies from muscle to muscle.
2. **Muscle retraction:** to detect a retraction, a contraction must happen before. Similarly, we establish a **reset threshold value** (e.g. values < 0.5) and when the signal goes below this threshold for a period of time, we consider that the user retracted his/her muscle. This threshold can also be adjusted.

EKG

The Shimmer EKG sensor (Figure 18) records the pathway of electrical impulses through the heart muscle.



Figure 18: Shimmer ECG sensor

To calculate the heart rate in beats per minute (BPM) from an electrocardiogram (Figure 20), we use the R waves (which are part of the QRS complex, shown in Figure 19).

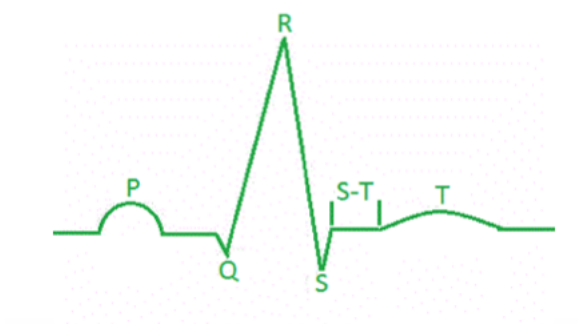


Figure 19: ECG waveform of a heart beat.

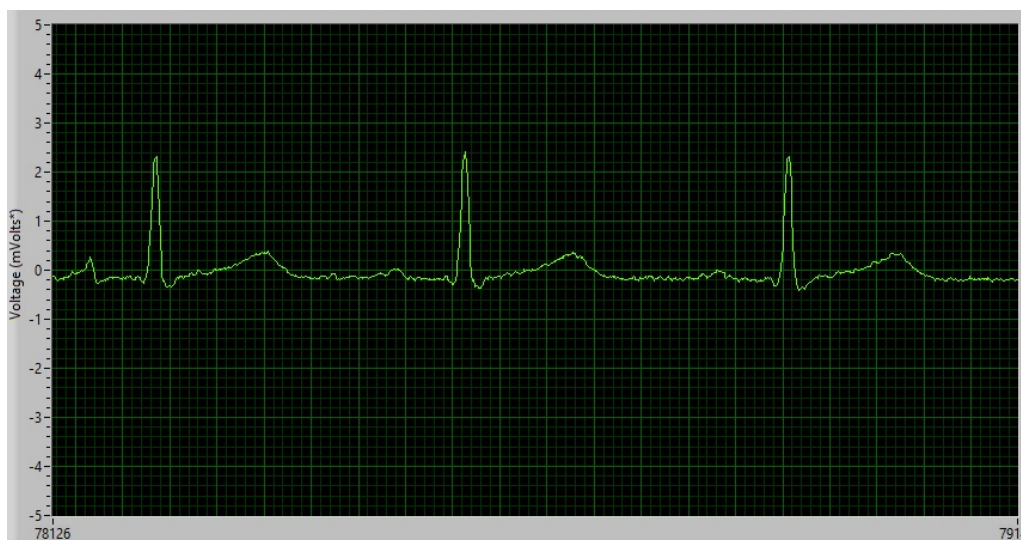


Figure 20: Sample electrocardiogram output of Shimmer ECG

The heart rate (HR) in BPM is given by the following formula:

$$HR(BPM) = 60 * (N_r - 1) / ((S_2 - S_1) / F_s)$$

where F_s denotes the sampling rate (in Hz) of the signal, S_1 denotes the sample number of the first R wave detected, S_2 denotes the sample number of the last R wave detected and N_r is the total number of R waves between S_1 and S_2 .

4.4 Adding sensors to the authoring environment

In this section we focus in explaining the required steps to add a new sensor in the authoring environment to enable the creation of applications that use them.

As stated in Chapter 3, users can compose applications from a set of screen templates. To integrate sensors in the composition of applications, we used a similar approach by giving users the possibility to use sensor templates to control the workflow of an application. The key difference is that sensors templates are used to control the behavior, while screen templates are used to customize the appearance. This will be covered with more detail in Chapter 5.

Adding a new sensor template in DETACH follows the same requirements as adding a new screen. Each template is composed by three files:

- (a) An image file.
- (b) A XML with the specification of the sensor details.
- (c) A JavaScript containing information about how DETACH runtime emulator can simulate sensor values.

All of these files should have the name of the new sensor and should be added in the *sensors* folder hosted in the server.

Image file

This file is used to represent the sensor in the sensors list and in the canvas. The file must have the `.png` format and for a better representation the image should have 65 x 65 pixels.

XML file

This file contains the details about the sensor, such as the name and available triggers. The format of the file is shown in Listing 4.4.


```

1 <sensor>
2   <type>EMG</type>
3   <description>
4     Uses an EMG sensor to control the screens behaviour.
5   </description>
6   <triggers>
7     <trigger id="1">Muscle contraction
8       <label>Contraction</label>
9       <sensor>
10        <name>emg</name>
11        <driver>contraction</driver>
12        <stream>no-data</stream>
13      </sensor>
14      <deactivation>
15        <type>time</type>
16        <data>0</data>
17      </deactivation>
18    </trigger>
19  </triggers>
20 </sensor>

```

Listing 4.4: Sensor XML file structure

<type tag>

Sensor type to be shown below the sensor image.

<description tag>

Description displayed when the user mouse hover a sensor template in the sensors list.

<label tag>

This tag is used to display a label in the connections between screens. Each connection has a label with the information of every trigger specified.

<sensor tag>

All the information contained inside the **sensor** tag is used to create the list of events associated with each sensor. This information is appended to each condition defined by

the end-user (while creating the transitions between each screen). Later, these events will be used by the sensor's Android module.

The **name** tag specifies to what sensor a specific event belongs and the **driver** tag maps an action which will be coded in the Android module later.

As an example, the connection shown in Figure ?? produces the following event (Listing 4.6):

```

1 <event>
2   <source>1</source>
3   <target>2</target>
4   <condition_id>0</condition_id>
5   <condition_data></condition_data>
6   <allowback>>true</allowback>
7   <sensor>
8     <name>emg</name>
9     <driver>contraction</driver>
10    <stream>no-data</stream>
11  </sensor>
12  <deactivation>
13    <type>time</type>
14    <data>10</data>
15  </deactivation>
16 </event>

```

Listing 4.5: Example of an Event representation in XML

In this example, if the user is viewing Screen 1 and contract his/her muscle, then Screen 2 will be shown.

<deactivation tag>

This tag allows the programmer to deactivate an event for a specific amount of time. If the **data** tag is equal to 0, then it will remain always active. In the example given above, the event can only be fired once every ten seconds.

JavaScript file

The JavaScript file must contain all the necessary code to generate a graphical interface in the runtime emulator which allows users to simulate sensor events (Figure 21 and Figure 22). The runtime emulator is a tool included in DETACH's authoring environment which allows users to quickly preview how applications would look like in a real

smartphone. With the inclusion of sensors in the system, developers must provide a way to simulate sensor data because sensors won't be available in the web environment. The updated runtime emulator will be presented in more detail in Section 5.4.

First, developers should add the function `set [Sensor] Tab (sensorName)` where `[Sensor]` should be replaced with the name of the sensor. In this function, developers can use the function `createTab (sensorName)` which automatically creates the HTML markup to generate a new *tab* in the emulator (Figure 21) with the name passed in the argument `sensorName`.

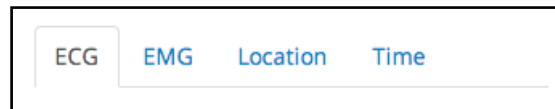


Figure 21: Emulator *tabs*

Developers should also define the graphical interface with the HTML elements that best suit the simulation of the events produced by the sensor. As a practical example let us consider the Location sensor. There are several distinct ways to create an interface that simulates the current geographical position of an user. On the one hand, we could create a set of input fields where users could insert the *latitude* and *longitude* of a position (Figure 22 - left). On the other hand, we could display a map where users could click and create a point which represents the current position (Figure 22 - right).

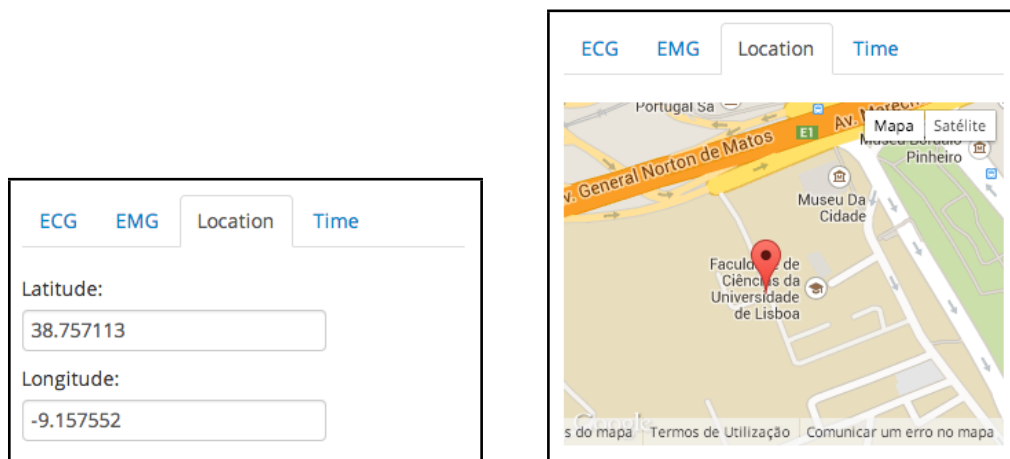


Figure 22: Alternative interfaces to simulate user's current location

After creating the interface, for every trigger previously defined in the XML file, developers must add a function `check [Sensor] Trigger [TriggerID]`. These functions are responsible to retrieve simulated data from the interface previously defined and

perform the evaluation of each trigger. Listing 4.6 shows the implementation of the Location sensor already available in DETACH.

```
1 function setLocationTab(sensor) {
2   var tab = createTab(sensor);
3   tab.append('<div id="mapCanvas"></div>');
4 }
5
6 function checkLocation1(eventConditionNumber, eventData, sourceScreen,
7   destScreen, connection_id) {
8   var data = eventData.split("_");
9   var lat = data[0];
10  var lon = data[1];
11  var rad = data[2];
12  var point = new google.maps.LatLng(lat, lon);
13  var currentPos = new google.maps.LatLng(marker.position.lat(), marker
14    .position.lng());
15  if (google.maps.geometry.spherical.computeDistanceBetween(currentPos,
16    point) <= rad){
17    setConditionAsTrue(sourceScreen, destScreen, connection_id,
18      eventConditionNumber);
19  }
20 }
```

Listing 4.6: Location sensor JavaScript file

4.5 Summary

Integrating sensors in DETACH required several modifications in the system's core components and in this chapter we presented an updated architecture of DETACH Mobile - an Android runtime environment. The renewed architecture comprises: a) a set of Sensor Processing Units responsible for collecting sensor's data and trigger events within the system; b) an Event Manager used to interpret events triggered; c) an Application View which renders mobile application's contents. This architecture uses an event-driven approach and ensures that: i) applications are capable of collecting and interpreting sensor data; ii) applications are capable of using sensor data to trigger certain behaviours within them.

Developers have an important role in creating and maintaining digital content that supports the requirements imposed by domain specialists. In this chapter we presented the necessary steps to add a sensor to both environments: runtime and authoring. Adding a new sensor in the runtime environment encompasses the development of an Android module that provides the mechanisms to communicate with a sensor and retrieve data. In the authoring environment, developers must provide the UI elements which enable the composition of applications that use the sensor.

Chapter 5

Authoring Sensors

Even though previous DETACH trials have been successful, we noticed that there was still some space to improve the user experience of this tool. Based on the feedback provided by users, we tried to improve several key interface elements and tasks, which are described with more detail in this chapter. Some of these changes are motivated by the fact that users are accustomed to some interaction patterns associated with their operating system, such as using the context menu triggered by a right mouse button click to find hidden functionalities and interactions. Besides, we also tried to automatize some tasks (e.g. creating connections between screens) to avoid the repetition of the same action over and over.

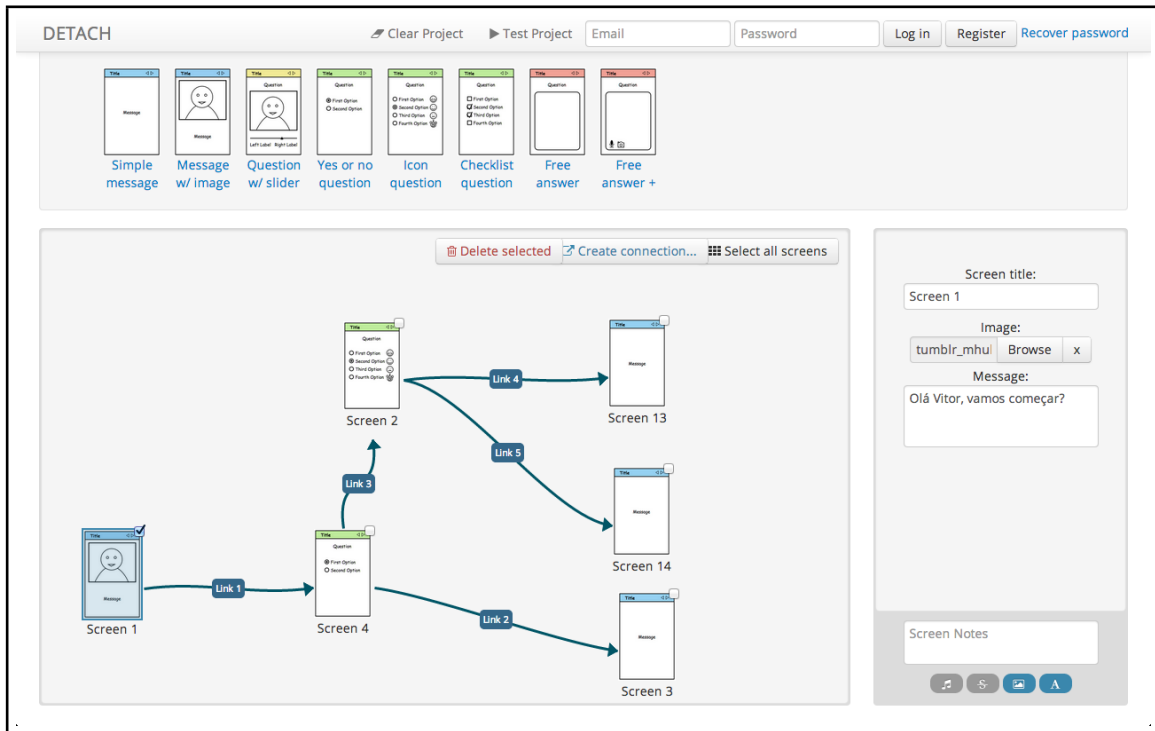
We also describe the necessary changes in the authoring environment in order to use sensors in the composition of applications. These changes include the addition of new sensor templates in the tool's interface, giving users the ability to use sensors to control the behavior of an application. The runtime emulator was also improved in order to properly emulate applications that rely on sensors.

5.1 DETACH interface

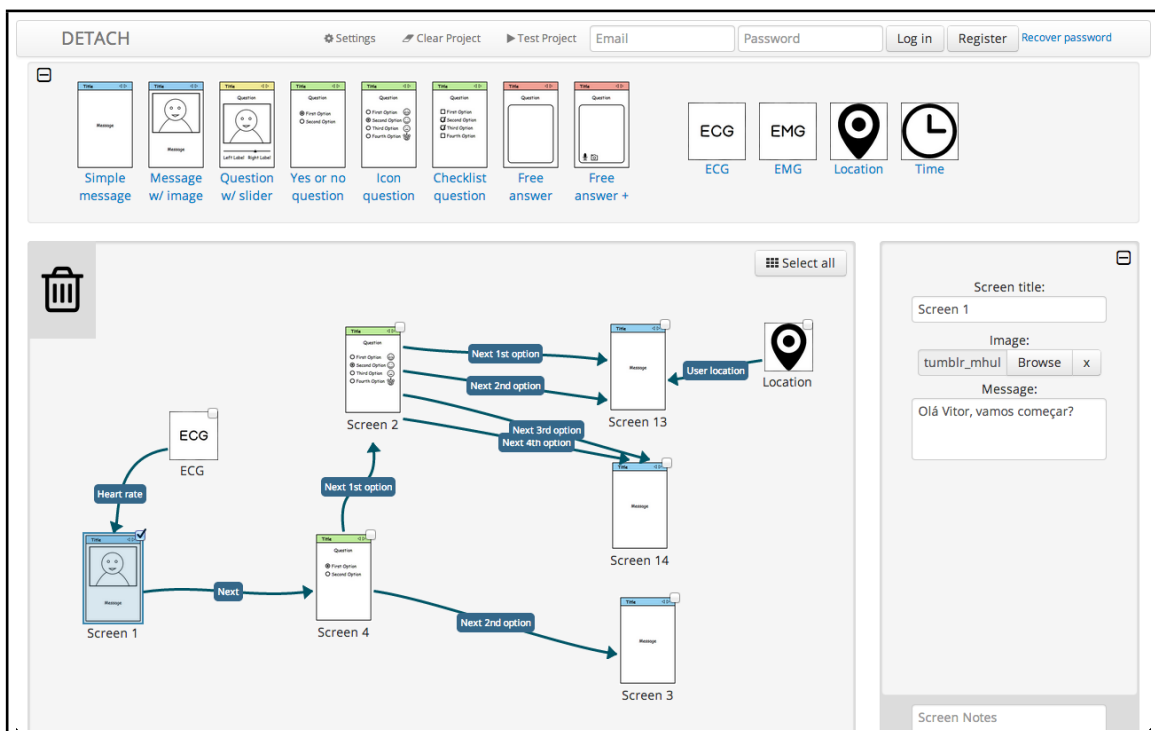
Although some elements were modified, DETACH's interface still remains divided in three distinct areas:

- (a) A top section which displays the available screens and sensors templates.
- (b) A central canvas to which the templates can be dragged and organized.
- (c) A configuration panel on the right side that enables the customization of the different elements in the canvas.

Figure 23 shows DETACH’s previous interface (on top) and DETACH’s newest interface (on bottom).



(a) Initial interface



(b) Improved interface

Figure 23: Comparison between initial and improved DETACH’s interface

The most significant difference in the newest interface is the inclusion of sensors in the authoring environment. The introduction of these elements are described in the Section 5.3.

5.2 Quality of life improvements

Feedback from domain specialists, who participated in previous DETACH trials, commented that during the composition of applications they needed to repeat the same action several times. This issue is mainly due to the creation of transitions from one screen to another, and it's particularly visible when a user composes a more complex application that has a large number of screens. Another issue is that when a user creates a new connection, a configuration panel pops up (Figure 24) which interrupts the user's workflow. This panel is used to establish the conditions required to transit from one screen to another. Inside the panel users have a visual indication of the source and target screens (Figure 24 - top right corner). Here, users can also delete the connection.

While more complex rules can be defined for some transitions, we noticed that most of the transitions occur when a user presses the "next" button. Therefore, to address the previous problems we made some modifications in order to automatize and ease the composition of an application.

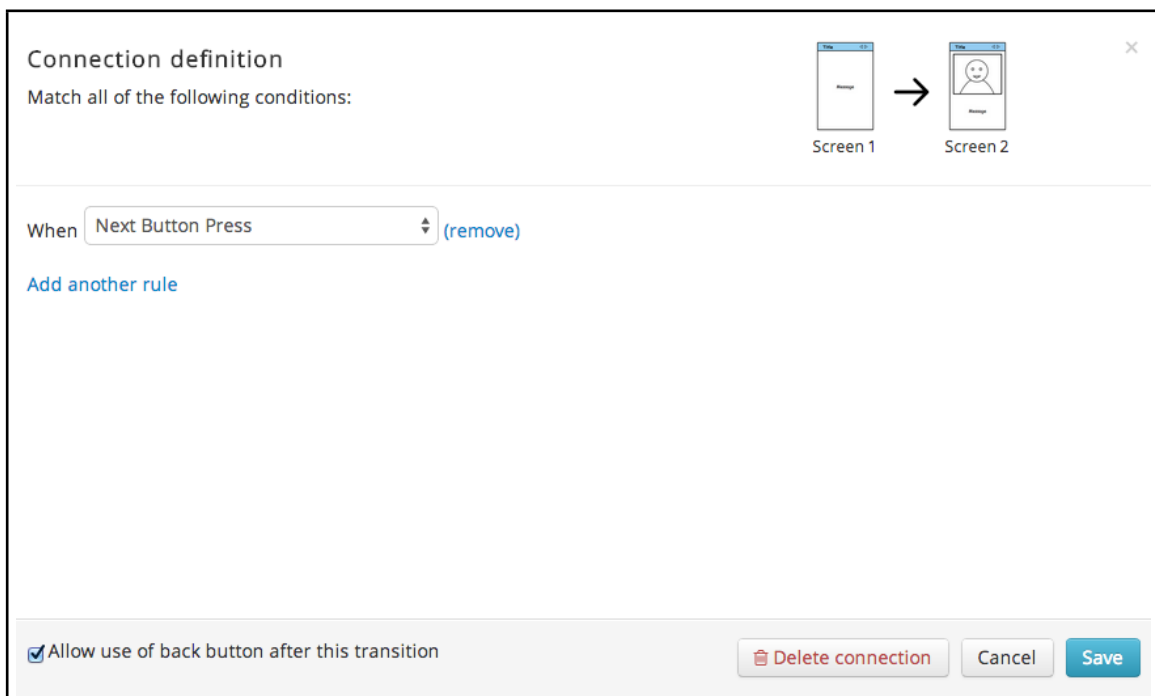


Figure 24: Panel which allows the configuration of a new transition between screens

5.2.1 Creating connections

In the most recent version of DETACH, we changed the way transitions are defined. Instead of selecting a source screen, hitting a button to create a new connection (Figure 25) and selecting a destination screen, now users can create transitions by right-clicking the source screen (automatically entering in "creation" mode) and selecting the destination screen. We still use the origin-destination connection strategy since it was the most popular strategy adopted by users in previous participatory design sessions.

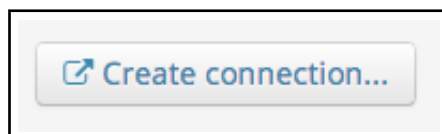


Figure 25: Button to create connections (old interface)

After the creation of a new transition, users aren't presented anymore with the panel which allows the configuration of a new transition (Figure 24). Instead, a default condition is chosen (the "next button" is the most common choice, as stated before) and users can configure the transition details later if they want. To edit transitions, users can select them in the canvas and use the panel located on the right side of the interface (this feature is explained with more detail later in Section 5.2.2). In alternative, they can also double-click in a transition (in the canvas) and the configuration panel described before (shown in Figure 24) pops-up again.

Visual feedback

With the possibility to drag sensors templates to the canvas, it also became necessary to add visual feedback during the connections' creation to help users validate their actions. Users can only create connections whose source is either a screen template or a sensor template; the destination must always be a screen template (Figure 26 - left). Creating a connection whose destination is a sensor template results in a invalid action (Figure 26 - right).

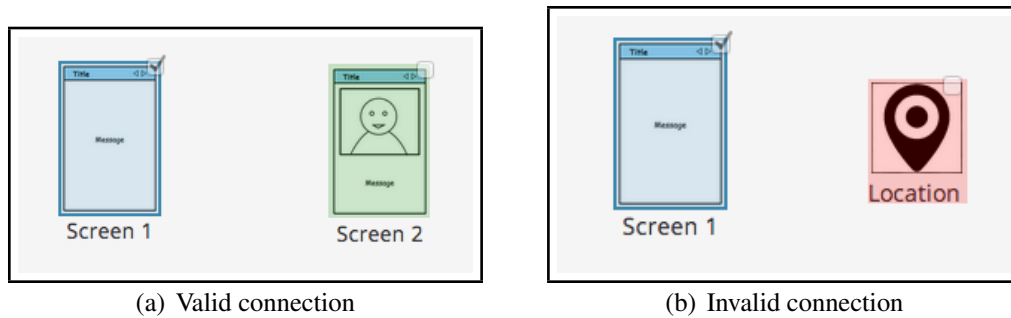


Figure 26: Visual feedback when creating connections

AND/OR connections

Another substantial difference in connection establishment is related to the possibility of establishing multiple connections from the same source to the same destination.

In the previous version it was possible to specify only one connection from the same origin to the same destination. Then, in the connection's configuration it was possible to use a combination of conditions to trigger the connection with the use of the conjunction operator (AND) or disjunction operator (OR) (Figure 27). Using the AND operator means that all conditions must be verified in order to move to the next screen. On the contrary, using the OR operator means that at least one condition must be true to activate the transition.

However, this approach limits the complexity of triggers that we may establish. For instance, with this approach it's impossible to specify a transition that matches [Condition A and Condition B] OR [Condition C and Condition D].

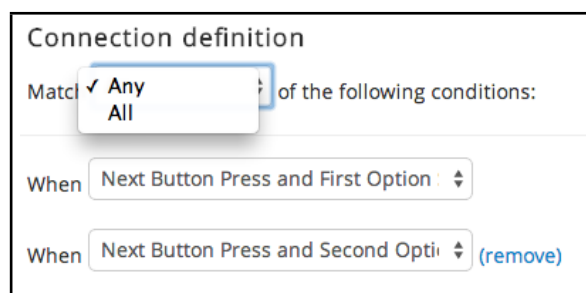


Figure 27: AND/OR evaluation mode

Now, with the possibility to create multiple connections from the same source to the same destination, each connection is seen as an alternative way to transit from one screen to another (disjunction operator - OR). The conditions configured inside each connection are verified with the conjunction operator (AND), meaning that all rules must be true in

order to transit from one screen to another. Figure 28 shows an example where there are two distinct ways to go from one screen to another.

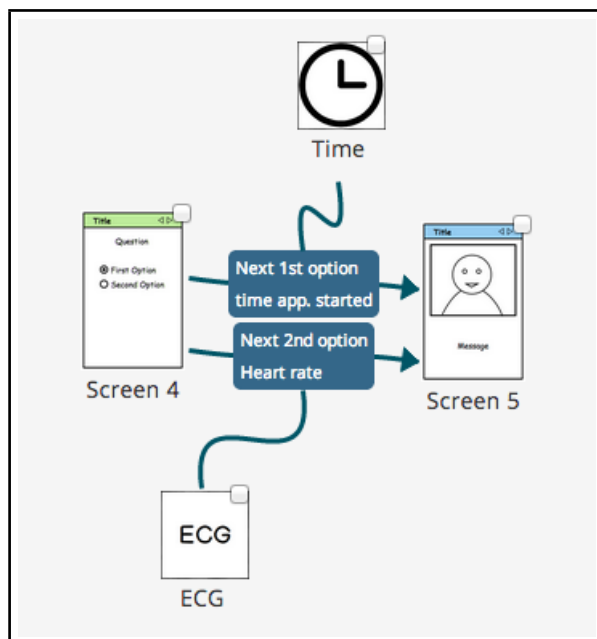


Figure 28: Alternative ways to transit from one screen to another

5.2.2 Editing connection details

In the previous interface, there was little visual feedback about the conditions that trigger a transition between screens. Each transition drawn in the canvas had a label containing the text "Link N ", where N denoted the *id* of the transition (Figure 29 - left). This information was rather useless, and if a user wanted to know what conditions trigger a transition, the only option was to open the transition configuration panel to get that information. The same happened if a user wanted to delete a transition.

The first step to improve this situation was the inclusion of a list of all conditions in the label referred before (Figure 29 - right). Now, it becomes much easier to visually identify the conditions that trigger the transition.

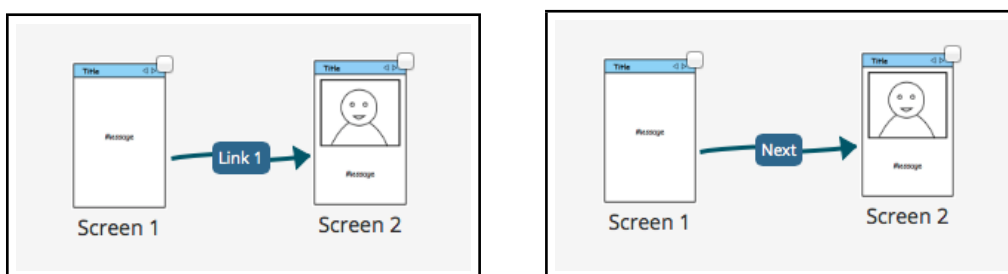


Figure 29: Information displayed in transition's label

When a user selects a screen in the canvas, the configuration panel on the right side displays all the customizations of the selected screen (Figure 30). However, selecting a connection in the canvas used to pop up the transition configuration panel where users could personalize the conditions.

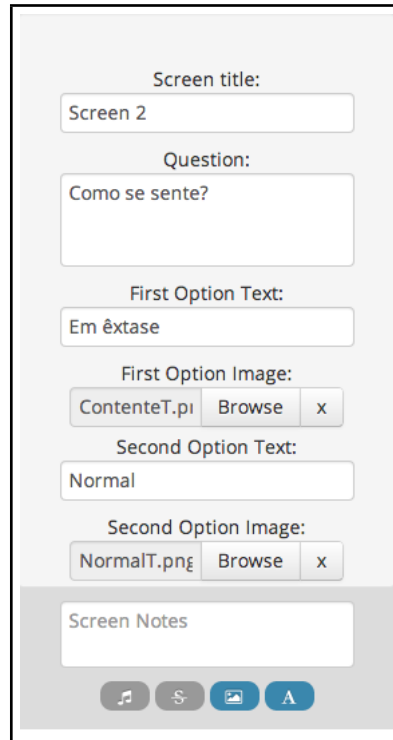


Figure 30: Screen configuration panel

Once again, we found that sometimes this panel was very intrusive and therefore we decided to follow the same pattern used in screen selection. This means that, now when a user selects a transition in the canvas, instead of showing the configuration panel to the user, the details about the transition are shown in the right side panel (Figure 31).

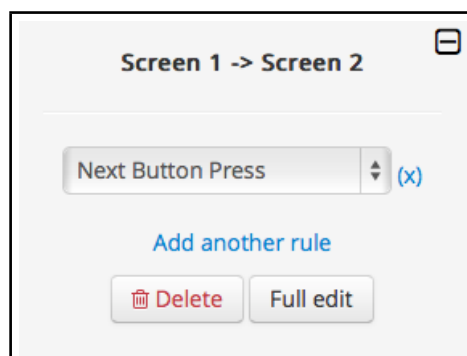


Figure 31: Transition details displayed in the right side panel

In this configuration section, users can add, edit and remove conditions. They can also

delete the transition. The full-screen configuration panel can still be accessed by double-clicking the condition in the canvas, or by using a button that exists in the right side panel (Figure 31 - bottom).

5.2.3 Deleting screens

In the most recent interface, we replaced the "Delete" button (Figure 32) by a trash can area (in the upper left corner of the canvas) where users can drag and drop screens (Figure 33 - left). This approach follows a common pattern among most of the operating systems and therefore is a well known practice for many users. Like in the previous version of DETACH, it's still possible to drag and drop back to the screen templates area or use keyboard shortcuts.

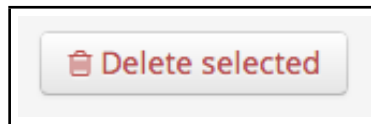


Figure 32: Button to delete selected screens

In addition, we also included a visual feedback indicating that a screen is being deleted when dragged on top of the trash can (Figure 33 - right).

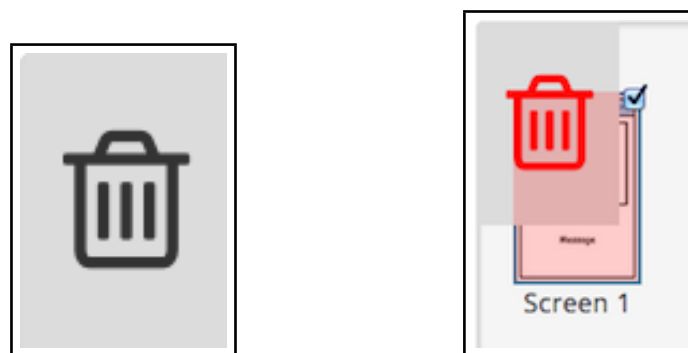


Figure 33: Trash can area, with visual feedback when deleting a screen

5.3 Using Sensors

A significant difference from the original interface comes from the possibility to use sensors during the composition of an application. In the following sections we describe several aspects encompassing the creation of applications with sensors, and how sensing applications work. We also present the available sensors in our authoring environment

and an updated runtime emulator that can simulate the applications created based on fake sensor data.

5.3.1 UI characteristics

We decided to follow the same approach used before with screen templates and therefore, we created a list of sensors templates (Figure 34). These templates are located in the top section of the interface and can be dragged into the canvas.

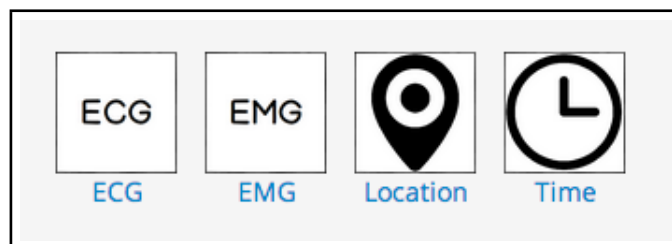


Figure 34: Available sensors templates

When a sensor template is clicked or dragged into the canvas, that sensor is added to the current application. Each sensor template features different triggers that can be used to control the behavior of an application.

5.3.2 Connection types

Sensors can be used to:

- (a) **Control transitions between screens:** users can add sensors to transitions, empowering that transition with additional triggers. For example, a *Simple message* screen contains only one screen trigger called *Next Button Press* (Figure 35). If we empower a transition with a *Time* sensor, then we can use three additional external triggers provided by the sensor (Figure 36). Adding sensors to transitions allow the specification of conditions based on sensor data (e.g user location, user heart rate).

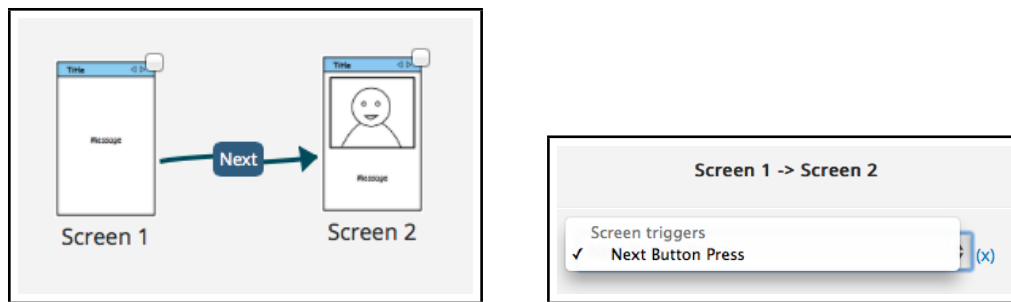


Figure 35: Transition between screens (without the use of sensors) and corresponding triggers

(b) **Trigger an external event:** Screens can be activated by external events (meaning that this type of events doesn't depend on a previous screen). This type of connections changes the way applications are activated. Before, without sensors in applications, users would have to manually start an application when they need to run it. With sensors, we can configure applications to start when facing specific context conditions, which is great to remove this burden from the user and allow the development of proactive applications. Common scenarios where proactive applications are helpful include:

- using a daily alarm in therapeutic applications to remind users to perform a specific action;
- opening the application when the user enters or leaves a specific location (e.g. hospital, home);
- activating an application when the user's heart goes above a specific value (Figure 37).

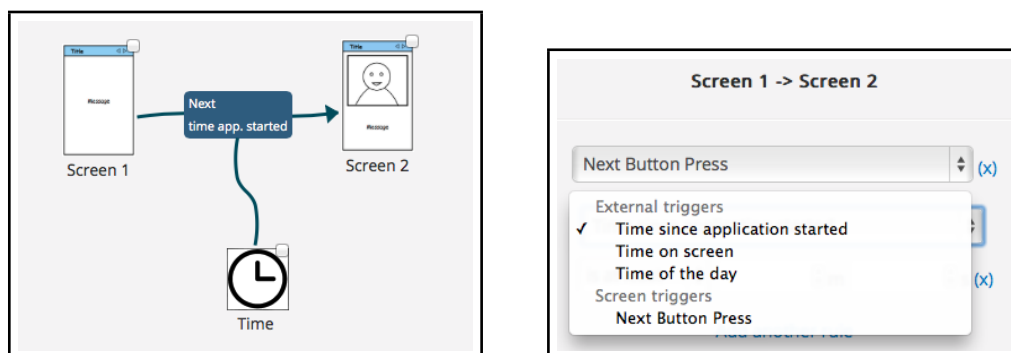


Figure 36: Transition between two screens using a sensor and corresponding triggers

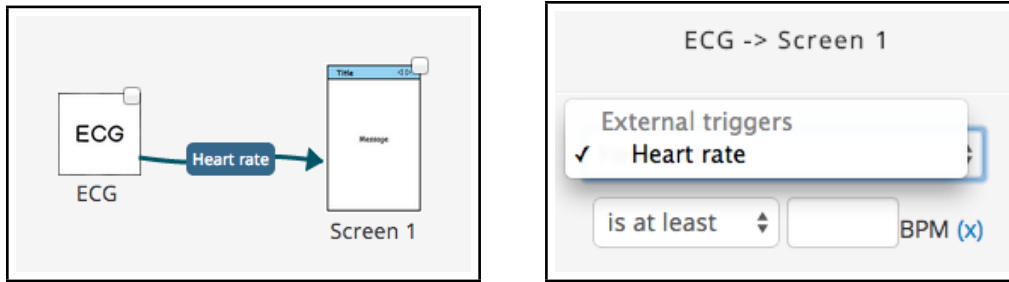


Figure 37: External transition using an ECG sensor and corresponding trigger

5.3.3 Available Sensors

In this section, we briefly present the sensors that are available in the authoring environment and can be used by non-expert programmers to compose applications.

Time

This sensor provides triggers based on time variables. Figure 38 shows the image representation of this sensor in the canvas.

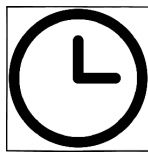


Figure 38: Time sensor representation

At this point, the following triggers are available:

- Elapsed time since the application started
- Elapsed time on screen
- Time of the day

Users can also apply a sub-condition of **is at least**, **is at most** or **is between values**. An example is shown in figure 39.



Figure 39: Time sensor condition trigger example

Location

The Location sensor provides one trigger based on the geographical location of the user. Figure 40 shows the image representation of this sensor in the canvas.



Figure 40: Location sensor representation

Non-expert programmers can make a specific trigger to occur on one or more geographical areas. Each geographical area is defined as a pair of coordinates in the form of (*latitude*, *longitude*) and a *radius*. However, to ease the creation of each area, users have at their disposal a map where they can quickly search for a specific location and mark each area.

This trigger is activated when the end-user walks inside of one of the specified areas. An example is shown in figure 41.

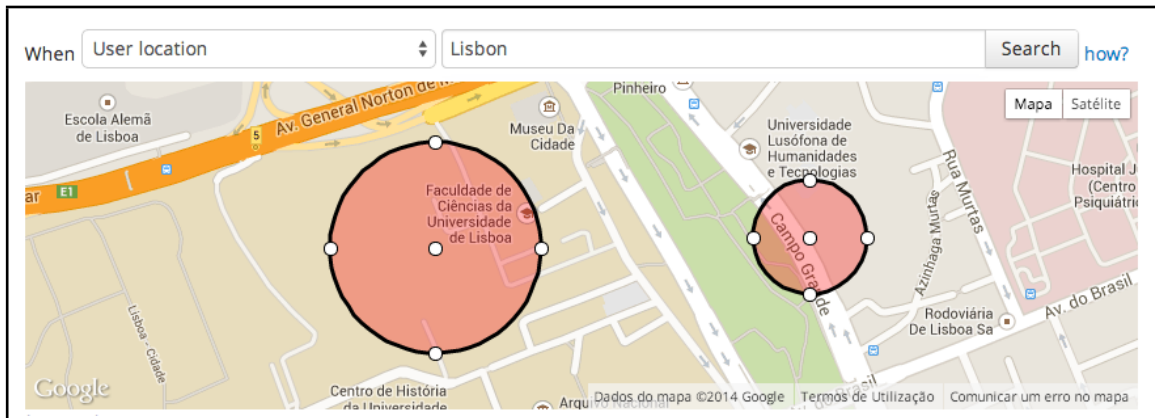


Figure 41: Location sensor condition trigger example

ECG

At this point, the ECG features one trigger based on the heart rate of an individual. Figure 42 shows the image representation of this sensor in the canvas.

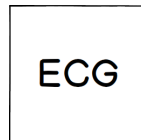


Figure 42: ECG sensor representation

Similar to the *Time* sensor, users can also apply a sub-condition of **is at least**, **is at most** or **is between**. An example is shown in figure 43.

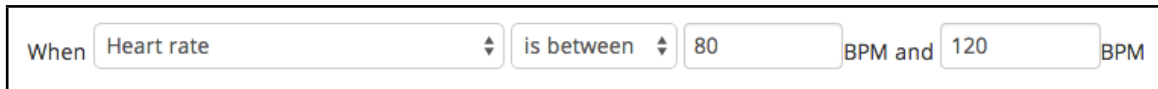


Figure 43: ECG sensor condition trigger example

EMG

This sensor uses triggers based on the electrical activity of the muscles. Figure 44 shows the image representation of this sensor in the canvas.

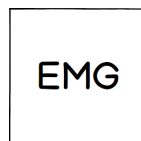


Figure 44: EMG sensor representation

So far, users can use the following triggers:

- Muscle contraction
- Muscle retraction (which occurs after a contraction)

Examples of conditions are shown in figure 45.

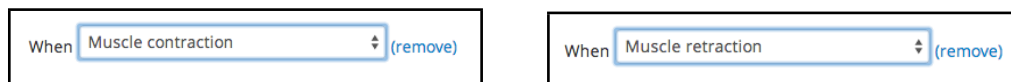


Figure 45: EMG sensor condition trigger examples

5.3.4 Scenario

As an example, we can imagine a scenario where João, a very successful Portuguese researcher, is going to Google's headquarters in London to have a technical interview.

Contrary to what would be expected, João does not feel anxious about the challenges that he will need to solve in the interview. However, in order to travel to London he must face one of his biggest fears: travel by plane.

Our goal is to create an application that helps João in his plane trip. In addition to the sensors available in his smartphone (Time and Location), he also travels with an ECG sensor to control his heart rate. This application will have the following behavior:

1. At any time, João can start this application manually. In this case, a screen questioning how is he feeling is shown. Depending on his answer, a different animation is displayed.
2. When João arrives at the airport in Lisbon, he will be prompted with a set of screens, containing a set of pre-flight questions.
3. Similarly, when he arrives in London, he must answer a set of post-flight questions.
4. During the flight, there may be situations where João starts to feel very anxious (his heart rate goes above 140bpm). In order to tranquilize him, a screen with a relaxing animation and his favorite music should be displayed for 6 minutes. After that, he must answer a question about his current state.

Let's start by accomplishing our first requirement. To do so, we will use an *Icon question* screen to question him about his feelings. Each answer will transit to a different *Message with Image* screen. Figure 46 shows João's application at this point.

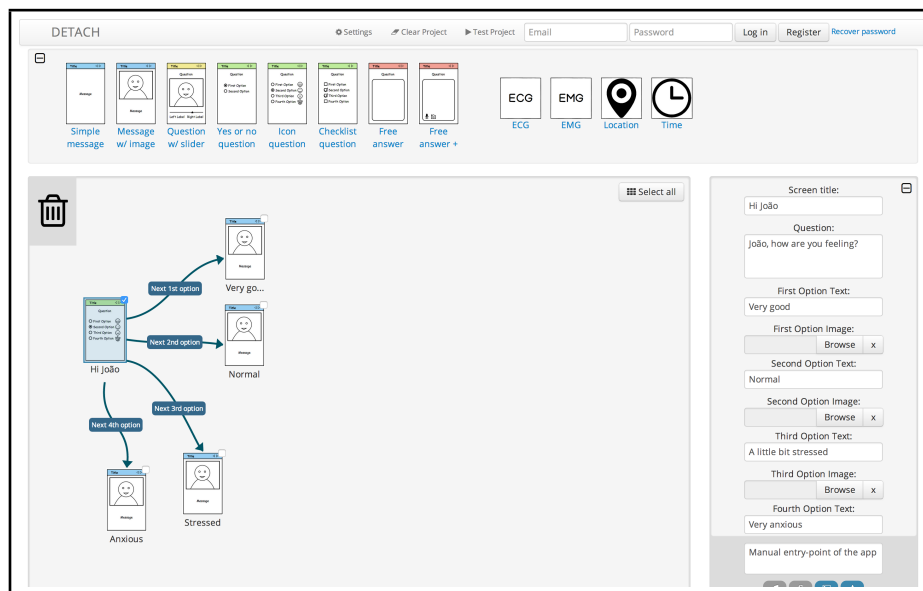


Figure 46: Scenario application - phase 1

Moving to 2), we need to add a Location sensor to track João's location. When he arrives at the airport in Lisbon, this sensor will automatically show the application to

João. After this, he will answer the pre-flight questions. Similarly, to achieve 3), we also use a Location sensor to know when João safely arrive at the London airport. When he does so, he will answer the post-flight questions. The implementation of these behaviors is shown in Figure 47.

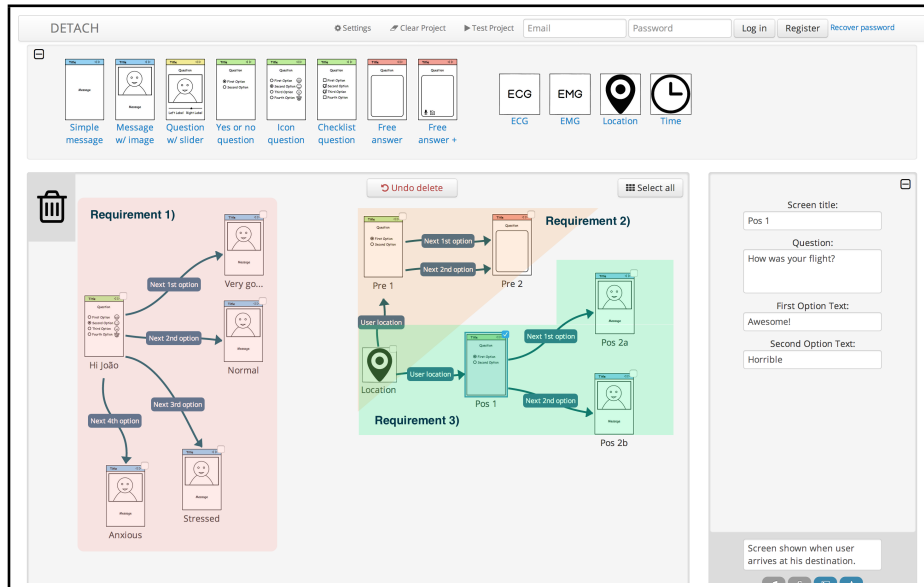


Figure 47: Scenario application - phase 2

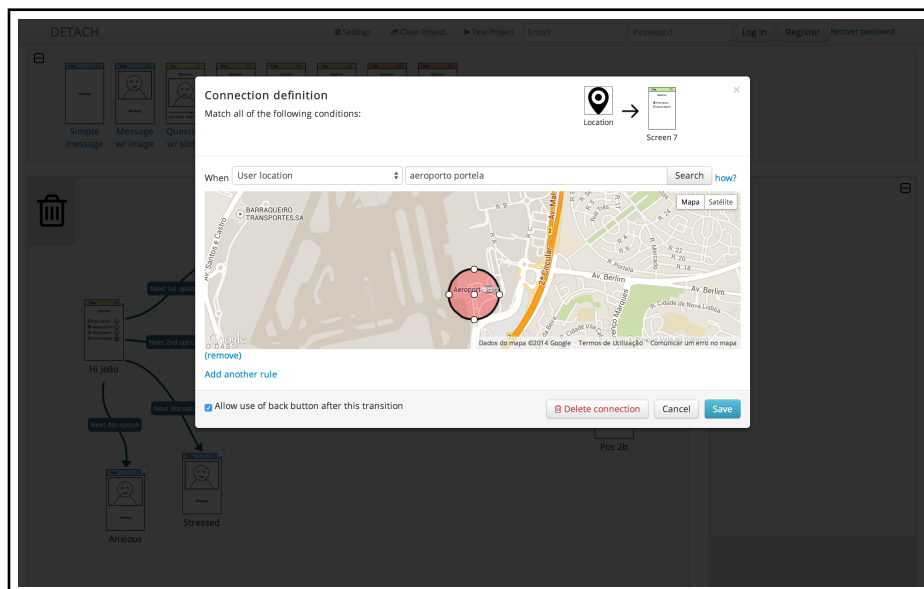


Figure 48: Specifying a location to trigger an event when the user goes inside it

Finally, to finish our application we need to implement the 4th requirement. We use an ECG sensor to control João's heart rate, and configure an event to show a screen when his heart rate goes above 140bpm. In this screen, we place a relaxing animation and

his favorite music playing. After 6 minutes, our Time sensor will automatically display the next screen, which is responsible to ask João about his current feelings. Our final application is shown in Figure 49.

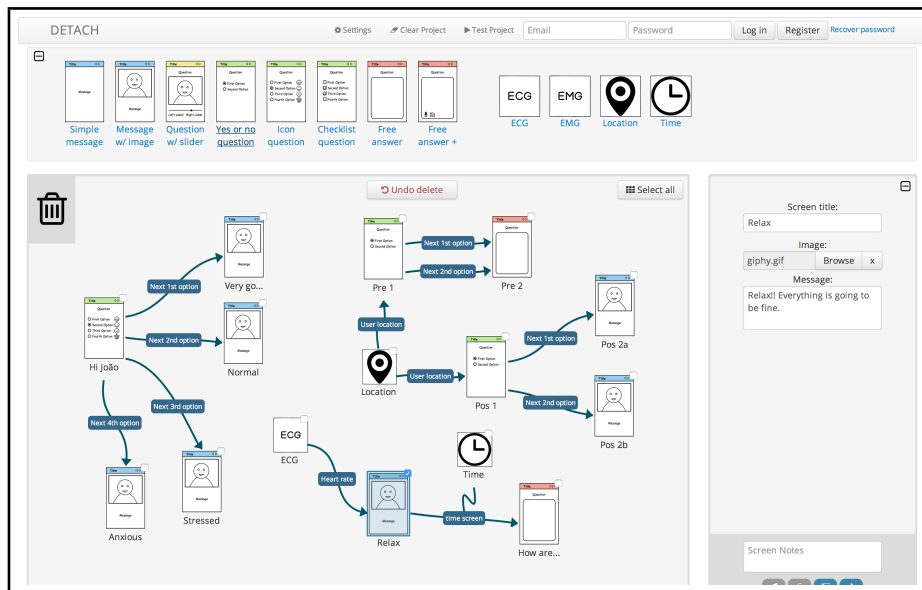


Figure 49: Scenario final application

With this example, we created an application that has multiple entry points (manually by intentionally activating the application at any time, automatically when João' heart rate is above 140bpm and when he arrives at the airports). We also use sensors to automatically transit from one screen to another (using a Time sensor that changes screen after 6 minutes).

5.4 Run-Time Emulator

DETACH features a Run-Time emulator which allows users to quickly preview how an application created would look in smartphones (Figure 50). This emulator allows a person to test their application's interface, content, styling and behaviour, and avoids the need of a real smartphone while the user is composing the application.

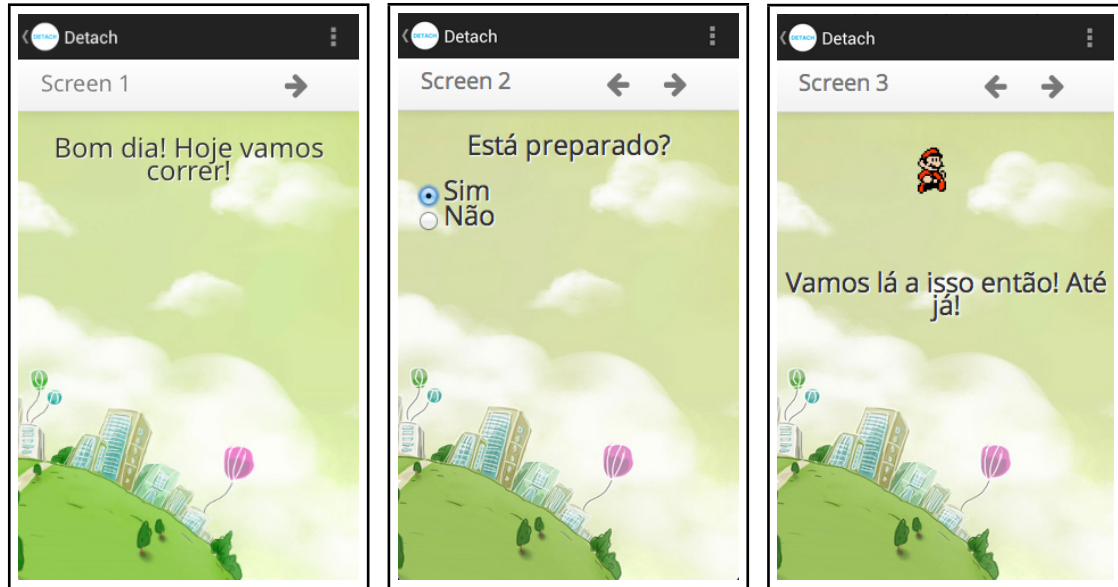


Figure 50: DETACH application running in the DETACH Run-Time emulator

With the inclusion of sensors in the authoring tool a new problem emerged because sensors (and consequently, sensor data) aren't available in the web environment. Also, since users can now control the behavior of an application with sensors, this emulator would fail to reproduce the events offered by sensors (due to the lack of sensor data) and therefore it becomes impossible to properly test the behavior of an application. To address this problem, we decided to append a new section to the emulator, which is responsible to emulate sensor data (e.g user's geolocation, user's heart rate). This data will be used to verify the veracity of events controlled by each sensor. Figure 51 shows the Run-Time emulator with the extra section.

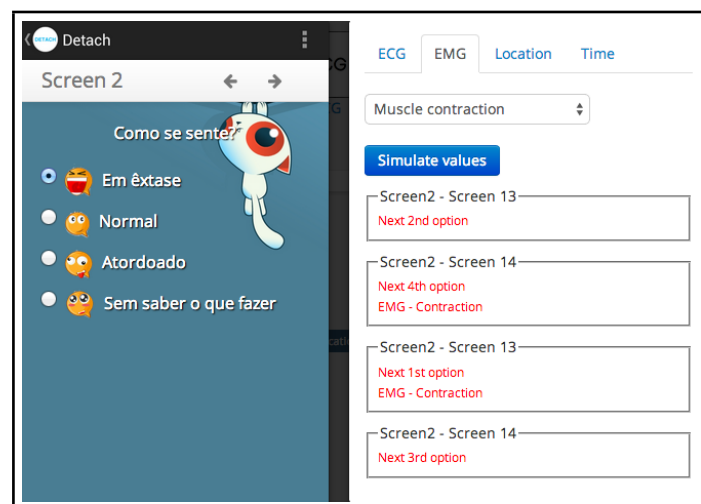


Figure 51: DETACH run-time emulator with an extra section to simulate sensor's events

In the above figure we can see that this section contains a set of *tabs*, each one corresponding to a sensor available in the tool. Each *tab* contains an interface to simulate input data from a sensor. This interface is designed by the developers who are responsible for adding new sensors to the system. Each interface can be different because each sensor provides different data types. This also gives developers the flexibility to design interfaces that best suit each sensor.

At the bottom of the extra section, we also appended a summary of all possible transitions from the current screen. This summary let the users know to what screens they can transit and how they can do it.

Finally, to properly simulate a state of a sensor, users must click the button "Simulate values". This button uses the data inserted in the fields (of all *tabs*) to evaluate conditions associated with each sensor. When a specific condition is true, the color changes from red to green. This behavior can be seen in Figure 52.

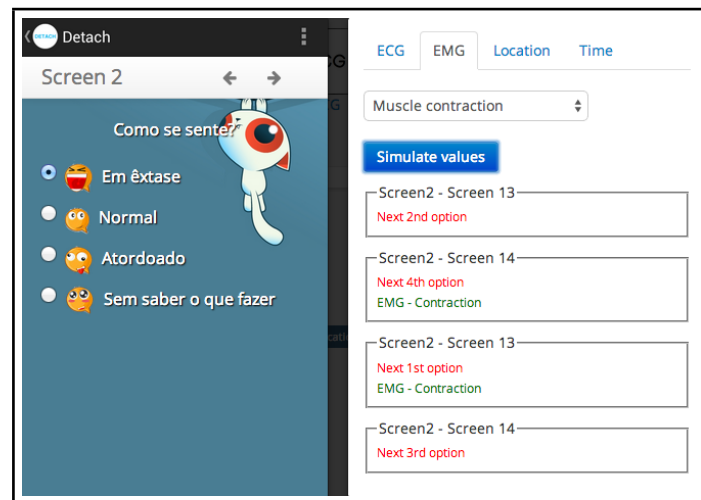


Figure 52: DETACH run-time emulator after pressing the "Simulate values" button

When all the conditions of a transition are true, the Run-Time emulator displays the next screen.

5.4.1 Scenario

To explore the runtime emulator, let us resume João's scenario presented before, and use the application created in the previous section. If we simulate our application in the emulator (Figure 53 - left), we are acting as if João would manually activate his application (obviously, without context-data from sensors we can't just start the emulator automatically). In this case, we see a screen asking about João's feelings. Based in our answer, a new screen will be presented (Figure 53 - right).

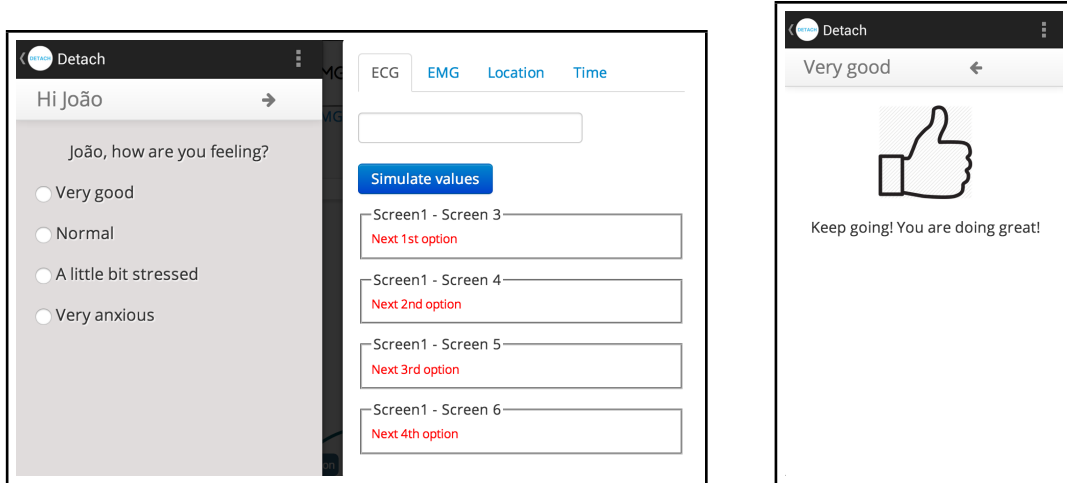


Figure 53: Scenario application emulation (left), new screen shown after answering the question (right)

To simulate when João arrives at the airports or when his heart rate is above 140bpm, we can use the right panel of the emulator interface to input some data. If this application was running in smartphones, the application would automatically open, but once again, we can't simulate this behavior here. However, using fake input data we can simulate a similar behavior just to preview screen's contents. Figure 54 emulates the screen presenting the pre-flight questions displayed to João when he arrives at the airport in Lisbon.

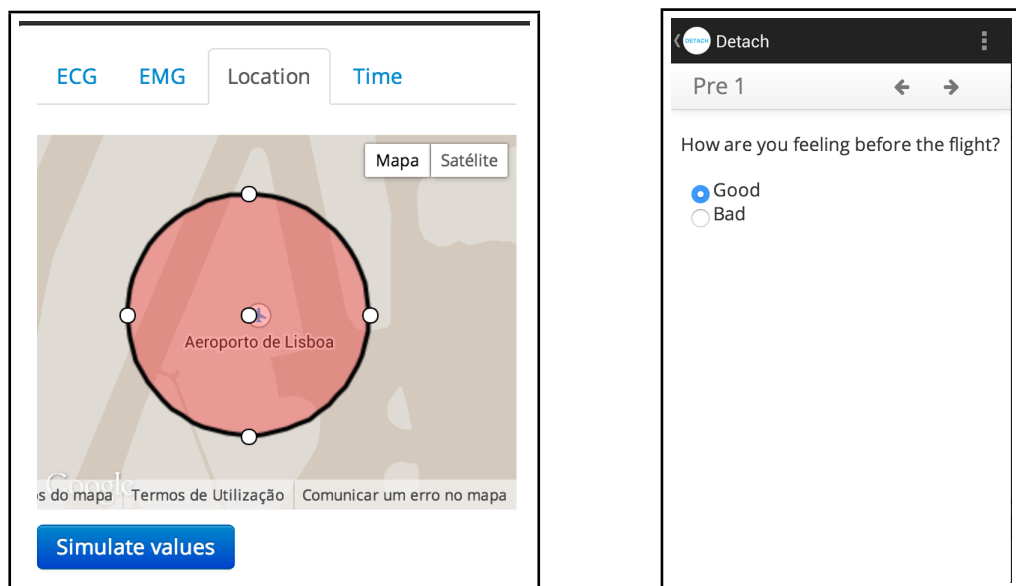


Figure 54: Screen activated (right) when João arrives at the airport (left)

With a similar approach, we can simulate João's heart rate to preview the screen we created when he gets too anxious during the flight (Figure 55). If we input a value of

160bpm (which is greater than 140bpm, that was the value we specified in the condition when we composed our application), this screen will be displayed to João.

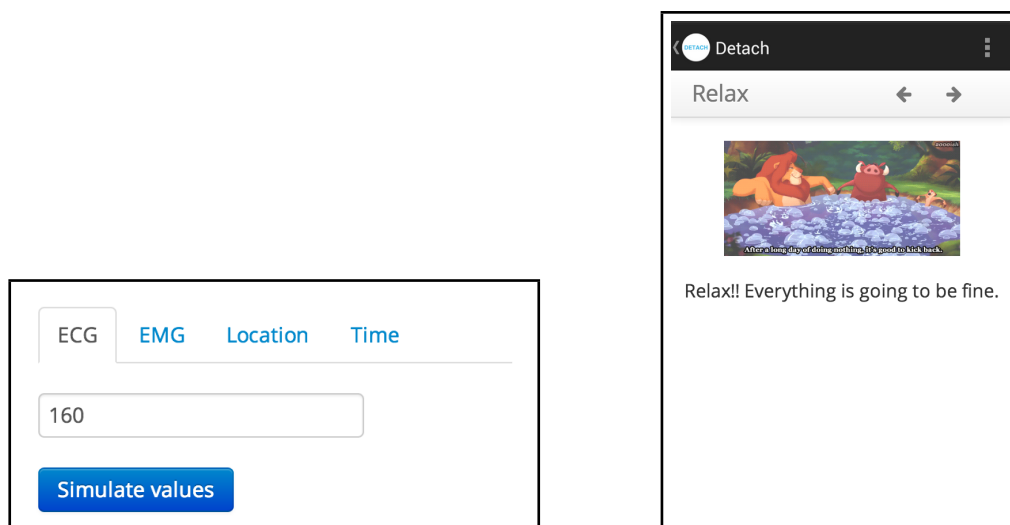


Figure 55: Screen activated (right) when João's heart rate goes above 140bpm (left)

There is no point in presenting all the screens that we previously created in our application. These examples are enough to demonstrate that even without having real context-data in our authoring environment, we can improvise and fake some context-data in order to preview the screens of our application.

5.5 Summary

In this chapter we presented several modifications in the authoring environment in order to improve the user experience. Some of these modifications were motivated by feedback given in previous DETACH trials while others were to provide the means to create applications with sensors.

The quality of life improvements presented in this chapter were done to facilitate some tasks, such as creating and editing transitions between screens. Some of these tasks were repeated over and over and this issue was particularly visible when users created more complex applications with a large set of screens. We also introduced general UI improvements, such as visual feedback to help users validating their actions and replacing UI buttons with common interaction patterns (e.g. drag and drop UI elements to a trash can).

Regarding sensors, we added a set of sensor templates that can be used to control the behavior of the applications. Sensor templates can be dragged into the canvas and can:

a) be attached to a screen transition to provide additional triggers; b) be used to trigger external events allowing the development of proactive applications. So far, DETACH features a Time sensor, a Location sensor and two physiological sensors: an ECG and an EMG. Lastly, we improved the existing runtime emulator in order to support the emulation of applications with sensors. Sensors aren't available in the authoring environment and therefore we extended the emulator UI interface to support the simulation of sensor events based on fake input data.

Chapter 6

Evaluation

After all the development process, the system was submitted to an experiment. Since the focus of this work is the inclusion of sensors in DETACH, our goal was to perceive the complexity of adding new sensors to the system in order to ensure future updates.

This experiment focused mainly in the process of adding new sensors to the system (as explained in Chapter 4). This process can be split in four different steps:

1. Create a new sensor template in DETACH (authoring tool).
2. Compose an application to test if the sensor template is working as expected.
3. Create and deploy an Android sensor module, that is responsible to communicate with the sensor, collect data and process it.
4. Use DETACH Mobile to run the DETACH application created in step 1 in order to evaluate if the outcome is working as expected.

In this session, subjects were asked to add an EMG (electromyography) sensor to DETACH with one trigger (muscle contraction).

6.1 Participants

13 subjects (12 male, 1 female) were recruited, with ages ranging from 23 to 35 years old ($\bar{x} = 27$, $SD = 3.4$). All of them have at least a bachelor's degree in one area of Information Technologies and were comfortable with both Portuguese (native language) and English (professional working proficiency) languages.

6.2 Equipment and Tools

Subjects were handed a laptop (MacBook 13”) with an external mouse attached, previously loaded with the required development tools to create and add a new sensor in the system. Even though some participants weren’t familiar with Apple’s OS, they easily got used to the differences to the other operating systems (mainly keyboard shortcuts) and therefore it didn’t affect their performance.

The tools used were:

- A text editor (Sublime Text) to create XML and JavaScript files.
- An IDE (Eclipse) that supports the development of Android applications, with the Google ADT’s plugin installed.

Subjects also had access to a Shimmer’s EMG Sensor prepared with 2 electrodes that were placed on the subject’s forearm as shown in Figure 56 and an Android smartphone (Google Nexus 5). During the session, subjects also had a developer’s guide¹ with example code and general guidelines to help them finish the required tasks. Besides, participants had at their disposal a researcher to clarify on any questions regarding the equipment or tools used.

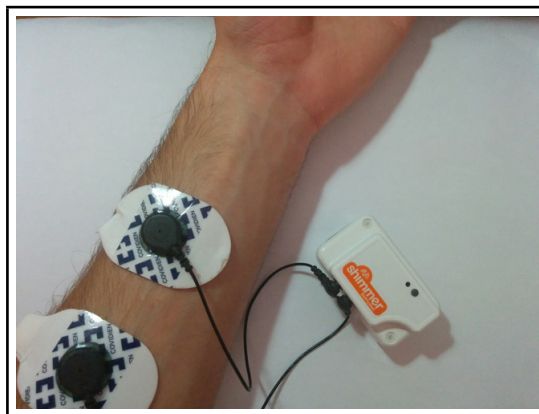


Figure 56: EMG electrodes placement

6.3 Metrics

We chose the following metrics:

- **Time to complete the integration process** - total time required to conclude all the necessary steps to add a new sensor.

¹<http://accessible-serv.lasige.di.fc.ul.pt/~afjusto/detach/webtool/howto.html>

- **Number of unsuccessful attempts in template creation** – this metric reflects the number of **unsuccessful attempts** by each participant to create a new sensor template in DETACH. Unsuccessful attempts were considered when the sensor’s trigger created by the subject wasn’t working properly or the runtime emulator couldn’t simulate the trigger defined.
- **Number of unsuccessful attempts to create the Android sensor module** – this metric reflects the number of **unsuccessful attempts** by each participant to create a new Android module which is responsible to collect and process the data. Unsuccessful attempts were considered when the sensor deployed by the subject wasn’t working as expected (e.g. couldn’t connect to the smartphone, wasn’t recognized by DETACH Mobile, couldn’t trigger events).

6.4 Procedure

This experimental session was comprised by a pre-task and a set of tasks that included all the necessary steps to add a new sensor to the system. Participants were asked to accomplish the following tasks:

- **Pre-task** - a development guide describing the process in detail was handed to subjects in order to give them a list of guidelines necessary to complete the tasks with success. This guide also had some code examples and the description of some relevant functions.
- **Task 1** - in this task we asked the participants to create a new sensor template. Each template is composed by three files:
 - A XML with the specification of the sensor details.
 - An image file.
 - A JavaScript containing information about how DETACH runtime emulator can simulate sensor values.
- **Task 2** - in order to evaluate if the template was working, participants had to compose a new application that used the trigger defined before in the XML file. This application was used in Task 4 to evaluate if the whole system was working as expected. Subjects also used the runtime emulator to evaluate if the emulator was properly simulating the trigger defined.
- **Task 3** - here, subjects created a new Android project in Eclipse and set-up the required library to develop a new Android module. After configurations were done, they were asked to program the sensor’s driver that is responsible to use the data

collected by the sensor and program the logic that should be applied to the data. After this, they deployed the sensor's APK to the phone.

- **Task 4** - in this task, users launched DETACH Mobile (already installed in the smartphone). Then, they synchronized the DETACH application created in Task 2 and started the EMG sensor, in order to start collecting data. Finally, they launched the application and checked if the behaviour of the sensor was working as they expected.
- **Post-task** - to conclude, participants were asked to give feedback about the session in order to evaluate and improve the process.

6.5 Results

All participants were able to perform the proposed tasks in an average time of 84 minutes. As shown in Figure 57, creating a sensor template (task 1) and an Android module (task 3) took in average 32 and 28 minutes, respectively. On a side note, a similar evaluation (prior to this work) reported in [44] accessed that developers took in average 35 minutes to add a new screen template (instead of a sensor template) to DETACH. Therefore, we can conclude that adding a screen or sensor template takes roughly the same time (35 and 32 minutes, respectively). Our efforts to use similar approaches regarding the addition of screens and sensors templates in the authoring environment are emphasized by obtaining identical development times.

Even though the applications created in Task 2 were very simple (two or three screens and one or two sensors), subjects took an average of 16 minutes to complete this task. Since most users didn't explore the tool at the beginning of the trial (the majority started to code the sensor template right away), this was the first time that they interacted with the tool's interface, which led to a bigger completion time. When subjects needed more than one attempt to successfully add a sensor template, they needed to re-create their applications (after fixing the bugs which caused the unsuccessful attempt) and we could observe that they started to compose applications much faster as soon as they felt familiarized with the tool. Nevertheless, our primary goal was to access the complexity of adding new sensors to the system and not the usability of the tool.

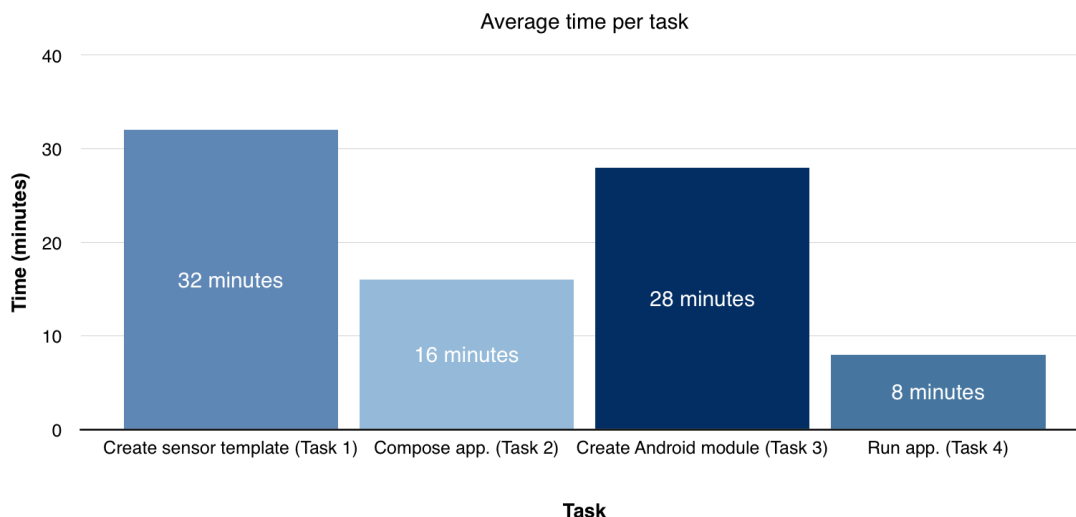


Figure 57: Average time per task (task 1 to 4, respectively)

Adding a new template in DETACH (Task 1 & 2)

Most participants preferred to reuse the code snippets provided in the developer's guide, instead of creating the files from scratch. They easily understood the pieces of code that required some modifications in order to match the experiment objectives. However, this approach also led to a few basic mistakes (variables and functions misspelling) which were the cause of some unsuccessful attempts, specially when filling the Javascript file. All subjects filled the XML file without any problems or questions.

When filling the Javascript file, some subjects had some doubts about what they were coding. This issue was due to the fact that all of the subjects started the coding process without exploring the functionalities of DETACH, in particular the runtime emulator. However, they didn't have problems understanding how to code the required functions.

As we explained in Section 4.4, developers need to code and generate a graphical interface in the runtime emulator which allows users to simulate sensor events. Some subjects with less expertise in Javascript and HTML suggested that we should provide a base implementation of this interface and let developers override this implementation when necessary, arguing that this would reduce the learning curve and help people with less expertise in these programming languages. This is an important observation which will be used as a guideline in future improvements in order to ease this task.

As shown in Figure 58, the majority of the participants implemented the template after one unsuccessful attempt.

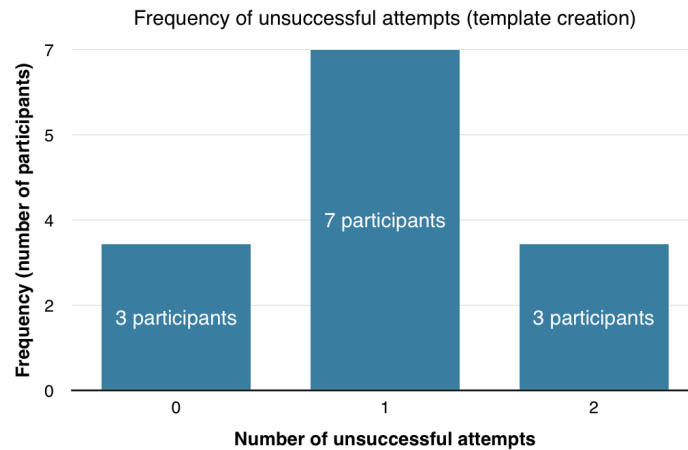


Figure 58: Frequency of unsuccessful attempts in template creation

Subjects didn't check any code from other sensors (except the code that was available in developer's guide) and the feedback about adding a new template was positive (Figure 59).

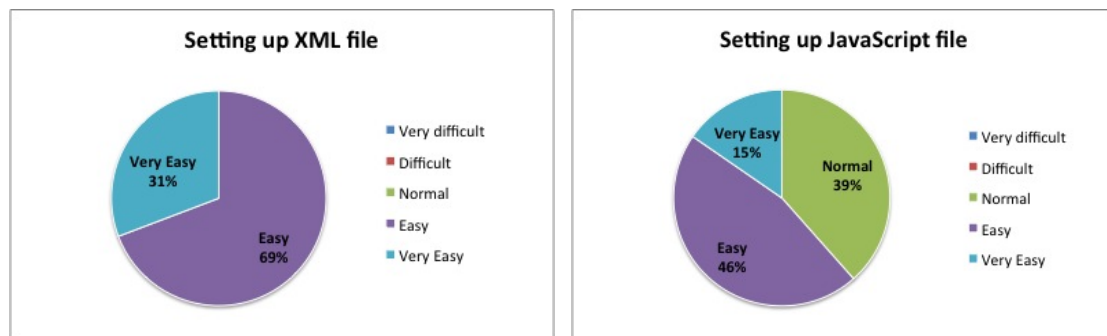


Figure 59: Feedback about adding a new template in DETACH

Participants didn't watch any tutorial before attempting to create an application. They were able to understand by themselves how to add screens/sensors to the canvas and how to delete them (by using the trash can area introduced in this work). However, initially some of them tried to click in the trash can to delete the selected elements before understanding that it was supposed to drag and drop. When they needed to specify the transitions to give the application some behavior, they weren't able to understand how to enter in creation mode. After explaining to them how to do so, the visual feedback helped them understand what actions were valid or invalid. The majority attempted to delete transitions the same way they did with screens/sensors but this feature is not implemented yet. Therefore, they got confused and asked how they could perform such action. Overall, all participants created their applications without wasting much time with doubts about how to perform a specific action. As we stated before, when they needed to re-create their

applications after unsuccessful attempts, they progressively started to create applications much faster.

Adding a new sensor module in Android (Task 3 & 4)

Once again, participants didn't check any code from others sensors and preferred to reuse the code provided in the developer's guide. Overall, we felt that subjects had less problems to create the module in Android when compared to the template creation. In the opinion of some of the participants, the development of this module was more straightforward. However, this may be explained by the fact that some of the developers are more proficient in Android programming than web programming (specially programming in Javascript).

Project and environment configurations were perfectly handled by everyone. When filling the sensor's driver, a few participants took some time to understand the relation between the trigger previously specified in the XML file (task 1) and how they would map it in order to fire an event when the muscle is contracted. This was the most common mistake made by 7 participants (Figure 60) who needed more than 1 attempt to successfully finish this task and was mainly due to the lack of information in the developer's guide.

Also, some users suggested that would be better if they had more detailed information about each function (in particular the `processData()` function, which confused some users that didn't understand if they should implement this function, or not).

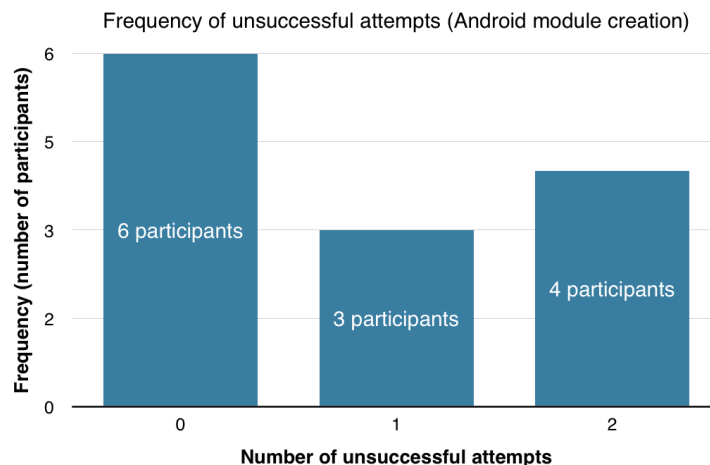


Figure 60: Frequency of unsuccessful attempts when developing the Android module

Overall, the feedback about these tasks was also positive (Figure 61).

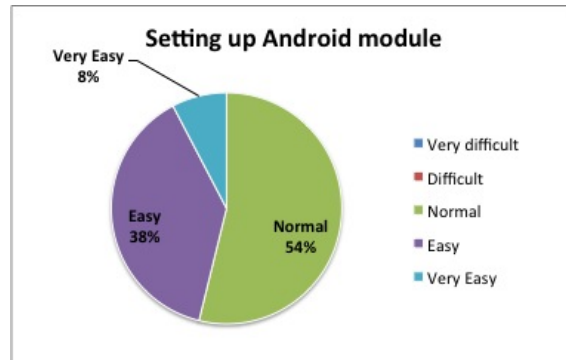


Figure 61: Feedback about adding a new sensor module in Android

6.6 Discussion

This experiment allowed us to perceive the complexity of adding new sensors to the system, which was our main goal. Before submitting this work to a proper usability test with non-expert users, it was important to validate if developers could still contribute with their expertise to extend the set of available sensors in DETACH.

Development perspective

Although we felt that creating a sensor module in Android (task 3) was easier than adding adding a sensor template (task 1), the number of participants who completed this task after zero or one unsuccessful attempt was slightly lower. This is most likely related to the fact that adding a sensor module in Android has more specific details which led to some small typos.

Regarding the process of development, some participants suggested some improvements that will be taken in account in future work. Those improvements are:

- **Provide generic graphical interfaces** - As we already stated, while filling the Javascript file, some participants pointed out that we should provide a generic implementation of the graphical interface used to input fake data in the runtime emulator in order to allow the simulation of sensor events.
- **Encapsulate to simplify the development process** - When developing the sensor's drive in Android, one participant said that were a few lines of code that use some less known Java functions that could be abstracted in a higher-level function in order to avoid confusion.

Usability perspective

Even though our main goal was not the evaluation of the usability of the tool after the introduction of sensors in the authoring environment, whenever developers need to add new sensors they must necessarily test if the sensor is working properly. To do so, they need to compose applications and test them with the runtime emulator and therefore, we took this opportunity to take some conclusions about some of the quality of time improvements that we made in this work. In general, we believe that most of the changes contributed to a smoother development of applications. With this evaluation and feedback given, we were able to perceive and conclude the following:

- Deleting screens/sensors feels more natural using drag and drop to a trash can instead of hitting a delete button.
- Most users missed the ability to drag and drop connections to delete them (at the moment, drag and drop to delete only works with screens/sensors);
- Visual feedback validation when creating transitions proved to be useful to avoid invalid actions;
- After entering in creation mode (right-click in a screen/sensor template), at first almost every participant tried to choose the target screen by right-clicking again instead of using a left-click. Since they entered in creation mode with a right-click, they right-clicked a few times in the target screen before asking if it was working as expected or something was wrong. This behavior should feel more natural since most of the operating systems and tools have hidden right-click context menus and options are usually selected with a left-click. Future evaluations with the ability to select a target screen with either a left or right click may provide additional information about this interaction pattern.
- Some users felt uncomfortable with the current approach used to create connections with sensors. They said it makes sense when creating connections to trigger external events (Figure 62 - left) but they would prefer an alternative way to use sensors to control transitions between screens (Figure 62 - right).

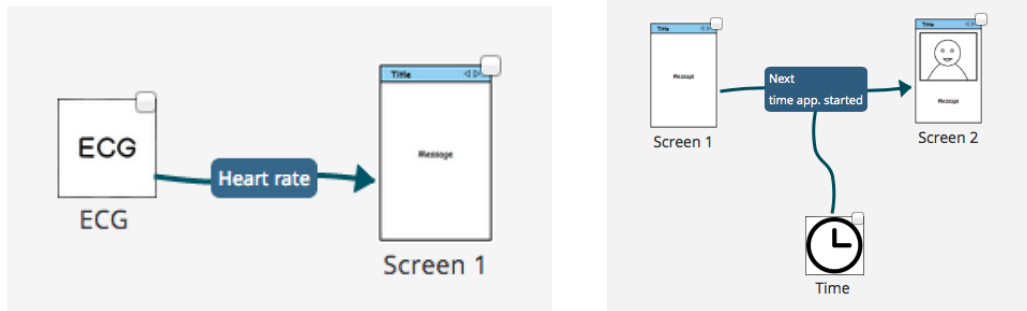


Figure 62: External trigger and a transition between screens based on a sensor trigger, respectively

A suggested alternative was the ability to drag a sensor template over a connection's label (instead of creating a connection between a sensor template and a label) in order to empower the connection with additional sensor triggers. A sketch of this suggestion is shown in Figure 63.

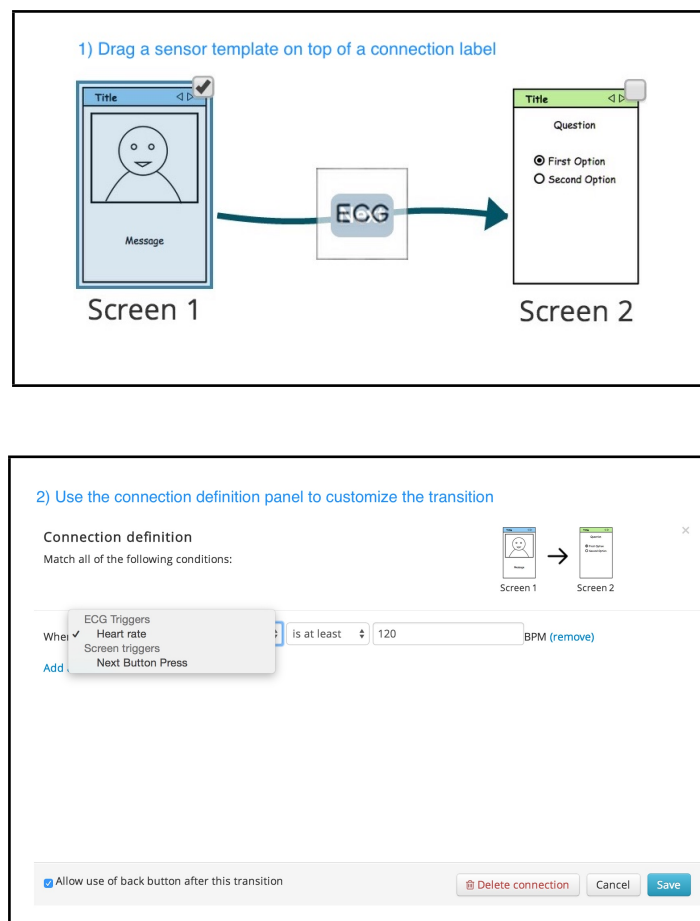


Figure 63: Sketch of an alternative way to use sensors in transitions

- A participant also suggested that when right-clicking in a screen/sensor template,

we should display a context menu with several options (e.g. create transition, edit screen/sensor, delete screen/sensor) instead of automatically entering in creation mode. A representative sketch of this behavior is shown in Figure 64.

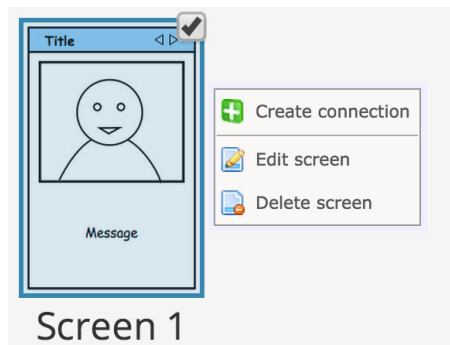


Figure 64: Context menu after right-clicking in a screen/sensor template

Chapter 7

Conclusions

Nowadays mobile devices are our daily companions which follow us anywhere. They proved to be a great opportunity and an important tool to create and improve context-aware systems which have been a central research topic in ubiquitous computing for the past few years. While some of this research has focused on developing frameworks and toolkits to support programmers in building this type of applications, there is still a lack of support for end-users to create context-aware applications. Usually, developers lack the domain knowledge that domain specialists cannot easily convey when establishing requirements for new applications. In addition, common development cycles are slow compared to how fast applications' requirements change.

With this dissertation, we proposed an ecosystem that enables non-programmers to build, deploy and run context-aware mobile applications without the need to write any programming code. Our objective was to join the efforts of different stakeholders to reach a common goal: provide better and richer applications to meet the requirements of a particular user/scenario. With that said, we want developers to provide the mechanisms and tools that enable non-experts users to create such applications. This ecosystem features two different tools:

1. DETACH, a web-authoring tool that uses a visual programming approach to let users customize the applications' look and feel. In addition, users can also use a set of sensors to create context-aware applications. This is achieved by letting users define a set of contextual events (that rely on sensor's context data) which are later triggered when the user is in presence of such contexts. Sensor events can be used to control the flow of applications (move from one screen to another) or to proactively activate applications when specific contexts are identified.
2. DETACH Mobile, an Android runtime environment that loads the developed applications and also provides the mechanisms to collect context data from sensors, which is later used to trigger events within mobile applications. This runtime en-

vironment follows an event-driven approach in order to support the dynamic nature of context-aware applications.

In Chapter 3 we analysed the state of DETACH prior to this work. This tool was already developed but it didn't support the creation of context-aware applications. In order to perceive the requirements of our work, we needed to understand the constraints regarding the integration of sensors in DETACH. With this analysis we found several limitations in both environments: runtime and authoring.

In Chapter 4 we covered the design of DETACH Mobile. This runtime environment was re-designed in order to support the dynamics of context-aware applications. In addition, we provided a detailed explanation about how developers can ensure future updates by adding new sensors to the runtime and authoring environment.

In Chapter 5 we changed our focus to the authoring environment. Feedback from DETACH's trials (prior to this work) motivated some quality of life changes in the tool's interface and tasks. Those changes aimed to enhance the usability of this tool, in order to ease the authoring process. In addition, we also presented the necessary changes regarding the integration of sensors in the composition of applications.

Finally, in Chapter 6 we conducted an evaluation to assess the complexity of adding new sensors to our system. Ensuring that developers can extend and improve our system with new components is a major requirement of our work. Without this, our authoring environment would fail since different domains use different context scenarios and sensors.

7.1 Future work

Even though our goals were achieved, there is still space to improve our system in different ways. From a perspective of usability, it's important to submit the authoring environment to a trial with non-expert users in order to perceive if the quality of life changes made in the authoring positively help users. In addition, we also need to assess if non-expert users are able to compose applications with sensors, which is the major goal of our work. There are other possible improvements/additions, such as:

- Validation of transition's conditions - at this point, when creating transitions between screens, there isn't any kind of validation about conditions that may already be in use or may be in conflict with others. This could be a major improvement to help non-experts users avoid creating transitions that are semantically incorrect.
- Use sensor's context-data in a passive way - Since we are constantly collecting context information, we could create new types of screen templates that display

such information. For example, we could create a screen template to display an ECG graph with the user's heart activity.

- Prioritization of certain events - Even though the architecture of our runtime environment is ready to support the prioritization of specific events, at the moment there isn't a way to specify events with a higher level of priority.
- Add actuators and similar devices - Integrating this kind of devices would follow similar requirements of adding sensors, with the difference that they would output (e.g. vibrate) actions.

Bibliography

- [1] Akio Sashima, Takeshi Ikeda, and Koichi Kurumatani. Toward mobile sensor fusion platform for context-aware services. In Vernon S. Somerset, editor, *Intelligent and Biosensors*. InTech, 2010.
- [2] M. Milosevic, M. T. Shrove, and E. Jovanov. Applications of smartphones for ubiquitous health monitoring and wellbeing management. In *Journal of Information Technology and Application (JITA)*, 2011.
- [3] S. Consolvo et al. Activity sensing in the wild: A field trial of ubifit garden. In *Proc. 26th Annual ACM SIGCHI Conf. Human Factors Comp. Sys.*, pages 1797–1806, 2008.
- [4] E. Miluzzo et al. Sensing meets mobile social networks: The design, implementation, and evaluation of the cenceme application. In *Proc. 6th ACM SenSys*, pages 337–50, 2008.
- [5] M. Mun et al. Peir, the personal environmental impact report, as a platform for participatory sensing systems research. In *Proc. 7th ACM MobiSys*, pages 55–68, 2009.
- [6] A. Thiagarajan et al. Vtrack: Accurate, energy-aware traffic delay estimation using mobile phones. In *Proc. 7th ACM SenSys*, Berkeley, CA, Nov. 2009.
- [7] N.D. Lane, E. Miluzzo, D. Peebles H. Lu, T. Choudhury, and A.T. Campbell. A survey of mobile phone sensing. *IEEE Comm. Magazine*, 48(9):140–150, Sept. 2010.
- [8] Irhythm. <http://www.irhythmtech.com>.
- [9] Corventis. <http://www.corventis.com/US>.
- [10] Sensium toumaz. <http://www.toumaz.com>.
- [11] B.Falchuk, A.Misra, and S.Loeb. Server-assisted context-dependent pervasive wellness monitoring. In *Proc. ICST International Workshop on Wireless Pervasive Healthcare*, London, 2009.

- [12] Korel B T. and Koo SGM. Addressing context awareness techniques in body sensor networks. In *21st International Conference on Advanced Information Networking and Applications Workshops*, pages 798–803, Niagara Falls, Ontario, 2007.
- [13] O. Etzion, Y. Magid, E. Rabinovich, I. Skarbovsky, and N. Zolotorevsky. Context aware computing and its utilization in event-based systems. In *DEBS*, 2010.
- [14] Context delivery architecture: Putting soa in context. <http://www.gartner.com/DisplayDocument?id=535313>.
- [15] K. Heverst, N. Davies, K. Mitchell, A. Friday, and C. Efstratiou. Developing a context-aware electronic tourist guide: some issues and experiences. In *Proc. CHI*, The Hague, Netherlands, 2000.
- [16] V. Bychkovskiy, S. Megerian, D. Estrin, and M. Potkonjak. A collaborative approach to in-place sensor calibration. In *Proceedings of IPSN'03*, 2003.
- [17] E. Elnahrawy and B. Nath. Cleaning and querying noisy sensors. In *Proceedings of ACM WSNA '03*, 2003.
- [18] G. Adomavicius and A Tuzhilin. Personalisation technologies: A process-oriented perspective. *Communications of the ACM*, 48(10):81–90, 2005.
- [19] B. Y. Lim, A. K. Dey, and D Avrahami. Why and why not explanations improve the intelligibility of context aware intelligent systems. In *Proceedings of CHI 2009-Studying Intelligent Systems*, pages 32–41, Boston, MA, USA, 2009.
- [20] Michael Beigl, Albert Krohn, Tobias Zimmer, and Christian Decker. Typical sensors needed in ubiquitous and pervasive computing. In *Proceedings of the First International Workshop On Networked Sensing Systems (ISSN '04)*, pages 153–158, 2004.
- [21] T. Choudhury et al. The mobile sensing platform: An embedded system for activity recognition. *IEEE Pervasive Comp*, 2(7):32–41, 2008.
- [22] M.-Z. Poh et al. Heartphones: Sensor earphones and mobile application for non-obtrusive health monitoring. In *IEEE Int'l. Symp. Wearable Comp.*, pages 153–54, 2009.
- [23] A. T. Campbell et al. Neurophone: Brain-mobile phone interface using a wireless eeg headset. In *Proc. 2nd ACM SIGCOMM Wksp. Networking, Sys., and Apps. on Mobile Handhelds*, New Delhi, India.
- [24] Y. Engel and O. Etzion. Towards proactive event-driven computing. In *Proceedings of the 5th ACM international conference on Distributed event- based system*, pages 125–136, 2011.

- [25] A. Robinson, J. Levis, and G. Bennett. Informs news: Informs to officially join analytics movement. *INFORMS, OR/MS Today*, 37(5), 2010.
- [26] D.C. Luckham. *The power of events*. Addison-Wesley, 2002.
- [27] O. Etzion and P. Niblett. *Event Processing in Action*. Manning Publications, 2010.
- [28] D. Husemann, C. Narayanaswami, and M. Nidd. Personal mobile hub. In *ISWC '04: Proceedings of the Eighth International Symposium on Wearable Computers (ISWC'04)*, pages 85–91, Washington, DC, USA, 2004. IEEE Computer Society.
- [29] E. Lubrin, E. Lawrence, and K. F. Navarro. Motecare: an adaptive smart ban health monitoring system. In *BioMed'06: Proceedings of the 24th IASTED international conference on Biomedical engineering*, pages 60–67, Anaheim, CA, USA, 2006. ACTA Press.
- [30] M. Blount et al. Remote health-care monitoring using personal care connect. *IBM Systems Journal*, 46(1):95–113, 2007.
- [31] Mohamed I., Misra A., Ebling M., and Jerome W. Contextaware and personalized event filtering for low-overhead continuous remote health monitoring. In *International Symposium on a World of Wireless, Mobile and Multimedia Networks*, 2008.
- [32] Margaret Burnett. What is end-user software engineering and why does it matter? In *Proceedings of the 2nd International Symposium on End- User Development (IS-EUD '09)*, pages 15–18, 2009.
- [33] John Maloney, Mitchel Resnick, Natalie Rusk, Brian Silverman, and Evelyn Eastmond. The scratch programming language and environment. *ACM Transactions on Computing Education (TOCE)*, 10(4):1–15, November 2010.
- [34] Wolfgang Slany. Catroid: a mobile visual programming system for children. In *Proceedings of the 11th International Conference on Interaction Design and Children*, Bremen, Germany, June 2012.
- [35] Robert Sheehan, Ducksan Cho, and Joon Ha Park. Improving on a physics-based programming system for children. In *Proceedings of the 11th International Conference on Interaction Design and Children (IDC '12)*, pages 312–315, New York, NY, USA, 2012.
- [36] A. Dey, T. Sohn, S. Streng, and J Kodama. icap: Interactive prototyping of context-aware applications. *Pervasive Computing*, 3968:254–271, 2006.
- [37] Y. Li, J.I. Hong, and J.A Landay. Topiary: a tool for prototyping location-enhanced applications. In *Proceedings of the 17th Annual ACM symposium on User Interface Software and Technology*, pages 217–226, 2004.

- [38] A.K. et al Dey. a cappella: Programming by demonstration of context-aware applications. In *CHI*, pages 33–40, 2004.
- [39] Valentim Realinho, Teresa Romão, Fernando Birra, and A. Eduardo Dias. Rapid development of mobile context-aware applications with ivo. In *Proceedings of the 8th International Conference on Advances in Computer Entertainment Technology*, ACE '11, pages 91:1–91:2, New York, NY, USA, 2011. ACM.
- [40] Valentim Realinho, Teresa Romão, Fernando Birra, and A. Eduardo Dias. Building mobile context-aware applications for leisure and entertainment. In *Proceedings of the 8th International Conference on Advances in Computer Entertainment Technology*, ACE '11, pages 29:1–29:8, New York, NY, USA, 2011. ACM.
- [41] Bjorn Hartmann, Leith Abdulla, Manas Mittal, and Scott R. Klemmer. Authoring sensor-based interactions by demonstration with direct manipulation and pattern recognition. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, San Jose, California, USA, 2007.
- [42] Filipe Fernandes, Luís Duarte, and Luís Carriço. Flow specification patterns of end-user programmers: Lessons learnt from a health mobile application authoring environment. In *Human-Computer Interaction - INTERACT 2013, 14th IFIP TC13 International Conference*, Cape Town, South Africa, 2013.
- [43] Filipe Fernandes, Luís Duarte, and Luís Carriço. Detach, criação de aplicações móveis para todos. In *5ª Conferência Nacional em Interação Pessoa-Máquina (Interacção'13)*, Portugal, 2013.
- [44] Filipe Fernandes. Detach: Design tool for smartphone application composition. Master's thesis, Universidade de Lisboa - Faculdade de Ciências, 2013.
- [45] Dartmouth College. Mobile sensing group. <http://sensorlab.cs.dartmouth.edu>.
- [46] Philip Robinson and Michael Beigl. Trust context spaces: An infrastructure for pervasive security in context-aware environments. In *Conference on Security in Pervasive Computing*, 2003.
- [47] A.K. Dey, D. Salber, and G.D Abowd. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human Computer Interaction*, 16(2):97–166, 2001.

