Development of a medication compliance system on mobile devices

Auteur : Servais, Fabrice
Promoteur(s) : Leduc, Guy
Faculté : Faculté des Sciences appliquées
Diplôme : Master en ingénieur civil en informatique, à finalité approfondie
Année académique : 2015-2016
URI/URL : http://hdl.handle.net/2268.2/1317

MASTER THESIS

In collaboration with A7 Software

Development of a medication compliance system on mobile devices

---

Graduation Studies conducted for obtaining the Master's degree in Computer Science and Engineering by

Fabrice SERVAIS

University of Liège
Faculty of Applied Sciences
Academic year 2015 - 2016

Promoter:
Guy LEDUC

Industrial sponsor:
Vincent KEUNEN

Jury:
Bernard BOIGELOT
Laurent MATHY

June 1st, 2016

Abstract

Development of a medication compliance system on mobile devices

Fabrice SERVAIS

Master in computer science and engineering
University of Liège - Academic year 2015-2016

Promoter: Pr. Guy Leduc

Patients may sometimes fail to comply to their medication whether it was from forgetting to take the medicine, from taking it at the wrong time or even from taking too much of it. This is called *medication noncompliance* or *medication non-adherence*. Indeed, depending on the diseases, those medications can be very complex. This noncompliance has two major consequences: it first affects the patient's health more or less seriously and it also costs billions to governments in health care.

On the other side, *Andaman7* is a Synchronized Health Record that connects together patients and doctors. It is a mobile platform allowing the patient to consult and share his health record.

One key solution to medication noncompliance is to establish a better communication between the patient and the physician. It thus seems natural to use *Andaman7* as a tool to enhance the non-adherence of the patients. This is the goal of this Master thesis: develop a medication compliance system integrated into *Andaman7*.

Using this system, the patient is able to add a medication to his medical record and also activate the reminders for this medication, so that the system will trigger a notification when it is time to take the prescribed medicine.

This manuscript will first detail all the medication compliance context before giving the details of the system, all the features that should be in it, in the scope of this work, how it was designed and integrated into *Andaman7*, implemented, as well as some guidelines for further development, from testing strategies to others features to enhance the system.

Another topic will be addressed in this Master thesis, which is the integration of a medicine database in *Andaman7*. The motivation is to allow the physician or patient to be able to select a drug from a complete list of medicines. To do so, this list has to be populated with the data from an external source. This work will show how a list of drugs can be retrieved from the CBIP/BCFI database.

Acknowledgement

I am grateful that I had the opportunity to work on such a wonderful project and I would like to thank the people who surrounded me during this year.

First, I would like to express my gratitude to Vincent Keunen, CEO of A7 Software, for his guidance. Also, I would like to sincerely thank Sébastien Hannay, Software Engineer at A7 Software, for his supervision and his precious advice during the whole project, as well as Antoine Smolders, CTO at A7 Software, also for his valuable counsels. Of course, I warmly thank the rest of A7 Software's staff that kindly integrated me into their team and work life.

Finally, I am also grateful to Pr. Guy Leduc, my promoter, for being always available when I needed some recommendations.

Contents

List of Figures

List of Tables

Chapter 1

Andaman 7 - Synchronized Health Record

This introductory chapter will present Andaman 7 and its story, from the context in which it was created, to what is its concept and what problems it answers.

1.1 Problem definition

After years spent in the hospital environment due to medical reasons for him and his family, Vincent KEUNEN, CEO of *A7 Software*, found out that the European medical system was composed of excellent physicians but there was a serious lack in the sharing of information, from the management of the medical data to the low patient involvement in its own monitoring. Too many times were the paper files not available or the wrong dossier was taken in a rendezvous. Even finding information in the record was time-consuming, which definitely should not be.

1.2 The solution: Andaman7

FIGURE 1.1: Andaman 7 logo

Andaman 7 is a **SHR** (Synchronized Health Record) for both patients (as well as their family and friends) and healthcare professionals (physicians, caregivers, and so on). It is a secure platform connecting all those actors involved in a patient health. It actually aims at bringing the patient in the loop, allowing a better communication between patients and doctors: this is **patient empowerment**. The application brings together two kinds of system as one:

- a **EHR** (Electronic Health Record) for the physicians

- and a **PHR** (Personal Health Record) for the patient.

Andaman 7 is a free and easy solution to communicate medical information amongst patients, amongst practitioners and between patients and practitioners. This sharing is done using rules added by the user, specifying the visibility of each record, as shown in FIGURE 1.2.

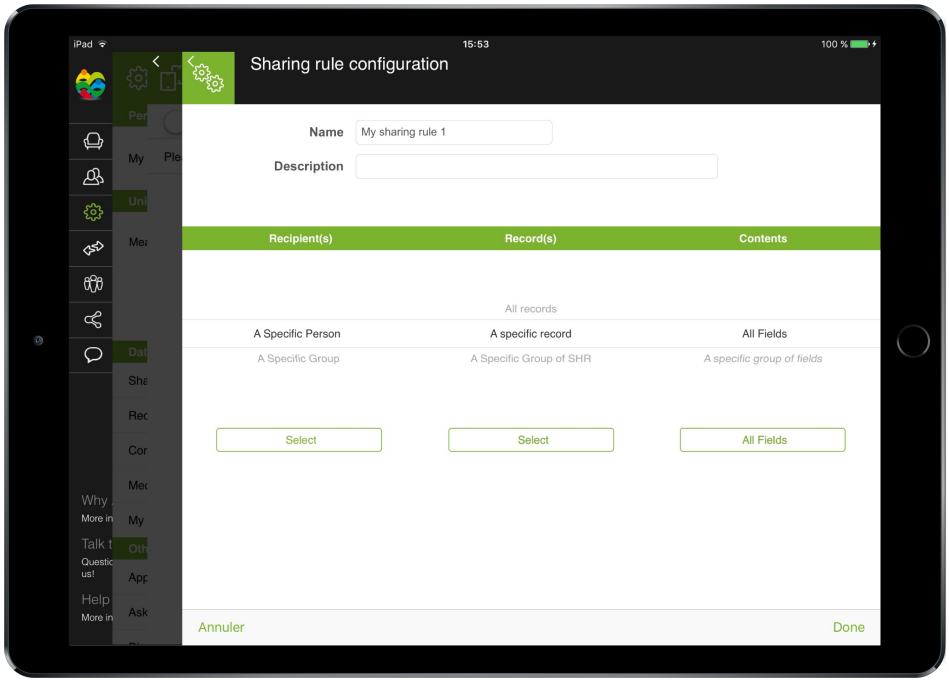


FIGURE 1.2: Andaman 7 sharing rules

This sharing is also made secure using a Peer-To-Peer communication amongst end devices, meaning that nothing is stored permanently in the cloud, rather an external server is still used for the communication as an intermediate between end devices during the transit that is encrypted. An attack on the server could only retrieve an encrypted version of the data that was in transit at time T.

> After having introduced and put Andaman 7 into a context, this paper will explain the two main parts of the thesis. The first one will describe the medication compliance system to be integrated into Andaman 7, while the second part will take a look at how to integrate a drug database into the application.
>
> The next chapter will introduce the medication noncompliance subject before jumping into the application and portray the context around it.

Part I

Medication reminder system

Chapter 2

Medication noncompliance context

In this chapter, the noncompliance regarding to a medication will be studied, starting by defining what it is, followed by the consequences that it can induce as well as the factors that can lead to a lack of adherence, what are the ways to measure it and how it would be possible to improve the situation.

2.1 Medication non-compliance

The *noncompliance* is defined as "Failure or refusal to comply." [4]. One can also refer it as *adherence* which can be defined, in the context of the medication, according to [3]:

> Adherence to, or compliance with, a medication regimen is generally defined as the extent to which a person takes medications as prescribed by their health care providers.

It obviously goes in pair with *persistence*, "the ability of a person to continue taking medications for the intended course of therapy." [18], which emphasizes on the patient capability over a long-term.

This non-adherence may take multiple forms. Even though it often results in a lack of taking the prescribed medicine, the noncompliance varies in the behaviour of the patients, that includes [2]:

- **Failure to take medication**: either missing a dose or multiple doses, prematurely discontinuing the medication, wrongly taking the dose, e.g. with prohibited food, liquid or with another medication, improperly using the medication administration device (an inhaler for example), and so on,
- **Too much medication**: when taking more that is was described in the perspective that the more medicine are ingested, the better their benefits, which is definitely not a good hypothesis.
- **Taking drugs for the wrong reason**: a drug may be ingested for a wrong reason, coming from a confusion of the purpose of that drug,
- **Wrong timing**: specially when the regimen is complex, the drugs may not be taken at the appropriate time.

Amongst other failure, one can also mention a failure to refill a prescription, taking outdated or damaged medication, or even storing them improperly.

FIGURE 2.1 shows the top 10 reasons for non-adherence, according to [5]. As can be seen, affordability, lack of knowledge, side-effects and forgetfulness are the main factors. In the scope of this paper, affordability and side-effects are not causes subject to discussion. However, lack of knowledge and specially forgetfulness are undoubtedly relevant. How a SHR can affect those factors will be discussed later in this paper.

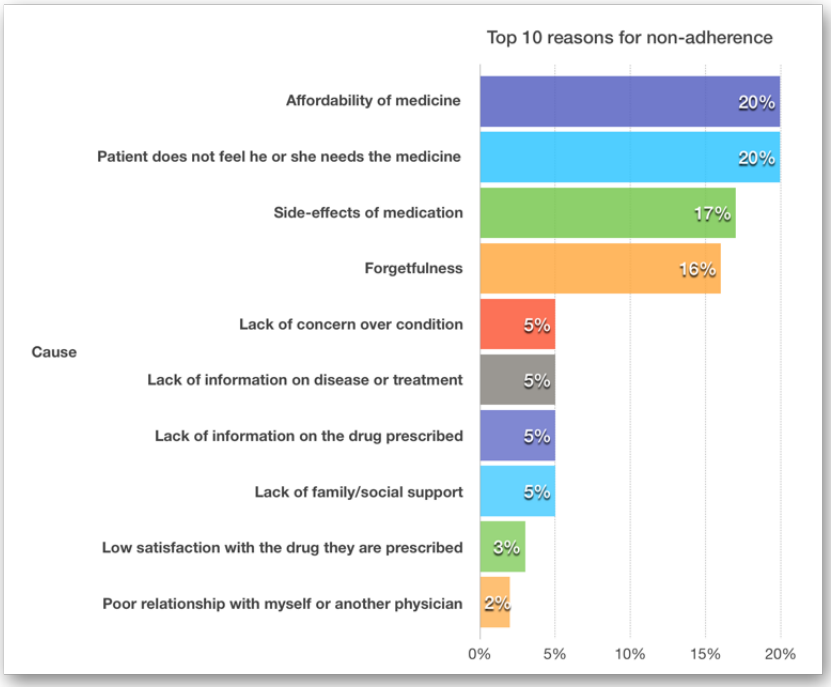


FIGURE 2.1: Top 10 reasons for non-adherence [5]

2.2 Consequences of noncompliance

Obviously, it has consequences on the health of the patients, from a disease harder to detect and to fight to much worst and serious situations. In the U.S., approximately 125.000 people with treatable ailments die each year from non compliance [2]. Another approach can be that when the patient meets the physician again and that he doesn't recognize the noncompliance, he can increase the dosage, which increase the risk of side effects, and so on. This can be represented as a downward escalating spiral.

Medication non-adherence is one of the biggest costs of health care. With 50% of the patients that are noncompliant [20], the cost to EU governments is estimated at €125 billion annually [21], and could even rise up to $ 289 billion in the U.S. [7]. This number may be even bigger by taking into account the indirect costs. Those are due to an increase usage of medical

resources, such as physician visits [18]. Also from the pharmaceutical industries point of view, there is a lack of income since people are buying less drugs.

2.3 Factors affecting medication adherence

In this section, the causes leading to noncompliance will be overviewed. Firstly, it is to note that a patient might be lacking of adherence intentionally or unintentionally.

According to the W.H.O. (World Health Organization), there are five main sets of factors related of adherence. There are listed and given some examples in [18].

- **Social and economic**: limited language proficiency, low health literacy, medication cost, lack of health care insurance, unstable living conditions,...
- **Health care system**: lack of continuity care, high drug costs, weak capacity of the system to educate patients and provide follow-up, not adapted patient information materials,...
- **Condition-related**: Lack of symptoms, depression, mental retardation,...
- **Therapy-related**: Complexity of the regimen, requiring mastery of techniques (injections,...), duration, lack of immediate benefit, unpleasant side effects, requires behavioral changes,...
- **Patient-related**: physical factors (visual impairments, hearing impairments, swallowing problems), lack of knowledge about the disease, understanding the reason of the medication, motivation, fear of dependence,...

Some of the factors are summarized in the FIGURE 2.2, taken from [3].

2.4 Measuring adherence

There are different ways of measuring the adherence of a patient. However, these are proxy measures that have limitations [9]:

- Patient self-report: the measures may be unreliable cause it depends on the memory of the patient. It could be inaccurate too.
- Pill count: Checking the number of remaining pills. Unreliable if the patient switches medicines between bottles, or simply get rid of the pills before visiting. It also only gives an information of the number of intake, not the timing [3].
- Biological monitoring: e.g., sampling blood, urine and so on. These methods are invasive and impractical to really measure adherence, since the measures should be done before and after the intake.

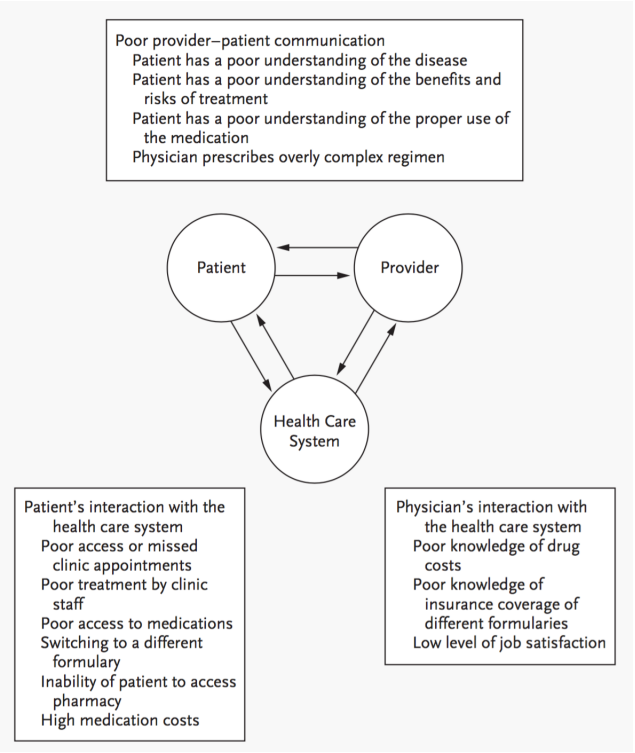


FIGURE 2.2: "The interactions among the patient, health care provider, and health care system depicted are those that can have a negative effect on the patient's ability to follow a medication regimen." [3]

- Refil rates: Those methods cannot determine if the patient did take his medication.

The self-report seems the most suitable by being less invasive, easy easy to settle and its accuracy could be enhances using tools, such as a medication reminder keeping tracks of the intakes when they should occur.

2.5 Enhance medication adherence

This section will examine the different solution that could be set up to improve the adherence of a patient. It will first highlight the role of the physician in the process and then review the potential impact of other actors or tools.

2.5.1 Role of the physician

One of the most crucial factor is the lack of knowledge of the patient regarding to his medication. It is thus clear that it should be improved, and it can be not so hard to put in place, even though the effectiveness of all those solutions should be accurately measured.

The simplest solution that can be provided is for the physician to make it clear to the patients that they should themselves perceive the medication as being important, by providing clear information about the causes, the symptoms (visible or not), the effect of the drugs on the disease, and so on.

The point is to inform the patient to make sure he is aware of the usefulness of his medication. In that spirit, the paper [2] mentions the social cognitive theory applied in the context of medication noncompliance as being able to assess the patient's perception of "vulnerability, severity, treatment effectiveness and costs" to be able to conceive solutions (e.g. messages) adapted to the patient to make him more compliant.

In the same way of thinking, one could add to the solutions the increase of the communication between the patient and his doctor by, for instance, establishing more appointments between them. The patient could also be taught to self-monitor his medication, that could give him a different vision of his medication by being able to situate himself in the medication using some monitored measures.

Another aspect that could potentially be improved is the regimen. Indeed, it is important to tailor it according to the patient's individual schedule so that it is not cumbersome to take the medication, but still keeping the medicine effective.

One could resume those solutions in one effective concept: *patient-centered approach*. It brings the patient in the center of his therapy, where his satisfaction and his needs are taken into account. The goal is to determine what really matters for the patient, specially what kind of relation [8]. Still according to [2], the outline of such a model are:

- "Accept where the patient is."
- "Accept what you do not know."
- "Acknowledge that the patient has the answers."
- "Build self-efficacy."
- "Set realistic expectation for self and patient. "
- "Share responsibility."

2.5.2 Role of other actors

The role of the pharmacist should also be highlighted in this problem. Indeed, he has a much frequent contact with the patient, more the physician. From that perspective, the involvement in the treatment is also direct and could be taken advantage from to change or not the regimen.

Of course, the patient plays the most important role in his medication since he is his own responsible for taking or not his medicine. However, all the solutions mentioned above are mainly addressing the awareness and thus motivation of the patient.

In the case of unintentional non-adherence, some tools exists to help the patient not to forget the intakes, including traditional reminders such as weekly pill boxes, and so on. Even though those methods are proven to be effective, it remains that they are "passive methods that could be cumbersome for complex regimen" [9]. To counterbalance these drawbacks, the usage of proactive systems can be privileged.

2.5.3 Medication compliance mobile application

Nowadays, when someone needs a tool to help him to achieve something or to assist him in daily tasks, it becomes more and more natural for him to use his smartphone via built-in applications, or even search on the store to find the cheapest application that satisfies all his needs. It can give maximal output for a little cost.

This scenario is perfectly applicable to someone who needs helps everyday with his complex regimen for instance. The smartphone becomes nearly constantly accessible in such a way that it can now be possible to have a pill reminder in our pocket. Besides, the capabilities are not limited to simply remind to take a medicine, even though it is the main usage, for the same cost, it could be extended to other features, such as remind to log the data about the intakes for further analysis, refill prescriptions, and so on.

2.5.4 Synchronized Health Record mobile application

A Synchronized Health Record application allows primarily the patient to have a better access to his medical record by being able to check easily his data from anywhere on his smartphone or tablet. The increase of the accessibility via the mobility and centralization is the key point of such an application. In addition of an easy usage, it can really improve the communication between a patient and his doctor whether in his office or in a hospital. It also extends to all medical jobs where the data about a patient is important.

Both ideas of a medication compliance application and a SHR application are not only compatible, they complete each other. Where the medication application lacks of information, other than the medication itself, about the patient, whereas the SHR is not primarily meant to be an everyday assistant to the patient. Both merged into a single application will create a central point for patients to follow their health records, including medications, analyse and have a better view of their own health, as well as for doctors or medical professionals to better understand the behaviour the patients regarding to their medications, and takes adequate actions if need be. The clear goal to achieve is to reconcile the patient with his medication. [6]

> The solutions mentioned in this section aim at being a (non-exhaustive) list of what can be done in general, with an emphasis on the improvement of the **physician-patient communication**,

as well as the enhancement that could be brought by a *Synchronized Health Record application with an integrated medication aspect*.

The integration of a medication compliance dimension into Andaman 7 will begin in the next part of the report by presenting the requirements of the project and see what is the scope of the work that has been done.

Chapter 3

Existing medication compliance applications

Medication reminder applications have already been existing for a while, the idea of such a reminder is not groundbreaking, which doesn't make it less useful. In this chapter, some applications will be rapidly described, first the ones I tested, and then from a complete and concise comparison available on *pharmacist.com* [19].

This process of analysing the existing solutions aims at reaching 2 main goals:

- Spot the key features for a medication reminder application, either necessary or not,
- Going through different UI (User Interfaces) and UX (User eXperiences) to have a reflection for the further development.

3.1 Applications tested

The medication compliance applications that have been tested on two iOS devices: iPhone and iPad. For each device, I noted the name, the price as well as the key features of the applications. They are shown in TABLE 3.1 for the iPad applications and in TABLE 3.2 for the iPhone applications.

In addition, the website *pharmacist.com* [19] has also made a comparison that is worth reading. Their observations has been re-transcribed in TABLE A.1 in APPENDIX A.2.

3.2 Analysis

This section will extract this interesting features from the applications that were described above. Some of them are directly or indirectly already implemented in Andaman 7, there will be listed in first place. After that, the other key features that could add an extra to the medication reminder system will be listed.

Application	**Cost**	**Features**
MedMemo (FIGURE A.1)	Free	• Has a drug database, • Able to select different dosage unit (e.g. pill, spoon, caps,...), • Summary in a calendar by day containing the drug, the dosage, the time and of if it has been taken, • Weight tracking system.
Mon agenda de Médication (FIGURE A.2)	Free	• Password protected, • Possibility to add notes, • History.
Pill reminder (FIGURE A.3)	Free	• Help bubbles at the first launch, • Summary by day or by month. • Refill reminder, • Expiration date, • Report on the intakes, • Manually add an intake, • Multiple users.

TABLE 3.1: Comparison of different medication compliance applications on iPad

Application	**Cost**	**Features**
MyMeds (FIGURE A.4)	Free (and Premium)	• An account is needed, • Possibility to add a pharmacy, a medication, a doctor and a drug allergy, • Possibility to upload a picture of the medicine, • System where the user can earn points when the medication is correctly taken, • Available on Android and via a Web platform.
MediSafe (FIGURE A.5)	Free	• Dashboard to see the medicines to take per day, • Health measure tracker (e.g. blood pressure, cholesterol, glycaemia,...), • Drug database and special scheduling when a contraceptive pill is selected, • Possibility to select amongst some icons the appearance of the pill, • Preselection of the dosage depending on the medicine selected, • Report tab to see the medicine correctly taken or not, • Prescription reminder.

TABLE 3.2: Comparison of different medication compliance applications on iPhone

3.2.1 On Andaman 7

The first feature is the ability to **share the medication information**. This is indeed inherent to concept of a Synchronized Health Record, which comes with the possibility to share the medical information with others. Thus, as soon as the medication is integrated into the system, it is possible to share it with the user's family, doctor, friends, and so on. An extension of this is that the medication data are thus written into the patient **health record** directly.

Andaman7 requires an account to use the application. A consequence of this feature is that the data are **password-protected** since they are linked to the account.

3.2.2 Other key features

Medication encoding

One of the features that can enhance the user experience and increase the feeling of using a professional application is having a drug database. This will come with the possibility, when adding a medication, to choose a medicine from a list, and thus link the drug information to the medication. This can be useful from simple extras, e.g. pre-filling the fields according to the drugs, suggesting a schedule,..., to more complex management, e.g. determine automatically if the patient takes inadequate or dangerous dosages, determine automatically if the combination of drugs (or active principles) can lead to undesirable effects, and so on.

Other features related to the encoding of the medication are, for instance, the ability to select the dosage unit corresponding to the prescription, the form and color of the pill to identify it instantaneously, or even the ability to take a picture of the drug.

Regarding to the schedule of the medication, certain medicines require a particular schedule, as the contraceptive pills for instance. This can be a great feature to be able to deal with more complex medications.

Alerts

In addition to receiving a notification when it is time to take the medication, notifying the system if the medicine has been taken, delayed or not taken, either via the native notification or the application, is a great improvement. It can make the tracking of the medication possible and make the patient able to measure how compliant he is.

Another kind of alert that could be added is a refill reminder. The user could add the number of pills left so that the application can also compute how long the patient can still take his medicine and, at some time, sends a notification when it is time to refill.

This chapter listed the different features that are already available in other medication compliance applications to have an overview of what is already existing.

The following chapter will describe the requirements that are needed for the medication system in Andaman 7.

Chapter 4

Requirements

The previous chapters gave a global introduction to the work, we have particularly seen the domain and the context in which it will fit into. The focus of this chapter will be to describe the requirements of the system to be implemented. This will guide the development by listing all the features to be implemented.

4.1 Functional requirements

The first requirements to be specified are the functional requirements. They are listed below.

4.1.1 Medication

- By accessing its medical record, a patient is able to
  - **add**,
  - **view**,
  - **modify**, or
  - **delete**

  a **medication**.
- A medication is composed of the following fields:
  - **Medicine**,
  - **Dosage**: how many units of medicine should by taken by intake,
  - **Unit**: the unit of medicine (e.g. pill, millilitre, spoon,...),
  - **Starting date** of the medication,
  - **End date** of the medication,
  - Which **days frequency** the medicine should be taken. The patient can choose between:
    - *Every day*,
    - Selecting *days of the week*,
    - A days *interval* (i.e. "every ... day(s)")

  - How many **times by day** the medicine should be taken. The patient can choose between:
    * How many *times per day*,
    * An hour *interval* (i.e. "every ... hour(s)")
  - Activating or not the iOS push **notifications**,
  - **Note**: possible to write a comment about the medication.

- The medicine is chosen from a **drug list**.
- If the patient chose to specify an hour interval, he is also able to select the **starting time** of the intakes.
- If the patient chose to specify the number of intakes per day, the patient is also able to select at **which hours** (hours-minutes format) those intakes should occur, even if the system pre-selects them.
- **Adding a medication** corresponds to fulfilling the field mentioned above. The mandatory fields have to be filled, if not, they will be highlighted by the system.
- A **modification of the medication** allows to edit any of the field related to the medication.
- The patient can **cancel** the process of adding or editing a medication.
- If the patient asks to **delete a medication**, the system should display a message to make the user confirms the action.

4.1.2 Notifications

If the user has enabled the push notifications:

- The system triggers a **notification at the time of the intake** to remind the user of his medication.
- Upon receiving the notification, the patient has the choice between 2 actions:
  - Mark the intake as **taken**, or
  - Mark the intake as **not taken**.

  The choice is then saved.

4.1.3 Medicine management

- The user can:
  - **add**,
  - **view**,
  - **edit**, or
  - **remove**

  a **medicine**. The user can do those actions from its medical record.

- A medicine contains the following information:
  - **Dosage**: Optional dosage of the drug, for instance "500mg" in "Dafalgan 500mg",
  - **Form**: form in which the medicine is kept, e.g. pill, liquid,...,

4.2 Platform requirements

As already mentioned in the introduction, the medication compliance system has to be integrated into the Andaman7 application, i.e. will be used on mobile devices.

For the scope of the thesis, the system has to be able to run on an iPad with an iOS version of at least 8.0. Developing for iOS means that the system will be implemented in Objective-C or Swift. As far as Andaman7 is concerned, it is currently implemented in Objective-C, however, it is planned to upgrade steps by steps the application to use more and more Swift. As a prevision for future development, the medication compliance system will thus be implemented in Swift, as well as in Objective-C when modifying existing parts will be needed.

> The requirements has been enumerated in the chapter, giving an overview of the expected capabilities. The next chapter will illustrate them by providing the corresponding use cases.

Chapter 5

Use cases

This chapter will describe the different use cases taken into account in this project. It will depict the general usage that can be done with the system by the actors. Also be advised that the uses cases have been a bit simplified to not have a excessively long chapter.

5.1 Actors

The actors participating in those use cases are :

- Patient
- Practitioner
- User: either a patient or a practitioner

5.2 Medication compliance system: Use cases

5.2.1 See current medications

See current medications		
Goal	:	Display all the medications in which the user is currently or was involved.
Precondition	:	• The User has opened the application.

Detailed interaction	
1. The User touches the button “My files” and touches the EHR of the concerned patient.	
	2. The application displays a summary of the recorded medications in the section “Medication”.
3. The User touches one of the current medications to display the details of all medications, not only the one he touches.	
	4. The list of all medications is displayed under "Current medications".

Alternative
If the User is a Physician, he will have to first select a Patient to see his medication.

5.2.2 See details of a current medication

See details of a current medication		
Goal	:	The goal is to display the details of a medication.
Precondition	:	• The User has opened the application. • Use case: *“See current medications”*

Detailed interaction	
1. The User taps on the medication about which he wants the details.	
	2. The System shows a window with all the information about the medication.
3. The User can close the window by pushing the cancel button.	

5.2.3 Add a medication

Add a medication		
Goal	:	The User wants to add a medication to the application, with or without notification.
Precondition	:	• *"See current medications"* use case

Detailed interaction	
1. The User touches the button to add a medication.	
	2. The application displays the fields to fill: – Medicine: to be selected from a list – Dosage – Unit – Start date – End date – Days * Every day * Select days of the week * Interval of time (every x days) – Frequency * Times per day * Interval of time (every x hours)
3. The User touches the "next" button to continue.	
	4. The application checks if the mandatory fields are filled. If no, it stays in the panel and shows in red the mandatory fields. If yes, it loads a list of the intakes that can occur during a day with their associated time.
5. If the User touches one of the reminders, he has the ability to change the time of the selected reminder. The User touches the "next" button to continue.	

	6. The application loads a third panel with the following fields: – Switch to activate or not the push notifications – Additional notes
7. The User pushes the "done" button to validate and save the medication.	
	8. The system closes the panel.

5.2.4 Change a medication

Change a medication		
Goal	:	When the User wants to change a parameter of a medication, he can do it with the following steps.
Precondition	:	• Use case: *"See details of a current medication"*

Detailed interaction	
1. The User touches the value of the field he wants to change and enters the new value, pushing on the "next" button to change panels and "done" to validate.	
	2. The system behaves the same way as if we want to add a medication.

5.2.5 Remove a medication

Remove a medication		
Goal	:	When the User has done with a medication, he can remove it from the system.
Precondition	:	• Use case: *"See details of a current medication"*

Detailed interaction	
1. The User pushes the "Delete" button.	
	2. The system displays a notification to ask the User for a confirmation of his action.
3. If the User wants to cancel his action, he touches the "Cancel" button.	
	4. The notification disappears and he comes back to the panel displaying the information about the medication.
3bis. If the User wants to remove his action, he touches the "Confirm" button.	
	5. The notification disappears as well as the panel displaying the information about the medication. The medication is deleted from the system.

5.2.6 Receive notification

Receive notification		
Goal	:	Upon receipt of the notification, the User can mark it as *taken* or *not taken*.
Precondition	:	• The User has registered a medication and activated the push notifications on it. • The User has accepted on its device that Andaman7 can trigger notifications.

Detailed interaction	
	1. At the time of the medication reminder (provided when the medication was registered), iOS triggers a push notification.
2. The User sees the notification and touches the "Taken" button.	
	3. The application marks the intake as *taken* and the notification disappears.
2bis. The User sees the notification and touches the "Not taken" button and the notification disappears.	
	4. The application marks the intake as *not taken*.
2ter. The User sees the notification and ignores it.	
	5. The notification disappears.

5.2.7 See medicine list

See medicine list		
Goal	:	The User can consult the list of already registered medicines.
Precondition	:	• The User has opened the application.

Detailed interaction	
1. The User touches the button “My files” and touches the EHR of the concerned patient.	
	2. The application displays a summary of the recorded medicines in the section "Drugs".
3. The User touches one of the elements in the section "Drugs" to display the details the subsections.	
	4. The System displays the medicines recorded in the subsection "Drugs".

5.2.8 See medicine information

See medicine information		
Goal	:	The User can consult the details concerning a already registered medicine.
Precondition	:	• Use case: *"See medicine list”*

Detailed interaction	
1. The User touches the name of the medicine that he wants more details on.	
	2. The application displays the information for the following fields: – Name – Dosage – Form – Comment

5.2.9 Add a medicine

Add a medicine		
Goal	:	If the medicine isn't in the database, the User can add one.
Precondition	:	• Use case: *"See medicine list"*

Detailed interaction	
1. The User touches the "Add item" button in the subsection "Drugs".	
	2. The application displays a panel with the fields to be filled: – Name – Dosage – Form – Comment
3. The User touches the "Back" button or the "Add" button to save the medicines.	
	4. The application closes the panel and displays the new medicine in the list

5.2.10 Change a medicine

Change a medicine		
Goal	:	The User can modify the information of a medicine.
Precondition	:	• Use case: *"See medicine information"*

Detailed interaction	
1. The User can change the fields displayed. When finished, he can push the "Back" button.	2. The application saves the modifications and closes the panel.

5.3 Summary diagrams

The use cases regarding the medications and the medicines has been summarized in diagrams, respectively in FIGURE 5.1 and FIGURE 5.2.

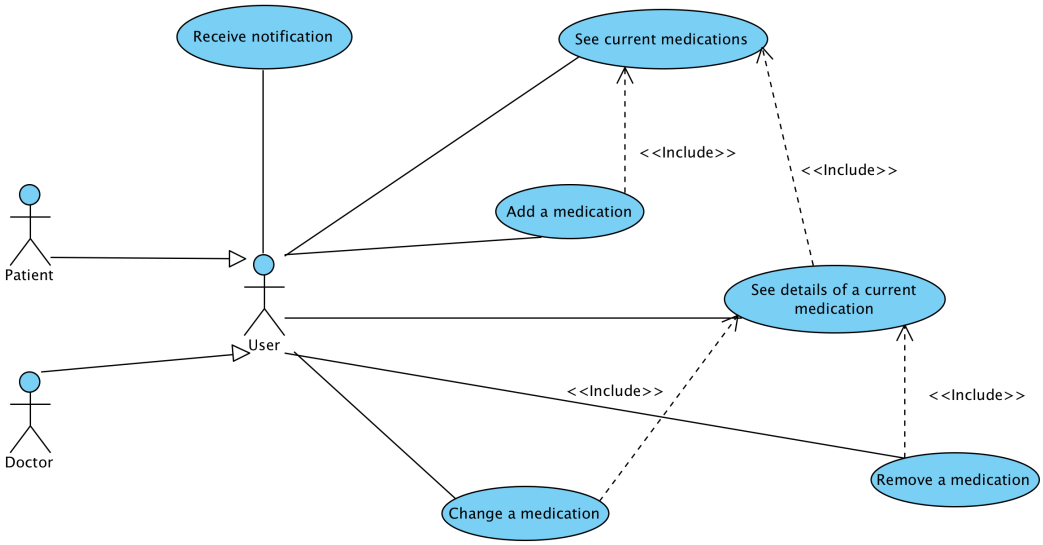


FIGURE 5.1: Medication-related use case diagram

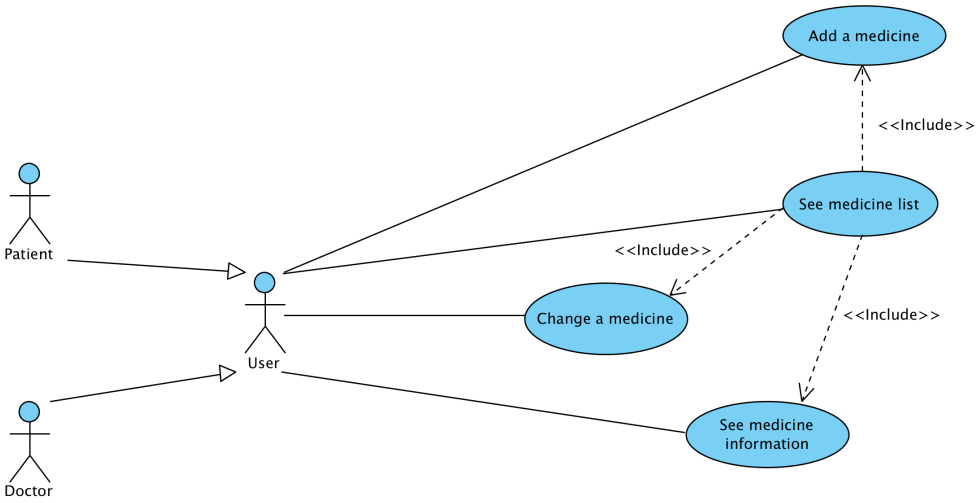


FIGURE 5.2: Medicine-related use case diagram

This chapter showed the multiple actions that the system is able to do, as well as the interactions that are possible with a user to be able to manage his medications.

The following chapter will describe the different elements defining the architecture of Andaman 7, into which the medication system will have to fit in.

Please note also that the source code created and modified for this project is available on *MatheO*, the ULg's platform for Master theses, and at the following address:
www.student.montefiore.ulg.ac.be/~s111093/TFE_Source_Code_SERVAIS.zip.

Chapter 6

Andaman 7 organisation

This is the first chapter on the architecture of the medication system. Before giving the details of this architecture, it is essential to understand beforehand the context in which the system will be implemented. That is the goal of this chapter that will set its focus on describing the principal aspects to be closely considered, specially the architecture elements used by Andaman7.

The two main pieces that will be tackled are how the medical data are represented internally to be processed and the GUI is constructed to display those data. The chapter will be closed by a small section commenting and criticising those mechanisms.

6.1 Ami structure

6.1.1 Ami

All the data related to a medical record are represented in the model using the concept of **AMI**: Atomic Medical Item. This is a generic representation of a medical data defined by:

- an **id**: unique string,
- a **type**: string denoting the type of the Ami, e.g. `string`, `number`, `date`, and so on. They are listed in TABLE B.1,
- a **selectionList id** if the type is `oneSelection` or `multiSelection`,
- one or multiple optional **tags**: enable to group the Ami around a tag, e.g. administrative, document, contact, disease,...

The goal of having such a representation is to allow all the data using this model to be treated in the same way. As the elements constituting the application are built over it, there is no need to take care of common functionalities since they are already been developed for all `Ami` data, as the persistence (i.e. saving the data) for instance. They can be considered without totally taking into account the data they carry.

An Ami is composed of:

- One or more `Qualifier`'s': property of an Ami, defined by:
  - an **id**,
  - a **type** (listed in TABLE B.1),
  - a **selectionList id** if the type is `oneSelection` or `multiSelection`
- One or more **markers**: characterize the Ami in a generic manner, allowing the application to behave differently depending on the markers. The different markers are listed in TABLE B.2.

The `Ami`'s are defined in a dictionary, in an XML file. The reader can find some examples of Ami definitions in APPENDIX B.3.

The `Ami` defined in XML in the `tami-dict` file are actually TAMI, Type of Atomic Medical Item. A `Tami` describes the data contained in the corresponding `Ami`. One can think of an `Ami` as an *instance of* a `Tami`.

6.1.2 Ami classes

After having parsed the `tami-dict` file, the application instantiates the corresponding `Tami` objects. An AMI containing a value is stored in an `AmiBase`. In the same way an `Ami` can contain `Qualifier`'s in `tami-dict`, an `AmiBase` can contain `AmiQual`'s. If we take the example of `ami.weight` on FIGURE B.1, once the piece of XML has been parsed, it creates a `Tami` with `ami.weight` as `tamiId` and two others `Tami` as qualifiers: `qualifier.date` and `qualifier.unit`. If the user gives some value for its weight in the application, an `AmiBase` is created using using a method of the `AmiBaseController` class. Moreover, if the user gives a value to `qualifier.date`, it will create a `AmiQual` with that value. This example is illustrated in FIGURE 6.1.

On the other hand, to retrieve the `AmiBase`'s, one has to use an `AmiContainer` object that will recover the `AmiBase`'s it contains, via their id for instance.

A class diagram shows the organisation of those classes in FIGURE 6.2.

FIGURE 6.1: Illustration of the utilisation of the `Ami` related classes

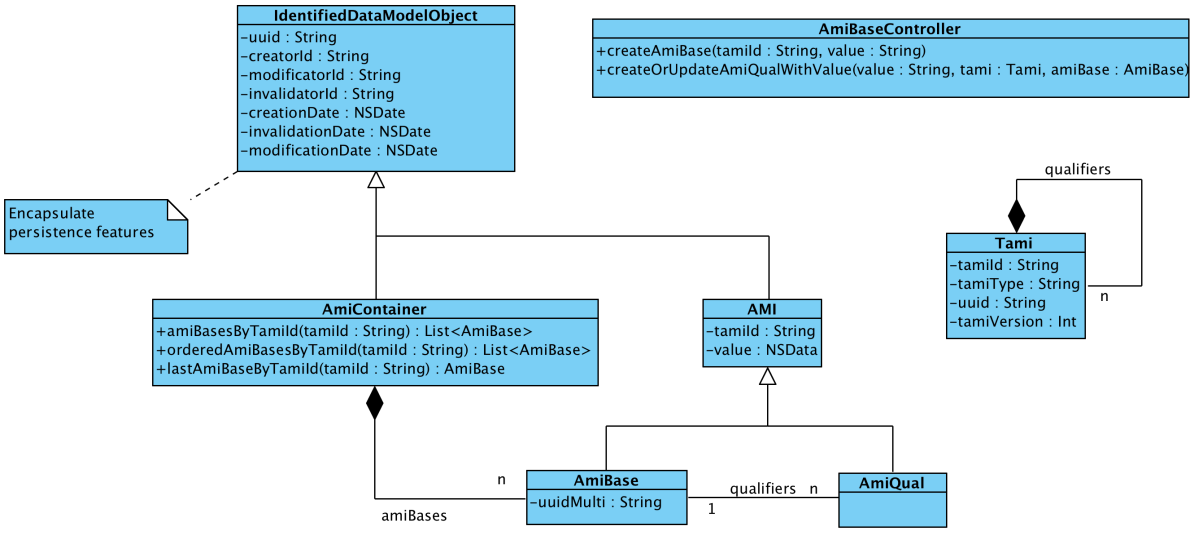


FIGURE 6.2: Class diagram of the `Ami` related classes

It is also important to note that the `AMI` are immutable, once their value has been set, it cannot be changed. Instead, the application creates a new `AMI` with a more recent creation date.

6.1.3 Selection List

An Ami or a qualifier could take its value in a predefined set of values, and only from one set of values. This set is called a **selection list**. For instance, the blood group can only be A+, A-, AB+, AB-, and so on.

A selection list is defined by an **id** and contains a set of items, themselves defined by an **id**. The selection lists are written in the same dictionary as an Ami. An example is shown in FIGURE B.3.

A list of Ami's and selection lists can be found on the Andaman7 developer portal [15].

6.1.4 AmiRef

An `AmiRef` is a reference to an `Ami`. It is defined by:

- an **id**,
- the **id of the Ami** to which it refers.

6.1.5 AmiSet

An `AmiSet` is structure allowing to group together several `Ami` via `AmiRef`. It is defined by:

- an **id**,
- a **type**: as for an `Ami` or a `Qualifier`,

- one or multiple optional **tags**: as for an `Ami` or a `Qualifier`.

It contains the `AmiRef`'s that are grouped in the `AmiSet`, as well as `Qualifier`'s and `Marker` if needed.

An example with an `AmiSet` and `AmiRef`'s is shown in FIGURE B.4.

6.1.6 Versioning

The dictionary in which the Amis and selections lists are defined are organized by version, one file corresponding to a version. For instance, the file *tami-dict-8.xml* is the 8th version of the dictionary. However, each file does not redefine each time the dictionary, it rather relies also on older versions.

The files are parsed to extract the dictionary when the application is first launched after an installation, but it can also be updated using a synchronization mechanism created by A7 Software.

6.2 GUI

Certain parts of the GUI are defined in an XML file, this is the case of the medical records. The file defines a hierarchy of views following the structure shown in FIGURE 6.3. The file is parsed at the first launch of the application and the GUI is dynamically generated afterwards.

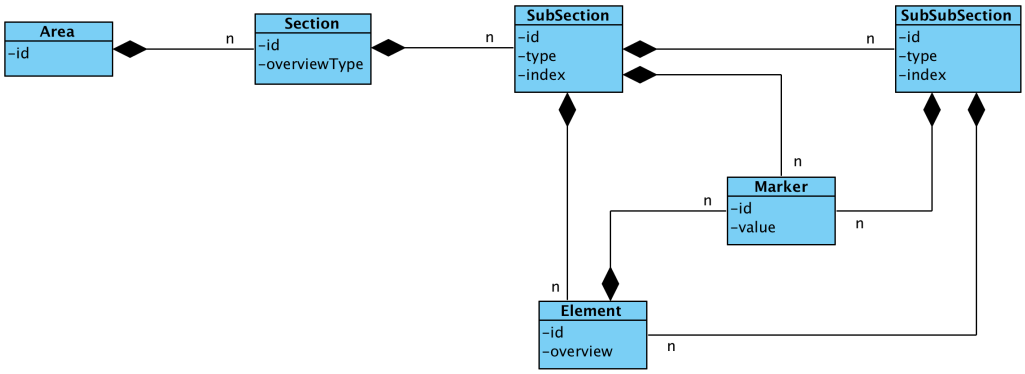


FIGURE 6.3: Hierarchy composing the *gui-dict* file

6.2.1 Area

An `area` defines a big zone in the application, identified by an **id**. There are two areas currently defined: the main one for the EHR (medical record) and another used to define the settings.

6.2.2 Section/SubSection/SubSubSection

An `Area` is divided into `Section`'s, themselves divided into `SubSection`'s that can, optionally, be also divided into `SubSubSection`'s.

The first screen on FIGURE 6.4 shows the `Section`'s of the EHR, e.g. Allergies, Exams,... The `overviewType` property describes how the overview of the elements will be displayed. For instance, on FIGURE 6.4, "Care facilities" and "Life habits and risk factors" are displayed differently.

The second screen on FIGURE 6.4 shows how are represented `SubSection`'s, here "Consultations". The `type` property defines how the elements of the `SubSection`'s will be displayed. In the same way, the `index` property defines in which order they will be displayed.

`SubSubSection` works in a similar manner as `SubSection`. The result in the GUI is shown in the third screen on FIGURE 6.4.

6.2.3 Element

An `Element` is the building block of the structure, it is the GUI element representing an Ami, an AmiRef, an AmiSet or a qualifier. An example is shown in the third screen on FIGURE 6.4.

6.2.4 Marker

In the same way `Marker` is used in `Qualifier`, it allows to change the default behaviour of the application regarding to the display of the `SubSection`, `SubSubSection` or `Element`.

6.3 Comments

The `Ami` structure is a nice way of organizing the medical data that, in most cases, is treated the same. It allows to re-use the portions of code dealing with the same kind of data. However, some particular data may need appropriate operations. To handle those special kind of data, the developer has several obvious solutions:

- Try to fit into the `Ami` model, for instance developing an encoding/decoding scheme to transform the data into a string and vice versa,
- Implement new `Ami` type, also meaning to handle some operations, as displaying, but other internal procedures are already taken care of,
- Bypass the `Ami` mechanism, meaning that most of the operations have to be re-implemented, such as persistency or displaying.

Naturally, the decision of selecting the best solution highly depend on the data to be handled.

Also, having an automated way of constructing the GUI from an XML file allows for more re-usability. While the data to be displayed follows the same structure or scheme, it seems good to treat them in the same way. It is also quite simple to use, once the `Ami` structure has been encoded, displaying it is a matter completing the *gui-dict* file. Finally, it allows to have a consistent UI and UX.

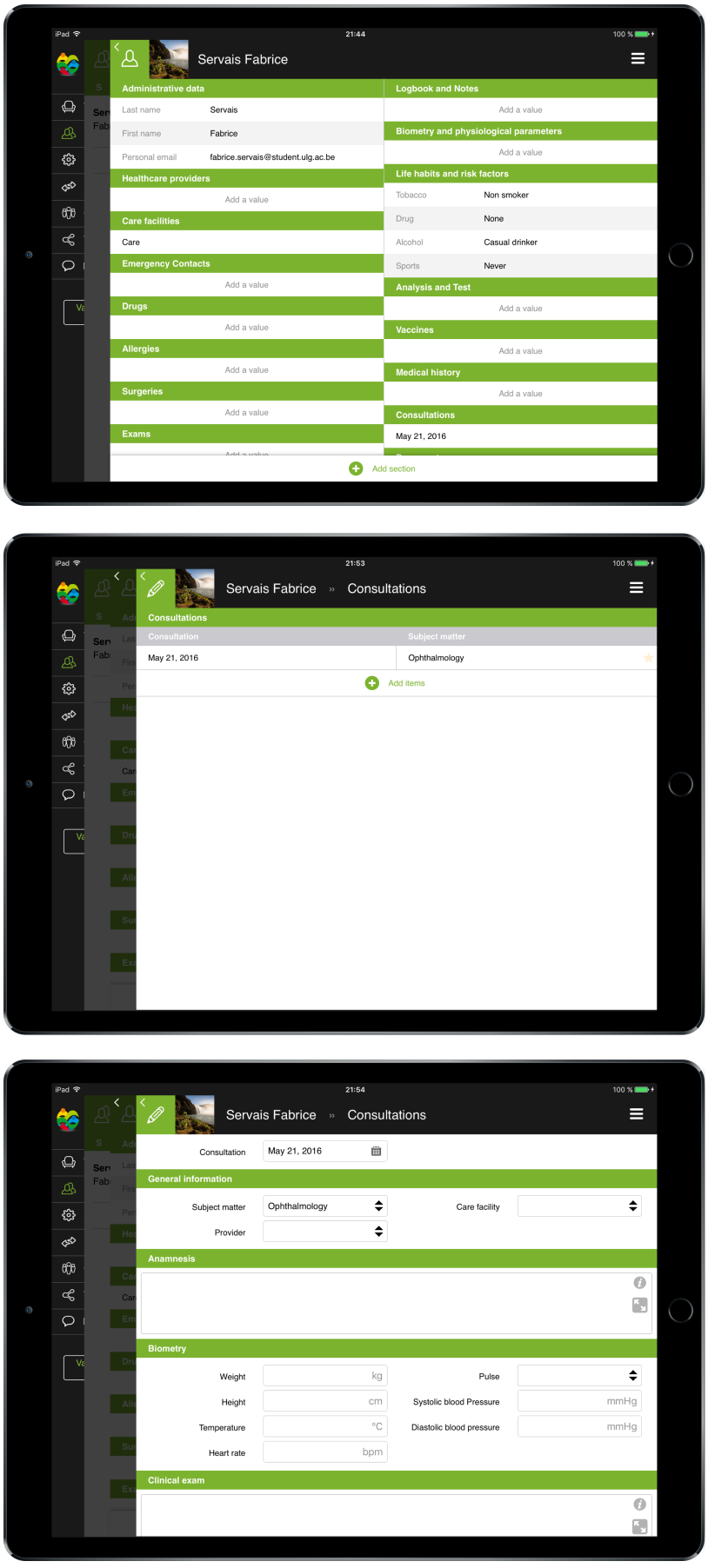


FIGURE 6.4: Examples of GUI elements: `Section`, `SubSection` and `SubSubSection`

As for downsides, such mechanism induces a lost of control over the GUI. As soon as there is a need for a different treatment, a more complex data management for instance, it becomes complicated, even impossible, to use the *gui-dict* file to achieve it. Here also, the choice of using this mechanism or bypass depends on the need, which may be either developing a new field or an entire section. Also, with lots of different GUI elements as well as exceptions for particular pieces, it could lead to overly complex code.

> This chapter has shown the relevant constitutive elements of the architecture of Andaman 7 that needed to be understood for the following. Indeed, in the next chapter, how those elements are used and customized to build a medication compliance system will be explained.

Chapter 7

Medication compliance system architecture

From previous chapter, we have seen the elements of architecture developed by A7 Software for Andaman7. In this chapter, we will see how those elements are used in the medication compliance system to form its architecture, as well as other elements proper to the system.

7.1 AMI representation

Following the `Ami` structure described in Section 6.1, the model needed for the medication compliance system has been defined in a new version of the `tami-dict` file, which is the version 15.

7.1.1 Medication

The `Ami` representation of the medication is shown in TABLE 7.1.

The noteworthy elements are:

- The value of `medication` is a UUID[1] so that a particular medication can be found from other parts of the system.
- The elements of `unit` are the items listed in `sl.dosageUnits` (cfr. TABLE 7.6).
- There are multiple ways of kind of days frequencies (recalling 4.1.1: *days every week* and *days interval*). It is thus needed to have an encoding/decoding scheme to be able to retrieve those frequencies from `days`.
  The content of `days` is of the form:

  <type>|<payload>

  where <type> is either:

[1] Universally Unique Identifier

ID	Ami type	Type	Description	Marker
medication	AmiSet	String	Unique identifier.	Multi
medicineName	Qualifier	String	Name of the medicine associated to the medication.	/
dosage	Qualifier	String	How many units of medicine.	/
unit	Qualifier	OneSelection	Unit of medicine. References sl.dosageUnits.	/
reasonForTaking	Qualifier	String	Note associated to the medication.	/
pushNotification	Qualifier	String	Boolean converted to string indicating if the notification are activated.	/
startDate	Qualifier	Date	Start date.	/
endDate	Qualifier	Date	End date.	/
days	Qualifier	String	Encoding of the days frequency.	/
frequency	Qualifier	String	Encoding of the time frequency.	/
frequency.startTime	Qualifier	String	Start time if a time frequency by interval is specified.	/
reminder	Qualifier	String	Encoding of the list of times of the reminders.	/

TABLE 7.1: AmiSet: **Medication**

  - 'W' if the frequency is of the type *days every week*. The <payload> is then days from {MO, TU, WE, TH, FR, SA, SU} separated with the character '|'. For example: "W|MO|WE|FR" represents "each Monday, Wednesday and Friday every week".
  - 'I' if the frequency is of type *days interval*. The <payload> is then of the form x|'D', where x is the number of days.

- In the same way as days, frequency can represent two types of frequency: *times per day* and *hours interval*.
  The content of days is of the form:

  <type>|<payload>

  where <type> is either:

  - 'T' if the frequency is of type *times per day*. The <payload> is then the number of times per day.
  - 'I' if the frequency is of type *hours interval*. The <payload> is then of the form x|'H', where x is the number of hours.

- The `reminder` qualifier holds the time of each reminder. Each time is encoded in the format $HH : mm$ and they are separated by the character `'|'`.

7.1.2 Medicinal product

The `Ami` representation of a medicinal product (i.e. medicine) is shown in `Table` 7.2.

ID	Ami type	Type	Description	Marker
`medicinalProduct`	`Ami`	String	Name of the medicine.	Multi
`dosage`	`Qualifier`	String	Dosage of the medicine.	/
`form`	`Qualifier`	String	Form in which is conserved the medicine.	/

TABLE 7.2: `Ami`: **Medicinal Product**

At first, a drug was a string field when adding a medication. However, as the second part of thesis will analyse, there could be in the future a list of medicines from which the user can select one. As a prevision, the medicinal product became a separate `Ami`.

7.1.3 Intake

As the user receives a notification for its medication, he has the ability to mark the intake as taken or not taken. Those intakes are recorded using the `Ami` mechanism. The representation is shown in `Table` 7.3.

ID	Ami type	Type	Description	Marker
`intake`	`Ami`	String	Unique identifier.	Multi
`medicationUUID`	`Qualifier`	String	Unique identifier of the medication.	/
`datetime`	`Qualifier`	Date	Date and time of the intake.	/
`quantity`	`Qualifier`	Date	How many units of medicine has been taken.	/
`taken`	`Qualifier`	String	Boolean indicating if the medicine has been taken or not.	/

TABLE 7.3: `Ami`: **Intake**

7.1.4 Registered event

As will be explained in Section 7.3.1, the recurring events associated to the medications where the push notifications are activated need to be stored. To do so, the `Ami` mechanism is used, as shown in TABLE 7.4.

ID	Ami type	Type	Description	Marker
registeredEvent	Ami	String	Encoding of the ID's of the related medications.	Multi

TABLE 7.4: Ami: **Registered event**

The registeredEvent can contain lists of UUIDs organised by key. The key, in the case of the medications, is the identifier of the AmiSet, i.e "amiset.medication". To achieve that, this dictionary is encoded in JSON format.

7.1.5 Selection lists

Two lists of items are needed for the medications:

- Name of the days, i.e Monday, Tuesday, and so on,
- Units possible for the intakes of the medicines, e.g. pill, ml, and so on.

They are represented using SelectionList's, shown in Table 7.5 and Table 7.6.

ID	Ami type	Type	Description	Marker
days	SelectionList	/	List of the days.	Ordered

TABLE 7.5: SelectionList: **Days**

ID	Ami type	Type	Description	Marker
dosageUnits	SelectionList	/	List of medicine units.	/

TABLE 7.6: SelectionList: **Dosage units**

7.2 Recurring events

The medication can be seen as a recurring event, occurring from a starting date with a particular pattern. In order to model it efficiently in the system, and not store each and every event, a representation of recurring events has to be developed. Moreover, the representation should be as generic as possible so that it can be applied to other contexts.

The architecture is inspired from [1] and is shown in FIGURE 7.1.

The main element is TemporalExpression. It is an interface allowing other objects to ask for the dates it represents. Putting in another way, a class implementing TemporalExpression has to be able to return its dates of occurrence. For instance, a medication could be to take a drug

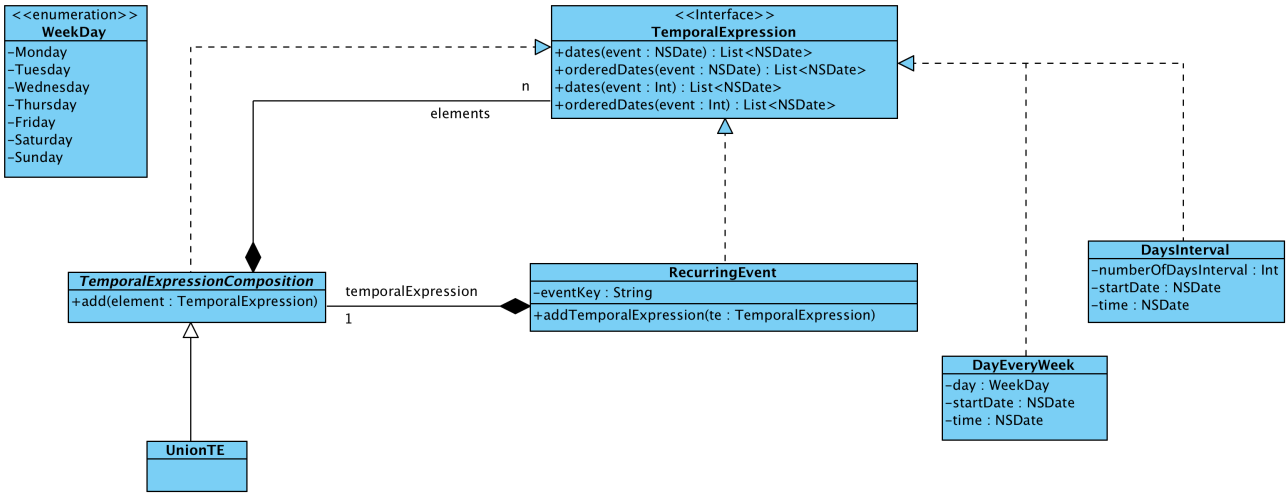


FIGURE 7.1: Recurring event architecture

once a week. The "once a week" part of the sentence should be representable by a class implementing `TemporalExpression`. This kind of recurrence is implemented by `DayEveryWeek`. Another kind is days interval, e.g. *every 3 days*, and it is represented using the `DaysInterval` class. The `TemporalExpression` interface (i.e. `protocol` in Objective-C/Swift) is shown in FIGURE 7.2.

```
protocol TemporalExpression {
    func dates(until end: NSDate) -> [NSDate]
    func dates(maxNumberOfDates nDateMax: Int) -> [NSDate]
    func orderedDates(until end: NSDate) -> [NSDate]
    func orderedDates(maxNumberOfDates nDateMax: Int) -> [NSDate]
}
```

FIGURE 7.2: `TemporalExpression` interface

Several `TemporalExpression`'s could be gathered together using another class `TemporalExpressionComposition`. It is an abstract way of grouping `TemporalExpression`'s, for instance, as shown in FIGURE 7.1, `UnionTE` inherits it to have a class representing an union of `TemporalExpression`'s. Other examples are `IntersectTE`, returning only the events common to all its elements, or even `DifferenceTE`, and so on. `UnionTE` also implements `TemporalExpression` so that it can appear as a `Temporal Expression`. One can do an analogy with sets, where the union of sets is also a set. Taking back the medication example, it could be possible to represent one where a drug should be taken on Monday and Wednesday every week. There would be one `TemporalExpression` for the expression "Monday every week" and one other for the expression "Wednesday every week".

From the previous example, a medication can be represented with a `RecurringEvent` class. It adds a key to `TemporalExpressionComposition` to identify the event.

7.3 Local notifications

At the specified time of intake, the user can be notified to take his medication if he activated this feature. This can be achieved using built-in iOS time-triggered notifications, i.e. local notifications. They simply need to be provided with some information (trigger time, title, body message,...), and then it is possible to schedule them.

7.3.1 `ScheduleManager`

This system has a limitation of having a limit of 64 notifications schedulable at the same time. To overcome this, a `ScheduleManager` has been designed to be the interface to the iOS notification system. It is possible to register `RecurrentEvent`'s to it and the `ScheduleManager` will take care of scheduling the notifications, even if more than 64 of them are needed.

The process of re-scheduling the notifications using the `ScheduleManager` is shown in the high-level activity diagram in FIGURE 7.3. This process is triggered when an event is registered or when a local notification is received.

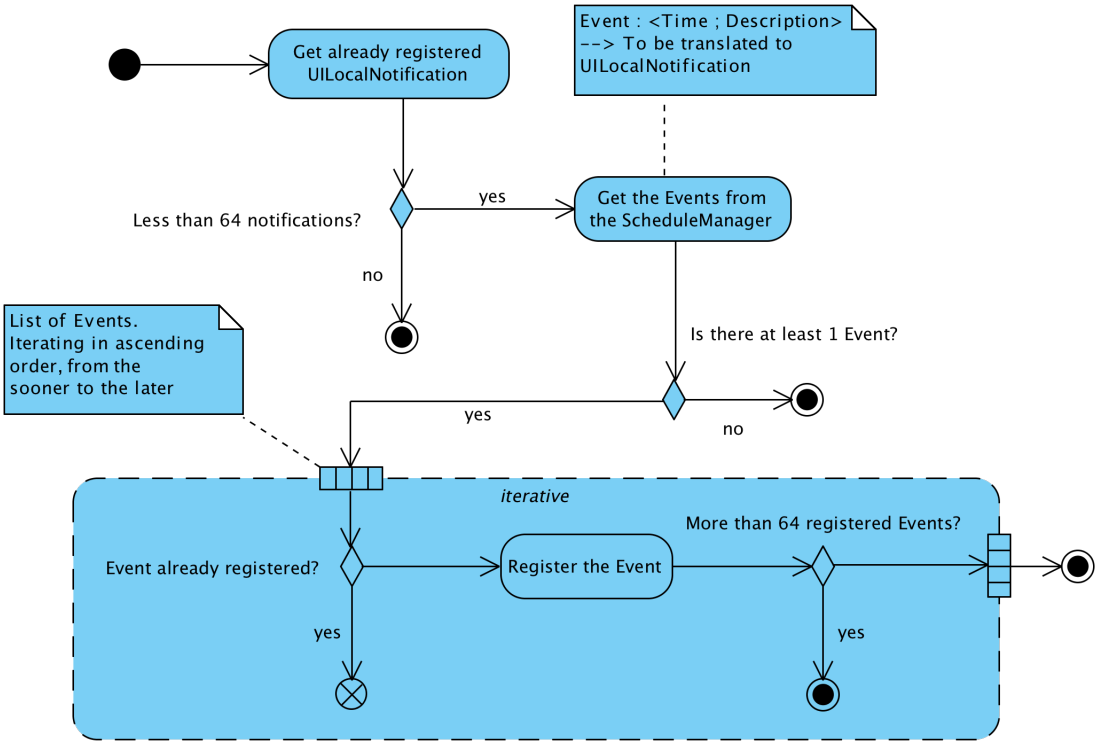


FIGURE 7.3: Schedule iOS local notification using the `ScheduleManager`

The class diagram of the `ScheduleManager` is shown in FIGURE 7.4, its interface is also shown in FIGURE 7.5.

Persistence

If the user quits the application, the data hold by the `ScheduleManager` goes away since they are not persistent. To store the data and make them available across the launches of the application, I used the `Ami` mechanism explained before, therefore, I created an `Ami ami.registeredEvent` of String type. It has no qualifier, all the information lies in its value in a JSON format that will be detailed a bit later.

Each time the user registers a recurring event into the `ScheduleManager`, the latter has to take care of storing it. This can be achieved by storing in a new `AmiBase` an encoding of all the events already registered, each time an event is registered. This encoding is a dictionary in a JSON format where the keys are the keys in `events` in the `ScheduleManager` (cfr. FIGURE 7.4), e.g. "amiset.medication", and where the values are lists of the key of `RecurringEvent` (cfr. FIGURE 7.1). In the case of the medications, the key of a `RecurringEvent` is the UUID of the medication.

In the same way, when the user unregisters the recurring event, the `ScheduleManager` has to take care to remove it. It simply removes it from its list and create a new `AmiBase` containing an encoding of all the events still registered.

When the `ScheduleManager` is instantiate for the first time, it will check to see if there are some already stored `AmiBases`. If yes, that means that the ScheduleManager has some registered events, it thus has to load them. To do so, it parses the JSON to obtain the UUIDs of the events. In the case of the medication, it will look into the Model to find the corresponding medication and re-build the corresponding `RecurringEvent`.

7.3.2 Actions

Upon receipt of the notification, the user has the possibility to:

- Press the "Taken" button,
- Press the "Not taken" button,
- Ignore the notification.

In the case where the user pushes the "Taken" or "Not taken" button, an `Ami intake` will be created and marked as taken or not accordingly. Two examples are shown in FIGURE 7.6, the first one is when the user receives a notification outside the application, the other one being when the user receives the notification inside the application.

> In this chapter, we saw how the `Ami` model was used to fit the data needed to represent medications. We also saw two other elements that are how to represent with classes a recurring event and how the notifications are handled by the system.

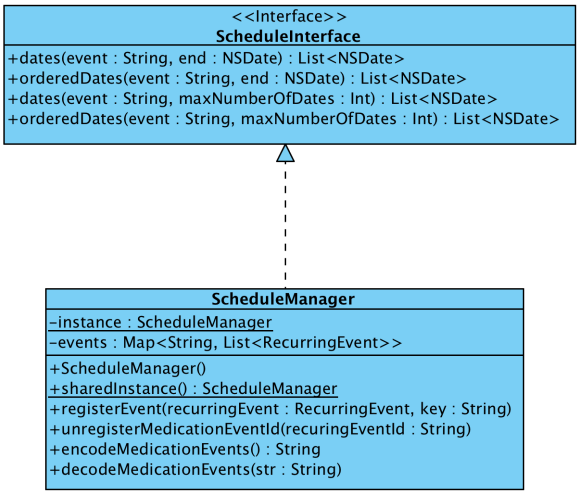


FIGURE 7.4: `ScheduleManager` class diagram

```
protocol ScheduleInterface {
    func dates(event: String, until end: NSDate) -> [NSDate]
    func dates(event: String, maxNumberOfDates nDateMax: Int) -> [NSDate]
    func orderedDates(event: String, until end: NSDate) -> [NSDate]
    func orderedDates(event: String, maxNumberOfDates nDateMax: Int)
            -> [NSDate]
}
```

FIGURE 7.5: `TemporalExpression` interface

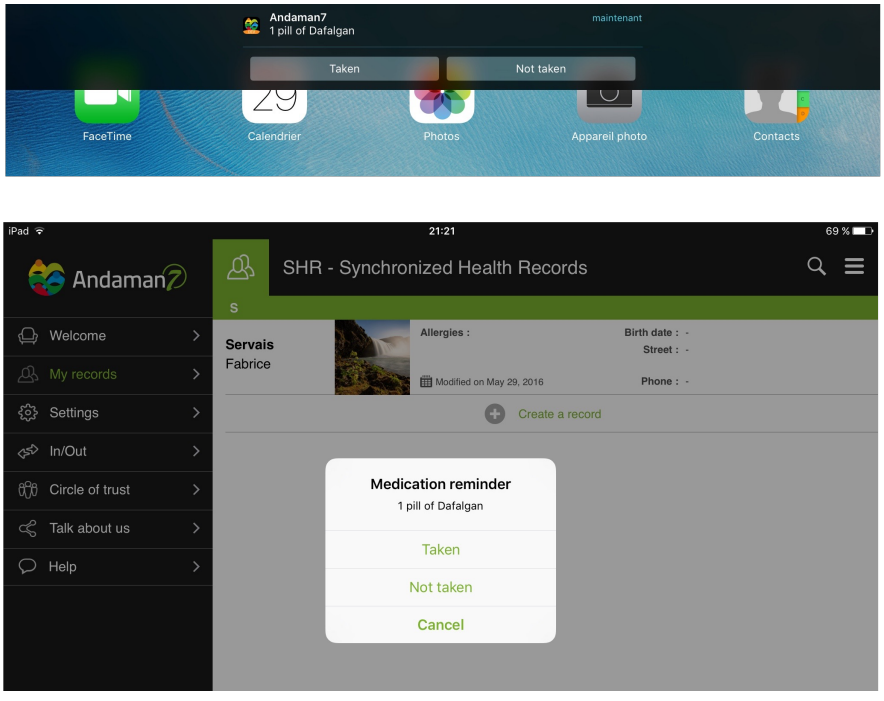


FIGURE 7.6: Example of notification: outside and inside Andaman7

Another big part of the architecture is how are organised the GUI and the elements that are are related to it. However, before diving into that subject, the next chapter will give a general overview of the development of applications under iOS, as well as the specificities related to it.

Chapter 8

iOS development

This chapter acts as a transition between the architecture that has been depicted and the next chapters more focused on the GUI and the implementation of other components. It will describe some particularities related to iOS development to better understand the next topics. First, it will explain the MVC pattern and its specificities in iOS, and then it will compare the existing methods to build a GUI.

8.1 MVC

The Model-View-Controller design pattern is famous in the software engineering world, it basically consists at separating the data (Model) from how they are processed (Controller) and how they are displayed (View), knowing that the operations are managed by the Controller so that the View and the Model are not interacting with each other.

This pattern is heavily used in iOS and is at the core when developing an application. FIGURE 8.1 from [14] shows the interactions that can happen between the Model and the Controller, as well as between the View and the Controller. This section will briefly describe them.

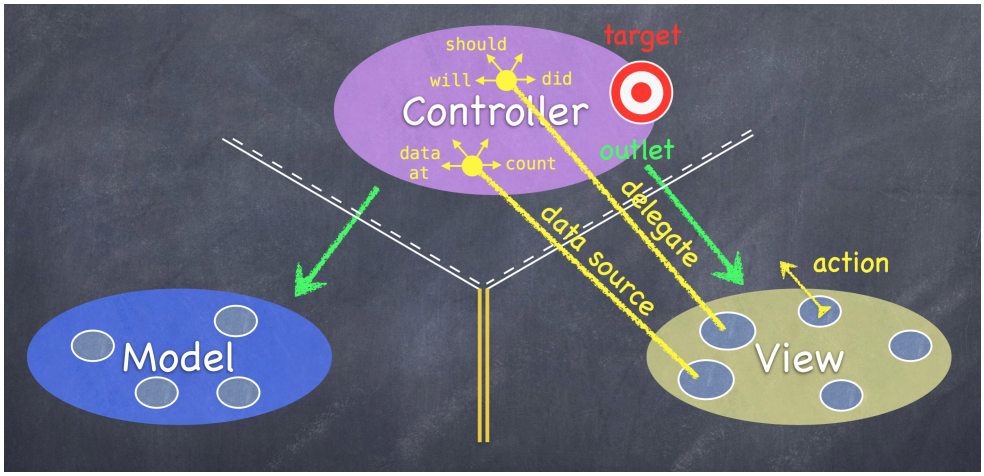


FIGURE 8.1: MVC pattern in iOS

- **Outlet**: an outlet is the reference that the Controller has on an element of the View to be able to interact with it.
- **Action/target**: the controller can register a particular action to be triggered when an event occurs on the View. For instance, when a button is touched, the method `func buttonPressed(sender:AnyObject)` belonging to the Controller can be called.
- **Delegate**: according to the iOS documentation ([12]): "Delegation is a simple and powerful pattern in which one object in a program acts on behalf of, or in coordination with, another object.". Simply put, a class can have a reference (i.e. a delegate) to an object that it knows to conform to a protocol (i.e. implements an interface[1]) or being an instance of a specific class, so that it can send a message when a particular event occurs. On FIGURE 8.1, the View has the Controller as its delegate so that when an predefined event occurs (view finished loading, view will appear,...), the View can send a message (i.e. calling a method) to the Controller corresponding to the event.
- **Data source**: the principle of the data source is the same as with delegates, except that the nature of the delegation is a bit different. Instead of giving control of the interface to the delegate, it gives the control of the data to the View though the Controller. For instance, a table view could list some elements in the Model, the View can ask them using a particular method of the Controller.

8.2 iOS User Interfaces

There are several ways to design user interfaces on iOS, summarized and compared in [17].

8.2.1 Programmatic UIs

A user interface can be created only using lines of code. This method allows the developer to really understand how the GUI elements are used and interacts together. Indeed, whatever the tool that might be used to build a GUI, it will always be translated in lines of code at the end of the process. Also, if the implementation is well designed, some elements could be reused.

As for the downsides, it is not a visual solution, it can quickly become hard to see what will be the result, so prototyping can be very time-consuming. Also, it can be complicated to maintain if the project goes to other developers a while after.

Programmatic UIs are good at what the other solutions can't do, as having more elaborate or dynamic UIs for instance.

[1]The terms *protocol* and *interface* are different words for the same concept: defining a set of methods to implements. In the exact same way, *implementing an interface* becomes *conforming to a protocol* in Apple jargon.

8.2.2 NIBs

In iOS, views (i.e. UI elements, inheriting `UIView`) are always managed by a view controller (i.e controller, inheriting `UIViewController`). While a view controller manages a single view, it could be possible to have a composite view, i.e. itself containing other views. Each of these views can be contained in a NIB file (NeXTStep Interface Builder), as illustrated on FIGURE 8.2. An example of what a NIB file could look like is shown in FIGURE 8.3.

This methods has a good trade-off between modularity and visual prototyping. Each component becomes easier to develop. It is a good method to use for reusable elements, such as templates for table view cells, and so on.

8.2.3 Storyboards

Storyboards are a visual tool to manage more easily multiple application views (called "scene"), as well as the transitions (called "segue") between them. FIGURE 8.4 shows an example of what a storyboard could look like.

Storyboard are rather useful when it concerns a set of views with transitions between them, as with a multi-step process for instance. It is also an easy tool to use for prototyping a user interface.

The drawbacks are that first, it diminishes the reusability since, as all is grouped together in one Storyboard, it is difficult to reuse a part of it. Another inconvenience is that there is no integrated mechanism to pass the data through the view controllers composing the Storyboard, even if it can still be done in practice.

8.2.4 Which one to use?

As always, there is no unique answer to that question, just as much as there are multiple ways of building the same UI. However, we cannot say that a project has to stick with only one method. Each of these has its own purpose and targets different problems. A suitable way is to mix those tools, for example using Storyboards to group related and linked screens, using NIB files to customize some parts of the UI and favour reusability, and finally use code for more complex views.

> This concludes this chapter on iOS development and how it works with a high-level view. The following chapter will explain the construction of the GUI for the medication compliance system.

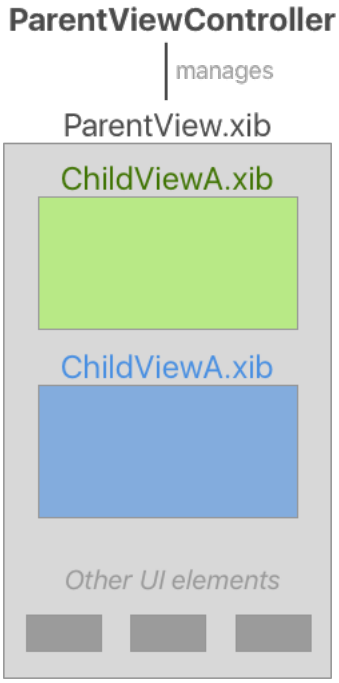


FIGURE 8.2: NIB files organisation

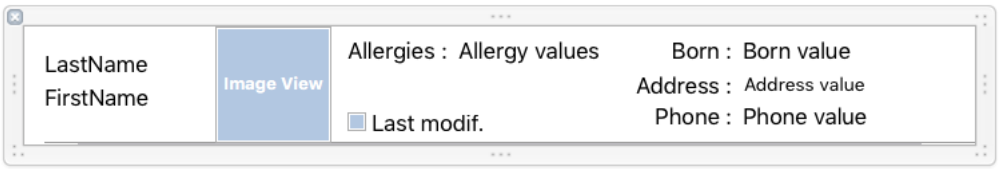


FIGURE 8.3: NIB file example

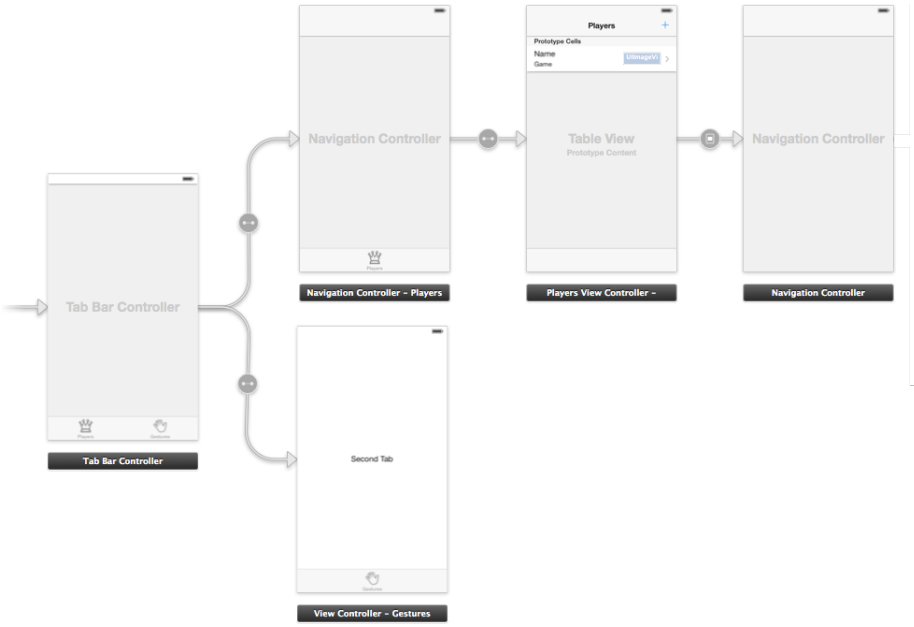


FIGURE 8.4: Storyboard example [10]

Chapter 9

Graphical User Interface

From the previous chapter, we have seen different methods and tools to build a GUI. This chapter will first list the different needs for the user interface that can be extracted or inferred from the requirements. Then, the automatic interface creation of Andaman7 will be overviewed, before explaining how I arrived at the final version of the GUI for the medication compliance system.

9.1 GUI specifications

From the requirements listed in Chapter 4, we can outline some specifications of the GUI, using the *iOS Human Interface Guidelines* [11].

- There should be field to write strings in it (FIGURE 9.1a),
- There should be drop-down lists to select one or more items (FIGURE 9.1f),
- There should be date and time pickers to select a date or a time (FIGURE 9.1c),
- There should be switch buttons or activate or deactivate an element (FIGURE 9.1b),
- There should be possible to display messages waiting for an action (e.g. a confirmation message asking to confirm the action or not) (FIGURE 9.1e),
- There should be a number picker to choose a number while restricting the user from enter something else (FIGURE 9.1d),
- Some of these elements should be displayed inside a view appearing dynamically on the touch of a button or a field.

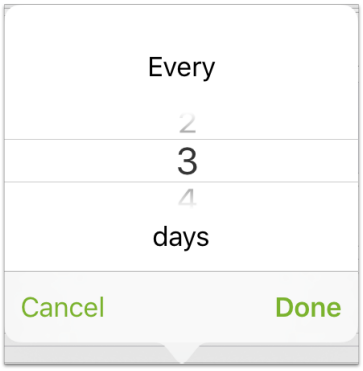
9.2 First attempt

As explained in Section 6.2, the GUI is automatically build from an XML file, constructing the field depending on the type of the `Ami` to display. While it is easy to use, it doesn't fit the needs of the medication compliance system. Let's first explain how this generation works.

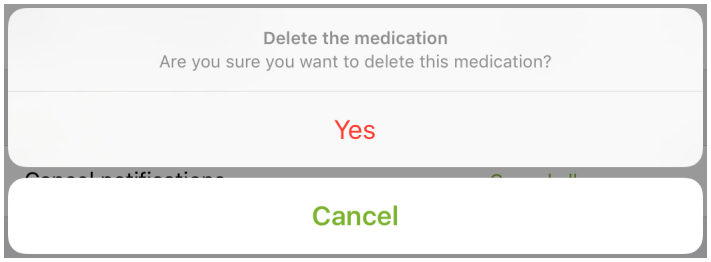
(A) Text field

(B) Switch

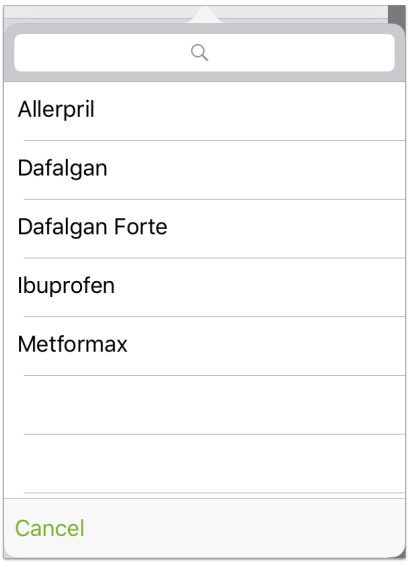
(C) Date picker



(D) Number picker



(E) Action sheet



(F) Drop-down

FIGURE 9.1: GUI elements

9.2.1 A7 automatic GUI creation

As shown in FIGURE 9.2, the organisation of the elements is done using table views (`UITableView`), one of the most common views in iOS to display data under the form of a list. In this case, each `SubSubSection` is a cell of the global `UITableView`. In the same way, in order to display the fields, each cell embeds itself a `UITableView` where the cells contains the fields. This hierarchy is shown in FIGURE 9.2. Also, recalling the Section 8.2, since the process is automated, the views created are both programmatically made and uses NIB files.

9.2.2 Medication compliance system

The first attempt was to try to mimic the automatic creation of the GUI and have a result similar to what is already existing, as in FIGURE 9.2, to have a visual aspect which is coherent with the rest of the application. I thus created a panel containing a global table view, itself containing two other table views, one for the general information (i.e. name of the medicine, dosage,...) and the other one to display the times of each reminder. The result is shown in FIGURE 9.3.

This solution came with several drawbacks:

- There is a lot of code overhead coming with the management of the GUI, even from simple elements, and problems from those embedded table views.
- The visual aspect, even if close to the rest of the application, is not quite what was expected.
- It is difficult to have a "step-based" process for registering a medication (i.e. multiple panels). In this solution, all the information have to be provided in a single panel, where it is not simple to have dynamic interactions between the elements that compose it.

9.3 Usage of storyboards

After consulting with A7 Software, the direction regarding to the UI changed. Since the application is being refactored for a new version and that the UI is being re-melted, the programmatic method and the NIB files are used to a lesser extent, introducing Storyboards for some parts of Andaman7. It thus seemed natural to try this solution by prototyping an encoding panel for a medication, considering that the visual coherence became less important.

9.4 Medication compliance system UI

The process to register a medication is done in three steps, each one is associated a view controller. The first one, `GeneralInformationViewController` asks the general information of the medication, as the medicine, the dates, the frequency, and so on. The second one, `ReminderListViewController`, displays the reminders on a 24-hour period, allowing the user to change the

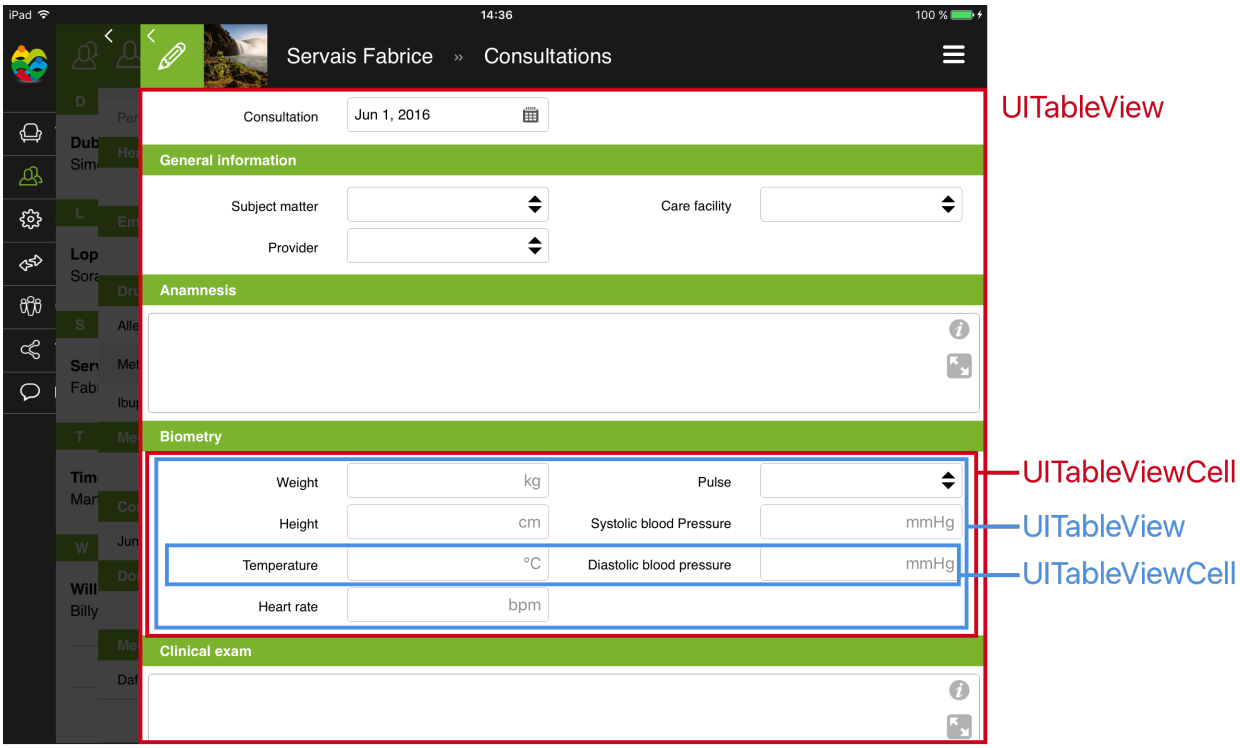


FIGURE 9.2: Consultations panel

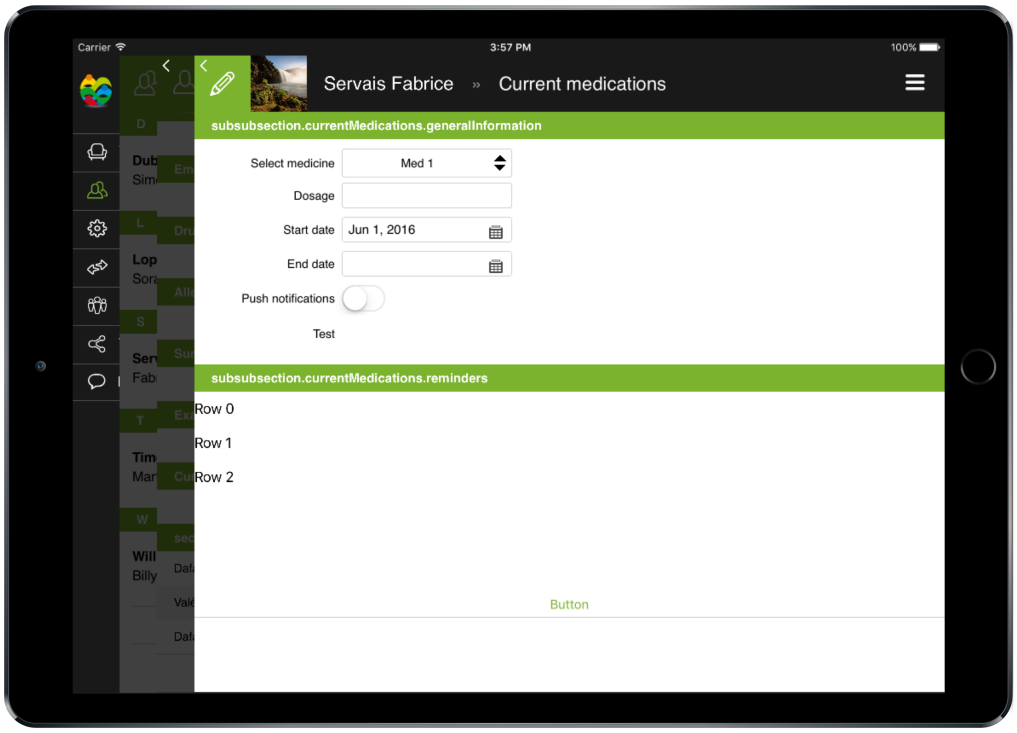


FIGURE 9.3: First attempt of designing the medication panel

times. The last one, `AdditionalInformationViewController`, displays the fields for the additional information, in this case, the switch to activate or not the notification, as well as the notes related to the medication.

The views are shown in FIGURE 9.4.

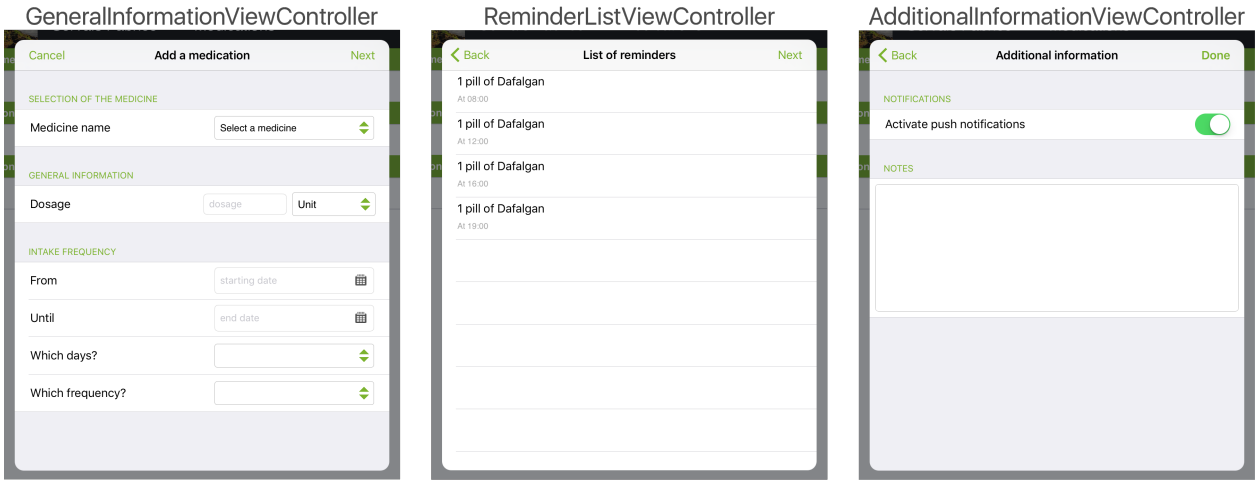


FIGURE 9.4: Medication encoding view controllers

> We have seen what kind of interface was needed, what does the final GUI look like, as well as the process to get to such a solution. The next chapter will focus on the implementation of the system in general, whether in the GUI or simply part of the system.

Chapter 10

Implementation details

The previous chapters had a focus on the requirements and on the architecture of the developed system. This chapter will describe some aspects of the code that needs to be explained or justified, as well as some extra implementations. It will first go through the elements constituting the GUI and give a commentary on some elements, and then it will review two features that has been added, the form validation and how to order a selection list form its definition.

10.1 User Interface

10.1.1 Storyboard

As explained in the previous chapters, the usage of Storyboard was finally privileged to build the UI. The final Storyboard is shown in FIGURE 10.1.

Navigation Controller

The scenes corresponding to the steps to encode a medication are embedded in a Navigation controller, the entry point of the Storyboard in FIGURE 10.1. A Navigation Controller manages a stack of other view controllers in which we want to navigate. In our case, we want to navigate through the different steps of the encoding shown in the previous chapter, in FIGURE 9.4. The root of the Navigation Controller's stack is the first view to be displayed, in our case `GeneralInformationViewController`. When a segue is called to go to another view, the corresponding view controller is pushed on the stack. In the same way, if the "Back" button of a view is pushed, the view controller is popped from the stack.

The Navigation Controller being itself a view controller, it thus has a view to manage. This view is naturally composed of the views managed by the view controllers in its stack, but it also adds a navigation bar, shown on the first scene on FIGURE 10.1, automatically creating the "Back" button on the left of the bar when a view controller is pushed. To navigate to the next scene, I added a button "Next" on the right of the bar, that triggers the segue to the next view controller when it is pushed.

Passing data

A problem that arises when using multiple panels to encode data is that those data need to be passed through the view controllers so that, when pushed the "Done" button in the last scene, the information that the user entered can be saved.

To solve that problem, I created a structure `MedicationFormData` which values will be the values entered by the user at any step of the encoding. It is created at the first scene and is passing through the view controllers, containing the information entered at each step.

As it currently is, the `MedicationFormData` structure suffers from flaws. It was initially created to be as simple as possible, the goal being to just carry some data across view controllers. However, it became more complex during the development, mostly because it was the only common structure across the view controllers. For instance, a reference to the medication `AmiBase` is kept in this, which is not quite related to the form. As well, there are two variables to keep track of the days selection, one when the user selects the days of the week and the other when the user selects an interval of days. This makes the checking of the values more complex. An enhancement would be to transform the structure into a class whose job would be to manage the data of the form more cleverly and not just keeping data.

10.1.2 Popovers

A popover is a container to display a view floating above the application's view when an element in touched, with an arrow pointing to the caller. Examples are shown in FIGURE 9.1c, FIGURE 9.1d, and FIGURE 9.1f in Chapter 9. It is used to present information or options to the user without being too intrusive.

There are two ways that can be used to present a popover: programmatically or using segues.

Programmatically

To display a popover programmatically, one needs to instantiate an object of type `UIPopoverController` and provide it with, amongst others, the content view controller to display and a size, and finally call `presentPopoverFromRect` on the popover controller to display it. An example is shown in FIGURE 10.2.

This process has been implemented in the function `showPopoverWithViewController` in `GeneralInformationViewController`.

Segue

Another way is to use the segue mechanism in the Storyboard. A segue is the transition between two view controllers in a Storyboard, it can be parametrized to let the destination view controller appear in a popover, as

shown in FIGURE 10.1. It is useful to choose a segue over lines of code when the view to present is not meant to be reusable.

10.1.3 Auto layout

It is worth noting that the Apple's Auto Layout mechanism has been used in the scope of this work. Auto Layout allows the developer to define constraints between the elements of the UI so that it can adapt when the size of the screen varies. Those constraints allows the elements to be disposed depending on the location of other elements, as for instance "center vertically view A with view B", "width of view A >= ...", "horizontal space between view A and view B = ...", and so on. An example is shown in FIGURE 10.3.

10.2 Form validation

Certain fields are mandatory, they need to have a value before continuing the process. For instance, the medicine has to be selected when the medication is created. It is also worth noting that the `Ami` structure does not manage mandatory fields yet.

The data are checked before the segue from `GeneralInformationViewController` to `ReminderListViewController` by overriding the method from `UIViewController`

```
func shouldPerformSegueWithIdentifier(identifier: String,
                                      sender: AnyObject?) -> Bool
```

stating if a particular segue with `identifier` should be performed or not.
The mandatory fields are checked and `shouldPerformSegueWithIdentifier` returns `false` if they are not filled.

In order to show that a mandatory field is not filled, the border will be coloured in red after the checking. To do so, I implemented the method

```
- (void)setErrorStyle:(BOOL)enabled
```

on the fields (`A7TextField` and `OneSelectionButton`) that set the border to red when `enabled` is `true`, and set back to normal when `false`.

10.3 Ordering a selection list

The `SelectionList` from the A7 architecture allows to define sets of values. However, there is no notion of order when defining them, because in general, the order in which they will be displayed doesn't matter, so that they are displayed in lexicographical order. Yet, in some circumstances, having a customized order could be an added value. This is the case with days for instance, a user does not want to see the days of the week listed in alphabetical order. We thus need a way to specify a personalised order.

Naturally, the order has to be defined with the `SelectionList`. To do so, the `Item` tag of `SelectionList` has been added a new attribute: `index`. Also, to mark the `SelectionList` as ordered, the tag `<Marker id="marker.ordered" />` has to be added within `SelectionList`. For instance, the `SelectionList` defining the days now looks like:

```
<SelectionList id="sl.days">
      <Item id="li.monday" index="1"/>
      <Item id="li.tuesday" index="2"/>
      <Item id="li.wednesday" index="3"/>
      <Item id="li.thursday" index="4"/>
      <Item id="li.friday" index="5"/>
      <Item id="li.saturday" index="6"/>
      <Item id="li.sunday" index="7"/>
      <Marker id="marker.ordered" />
</SelectionList>
```

As a first thought, it would have made more sense if the order, or at least the tag `<Marker id="marker.ordered" />` was defined in the *gui-dict* file rather than the *tami-dict* file since this is only a matter of display. However, since the medication encoding panels do not use the *gui-dict* file, it would have been impossible to achieve. Using the solution explained above, the order is now defined wherever the `SelectionList` is used.

As far as the implementation is concerned, a boolean was added to the `A7OneSelectionList` object (representing a `SelectionList`) to mark the list as ordered or not. The value is set to true when `Marker` is present. Besides, the method `getOrderedArrayItems` was added to take advantage of the ordering.

> This closes the part related on the implementation. It reviewed some elements that are brought out in the implementation, that were worth noticing and explaining to have a deeper understanding of the system.
>
> The next chapter will face the different testing strategies existing to test an iOS application.

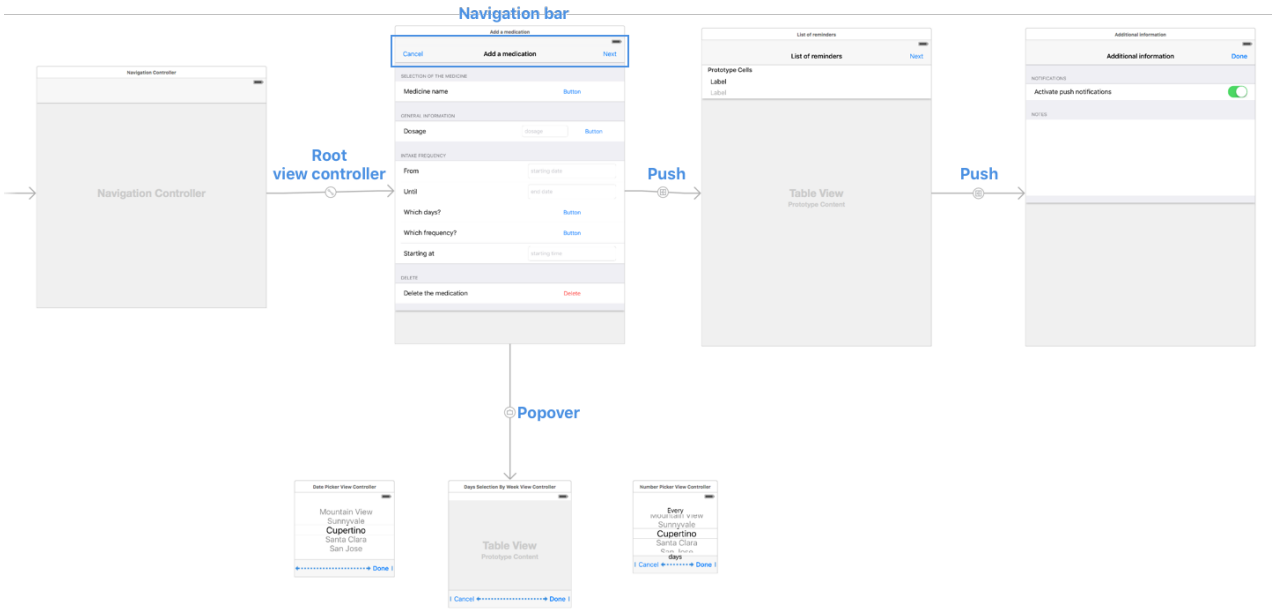
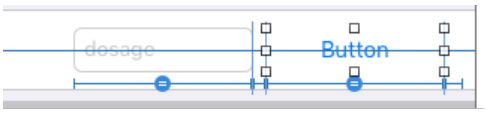


FIGURE 10.1: Storyboard of the medication compliance system UI

```
// Instantiate
let popController:UIPopoverController =
        UIPopoverController(contentViewController:
                viewControllerToDisplay)
// Set size
popController.setPopoverContentSize(size, animated: true)
// self have to conform to UIPopoverControllerDelegate
popController.delegate = self
// Display
popController.presentPopoverFromRect(elementTouched,
                inView: self.view,
                                permittedArrowDirections: .Any,
                                animated: true)
```

FIGURE 10.2: Example to programmatically present a popover



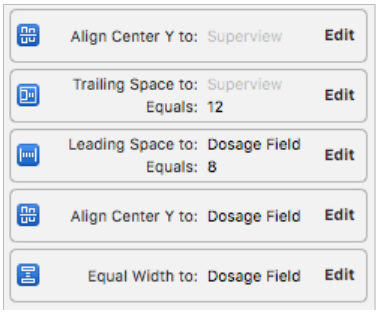


FIGURE 10.3: Example of Auto Layout utilisation

Chapter 11

Testing

One of the weaknesses of this project is that it lacks in an automated testing strategy. Indeed, the development process was composed of, after completing the researches and requirements, small iterations consisting of architecture, implementation and testing phases. This approach is suitable when dealing with new technologies that has to be learnt. Developing by small parts allows to uncover a few aspects of the technology at a time.

During those iterations, the use cases have been manually tested so that it is possible to add a notification, be notified, and so on. However, those test cases are not robust and corner cases may not have been correctly grasped. This chapter will list testing strategies that could be implemented in the future to thwart those flaws.

11.1 Testing categories

Xcode, Apple's IDE for developing applications, has a built-in support for three main types of testing [16].

11.1.1 Unit testing

Unit tests aims at testing smallest possible parts of code by writing *test cases* for "micro-features". A good policy would be to start writing test cases for the foundations of the code, for instance testing the Model before the Controllers to make sure that the basis is well tested before working on what use it.

11.1.2 Performance testing

Performance can be tested beyond the feeling that the user has when using the application. Xcode provides tools for time-based performance testing. A threshold has to be set by running the application several times, so that the next time the application will be running, it will be possible to compare timing results.

11.1.3 User Interface testing

Another way of testing is to simulate all the interactions that the user may have with the application and testing the results of those actions. This gives a high level focus to the testing approach but it is necessary since it will mimic the operations of the user and hence produces the outcomes that he will see.

Once again, Xcode allows the developer to record some scenarios on the iOS simulator by acting like a user. Afterwards, the developer puts some assertions to check if the workflow did well or not. A good policy to start with the most frequent or likely workflows. In our case, checking automatically the use cases as they are looks like a good start. Afterwards, it could be useful to add corner cases that could break the system.

11.2 Going further

Some other policies could be added in the future to not only chase the existing bugs, but also improve the application. The most suitable way to do it is by *User Testing*. Behind this generic designation resides all the methods involving the user. Amongst the processes, one can find:

- *usability testing* that records the interaction of users with the application to observe their behaviour and see if it is easy to use or not,
- *beta testing* that consists in finding testers to try the application and give a feedback on it.

> We have seen the different kind of testing that can be put in place to verify the application. As a last chapter of this part about the medication compliance system, we will go through the possible enhancements that can be done to improve the application.

Chapter 12

Enhance the system

To close this part on the medication compliance system, here are listed some improvements and new features that can be added to enhance the system and enlarge its capacities.

12.1 Improving the current system

12.1.1 iPhone compatibility

Some UI elements are not compatible with the iPhone, this is the case of the popovers used to display a view controller. One solution could be to, in the function displaying the popovers, to replace them by another view container available on iPhone that will incorporate the same view controller, e.g. a new panel sliding to display the view. Furthermore, the Auto Layout constraints specific to the iPhone should be added if need be.

12.1.2 Notification subsection

The application could display in the "Medication" section a new section containing the next notifications. In addition, it could also display the notifications that has been ignored to be able to mark them as taken or not later on, and not at the time of the intake. Going even further, it could also be possible to mute a single notification via this subsection.

12.1.3 Medication duration

For now, the user is able to set the end date of its medication. As an improvement, the application could also be able to set the duration of the medication, instead of its end date.

12.1.4 Translations

There is a translation mechanism in Andaman7 working with a key identifying the string to translate. The server contains the mapping between that key and the translations. Currently, the name of the `Ami` have that key, however, certain fields of the GUI do not use this mechanism and are written directly in English. An improvement can be to replace them using the translation mechanism.

12.2 New features

12.2.1 Graphical summary

The application could display in the "Medication" section a graph summarising the intakes that has been taken or not for different time scales, so that the user could see when the medications have been correctly followed or not.

12.2.2 Delay intake

Sometimes, when receiving the notification, the user may not have access to his medicine and may not be able to take its medication right away. One improvement would be to add a third action to delay the notification and remind the user later on.

12.2.3 Refill reminder

The user could add the current number of units of medicine left, so that the application could track how many units of medicine are left intake by intake, so that it can notify the user when it is time to refill. It could event take advantage of the iOS reminder application to do so.

12.2.4 Contraindication

Simply, a field could be added to the medicine to indicate what are the contraindications of the drug.

12.2.5 Picture

The application could have the ability to add a picture of the medicine or the pill to help the user recognize the drugs if need be. Going further, the picture of the medicine could even be displayed when receiving the notification.

> Those improvements close the part on the medication compliance system. The following chapters will address the second part of the Master thesis concerning the medicine list database integration into Andaman7.

Part II

Medicine database integration

Chapter 13

Introduction

This part of the manuscript is related to medicines. This preliminary chapter will introduce the subject and the context, as well as describing the final goal. The following chapters will portray the methodology developed to achieve it. As a conclusion, the procedure will be commented and criticise.

13.1 Context and goal

In the current version of Andaman7, the drugs are managed by the user that has the ability to add, modify and remove medicines. They are defined using the `Ami` structure (cfr. Section 6.1). The FIGURE 13.1 shows the panel that can be used to add a drug.

The goal of this part of the thesis is to change this management in such a way that the medicine list will be pre-populated from an external source. The purpose is to have a centralized administration of the drugs, that thus should be encoded by A7 Software to be available for every Andaman7 user. However, it is important to let the opportunity to the user to add its own medicines. Indeed, the external source, whatever it may be, might not cover all medicines, it could exclude, for instance, what a user would like to consider as a medicine but is not officially registered as such, from dietary supplement to experimental drug.

This also raises the problem of the source of those drugs. With Andaman7 not targeting a single country, the internationalisation of the database is not a trivial problem. There is currently no such thing as a common international database covering, accurately, all medicines. This is another reason to let the user add its own drugs.

13.2 Scope of the work

This part about the medicine list was actually the first in time work produced in the scope of the thesis. It acts as a reflection on the methodology that can be used to populate the medicine database. The procedure has been implemented on the server but was not entirely tested, as my work

changed direction before I was able to finish it. The module is capable of retrieving the data but not insert them into the Andaman7 database.

> After having set the context and the goal, the first step is to determine which external source will be used to populate the database. That will be the topic of the next chapter.

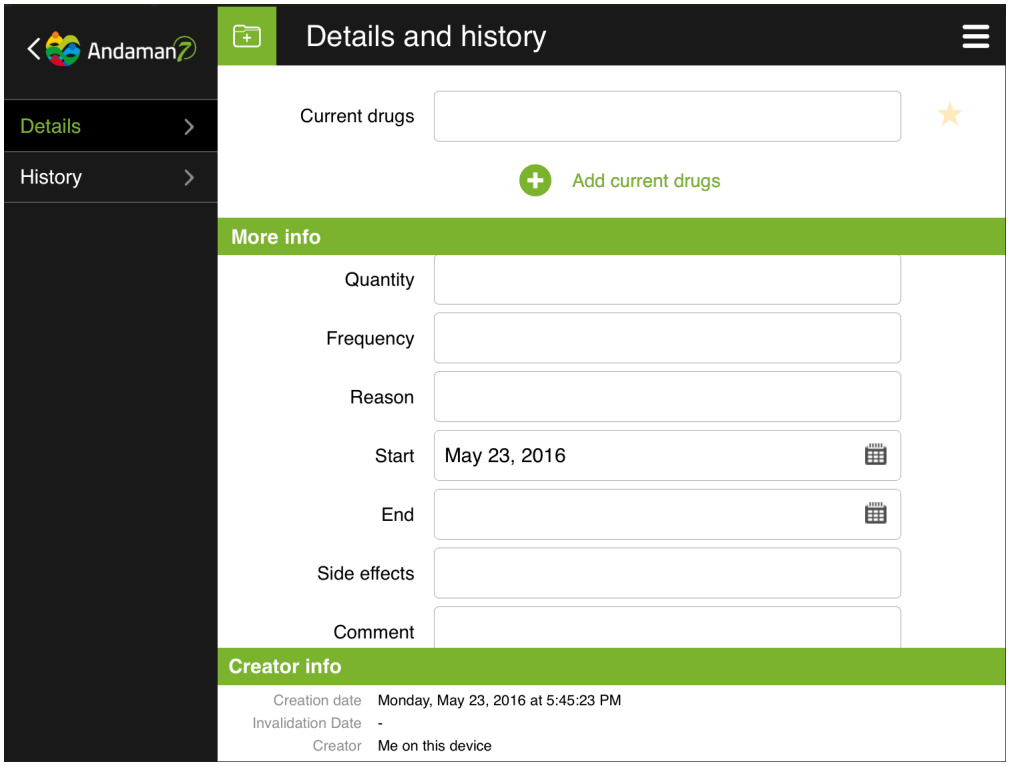


FIGURE 13.1: Panel to add a drug

Chapter 14

CBIP database

This chapter will describe the source used to have a list of drugs, from the selection to its organisation and how it is structured.

14.1 Selection

The selection process did not last long, the need for a database was quite basic. In a first analysis, the database has to be targeting Belgium only. It has been decided to use the CBIP/BCFI database from the start as it seemed to fulfil the needs by its completeness.

14.1.1 CBIP/BCFI

The CBIP (*Centre Belge d'Information Pharmacothérapeutique*, or BCFI *Belgisch Centrum voor Farmacotherapeutische Informatie*) is, according to [13], non-profit association "assuring the ongoing training as well as the update of the knowledge in the field of pharmacotherapy of the physicians, pharmacists and dentists". It is also worth the note that the association is subsidized by the Belgian authorities and that the medical data comes from the INAMI.

The database given by the CBIP is freely available on demand for health professionals and is available for consultation online. Please note also that the version of the database used is from September 2016.

14.2 Organisation

The entries of the database are given in CSV files, one file corresponding to one table. The scheme of the database is shown in FIGURE C.1 in APPENDIX.

It is composed of the following tables [22]:

- **MP**: List of all medicinal products. i.e. brand names.
- **MPP**: List of all medicinal product packages.
- **INNM**: All active ingredients at least present in one medicinal product.

- **SAM**: Composition of each medication package. It links together **MPP** and **INNM**.
- **IR**: Entities responsible for the information about the Medicinal Products.
- **GAL**: Galenic terms[1] for the medication package.
- **GGR_LINK**: Links to the CBIP website.
- **HYR**: Hierarchical tree of the current repertoire listing the different fields (e.g. cardiology, infections,...).

> This chapter described the data received from the CBIP and how it is organised. The next chapter will deal with how it will be used to integrate them into Andaman7.

[1] The galenic form is the form under which the constitutive elements (ingredients) are conserved, e.g. cream, pill, and so on.

Chapter 15

Integration

The focus on this chapter will be on how the data can be used and processed to be exploited in Andaman7.

15.1 Overview

A high-level view of the procedure of fetching the data and processing it, from the beginning to the end, is represented in FIGURE 15.1.

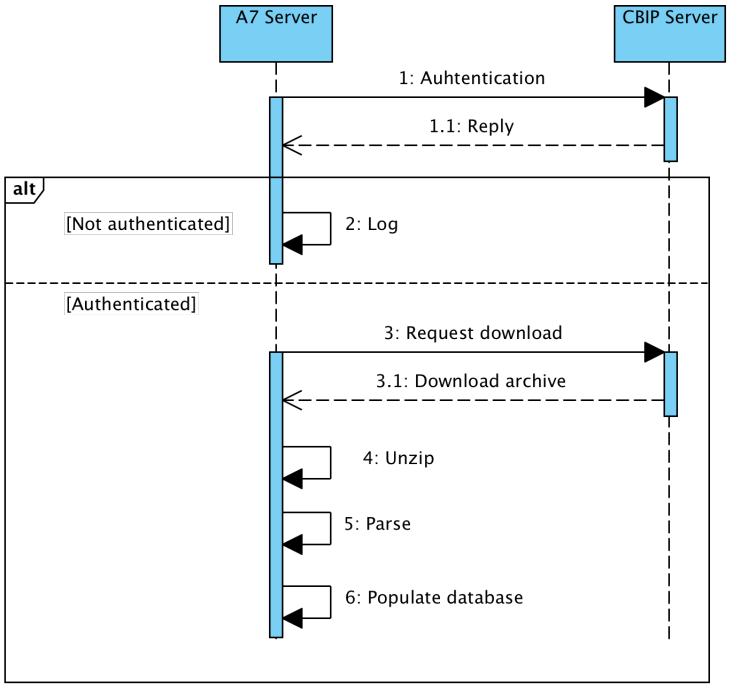


FIGURE 15.1: Sequence diagram of the procedure to fetch and process the data from CBIP

The communication diagram in FIGURE 15.2 also shows the objects, from the Java classes implemented, involved in the process.

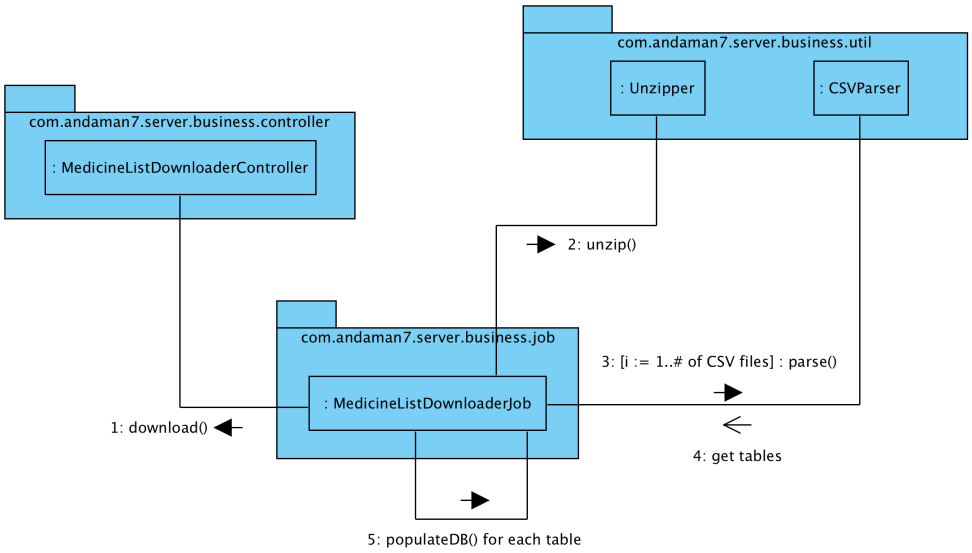


FIGURE 15.2: Communication diagram of the objects

15.2 Fetching

To use the data given by the CBIP, it first needs to be fetched. One solution is to do it manually and then upload it to the server for further processing. However, since the CBIP changes their data every month, the process needs to be done regularly. Furthermore, it means that someone has to do those operations. Another solution is to automate the procedure and download automatically, at a given frequency, the data from the Andaman7 server.

To download the data regularly, a `Cron` job was scheduled on the server using a Java Spring annotation.

```
@Scheduled(fixedDelay = DELAY)
public void downloadRepertoire(){
        ...
}
```

As healthcare professional, an account is needed to download the database. The first step is thus to connect to the system via a `POST` request containing the credentials to obtain a authentication token. Afterwards, a `GET` request can be send to the system to retrieve the medicine list.

The file received is an `zip` archive, the data thus needs to be extracted. A dedicated class has been created to do so. Once unzipped, we obtain 8 CSV files, one per table.

15.3 Parsing

Using `CSVParser` from Apache, the CSV files are parsed. However, instead of retrieving all the columns, some of them are removed because of the lack of information they bring. The resulting database scheme is shown in FIGURE 15.3. It changes from FIGURE C.1 in APPENDIX in several ways:

- It contains a few less fields that are less important,
- Some translation fields are added to manage multiple languages,
- **WadaCode** table was added, containing the WADA (World Anti-Doping Agency) code and their description. It is actually described in [22] however, to handle multiple languages, it has moved into the database.

Once the data parsed, there are saved into the Andaman7 database.

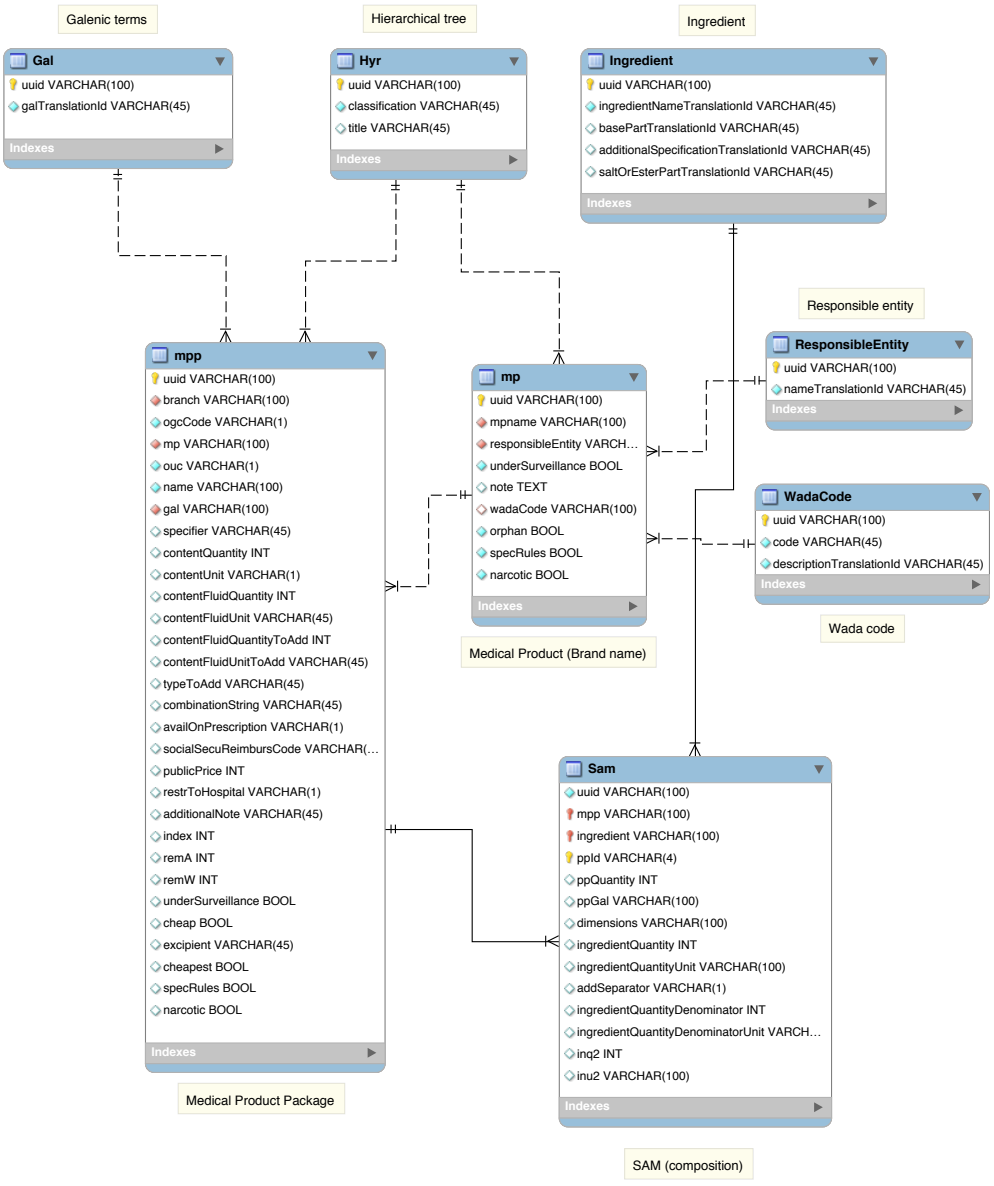


FIGURE 15.3: Actual medication database scheme

15.4 Comments

The choice of the CBIP database is sufficient for a national scope, it contains a lot of data. However, for an international targeting, it does not seem to be a good reference as the data may not be organised the same way in other

countries. Nonetheless, for further analysis, it seems acceptable to rely on a model containing medicinal products and their variant, medicinal product packages, as well as the active principles that they contain.

Also, the scope stops at populating the Andaman7 database. Naturally, the list needs to be sent to the mobile devices. A synchronisation mechanism exists in parallel to the synchronization of the medical data which actually interacts with web services on the server. This mechanism can be used to automatically download the medicines on the application.

> This concludes the part on retrieving a list of medicines from an external source and analysing how it can be integrated into Andaman7, in order to enable the user to have a catalog of drugs. The next part will conclude this manuscript.

Part III

Conclusions

Chapter 16

Conclusion of this report

The thesis was divided in two parts. Throughout the first part on the medication compliance system, we have seen what is medication noncompliance, its consequences and what are some possible solutions to it, specially how a medication compliance system integrated to Andaman7 can help. To complete this research part, the existing medication reminder applications were briefly tested and compared.

Afterwards, we reviewed the requirements for such a system to see what is was supposed to do and how it was supposed to be used. To complete that, the use cases of the main scenarios were also described.

The core parts concerned the architecture, first about the tools and structure provided by Andaman7 which had to be used, and then how the medication compliance system was designed to fit into this architecture as well as other necessary components.

After having reviewed how development for iOS works, the other main topic was the GUI, what we needed and what was the process to end up with the current GUI, going through the existing solution in Andaman7 and explaining how it was inadequate.

Subsequently, the implementation of the system was explained through the elements that needed more explanation. From there, we reviewed possible testing strategies that could be set up to increase the robustness of the system. As a last chapter for this part, some enhancements were listed that could be useful for future development.

The second part of this Master thesis concerned the integration of a medicine database into Andaman7. We saw why it was needed, how was structured the selected database, as well as a proof of concept of the integration procedure using a Java implementation on the Andaman7 server to retrieve the database.

The main obstacle I faced during the development was the lack of documentation. Indeed, there was no comments in the code nor files describing

the architecture, which made the learning curve of the existing system quite steep and thus time-consuming. As an upside, this made me completely go through the code I needed from their usage in other parts of the application, which made me understand the code by debugging it and have a deeper insight.

This project was a great opportunity for me and brought me a rich experience. It first allowed me to learn how to design and develop an iOS application, a topic in which I had no experience. Also, it was a challenging project, entering into another team's project and work on it can be tedious at first but is also rewarding. As a personal opinion, I think that the two parts of the resulting work form a pretty good first step to a complete medication compliance system since it already provide the main features for such a system integrated to Andaman7.

Part IV

Appendices

Appendix A

Medication compliance applications comparison

A.1 Screenshots

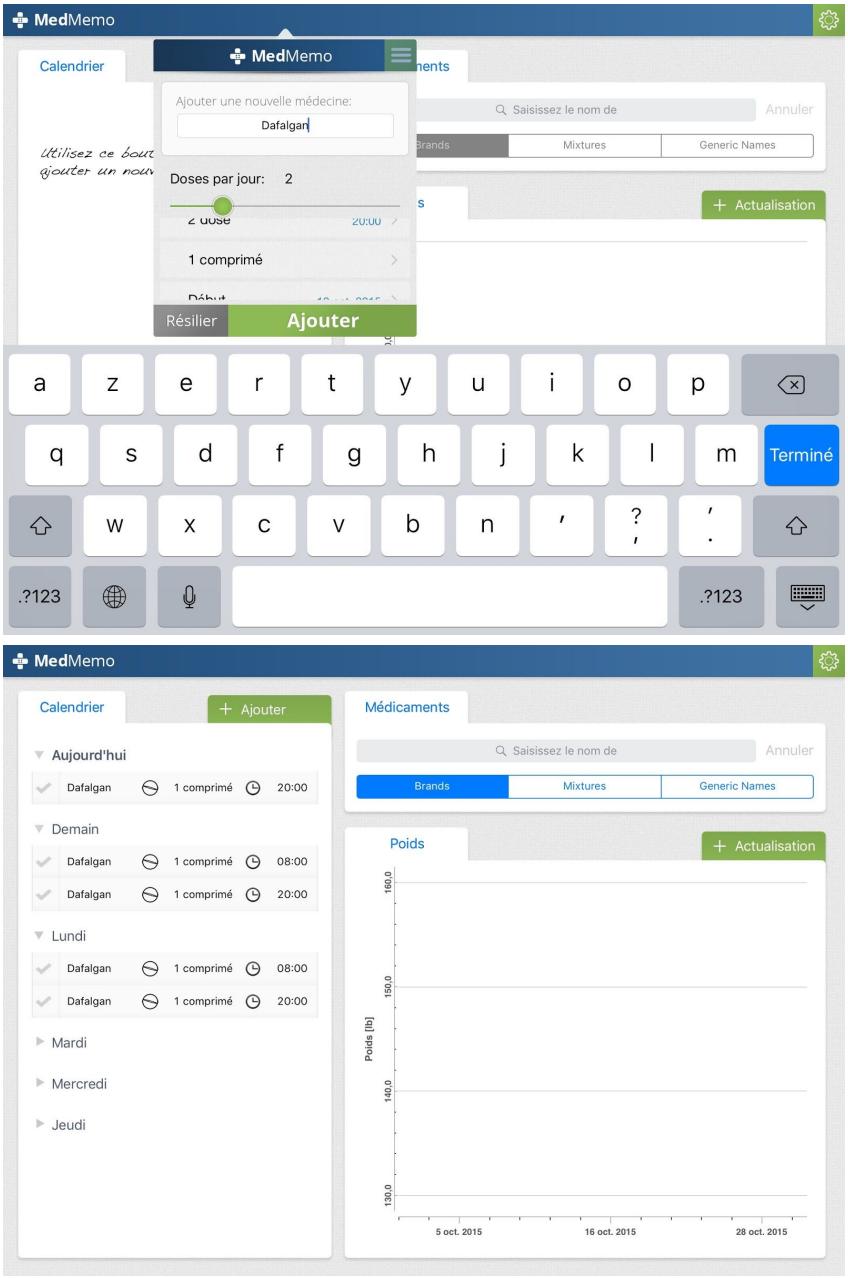


FIGURE A.1: Medication compliance application: *MedMemo*

FIGURE A.2: Medication compliance application: *Mon Agenda de Médication*

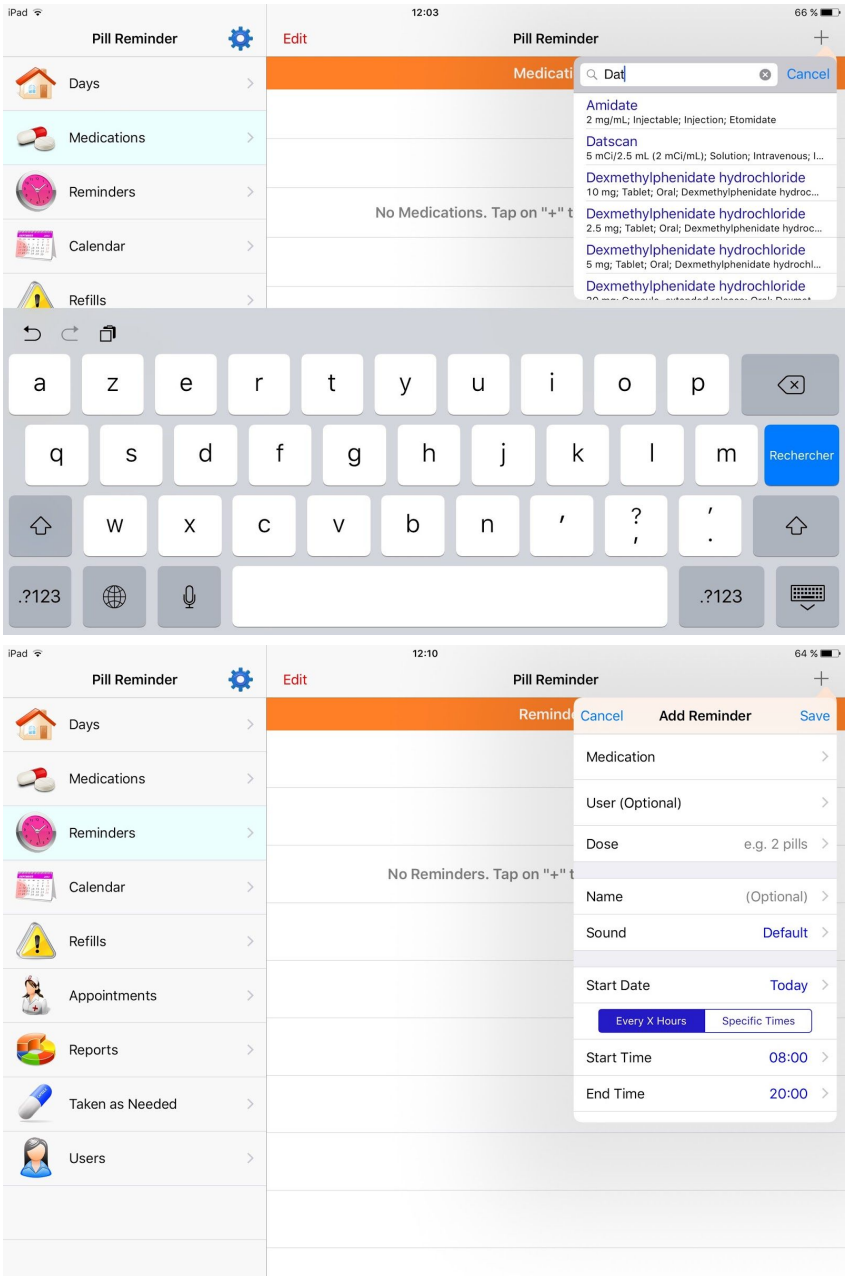


FIGURE A.3: Medication compliance application: *Pill reminder*

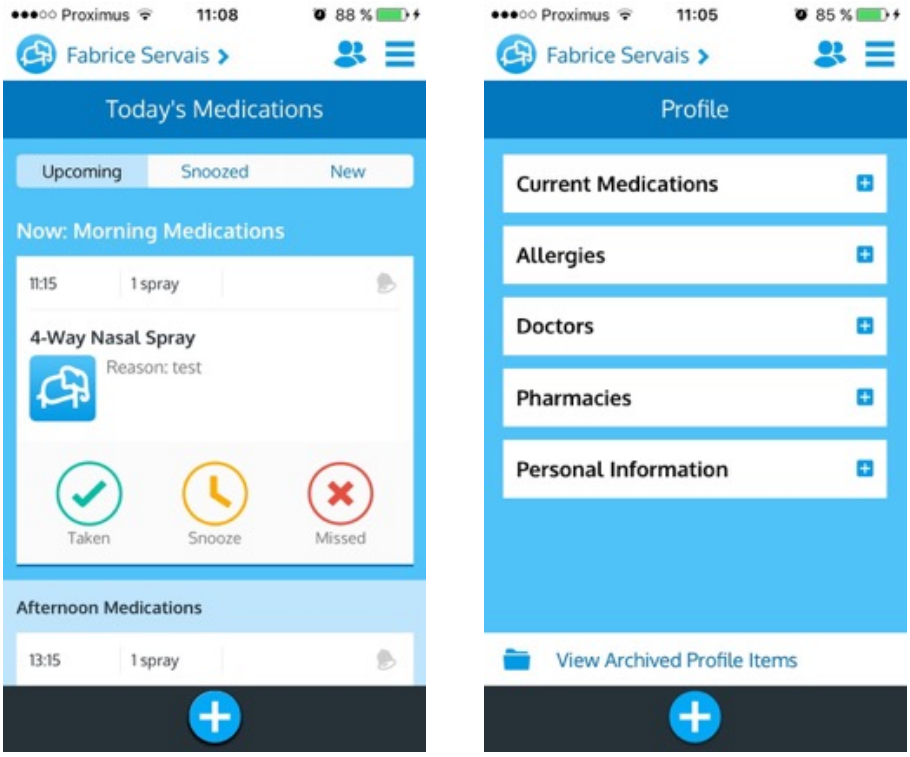


FIGURE A.4: Medication compliance application: *My Meds*

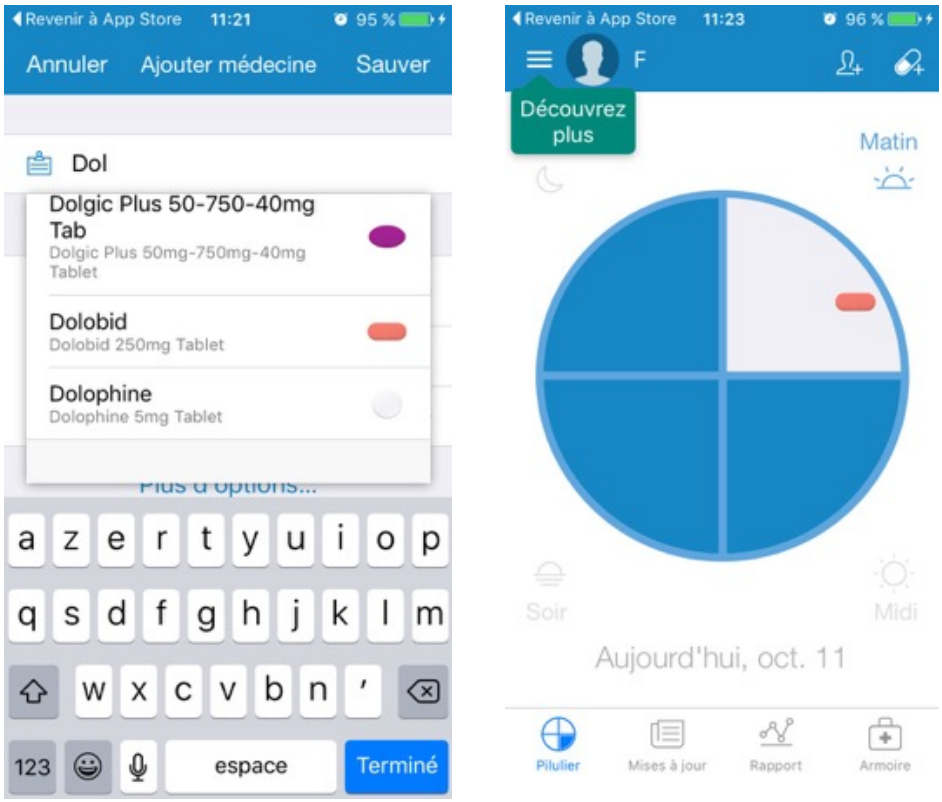


FIGURE A.5: Medication compliance application: *MediSafe*

A.2 Comparison from *pharmacist.com*

TABLE A.1: Comparison of different medication compliance applications on mobile devices from *pharmacist.com* [19]

Application	Compatibility	Cost	Features
Dosecast	Both	Free	• Offers customizable dose amounts and instructions for multiple patients • Allows reminder postponing and "Smart Silencing" feature • Tracks dose history and adherence, provides refill alerts, shares information with provider and pharmacy
Mango Health	iOS	Free	• Records timing and dosing for medications and supplements with personal health journal • Offers reminder alerts, including notifications of potential interactions with medications, food, or drink • Allows users to earn points for real-world rewards (e.g., from Target)
MedCoach	Both	Free	• Shares medication history and refill times with pharmacist and primary provider • Presents visual reminders (alerts are triggered one time only, not continuously) • Offers immediate access to customer service agents and ability to call 911

MediSafe	Both	Free	• Synchronizes information to a "family pillbox" • Shares information with caretakers who can be notified if patient has not checked into the app • Shows daily medication list that can be checked off throughout the day
Medi-Prompt	iOS	$ 1.99	• Offers dynamic scheduling with maximum or minimum daily doses • Works with multiple patients and multiple medications; includes password protection • Offers scheduled or ad-hoc doses, calculated doses, regular intervals, or combinations • Last dose
MedMory	iOS	Free	• Sets visual and audio reminders for individual medications, including as-needed or unscheduled • Tracks medication quantities, refill reminders, and adherence history • Alarm may continue after taking medications; alerts may sound at wrong times
MyMedSchedule	Both	Free	• Records pictures and notes for medications with daily reminders

MyMeds	Both	Free	• Tracks medication history, allergies, and immunizations for entire family with customizable reminders • Shares information with pharmacist and primary provider and locates nearby pharmacies • Requires registration at www.my-meds.com for use
PillMonitor	iOS	Free	• Schedules reminder time, repeat date, and dosage with photos, notes, and history for each medication • Shares medication list and history to primary care provider via e-mail
RxCase Minder	Android	Free	• Stores information about multiple patients, caregivers, pharmacists, and primary care providers • Available in Spanish
RxmindMe	iOS	Free	• Nine types of alerts that recur until dismissed • Tracks remaining quantity, time to refill, and adherence history with password protection • Exports data and history, saves medication pictures, and accesses FDA medication database

Appendix B

Ami

B.1 Types

The types of Ami or qualifier are listed in TABLE B.1.

Type	Description	Example
`string`	String describing the property.	Allergy, vaccine,...
`number`	Numerical property.	Weight, height, BMI,...
`date`	Date property.	Consultation date,...
`document`	Document in the application. The user has the ability to send a document to the application and use it in an Ami of type `document`.	Prescriptions, test results,...
`note`	Note describing the property. Will be displayed differently than `string`, with a bigger area to write text.	Consultation note, personal note,...
`oneSelection`	The property takes its value in the referenced `SelectionList`.	Drinker type, units, sex,...
`multiSelection`	The property takes multiple values in the referenced `SelectionList`	Days,...

TABLE B.1: Types of Ami and qualifier

B.2 Markers

The different markers of Ami are listed in TABLE B.2.

Marker	Description	Example
`marker.multi`	Allows the application to have multiple values of a single Ami at the same time.	Allergies, doctors,...
`marker.hidden`	...	...
`marker.cumulative`	...	...
`marker.deviceSyncOnly`	...	...
`marker.sharing.never`	...	...

TABLE B.2: Markers of Ami

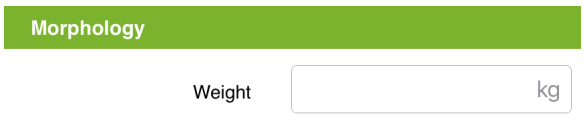
B.3 Examples

Two examples of definition of Ami are shown in FIGURE B.1 and FIGURE B.2, and one example of a selection list is shown in FIGURE B.3.

In all the figures, at the top is written the XML code while at the bottom is the corresponding result in the application.

```
<Ami id="ami.weight"
       type="number"
       tags="tag.biometry, tag.consultation">
      <Qualifier id="qualifier.date"
                     type="date" >
            <DefaultValue>[currentDate]</DefaultValue>
      </Qualifier>
      <Qualifier id="qualifier.unit"
                        type="oneSelection"
                        selectionListId="sl.unit.mass" />
</Ami>
```

(A) XML code



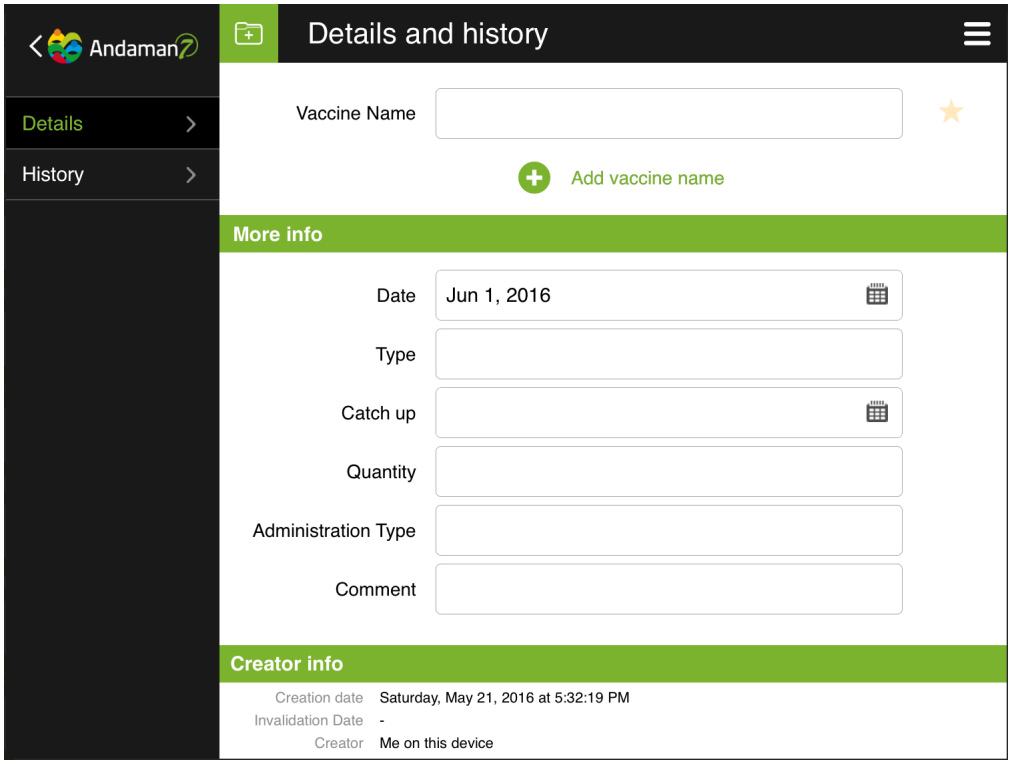
(B) Result in the application

FIGURE B.1: Ami example: *Weight*

```
<Ami id="ami.vaccine" type="string" tags="tag.drug">
      <Qualifier id="qualifier.date" type="date">
            <DefaultValue>[currentDate]</DefaultValue>
      </Qualifier>
      <Qualifier id="qualifier.type" type="string" />
      <Qualifier id="qualifier.catchup" type="date" />
      <Qualifier id="qualifier.quantity" type="string" />
      <Qualifier id="qualifier.administrationType" type="string" />

      <Marker id="marker.multi" />
</Ami>
```

(A) XML code

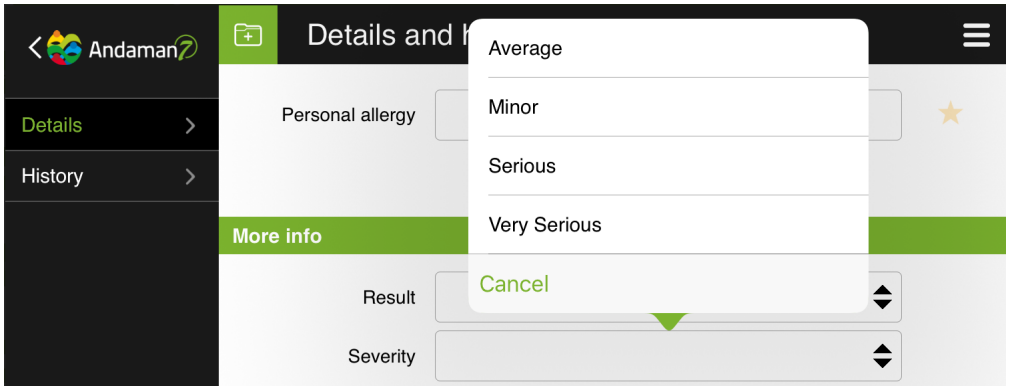


(B) Result in the application

FIGURE B.2: `Ami` example: *Vaccine*

```
<SelectionList id="sl.allergySeverity">
      <Item id="li.serious" />
      <Item id="li.soft" />
      <Item id="li.verySerious" />
      <Item id="li.average" />
</SelectionList>
```

(A) XML code



(B) Result in the application

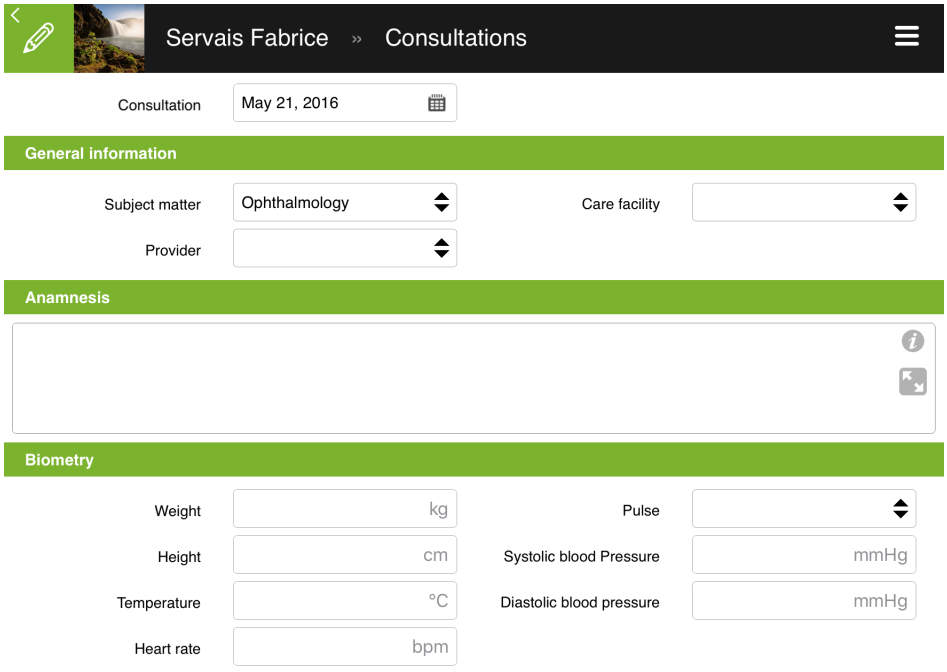
FIGURE B.3: `SelectionList` example: *Allergy severity*

```
<AmiSet id="amiset.consultation" type="date" tags="tag.consultation">
   <DefaultValue>[currentDate]</DefaultValue>
   <Marker id="marker.multi" />

   <Qualifier id="qualifier.subjectMatter" type="oneSelection"
            selectionListId="sl.subjectMatter" />
   <AmiRef id="amiref.doctor.executor" refId="ami.doctor" />
   <AmiRef id="amiref.careFacility" refId="ami.careFacility" />

   <AmiRef id="amiref.additionalExam" refId="ami.additionalExam">
      <Marker id="marker.template" />
      <Marker id="marker.multi" />
   </AmiRef>
   ...
</AmiSet>
```

(A) XML code



(B) Result in the application

FIGURE B.4: `AmiSet` and `AmiRef` example: *Consultation*

Appendix C

Medicine database integration

C.1 CBIP database

The scheme of the database given by the CBIP is shown in FIGURE C.1.

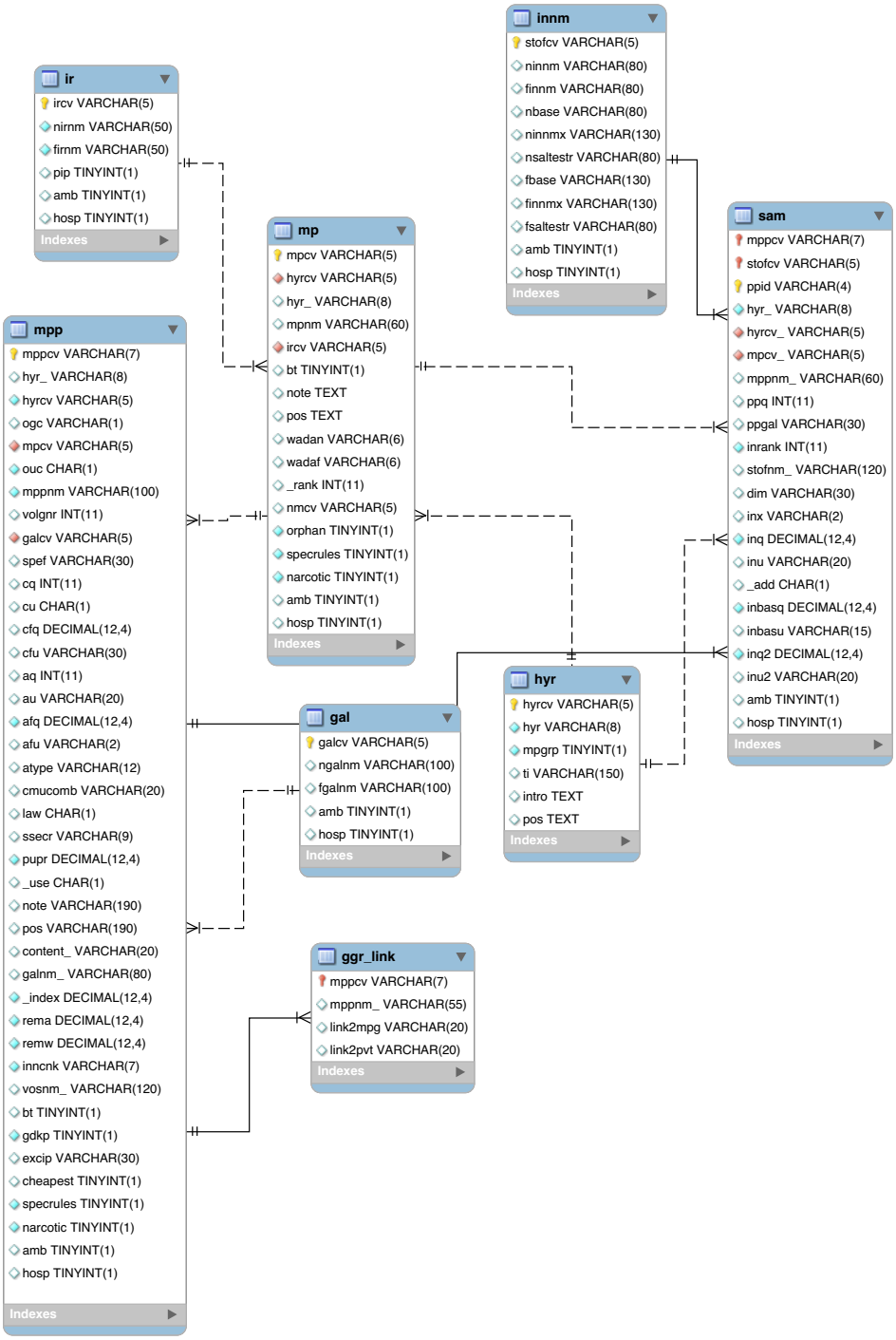


FIGURE C.1: Scheme of the database given by the CBIP (September 2015)

Bibliography

[1] Martin Fowler. "Recurring events for calendars". In: *URL http://martinfowler.com/apsupp/recurring.pdf. Affiliation: Garrett Grolemund Rice University Houston, TX* (1997), pp. 77251–1892.

[2] Albert I Wertheimer and Thomas M Santella. "Medication Compliance Research: Still So Far To Go." In: *Journal of Applied Research* 3.3 (2003).

[3] Lars Osterberg and Terrence Blaschke. "Adherence to Medication". In: *New England Journal of Medicine* 353.5 (2005). PMID: 16079372, pp. 487–497. DOI: 10.1056/NEJMra050100. eprint: http://dx.doi.org/10.1056/NEJMra050100. URL: http://dx.doi.org/10.1056/NEJMra050100.

[4] *The American Heritage® Medical Dictionary*. 2007. URL: http://medical-dictionary.thefreedictionary.com.

[5] *AstraZeneca Digs Into the Cause of Non-adherence*. 2008. URL: http://www.pharmexec.com/astrazeneca-digs-cause-non-adherence.

[6] Hayden B Bosworth. "How can innovative uses of technology be harnessed to improve medication adherence?" In: *Expert review of pharmacoeconomics & outcomes research* 12.2 (2012), pp. 133–135.

[7] Brian Fung. *The 289 Billion Cost of Medication Noncompliance, and What to Do About It*. 2012. URL: http://www.theatlantic.com/health/archive/2012/09/the-289-billion-cost-of-medication-noncompliance-and-what-to-do-about-it/262222/.

[8] James Rickert. *Patient-Centered Care: What It Means And How To Get There*. 2012. URL: http://healthaffairs.org/blog/2012/01/24/patient-centered-care-what-it-means-and-how-to-get-there/.

[9] Lindsey Dayer et al. "Smartphone medication adherence apps: potential benefits to patients and providers". In: *Journal of the American Pharmacists Association* 53.2 (2013), pp. 172–181.

[10] Matthijs Hollemans. *Storyboards Tutorial in iOS 7: Part 1*. 2013. URL: https://www.raywenderlich.com/50308/storyboards-tutorial-in-ios-7-part-1.

[11] Apple Developer. *iOS Human Interface Guidelines*. 2014. URL: https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/.

[12] Inc Apple. *Cocoa Core Competencies - Delegation*. 2015. URL: https://developer.apple.com/library/ios/documentation/General/Conceptual/DevPedia-CocoaCore/Delegation.html.

[13] CBIP/BCFI. *CBIP/BCFI website*. 2015. URL: http://www.cbip.be/fr/about.

[14] Paul Hegarty. "Developing iOS 8 Apps with Swift". In: *Lecture notes, University of Stanford, iTunesU* (2015).

[15] A7 Software. *Andaman7 developer portal*. 2015. URL: http://developers.andaman7.com/.

[16] Inc Apple. *Xcode overview*. 2016. URL: https://developer.apple.com/library/ios/documentation/ToolsLanguages/Conceptual/Xcode_Overview/.

[17] Antonio Bello. *iOS User Interfaces: Storyboards vs. NIBs vs. Custom Code*. 2016. URL: https://www.toptal.com/ios/ios-user-interfaces-storyboards-vs-nibs-vs-custom-code.

[18] *Adult Meducation, Improving Medication Adherence in Older Adults*. URL: http://www.adultmeducation.com/downloads/adult_meducation.pdf.

[19] *Medication adherence: There's an app for that*. URL: http://www.pharmacist.com/medication-adherence-there%E2%80%99s-app.

[20] *Patient Adherence– 50% of patients don't take their medicine properly*. URL: http://www.efpia.eu/topics/people-health/patient-adherence.

[21] European Council Policy makers Debate. *An EU response to medication non-adherence.* Brussels, 2010.

[22] CBIP/BCFI. *DB4EMD - Description of the database*. September 2015. URL: http://www.cbip.be/assets/pdf/db4emd-docEN.pdf.