Annex

Index

Figures

1. Annex: Use Cases Diagrams

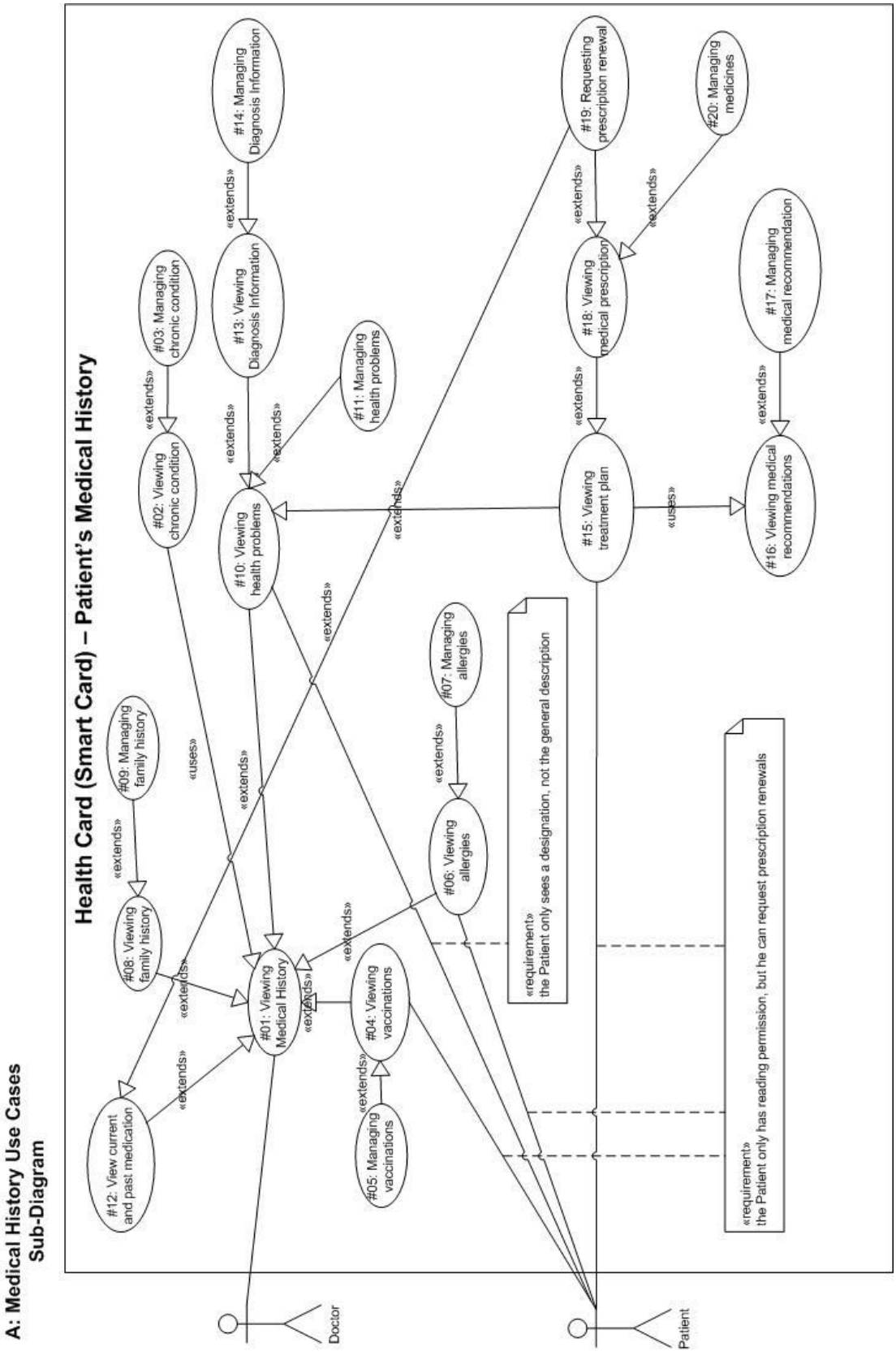

Figure i. Medical History Use Cases Sub-Diagram

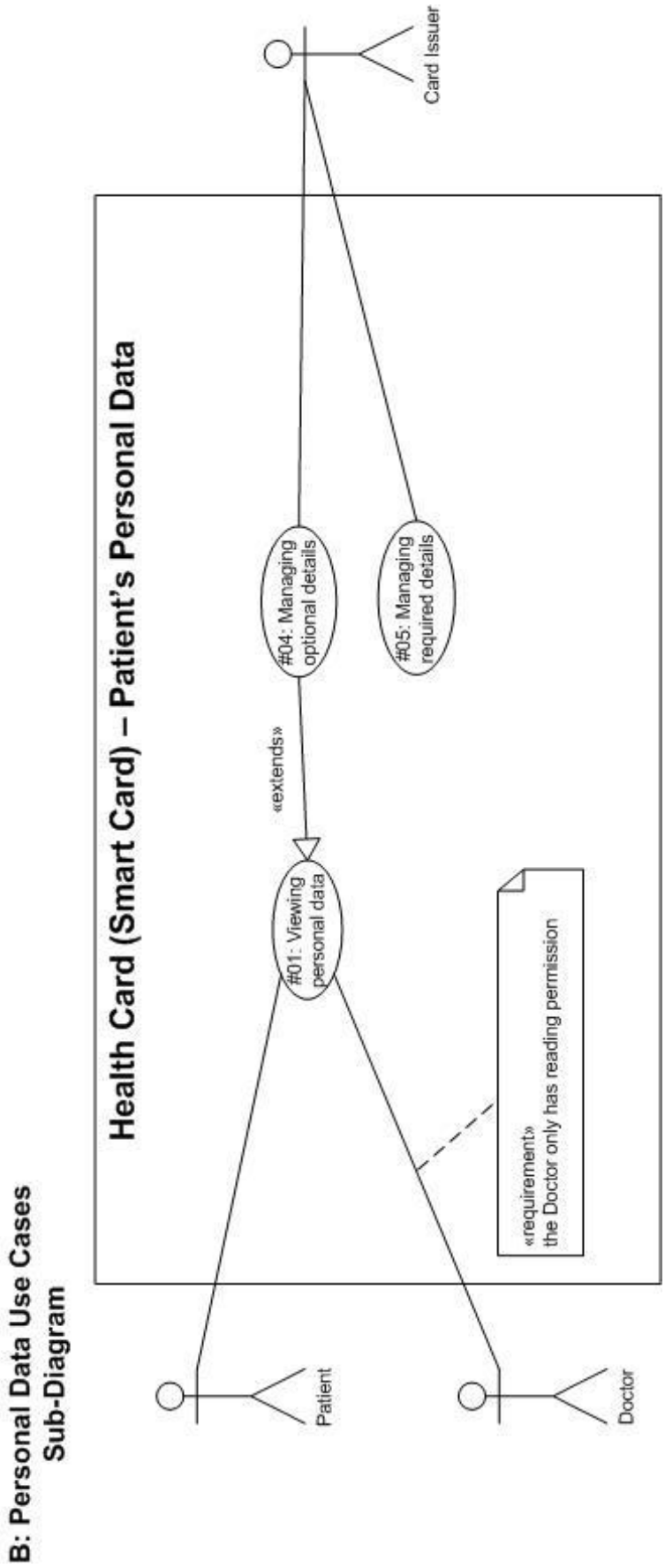

Figure ii. Personal Data Use Cases Sub-Diagram

C: Appointments Use Cases Sub-Diagram

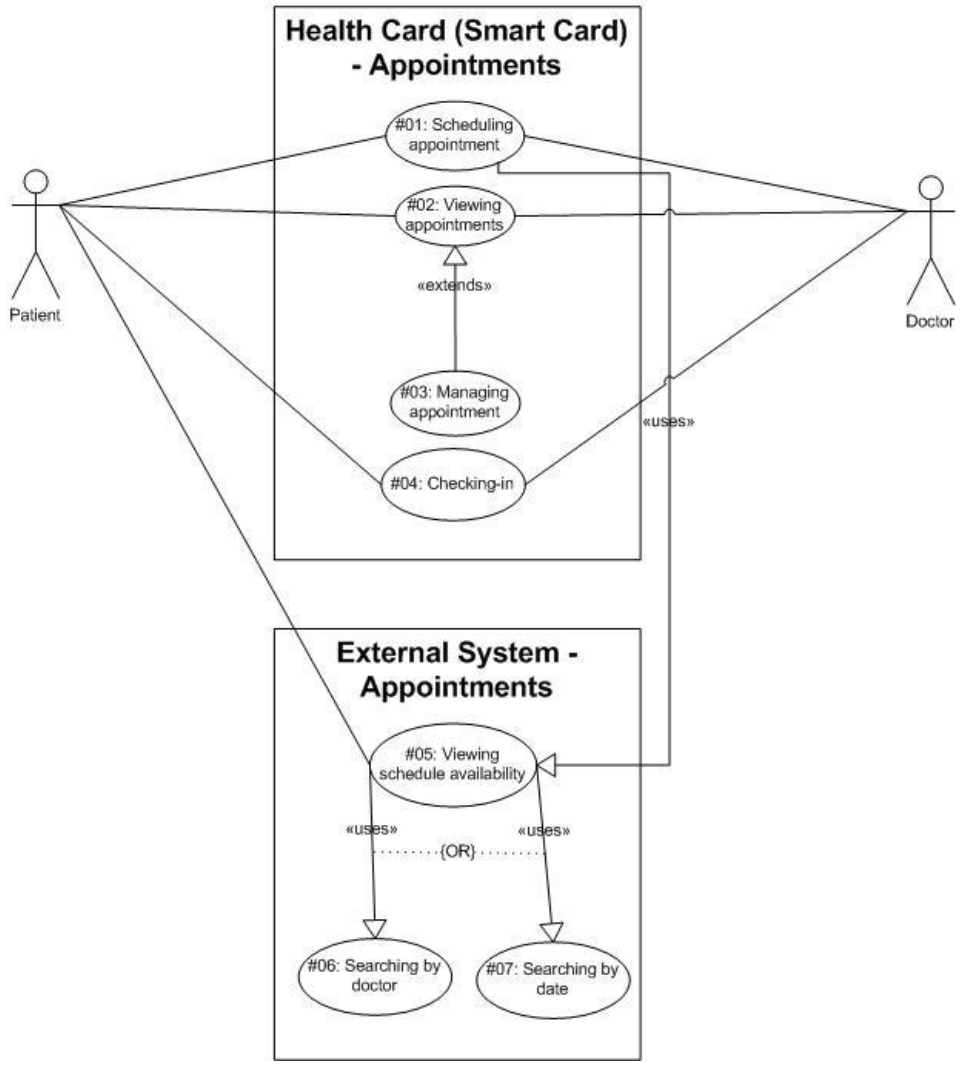

Figure iii. Appointments Use Cases Sub-Diagram

D: System Administration Use Cases Sub-Diagram

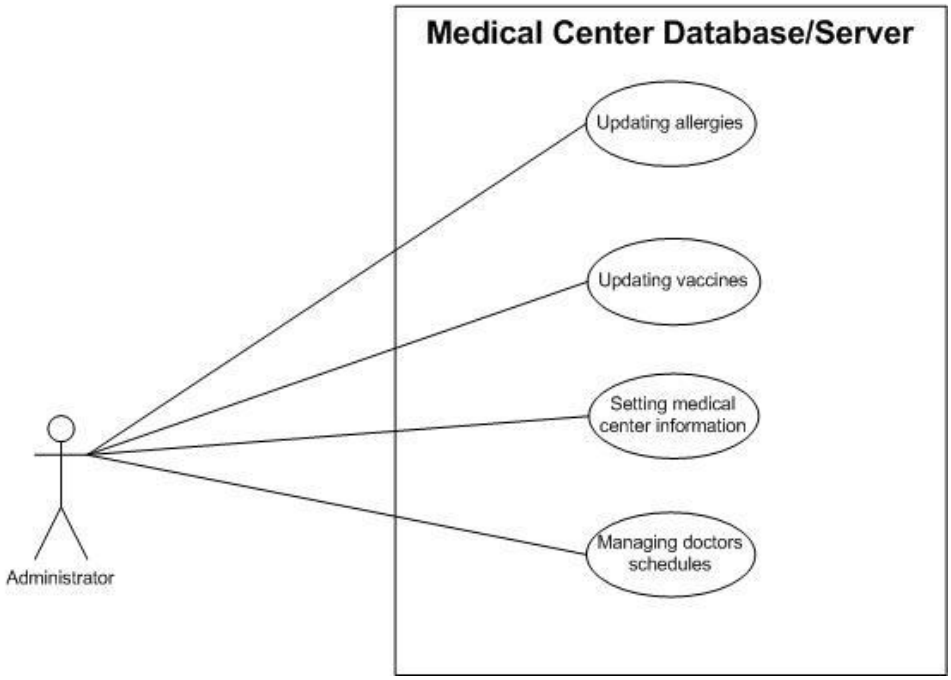

Figure iv. System Administration Use Cases Sub-Diagram

2. Annex: Use Cases Textual Specification

A: Medical History Use Cases

A01: Viewing Medical History

Name: ***Viewing Medical History***	**ID:** ***A01***
Main Scenario This use case initiate after successfully validating the owner's card and selecting to view the patient's medical history. The system will show a summary of the patient's medical history. The most recent allergies, vaccinations and health problems will be shown ordered chronologically. Besides that, the system will show the patient's chronic conditions. This use case includes the Viewing Chronic Condition use case. Also the system offers a way to view in detail the health problems history, the patient's allergies and vaccinations. Here the user has the ability to choose one of the options offered by the system. When the user chooses one of the options, the system expands in the screen the related view.	

A02: Viewing Chronic Condition

Name: ***Viewing Chronic Condition***	**ID:** ***A02***
Main Scenario (the user is the doctor) This use case is included in the Viewing Medical History use case. The user can read what chronic conditions the patient has (for example, diabetes, pregnancy, osteoporosis, asthma etc.). The system will show to the user the chronic conditions as a list, in which the chronic conditions are represented with a designation and the date that the problem was diagnosed. Also, the system gives the option to manage this list. From this task the user can choose to manage the user patient chronic conditions.	
Alternative Scenario 1 (the user is the patient) The user can't manage his chronic conditions. So the system won't give the option to manage his chronic conditions.	

A03: Managing Chronic Condition

Name: ***Managing Chronic Condition***	**ID:** ***A03***
Main Scenario This use case extends the Viewing Chronic Condition use case. The user selects to manage the patient's chronic condition, and then the system gives to the user the ability to manage the chronic condition information of the patient. From here the user can insert, modify or remove a patient's chronic condition data.	
Alternative Scenario 1 (the user chooses to insert) The user in this use case chooses to insert, then the system offers a way for the user to insert the date and a designation to the chronic condition. The user inserts the data. After that, the user confirms the insertion. The system stores the insertion into the card with an appointment identification value associated.	
Alternative Scenario 2 (the user chooses to modify) The user in this use case selects a chronic condition and chooses to modify it. The system offers a way to change the date and the condition designation. The user makes the changes and confirms the modification. Then, the system stores the modifications into the card with an appointment identification value associated.	

Alternative Scenario 3 (the user chooses to remove) The user in this use case selects a chronic condition and chooses to remove it. The system temporarily removes the chronic condition leaving the ability to the user to recover it until the he exits the chronic condition management.
Alternative Scenario 3.1 (the user chooses to recover a removed item) After removing a chronic condition, the user decides to recover it and chooses to recover. The system changes the item status and won't remove anymore if the user exits the chronic condition management.

A04: Viewing Vaccinations

Name: *Viewing Vaccinations*	**ID: *A04***
Main Scenario (the user is the doctor) This use case extends the Viewing Medical History use case. The user selects to view the patient's vaccinations. Then the system shows a list of vaccines administrated to the patient. That list contains the date of the administration and the vaccine designation, and is ordered chronologically. Also, the system offers abilities to manage the vaccinations list. The user can read the patient's vaccinations history. From this task the user can manage the vaccinations.	
Alternative Scenario 1 (the user is the patient) The user can't manage his vaccinations list. So the system won't give the option to manage his vaccinations list.	

A05: Managing Vaccinations

Name: *Managing Vaccinations*	**ID: *A05***
Main Scenario This use case extends the Viewing Vaccinations use case. The user selects to manage the patient's vaccinations, and then the system gives to the user the ability to manage the vaccinations information of the patient. From here the user can insert or remove a patient's vaccination.	
Alternative Scenario 1 (the user chooses to insert) The user in this use case chooses to insert, then the system offers a way for the user to insert the date and a designation to the vaccine. The user inserts the data. After that, the user confirms the insertion. The system stores the insertion into the card with an appointment identification value associated.	
Alternative Scenario 2 (the user chooses to remove) The user in this use case selects a vaccination and chooses to remove it. The system temporarily removes the vaccination leaving the ability to the user to recover it until the he exits the vaccinations management.	
Alternative Scenario 2.1 (the user chooses to recover a removed item) After removing a vaccination, the user decides to recover it and chooses to recover. The system changes the item status and won't remove anymore if the user exits the vaccinations management.	

A06: Viewing Allergies

Name: *Viewing Allergies*	**ID: *A06***
Main Scenario (the user is the doctor) This use case extends the Viewing Medical History use case. The user selects to view the	

patient's allergies. Then the system shows a list of allergies of the patient. That list contains the date of when the allergy was identified and the allergy designation, and is ordered chronologically. Also, the system offers abilities to manage the allergies list. The user can read the patient's allergies history. From this task the user can manage the allergies.
Alternative Scenario 1 (the user is the patient) The user can't manage his allergies list. So the system won't give the option to manage his allergies.

A07: Managing Allergies

Name: *Managing Allergies*	**ID: *A07***
Main Scenario This use case extends the Viewing Allergies use case. The user selects to manage the patient's allergies, and then the system gives to the user the ability to manage the allergies information of the patient. From here the user can insert or remove a patient's allergy.	
Alternative Scenario 1 (the user chooses to insert) The user in this use case chooses to insert, then the system offers a way for the user to insert the date and a designation to the allergy. The user inserts the data. After that, the user confirms the insertion. The system stores the insertion into the card with an appointment identification value associated.	
Alternative Scenario 2 (the user chooses to remove) The user in this use case selects an allergy and chooses to remove it. The system temporarily removes the allergy leaving the ability to the user to recover it until the he exits the allergies management.	
Alternative Scenario 2.1 (the user chooses to recover a removed item) After removing an allergy, the user decides to recover it and chooses to recover. The system changes the item status and won't remove anymore if the user exits the allergies management.	

A08: Viewing Family History

Name: *Viewing Family History*	**ID: *A08***
Main Scenario (the user is the doctor) This use case extends the Viewing Medical History use case. The user selects to view the patient's family history. Then the system shows the family history of the patient. The family history is shown as a list that contains in each line the family member degree and this health problem which could be inherited by the patient. Also, the system offers abilities to manage the family medical history. The user can read the patient's family medical history. From this task the user can manage the family medical history.	
Alternative Scenario 1 (the user is the patient) The user can't manage his family medical history. So the system won't give the option to manage it.	

A09: Managing Family History

Name: *Managing Family History*	**ID: *A09***
Main Scenario	

This use case extends the Viewing Family History use case. The user selects to manage the patient's family medical history, and then the system gives to the user the ability to manage the family history information of the patient. From here the user can insert, modify or remove an entry about a patient's family member.
Alternative Scenario 1 (the user chooses to insert) The user in this use case chooses to insert, then the system offers a way for the user to insert the family member's degree and a designation about the health problem. The user inserts the data. After that, the user confirms the insertion. The system stores the insertion.
Alternative Scenario 2 (the user chooses to modify) The user in this use case selects a family member health problem and chooses to modify it. The system offers a way to change family member degree and his health problem. The user makes the changes and confirms the modification. Then, the system stores the modifications into the card.
Alternative Scenario 3 (the user chooses to remove) The user in this use case selects a family member health problem and chooses to remove it. The system temporarily removes it leaving the ability to the user to recover it until the he exits the family medical history management.
Alternative Scenario 3.1 (the user chooses to recover a removed item) After removing a family member entry, the user decides to recover it and chooses to recover. The system changes the item status and won't remove anymore if the user exits the family medical history management.

A10: Viewing Health Problems

Name: *Viewing Health Problems*	**ID: *A10***
Main Scenario (the user is the doctor) This use case extends the Viewing Medical History use case. The user selects to view the health problems. Then the system shows the health problems history of the patient. The health problems history is shown as a list that contains in each line the date of when the problem was identified, a designation of the health problem and its status. This list is shown ordered chronologically and by active status. The system offers the ability to manage the health problems, that is, it gives a way of creating new entries about health problems and to modify or remove them. From this use case the system allows the user to choose to view the health problem's diagnosis and treatment plan written by a doctor.	
Alternative Scenario 1 (the user is the patient) From this task, the user can only see the health problems dates and designations and the associated treatment plans. The system won't give to the user the ability of managing the health problems and the ability to view the diagnosis of each health problem.	

A11: Managing Health Problems

Name: *Managing Health Problems*	**ID: *A11***
Main Scenario This use case extends the Viewing Health Problems use case. The user selects to manage the patient's health problems, and then the system gives to the user the ability to manage the health problems' entries. From here the user can insert, modify or remove an entry about a patient's health problem.	
Alternative Scenario 1 (the user chooses to insert) The user in this use case chooses to insert, then the system offers a way for the user to insert a new health problem entry by letting the user to insert the date, the health problem	

designation, and the status of the health problem. The user inserts the data. After that, the user confirms the insertion. The system stores the insertion into the card.
Alternative Scenario 2 (the user chooses to modify) The user in this use case selects a health problem entry and chooses to modify it. The system offers a way to change the date, the health problem designation and the status. The user makes the changes and confirms the modification. Then, the system stores the modifications into the card.
Alternative Scenario 3 (the user chooses to remove) The user in this use case selects a health problem and chooses to remove it. The system alerts the user that all the information associated with that health problem will be remove as well. The patient can confirm. If the patient confirms, the system temporarily removes the entry leaving the ability to the user to recover it until the he exits the medical history management. If the user exits the medical history management, the system removes the health problem entry including the diagnosis and treatment plan associated.
Alternative Scenario 3.1 (the user chooses to recover a removed item) After removing a health problem entry, the user decides to recover it and chooses to recover. The system changes the item status and won't remove anymore if the user exits the medical history management.

A12: Viewing Current and Past Medication

Name: ***Viewing Current and Past Medication***	**ID:** ***A12***
Main Scenario This use case extends the Viewing Medical History use case. The user selects to view the current and past medication of the patient. The system shows two lists, where in each of them is shown the current and past medication. The system shows the current medication designations associated with the date of prescription, and in another list below, shows the past medication in the same way.	
Alternative Scenario 1 (the user is the patient) The system offers a way of requesting prescription renewals of medication. The user can select what medications he wants to request a prescription renewal.	

A13: Viewing Diagnosis Information

Name: ***Viewing Diagnosis Information***	**ID:** ***A13***
Main Scenario (the user is the doctor) This use case extends the Viewing Health Problems use case. After the user selects the health problem e chooses to view its diagnosis. The system presents the health problem diagnosis, in which it shows a short description written by a doctor during an appointment and a list of measures and test results of the patient. Also the system offers a way of managing the diagnosis information.	

A14: Managing Diagnosis Information

Name: ***Managing Diagnosis Information***	**ID:** ***A14***
Main Scenario This use case extends the Viewing Diagnosis Information use case. The user selects to manage the health problem's diagnosis, and then the system gives to the user the ability to manage the diagnosis information. From here the user can insert or modify diagnosis information.	

Alternative Scenario 1 (the user chooses to insert) The user in this use case chooses to insert. Then the system offers a way for the user to insert a short diagnostic description and measurements/test result items in a list. The user inserts the data. After that, the user confirms the insertion. The system stores the insertion into the card.
Alternative Scenario 2 (the user chooses to modify) The user in this use case chooses to modify the diagnosis information. The system offers a way to change the diagnostic description and the measurements/test result items. The user makes the changes and confirms the modification. Then, the system stores the modifications into the card.

A15: Viewing Treatment Plan

Name: *Viewing Treatment Plan*	**ID: *A15***
Main Scenario (the user is the doctor) This use case extends the Viewing Health Problems use case and includes the Viewing Medical Recommendations use case. After the user selects the health problem e chooses to view its treatment plan. The system presents the health problem's treatment plan, in which it shows a short medical recommendation text written by a doctor at some point of an appointment. Also, the system offers a way of viewing the medications prescribed by the doctor.	

A16: Viewing Medical Recommendations

Name: *Viewing Medical Recommendations*	**ID: *A16***
Main Scenario (the user is the doctor) This use case is included in the Viewing Treatment Plan use case. The system shows a short medical recommendation text and offers to the user the ability of managing it.	
Alternative Scenario 1 (the user is the patient) The system won't offer the ability of managing the medical recommendation.	

A17: Managing Medical Recommendation

Name: *Managing Medical Recommendation*	**ID: *A17***
Main Scenario This use case extends the Viewing Medical Recommendation use case. The user selects to manage the medical recommendation, and then the system gives to the user the ability to manage the recommendation text. From here the user can insert or modify the medical recommendation text.	
Alternative Scenario 1 (the user chooses to insert) The user in this use case chooses to insert. Then the system offers a way for the user to insert a short medical recommendation. The user inserts the data. After that, the user confirms the insertion. The system stores the insertion into the card.	
Alternative Scenario 2 (the user chooses to modify) The user in this use case chooses to modify the medical recommendation text. The system offers a way to change the recommendation. The user makes the changes and confirms the modification. Then, the system stores the modifications into the card.	

A18: Viewing Medical Prescription

Name: ***Viewing Medical Prescription***	**ID:** ***A18***
Main Scenario (the user is the doctor) This use case extends the Viewing Treatment Plan use case. The user selects to view the medical prescription. Then, the system shows a list of medications with dosages and periods of administration associated. The system also shows the date of when the prescription was passed to the patient. From this use case, the system offers a way to manage the prescription's medication.	
Alternative Scenario 1 (the user is the patient) The system won't offer the ability of managing the medical prescription but offers instead the possibility of requesting the renewal of the medical prescription.	

A19: Requesting Prescriptions Renewal

Name: ***Requesting Prescriptions Renewal***	**ID:** ***A19***
Main Scenario (the user is the doctor) This use case can extend the Viewing Medical Prescription or the View Current And Past Medication use cases. The user chooses to request a medical prescription renewal. Then, the system shows a list of medications for the user select what medicines he wishes to renew. The user selects the medicines and sends the request by confirming it. The system sends a note to the medical staff with the pretended medication to renew and the information of the health problem associated to that prescription (the health problem status, diagnosis and treatment plan).	

A20: Managing Medicines

Name: ***Managing Medicines***	**ID:** ***A20***
Main Scenario This use case extends the Viewing Medical Prescription use case. The user selects to manage the medicines in the medical prescription, and then the system gives to the user the ability to manage the medicines. From here the user can insert, modify or remove prescription's medicines.	
Alternative Scenario 1 (the user chooses to insert) The user in this use case chooses to insert, then the system offers a way for the user to insert a new medicine by letting the user to insert the designation, the dosage, and the period of administration. The user inserts the data. After that, the user confirms the insertion. The system stores the insertion into the card.	
Alternative Scenario 2 (the user chooses to modify) The user in this use case selects a medicine and chooses to modify it. The system offers a way to change the designation, the dosage and the period of administration. The user makes the changes and confirms the modification. Then, the system stores the modifications into the card.	
Alternative Scenario 3 (the user chooses to remove) The user in this use case selects a medicine and chooses to remove it. The system temporarily removes the medicine leaving the ability to the user to recover it until the he exits the medical prescription management. If the user exits the medical prescription management, the system removes the temporarily removed medicines from the card.	
Alternative Scenario 3.1 (the user chooses to recover a removed item) After removing a medicine from the prescription list, the user decides to recover it and	

chooses to recover. The system changes the item status and won't remove anymore if the user exits the medical prescription management.

B: Personal Data Use Cases

B01: Viewing Personal Data

Name: ***Viewing Personal Data***	**ID:** ***B01***
Main Scenario (the user is the patient) This use case initiate after successfully validating the owner's card and selecting to view the patient's personal data. The system will show the personal data details of the patient. The patient's name, gender, age, blood type, etc, are shown in the personal data view. The system offers the ability of managing those personal details.	

B04: Managing Optional Details

Name: ***Managing Optional Details***	**ID:** ***B04***
Main Scenario This use case extends the Viewing Personal Data use case. The user selects to manage the personal data details, and then the system gives to the user the ability to manage it. From here the user can insert or modify non-obligatory personal details.	
Alternative Scenario 1 (the user chooses to insert) The user in this use case chooses to insert, then the system offers a way for the user to insert personal details that haven't yet been inserted. The user inserts the details that he wants to insert and then confirms the insertion. Then, the system stores the insertions into the card.	
Alternative Scenario 2 (the user chooses to modify) The user in this use case chooses to modify some personal details. The system offers a way to change the non-obligatory details. The user makes the changes and confirms the modification. Then, the system stores the modifications into the card.	

B05: Managing Required Details

Name: ***Managing Required Details***	**ID:** ***B05***
Main Scenario (the user is the card issuer) The user selects to manage the required (obligatory) personal data details, and then the system gives to the user the ability to manage it. From here the user can insert or modify the personal obligatory details. This use case is made when creating the patient's card by the card issuer.	
Alternative Scenario 1 (the user chooses to insert) The user in this use case chooses to insert, then the system offers a way for the user to insert personal obligatory details. The user inserts the details and then confirms the insertion. Then, the system stores the insertions into the card.	
Alternative Scenario 2 (the user chooses to modify) The user made an error when inserting and in this use case chooses to modify some personal details. The system offers a way to change the obligatory details. The user makes the changes and confirms the modification. Then, the system stores the modifications into the card.	

C: Appointments Use Cases

C01: Scheduling Appointment

Name: ***Scheduling Appointment***	**ID:** ***C01***
Main Scenario This use case includes the Viewing Schedules Availability use case from the external system. This use case initiate after successfully validating the owner's card and selecting to schedule an appointment. The system then offers three ways of scheduling an appointment, by searching a doctor, by searching a date or scheduling by type of appointment. When the user chooses one of the previous options the system obtains and shows the availability of schedules.	

C02: Viewing Appointments

Name: ***Viewing Appointments***	**ID:** ***C02***
Main Scenario This use case initiate after successfully validating the owner's card and selecting to view the scheduled appointments. The system then shows in the screen the list of scheduled appointments. In each entry of the list the system describes the date and hour, the medical center name, and optionally the name of the doctor and the type of appointment. Also the system offers ways of managing the scheduled appointments. The user can choose to manage schedule entries by selecting an entry and choosing what to do.	

C03: Managing Appointments

Name: ***Managing Appointments***	**ID:** ***C03***
Main Scenario This use case extends the Viewing Appointments use case. In this use case the user chooses to manage an appointment after selecting it. Then, the system offers ways of modifying and canceling the selected appointment. The user chooses one of the options.	
Alternative Scenario 1 (the user chooses to modify) The user chooses to modify the scheduled appointment after selecting the schedule entry. The system then offers a way of modifying the date and the doctor or type of appointment by showing to the user the available schedules. The user then chooses the pretended date, doctor or appointment type and confirms. The system stores the changes into the card.	
Alternative Scenario 2 (the user chooses to cancel) The user chooses to cancel a scheduled appointment after selecting the schedule entry. The system then asks for the confirmation. The user confirms it and then the system removes the scheduled appointment.	

C04: Checking-in

Name: ***Checking-in***	**ID:** ***C04***
Main Scenario This use case initiate after successfully validating the owner's card and selecting to check-in. The system shows to the user the appointment scheduled in the card within a certain time and offers a way of confirming the check-in. The user confirms the check in and then the system sends a notice to the doctor's computer that the patient has checked-in.	

C05: Viewing Schedule Availability

Name: ***Viewing Schedule Availability***	**ID:** ***C05***
Main Scenario This use case is included in the Scheduling Appointment use case and includes either the Searching by Doctor or Searching by Date use cases. The user chooses to view the available schedules and then the system offers a way of scheduling by type of appointment and searching by doctor or by Date. The user chooses one of the options and then the system shows to the user the available schedule. The user then chooses the schedule date in the available schedule presented. The system shows the summary of the schedule done and asks for the user confirmation. The user confirms it and the system stores the scheduled appointment into the card.	

C06: Searching by Doctor

Name: ***Searching by Doctor***	**ID:** ***C06***
Main Scenario This use case is included in the Viewing Schedule Availability use case. The user chooses to search available schedules by doctor. The system offers a way of inserting the doctor's name or browsing through resident doctors. The user chooses the doctor by inputting his name. Then, the system shows the available schedule for that doctor.	

C07: Searching by Date

Name: ***Searching by Date***	**ID:** ***C07***
Main Scenario This use case is included in the Viewing Schedule Availability use case. The user chooses to search available schedules by date. The system offers a way of inserting the date. The user chooses the day and month. Then, the system shows the available schedule for that date.	

3. Annex: Informal Functional Requirements

In this section we describe the functional requirements of our system. They were gathered from discussions with stakeholders and from reading documents about medical appointments.

3.1. In-Card Requirements (server applications)

General

FR1. The card must have information about the patient's personal data, his medical history, and appointment schedule.

FR2. The patient's medical history must be editable by the medical staff, and only by the medical staff.

FR3. The patient's medical history must include entries of diagnostics and associated treatments of health problems made in appointments, as well as a list of administrated vaccines and the patient's allergies.

FR4. The medical history is composed of appointments (including diagnostics, treatments and medications), allergies, vaccinations and chronic conditions.

FR5. The system must represent on the card all dates with the following format: mm/dd/yyyy, where month and day are composed of two digits and year is represented by four digits.

FR6. The system must respect the number of days of February according to the leap years when dealing with dates.

FR7. The system must respect the number of days of the respective months.

FR8. It is only possible to have dates of the years starting with 18xx, 19xx and 20xx.

FR9. The system must represent on the card the time (hour) with the following format: hh:mm, where hour and minutes are represented by two digits each, and where the hour has a value from 0 to 23.

FR10. The information related to a date or hour, must be stored in the card in the form of hexadecimal values of the decimal values, and not in hexadecimal values of characters.

Security

FR11. For security reasons, the card must have an owner Pin-code.

FR12. The card must have a card issuer/provider Pin-code.

FR13. Whenever the card is inserted into a machine, the card pin-code must be asked.

FR14. It must be possible for a card owner to change his pin-code.

FR15. The card pin-code must have a maximum of 4 digits.

FR16. The card must allow 3 attempts to insert the right pin-code in maximum.

FR17. The card must be blocked to the user if he fails the 3 pin validating attempts.

FR18. The card must allow the card issuer to unlock a blocked card.

Personal Data

FR19. The patient's personal data must be divided in required data and optional personal data.

FR20. The required personal data must contain the patient's name, gender, birth date, blood type, ID number or passport ID, his birthplace and nationality.

FR21. The optional personal data includes a personal phone contact, a patient's relative phone contact, an address (including the city, country and zip code), a social security number, and a preferred language.

FR22. The optional personal data must be editable by the patient.

FR23. The required personal data must be only modifiable only by administrator.

FR24. The system must not permit to enter values of a patient's name, passport ID, ID number, phone contacts, address, city, zip code and social security with a length no longer than the respective stipulated limit.

FR25. The system must represent the address country, nationality and birth place as a country code.

FR26. The system must not permit to enter values of country code and preferred language with a length no longer or lesser than the respective stipulated limit.

FR27. The only possible values for the patient's gender are *male*, *female*, and *undefined*.

FR28. By default, the patient's gender is initialized with the value *undefined*, until the card issuer inserts the patient's information.

FR29. The only possible values for the patient's blood type are *A+*, *A-*, *B+*, *B-*, *AB+*, *AB-*, *O+* and *O-*. That makes eight possible values, plus one for initialization, for representing the blood type.

FR30. The system must represent the values of blood types as numbers from zero to eight, where zero represents the initialization value of a blood type until the card issuer inserts the patient's information, and each value from 1 (one) to 8 (eight) represents respectively the values *A+*, *A-*, *B+*, *B-*, *AB+*, *AB-*, *O+* and *O-*.

FR31. The information related to personal data, must be stored in the card in the form of hexadecimal values of the characters of its textual descriptions (i.e. *strings*). Excepts the birth date, gender and blood type.

FR32. The card owner's name, and city (from address) must only contain letters (upper and lower case) and special characters like spaces or letter accents.

FR33. The card owner's birth date must have the same format as all other dates in the system.

FR34. The card owner's birth place, nationality, country (from address) and preferred language must be represented by a code in the card and must only contain letters (upper and lower case).

FR35. The card owner's id number and passport id must only contain letters (upper and lower case) and number digits (0 to 9). No special characters.

FR36. The card owner's address and zip code can contain any characters.

FR37. The card owner's phone contact, relative phone contact and social security number must only contain number digits (0 to 9).

Allergies

FR38. The system must specify an allergy reference code to the designation and type of allergy and its date of identification.

FR39. Each patient allergy must be associated with a designation, a type of allergy and a date.

FR40. The date associated with an allergy must be the date of the allergy identification.

FR41. When adding an allergy information it is required to specify an allergy reference code to the designation and type of allergy and its date of identification.

FR42. The user can remove any allergy that is in the card.

FR43. if the limit of possible allergies insertions is achieved, the system must not allow insertions of allergies into the card.

FR44. Patient's allergy designations and types that are known and exists on the external system database must be represented by reference codes in the card.

FR45. The designation and type of allergies must be represented by a reference code in the card, and its format must only contain letters (upper and lower case) and numbers digits (0 to 9), so no special characters are allowed and not spaces too.

FR46. The given allergy reference code must have exactly the stipulated length.

FR47. The allergies designation must be stored in the card in the form of hexadecimal values of the characters of its textual descriptions (i.e. *strings*).

FR48. The allergies identification date must have the same format as all other dates in the system.

FR49. The system can only allow a member of the medical staff to insert or remove a patient's allergy in the card.

FR50. There must not exist duplicated entries of an allergy with the same allergy reference code.

Vaccines

FR51. The system must represent vaccine information by specifying its designation and its date of administration.

FR52. The system must allow inserting data related to the patient's vaccination history.

FR53. The system must only allow a medical staff to insert or remove a vaccine from the vaccination list.

FR54. When adding vaccine information it is required to specify its designation and its date of administration.

FR55. The designation of a vaccine must be represented by a reference code in the card, and its format must only contain letters (upper and lower case) and numbers digits (0 to 9), so no special characters are allowed and not spaces too.

FR56. The system must permit the insertion of a designation or administration date value of a vaccine that has a length with the same stipulated limit.

FR57. The information related to vaccines, except the administration date, must be stored in the card in the form of hexadecimal values of the characters of its textual descriptions (i.e. *strings*).

FR58. The vaccines administration date must have the same format as all other dates in the system.

FR59. The user can't remove any vaccine that isn't in the card.

FR60. The system must not allow the insertion into the card anymore vaccines information if the limit of possible vaccines insertions is achieved.

FR61. The system must store the date with its decimal values converted to hexadecimals.

Diagnostics (dependent to Appointments)

FR62. The system must allow to represent a short description made by the doctor during an appointment and has a title to inform about its contents.

FR63. Each diagnostic must be related to only one appointment.

FR64. When adding a diagnostic it is required a short description made by the doctor during an appointment and has a title to inform about its contents.

FR65. The system must not permit to add a diagnostic to an appointment if it has a schedule status.

FR66. The information related to diagnostics, namely, short description and title, must be stored in the card in the form of hexadecimal values of the characters of its textual descriptions (i.e. *strings*).

FR67. The diagnostic's short descriptions and titles may contain letters (upper and lower case), number digits (0 to 9) and special characters.

FR68. Several diagnostics can be associated with an appointment.

FR69. There can't be any data(treatments or medicines) related to an appointment if there's no diagnostic made.

FR70. The system must delete from the card any appointment where no diagnostics were made, after 24 hours of the schedule time .

FR71. The system must allow a doctor to modify a diagnostic during an appointment.

FR72. If a diagnostic is deleted from the card, all treatments and medicines associated with it must be deleted as well.

FR73. When trying to delete a diagnostic the system must warn about its dependencies if any.

Treatments (dependent to Appointments)

FR74. The system must allow to represent health problems by a designation and associated with the treatment prescribed during an appointment.

FR75. Medical treatments must be associated with diagnostics made during an appointment.

FR76. To insert a treatment there must be a diagnostic associated.

FR77. A medical plan (treatment) must be associated with a medical appointment and can contain medication prescribed by a doctor, medical recommendations and the health problem associated.

FR78. When adding a treatment it is required to have a medical recommendations and the health problem associated.

FR79. The information related to treatments, namely, medical recommendations and health problem must be stored in the card in the form of hexadecimal values of the characters of its textual descriptions (i.e. *strings*).

FR80. The treatments' medical recommendations and health problem associated may contain letters (upper and lower case), number digits (0 to 9) and special characters.

FR81. The system must relate one treatment per health problem, so an appointment may have more than one treatment associated, because it is possible to have more than one health problem identified in one appointment.

FR82. The system must allow having several treatments associated with a diagnostic.

Medicines (dependent to Treatments)

FR83. The data related to medication prescribed by a doctor are the designation of the medicine, the period of administration (period of time in days that the patient should take the medicine), the administration description (how the medicine should be administrated, i.e. dosages and frequencies) and the date of the prescribed medication.

FR84. To insert a Medicine there must be a treatment associated.

FR85. When adding a Medicine it is required to have a designation of the medicine, the period of administration (period of time in days that the patient should take the medicine), the administration description (how the medicine should be administrated, i.e. dosages and frequencies) and the date of the prescribed medication.

FR86. The information related to medications, namely, designation of the medicine administration description, must be stored in the card in the form of hexadecimal values of the characters of its textual descriptions (i.e. *strings*).

FR87. The information about the period of administration must be stored as a number that represents the number of days.

FR88. The designation of a medicine must be represented by a reference code in the card, and its format must only contain letters (upper and lower case) and numbers digits (0 to 9), so no special characters are allowed and not spaces too.

FR89. The given medicines designation reference codes must have exactly the stipulated length.

FR90. The medicines' administration description may contain letters (upper and lower case), number digits (0 to 9) and special characters including spaces.

FR91. The medicine date must have the same format as all other dates in the system.

FR92. The prescription date of a medicine must be the same as the respective appointment's date or a later date.

FR93. The prescription date of a medicine must be bigger than or equal to the date of the appointment in which the medicine was prescribed.

Chronic Conditions

FR94. The system must specify a chronic condition reference code to the designation of chronic conditions and its date of identification.

FR95. Each patient chronic condition must be associated with a designation and a date.

FR96. The date associated with a chronic condition must be the date of the chronic condition identification.

FR97. When adding an chronic condition information it is required to specify an chronic condition reference code to the designation and its date of identification.

FR98. The user can only remove a chronic condition that is in the card.

FR99. If the limit of possible chronic condition insertions is achieved, the system must not allow insertions of chronic condition into the card.

FR100. Patient's chronic condition designation are known and exists on the external system database must be represented by reference codes in the card.

FR101. The designation of chronic condition must be represented by a reference code in the card, and its format must only contain letters (upper and lower case) and numbers digits (0 to 9), so no special characters are allowed and not spaces too.

FR102. The given chronic condition reference code must have exactly the stipulated length.

FR103. The chronic condition designation must be stored in the card in the form of hexadecimal values of the characters of its textual descriptions (i.e. *strings*).

FR104. The chronic condition identification date must have the same format as all other dates in the system.

FR105. The system can only allow a member of the medical staff to insert or remove a patient's chronic condition in the card.

FR106. There must not exist duplicated entries of a chronic condition with the same allergy reference code.

Appointments

Scheduling

FR107. It must be possible to schedule appointments.

FR108. A scheduled appointment must contain data of a place (local), a date/time and a doctor or an appointment type.

FR109. When adding a new scheduled appointment information it is required to specify its date, hour, local and also the doctor or the type of appointment.

FR110. If the limit of possible appointment insertions is achieved, the system must not allow insertions of appointments into the card.

FR111. The system must represent a doctor by a reference code associated with the local of practicing (hospital, clinic, etc.).

FR112. The doctor of an appointment must be represented by a reference code in the card, and its format must only contain letters (upper and lower case) and numbers digits (0 to 9), so no special characters are allowed and not spaces too.

FR113. The given doctor reference code must have exactly the stipulated length.

FR114. The doctor must be stored in the card in the form of hexadecimal values of the characters of its textual descriptions (i.e. *strings*).

FR115. The data about the place of the appointment must contain the name of the medical center, the name of the city and a reference code of the country.

FR116. The system must represent the local as a code where the first two digits represent the country, the three next digits represents the city, and the last three digits represents the place (medical center, hospital, etc). In total the system must represent the local as an 8 digits code.

FR117. Data about the appointment's date must include the date (year, month and day) and hour.

FR118. The appointment date must have the same format as all other dates in the system.

FR119. The appointment hour must have the same format as all other hours in the system.

FR120. The given appointment type reference code must be within the stipulated range.

FR121. It must be possible to modify a scheduled appointment.

FR122. The system must not allow modifying the data (i.e. date and time, local, doctor, type of appointment) of an appointment after checking in to that appointment.

FR123. It must be possible to cancel a scheduled appointment.

FR124. It must not be possible to overlap schedules in the same date and hour.

Checking-in

FR125. When a check-in of an appointment is made, that scheduled appointment must be turned into a checked-in appointment.

FR126. When a check-in of a scheduled appointment is not made in a period of 1 day, that scheduled appointment must be erased from the card.

FR127. If an appointment check-in was made, but there is no related data entries in a period of 1 day, that appointment should be erased from the card.

Effective Appointments

FR128. To an effective appointment there must be related diagnostic inserted by the appointment doctor.

FR129. An appointment has an effective status when it has medical information associated.

FR130. An appointment cannot be deleted from the card if its status is effective.

3.2. Off-Card Requirements (client applications)

FR131. When trying to schedule an appointment by date and hour, the available doctors should be showed.

FR132. When trying to schedule an appointment by a doctor, the available date and hour should be showed.

FR133. To a doctor it must be associated his name and his specialty.

FR134. When scheduling an appointment the user only have to indicate a date/hour and a doctor or type of appointment.

FR135. The system must have the responsibility of storing the data about the place in the card when the patient is scheduling an appointment.

FR136. It must be possible to medical staff to visualize and authorize requests of medical prescriptions renewals.

FR137. The idiom of the card owner must be recognized.

FR138. There must be a way of giving to the patient the ability of turning confidential some personal data.

FR139. There must be a way of giving to the patient the ability of requesting the renewal of medical prescription using the card.

FR140. It must be possible to make an appointment check-in using the card.

FR141. The check-in must be done before the appointment scheduled time or until the previous patient is called.

FR142. If a patient is making a check-in out of time the system must alert him and let him schedule another appointment.

FR143. The system must have a list of all allergies identified by medicine.

FR144. The card owner only can see from his medical history the designations of his health problems, prescriptions, vaccinations and allergies. He must not see the general descriptions made by the doctors during the appointment.

FR145. The medical staff can only access the patient's personal data that hasn't a confidential state.

FR146. The system must permit the insertion of allergy designations.

FR147. The system must permit the insertion of doctors and their availability schedules for appointments.

FR148. The system must permit the configuration of data related to the medical center such its name, place and country.

3.3. General System Requirements

FR149. Doctors that will use the system to insert data on the patient's card must have a registry on the central system database, so he can be represented by a reference code when entries on the card make a reference to him.

FR150. Hospitals, Medical Centers and other facilities using the system must register themselves in the central system database, so they can be represented by a reference code when entries on the card make a reference about them.

FR151. Allergy designations, types of allergies, vaccine designations and medicine designations represented on the patient's card by reference codes must exist in the central system database, and must be synchronized with all systems implemented on other medical facilities.

4. Annex: Semi-Formal Requirements

From the informal functional requirements, the semi-formal requirements (SFR) were obtained.

Personal Data (Semi-Formal Requirements)

ID	From Requirements	Semi-Formal Requirements
SFR24	FR24	If passing a value of a patient's name, passport ID, ID number, phone contact, address, city, zip code or social security and its value has a length longer than the stipulated; then the system must do no changes.
SF26	FR26	If passing a value of a country code or preferred language and its value has a length different from the stipulated limit; then the system must do no changes.
SFR31	FR31	If passing a value of an attribute, that is not a birth date, gender or blood type, then these data are stored in the form of hexadecimal values of the characters of its textual descriptions.
SFR32	FR32	If passing a value of a patient's name or city, then they must contain only letters (upper and lower case) and special characters like spaces and accents.
SFR34	FR34	If passing a value of a birth place, nationality, country (from address) or preferred language, then they must be a code containing only letters (upper and lower case).
SFR35	FR35	If passing a value of ID number or passport ID, then they must contain only letters (upper and lower case) and numbers digits (0 to 9).
SFR37	FR37	If passing a value of a phone contact, relative contact or a social security number, then they must contain only numbers digits (0 to 9).

Allergies (Semi-Formal Requirements)

ID	From Requirements	Semi-Formal Requirements
SFR41	FR41	If adding a new allergy, then is necessary to insert the allergy reference code and the date of identification.
SFR42	FR42	If removing an allergy and there isn't any allergy, then the system must do no changes.
SFR43	FR43	If adding an allergy and the limit has been achieved, then the system must do no changes.
SFR45	FR45	If passing a value of an allergy reference code, then the value of the reference code must only contain letters (upper and lower case) and number digits (0 to 9) and no special characters and spaces are allowed.
SFR46	FR46	If passing a value of an allergy reference code, then its length must be exactly the stipulated length.
SFR47	FR47	If passing an allergy reference code, then the reference code must contain only letters and numbers and not spaces or other

		special characters and the system must store them as hexadecimal values of the reference code characters.
SFR50	FR50	If passing an allergy reference code, then it should not exists another stored allergy with the same reference code.

Vaccines (Semi-Formal Requirements)

ID	From Requirements	Semi-Formal Requirements
SFR54	FR54	If adding a new vaccine, then it is required to insert its designation and its date of vaccine administration.
SFR55	FR55	If passing a value of vaccine designation code, then the value of the reference code must only contain letters (upper and lower case) and number digits (0 to 9) and no special characters and spaces are allowed.
SFR56	FR56	If passing a value of vaccine designation code or a value of a date, then its length must be exactly the stipulated length.
SFR57	FR57	If passing a value of vaccine designation code, then the system must store it as hexadecimal values of the characters.
SFR59	FR59	If removing a vaccine and there isn't any vaccine information, then the system must do no changes.
SFR60	FR60	If adding a vaccine and the limit has been achieved, then the system must do no changes.
SFR61	FR61	If passing a value of vaccine administration date, then the system must store its decimal values as hexadecimals.

Diagnostics (Semi-Formal Requirements)

ID	From Requirements	Semi-Formal Requirements
SFR64	FR64	If adding a new diagnostic, then it is required to insert its description and its title.
SFR65	FR65	If adding a diagnostic and it has an appointment with schedule status; then the system must do no changes.
SFR66	FR66	If passing a value of diagnostic's description or title, then the system must store it as hexadecimal values of the characters.
SFR67	FR67	If passing a value of diagnostic's description or title, then the value the values can contain letters (upper and lower case), number digits (0 to 9) and special characters.

Treatments (Semi-Formal Requirements)

ID	From Requirements	Semi-Formal Requirements
SFR78	FR78	If adding a new treatment, then it is required to insert its medical recommendations and its health.
SFR79	FR79	If passing a value of treatment's medical recommendation or health problem, then the system must store it as hexadecimal values of the characters.

SFR80	FR80	If passing a value of treatment's medical recommendation or health problem, then the value the values can contain letters (upper and lower case), number digits (0 to 9) and special characters.

Medicines (Semi-Formal Requirements)

ID	From Requirement	Semi-Formal Requirements
SFR85	FR85	If adding a new medicine, then is necessary to insert a medicine reference code, the period of administration, the administration description and the date of the prescribed medication.
SFR86	FR86	If passing a value of medicine's designation or administration description, then the system must store it as hexadecimal values of the characters.
SFR87	FR87	If passing a value of period of administration, then the system must store it as a number.
SFR88	FR88	If passing a value of a medicine's designation, then the value of the reference code must only contain letters (upper and lower case) and number digits (0 to 9) and no special characters and spaces are allowed.
SFR89	FR89	If passing a value of a medicine's designation, then its length must be exactly the stipulated length.
SFR90	FR90	If passing a value of medicine's description, then the value the values can contain letters (upper and lower case), number digits (0 to 9) and special characters.

Chronic Conditions (Semi-Formal Requirements)

ID	From Requirements	Semi-Formal Requirements
SFR97	FR97	If adding a new chronic condition, then is necessary to insert the chronic condition reference code and the date of identification.
SFR98	FR98	If removing a chronic condition and there isn't any chronic condition, then the system must do no changes.
SFR99	FR99	If adding a chronic condition and the limit has been achieved, then the system must do no changes.
SFR101	FR101	If passing a value of a chronic condition reference code, then the value of the reference code must only contain letters (upper and lower case) and number digits (0 to 9) and no special characters and spaces are allowed.
SFR102	FR102	If passing a value of a chronic condition reference code, then its length must be exactly the stipulated length.
SFR103	FR103	If passing a chronic condition reference code, then the reference code must contain only letters and numbers and not spaces or other special characters and the system must store them as hexadecimal values of the reference code characters.
SFR106	FR106	If passing a chronic condition reference code, then it should not exists another stored chronic condition with the same reference code.

Appointments (Semi-Formal Requirements)

ID	From Requirements	Semi-Formal Requirements
SFR109	FR109	If adding a new scheduled appointment, then it must be inserted a date, an hour, a place and doctor or a type of appointment.
SFR110	FR110	If adding a new scheduled appointment and the limit has been achieved, then the system must do no changes.
SFR112	FR112; FR114	If passing a value of a doctor reference code, then the value of the reference code must only contain letters (upper and lower case) and number digits (0 to 9) and no special characters and spaces are allowed and the system must store them as hexadecimal values of the reference code characters.
SFR113	FR113	If passing a value of a date, hour, local or doctor, then its length must be exactly the stipulated length.
SFR120	FR120	If passing a value of an appointment type, then its value must be between zero and the stipulated range.
SFR122	FR122	If an appointment is already checked-in, then the appointment header cannot be modified (date and time, local, doctor, type of appointment).
SFR125	FR125	If an appointment is checked-in, then that appointment must turn into a checked-in state.
SFR130	FR130	If removing an appointment and its status is effective, then the system must do no changes.

5. Annex: Class Invariants

General/Common (Class Invariants)

ID	From Requirements	Class Invariants
CI5	FR108	Common.date must be array byte of length equal to 4 and Common.date[0] is *month,* Common.date[1] is *day* and (Common.date[2], Common.date[3]) is *year*.
CI6	FR6; FR10	If isLeapYear(*year*) then if *month* equal to February (i.e., 0x02), then *day* less than 29 (0x1D) and more than 0 (0x00). Else, *day* less than 28 (0x1C) and more than 0 (0x00).
CI7	FR7; FR10	For all possible instances of *month* except *month* equal to February (i.e., 0x02), there is a respective *day* less than 30 (0x1E) or 31 (0x1F).
CI8	FR8; FR10	For all possible instances of year, such that Common.date[2] equals to 18 (0x12) or 19 (0x13) or 20 (0x14) and Common.date[3] more or equal to 0 (0x00) and less or equal to 99 (0x63).
CI9	FR9	Common.hour must be array byte of length equal to 2 and Common.hour[0] is *hour,* Common.date[1] is *minutes.*
CI10	FR9; FR10	For all possible instances of Common.hour, such that *hour* more than or equal to 0 (0x00) and *hour* less than or equal to 23 (0x17) and *minutes* more than or equal to 0 (0x00) and *minutes* less than or equal to 59 (0x3B) and *minutes.*

Personal Data (Class Invariants)

ID	From Requirements	Class Invariants
CI20	FR20	For all object ***p*** of type personal, such that name(***p***) not equal to null, gender(***p***) not equal to null, birthdate(***p***) not equal to null, bloodtype(***p***) not equal to null, id(***p***) or passport(***p***) not equal to null, birthplace(***p***) not equal to null, nationality(***p***) not equal to null
CI21	FR21	For all object ***p*** of type personal, such that phone(***p***) is nullable, relativephone(***p***) is nullable, address_city(***p***) is nullable, address_country(***p***) is nullable, address_zipcode(***p***) is nullable, socialsecurity(***p***) is nullable and language(***p***) is nullable
CI25	FR25	For all object ***p*** of type personal, such that address_country(***p***) and nationality(***p***) and birthplace(***p***) is *country_code*
CI27	FR27	For all object ***p*** of type personal, such that gender(***p***) equal to Undefined (0x00) or gender(***p***) equal to Male (0x01) or gender(***p***) equal to Female (0x02)
CI28	FR28	For all *new* object ***p*** of type personal, such that gender(***p***) equal to Undefined (0x00)
CI29	FR29; FR30	For all object ***p*** of type personal, such that bloodtype(***p***) maximum is value 8 and bloodtype(***p***) equal to (Undefined (0x00) or A+ (0x01) or A- (0x02) or B+ (0x03) or B- (0x04) or AB+ (0x05) or AB- (0x06) or O+ (0x07) or O- (0x08))
CI33	FR33	For all object ***p*** of type personal, such that birthdate(***p***) is Common.date

Allergies (Class Invariants)

ID	From Requirements	Class Invariants
CI38	FR38	For all object ***a*** of type allergy, such that designation(***a***) not equal to null, type(***a***) not equal to null, identification_date (***a***) not equal to null
CI48	FR48	For all object ***a*** of type allergy, such that identification_date(***p***) is Common.date

Vaccines (Class Invariants)

ID	From Requirements	Class Invariants
CI51	FR51	For all object ***v*** of type vaccine, such that designation(***v***) not equal to null, administration_date (***v***) not equal to null
CI58	FR58	For all object ***v*** of type vaccine, such that administration_date(***v***) is Common.date

Diagnostics (Class Invariants)

ID	From Requirements	Class Invariants
CI62	FR62	For all object ***d*** of type diagnostic, such that title(***d***) not equal to null, description(***d***) not equal to null
CI63	FR63	For all object ***d*** of type diagnostic, Exists object ***a*** of type appointment, such that appointmentID(***d***) equals to appointmentID(***a***)

Treatments (Class Invariants)

ID	From Requirements	Class Invariants
CI74	FR74	For all object ***t*** of type treatment, such that health_problem (***t***) not equal to null, recommendation(***t***) not equal to null

Medicines (Class Invariants)

ID	From Requirements	Class Invariants
CI83	FR83	For all object ***m*** of type medicine, such that designation(***m***) not equal to null, administration_period(***m***) not equal to null, administration_description(***t***) not equal to null, prescription_date(***t***) not equal to null
	FR92; FR92	For all object ***m*** of type medicine, such that date(m) is Common.date

Chronic Conditions (Class Invariants)

ID	From Requirements	Class Invariants
CI94	FR94	For all object ***cc*** of type chronic_condition, such that designation(***cc***) not equal to null, identification_date (***cc***) not equal to null
CI104	FR104	For all object ***cc*** of type chronic_condition, such that identification_date(***cc***) is Common.date

Appointments (Class Invariants)

ID	From Requirements	Class Invariants
CI108	FR108	For all object ***a*** of type appointment, such that local(***a***) not equal to null, date(***a***) not equal to null, hour(***a***) not equal to null and (doctor (***a***) not equal to null or type(***a***) not equal to null).
CI116	FR116	For all objects ***a*** of type appointment, such that local(**a**) must be array byte of length equal to 8 and (local(***a***)[0], local(***a***)[1]) equal to *country_code*, (local(***a***)[2], local(***a***)[3], local(***a***)[4]) equal to *city_code* and (local(***a***)[5], local(***a***)[6], local(***a***)[7]) equal to *place_code*.
CI117	FR117; FR118; FR119	For all object ***a*** of type personal, such that date(a) is Common.date and hour(***a***) is Common.hour
CI124	FR124	For all objects ***a1*** and ***a2*** of type appointment, if ***a1*** not equal to ***a2*** then (date(***a1***) not equal to date(***a2***) and hour(***a1***) not equal to hour(***a2***)).
CI126	FR126	For all objects ***a*** of type appointment, such that if status(***a***) less than CHECK_IN then timeDifference(schedule_time(***a***), actual_time) must be less than 24 hours.

6. Annex: System Invariants

ID	From Requirements	System Invariants
SI69	FR69; FR72; FR75; FR76	For all object ***t*** of type treatment, Exists object ***d*** of type diagnostic, such that diagnosticID(***t***) equals to diagnosticID(***d***)
SI69b	FR69; FR72	For all object ***m*** of type medicine, Exists object ***d*** of type diagnostic, such that diagnosticID(***m***) equals to diagnosticID(***d***)
SI84	FR84	For all object ***m*** of type medicine, Exists object ***t*** of type treatment, such that treatmentID(***m***) equals to treatmentID(***t***)
SI93	FR93	For all object ***m*** of type medicine and For all object ***a*** of type appointment such that appointmentID(***m***) equals to appointmentID(***a***) and date(***m***) is bigger than or equal to date(***a***).
SI126	FR127; FR72	For all objects ***a*** of type appointment, such that If status(***a***) more or equal to CHECK_IN and timeDifference(schedule_time(***a***), actual_time) more than 24 hours then exists object ***d*** of type diagnostic, such that appointmentID(***d***) equals to appointmentID(***a***).

7. Annex: Class Diagrams

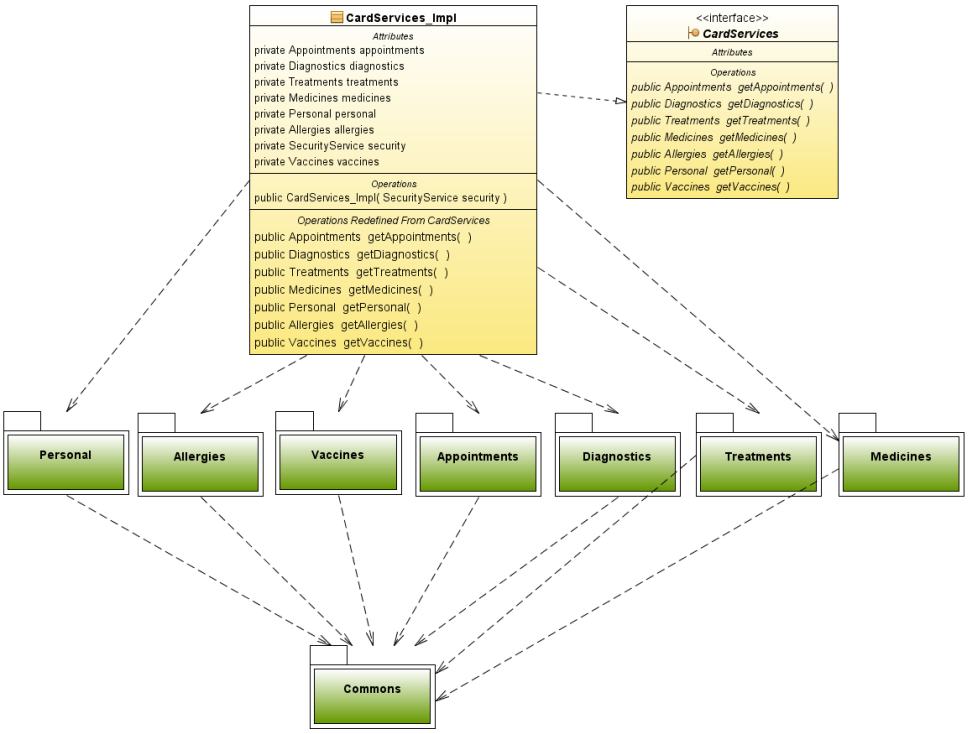

Figure v. CardServices module Class Diagram

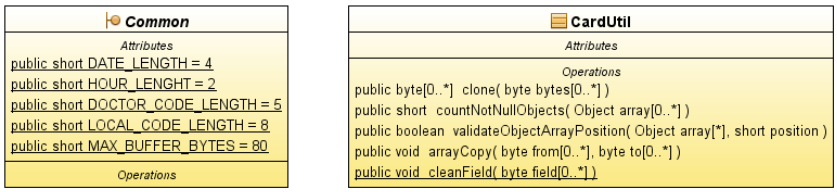

Figure vi. Common module Class Diagram

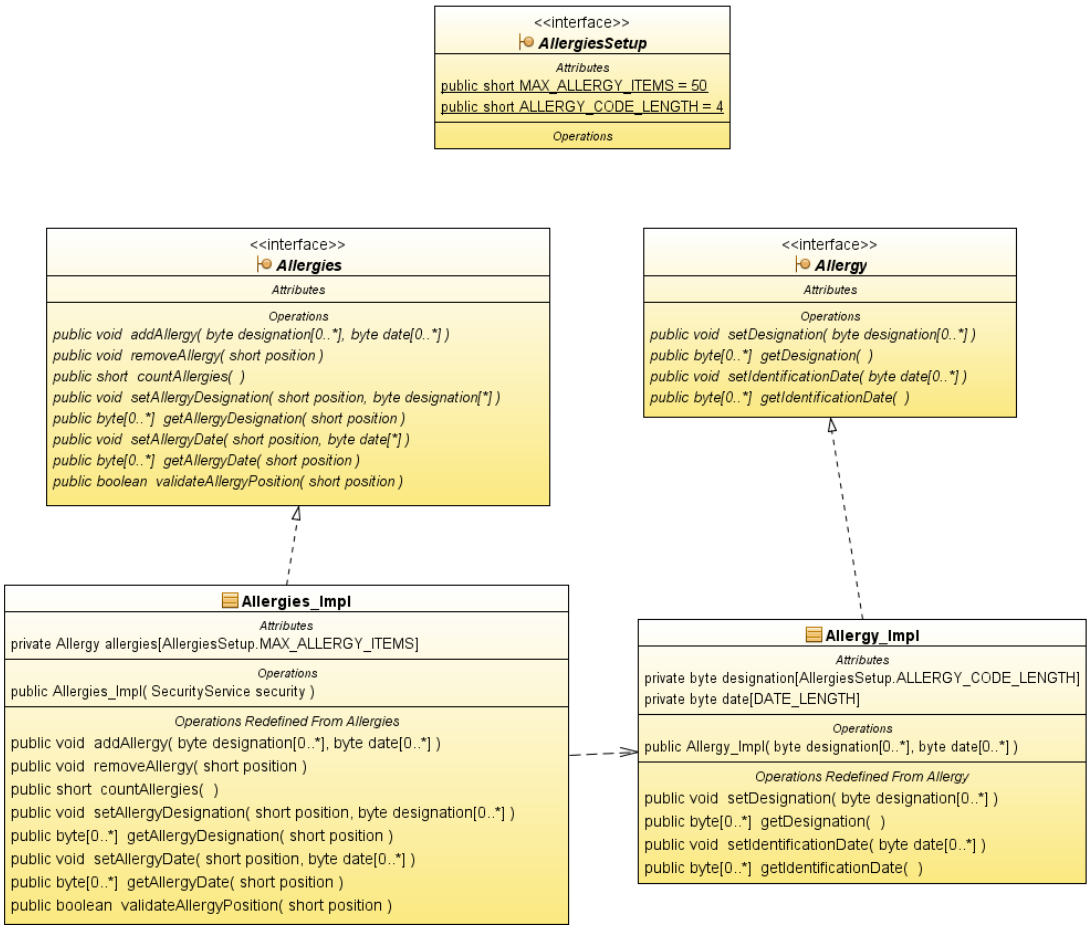

Figure vii. Allergies module Class Diagram

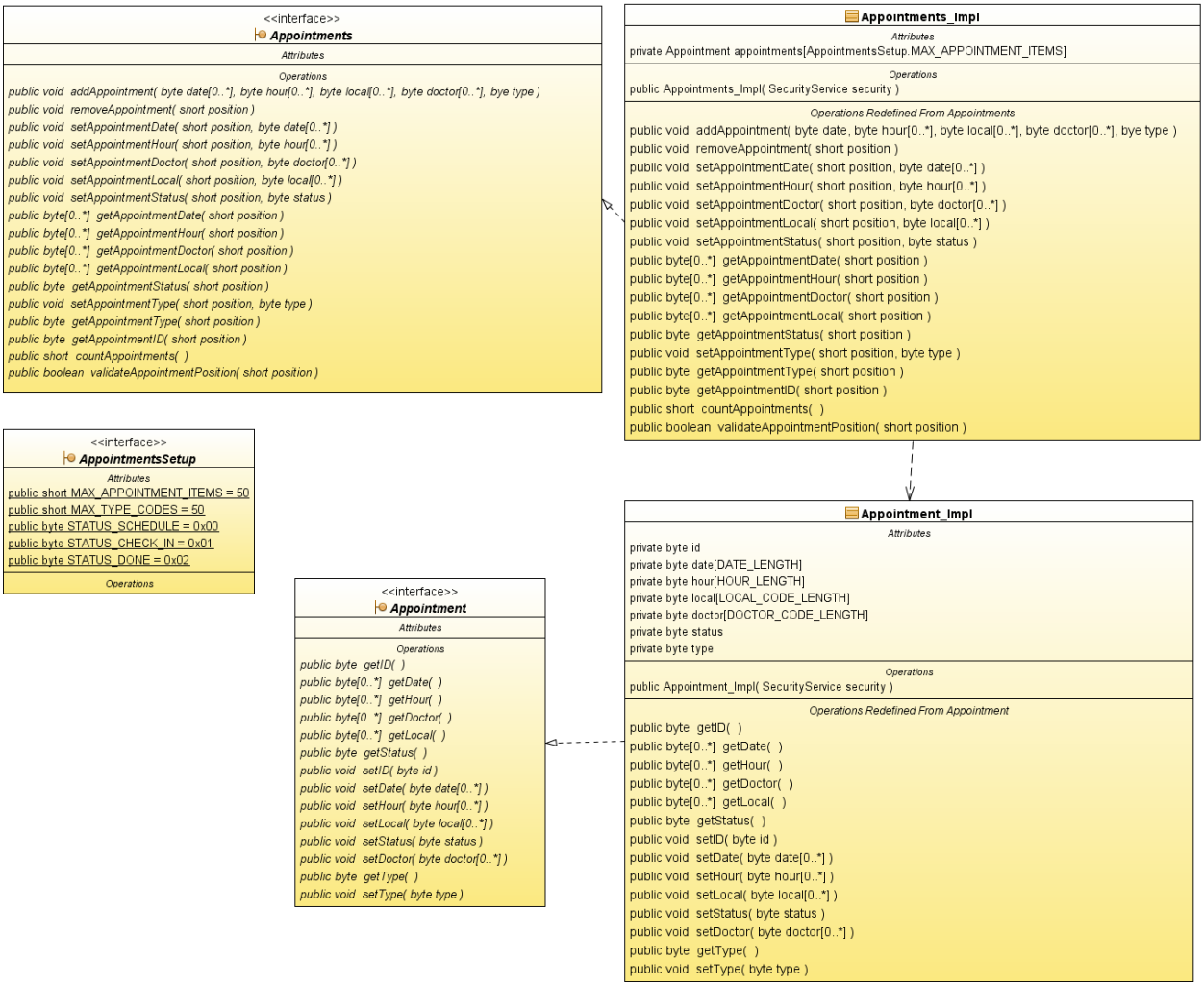

Figure viii. Appointments module Class Diagram

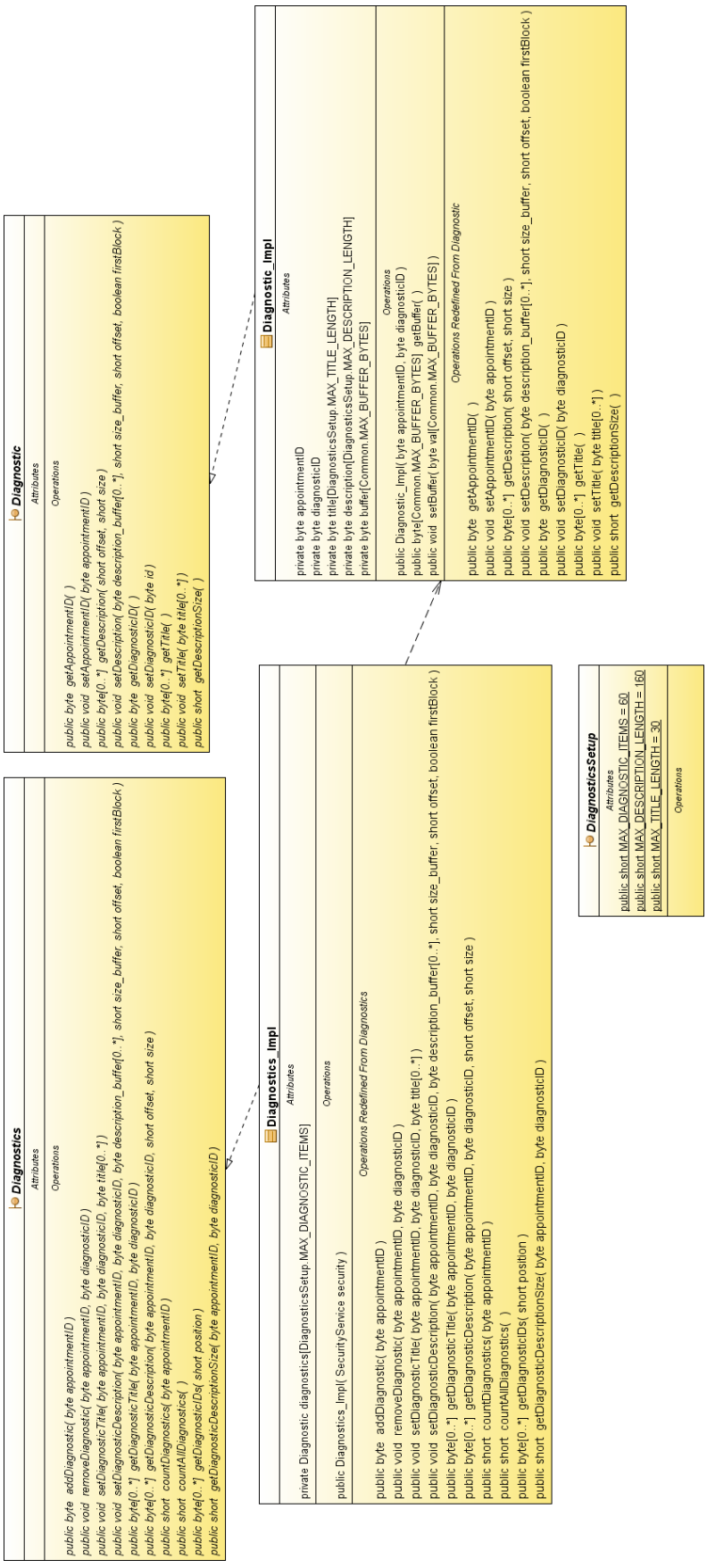

Figure ix. Diagnostics module Class Diagram

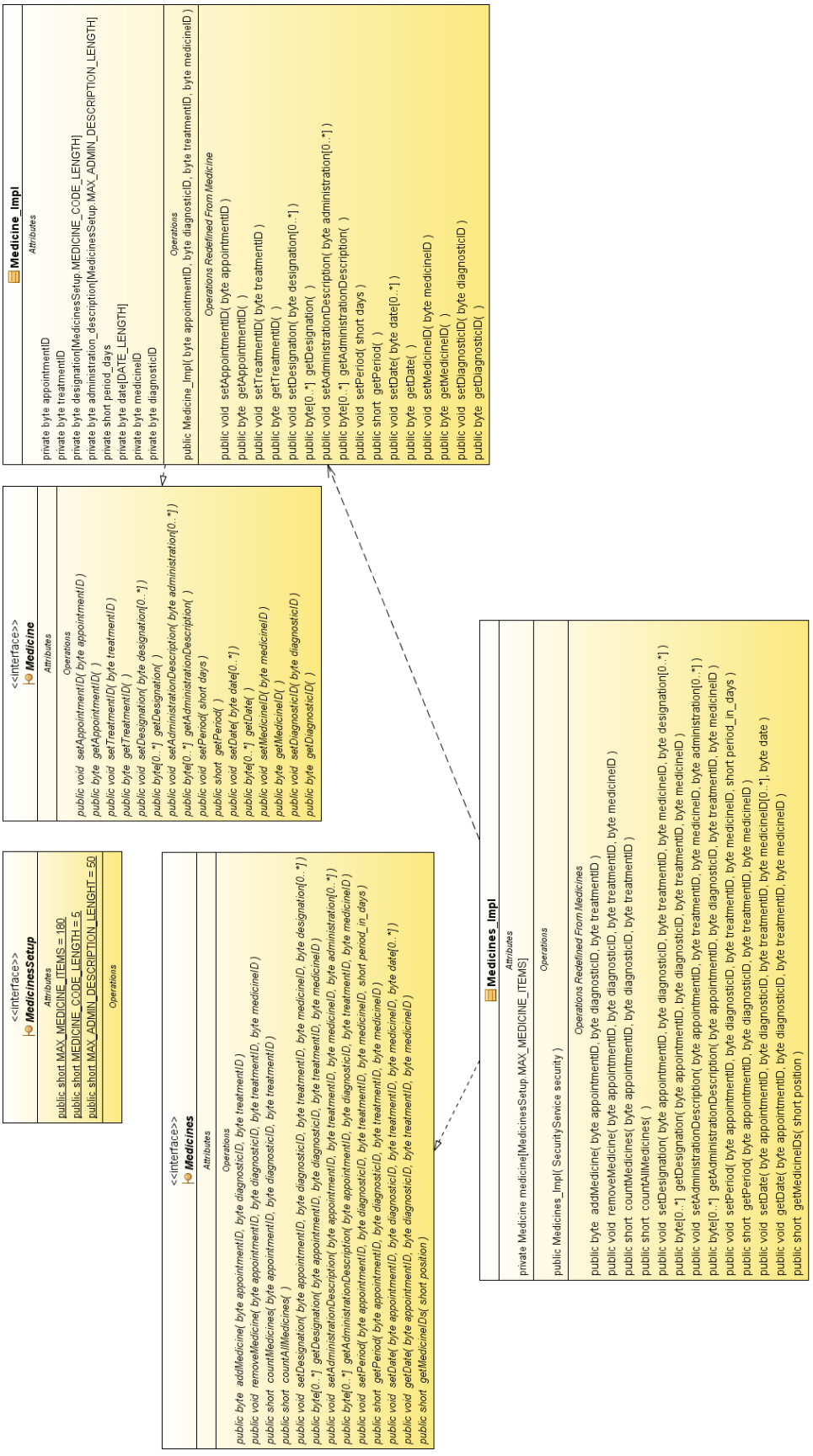

Figure x. Medicines module Class Diagram

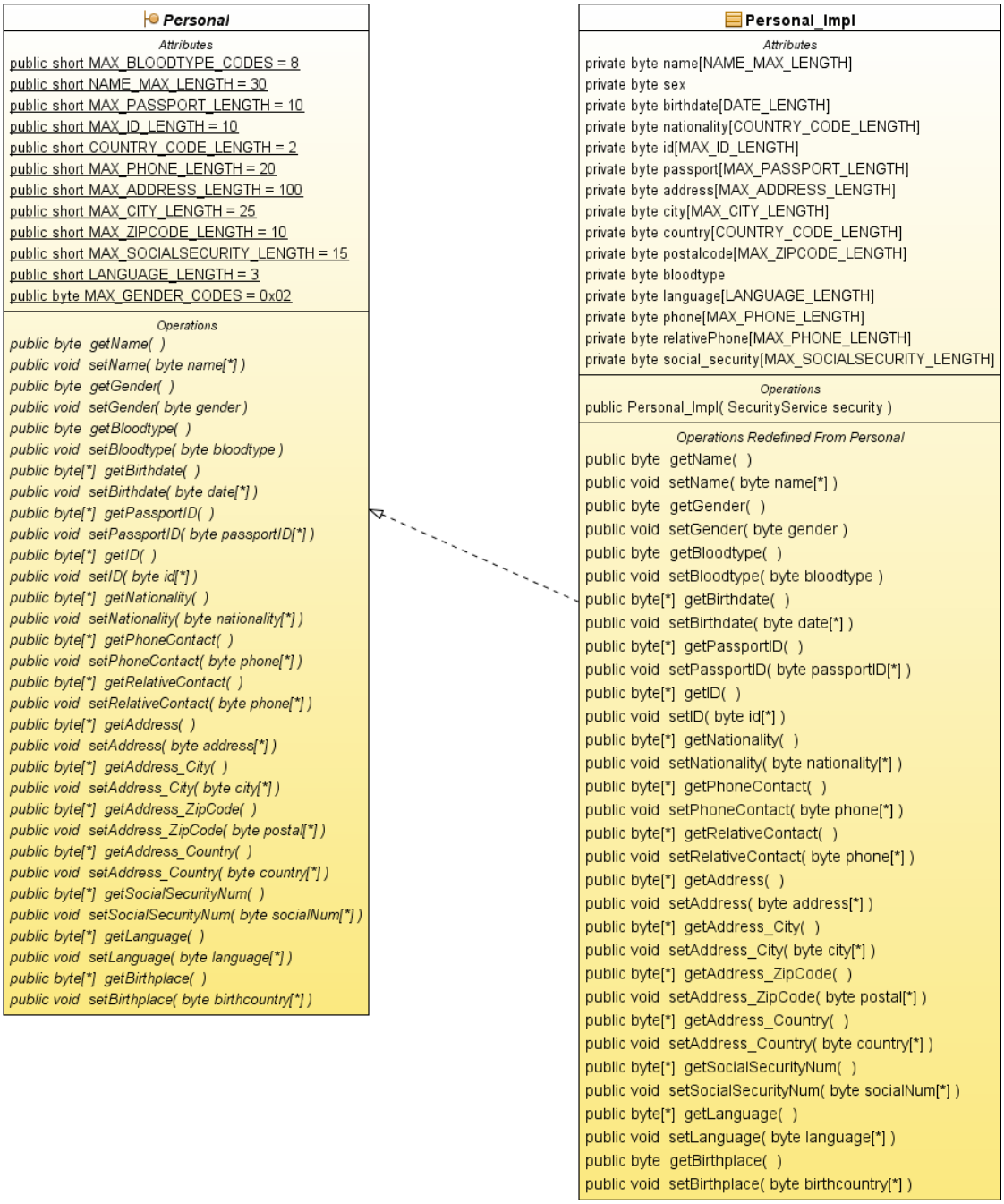

Figure xi. Personal module Class Diagram

Figure xii. Treatments module Class Diagram

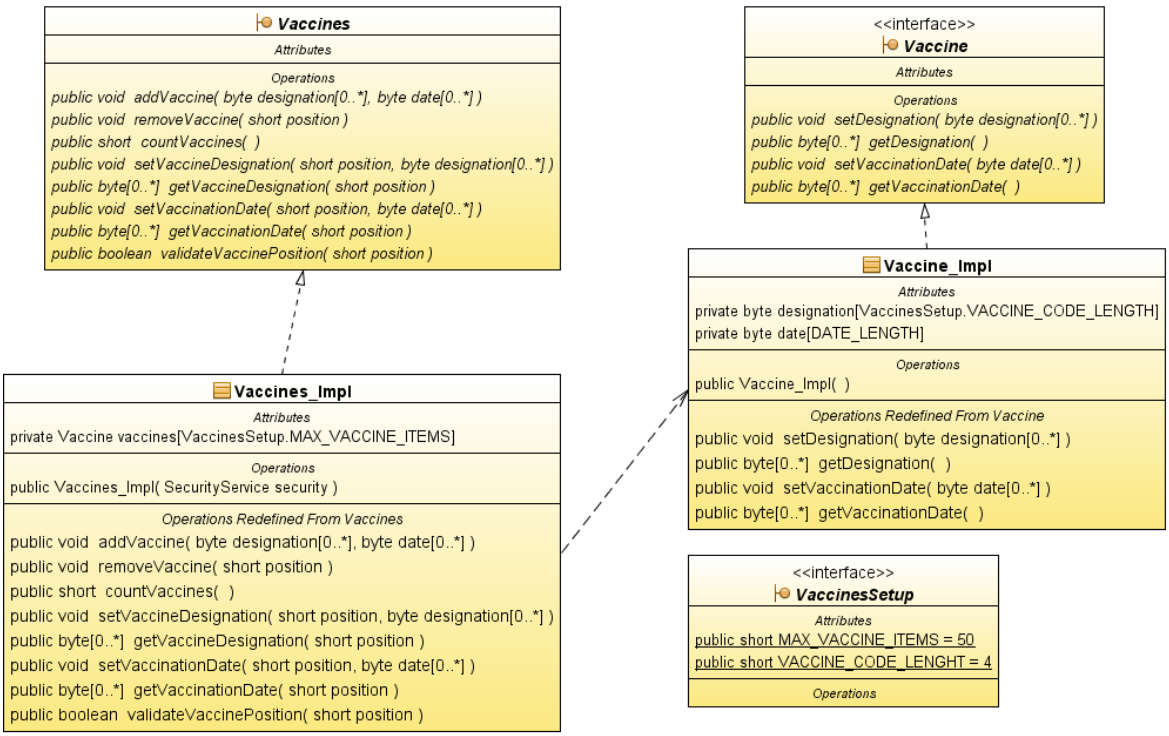

Figure xiii. Vaccines module Class Diagram

8. Tools Manual

During the development of the Health Card application, we spent some time to research and to acquire a minimum experience with JML and Java Card as well as managing the necessary tools. First, although Java Card is a subset language of Java, we had to understand its limitations, because Java Card isn't extensive as Java, we had to learn what we could and what we couldn't do with Java Card. Next, we had to learn how to use the Java Card Remote Method Invocation to establish a communication between external clients and the card applications, and we had to learn how we could simulate a smart card, i.e., the creation of a smart card, the execution of a Java Card application, and how to access the services supplied by that application running in a smart card simulation. Finally, in what respects to our tool and language research, we had to learn the JML basics and how to work with it as well as its tools. In this section we describe how to setup and install the Java Card and JML tools, and how to use them. The Health Card system was developed in a Windows Vista environment; therefore the following described steps are related with it.

8.1. Java Card Installation and Usage

We already described the Java Card in Section **Error! Reference source not found.**, but in this section we present the tools needed for installing and using it, including some necessary steps of its use in the card side and external side.

8.1.1. Installation and setup of Java Card

We start by referencing some of the tools needed for developing a Java Card application. We used these tools to develop the Health Card application (card side):

- Eclipse version 3.2.2. [1]
- Java Card Development Kit version 2.2.2. [2]
- EclipseJCDE, Eclipse plug-in [3]

The Eclipse is an Integrated Development Environment (IDE) that supports the development of software applications in various programming languages, and also supports the IDE extensibility through the installation plug-ins. To install it, you just need to download it and extract it to a system directory, for instance *"C:\"*. The Eclipse will be ready to use afterwards.

We have also used the Java Card Development Kit (JCDK) that comes with tools and libraries necessary for the development and testing of Java Card applications. To install it, one has to download it and extract it to a system directory, for instance it could also be "*C:\"*, and next one has to setup the environment variables of the OS and add the following lines (assuming that JCDK is installed in *"C:\"*):

- *JC_HOME : C:\java_card_kit-2_2_2*
- *Path: %JC_HOME%\bin*

We also installed the *EclipseJCDE* plug-ins (http://eclipse-jcde.sourceforge.net/) for the Eclipse IDE. The Eclipse plug-in allow to apply AID (Application Identifiers) to Java Card class packages and to applets, generate scripts from those packages and execute them by wrapping their APDU commands to communicate with a smart card simulator. Without this plug-in the process of writing the scripts would take much longer and would be tiresome. To install the plug-in, one should download it and uncompress it into the plug-in directory of the Eclipse installation directory. Then, one has to initiate Eclipse and setup the Java Card directory in "Java Card → Preferences" then fill it in "Java Card Home". This has to be the directory where we installed the Java Card Development Kit. [31] In the following we describe some of the features of the plug-in in more detail.

EclipseJCDE has two functionalities for applying AIDs to Java Card packages and applets. These are the "Set Package AID" and "Set Applet AID". Each Java Card applet and package on a card is uniquely identified by an Application ID (AID), by this mechanism one can select the applets to be used when executing the card applications. [2] Also, the *EclipseJCDE* has three more tools to process the Java Card class files into optimized formats for smart cards. These tools are a CAP (*Converted Applet*) file converter, a script generator and a script runner. The CAP file converter, labelled as *"Converter"* is used to convert Java Card class files into CAP files to be fitted into smart cards. The CAP file is a JAR-format file that contains the executable binary representation of the classes in a package. The CAP format is a highly optimized binary format for Java Card systems. Also the Converter generates other two optional files, an EXP

[1] Eclipse IDE: http://www.eclipse.org/downloads/
[2] Java Card Development Kit: http://java.sun.com/products/javacard/
[3] EclipseJCDE, Eclipse plug-in: http://sourceforge.net/project/showfiles.php?group_id=176931

and a JCA file. An EXP file is a Java Card export file that contains the public API linking information of classes in a package, and a JCA is a Java Card assembly file, which could be used to regenerate a CAP file. Besides the converter tool, the *EcpliseJCDE plug-in* provides a script generator tool, labelled as *"Generate Script"*. This script generator produces APDU script files with APDU commands to install applets and packages into the smart cards. Finally, the last tool provided by the plug-in is a script runner labelled as "Run Script" that has the function of sending APDU commands from the scripts generated by the previous tool. [32] One must notice that all this tools for processing the Java Card classes are base in tools provided by the JCDK. The plug-in tools are automatic and easies the process of AID appliance, cap file conversion and script files generation.

The *EclipseJCDE* plug-in also provides two tools used for simulating a Java Card environment of smart cards. These tools are *C Reference Implementation Simulator* (CREF) and *Java Card Workstation Development Environment* (JCWDE). These two tools are available in the JCDK, but one has to use them through command lines. The CREF is implemented in C language and supports the execution of Java Card applications, but it doesn't permit debugging. Through the use of CREF one creates an image file simulating a smart card with the applets installed and a memory space included for holding data. The JCWDE is developed in Java and supports debugging, but it is too limited as it doesn't create any image file of the smart card and it isn't possible to store data while simulating the card execution. In our work we decided to use the first simulating tool, the CREF, because it provided us with a smart card image where we could store data for posterior tests, i.e., we could store information and later read it through the use of external applications. Also one must notice that we have used the CREF tool through command line rather than the graphical version from the plug-in. The reason of this choice was because of the lack of documentation on how we could use the tool through the plug-in, and the difficulty on using by ourselves.

8.1.2. Using the Java Card

After installing and setting up the Java Card environment and its tools, we describe the Java Card usage in the card side and client (external) side.

8.1.2.1. The card side

Using the *Eclipse* and after installing the EclipseJCDE, for creating a new Java Card project one must choose the option "Java Card Project" and give it a name. The needed libraries for developing Java Card applications are automatically added to the project. After implementing a functional Java Card application, one has to compile the Java Card files and has to process them into a format optimized for smart cards.

First, it's necessary to apply an AID to the applet and packages to uniquely identify them. This can be done through the use of *"Set Package AID"* and *"Set Applet AID"* from the EclipseJCDE plug-in. One must notice that the external applications calling the card services from a certain applet must have the same applet AID in their code mechanisms to select a remote applet. The AID is used by external applications to select the respective card applets. In case of having multiple packages, each one of them must have a unique AID, otherwise it will cause conflicts and it won't be possible to convert them into CAP files. [32]

Having applied the AIDs, one compiles the Java Card files like normal Java files. Then one has two options while using the EclipseJCDE plug-in to optimize the files for smart cards. First, one can convert the compiled classes to CAP, EXP and JCA by using the "Converter". By selecting each package, one right click on them and chooses "Java Card Tools → Convert", then the tool outputs three files CAP, EXP, and JCA. Having these files one has to generate the APDU scripts with APDU commands for the applet installation in the smart cards. The second option is more direct and faster, because when using the APDU script generator when selecting the packages, the conversion to CAP, EXP and JCA files is made automatically. To generate the APDU scripts one must select each package and by clicking the right button one chooses "Java Card Tools → Generate Script". This procedure must be done for each package, beginning by the least dependent packages, i.e., if package A depends on package B, one must generate the script of B first. In the end of this process we have the script files and the CAP, EXP and JCA files. The script files created are the "*create-<applet name>.script"* and "*select-<applet name>.script"* for the package containing the applet and "*cap-download.script"* for each package.

After generating the script files we have to edit manually the *"cap-download.script"* of each package. This is basically needed for completing the script that is necessary for sending installation APDU commands into the smart card. For the script files from the package containing the applet, in the *"create-<applet name>.script"*, we have to copy the line referent to the applet creation, i.e., the APDU command line below the comment *"// create <applet name> applet"*, and then we have to paste it in the "*cap-download.script"* of the same package just before the APDU command "0x80 0xBA 0x00 0x00 0x00 0x7F;" and the "powerdown" lines at the end of the file.

The next step, if the Java Card application has multiple packages, is more complex. For each package's script file *"cap-download.script"*, and beginning from the least dependent package, we have to copy all the APDU commands between the "powerup" and "powerdown" lines and then we have to paste it in the same script file where we pasted the create line, i.e., in the *"cap-download.script"* from the applet's package. We paste it after the "powerup" line and before the *"// select the installer applet"* line, moreover those pasted scripts must be ordered by the dependency level of each respective package, i.e., if package A depends on package B, then we have to paste the script lines of B after the "powerup" line and then we have to paste the script lines of A after the script lines of B.

After creating and editing the scripts we end up with a single script file containing the APDU commands from all other scripts. From this point we can execute this script to install the Java Card applet into a smart card or a Java Card environment simulation (i.e., image file simulating a smart card). To simulate a Java Card environment, where we can download the script, we can use the CREF or the JCDWE tools. We have decided to use CREF instead of JCDWE because with the first one it is possible to create an image file where we can store and retrieve later information like it was a smart card, making it a useful way for testing the application before downloading it for a real smart card.

Using the CREF tool through command line we create an image file which waits for the script downloading. First we write by command line the following command *"cref –o <image file name>"*, and then the execution of CREF waits for the script downloading. Next, through Eclipse we select the script containing the APDU commands and by right clicking we choose *"Java Card Tools → Run Script"* to download the script into the image file to install the Java Card application. If all ends well, after downloading the script, we have created the image file simulating a smart card that contains the Java Card application.

When running external Java applications to access the image file, like it was a smart card, we use the CREF tool, but the command line given is *"cref –i <image file name>"*. The execution of CREF waits for external APDU commands sent to the simulated smart card. The command *"-i"* makes the image file readable, but we can make the image file permanently writable (i.e., to hold information after the execution) by adding the command "-o", making possible to store permanently the information inserted in the image file, during the simulation (i.e., *"cref –i –o <image file name>"*).

Next we describe the use of Java Card in external applications, i.e., the client side.

8.1.2.2. The client side

The client side or external applications that will communicate with the card can be implemented in normal Java language, but one has to include the following Java Card Development Kit libraries into the project: *apduio.jar*, *jcclientsamples.jar* and *jcrmiclientframework.jar*. Also, one must notice that the AID used for selecting a card applet must be equal to the one given in the card side. This AID in the client side is given through the selecting mechanism code in the client programming.

Having in mind that this is a system implementing the Java Card RMI (Remote Method Invocation), for each remote interface of the card application, it must be created a stub of their implemented remote objects. This stub must be present in the card side applications to be possible a communication between the client side and the card side (i.e. server side). To create a stub we present the following command line example: - *"rmic -classpath C:\java_card_kit-2_2_2\lib\javacardframework.jar;. card.Personal_Impl"* – where following the –classpath we write the path to the needed Java Card library and *";."* to refer the directory of *card* where the compiled class *Personal_Impl* is. The directory *card* is the package where *Personal_Impl* is and *Personal_Impl* is the implementation of the remote interface *Personal* from the card side. The result of this command line is the creation of a stub file of *Personal_Impl*.

After creating the stubs, we copy them and the remote interfaces to the client side. In our case we have done a *.bat file to execute all this command lines and copy the files automatically, as it is a tiresome task. At this point we have all prepared to execute the external application and establish a connection with the card side, the only thing necessary before running the client applications is to use the CREF tool to execute the image file to simulate the insertion of a real smart card.

8.2. JML Installation and Usage

In this section we describe the tools needed for installing and using the Java Modelling Language as well as some necessary steps of its use.

8.2.1. Installation and setup of JML

The following tools are necessary to install and use de JML:

- Java Runtime Environment version 1.4.* (JML only works with this version)

- Eclipse version 3.4.1. (The only compatible version with the existing JML plug-ins for Eclipse.)
- JML Common Tools [4] (we used version 5.4.)
- JML2 Eclipse Plug-in Project
- JUnit 3 [5]

This Eclipse IDE used for working with JML is a different version to that used for working with Java Card. This is because of each plug-in of Java Card and JML being neither compatible with the same Eclipse version. We had to write the JML specifications and Java Card code in the Eclipse version with the JML plug-in installed, and the Eclipse version with the Java Card plug-in installed was only used for writing the Java Card applet and security coding, and running the Java Card application. To install the Eclipse we only have to download it and extract it in a directory.

The JML Common Tools is a suite of basic tools for supporting the JML usage. We have already described it in Section **Error! Reference source not found.**. To install the JML Common Tools we need to download them and extract them in a directory (for example, *"C:\JML"*) and next we have to setup the necessary environment variables of the OS as it follows:

- *Path: %JML_HOME%\bin*
- *JML_HOME : C:\JML*

And, in the *classpath* environment variable we have to add the following:

- …JML\bin\jml-release.jar; …j2re1.4.2_18\lib\rt.jar; …j2re1.4.2_18\lib\sunrsasign.jar; …j2re1.4.2_18\lib\jsse.jar; …j2re1.4.2_18\lib\jce.jar; …j2re1.4.2_18\lib\charsets.jar; …j2re1.4.2_18\classes; ...JML\specs\; …JML\org\jmlspecs\models; …java_card_kit-2_2_2\lib\api.jar

One must notice that we have also installed Java Runtime Environment version 1.4.2. - update 18, because it was the latest compatible version with the most recent version of the JML Common Tools (at the time it was 5.4.). At last, we have to make sure that the Eclipse has the JUnit 3. The JUnit 3 is needed for executing the Runtime Assertion Checking. The JUnit is a unit testing framework for the Java programming language and along with the *jmlunit* tool from JML Common Tools we can make runtime assertion checks. For more information about Runtime Assertion Checking and used tools, see Section **Error! Reference source not found.**.

The JML2 Eclipse Plug-in Project is an Eclipse plug-in that integrates the JML Common tools into the Eclipse. Through this plug-in, the process of writing the JML specifications is easier, especially its functionality of checking the JML syntax while writing the specifications. To install this plug-in we have to start Eclipse and then we have to go to "help → software updates… → Available Software → Add Site". In there we insert the following website http://www.pm.inf.ethz.ch/research/universes/tools/eclipse/ to obtain the JML2 Eclipse Plug-in Project.

[4] The JML Common Tools: http://sourceforge.net/projects/jmlspecs/files/
[5] JUnit: http://www.junit.org/

To setup the automatic JML checker for automatically make the static checking of JML specifications we have to right click on a project and select Properties, the we have to select "JML2 Plug-in → Automatically run JML2 Checker". Now the Eclipse will warn the developer when he makes any JML syntax mistake while writing the specifications, and also makes automatically a static checking of the JML specifications and the Java code. This plug-in has a tool for static checking labelled as JML2 checker" and has another tool for compiling java code with jml specification labelled as "JML2 compiler". These two tools are based on the JML Common Tools *jml* for the static checking and the *jmlc* for compiling. We were particularly interested in the use of the automatic JML2 checker tool as it facilitated our work while writing the JML specifications.

Also, before using this Eclipse plug-in for working with JML we need to include the following libraries into the Eclipse project:

…JML\bin\jml-release.jar
… JML\specs
… java_card_kit-2_2_2\lib\apduio.jar
…java_card_kit-2_2_2\lib\api.jar

8.2.2. Using the JML Common Tools

The JML Common Tools are described in Section **Error! Reference source not found.**. To use the JML Common Tools we may execute the *jml-release.jar* to launch a graphical version, or we can use its tools through command lines.

Once we have Java code specified with JML we may use the *jml* tool from the common tools to make a static assertion checking. The alternative way of making a static assertion checking is to use the automatic tool supplied by the JML2 Eclipse Plug-in. This alternative is better because while one writes the JML specifications, the tool checks automatically.

To compile Java files with JML we can use the plug-in function "JML2 Compiler" or we can compile through the *jmlc* tool from the common tools. This compiles like the compilation of normal Java files but with the addition of JML. The compilation is necessary before executing a runtime assertion checking. If using the jmlc to compile, one has to copy and paste the resulting class files to the *"…/bin"* directory of the working Eclipse project and replace any previous compiled file. By default the compiled files are put in the bin directory and we have to replace those compiled files because normally the Eclipse compiles automatically as normal Java files when we save the changes in the code (or in JML specifications).

To make a runtime assertion checking of the JML specifications and implementation code we have to generate the testing files and provide some test data, after compiling the Java files with JML. This process is described in Section **Error! Reference source not found.**.

8.2.3. Other Tools for JML

Besides the tools that we described and used, there are other tools that allow us to test and verify Java code with JML specifications.

8.2.3.1. Krakatoa

The *Krakatoa* is a verification tool for Java/Java Card programs specified in JML. This tool focuses on verifying the correctness of implementations according to pre and post-conditions (specified in JML) and class invariants. This tool is from the *Why* platform for deductive program verification (this platform includes tools for software verification in Java and C languages). The *Krakatoa* has three components [33]:

- The *Krakatoa* tool, to read the Java/Java Cards and produce their specifications for *Coq* and a representation of semantics of the Java/Java Card program into *Why's* input language.
- The *Coq* proof assistant, for specification modeling and development of proofs.
- The *Why* tool, for computing proof obligations for a core imperative language annotated with pre and post conditions.

8.2.3.2. ESC/Java2

The *Extended Static Checker for Java version 2* (ESC/Java2) is a programming tool statically checks for common run-time errors in JML-annotated Java programs. The static check is made by a static analysis of the program code to verify if it meets with its formal annotations. The amount and kinds of static checks can be controlled by the users. The Users can control these checks that ESC/Java2 performs, by annotating their Java/Java Card programs with specially formatted comments called *pragmas* [17].

ESC/Java2 can be used through three forms:

- The ESC/Java2 built into the Mobius Program Verification Environment[6], which is is integrated with Eclipse.
- The ESC/Java2 as a command-line tool[7] with a simple Swing GUI front-end.
- The ESC/Java2 as an Eclipse 3.5 plug-in[8].

8.2.3.3. JACK

The JACK (Java Applet Correctness Kit) tool also provides an environment for verification of Java/Java Card applications specified with JML. This tool makes an automated weakest precondition calculus that generates proof obligations from annotated Java or Java Card sources. Later, these proof obligations can be proven through different theorem checkers (for example, the Coq proof assistant).

The JACK developers claim that the most important design goal of JACK is that it is easy to use by normal Java developers to validate their own code. That is, the JACK tool is design to hide mathematical complexity of the underlying concepts, simplifying and facilitating the Java

[6] Mobius Program Verification Environment: http://kind.ucd.ie/products/opensource/Mobius/
[7] ESC/Java2 command-line version, source and binary versions: http://secure.ucd.ie/products/opensource/ESCJava2/download.html
[8] ESC/Java2 plug-in for Eclipse 3.5: http://kind.ucd.ie/products/opensource/Mobius/updates/

programs verification. The JACK provides a graphical viewer that presents the proof obligations connected to execution paths within the verified program, where for each proof obligation, the related source code is highlighted. The JACK tool is available as an Eclipse IDE plug-in[9]. [34]

8.2.3.4. JMLE

The jmle tool is a Java based tool created by Professor Tim Wahls and Ben Krause. This tool is available in the latest JML Common Tools pack and it is an adaptation of the jmlc tool (the JML tool that generates runtime assertion checking code, see Section **Error! Reference source not found.**) in order to compile specifications written in JML by translating them to constraint programs, which are then executed via the Java Constraint Kit (JCK)[10] which is a system for creating Java implementations of constraint solvers. One important thing to notice is that the jmle tool ignores completely any Java code in method bodies when compiling Java or JML files, as its purpose is only to make the JML specifications executable. Having executable specifications makes the formal specifications more useful and easier to develop.

The jmle automatically compiles JML specifications to JCK programs as follows: - a JML class specification is compiled to a Java class, then only the specification model fields become actual fields of this Java class and each JML method specification is compiled to a method implementation. After compiling the specifications, they can then be executed using normal Java “driver” code along with JCK libraries that creates an instance of the class and calls its methods, by writing JUnit tests for the class specification, or in any other manner that a hand-coded implementation could be used. [35]

The main difference between the use of the jmle and jmlc tools is that the jmle only compiles JML specifications for making them executable, while jmlc compiles Java code with JML specifications for runtime assertion checking.

[9] JACK tool and its Plug-ins for Eclipse IDE: http://www-sop.inria.fr/everest/soft/Jack/download.html
[10] Java Constraint Kit libraries: http://www.pms.ifi.lmu.de/software/jack/index.html