Concordia University

Team BladeRunners

SAD Document

Submitted by group of:

Alexandre Hudon
Matthew Tam
Christopher Di Fulvio
Simon Symeonidis
Yuri Kitaev
Mitchell Smolash
Jaspreet Singh Lidder

FSTS
Software Architecture Document
Version 1.1

Revision History

Date	Version	Description	Author
11/03/12	1.0	First coverage of the SAD document	Team BladeRunners
28/03/12	1.1	Updated SAD Document with latest ADs	Team BladeRunners

Table of Contents

Software Architecture Document

1. Introduction

1.1 Purpose

This document provides a comprehensive architectural overview of the system, using a number of different architectural views to depict different aspects of the system. It is intended to capture and convey the significant architectural decisions which have been made on the system.

1.2 Scope

The Software Architecture Document comprises all the relevant architectural designs and choices that are used in the development of the FSTS.

1.3 Definitions, Acronyms and Abbreviations

Please see the Project's Glossary provided in the SRS document <bladerunners_srs_sprint4.pdf>.

1.4 References

[1] WPI, <SAD Template>, Software Architecture Document (March 1, 2012): http://web.cs.wpi.edu/~gpollice/cs4233-a05/rup_sad.html

1.5 Overview

The Software Architecture Document presents the architectural representation of the system as well as the architectural goals and constraints. It also provides different views of the software such as Use-Case View, Logical View, Process View, Deployment View , Implementation View and Data View. It finally concludes with a justification on the choice of our architecture to build the new FSTS.

2. Architectural Representation

The Architectural Representation is the 4+1 view model. The Use Case View (the "+1" from the "4+1") is the set of use cases that explain and describe the functionality and expected responses from the system when given input from the user, as well as external systems. Sequences of such interactions and responses are illustrated in this view through the use of UML Use Case diagrams. The Logical View is the view that shows the internal interactions between components to realize the use cases described by the previous mentioned view, the Use Case View. This view also shows the public interface provided by the system at various levels, to illustrate how these components interact with each other and what functionality can be used from each component. That is, this view exposes the abstractions provided by specific components, such that those functionalities are exposed publically. This view is illustrated by UML Class Diagrams, Communication Diagrams, and Sequence Diagrams. The next view, the Implementation View is the view that shows all of the difference components of the system that,

together, make up the physical system. This view is shown through UML Package Diagrams and Component Diagrams. The process view is shown through an activity diagram to illustrate the different flows of execution. The final view of the 4+1, is the Physical View. This view shows the different components of the physical system, and the software built on top of them, and the forms of communication between the two levels (hardware and software). This view is represented through UML Deployment Diagrams.

3. Architectural Goals and Constraints

The FSTS requested by WHM requires security and efficiency as their main objectives. Because WHM deals with some personal information such as Medicare numbers, names and addresses of their clients, security needs to be addressed to protect that information. As well, the application is a management tool, therefore efficiency is a high priority. Another characteristic of this system is that it is meant for WHM use only and needs to run on a windows platform.

However, the client's software requirements and objectives have more of an impact on the architecture depending on which approach is taken. Our team went with a standalone application as opposed to an online web based application. This removes security as a concern since the system set up at WHM is based on thin clients that requires user authentication for access. Therefore, security is handled by WHM.

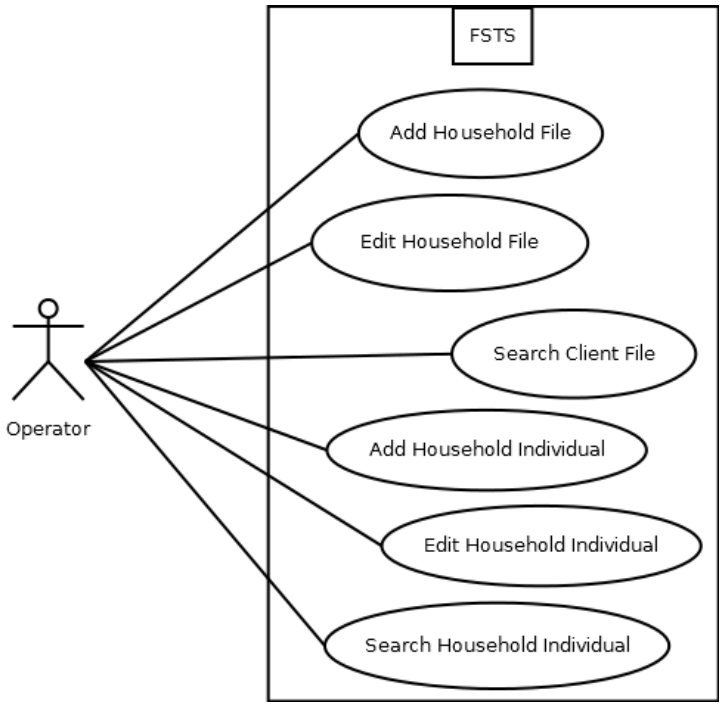
With the remaining requirements, a variety of architectures are available that satisfies the customer's conditions; our group decided on the three layered architecture. This is an architecture that all members in our group were familiar with and is one that allows for concurrent development because of the separation of concerns between the Presentation, Domain and Persistence layers. Specifically, we are applying an opaque layering, which allows our application to be more maintainable as coupling is low; layers can only access the layer immediately below it. Although a transparent layering is more efficient, there isn't much of a tradeoff in efficiency because we are developing a small application. The decision for improved maintainability is also for unexpected changes, such as changing requirements. As we are in an agile setting with clients that did not differentiate needs and wants, we had to develop with change in mind.

4. Use-Case View

4.1 Use-Case Realizations

CLIENT FILE MANAGEMENT

Use Case Model



Use Cases

Add Household File

ID	UC1.01
Use Case	Add Household File
Description	Operator creates a new household file which relates many individuals to a similar address.
Level	Sea
Primary Actor	Operator
Supporting Actors	N/A
Stakeholders and Interests	Operator wants to create a new household file in order to record information about clients.
Pre-conditions	N/A
Post Conditions	Success end condition: System adds the household item successfully. Failure end condition: System is unable to add the household item. The system prompts the operator about the issue.

	Minimal Guarantee: The information is retained on the client part of the system so that the operator may retry to send the information.
Main Success Scenario	1. Operator requests the household addition view from system 2. System displays the household addition view 3. Operator inserts all the required information for the form 4. System saves the file
Extensions	3.a Invalid Medicare Entered 3.a.1 The system prompts the user that the inserted medicare number is erroneous. 3.a.2 The system ends with a failure end condition
Special Requirements	N/A

Edit household File

ID	UC1.02
Use Case	Edit Household Files
Description	Operator updates Household files on the FSTS application.
Level	Sea
Primary Actor	Operator
Supporting Actors	None
Stakeholders and Interests	Operator: Their interest is to manage Household presence during events by sending via wireless device attendance confirmations directly to Household file.
Pre-conditions	• Desired Household file exists
Post Conditions	Success end condition: System performs update to a Household's file. Failure end condition: Operator is notified that the system failed to commit updates to a Household's file. Minimal Guarantee: The client file remains in the current database without changes. The new edited information remains in the current view.
Main Success Scenario	1. Operator opens event system for the CFM. 2. Operator searches for a Household file. 3. System returns collections of matching household files 4. Operator selects the desired Household file. 5. System returns the selected household file details 6. Operator edits the required fields of the Household 7. System saves the new information.
Extensions	2.a Connection to the database backend fails 2.a.1 System prompts the operator that there is a problem with the database connectivity 2.a.2 Use case ends with a failure end condition 7.a Connection to the database backend fails 7.a.1 System prompts the operator that there is a problem with the database connectivity 7.a.2 Altered information is retained in the the view form 7.a.3 Use case ends with a failure end condition
Special Requirements	Whenever the Operator checks off households, the system must immediately give feedback to confirm proper updates have been received and done in the FSTS.

Search Client File

ID	UC1.03
Use Case	Search Client File
Description	Operator searches client file by specifying partial information.
Level	Sea
Primary Actor	Operator
Supporting Actors	N/A
Stakeholders and Interests	Operator: Their interest is to manage Household presence during events by sending via wireless device attendance confirmations directly to Household file.
Pre-conditions	• Desired Household file exists
Post Conditions	Success end Condition: System retrieves collection of matching client files to the operator's input. Failure end condition: No client files displayed. Minimal Guarantee: No client files displayed.
Main Success Scenario	1. Operator specifies that a search is needed 2. Operator inserts the partial information which is provided 3. Operator queries the system 4. The system returns the collection of client files matching provided information
Extensions	N/A
Special Requirements	N/A

Add Household Individual

ID	UC1.04
Use Case	Add Household Individual
Description	Operator adds a household individual to the individuals list.
Level	Sea
Primary Actor	Operator
Supporting Actors	N/A
Stakeholders and Interests	Operator: Their interest is to add individuals to the household.
Pre-conditions	• Desired Household file exists • Desired Household file is in editing mode
Post Conditions	Success end Condition: System adds the individual to the household. Failure end condition: System fails to add the individual to the household Minimal Guarantee: The client file remains the way it was originally opened as. The added individual remains on the list
Main Success Scenario	1. Operator specifies to the system that an individual is to be added. 2. Operator fills in the required information for the individual form. 3. Operator confirms individual addition 4. System adds the individual to household
Extensions	3.a: Incorrect medicare

	3.a.1 The system prompts the user that the medicare number is incorrect 3.a.2 The system ends with failure end condition 3.b: Existing Medicare 3.b.1 The system prompts the user that the medicare already exists 3.b.2 The system ends with a failure end condition
Special Requirements	N/A

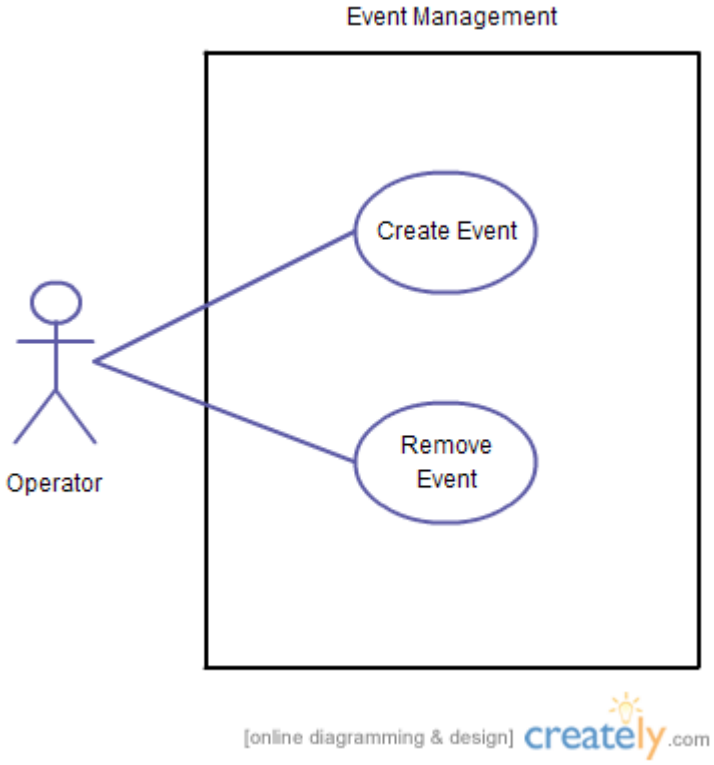
Edit Household Individual

ID	UC1.05
Use Case	Edit Household Individual
Description	Operator adds a household individual to the individuals list.
Level	Sea
Primary Actor	Operator
Supporting Actors	N/A
Stakeholders and Interests	Operator: Their interest is to edit existing individuals in the household.
Pre-conditions	● Desired Household file exists ● Desired Household file is in editing mode ● Desired Individual is in edit mode
Post Conditions	Success end Condition: System updates the individual to the household. Failure end condition: System fails to update the individual to the household Minimal Guarantee: The individual updated information remains updated on the client. The individual information remains unchanged on the backend.
Main Success Scenario	1. Operator selects desired Individual 2. Operator alters desired information of individual 3. System updates the individuals information.
Extensions	2.a: Database Connectivity Error 2.a.1 System prompts the operator that there is a problem with the database connectivity 2.a.2 Use case ends with a failure end condition
Special Requirements	N/A

Search Household Individual

ID	UC1.06
Use Case Name	Search Household Individual
Description	Operator searches for household individual in system.
Level	Sea
Primary Actor	Operator
Supporting Actor(s)	N/A
Stakeholders and Interests	Operator: Wants to find a individual belonging to a certain household in the system.
Preconditions	● Individual exists in the system. ● Individual belongs to a household.
Post Conditions	Success end condition: ● Household individuals best matching the search will be displayed in a list. Failure end condition: ● No household individuals are displayed. Minimal Guarantee: ● Result List will be displayed.
Main Success Scenario	1. Operator writes household individual's name in search field. 2. Operator executes search. 3. System populates results list with household individuals best matching search field.
Extensions	N/A
Special Requirements	N/A

EVENT MANAGEMENT



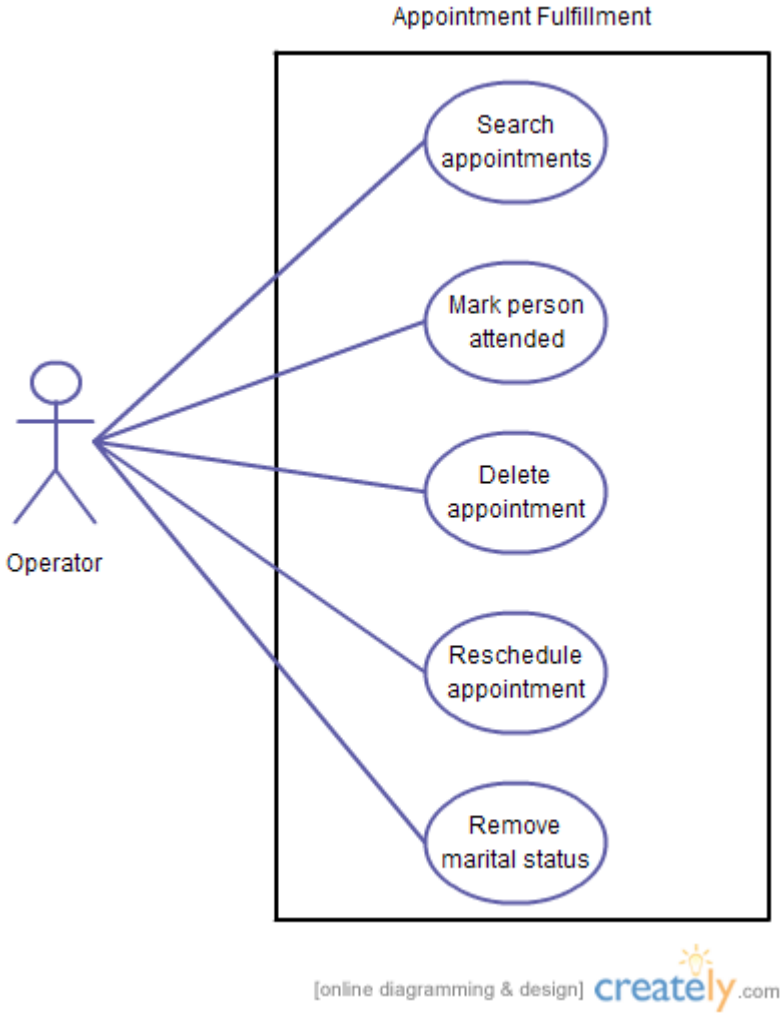
ID	UC2.01
Use Case Name	Create event
Description	The operator wants to create a new event so that this event can be registered for by the households.
Level	User Level
Primary Actor	Operator
Supporting Actor(s)	None
Stakeholders and Interests	Operator: Wants to create a new event Household : Wants to go to an event
Preconditions	N/A
Post Conditions	Success end condition: • The new event is created and saved to persistent storage. • The selected time slots for the event are created and saved to persistent storage. Failure end condition: • The new event is not added. • No new time slots are added.

	Minimal Guarantee: ● Data in persistent storage remains accurate.
Main Success Scenario	1. Operator selects an event type from the list of event types 2. Operator selects the start and end date of the new event 3. Operator selects which days of the week the event will take place 4. System auto populates the list of dates 5. Operator selects the start time and end time of the event 6. Operator selects an interval that time slots should be created 7. Operator confirms the addition of the new event
Extensions	1a: The event type needed does not yet exist in the system 1a1. Operator requests to make a new event type 1a2. Operator enters the name of the new event type 1a3. Operator confirms the changes 1a4. System refreshes the event type list with the new event 7a: An event with the date and event type specified already exists in the system 7a1. System prompts the user that the date already has an event of the same type 7a2. Use case ends with failure
Special Requirements	● The dates entered cannot be in the past. ● The dates start date cannot be after the end date.

ID	UC2.02
Use Case Name	Remove event
Description	The operator wants to remove an event so that appointments cannot be made for this event
Level	User Level
Primary Actor	Operator
Supporting Actor(s)	None
Stakeholders and Interests	Operator: Wants to delete an event
Preconditions	● An event with the event type and date that needs to be deleted already exists ● Event has no appointments associated to it
Post Conditions	Success end condition: ● The event is removed from the system, and deleted from the persistent storage. Failure end condition: ● The event is not removed from the system or deleted from permanent storage. Minimal Guarantee:

	• All households that were attending the event will still exist and function correctly for other functionalities. • Data in persistent storage remains accurate.
Main Success Scenario	1. Operator selects to expand the list of available dates for the event 2. Operator selects the date to delete and presses the delete button 3. System prompts the user with a dialog for confirmation 4. Operator confirms the deletion 5. System refreshes the event list, with the deleted event no longer in the list
Extensions	4a: Event has appointments associated to it 4a1. System asks for confirmation to delete appointments 4a2. Operator confirms for appointments to be deleted 4a3. System deletes the appointments and removes them from persistent storage
Special Requirements	None

APPOINTMENT FULFILLMENT



ID	UC3.01
Use Case Name	Search Appointments
Description	The operator wants to search appointments in order to find the one for the current individual
Level	User Level
Primary Actor	Operator
Supporting Actor(s)	N/A
Stakeholders and Interests	Operator: Wants to find appointment Individual: The person who attends the appointment, the one being searched for
Preconditions	● The appointment exists ● The individual is a member of the household for which the appointment was created
Post Conditions	Success end condition: ● The list of appointments returned includes the one being searched for Failure end condition: ● The list of appointments returned does not include the one being searched for Minimal Guarantee: ● The data of all appointments being searched for is not changed
Main Success Scenario	1. Operator enter the Event name and date 2. Operator enters the individual's medicare # 3. The system returns a list of appointments with the entered information 4. The list contains the sought-after appointment
Extensions	3a: The operator enters the individual's surname 3a1. Continue from step 3 3b: The operator enter the individual's first name 3b1. Continue from step 3
Special Requirements	N/A

ID	UC3.02
Use Case Name	Mark Individual as Attended
Description	Mark the individual who is a member of the household for whom the appointment was created as "attended"
Level	User Level
Primary Actor	Operator

Supporting Actor(s)	N/A
Stakeholders and Interests	Operator: wants to mark the appointment as fulfilled Individual: is attending his appointment
Preconditions	• The appointment exists • The individual is a member of the household for the appointment • The appointment has relevant information to be found
Post Conditions	Success end condition: The appointment is marked as having been attended Failure end condition: The appointment is not marked as having been attended Minimal Guarantee: All information regarding the individual is not changed.
Main Success Scenario	1. The operator searches for the appointment (UC3.01) 2. The operator selected the correct appointment from the search results 3. The operator presses the attended button to confirm 4. The system marks the appointment as attended
Extensions	N/A
Special Requirements	N/A

ID	UC3.03
Use Case Name	Delete Appointments
Description	Delete an appointment for a household/event
Level	User Level
Primary Actor	Operator
Supporting Actor(s)	N/A
Stakeholders and Interests	Operator: wants to delete the Appointment Household: has their appointment removed from the system
Preconditions	• The appointment exists • The appointment is in the future
Post Conditions	Success end condition: The appointment is deleted from the system Failure end condition: The appointment is not deleted from the system Minimal Guarantee: The appointment is deleted from the system
Main Success Scenario	1. The operator searches for the appointment (UC3.01)

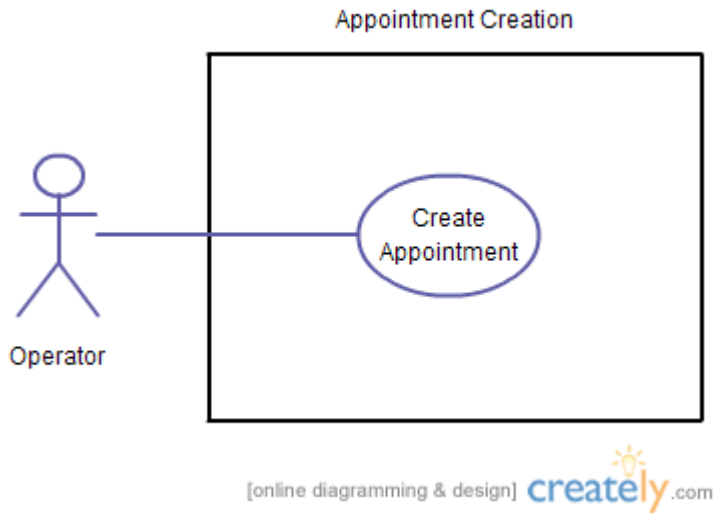
	2. The operator selects the relevant appointment from the search results 3. The operator presses the "Delete Appointment" button 4. The system deletes the appointment from the persistent storage
Extensions	N/A
Special Requirements	N/A

ID	UC3.04
Use Case Name	Reschedule Appointment
Description	Reschedule an Appointment for another date/time combination
Level	User Level
Primary Actor	Operator
Supporting Actor(s)	N/A
Stakeholders and Interests	Operator: wants to reschedule an Appointment Household: has their appointment rescheduled in the system
Preconditions	● The appointment exists ● The appointment is in the future
Post Conditions	Success end condition: The appointment is rescheduled for another date/time combination Failure end condition: There are no other times available; the Appointment can not be rescheduled Minimal Guarantee: If there is a time slot available, the Appointment can be rescheduled
Main Success Scenario	1. The operator searches for the appointment (UC3.01) 2. The operator selects the relevant Appointment from the search results 3. The operator presses "Reschedule" 4. The system prompts the user with a dialog window 5. The operator selects a valid date from the Calendar 6. The operator selects a valid time from the dropdown list 7. The operator presses "confirm" 8. The system removes the old appointment and adds the new one into the persistent storage
Extensions	6a. The operator presses cancel, to avoid the current changes
Special Requirements	N/A

ID	UC3.05
Use Case Name	Print Attendee List

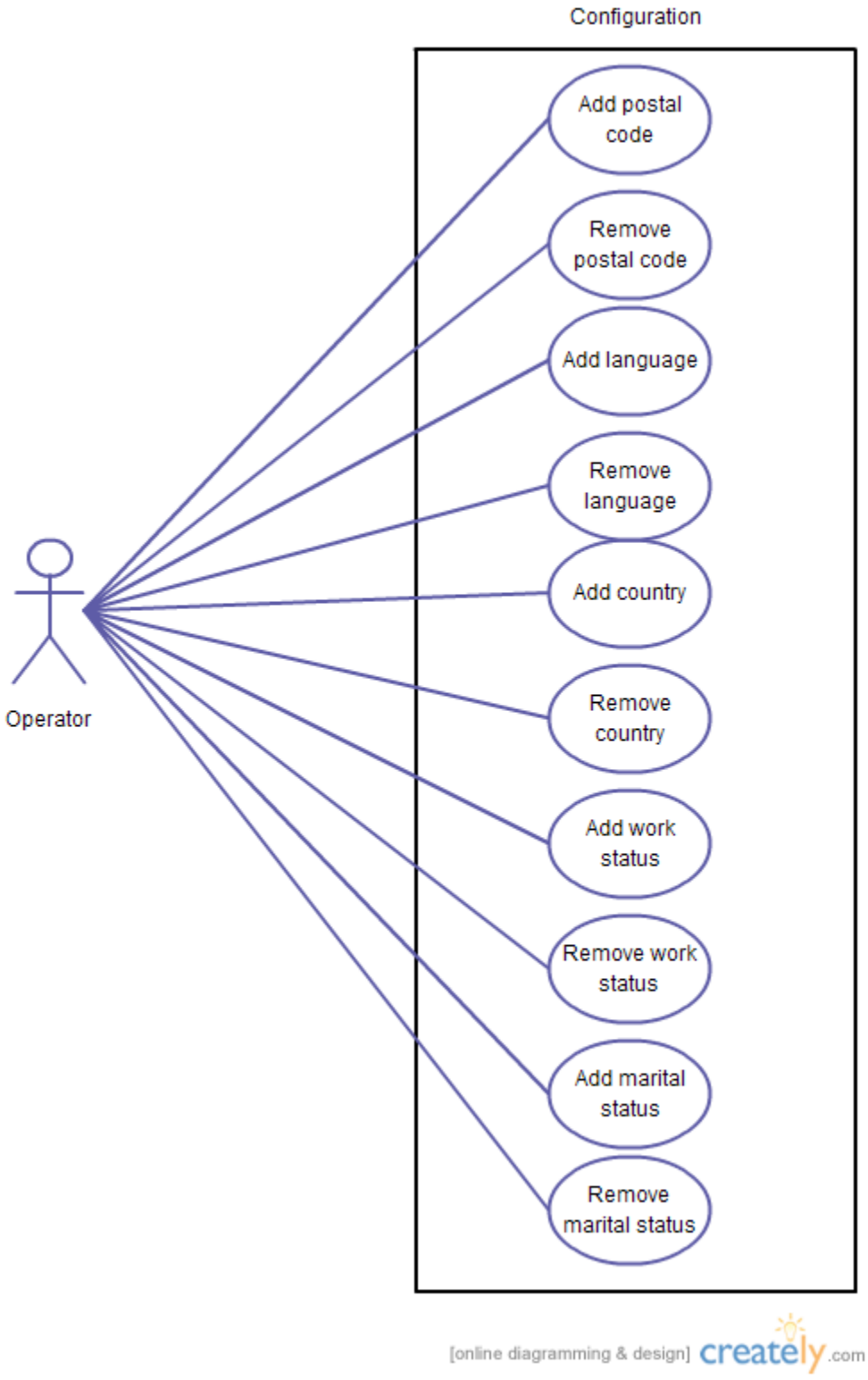
Description	Print the list of all attendees for an event in order to mark them as attended manually
Level	User Level
Primary Actor	Operator
Supporting Actor(s)	N/A
Stakeholders and Interests	Operator: wants to print a list of all of the attendees that are supposed to show up for an event
Preconditions	• The Event exists • There are Appointments associated with the Event
Post Conditions	Success end condition: The list is printed Failure end condition: The list is not printed Minimal Guarantee: The information for each appointment is not changed.
Main Success Scenario	1. The operator opens the Search 2. The operator enters only the event and date 3. The operator presses the "Print" button 4. The system shows a window with the results, formatted with Crystal Reports
Extensions	N/A
Special Requirements	N/A

APPOINTMENT CREATION



ID	UC4.01
Use Case Name	Create Appointment
Description	The operator wants to create an appointment for a household
Level	User Level
Primary Actor	Operator
Supporting Actor(s)	None
Stakeholders and Interests	Operator: Wants to create an appointment Household: Is receiving an appointment for an event
Preconditions	● An event has already been created (UC2.01)
Post Conditions	Success end condition: ● An appointment is created for the household. Failure end condition: ● An appointment was not created for the household. Minimal Guarantee: ● The information of the household is not changed.
Main Success Scenario	1. Operator searches for a file. (UC1.03) 2. Operator chooses to make a new appointment 3. System opens the make new appointment dialog 4. Operator chooses the appropriate event type, date and time slot and presses the OK button 5. System prompts the operator for confirmation 6. Operator confirms the appointment
Extensions	1a: If the file does not yet exist 1a1. Operator creates a new file (UC1.01) 1a2. Return to step 1
Special Requirements	N/A

CONFIGURATION



ID	UC5.01
Use Case Name	Add Postal Code
Description	Add a postal code to the list of known postal codes
Level	User Level
Primary Actor	Operator
Supporting Actor(s)	N/A
Stakeholders and Interests	Operator: Wants to add a postal code
Preconditions	● N/A
Post Conditions	Success end condition: ● The postal code is added Failure end condition: ● The postal code is not added Minimal Guarantee: ● All other information on the configuration panel is unchanged
Main Success Scenario	1. The operator opens the configuration window 2. The system shows the configuration window 3. The operator presses the "Postal Codes" menu option 4. The system shows the Postal Code view 5. The operator enters a postal code in the postal code text box 6. The operator enters a city in the city text box 7. The operator presses the add button 8. The system adds the new postal code to the persistent storage
Extensions	5a. The operator presses enter on their keyboard 1a1. Return to step 6
Special Requirements	N/A

ID	UC5.02
Use Case Name	Remove Postal Code
Description	Remove a postal code from the list of known postal codes
Level	User Level
Primary Actor	Operator
Supporting Actor(s)	N/A

Stakeholders and Interests	Operator: wants to remove the postal code
Preconditions	● There exists a postal code to remove
Post Conditions	Success end condition: ● The postal code is removed Failure end condition: ● The postal code is not removed Minimal Guarantee: ● All other information on the configuration panel is unchanged
Main Success Scenario	1. The operator opens the configuration window 2. The system shows the configuration dialog 3. The operator presses the "Postal Codes" menu option 4. The system shows the Postal Code view 5. The operator selects the desired postal code 6. The operator presses "Deleted Selected" 7. The operator selects "OK" at the prompt to confirm deletion 8. The selected postal code is deleted from the persistent storage
Extensions	7a. The operator selects "Cancel" to cancel the current changes 7a1. The selected postal code is not deleted
Special Requirements	N/A

ID	UC5.03
Use Case Name	Add Language
Description	Add a language to the list of known languages
Level	User Level
Primary Actor	Operator
Supporting Actor(s)	N/A
Stakeholders and Interests	Operator: wants to add a language
Preconditions	● N/A
Post Conditions	Success end condition: ● The language is added Failure end condition: ● The language is not added Minimal Guarantee: ● All other information on the configuration panel is unchanged
Main Success Scenario	1. The operator opens the configuration window

	2. The system shows the configuration dialog 3. The operator opens the Drop Downs submenu 4. The system shows the Drop downs view 5. The operator presses the "Language" menu option 6. The operator enters a language in the text box 7. The operator presses the add button 8. The new language is added to the persistent storage
Extensions	7a. The operator presses enter on their keyboard 1a1. Return to step 6
Special Requirements	N/A

ID	UC5.04
Use Case Name	Delete Language
Description	Remove a language from the list of known languages
Level	User Level
Primary Actor	Operator
Supporting Actor(s)	N/A
Stakeholders and Interests	Operator: wants to remove the language
Preconditions	● There exists a language to remove
Post Conditions	Success end condition: ● The language is removed Failure end condition: ● The language is not removed Minimal Guarantee: ● All other information on the configuration panel is unchanged
Main Success Scenario	1. The operator opens the configuration window 2. The system shows the configuration dialog 3. The operator opens the Drop Downs submenu 4. The operator presses the "Language" menu option 5. The system shows the Language view 6. The operator selects the desired language 7. The operator presses the "Delete Selected" button 8. The operator selects "OK" to confirm the deletion 9. The language is deleted from the persistent storage
Extensions	8a. The operator selects "Cancel" to cancel the deletion process 8a1. The language is not deleted
Special Requirements	N/A

ID	UC5.05
Use Case Name	Add Country of Origin
Description	Add a Country of Origin to the list of known countries
Level	User Level
Primary Actor	Operator
Supporting Actor(s)	N/A
Stakeholders and Interests	Operator: wants to add a Country of origin
Preconditions	● N/A
Post Conditions	Success end condition: ● The country is added Failure end condition: ● The country is not added Minimal Guarantee: ● All other information on the configuration panel is unchanged
Main Success Scenario	1. The operator opens the configuration window 2. The system shows the configuration dialog 3. The operator opens the Drop Downs submenu 4. The system shows the Drop Downs view 5. The operator presses the "Country of Origin" menu option 6. The operator enters a country in the text box 7. The operator presses the add button 8. The new country is added to the persistent storage
Extensions	7a. The operator presses enter on their keyboard return to step 6
Special Requirements	N/A

ID	UC5.06
Use Case Name	Delete Country of Origin
Description	Remove a country from the list of known countries
Level	User Level
Primary Actor	Operator
Supporting Actor(s)	N/A
Stakeholders and Interests	Operator: wants to remove the country

Preconditions	● There exists a country to remove
Post Conditions	Success end condition: ● The country is removed Failure end condition: ● The country is not removed Minimal Guarantee: ● All other information on the configuration panel is unchanged
Main Success Scenario	1. The operator opens the configuration window 2. The system shows the configuration dialog 3. The operator opens the Drop Downs submenu 4. The system shows the Drop Downs view 5. The operator presses the "Country of Origin" menu option 6. The operator selects the desired country 7. The operator presses the "Delete Selected" button 8. The operator selects "OK" to confirm the deletion 9. The country is deleted from the persistent storage
Extensions	8a. The operator selects "Cancel" to cancel the deletion process 8a1. The country is not deleted
Special Requirements	N/A

ID	UC5.07
Use Case Name	Add Work Status
Description	Add a Work Status to the list of known Work Status'
Level	User Level
Primary Actor	Operator
Supporting Actor(s)	N/A
Stakeholders and Interests	Operator: wants to add a Work Status
Preconditions	● N/A
Post Conditions	Success end condition: ● The Work Status is added Failure end condition: ● The Work Status is not added Minimal Guarantee: ● All other information on the configuration panel is unchanged
Main Success Scenario	1. The operator opens the configuration window 2. The system shows the configuration dialog

	3. The operator opens the Drop Downs submenu 4. The system shows the Drop Downs view 5. The operator presses the "Work Status" menu option 6. The operator enters a Work Status in the text box 7. The operator presses the add button 8. The new Work Status is added to the persistent storage
Extensions	7a. The operator presses enter on their keyboard return to step 6
Special Requirements	N/A

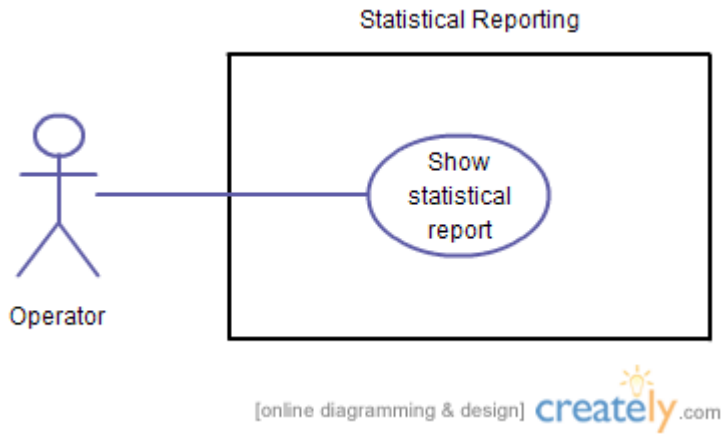
ID	UC5.08
Use Case Name	Delete Work Status
Description	Remove a Work Status from the list of known Work Status'
Level	User Level
Primary Actor	Operator
Supporting Actor(s)	N/A
Stakeholders and Interests	Operator: wants to remove the Work Status
Preconditions	● There exists a Work Status to remove
Post Conditions	Success end condition: ● The Work Status is removed Failure end condition: ● The Work Status is not removed Minimal Guarantee: ● All other information on the configuration panel is unchanged
Main Success Scenario	1. The operator opens the configuration window 2. The system shows the configuration dialog 3. The operator opens the Drop Downs submenu 4. The system shows the Drop Downs view 5. The operator presses the "Work Status" menu option 6. The operator selects the desired Work Status 7. The operator presses the "Delete Selected" button 8. The operator selects "OK" to confirm the deletion 9. The Work Status is deleted from the persistent storage
Extensions	8a. The operator selects "Cancel" to cancel the deletion process 8a1. The Work Status is not deleted
Special Requirements	N/A

ID	UC5.09
Use Case Name	Add Marital Status
Description	Add a Marital Status to the list of known Marital Status'
Level	User Level
Primary Actor	Operator
Supporting Actor(s)	N/A
Stakeholders and Interests	Operator: wants to add a Marital Status
Preconditions	● N/A
Post Conditions	Success end condition: ● The Marital Status is added Failure end condition: ● The Marital Status is not added Minimal Guarantee: ● All other information on the configuration panel is unchanged
Main Success Scenario	1. The operator opens the configuration window 2. The system shows the configuration dialog 3. The operator opens the Drop Downs submenu 4. The system shows the Drop Downs view 5. The operator presses the "Marital Status" menu option 6. The operator enters a Work Status in the text box 7. The operator presses the add button 8. The new Marital Status is added to the persistent storage
Extensions	7a. The operator presses enter on their keyboard 7a1. Return to step 8
Special Requirements	N/A

ID	UC5.10
Use Case Name	Delete Marital Status
Description	Remove a Marital Status from the list of known Work Status'
Level	User Level
Primary Actor	Operator
Supporting Actor(s)	N/A
Stakeholders and Interests	Operator: wants to remove the Marital Status
Preconditions	● There exists a Marital Status to remove

Post Conditions	Success end condition: ● The Marital Status is removed Failure end condition: ● The Marital Status is not removed Minimal Guarantee: ● All other information on the configuration panel is unchanged
Main Success Scenario	1. The operator opens the configuration window 2. The system shows the configuration dialog 3. The operator opens the Drop Downs submenu 4. The system shows the Drop Downs view 5. The operator presses the "Marital Status" menu option 6. The operator selects the desired Marital Status 7. The operator presses the "Delete Selected" button 8. The operator selects "OK" to confirm the deletion 9. The Marital Status is deleted from the persistent storage
Extensions	8a. The operator selects "Cancel" to cancel the deletion process 8a1. The Marital Status is not deleted
Special Requirements	N/A

REPORTING



ID	UC6.01
Use Case Name	Show statistical report
Description	View Statistical data in predefined format.
Level	User Level
Primary Actor	Operator
Supporting Actor(s)	N/A

Stakeholders and Interests	Operator: wants to view statistical data about the clients, appointments and events.
Preconditions	● Client File, Events and appointments exist in the system.
Post Conditions	Success end condition: ● Statistical Report is displayed. Failure end condition: ● Statistical Report is not displayed. Minimal Guarantee: ● System attempts to compile report.
Main Success Scenario	1. Operator selects Statistical reports. 2. System compiles report 3. System presents statistical report.
Extensions	2a: System fails to compile report. 2a1. System displays an error message.
Special Requirements	N/A

5. Logical View

5.1 Overview

Vertical Partitioning (Layers)

Forms Presentation Layer
Model Domain Layer
Persistence Persistence Layer

Horizontal Partitioning (Packages within layers)

Presentation Layer:

AppointmentFulfilment Appointment rescheduling, deleting, and fulfillment.
ClientFileManagement Client File Management
ClientFileSearch Client File Search
EventManagement Event Type, Event Instance (Series) and Timeslot Creation
EventRegistration Registration of a Family to an Existing Event
Misc Configuration
Reporting Report Viewers

Domain Layer

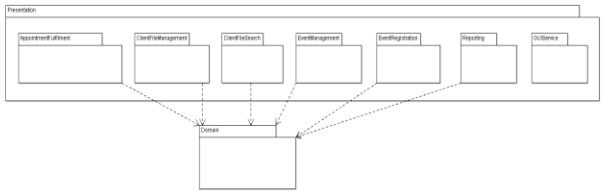
Facades Facades that Abstract the Domain Layer Interface
helpers Models of simple the Real-World Concepts (eg. Medicare number, Postal Code etc)
model Models of composite and of Architecturally-Significant Real-World Concepts (eg. Household)

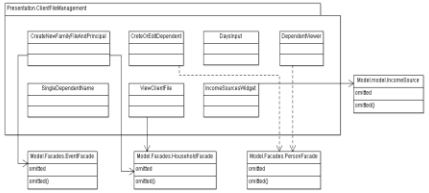
Persistence Layer:

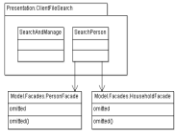
Same as Presentation

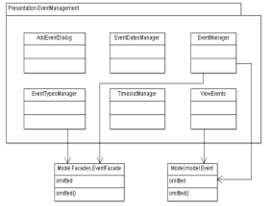
5.2 Architecturally Significant Design Packages

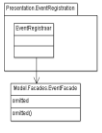
Presentation

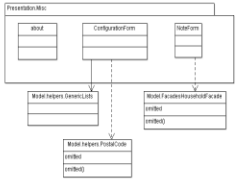


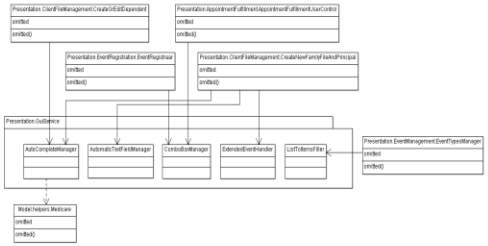


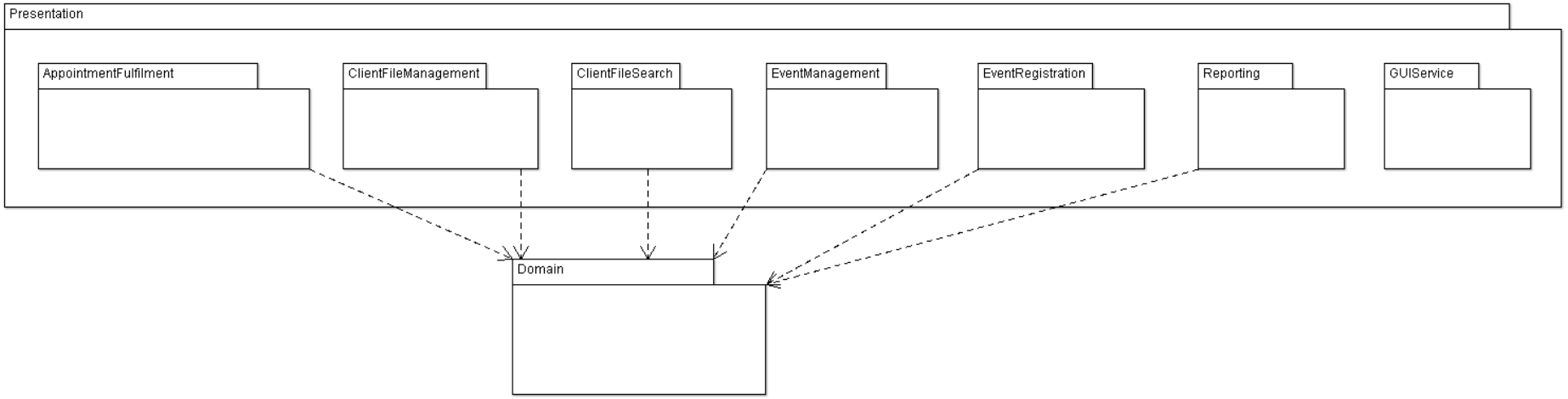


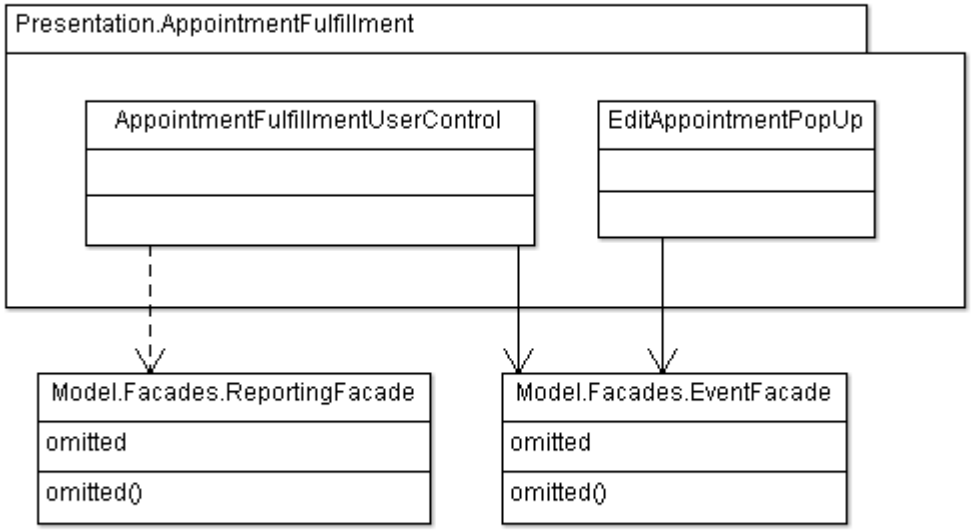


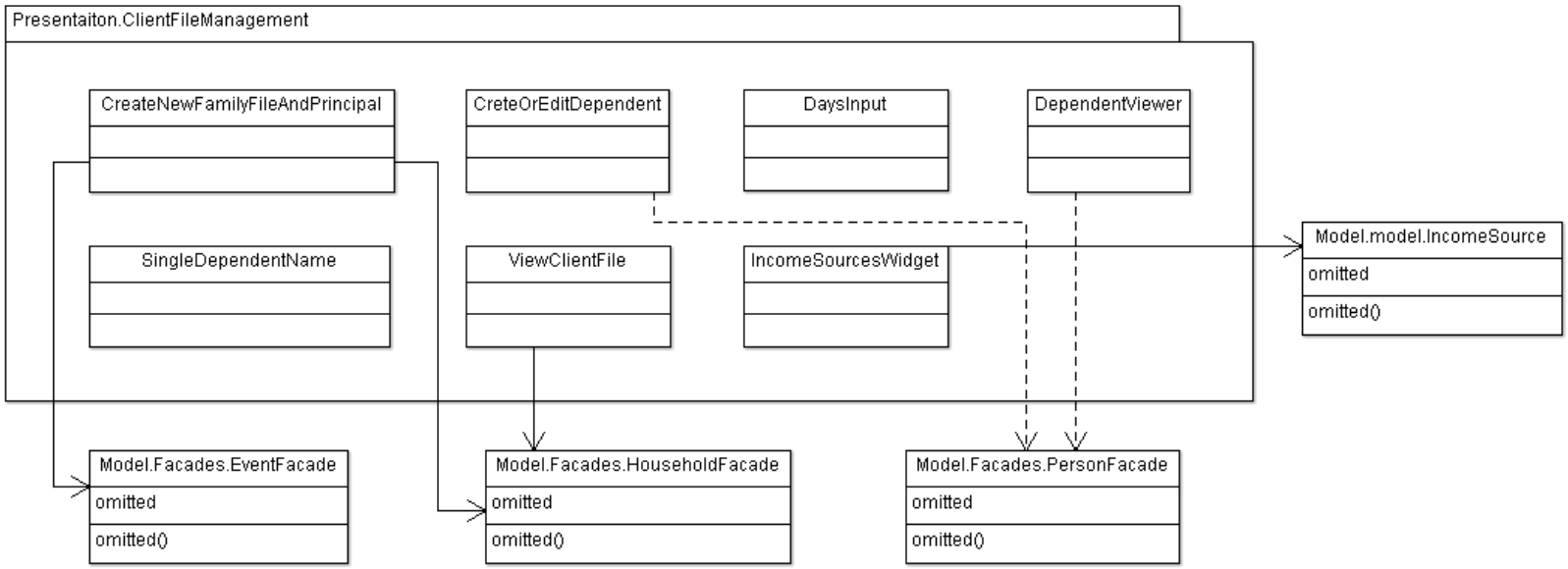


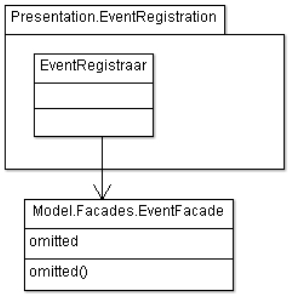


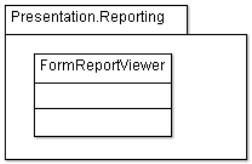


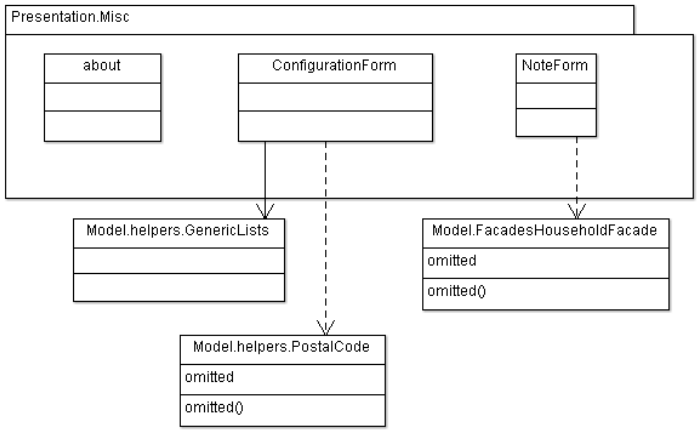

Presentation
AppointmentFulfilment
ClientFileManagement
ClientFileSearch
EventManagement
EventRegistration
Reporting
GUIService
Domain


Presentation.AppointmentFulfillment
AppointmentFulfillmentUserControl
EditAppointmentPopUp
Model.Facades.ReportingFacade
omitted
omitted()
Model.Facades.EventFacade
omitted
omitted()

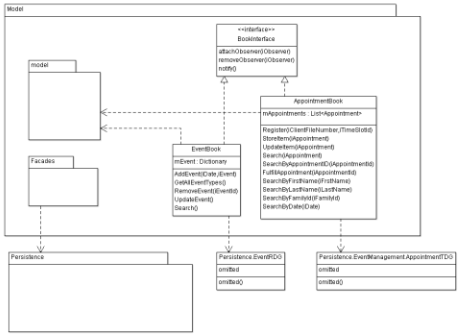

Presentaiton.ClientFileManagement
CreateNewFamilyFileAndPrincipal
CreteOrEditDependent
DaysInput
DependentViewer
SingleDependentName
ViewClientFile
IncomeSourcesWidget
Model.model.IncomeSource
omitted
omitted()
Model.Facades.EventFacade
omitted
omitted()
Model.Facades.HouseholdFacade
omitted
omitted()
Model.Facades.PersonFacade
omitted
omitted()

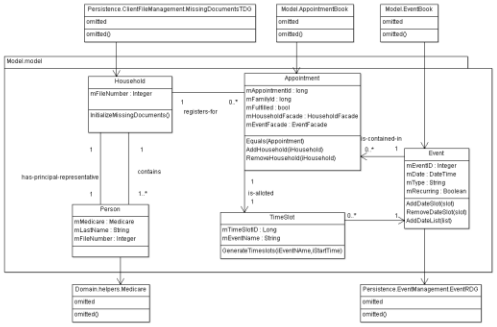

Presentation.EventRegistration
EventRegistraar
Model.Facades.EventFacade
omitted
omitted()

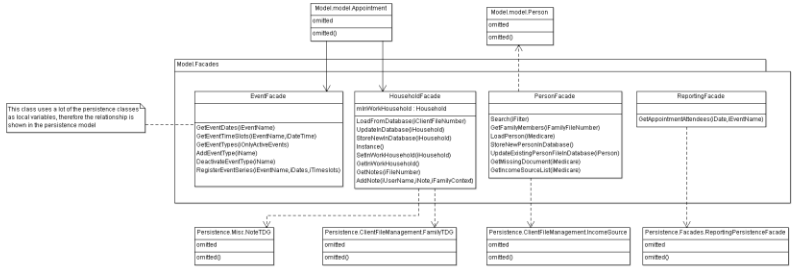

Presentation.Reporting
FormReportViewer

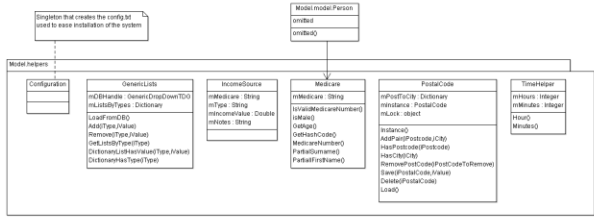

Presentation.Misc
about
ConfigurationForm
NoteForm
Model.helpers.GenericLists
Model.FacadesHouseholdFacade
omitted
omitted()
Model.helpers.PostalCode
omitted
omitted()

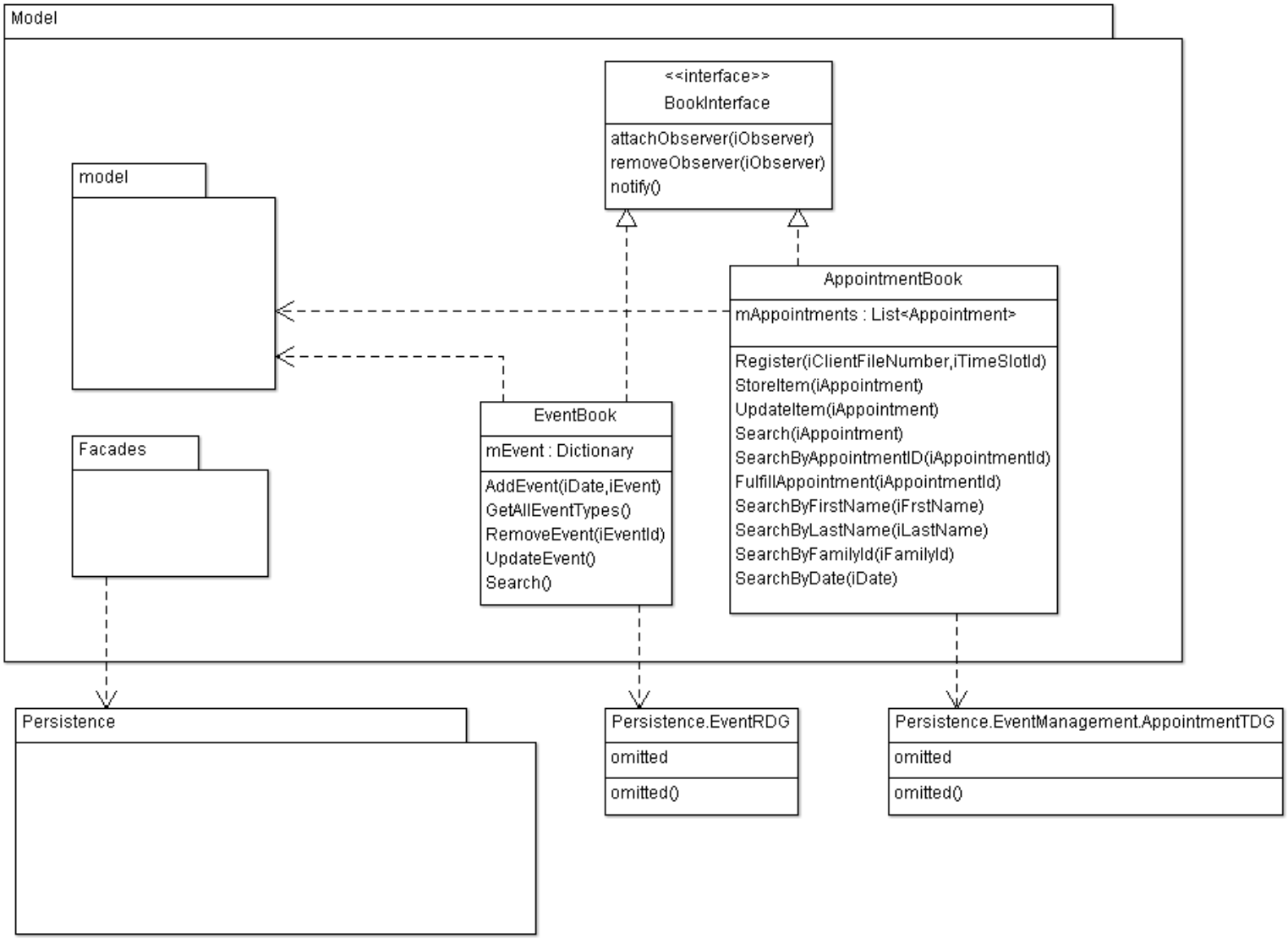
Domain Layer

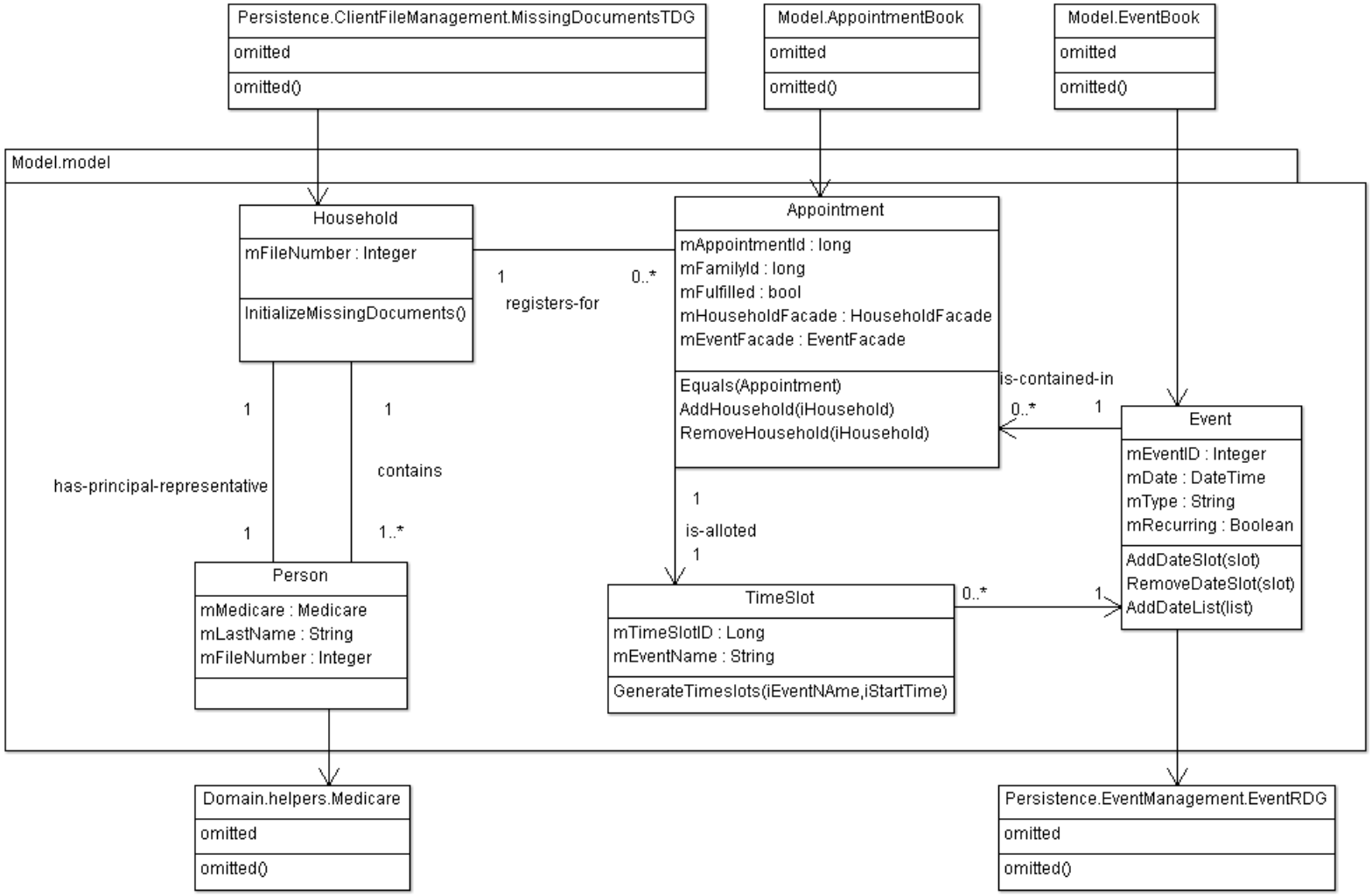


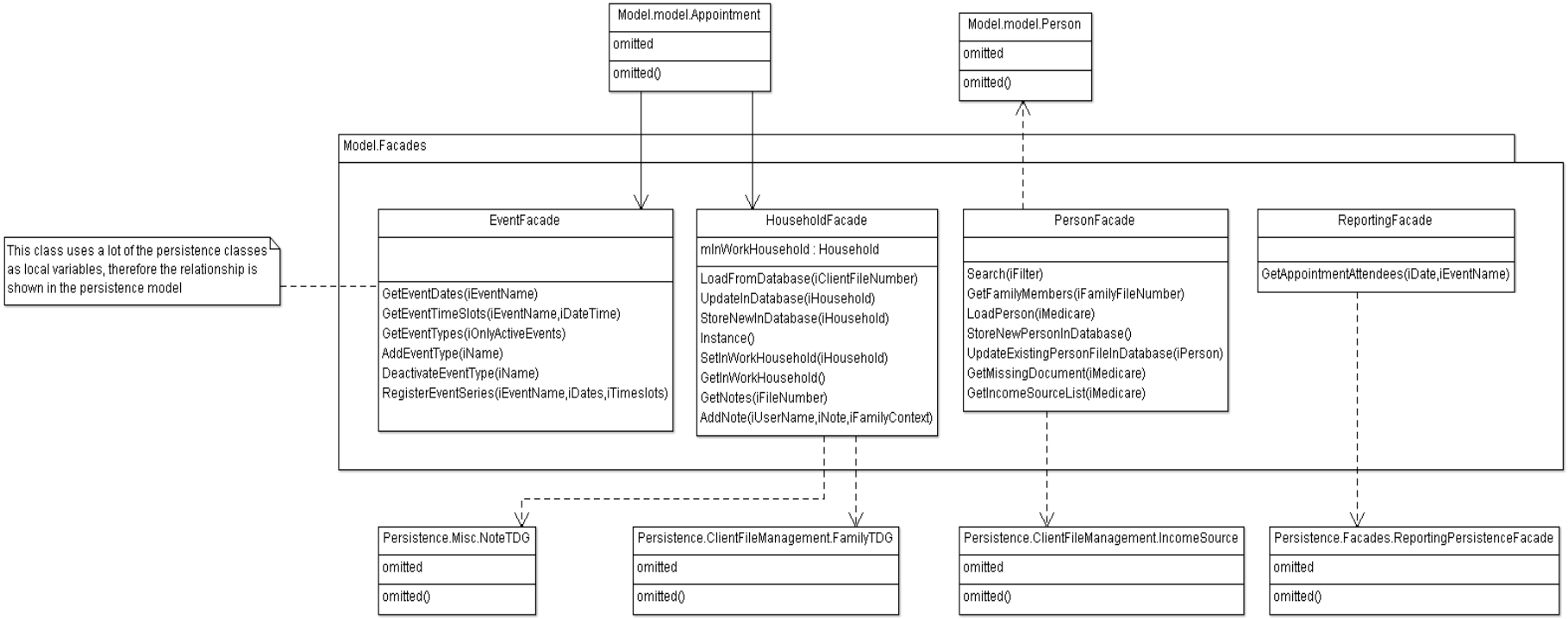


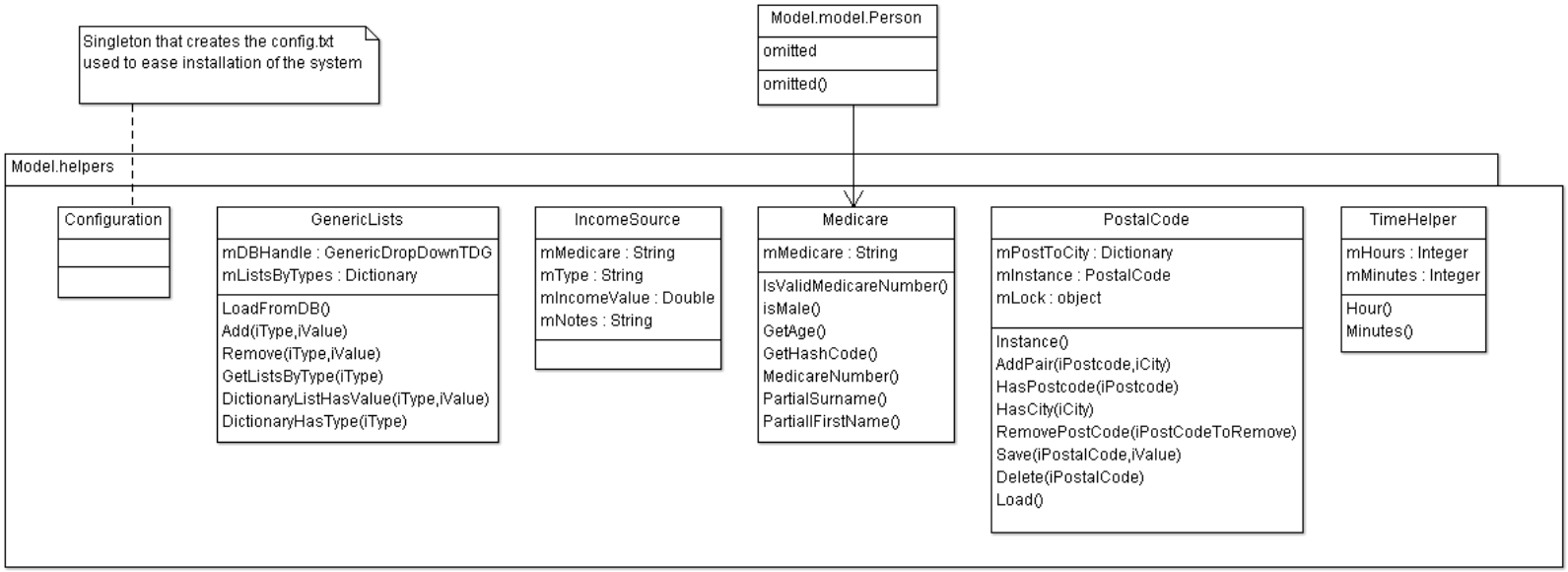




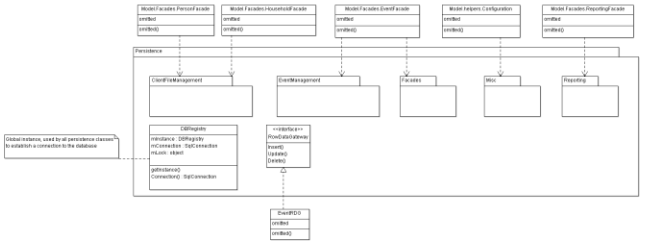

Model
<<interface>>
BookInterface
attachObserver(iObserver)
removeObserver(iObserver)
notify()
model
AppointmentBook
mAppointments : List<Appointment>
Register(iClientFileNumber,iTimeSlotId)
StoreItem(iAppointment)
UpdateItem(iAppointment)
Search(iAppointment)
SearchByAppointmentID(iAppointmentId)
FulfillAppointment(iAppointmentId)
SearchByFirstName(iFrstName)
SearchByLastName(iLastName)
SearchByFamilyId(iFamilyId)
SearchByDate(iDate)
EventBook
mEvent : Dictionary
AddEvent(iDate,iEvent)
GetAllEventTypes()
RemoveEvent(iEventId)
UpdateEvent()
Search()
Facades
Persistence
Persistence.EventRDG
omitted
omitted()
Persistence.EventManagement.AppointmentTDG
omitted
omitted()

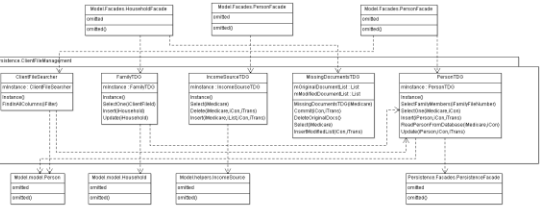

Persistence.ClientFileManagement.MissingDocumentsTDG
omitted
omitted()
Model.AppointmentBook
omitted
omitted()
Model.EventBook
omitted
omitted()
Model.model
Household
mFileNumber : Integer
InitializeMissingDocuments()
1
registers-for
0..*
Appointment
mAppointmentId : long
mFamilyId : long
mFulfilled : bool
mHouseholdFacade : HouseholdFacade
mEventFacade : EventFacade
Equals(Appointment)
AddHousehold(iHousehold)
RemoveHousehold(iHousehold)
is-contained-in
0..*
1
Event
mEventID : Integer
mDate : DateTime
mType : String
mRecurring : Boolean
AddDateSlot(slot)
RemoveDateSlot(slot)
AddDateList(list)
1
has-principal-representative
1
1
contains
1..*
1
is-alloted
1
Person
mMedicare : Medicare
mLastName : String
mFileNumber : Integer
TimeSlot
mTimeSlotID : Long
mEventName : String
GenerateTimeslots(iEventNAme,iStartTime)
0..*
1
Domain.helpers.Medicare
omitted
omitted()
Persistence.EventManagement.EventRDG
omitted
omitted()

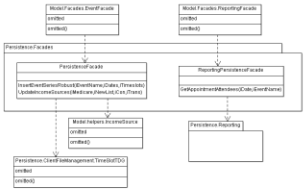

Model.model.Appointment
omitted
omitted()
Model.model.Person
omitted
omitted()
Model.Facades
EventFacade
GetEventDates(iEventName)
GetEventTimeSlots(iEventName,iDateTime)
GetEventTypes(iOnlyActiveEvents)
AddEventType(iName)
DeactivateEventType(iName)
RegisterEventSeries(iEventName,iDates,iTimeslots)
HouseholdFacade
mInWorkHousehold : Household
LoadFromDatabase(iClientFileNumber)
UpdateInDatabase(iHousehold)
StoreNewInDatabase(iHousehold)
Instance()
SetInWorkHousehold(iHousehold)
GetInWorkHousehold()
GetNotes(iFileNumber)
AddNote(iUserName,iNote,iFamilyContext)
PersonFacade
Search(iFilter)
GetFamilyMembers(iFamilyFileNumber)
LoadPerson(iMedicare)
StoreNewPersonInDatabase()
UpdateExistingPersonFileInDatabase(iPerson)
GetMissingDocument(iMedicare)
GetIncomeSourceList(iMedicare)
ReportingFacade
GetAppointmentAttendees(iDate,iEventName)
This class uses a lot of the persistence classes as local variables, therefore the relationship is shown in the persistence model
Persistence.Misc.NoteTDG
omitted
omitted()
Persistence.ClientFileManagement.FamilyTDG
omitted
omitted()
Persistence.ClientFileManagement.IncomeSource
omitted
omitted()
Persistence.Facades.ReportingPersistenceFacade
omitted
omitted()

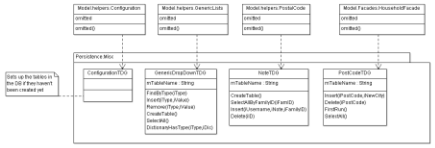

Singleton that creates the config.txt
used to ease installation of the system
Model.model.Person
omitted
omitted()
Model.helpers
Configuration
GenericLists
mDBHandle : GenericDropDownTDG
mListsByTypes : Dictionary
LoadFromDB()
Add(iType,iValue)
Remove(iType,iValue)
GetListsByType(iType)
DictionaryListHasValue(iType,iValue)
DictionaryHasType(iType)
IncomeSource
mMedicare : String
mType : String
mIncomeValue : Double
mNotes : String
Medicare
mMedicare : String
IsValidMedicareNumber()
isMale()
GetAge()
GetHashCode()
MedicareNumber()
PartialSurname()
PartialFirstName()
PostalCode
mPostToCity : Dictionary
mInstance : PostalCode
mLock : object
Instance()
AddPair(iPostcode,iCity)
HasPostcode(iPostcode)
HasCity(iCity)
RemovePostCode(iPostCodeToRemove)
Save(iPostalCode,iValue)
Delete(iPostalCode)
Load()
TimeHelper
mHours : Integer
mMinutes : Integer
Hour()
Minutes()

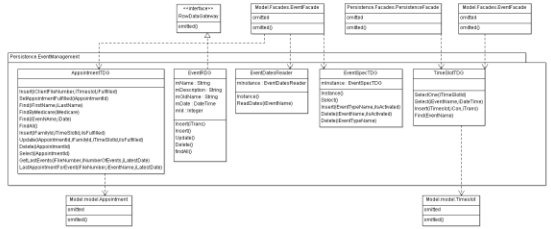
Persistence Layer

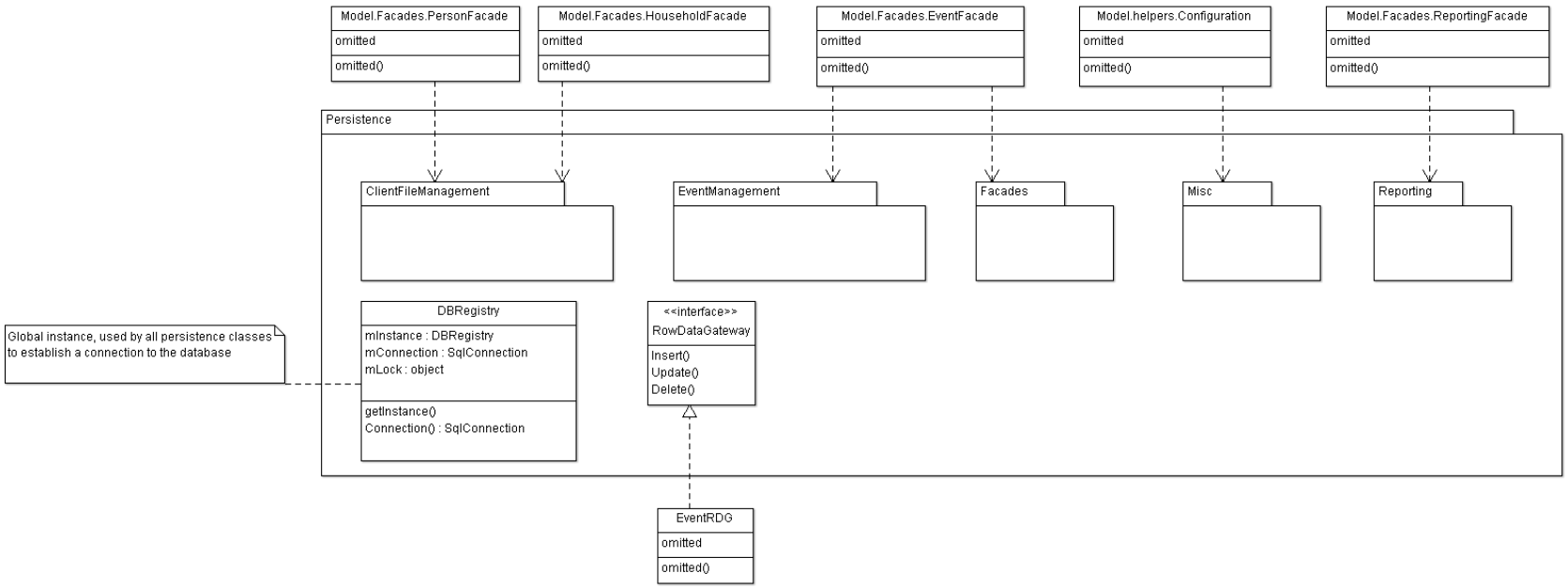


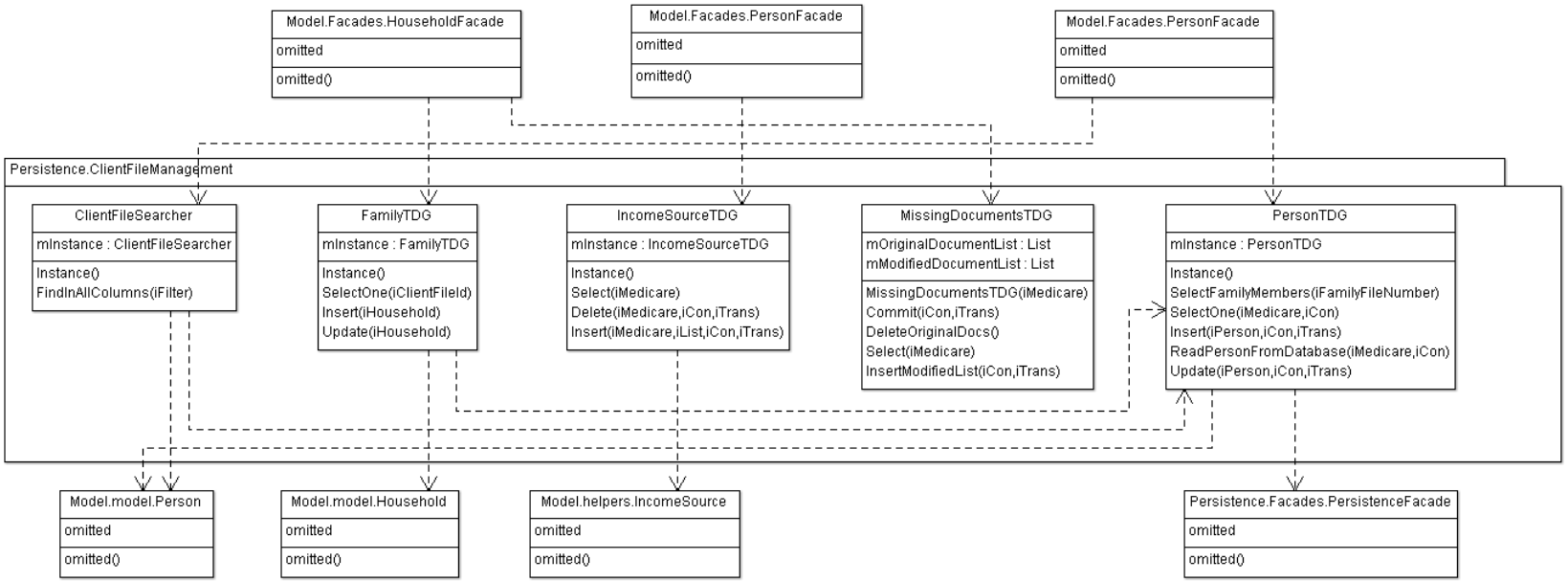


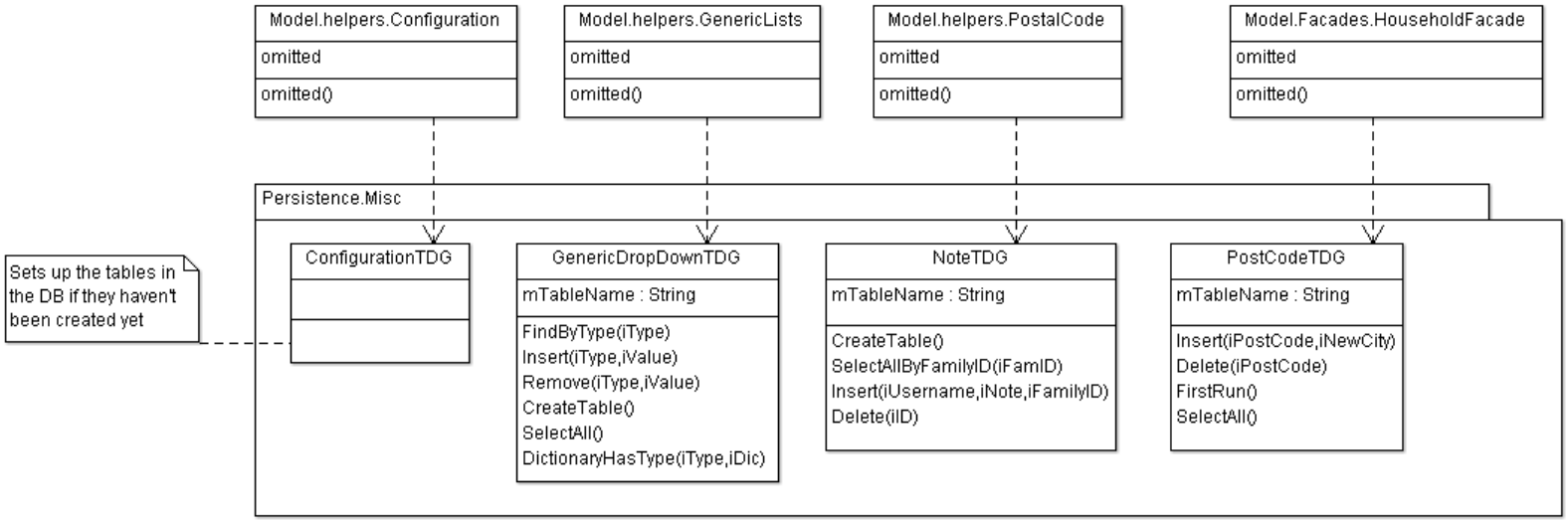


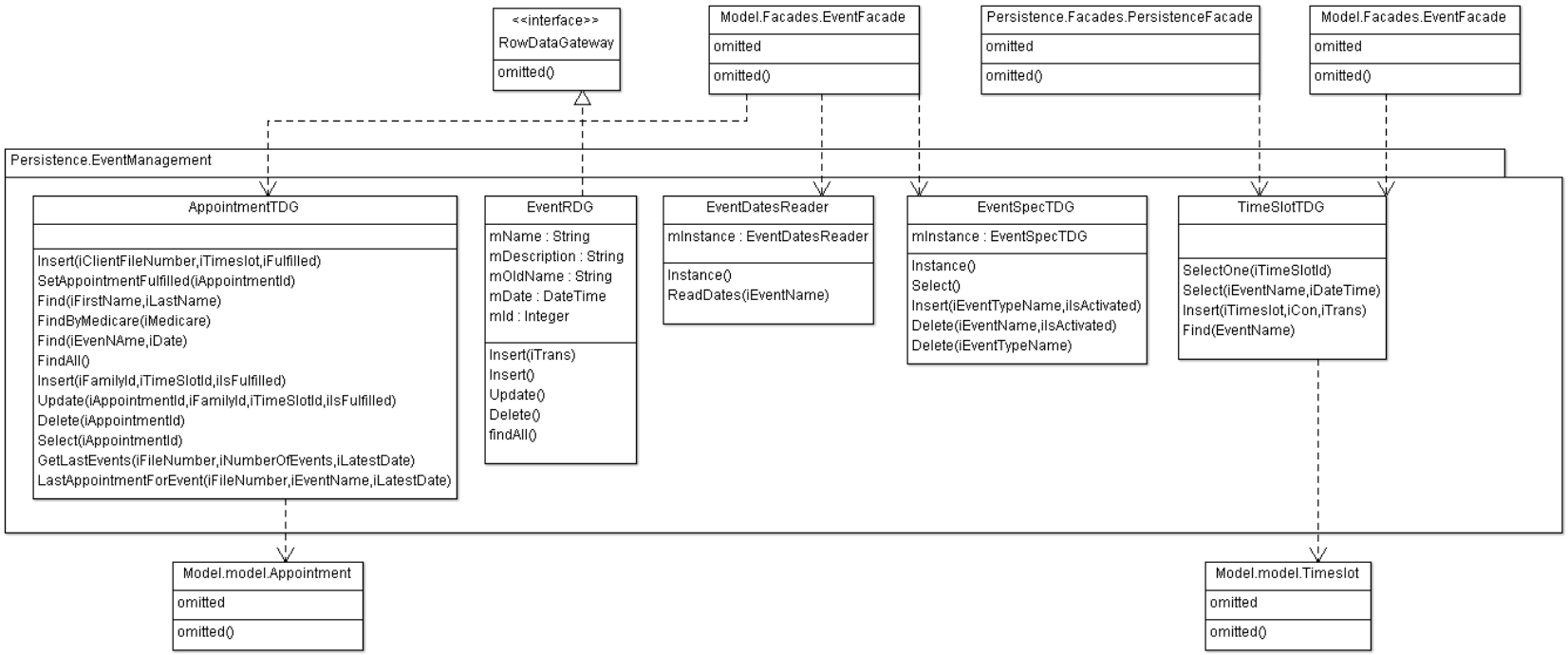


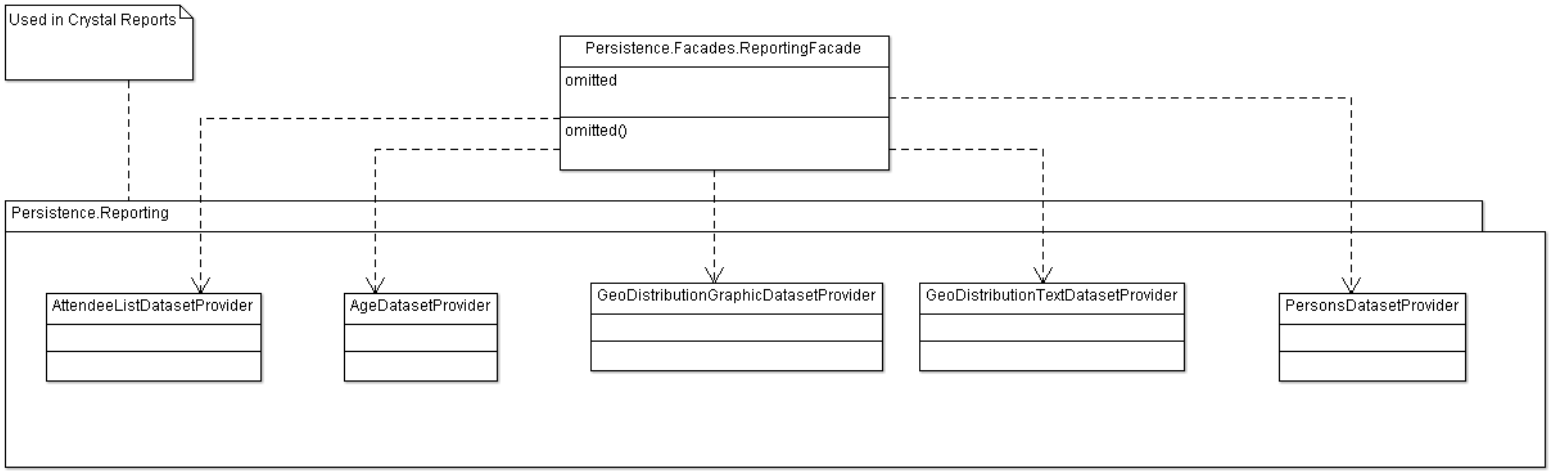



Model.Facades.PersonFacade
omitted
omitted()
Model.Facades.HouseholdFacade
omitted
omitted()
Model.Facades.EventFacade
omitted
omitted()
Model.helpers.Configuration
omitted
omitted()
Model.Facades.ReportingFacade
omitted
omitted()
Persistence
ClientFileManagement
EventManagement
Facades
Misc
Reporting
DBRegistry
mInstance : DBRegistry
mConnection : SqlConnection
mLock : object
getInstance()
Connection() : SqlConnection
<<interface>>
RowDataGateway
Insert()
Update()
Delete()
Global instance, used by all persistence classes to establish a connection to the database
EventRDG
omitted
omitted()


Model.Facades.HouseholdFacade
omitted
omitted()
Model.Facades.PersonFacade
omitted
omitted()
Model.Facades.PersonFacade
omitted
omitted()
Persistence.ClientFileManagement
ClientFileSearcher
mInstance : ClientFileSearcher
Instance()
FindInAllColumns(iFilter)
FamilyTDG
mInstance : FamilyTDG
Instance()
SelectOne(iClientFileId)
Insert(iHousehold)
Update(iHousehold)
IncomeSourceTDG
mInstance : IncomeSourceTDG
Instance()
Select(iMedicare)
Delete(iMedicare,iCon,iTrans)
Insert(iMedicare,iList,iCon,iTrans)
MissingDocumentsTDG
mOriginalDocumentList : List
mModifiedDocumentList : List
MissingDocumentsTDG(iMedicare)
Commit(iCon,iTrans)
DeleteOriginalDocs()
Select(iMedicare)
InsertModifiedList(iCon,iTrans)
PersonTDG
mInstance : PersonTDG
Instance()
SelectFamilyMembers(iFamilyFileNumber)
SelectOne(iMedicare,iCon)
Insert(iPerson,iCon,iTrans)
ReadPersonFromDatabase(iMedicare,iCon)
Update(iPerson,iCon,iTrans)
Model.model.Person
omitted
omitted()
Model.model.Household
omitted
omitted()
Model.helpers.IncomeSource
omitted
omitted()
Persistence.Facades.PersistenceFacade
omitted
omitted()


Model.helpers.Configuration
omitted
omitted()
Model.helpers.GenericLists
omitted
omitted()
Model.helpers.PostalCode
omitted
omitted()
Model.Facades.HouseholdFacade
omitted
omitted()
Persistence.Misc
Sets up the tables in the DB if they haven't been created yet
ConfigurationTDG
GenericDropDownTDG
mTableName : String
FindByType(iType)
Insert(iType,iValue)
Remove(iType,iValue)
CreateTable()
SelectAll()
DictionaryHasType(iType,iDic)
NoteTDG
mTableName : String
CreateTable()
SelectAllByFamilyID(iFamID)
Insert(iUsername,iNote,iFamilyID)
Delete(iID)
PostCodeTDG
mTableName : String
Insert(iPostCode,iNewCity)
Delete(iPostCode)
FirstRun()
SelectAll()


<<interface>>
RowDataGateway
omitted()
Model.Facades.EventFacade
omitted
omitted()
Persistence.Facades.PersistenceFacade
omitted
omitted()
Model.Facades.EventFacade
omitted
omitted()
Persistence.EventManagement
AppointmentTDG
Insert(iClientFileNumber,iTimeslot,iFulfilled)
SetAppointmentFulfilled(iAppointmentId)
Find(iFirstName,iLastName)
FindByMedicare(iMedicare)
Find(iEvenNAme,iDate)
FindAll()
Insert(iFamilyId,iTimeSlotId,iIsFulfilled)
Update(iAppointmentId,iFamilyId,iTimeSlotId,iIsFulfilled)
Delete(iAppointmentId)
Select(iAppointmentId)
GetLastEvents(iFileNumber,iNumberOfEvents,iLatestDate)
LastAppointmentForEvent(iFileNumber,iEventName,iLatestDate)
EventRDG
mName : String
mDescription : String
mOldName : String
mDate : DateTime
mId : Integer
Insert(iTrans)
Insert()
Update()
Delete()
findAll()
EventDatesReader
mInstance : EventDatesReader
Instance()
ReadDates(iEventName)
EventSpecTDG
mInstance : EventSpecTDG
Instance()
Select()
Insert(iEventTypeName,iIsActivated)
Delete(iEventName,iIsActivated)
Delete(iEventTypeName)
TimeSlotTDG
SelectOne(iTimeSlotId)
Select(iEventName,iDateTime)
Insert(iTimeslot,iCon,iTrans)
Find(EventName)
Model.model.Appointment
omitted
omitted()
Model.model.Timeslot
omitted
omitted()


Used in Crystal Reports
Persistence.Facades.ReportingFacade
omitted
omitted()
Persistence.Reporting
AttendeeListDatasetProvider
AgeDatasetProvider
GeoDistributionGraphicDatasetProvider
GeoDistributionTextDatasetProvider
PersonsDatasetProvider

5.3 Important Operation Contracts:

Event Management

Contract CO1	AddEventType
Operation:	AddEventType(name)
Cross References:	Manage Event Types Use Case
Pre-Conditions:	An event with the same name does not already exist in the Event Spec book
Post-Conditions:	1. An instance of EventSpec e was created. 2. e was associated with EventSpecBook

Contract CO2	UpdateEventType
Operation:	UpdateEventType(event, name)
Cross References:	Manage Event Types Use Case
Pre-Conditions:	The event already exists in the Event Spec book
Post-Conditions:	1. An instance of EventSpec e was updated.

Contract CO3	DeleteEventType
Operation:	DeleteEventType(name)
Cross References:	Manage Event Types Use Case
Pre-Conditions:	The event already exists in the Event Spec book. No events are associated to the event spec with name "name".
Post-Conditions:	1. An instance of EventSpec e was deleted. 2. The association between e and EventSpecBook was removed.

Contract CO4	AddEvent
Operation:	AddEvent(event)
Cross References:	Manage Events Use Case
Pre-Conditions:	None
Post-Conditions:	1. An instance of Event e was created. 2. e was associated with EventBook. 3. e was associated with Timeslot

Contract CO5	RemoveEvent
Operation:	RemoveEvent(event)
Cross References:	Manage Events Use Case
Pre-Conditions:	Event must exist beforehand
Post-Conditions:	1. An instance of Event e was deleted. 2. The association between e and EventBook was removed. 3. The association between e and Timeslot was removed.

Contract CO6	RemoveEventById
Operation:	RemoveEventById(id)

Cross References:	Manage Events Use Case
Pre-Conditions:	Event must exist beforehand
Post-Conditions:	1. An instance of Event e is deleted. 2. The association between e and Eventbook was removed. 3. The association between e and Timeslot was removed.

Client File Management

Contract CO7	CreateFamilyFile
Operation:	CreateFamilyFile()
Cross References:	Manage Family Files Use Case
Pre-Conditions:	None
Post-Conditions:	1. An instance of Household h was created. 2. An association between h and FamilyBook is created.

Contract CO8	UpdateFamilyFile
Operation:	UpdateFamilyFile(household)
Cross References:	Manage Family Files Use Case
Pre-Conditions:	The household object should exist already.
Post-Conditions:	1. Some or all attributes of household h were changed.

Contract CO9	AddDependant
Operation:	AddDependant(household, person)
Cross References:	Manage Family Files Use Case
Pre-Conditions:	None
Post-Conditions:	1. Attribute dependantList of Household object h was modified to contain the new Person p. 2. An association between h and p was created.

Contract C10	UpdateDependant
Operation:	UpdateDependant(household, person)
Cross References:	Manage Family Files Use Case
Pre-Conditions:	The household object should exist with the person already as a dependant.
Post-Conditions:	1. Some or all attributes of Person p were changed. 2. The attribute dependantList of Household h was changed.

Appointment Creation

Contract C11	CreateAppointment
Operation:	CreateAppointment(familyFileNumber, timeSlot)
Cross References:	Create Appointment Use Case
Pre-Conditions:	An event has already been created. A Family has already been created.

Post-Conditions:	1. An instance of Appointment a is created. 2. An association between a and Event is created. 3. An association between a and Family is created.

Contract C12	UpdateAppointment
Operation:	UpdateAppointment(appointmentID, appointment)
Cross References:	Update Appointment Use Case
Pre-Conditions:	The appointment with appointmentID already exists.
Post-Conditions:	1. The instance a of Appointment has some or all of it's attributes changed.

Contract C13	CancelAppointment
Operation:	CancelAppointment(appointmentID)
Cross References:	Cancel Appointment Use Case
Pre-Conditions:	The appointment with appointmentID already exists.
Post-Conditions:	1. An instance of Appointment a with appointmentID is deleted. 2. An association between a and Event is deleted. 3. An association between a and Family is deleted.

Appointment Fulfillment

Contract C14	GetAttendeesByName
Operation:	GetAttendeesByName(name)
Cross References:	Get Attendees Use Case
Pre-Conditions:	None
Post-Conditions:	None

Contract C15	GetAttendeesByEvent
Operation:	GetAttendeesByEvent(eventName)
Cross References:	Get Attendees Use Case
Pre-Conditions:	None
Post-Conditions:	None

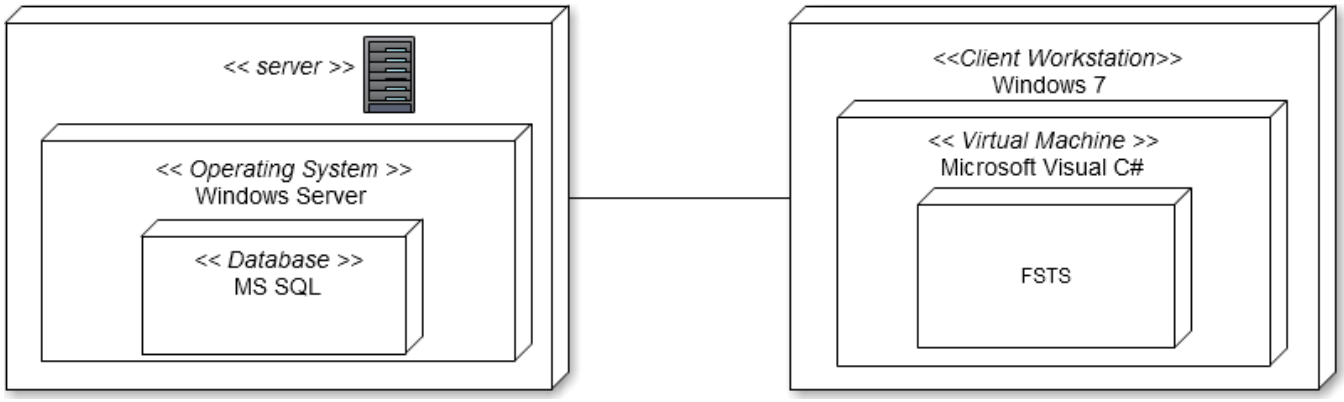
Contract C16	GetAttendeesByMedicare
Operation:	GetAttendeesByMedicare(medicareNum)
Cross References:	Get Attendees Use Case
Pre-Conditions:	None
Post-Conditions:	None

Contract CO17	FulfillAppointment
Operation:	FulfillAppointment(appointmentID)
Cross References:	Fulfill Appointment Use Case
Pre-Conditions:	None
Post-Conditions:	1. Appointment a with appointmentID has it's attribute *Fulfilled* set to true.

Contract CO18	PrintAttendees

Operation:	PrintAttendees(listOfAppointments)
Cross References:	Print Attendees Use Case
Pre-Conditions:	None
Post-Conditions:	None

6. Deployment View



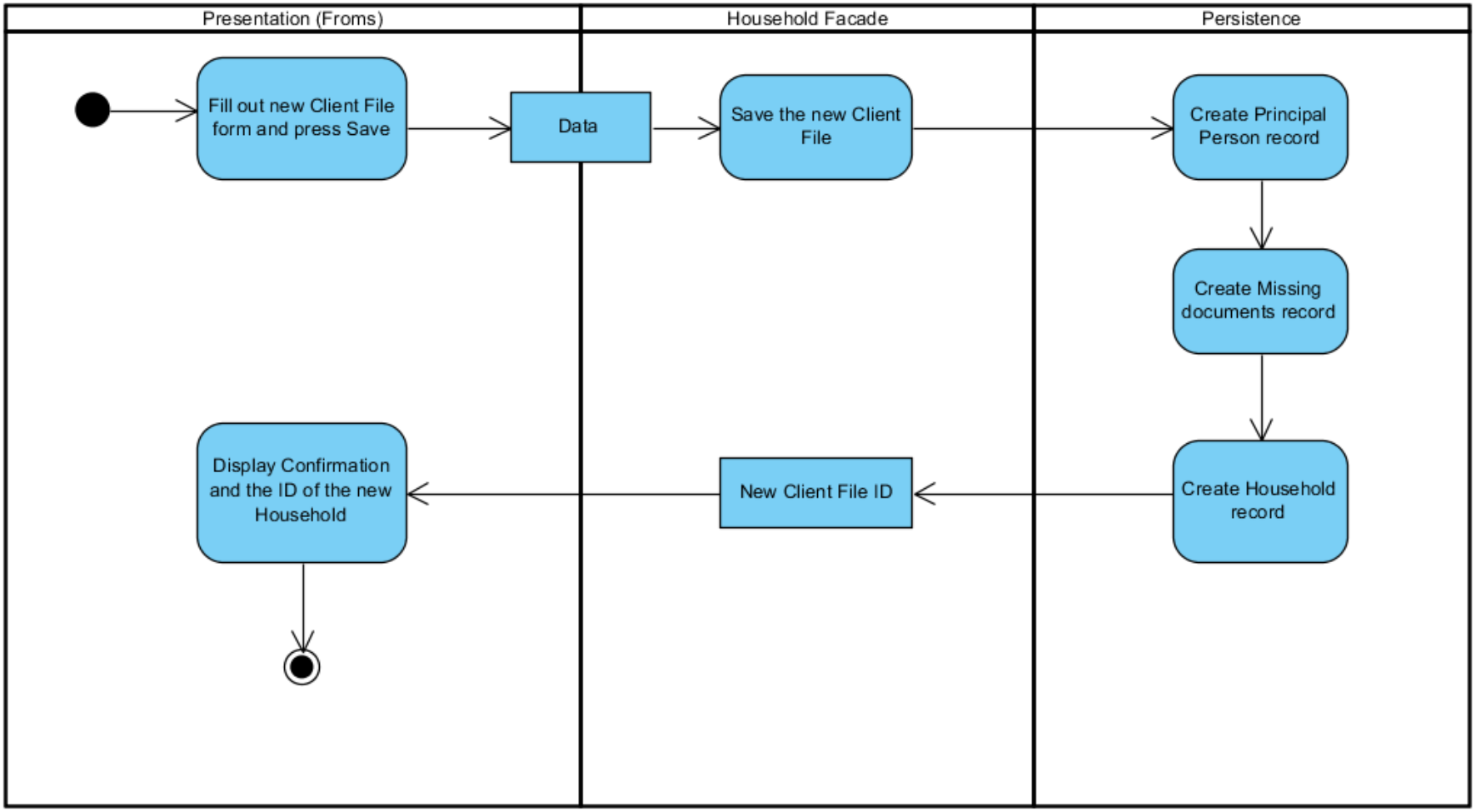
This diagram depicts the physical view of the system in its current state. The diagram shows the interaction between the server and the client. Within the physical hardware of the server itself, lies the Windows Server Operating System, and installed on that server is Microsoft SQL Server software. The client has within it the Microsoft Visual C# Virtual Machine, which interprets and runs C# bytecode. Within this Virtual Machine runs our system, the Family Services Tracking System (FSTS). These two components (client & server) are directly connected between the FSTS system in the client, and MS SQL server on the server.

7. Process View

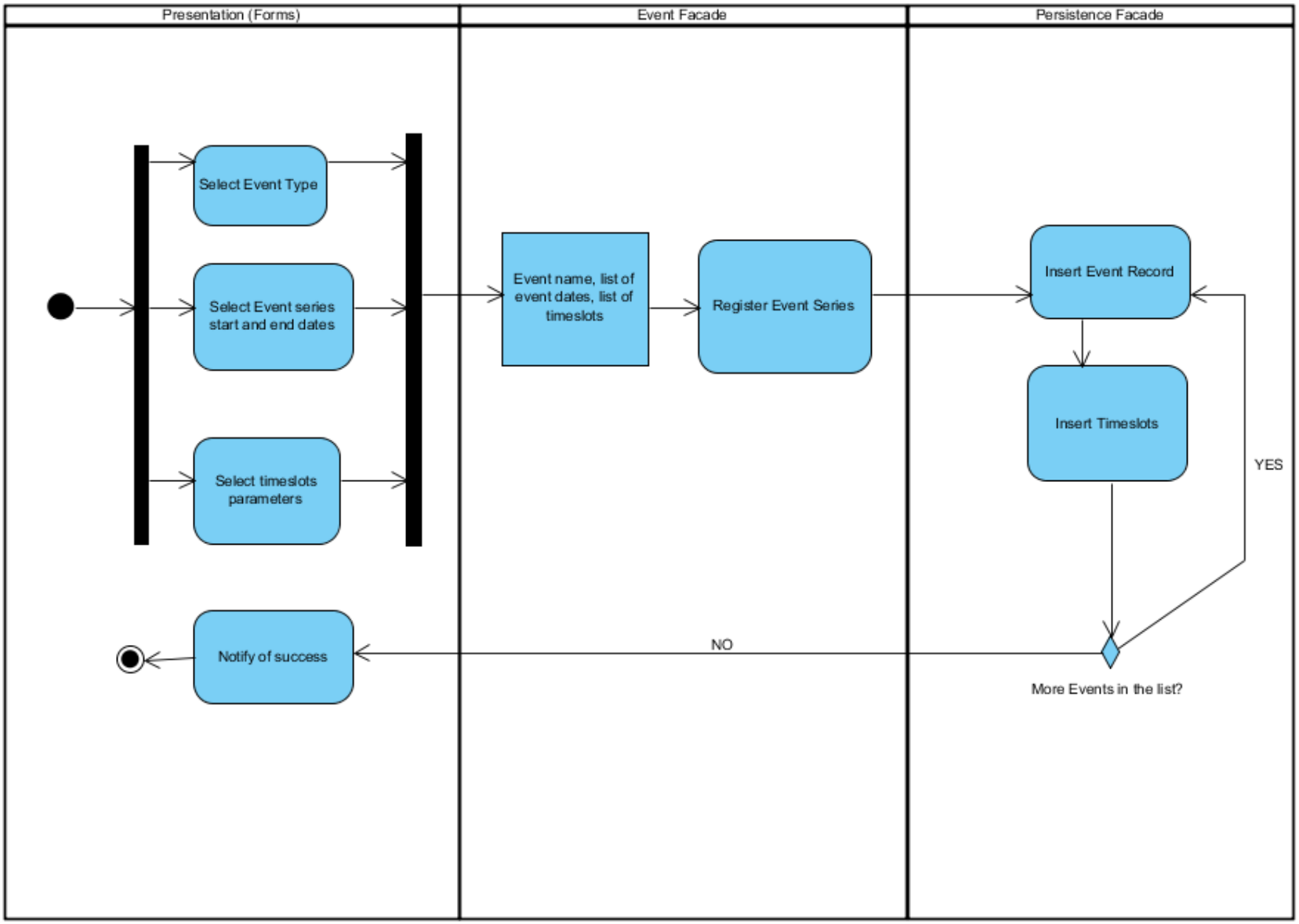
7.1 Overview

There are four main activities in the FSTS: Create Client Files, Create Events, Make Appointment and Open Client File. These four activities enclose all other actions in the system and are portrayed in the next sections.

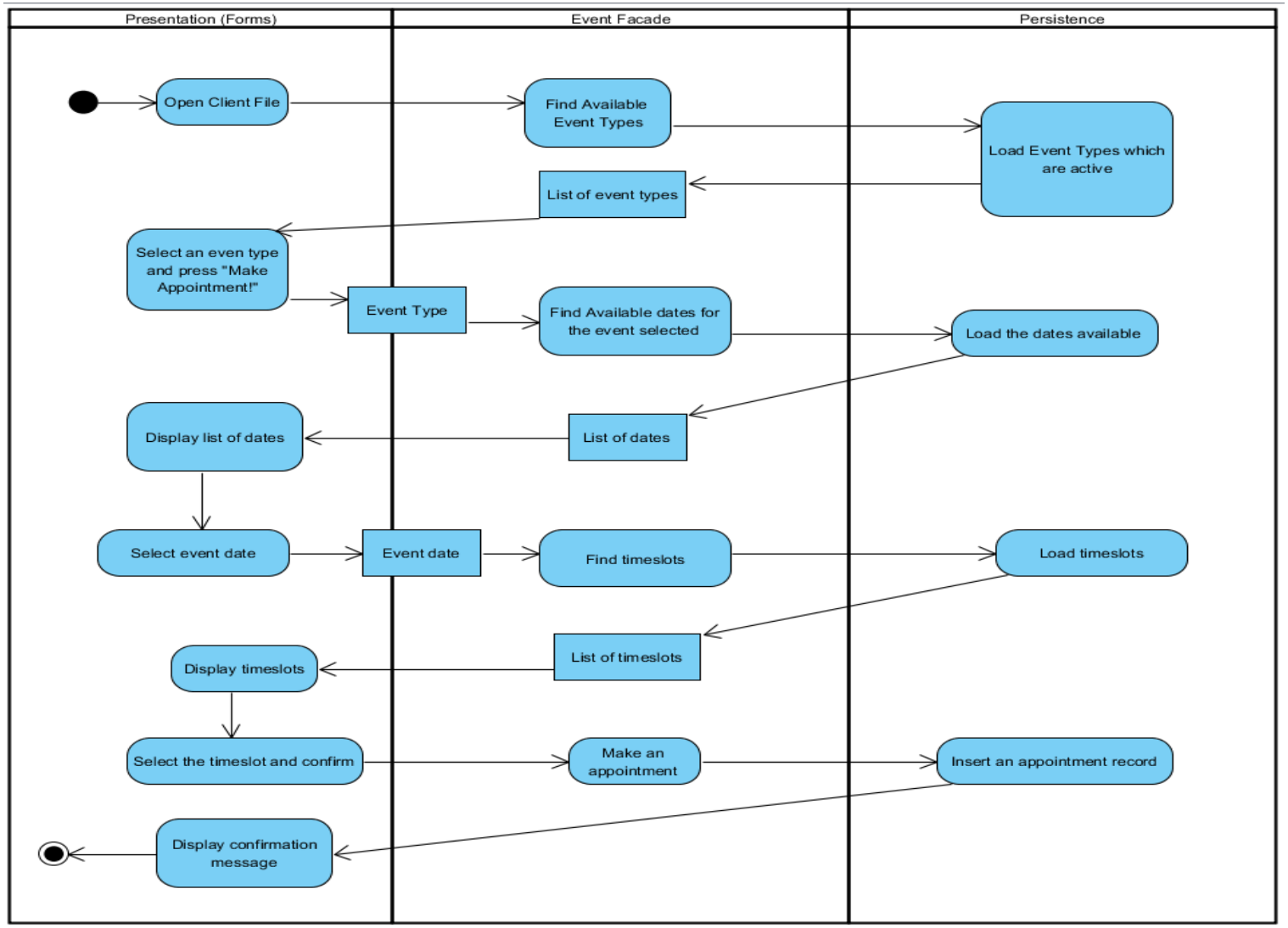
7.2 Create Client File



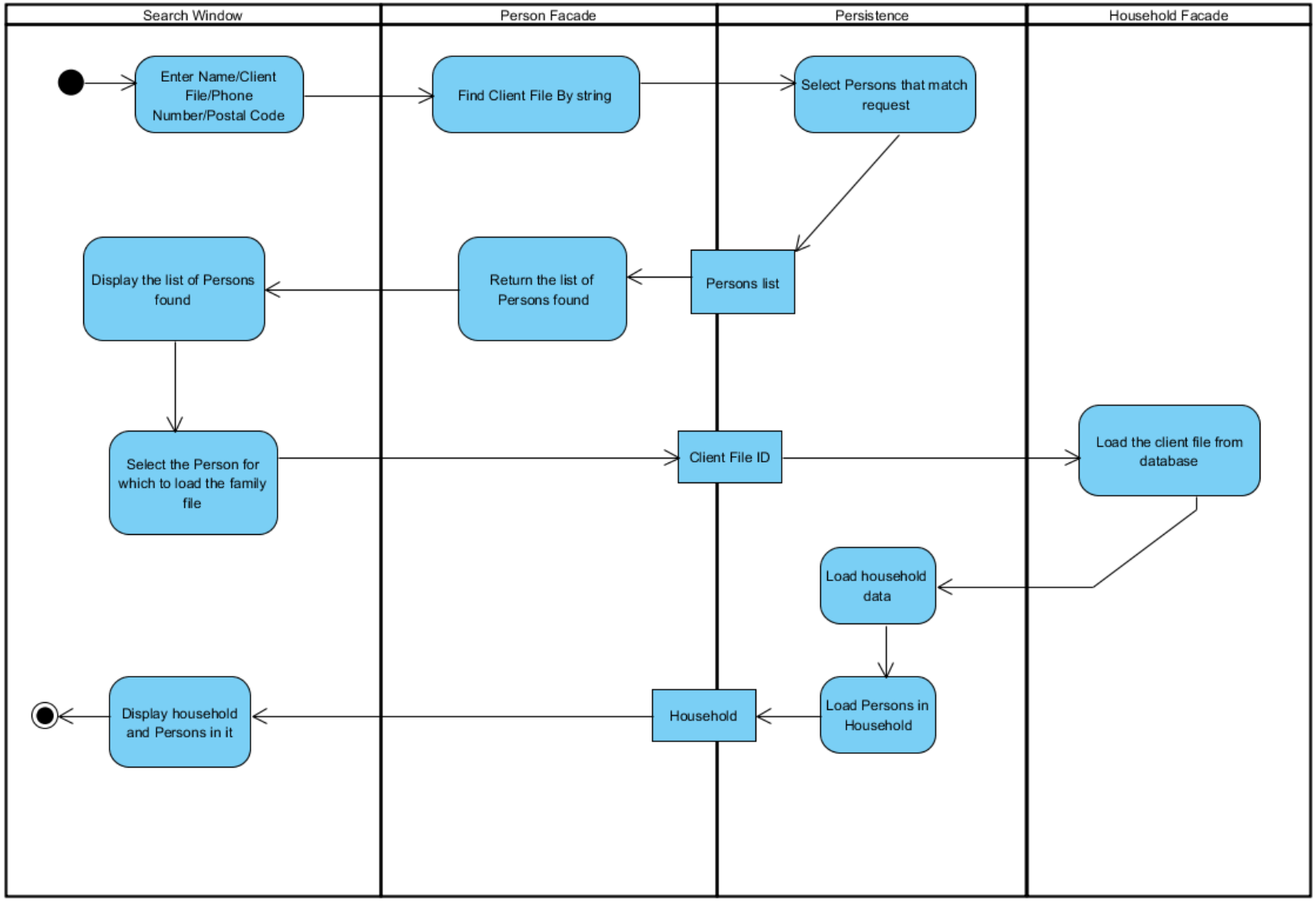
7.3 Create Event



7.4 Make Appointment



7.5 Open Client File

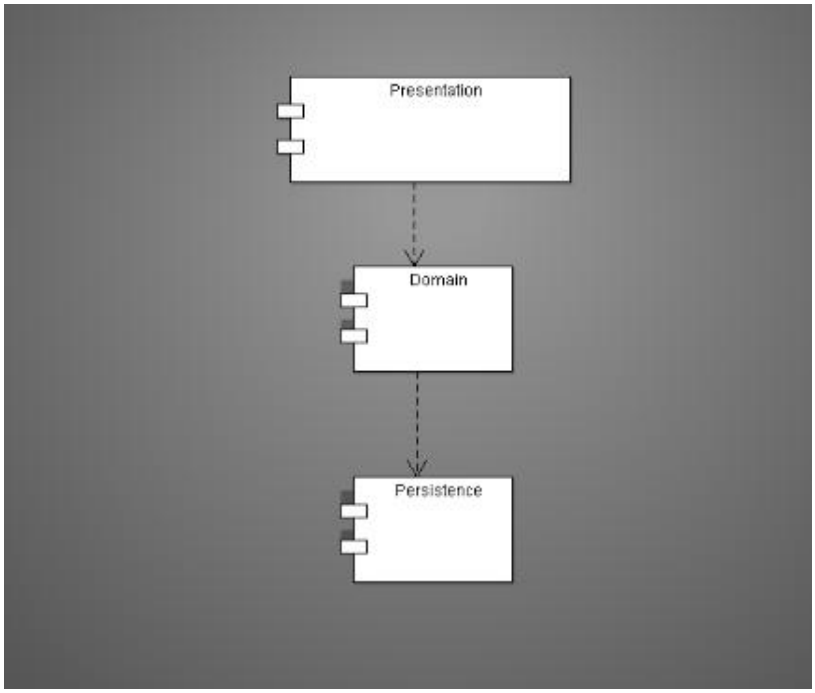


8. Implementation View

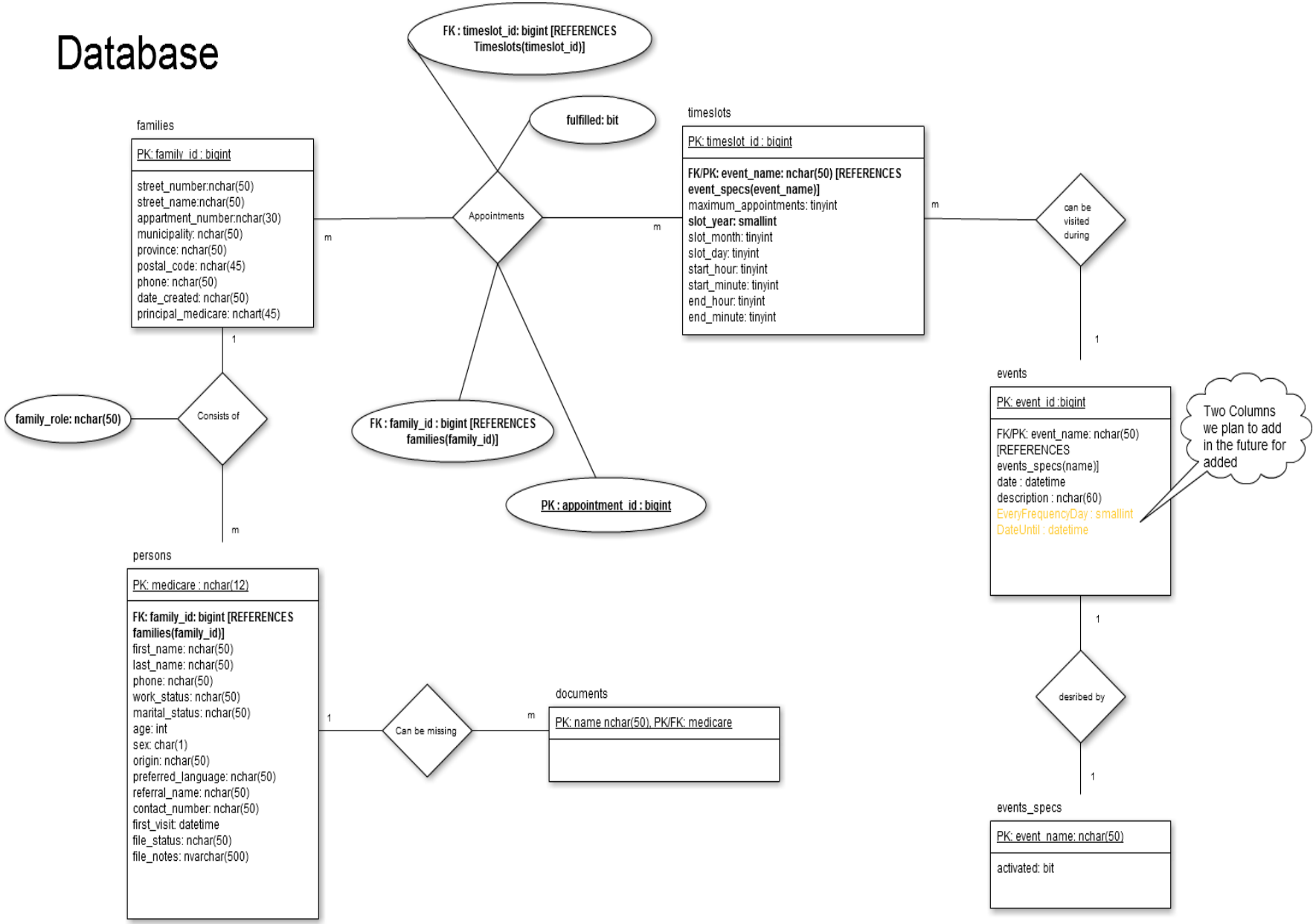
8.1 Overview

The Architecture style follows the 3-layered model (opaque). Each layers can communicate with the layer directly under it. The three layers are Presentation, Domain and Persistence. Discussion about this architecture is found in section 10.

8.2 Layers



9. Data View


Database
FK : timeslot_id: bigint [REFERENCES Timeslots(timeslot_id)]
fulfilled: bit
families
PK: family_id : bigint
street_number:nchar(50)
street_name:nchar(50)
appartment_number:nchar(30)
municipality: nchar(50)
province: nchar(50)
postal_code: nchar(45)
phone: nchar(50)
date_created: nchar(50)
principal_medicare: nchart(45)
Appointments
timeslots
PK: timeslot_id : bigint
FK/PK: event_name: nchar(50) [REFERENCES event_specs(event_name)]
maximum_appointments: tinyint
slot_year: smallint
slot_month: tinyint
slot_day: tinyint
start_hour: tinyint
start_minute: tinyint
end_hour: tinyint
end_minute: tinyint
can be visited during
family_role: nchar(50)
Consists of
FK : family_id : bigint [REFERENCES families(family_id)]
PK : appointment_id : bigint
events
PK: event_id :bigint
FK/PK: event_name: nchar(50) [REFERENCES events_specs(name)]
date : datetime
description : nchar(60)
EveryFrequencyDay : smallint
DateUntil : datetime
Two Columns we plan to add in the future for added
persons
PK: medicare : nchar(12)
FK: family_id: bigint [REFERENCES families(family_id)]
first_name: nchar(50)
last_name: nchar(50)
phone: nchar(50)
work_status: nchar(50)
marital_status: nchar(50)
age: int
sex: char(1)
origin: nchar(50)
preferred_language: nchar(50)
referral_name: nchar(50)
contact_number: nchar(50)
first_visit: datetime
file_status: nchar(50)
file_notes: nvarchar(500)
Can be missing
documents
PK: name nchar(50), PK/FK: medicare
desribed by
events_specs
PK: event_name: nchar(50)
activated: bit
1
m

The ER Diagram represents the entities that represent vital information components of the system and the Relationships illustrate the connection between these entities. In the Current System, Persons, Families, Timeslots, documents, Events and Event_Specs represent entities. There exists relationships between these entities or tables in the Database, but many to one relationships are represented by only foreign keys in one of the tables, because a separate table which represents all the relationships would be redundant. The only case where we implement a table for maintaining relationships is for storing Appointments, because many to many relationships cannot be maintained with only one foreign key in one of the tables. The relationship contains properties which are stored in the relationship table.

10. Architecture Justification

In our system, we chose to use a three-layered architecture, mostly of the opaque variety (opaque meaning that a given layer can only access the layer directly below itself). The three layers in our system are the classic Presentation, Domain Logic, and Data Source. We chose to use this style for several reasons, namely: in this architectural style, code is highly maintainable and much easier to comprehend by developers, due to the fact that it employs a high level of abstraction. In addition to this, an opaque three-layered architecture provides the ability to easily swap out a layer, and replace it with a new one, with minimal effort required. For a small part of the codebase however, transparent layering was used rather than opaque. The reason for this was not a design-based decision, but was due rather to the time-constrained nature of this project, and other priorities within the system: there was simply not enough time and resources to fully implement the opaque layering between the persistence and the domain layers. Rather, the persistence layer directly accesses the model. Another important aspect of the three-layered architecture that influenced our decision to use it is a tried and true principle of computer science: divide-and-conquer. That is to say, the three-layered architecture takes the complex nature of the system, and splits it into manageable, measurable, somewhat independent units that are much easier to deal with than the system as a whole. In addition, we have chosen an opaque layering rather than a transparent layering due to the fact that easy maintenance is higher on our priority list than extreme performance.

The high level of abstraction that a three-layered architecture provides has many benefits. But first, we must ask, what exactly is meant by “high level of abstraction”? What is meant by this is that rather than doing complex tasks in our system on a low level (e.g. accessing the persistent data store) directly, we can use abstraction from one layer to another to make that operation much simpler and easy to perform. To elaborate on the above example of accessing the persistent data store: in our Presentation layer, which is where the user interacts directly with the system, suppose the user performs an action that displays all persisted objects of a given type. This obviously requires access to the database. But rather than have the programmer who is doing the code for this layer (Presentation) needing explicit knowledge of the database schema being used (Data Source), as well as the necessary knowledge of SQL to get that data, all that is required is a call to a method from the layer beneath it (Domain Logic), passing the necessary information in that method call, and have the Domain Logic layer, in a black-box fashion, deal with that information and eventually return the information requested. This may seem quite simple, but is actually extraordinarily powerful and enables development to be done much more rapidly, by aiding the developers in not needing specific knowledge of every aspect of the

system, but rather requesting a service from another layer, and having all of the work get done in the layer below. Another benefit other than rapid development is code comprehension. With a higher level of abstraction, the comprehension of the code becomes extremely simple: all calls within the program are simple abstractions from other layers, and understanding the code is only a matter of understanding what services are provided by these abstracted methods.

The ability to easily swap out a layer for a new one is vitally important for this project, since it is being done in a rapid development environment, and so certain layers, especially the Presentation layer, will undoubtedly be replaced throughout the development. For example, because of the rapid development process, the initial graphical user interface was created mainly as a proof-of-concept, to serve as a way of demonstrating functionality to the client. This functionality however, was very much "isolated functionality", meaning that it showed individual features, but not how the system would work as a whole. This initial interface was quickly replaced by a more full-fledged GUI that would demonstrate not only the individual functionality of specific features, but also work-flow behaviours and how the user will actually be interacting with the system as a whole. This ability to code the behaviour in the Domain Logic layer, and have a simple dummy Presentation layer allowed us to concentrate on implementing highly important functionality, without needed to sacrifice time and effort on creating the GUI at the same time. We were easily able to create a "dummy" proof-of-concept GUI to show to the stakeholders, and then when the basic functionality was completed we were able to easily rewrite the presentation layer, using all of our previous work in the other layers by simply using the abstraction discussed above.

The concept of divide-and-conquer is widespread in the fields of Computer Science & Software Engineering. It is very often impossible to solve problems by simply tackling it head on, but rather it must be split into more manageable components that can, when working together, combine to solve the problem at hand. By dividing our system into the three layers of Presentation, Domain Logic, and Data Source, we have created three components that can be logically divided and managed independently of the others. This allows us to likewise focus our efforts on one problem at a time, and build from the bottom up. It would be unreasonable to try to create all of the functionality of the system at once, but since we focus on one layer at a time, we can easily build the entire system piece by piece, and through the power of object oriented programming, we can put the components together to easily create a fully functioning system. The concept of divide-and-conquer is not only important for the initial development of our three-layered architecture, but also the maintenance of it. Because we have divided the system into components, the maintenance becomes extremely easy to do: when certain functionality is discovered to be malfunctioning, it is easy to pinpoint which area in the code the problem is occurring in due to the logical division of functionality between the layers.

On the topic of maintainability, we decided that since this was not to be a real-time critical system where extreme performance is vital to system operation, that an opaque layering was more appropriate. As opaque layering only allows layers to access the layer directly below, rather than any layer somewhere underneath itself, the maintainability of the system rises significantly. The coupling between layers is massively reduced, and swapping layers becomes much more feasible. The theoretical performance benefits of a transparently layered system were not estimated to be nearly enough to sacrifice the enormous maintenance benefits of an opaquely

layered system. However, due to time constraints, the system is not fully opaque, and in certain areas uses transparent layering. This was not a design goal for the architecture, but rather a consequence of the conditions in which the system was built, namely, lack of time. Specifically, the transparent areas of the system are the domain and persistence layers, that directly interact with each other without having a layer in the middle to handle their inter-behaviour.

To recap, we chose an opaquely layered, three-layer architecture for several reasons. We really value the level of abstraction that this architecture provides, since it allows us to more rapidly and more readable, comprehensible code. Due to lack of time and resources however, not the entire system was made to be fully opaque, and there are still some areas that remain transparent. In addition, the extremely useful ability to easily swap layers in and out of the system cannot be ignored. Because of this architecture property, we were able to concentrate our efforts on specific functionality and not waste resources on the others, but rather creating them as dummy layers, so as to have something to show the stakeholders, and then easily replace those dummy layers when appropriate. Divide-and-conquer was another aspect of this architecture that we really felt was important, as it allows us to split the problem into more manageable components to put together to create the system as a whole. Finally we chose an opaque layering rather than a transparent one, since we felt the benefits opaqueness gave us in terms of code maintenance highly outweighed the supposed performance benefits transparent layering might offer, especially considering that in our system, performance is not a primary goal.

ANNEX A: Glossary

Appointment (Syn. Time Slot)
A time interval during which a *client* is scheduled to come to an *event.*

Appointment Attendee (Syn. Attendee)
A *client* who is registered for an *event* at a specific *time slot.*

Appointment Fulfilment
An act of honoring a *time slot* reservation by a *client* for a specific *event*.

Client
A person who uses, used, or has an intention to use services provided by *WHM*

Client File (Syn. File)
A collection of data fields pertinent to a specific *client.*

Employee
A person whose duties permanently or temporarily involve working for *WHM* due to employment, volunteering work, or contractual obligations.

Event
An activity of a specific type lasting no more than one day held by *WHM* intended to serve *clients'* needs.

Family
A group of *clients* and other persons who permanently or temporarily reside at the same civic address.

FSTS (Syn. System, Application, Program, Software)
Family Services Tracking System. The system to be developed.

Household
A set of individuals composed of a primary individual and other individuals linked by a relationship.

Individual
A WHM customer.

iPad (Syn. Mobile Device)
Is a tablet device. References the iPad2 whenever mentioned in the document, unless stated otherwise.

Old FSTS (Syn. Existing Software)
The legacy counterpart of *FSTS* which is currently being used by *WHM* and is to be completely replaced by *FSTS*. A baseline of performance, usability, and functionality of the new system.

.

Operator (Syn. User)
A *WHM employee* whose duties involve interacting with any part of *FSTS* for the purpose of storing, retrieving, or modifying information pertinent to *clients* and their interaction with *WHM.*

Statistical Analyst

A *WHM employee* whose duties involve obtaining statistical information and generating reports on *WHM* business, the *clients'* information, other organizations *WHM* works with or any other information which is derived from the aforementioned data.

Report

A document that is automatically generated by the system which holds valuable information for statistical analysts. Reports are generated and exported in different formats and file types.

WHM (Syn. The Organization, MBA)

Welcome Home Mission. The non-profit organization, the primary user and product owner of *FSTS*.

.

ANNEX B: Educational Comparison Chart