Subsea7 People Finder

Projectcource Systemdevelopment 2012

Jonas Brunsgaard - 141185

Martin Bjerregaard Jepsen
Rasmus Wriedt Larsen
Rune Juhl Jacobsen

May 31st 2012

Videolink: http://www.youtube.com/watch?v=aTGoGxTlfmY

Department of Computer Science
University of Copenhagen

List of Corrections

Contents

Part I

About the IT project

1 Introduction, purpose and frame

We have been developing an iPad application for Subsea7, a subsea engineering and construction company from Norway. They have around 600 employees at their office in Stavanger, and a new building of the same size is planned for December 2012. Due to the large amount of employees, it can be problematic for an employee to locate those he wants to talk to. Therefore we're creating a "People Finder"-application, to be placed across the buildings besides the main stairs, that will be able to display the location and work-status of employees.

The project is done in C# with MonoTouch, which cross compiles to iOS. Subsea7 delivers a Web Service for communicating data (using XML) to the iPads on a regularly basis. Each iPad is uniquely identified by its MAC address, so data and configuration can be set uniquely for a given iPad.

FACTOR (as mentioned in [MMMNS00], please see section "References") is used in this project to formulate a system definition.

Function
Enable employees to quickly and effortlessly locate another employee in the company, either by name, project or department.

Application domain
Company employee finder.

Conditions
Usable without requiring any prior knowledge (highly intuitive). No discernible delays when using the application, no matter the condition.

Technology
Target platform is an Apple iPad 2, while enabling easy porting of the application to other, highly similar, target (e.g. iPhone). Specifically, the application should be written in C#. Backend is highly irrelevant, with the only restriction that the data should be delivered in XML format validating against our XML Schema Definition.

Objects
Employees, departments, projects.

Responsibility
Automatic updates of data.

The project is part of the course Projektkursus: Systemudvikling[1] which forms the frame around the project. The goal with the course is to give the students a methodical well-founded and practical applicable introduction to IT-systemdevelopment.

2 Specification requirements

2.1 Functional requirements

Graphical User Interface

- When the user first opens the application he is met by an "Employees" category view. At the bottom of the screen a bar with three buttons with icon and description ("Employees", "Departments" and "Projects") is shown. From here the user can tab the buttons to change the content of the list above.
- The "Employees" category is a list of all the employees. The employees are sorted by either first or last name, selected by a button at the top of the interface. Next to every employee name there is a small picture of them to aid identification. If there are ever less than 30 employees in a list, grouping on the first letter will not be used, to give the user a less cluttered view. There should be a coloured overlay to show if the user is available, maybe available or not available (green/yellow/red).
- The "Departments" and "Projects" categories shows a list of all departments/projects, sorted by name. A user can tap on any one of these, and will then be shown an "Employees" category listing only the employees tied to that particular employee group.

[1] `http://sis.ku.dk/kurser/viskursus.aspx?knr=131371`

- When displaying an employees list with a department or project, the main area where involved people are sitting should be displayed.
- When an employee has been selected, the S7Finder app shows the user a profile view of the employee. This view contains a large picture of the employee, their name, phone number, job title, department, a listing of projects they're involved in and desk location [2] , and availability status [3].
- When the desk location is tapped it should open up in a larger view.
- When the S7Finder app is not in use, it displays a pause screen containing the floor number and the text "People Finder".
- If the application gets in an error state, it should display an "out of order" screen.
- The screen should turn black at night at specified hours.

Application data and configuration

- S7Finder has a config file that is requested from a central server at intervals set by the config file provided from the server, or by default, every night. When the iPad connects to the web service to acquire the config file, it provides the iPads MAC address as a UUID (Universally Unique ID). The web service returns an XML file with the following data:
  - Screen turning black hours
  - Interval for retrieving employee data
  - Interval for retrieving config files
  - Interval for checking for new images
  - iPad Location floor
  - Base URL for fetching all data (including config and images)
  - Seconds before going back to Standby Screen
  - The default sort method for the employee list (First Name or Last Name)
  - The sequence of letters displayed on the right of the employee list
- To keep the data up to date, the S7Finder app requests updates (in XML) from a web service every 5-10 minutes. The interval can be changed through the config file. It will also request an update when a user activates the app, to ensure that the data displayed is up-to-date.

2.2 Non-functional requirements

Quality requirements

1. Any user must be able to locate a specific employee within 1 minute, without prior training or experience with the system. [Usability requirement]
2. The information displayed by the system must be presented in a clear and concise way. [Usability requirement]
3. There must be no noticeable latency when using the system. All the different views of the application must be shown within 0.5 seconds. [Performance requirement]
4. The user interface elements must be of a size, so that unexperienced tablet users still fell comfortable with the app. [Usability requirement]

We are aware that the quality requirements should be measurable in order to assert that the final product conforms to these, but due to the scope of the project we acknowledge that we are unable to do research sufficient to determine reasonable values for list items 2 and 4. Further, should we be able to determine reasonable values for these requirements, we would still be unable to test these to the amount required for statistical correctness.

[2] Desk location is a number indicating the exact placement of an employees desk, and an image showing the main area of the floor, with the location marked. If this desk location does not match a known image, an unknown location image is shown.

[3] Availability statuses consist of a display showing a status from an Exchange service (available, at lunch, sick, vacation etc.).

Constraints

- The system must be written on the .NET platform. [Implementation requirement]
- The system must run on (minimum requirement) an iPad 2 with iOS version 5.1. [Implementation requirement]
- All code documentation must be written in the standard .NET format using XML. [Implementation requirement]
- The user interface must include the company branding. [User Interface Requirement]
- The status codes used for the app should be the same as the ones used in the company Exchange service. [Implementation requirement]

The client (Subsea7) will be responsible for installation and maintenance, including encasing the iPad to stop end-users from exiting the application. [4]

2.3 Use case overview

In Figure 1 all our use cases are shown. We have two users, the regular user of our application `Person` and the `Administrator`. The administrator is a developer working for Subsea7, who has access to an administration interface.

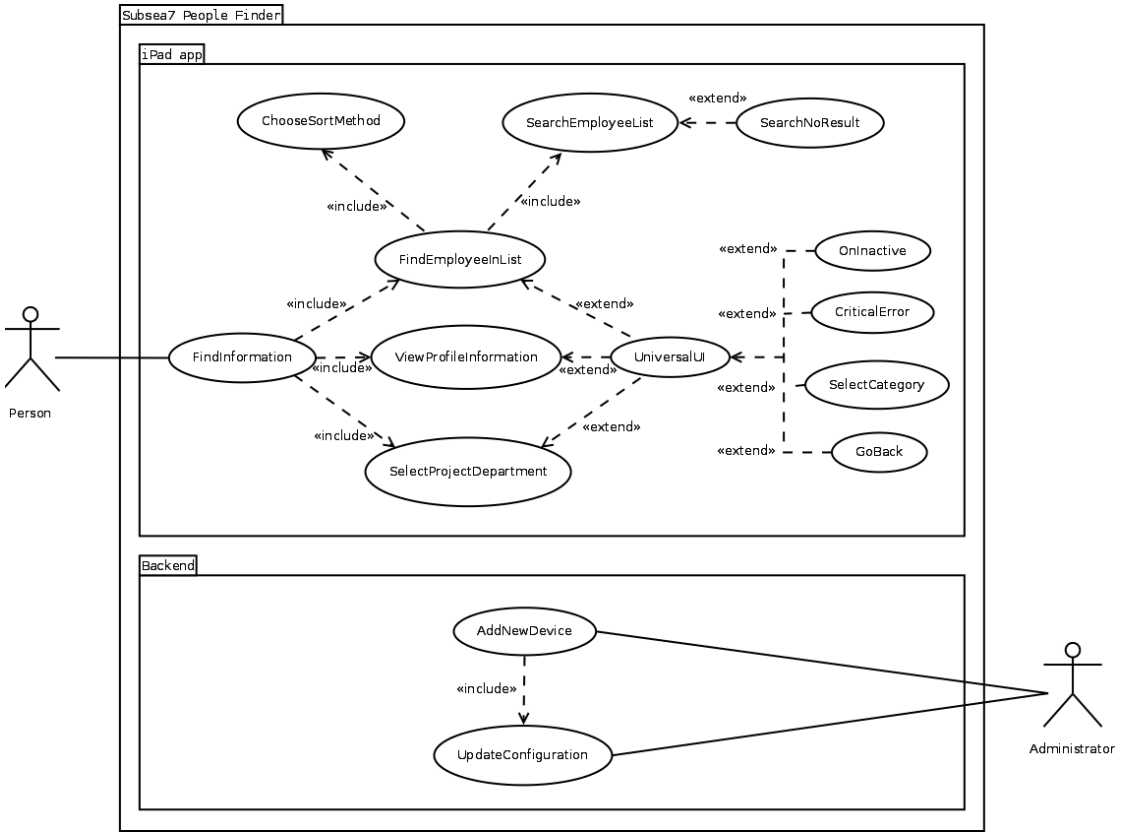

Figure 1: Use case overview

[4] The Apple iPad features a "home" button on the bottom of the device, which returns the user to the desktop. As the functionality of this button is impossible (prior to iOS version 6 and within reasonable limits) to override, it's necessary to provide a physical barrier to avoid a user pressing this button.

2.4 Use Cases

2.4.1 Use case 1: FindEmployeeInList

Use case name:	FindEmployeeInList [5]
Paticipating actors:	`Person`.
Flow of events	1. `S7Finder` presents a table containing a row for all employees (possibly filtered on a project or department. For each employee in there is a thumbnail-picture (with a coloured overlay indicating the availability of the employee), their full name, title, department, location is also shown. At the top of the screen there is a switchbutton with the text "sort by: Firstname | Lastname". Just above the table a search bar is displayed. 2. `Person` can at any time use the switchbutton, invoke the use case `ChooseSortMethod`. 3. `Person` uses his finger to scroll up and down the list to find the employee he is looking for, or uses the `SearchEmployeeList` use case. When `Person` finds an employee meeting the criteria he is looking for, he taps the row with the employee.
Entry condition	The `Person` has selected the top level category for all employees, or a specific project or department.
Exit conditions	The `Person` has selected an employee meeting the requirements he was looking for.
Quality requirements	A Person must be able to locate an employee meeting the requirements he was looking for within 1 minute, without prior training or experience with the system.

[5] This use case's sequence diagram is shown in Figure 3 on page 12

2.4.2 Use case 2: GoBack

Use case name:	GoBack[6]
Paticipating actors:	`Person`
Flow of events	1. `Person` taps the back button located in the upper left corner of the screen. 2. `S7Finder` respond be presenting the previous screen.
Entry condition	This use case extends any other use case that takes `Person` one level down in the application.
Exit conditions	`Person` has been taken one level up in the view stack.
Quality requirements	None.

[6]This use case's sequence diagram is shown in Figure 4 on page 13

2.4.3 Use case 3: SearchEmployeeList

Use case Name:	SearchEmployeeList[7]
Paticipating actors:	`Person`
Flow of events	1. `Person` tabs the search field 2. `S7Finder` responds by removing the switchbutton with the text "sort by: Firstname | Lastname" from the top right, removing the back button (if present) and adding a cancel button to the top left of the screen with the text "Cancel". A virtual keyboard is added to lowest area of the screen. 3. `Person` starts typing in the firstname, middelname or lastname of the employee he is looking for. 4. On each key stroke `S7Finder` updates the list, so it only contains employees' whose names (first/middle/last) begins with the search text. 5. `Person` may scroll though the list to find the employee he is looking for, and may hide the keyboard.
Entry condition	`Person` is on the employee list screen, and knows at least part of the name for the employee he is looking for.
Exit conditions	`Person` has found the employee he is looking for (also see SearchNoResult).
Quality requirements	The employee must be found within 1 minute.

2.5 Class diagram

The class diagram is shown in Figure 2, page 11. Some methods have been omitted for simplicity (E.g. for the GUI `ViewWillAppear`, `ViewDidAppear`, `ViewWillDisappear`, `ViewDidDisappear` `PrepareForSegue`). Relationships to the Utility and Config classes have also been removed for brevity. For a complete description of all parts of the class diagram, please see **??**.

[7] This use case's sequence diagram is shown in Figure 5 on page 14

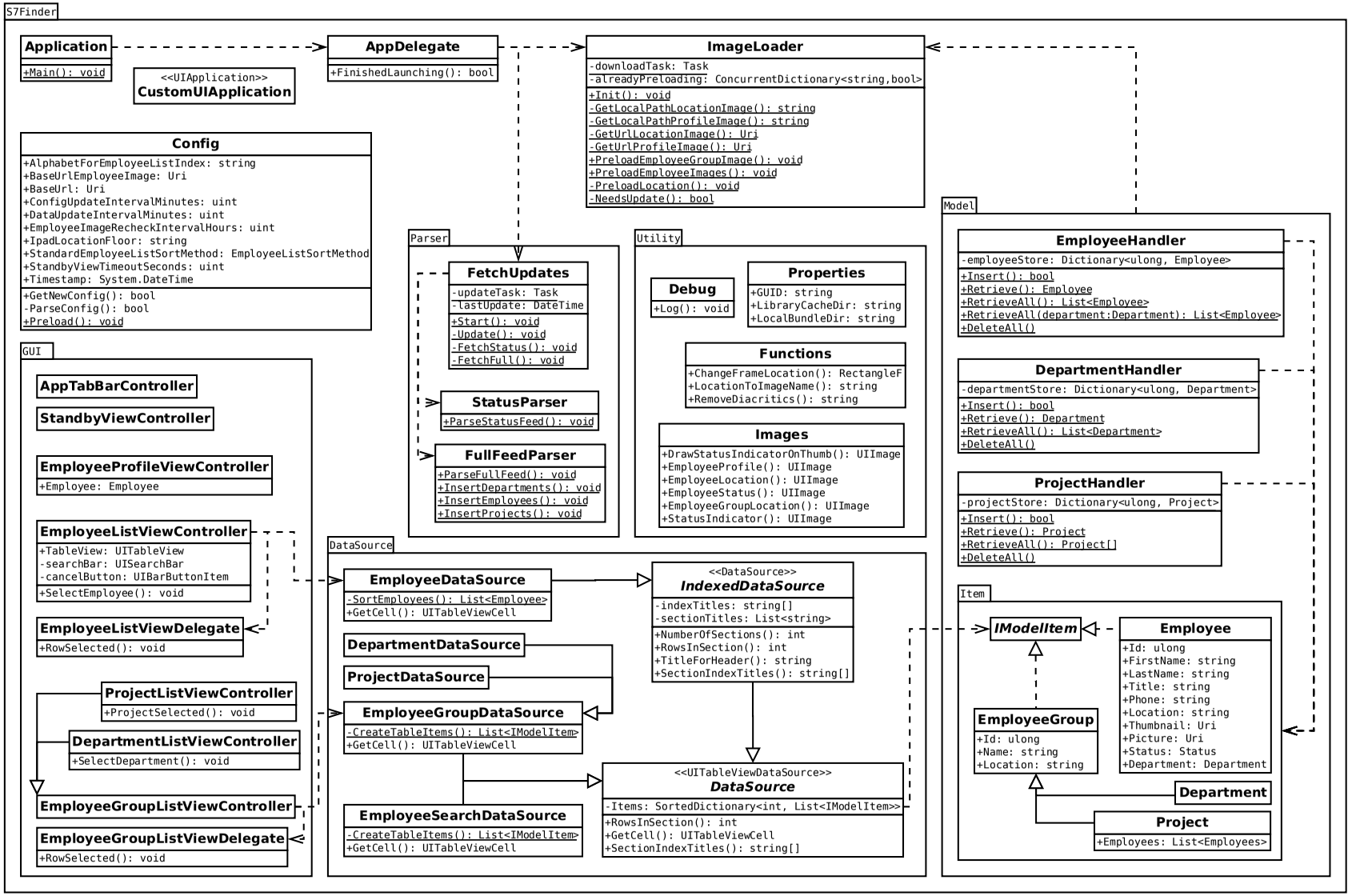

Figure 2: For a complete description of all parts of the class diagram, please see **??**

2.6 Sequence Diagrams

The sequence diagrams for the three use cases described in section 2.4 are shown below. Objects marked with a grey background are part of the iOS API.

2.6.1 Sequence diagram 1: FindEmployeeInList

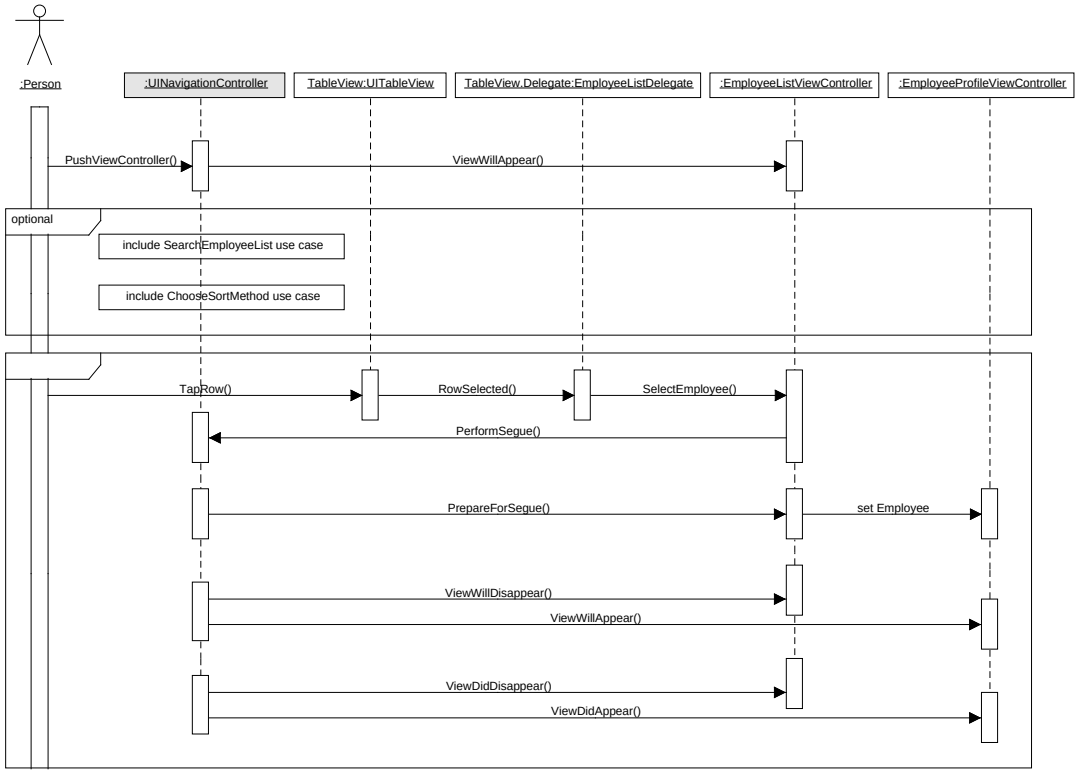

Figure 3: Sequence diagram of use case FindEmployeeInList

In Figure 3 the `TableView` is a property of the `UIEmployeeListViewController`. The lower part of the diagram has been inserted into a interaction frame, because all these events are started by the `TapRow`.

2.6.2 Sequence diagram 2: GoBack

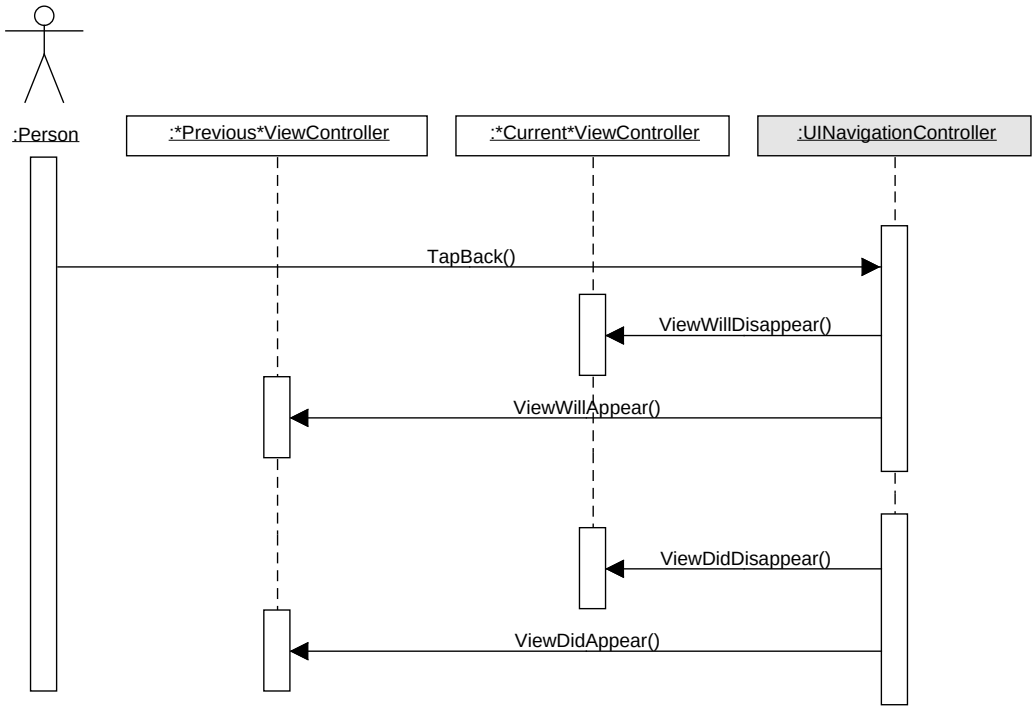

Figure 4: Sequence diagram of use case GoBack

In Figure 4 `*Previous*ViewController` and `*Current*ViewController` are placeholders for the actual `ViewControllers`, and is just used here to show the event flow.

2.6.3 Sequence diagram 3: SearchEmployeeList

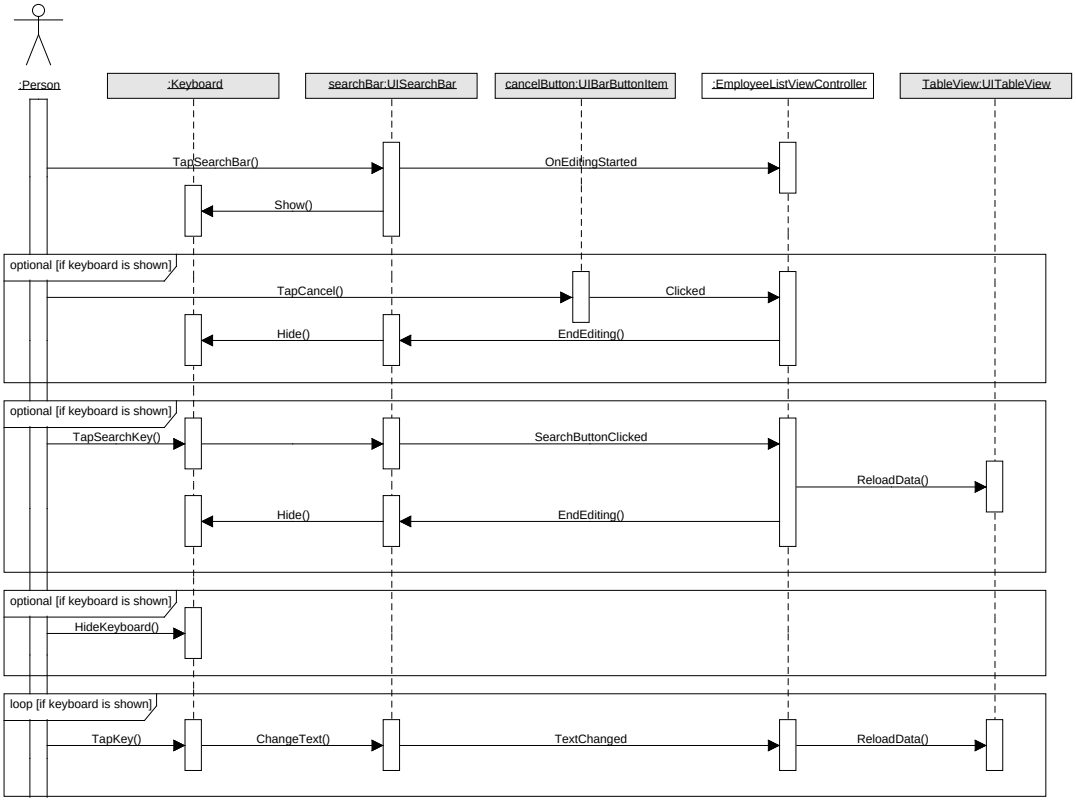

Figure 5: Sequence diagram of use case SearchEmployeeList

In Figure 5 the loop and three optional sequences can be executed in any order and several times.

3 System design

The S7Finder system is a loosely coupled high cohesion iPad application, which is used to find the location of employees at the Subsea7 offices. The system is highly compartmentalised, to ensure that it is both robust and easy to maintain. Data is passed to the system from a gateway server deployed by Subsea7. This gateway is outside of the scope of the S7Finder system, but will communicate with it using a protocol defined in the `Parser` subsystem.

AppDelegate
: The `AppDelegate` is the standard iOS controller for system events. It ensures that the application listens to startup events, and that all subsystems are properly initialized.

GUI
: The `GUI` subsystem handles the user interface in accordance with the principles of MVC. It uses the standard components of iOS to implement a cohesive user interface and handle the transitions between the different views.

DataSource
: The `DataSource` subsystem handles the mapping of data from our model into an iOS standard format for table data. It also handles indexing and filtering based on search, and ensures that all functions needed for the standard GUI components are implemented and overloaded.

Model
: Handles the loading of data into predefined classes to ensure that the data passed between the subsystems is well-formed and properly typed. The `Model` subsystem also holds the data in memory to ensure fast response times for queries. It also makes sure that no unnecessary or duplicate object creation is performed.

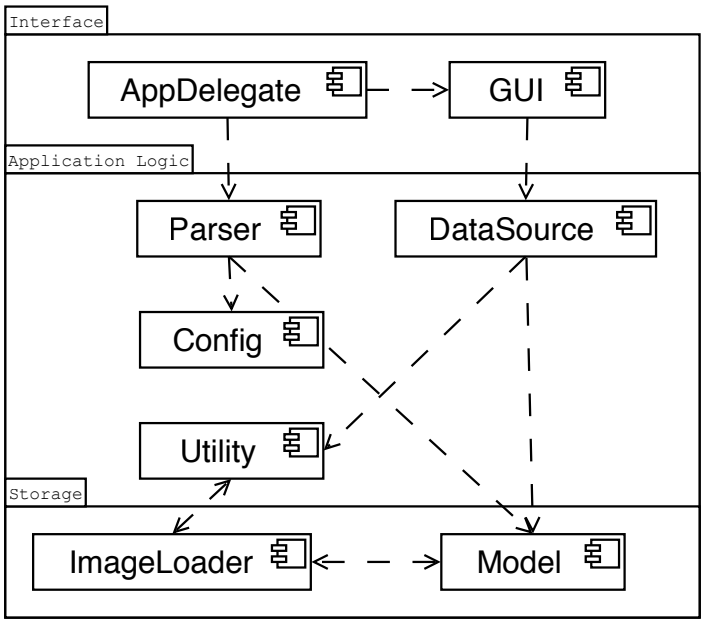

Figure 6: S7Finder subsystem decomposition

ImageLoader
Handles the asynchronous loading of image resources (employee-, location-pictures, and so on ...) into the GUI tables, and ensures that they are properly cached.

Parser
This subsystem handles the parsing of a communication feed gathered from the gateway server. It is responsible for extracting data from this feed for the initialization and insertion of the employee data. The `Parser` communicates with the `Model` subsystem to insert this data into memory.

Config
Enforces a standard for configuration of the application that one is unable to handle outside of a local scope. Also handles loading and caching of the configuration settings to ensure a minimum of network requests.

Utility
The `Utility` subsystem contains general methods needed to solve standard iOS problems across the system. It holds parameters generated at runtime, such as the unique identifiers for the physical device. The `Utility` subsystem also handles normalization of UTF8 text to ensure a sane environment for searching and grouping of employee names.

The view and controller has been combined as one class, called a UIViewController. Changing views is handled by a UINavigationController that manages views as a stack, so you can push to display a new UIViewController, and pop to remove the current. The actual user interface definitions will be created in a separate *.storyboard* file. The data tables defined in our storyboard is populated using the DataSource class, that contains implementations of the *UITableViewDataSource* interface.

4 Program and system test

Automated tests have been performed on the code using the NUnit framework included with Monotouch. The testing has been randomized to ensure that all edge cases are properly tested. A test server has also been created to serve randomly generated and updated test data for manual testing purposes. The tests run in the iOS Simulator and on iOS devices, to ensure correct behaviour on the target platform.

For more information, and the complete test specifications, please see Appendix A-D.

5 GUI and interaction design

5.1 User Interface

In the following we have included screenshots for some of the most important parts of our user interface. All data shown is our test data, has been randomly generated, so many people share the same image, have strange titles, projects and department names.

5.1.1 Employee List

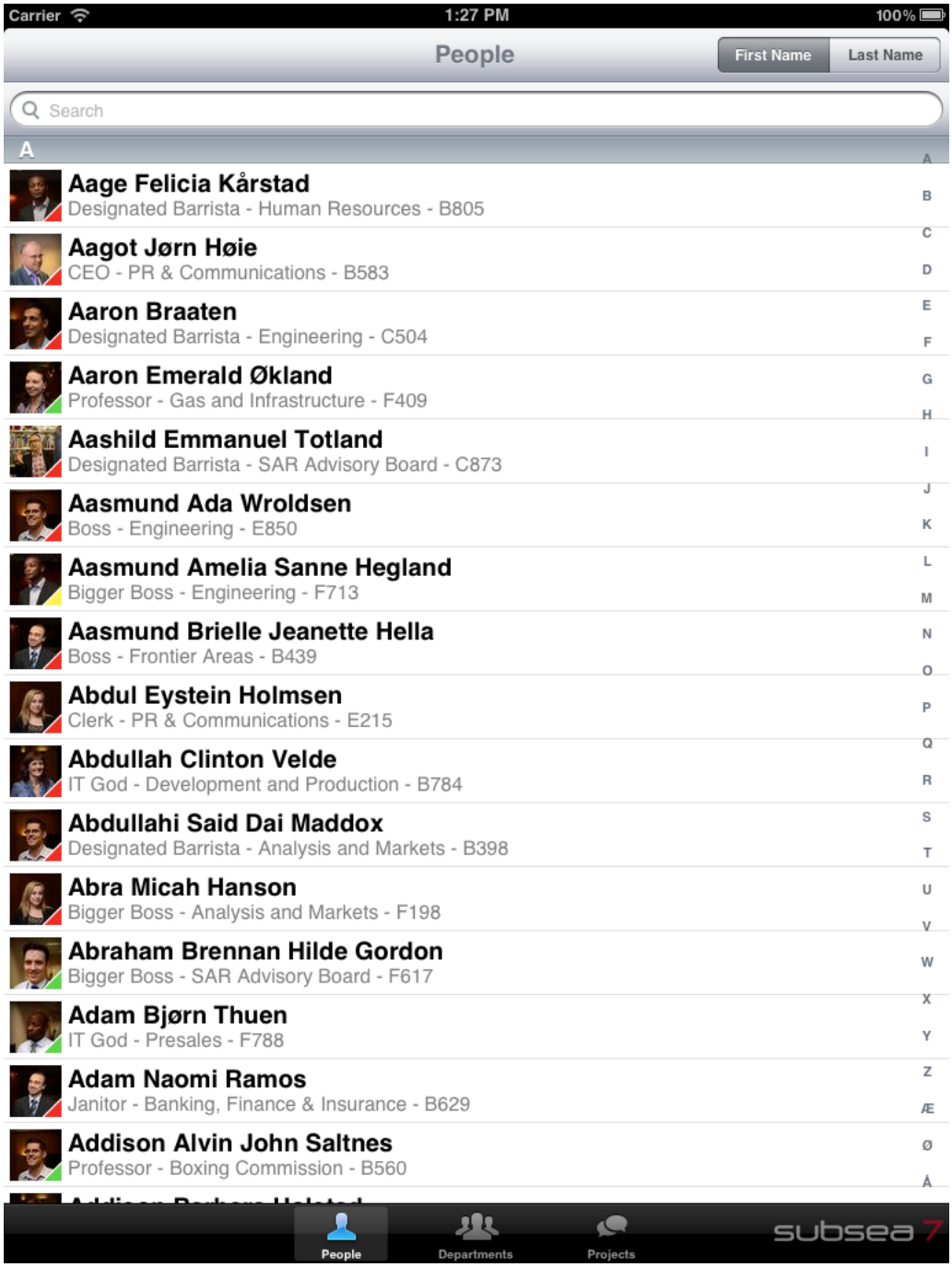

Figure 7: The top level employee list containing all employees.

5.1.2 Searching an Employee List

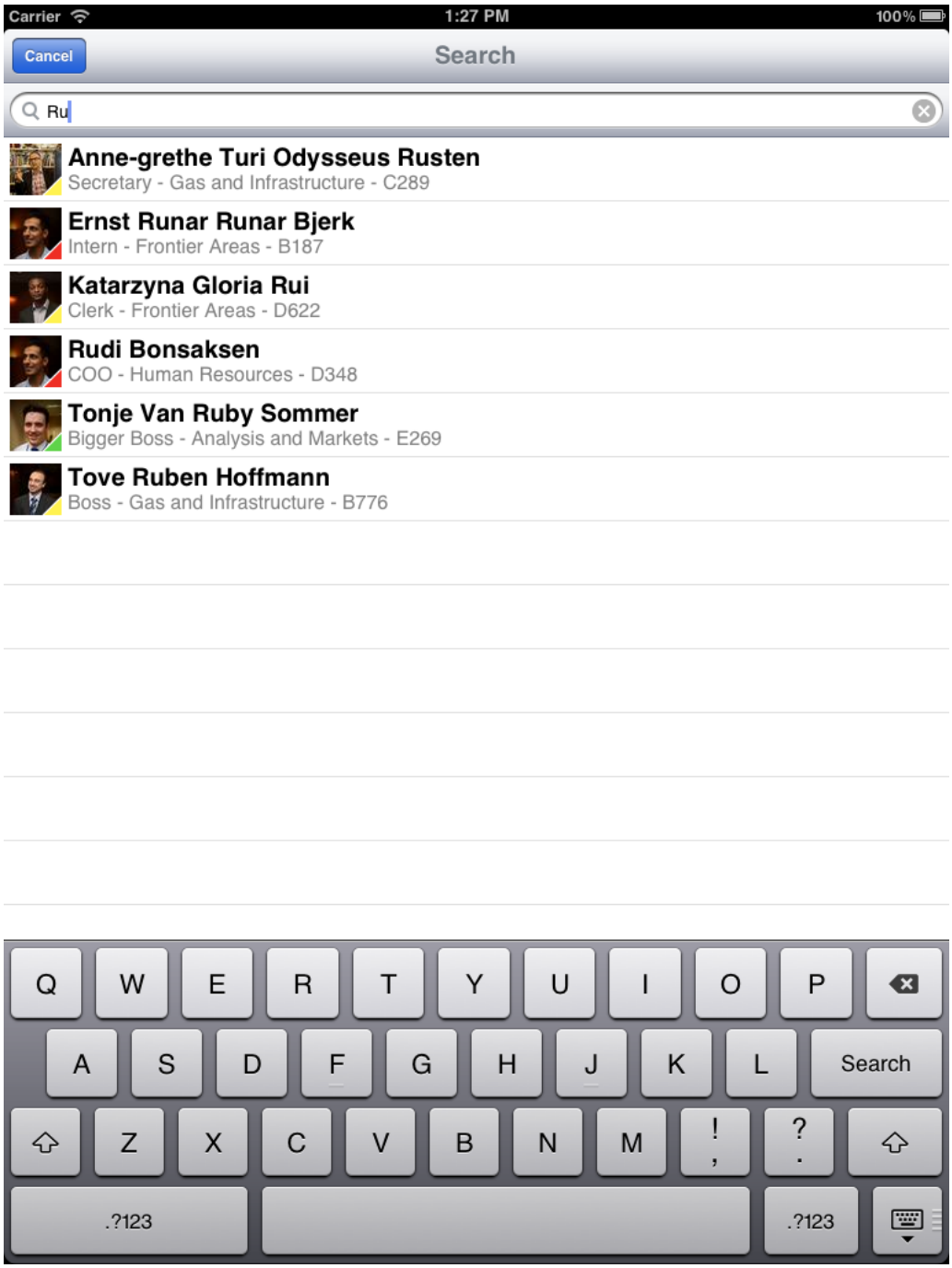

Figure 8: Searching an employee list with the text "Ru".

5.1.3 Department List

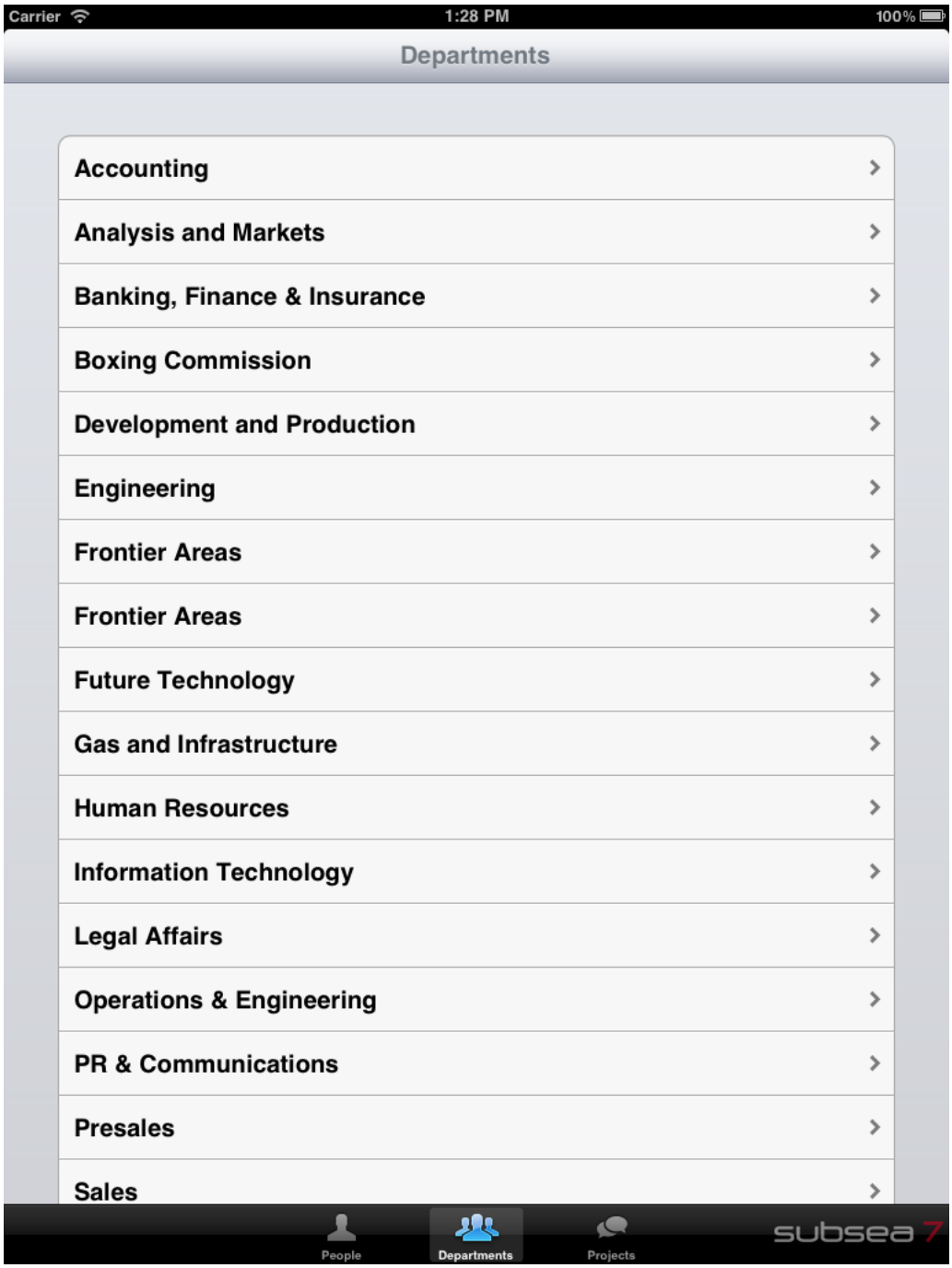

Figure 9: The top level screen displaying all departments.

5.1.4 Employee Profile

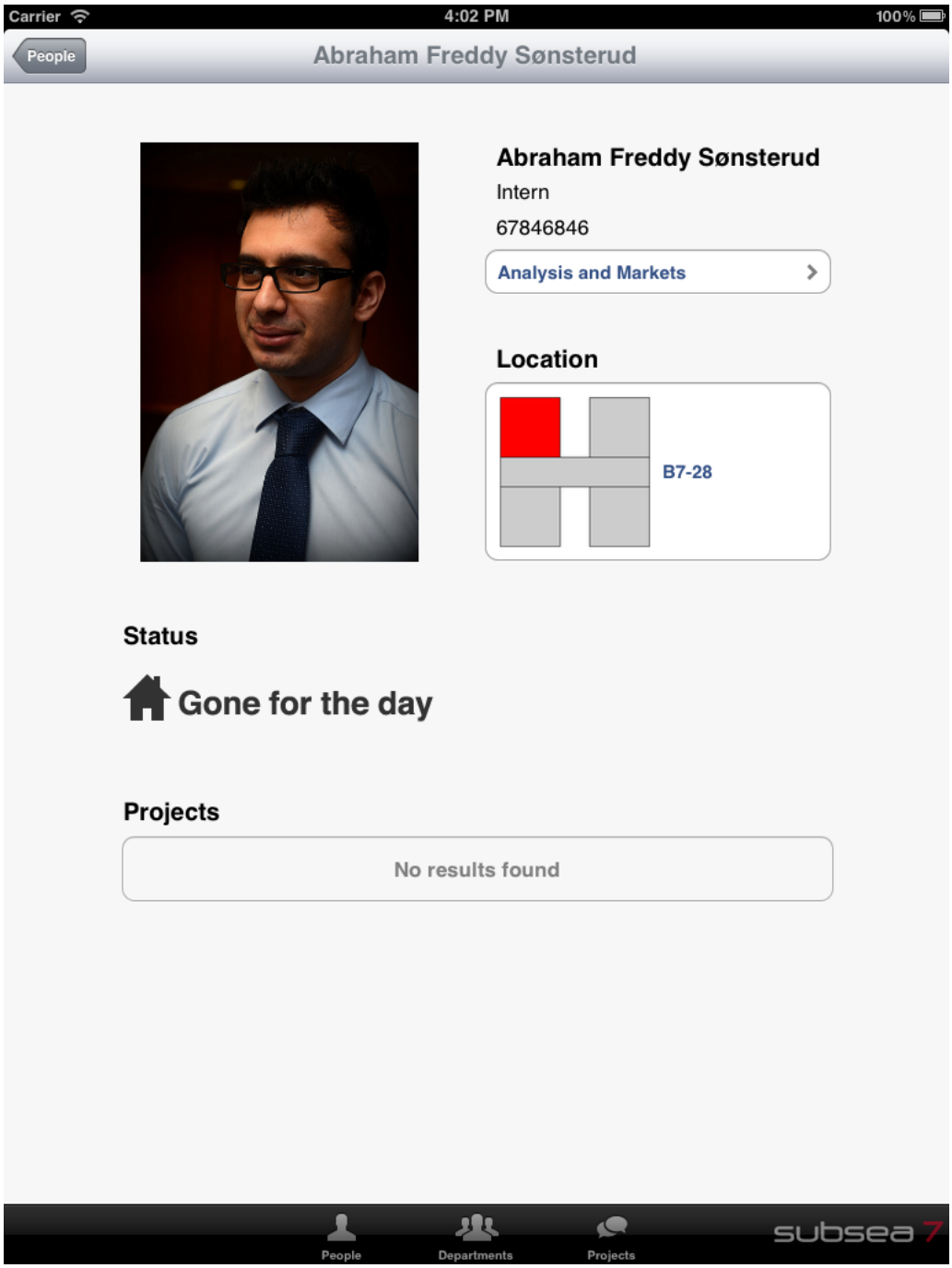

Figure 10: A profile for an employee.

5.2 User Interaction Diagram

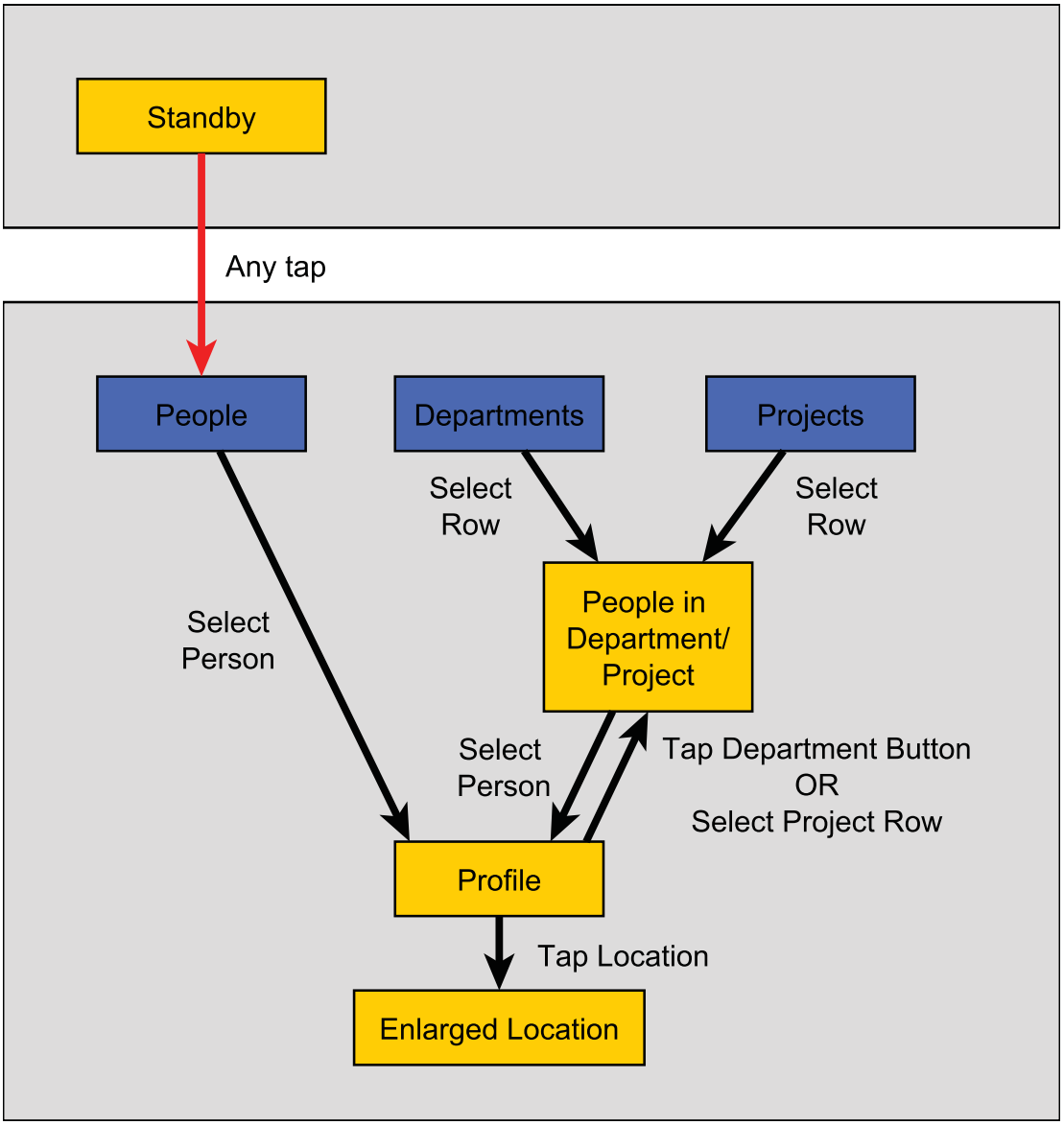

Figure 11: Navigation diagram for the user interaction.

Figure 11 shows the interaction flow for our application.

Small boxes represent a screen in the application. The blue boxes indicate top level screens when the program is active. Switching to one of these can be done at any time (except when the location image is shown in an enlarged version, in which the pop-up has to be closed first). Doing so will reset the current state.

The two large boxes group the screens, showing if they're used when the program is user interaction enabled and actively used (lower) or when inactive (upper).

Arrows indicate a transition from one screen to an other. It is possible to follow the arrow back to the previous screen. Black arrows are initiated by users, and the users control when (and if) to go back. Red arrows are initiated by users, but the application controls when to go back.

5.3 Think-aloud experiment

We did a little test of our GUI when we first arrived at Subsea7 at our final meeting in Stavanger. We needed to test if the user interface was clearly understandable. For the think-aloud experiment, the test subject, a woman of age 32, were asked to complete the following tasks:

Where does "Adam Bjørn Thuen" sit? She looked at the standby screen, quickly tapped it and started reading the names on the list, from the top. She found him pretty quickly and tapped his row, and could tell us the correct answer: "F788".

What status does "Jørgen Kjær" have? She spent a little time looking around at the screen, but found the back button pretty fast. Then she started scrolling down the list. After reaching "D" she got tired of that, and tapped the search bar. Then she wrote in "J Ø R G E N", and looked up to see if he was there yet. "Jørgen Kjær" had been the only result since "J Ø R", but she hadn't noticed. She tapped his row and could tell us that he was on vacation.

Find someone "available" in Jørgen Kjærs' Department. She noted his department "Human Resources" and tapped the "Departments" button on the tap bar in the button. Then she found the correct row, and tapped that. Then she started tapping the first row, going back because the employee was unavailable, tapping the second row, going back because the next employee wasn't available either, and so on until she reached someone who was available.

Find out where to go for visiting the "Sales" department. She tapped the departments button again, this time selecting "Sales". She didn't notice the button that would bring up the general location, but selected a few different employees to try finding a pattern in their location (there wasn't one because we hadn't made any such connection when generating our data). So she told us they where sitting all over the place in locations that didn't even exists in their building.

So testing with real data would definitely have been a lot better, as our generated data had a bad effect on the user test – things didn't work as our test person was used to. But, we did see some problems with our user interface that could be improved, and resulting in the following improvements:

- We made the status indicator overlays much larger and with much stronger colours.
- We added a disclosure indicator on top of the department button (the little arrow), so people would easier understand it was a button.
- We changed the way to display project/department locations, so they're just shown in the top no matter what.

After implementing the above, we performed the same test again with real data, albeit with altered names, and it went a lot better. The new test person understood the coloured triangles, that the department button was a button, and couldn't miss the location for the department. The screenshots shown in User Interface, subsection 5.1, page 16 are from this improved version.

6 Audio/Visual presentation

The presentation can be found at `http://www.youtube.com/watch?v=aTGoGxTlfmY`.

7 Implementation of the parser

This section is about the decision making and implementation of the parser. I have chosen the parser because it has been rewritten in the process and I will explain why this was a good decision.

First, take a look at the initial parser code, parsing the employees from the XML file.

```
XNamespace f = "http://internal.s7.net/S7finder";
XDocument doc = XDocument.Load (dataFeed);

XElement employeeRoot = doc.Root.Element (f + "employees");
if (employeeRoot != null) {
    IEnumerable<XElement> employees =
        employeeRoot.Elements (f + "employee");

    foreach (XElement e in employees) {

        if (e.Attribute ("delete") != null &&
            Convert.ToBoolean (e.Attribute ("delete").Value)) {
            model.Employee.delete (db,Convert.ToInt64(e.Attribute("id").Value));
        } else {
            byte[] test = {1,2,3,4};
            Int64 employeeID = Convert.ToInt64 (e.Attribute ("id").Value);
            string firstName = e.Element (f + "firstName").Value;
            string lastName = e.Element (f + "lastName").Value;
            string title = e.Element (f + "title").Value;
            Int64 phone = Convert.ToInt64 (e.Element (f + "phone").Value);
            string location = e.Element (f + "location").Value;

            model.Employee.create (db, employeeID, firstName, lastName,
                                   title, phone, location, test, test);
        }
}}
```

The code above was part of the very first parser.

This simple part of the parser looks for an attribute called delete. If the delete attribute is true, then we delete the employee from the database, else we insert the employee.

After more research in the field of C# and XML, we found the LINQ package, could help us with readability and robustness. Also we had some changes to the design of the application, as an example the images and thumbnails should no longer be a part of the XML file, but was loaded separately from a webserver, based on the employeeId.

So we had more reasons to do a code rewrite. Below are shown the same code snip after the rewrite:

```
var ns = xdoc.Root.Name.Namespace; var results = from el in xdoc.Root.Elements
(ns + "employees").Elements (ns + "employee") select el;

foreach (var r in results) {

    ulong id = (ulong)r.Attribute ("id");
    string firstName = (string)r.Element
    (ns + "firstName");
    string lastName = (string)r.Element (ns + "lastName");
    string title = (string)r.Element (ns + "title");
    string phone = (string)r.Element (ns + "phone");
    string location = (string)r.Element (ns + "location");
    Status status = (Status)Enum.Parse ( typeof(Status),
                                        r.Element (ns + "status").Value);
    ulong department = (ulong)r.Element (ns + "department");

EmployeeHandler.Insert (id, firstName, lastName, title, phone, location,
status, department );
```

After the code rewrite I think that the code was improved in the following ways. In regard to readability, the code now uses LINQ, thus a SQL-like syntax "from * in * select" are used, this syntax should be well known for many programmers, so in that regard I will argue that we improved readability. We found that LINQ, did a better job at handling flawed XML files and dynamic namespaces, thus we improved robustness. Because we found a more optimal solution for image fetching, the parser has become more simple. I think in all way this code rewrite improved the implementation of the parser.

Part II
About collaboration with users and within the team

8 Schematic course of events

Below are shown a schematic overview, listing the course of events in ascending order by date. Our weekly monday meeting are excluded from this overview.

Date	*Event*
2012-02-07	First PKSU lecture
2012-02-20	Initial group meeting
2012-02-23	First Skype meeting with Subsea7
2012-02-27	Discussed an initial design for the application
2012-03-01	Subreport 1 submitted
2012-03-21	First on location meeting with Subsea7, Stavanger
2012-03-29	Subreport 2 submitted
2012-04-17	Deadline for running beta version
2012-04-18	Second on location meeting with Subsea7, Stavanger
2012-04-26	Subreport 3 submitted
2012-05-20	Test phase for stable version started
2012-05-22	Stable version deadline
2012-05-23	Work at Subsea7, Stavanger
2012-05-24	Work at Subsea7, Stavanger
2012-05-31	Subreport 4 deadline
2012-07-	Finale handover

Overall our planning and initial scheduling for the project has been good. We have been good at discussing realistic estimates for specific tasks. In some ways it was an advantage that Subsea7 is located in Stavanger, because from the start of the project, we were forced to figure out flight dates etc. Thus a detailed schedule was made early in process.

9 Parties involved in the project

There have been two parties involved in the project, besides our development team. Below are a brief description of the parties:

Subsea7 IT-department The IT-department has been our primary contact. All communication with Subsea7 has been handled through this department. The department has IT-knowledge which allowed discussions about the application to happen in technical terms.

End users at Subsea7 The contact with the end users, has been handled by the IT-department. The only direct contact we have had with the end users has been as part of think-aloud experiments and similar tests.

The fact that Subsea7 has a quite big IT-department has had a big impact on the project. The IT-department had already made many decisions regarding the application before we entered the project, and at times it has been difficult to bring new ideas to the table.

As an example, they did not see any requirement for the think-aloud experiments. They had already discussed how they wanted the application, and in their eyes, our job was more or less "just" to code it, while we wanted to discuss and challenge the decisions they had made in advance. Thus there has been some discussions, where we needed to explain that it was a requirement, that we could do certain experiments and present our suggestions for improvements. Otherwise we would not be able to produce satisfactory subreports for the course.

Besides the matter mentioned above, all collaboration has been more or less frictionless.

10 Team collaboration

The internal team work in the group has been working remarkably well. We have been working quite homogeneously and where able - as a team - to increase the work pace towards deadlines and presentations. As an example we introduced daily Skype meetings the week before our last presentation in Norway, where we discussed the development progress, technical implementations and each team member presented his personal work progress. This enabled us to keep a high pace, and enabled the group to immediately identify problems in the work flow. We also established an IRC channel, where all members are present (mostly 24/7). Thus, it is easy get feedback on implementation ideas etc., doing the day.

At that start of the project we decided to use Pivotal for project management, but because the tasks and architecture were not that complex yet, it seemed as if we did not benefit enough from the tool, if we took into account the time required to keep Pivotal up to date. Thus we forgot/stopped using it over time. As the project has grown we found that it has been hard to sustain an overview of the tasks that needed to be completed. To restore a clear overview we agreed that every team member will use Pivotal for the rest of the development process.

The only strenuous situation we faced was in regard to the hand in of the second sub-report. At that time, due to other exams, the communication was not good enough. Thus there was not enough clarity in regard to the progress of writing the sub-report. To prevent such situations in the future we did some ongoing evaluation of each others work towards the deadline, and at the weekly meetings we used the report requirements to check that we had remembered all the required parts.

11 Strength and weekness's in the planing

I find that we as an developemnt team have had a reasonable allocation of the different parts in the developmentprocess. In the group we have been good at continuous evaluation, at our weekly monday meetings and also we have distributed responsibily as it is mentioned in [BD10] s. 113. This has been very importent because we have been able to react and adjust the schedule when new and better implementation ideas have come to the table or unforeseen changes has occured.

Subsea7 has been a good and reliable business partner in this project, but the planing regarding the webservice they should deliver has not been optimal. They are very busy and it would have been good for the project, if they earlier in the process would have known how they would like to serve updates to the ipads. Because we did not, know how they would serve updates we made the system way to advanced, so it would be able to handle incemental updates. If we from the start had known, that they did not intend to serve incemental updates, we would have had a better idea how to analyse and design the application correct earlier in the process.

In the allocation of resources I also find, that we have had a far too linear approch to analyse, design and implementation, instead of the approach described in [CCD+98] s. 23 (see figure 7). When we finised the first revision of the analyse, design og implemtation we thought of it as more or less finale. And only later in the process we realised that many revisions were needed to come up with an optimal solution. So it would have been better to do analyse, design and implemtation more losely, and then repeat the process instead of trying to get it perfect the first time.

12 Suggestions for improvement

One thing we have learned from this project, is that software development requires many iterations, and thus one have to be aware, not to approach the result from the design, analyze or implementation as finale, but rather as a state of some running evaluation. The cource material seams to agree with the quote "a project plan is developed iterativly"[BD10] p.698 and also in [CCD+98] s.23 fig 7.

Also we would have been able to improve the project process, if we had been better to set demands for deadline and informations from Subsea7. Mabye we felt, that we "needed" Subsea7, more than they needed us, and thus we have been reluctant with demands. With more (and still fair) demands we would have been able to get a more comprehensive specification requirement, earlier in the process, which would have saved us some code. Also we would have been completely done at this point, if Subsea7 have had their xml-feed service ready in time. To hold our own deadlines we have been using a scrum-like organization model [BD10] p. 647, and we have had great success with this model. Thus we will suggest other devlopment teams to check out this model if they are not familiar with it.

As mentioned earlier, our project is 98 pct. done and subsea7 is pleased with the result, and so are we. Furthermore there have been only minor problems in the planing and development process. I think we did a good job, and looking back I don't see other things that we could have done (but not did), to improve the planing and the development process.

Part III
Article discussion

13 Literature review

In the following section we will review two articles, namely "Programming as theory building" by Peter Naur (of DIKU fame), and "The M.A.D. Experience: Multiperspective Application Development in evolutionary prototyping" by Michael Christensen, Andy Crabtree, Christian Heide Damm, Klaus Marius Hansen, Ole Lehrmann Madsen, Pernille Marqvardsen, Preben Mogensen, Elmer Sandvad, Lennert Sloth, Michael Thomsen from the computer science department at Århus University. [Nau85, CCD+98]

13.1 Programming as theory building

In "Programming as theory building", Peter Naur discusses the notion of programming not as a discipline where program code is produced ("program texts" in the article), but rather a discipline where the product is the design, including documentation and rationale for the design decisions.

To support the Theory Building View, Naur lists a few anecdotes, which we may gain some insight from, and draw some parallels to.

With our project, we're essentially working as external software developers, catering to the needs of a client. Some of the restrictions regarding implementation are based on the hardware platform, while others are requirements set by Subsea7. One of these restrictions is that the code should be developed in C#, in order to enable developers at Subsea7 to continue development if necessary.

In *Program Life, Death and Revival*, Naur argues strongly that it is insufficient for a new programmer joining a project to become familiar with a certain program and its corresponding documentation, but that it is an actual requirement to work alongside other programmers with an intimate knowledge of the system, as to convey the program theory. By extension, a revival of a program when following the Theory Building views, is impossible, as the need for a means of conveying the program theory remains unsatisfied.

As external developers, the project we're delivering has some detachment issues when considering Naurs model, as at is unlikely that a future developer would take contact to our project group. One might argue that the need for contact follows the size of the project, and that for a project of the size of S7Finder, any developer should be able to understand the code after reading it a few times. On the other hand, we may also argue that some of the original design ideas are likely to be lost, and inconsistency may arise. Although the code is fairly well-documented, some features may seem less than intuitive, e.g. the use of file timestamps and the `touch`-ing of files.[8] This may can be seen as a loss of theory, hence supporting Naurs reasoning.

Though the text is written in 1985, 27 years ago, the notion of programming as theory building as opposed to text production still holds. Drawing parallels from our own project, we realised during the fourth meeting with Subsea7 at their offices in Stavanger, as discussed in **??**, **??**, page **??**, that they wouldn't make use of the originally implemented features.

13.2 The M.A.D. Experience

The article "The M.A.D. Experience: Multiperspective Application Development in evolutionary prototyping", the development of an application, a Global Customer Service System (GCSS) for a Danish goods transport company is described. Although the article is extremely explicit and favours unreasonably long paragraphs (a third of a page is not uncommon) and intellectual-sounding words over simplicity, while bordering on incomprehensibility on account of the required level of understanding of ethnography, some valuable points may be found in the article.[9]

As we have demonstrated in our development process, the design phase is a cooperative activity, and at our visits to Stavanger, we took the opportinuty to act on inputs from Subsea7. In the development of the GCSS, the group from CIT followed a similar model, although much more in-depth, as the GCSS project is bigger both in scope and duration. Where we thrice took the trip to Stavanger and listened to ideas, complaints and constructive criticism, the CIT group had opportunity to visit several offices, both locally and globally (Malaysia, Singapore), for a duration of multiple weeks.

In some places the two design processes differ. In "The object-oriented perspective", the CIT group notes that their focus was "...on modelling as opposed to technical concepts like encapsulation and inheritance". As this has been the first major OO project done by either of the group members, our modelling have served the same goals, but might have been more based

[8] `touch` is a standard Unix/Linux command, that in its base case updates the timestamp of a file to "now".

[9] E.g.: what is the difference between a "naturalistic description" and a regular one? (pg. 10)

on programming experience rather than pure modelling. Our choice of programming language and development platforms may also have been a contributer to the difference in modelling, as we've had the task to not only structure the project, but also learn a new programming language (only one group member had any real experience with C#) and a full set of development tools (XCode and MonoDevelop). Further, our group consists of four programmers, while the CIT group consisted of an ethnographer, a participatory designer and developers.

As they experienced, noted in "The evolution of the model", our initial model differs quite a bit from the final model. As explained in **??**, **??**, page **??**, the final meeting with Subsea7 resulted in removal of some features, and the simplification of others. While it may have eased the design process to have an initial model more like the result, it is our experience that this hasn't been a hinderance. Rather like the conclusions of the CIT group, it is our understanding that "a model cannot be effectively created in isolation from implementation". While this may not always be true, preserving true separation between code and model might be detrimental to implementation efforts.

References

[BD10] Bernd Bruegge and Allen H. Dutoit. *Object-Oriented Software Engeniering.* Pearson, 2010.

[CCD+98] Michael Christensen, Andy Crabtree, Christian Damm, Klaus Hansen, Ole Madsen, Pernille Marqvardsen, Preben Mogensen, Elmer Sandvad, Lennert Sloth, and Michael Thomsen. The m.a.d. experience: Multiperspective application development in evolutionary prototyping. In Eric Jul, editor, *ECOOP'98 — Object-Oriented Programming*, volume 1445 of *Lecture Notes in Computer Science*, pages 13–40. Springer Berlin / Heidelberg, 1998. 10.1007/BFb0054085.

[MMMNS00] L. Mathiassen, A. Munk-Madsen, P.A. Nielsen, and J. Stage. *Object-Oriented Analysis & Design.* Marko Publishing House, 2000.

[Nau85] Peter Naur. Programming as theory building. 1985.

A Test plan

A.1 Introduction

The main objective of this test plan is to determine the robustness of the application's core functionality.

In consultation with the client we have decided to test only the core functionality of the application. This decision has been made in regard to the remaining time to deadline, where we find other tasks to have greater priority.

A.2 Relationship to other documents

To better understand the scope of the application and this test plan, we recommend that the reader read subreport four before proceeding. After reading subreport four, the reader will have a clear overview regarding the functional requirements, non-functional requirements, technical design and purpose of the application.

A.3 Features to be tested/not to be tested

All functionality in the model handlers will be tested. For Employees, Departments and Projects, this include all methods.

The DataSources will be partially tested, this is due to the fact that the DataSources extend many iOS classes, and we do not intend to test the methods implemented by MonoTouch.

The application as a whole, will be tested with around 130.000 random generated datasets. These tests will be quantitative, and thus we will not test that the output data are as expected, the purpose of these tests are to test robustness of the data fetcher class (`FetchUpdates.cs`) and the two XML parser classes (`FullFeedParser.cs` and `StatusParser.cs`)

As explained in the introduction, only core functionality is tested. Therefore this testplan does not include tests of functionality contained in the `Config.cs` class.

Also it has been decided that actual GUI testing falls outside the scope of this test plan.

A.4 Approach

Where it is possible we will use the NUnit testing framework.[10] NUnit enables us to isolate each part of the program and show that the individual parts works as expected. NUnit is included in the Monotouch framework and runs on the iOS Simulator, allowing us to test all functionality on the target platform. By testing the parts of a program first and then testing the program itself, integration testing becomes much easier.

Our unit testing has two main purposes, verification and validation. The verification process confirms that the software meets its technical specifications. A specification is a description of a function in terms of a measurable output value given a specific input value under specific preconditions.

The validation process confirms that the software meets the business requirements. A defect is a variance between the expected and actual result.

Where unit testing are not feasible, we use manual testing to achieve verification and validation, and thus that our code works as expected.

Finally we have developed a web service in python, serving XML-feeds to the application. This web service is used to emulate the production environment, and we generate and serve around 130.000 random-generated datasets this way. These tests are quantitative and for robustness only, and thus they do not focus on validation and verification.

[10] NUnit is an open source unit testing framework for Microsoft .NET.

B ModelHandlerTests

B.1 Test items

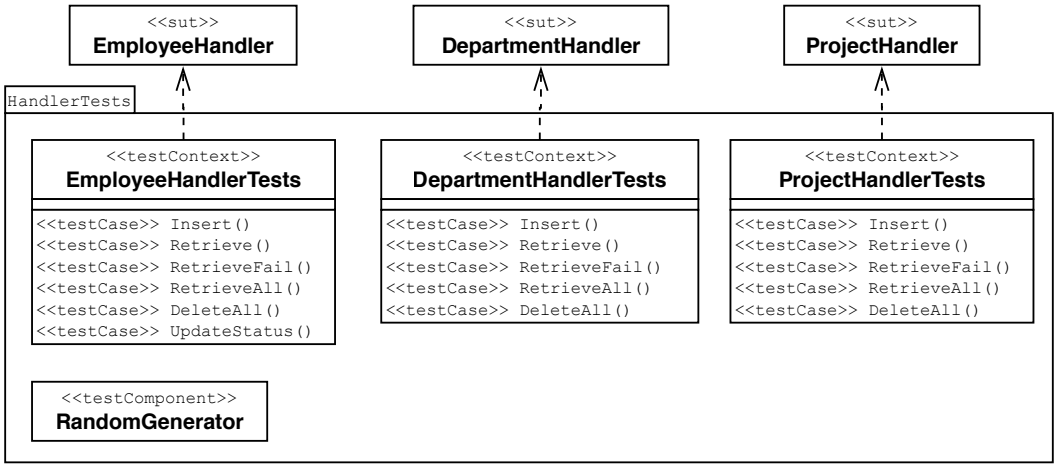

Figure 12: Test system for the S7Finder model handlers

This tests case covers the handlers for the model objects. These handlers control the life cycle of all model objects, and are responsible for caching them. All features of the handlers will be tested to ensure expected behaviour. This includes the throwing of exceptions in error cases. Random input is fed to the classes to ensure that all edge cases are found.

B.2 Input specifications

To ensure robust testing, all inputs are randomly generated. This includes IDs for non-existant employees, and the values for randomly generated employee, department and project objects. This enables us to think about the conceptual operations the methods under testing perform, instead of the specifics of one object.

B.3 Output specifications

Outputs from all methods are tested against the randomly generated values. No transformations are made on the data, as all processing is done on the gateway server. This enables simple output tests which contain logic that is easy to validate.

B.4 Environmental needs

As specified in the Test Plan, all tests must be run in the iOS Simulator through the Monotouch port of NUnit.

C DataSourceTests

C.1 Test items

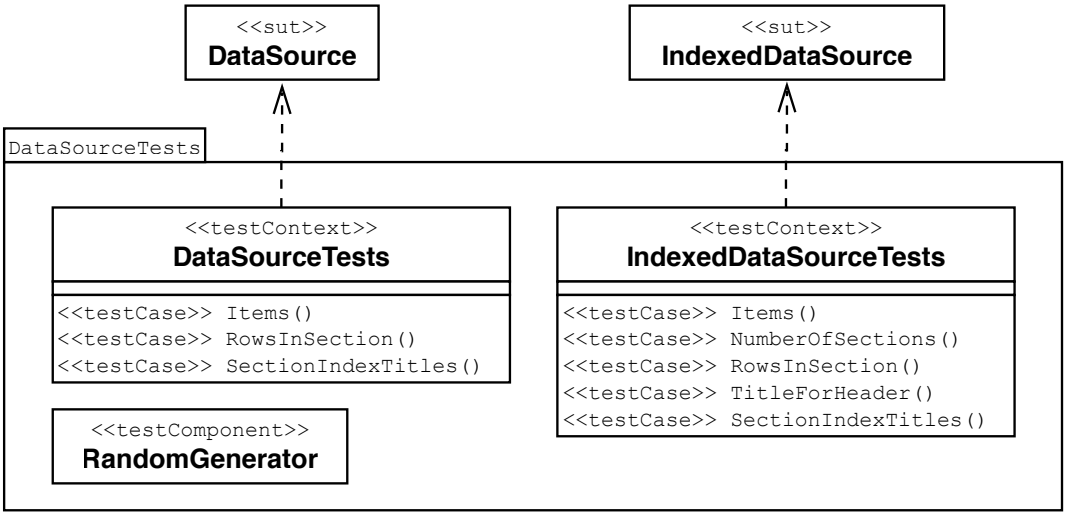

Figure 13: Test system for the S7Finder data source

This tests case covers the datasource. The datasource maps data from the model into the GUI tables defined by iOS. Only a limited subset of the functionality can be tested due to the enormous amount of predefined iOS methods inherited from the neccessary base classes. The scope of the test will therefore be limited to the most basic functionality.

C.2 Input specifications

All testing is done using randomly generated IModelItems. The datasource only holds these items in a predefined iOS format. It is therefore the execution of the mapping concepts that are tested, and not specifics of an item list.

C.3 Output specifications

As all values are random, the output tests are made against the generated input. All that must be ensured is that the items are grouped correctly, and that no data is lost during the mapping.

C.4 Environmental needs

As specified in the Test Plan, all tests must be run in the iOS Simulator through the Monotouch port of NUnit.

D Test summary

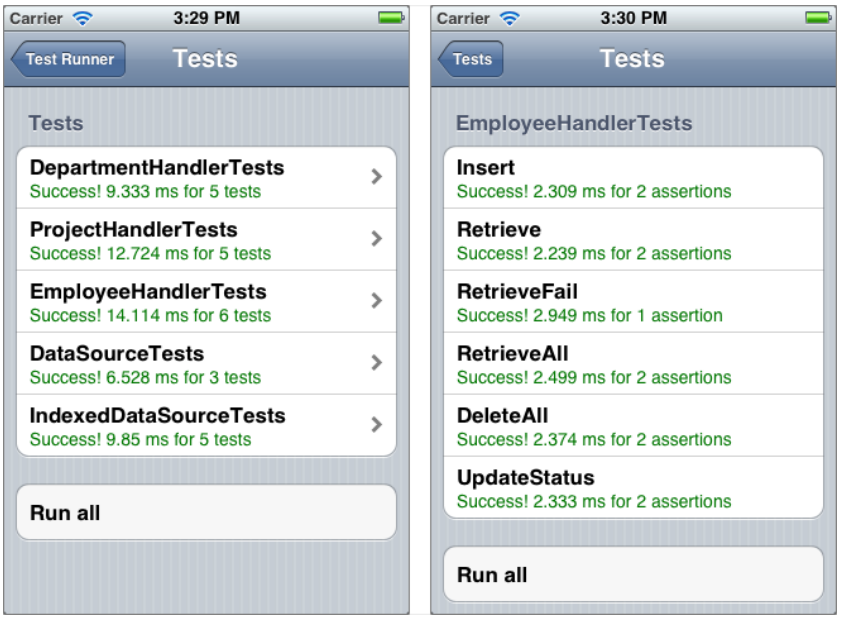

Figure 14: Test run results

The current results of the automated tests show that both the model handlers and the data source conforms to our specifications on the target platform. Future tests will focus on expanding the randomization of tests to ensure that all possible edge cases are covered. A future test focus will also be the Parser. Tests for this subsystem have thus far been based on the randomized test server, and verified manually. We wish to generate a static test set that we can test the functionality against. This ensures that all output from the parser conforms to all expectations for the data.