An agricultural Cyber-Foraging system for resource-challenged environments

Reuel Brion

Faculty of Exact Sciences

VU University Amsterdam

A thesis submitted for the degree of

Computer Science, Track Software Engineering and Green IT

2015

1. First Reader: Patricia Lago

2. Daily Supervisor: Grace Lewis

3. Second Reader: Christophe Guéret

Abstract

Cyber-foraging is a technique that extends the capabilities of mobile devices by using external resources, often in the form of surrogate computers. The proliferation of mobile devices with increased capabilities in developing regions offers possibilities of using cyber-foraging tactics to create meaningful services for people living and working in these regions.
In this paper, a set of usage scenarios for an agricultural knowledge exchange system to be used in resource-challenged regions is proposed, from which a set of requirements is created. Based on those, a set of cyber-foraging tactics was selected and used to create an architecture for the system.
A functioning demo implementation was created using this architecture, indicating that the selected tactics can be used to successfully realize the functional as well as non-functional requirements of this kind of system.

Contents

List of Figures

List of Tables

LIST OF TABLES

1

Introduction

1.1 Context

Along with the proliferation of mobile devices around the world comes a large potential to deliver many valuable services to developing areas. Initiatives such as the VOICES project (1) have been developing these kinds of services, particularly for feature phones. Now that there are more and more smartphones being released that target developing markets, and with prices approaching those of feature phones (2), there is merit in making use of the increased capabilities and improved usability that smartphones can offer.

Cyber-Foraging is a technique that focuses on extending the capabilities of mobile devices with the computational capabilities and storage of external resources, often in the form of local servers called surrogates. As many developing areas have to deal with a lack of proper access to resources such as Internet and electricity, Cyber-Foraging offers potential solutions to these resource challenges by leveraging proximate surrogates that can provide services that involve heavy computation such as image processing, store large sets of data collected in the field, or store information retrieved from data centers during scarce moments of Internet connectivity.

1.2 Objective

The objective of this project is to demonstrate the value of using Cyber-Foraging architectural tactics (3) for the design and development of an agricultural knowledge exchange system to be used in a resource-challenged region.

1.3 Research Question

1.3.1 Main Research Question

What cyber-foraging architectural tactics can be used to develop an agricultural knowledge exchange system to be used in a resource-challenged region?

1.3.2 Subquestions

RQ1 What are the usage scenarios for an agricultural knowledge exchange system to be used in a resource-challenged region?
RQ2 Which of the proposed cyber-foraging architectural tactics can be used in the development of the system and how do they map to the system requirements?
RQ3 What system architecture and design follows from using the selected tactics?
RQ4 Does the developed system based on these tactics meet all its functional and non-functional requirements?

1.4 Research Method

The project will be divided into two main parts.

Subquestions RQ1, RQ2 and RQ3 will be answered by the creation of a requirements and design document for a Weather Information system aimed at users in developing regions that makes use of architectural tactics for cyber-foraging . This system takes the VOICES (1) projects as inspiration and aims to expand on them, moving the focus from feature phones to smartphones.

Subquestion RQ4 will be answered by implementing a demonstrator of the system that showcases the tactics working in practice.

2

System description

2.1 Outline

The aim of this project is to create a design document as well as a (demo) implementation for a system which makes it possible for people involved in agriculture (for example farmers and NGO employees) who work in environments with little to no access to the internet and electricity, to collect and retrieve data about the weather. The system should be able to perform computations on the collected data as examples of valuable services for the aforementioned stakeholders. An important aspect of the project will be to demonstrate the value of applying cyber-foraging tactics to this kind of system. End users of the system interact with smartphones, the proliferation of which in developing regions is predicted to rise significantly in the coming years (4)(5). The capabilities of the mobile application are extended by surrogates in the form of single board computers running on solar power. To be able to eventually store all collected data in a cloud based back-end, a mobile hub carrying a computer system with increased storage capabilities will connect to each surrogate periodically, and eventually connect to the internet. This also makes it possible to propagate data from the internet to the surrogates and mobile devices. This setup was inspired by the DakNet project in India (6).

Illustration 2.1 shows an overview of the system. The mobile hub (3) is a computer system with networking and storage capabilities (such as a laptop), carried by for example an automobile. It can move from villages that lack access to the internet to a larger city that does have access (1), such that it can store as well as fetch data that can be used for services in the villages. Surrogates (4) are single board computers (like for example Raspberry Pi (7)), extended with network adapters and solar batteries. Mobile devices (2) are smartphones, most of which will be in the low-end range, generally with

computational ability and storage capacity lower than the surrogates. Throughout this document, the surrogate, mobile device and mobile hub components will occasionally be addressed as "system nodes".

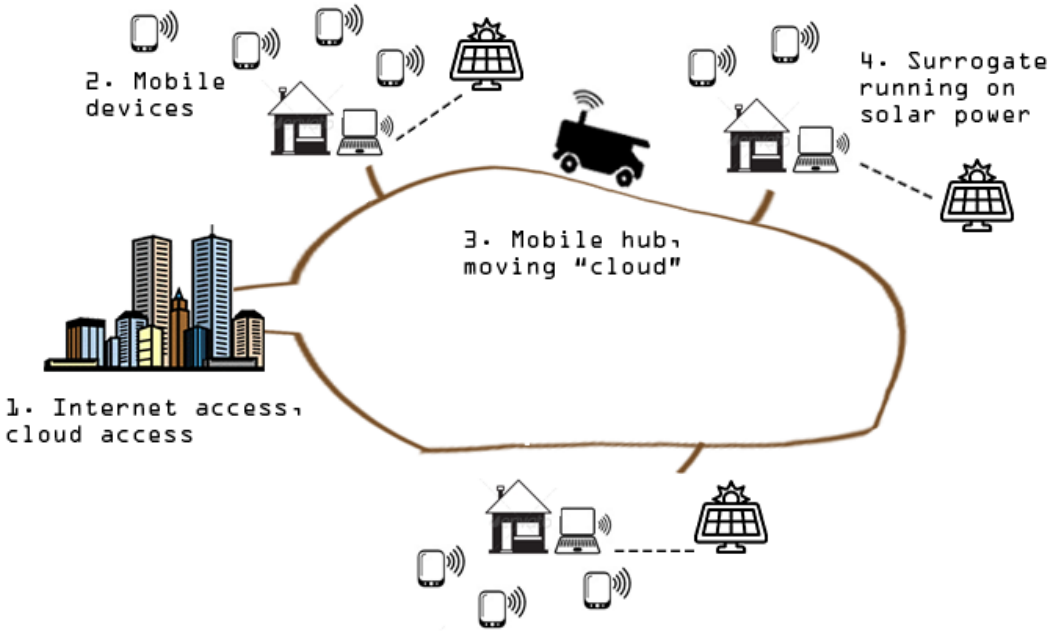

Figure 2.1: Basic Setup

2.2 Usage scenarios

This section contains the descriptions of the usage scenarios for the system. The scenarios are accompanied by activity diagrams depicting the flow of actions performed by the users of the system. Figure 2.2 shows the legend for the activity diagrams used.

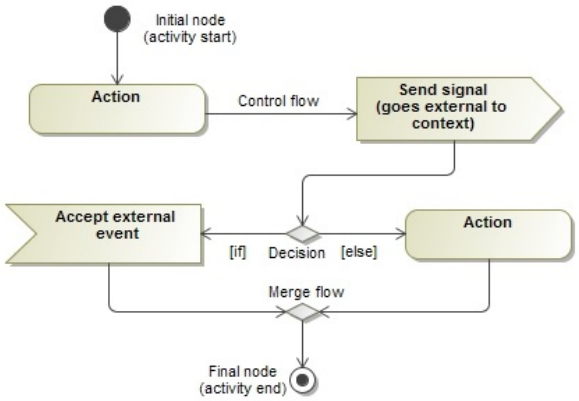

Figure 2.2: Activity Diagram Legend

Scenario S1 - Collect temperature data, surrogate connection available
A farmer wants to save the temperature he just measured in a certain region. He opens the mobile app. Because he has voice support enabled, the icons he can select are described by a pre-recorded voice. He selects the "Enter Data" icon and then selects the "Temperature" field. He enters his measurement and selects the "Submit" button. Figure 2.3 shows an activity diagram for this scenario.

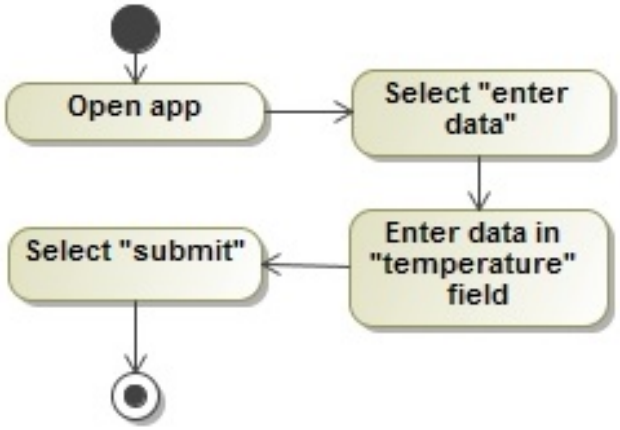

Figure 2.3: Scenario 1 Activity Diagram

Scenario S2.1 - Perform regression on rainfall data (or request prediction), expensive computation, surrogate connection available
An NGO employee wants to perform a regression on all data collected in the last year

(or request a prediction). He selects the "Statistics" icon in the app main menu. In the statistics menu, he selects the "Linear Regression" button (or "Prediction" in the case of a prediction request), enters the variables for his query, and submits it. A message states that this computation may take some time, so he minimizes the app. After a while, a notification becomes visible on his phone, stating that his computation is ready. Upon opening the app, he is able to view a visualization of the regression function and the data points by tapping the notification. Figure 2.4 shows an activity diagram for this scenario.

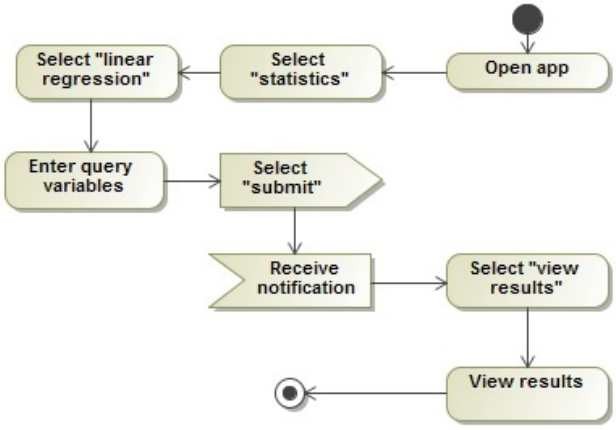

Figure 2.4: Scenario 2.1 Activity Diagram

Scenario S2.2 - Perform regression on rainfall data (or request prediction), expensive computation, no surrogate connection

An NGO employee wants to perform a regression on all data collected in the last year (or request a prediction). He selects the "Statistics" icon in the app main menu. In the statistics menu, he selects the "Linear Regression" button (or "Prediction" in the case of a prediction request), enters the variables for his query, and submits it. A dialog is presented, stating that this computation can only be performed when a connection to a surrogate is available. His inputs are retained on the query input screen. After a while, when he gets closer to a surrogate, an icon indicates that there is a connection to the surrogate. He tries again, and this time the computation is performed and the results are sent to his mobile device. Figure 2.5 shows an activity diagram for this scenario.

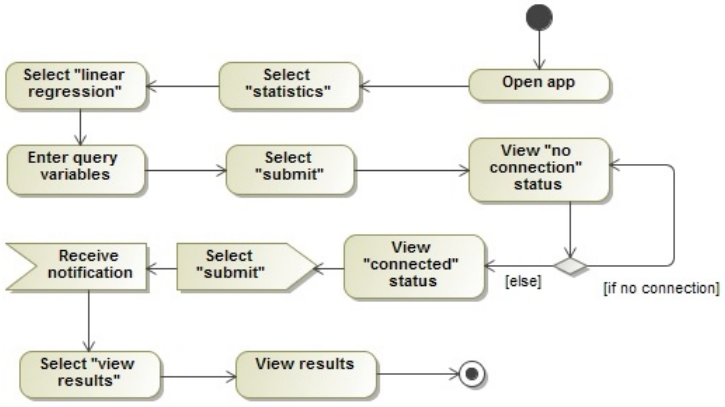

Figure 2.5: Scenario 2.2 Activity Diagram

Scenario S2.3 - Perform regression or prediction on rainfall data (or request prediction), expensive computation, connection breaks during offloaded computation

An NGO employee wants to perform a regression on all data collected in the last year. He selects the "Statistics" icon in the app main menu. In the statistics menu, he selects the "Linear Regression" button (or "Prediction" in the case of a prediction request), enters the variables for his query, and submits it. A message states that this computation may take some time, so he minimizes the app. He then moves out of range of the surrogate signal. A few minutes later, he opens the app again, and is informed that he has gone out of range of the surrogate, and can only receive the results when he connects to the surrogate. He minimizes the app and continues his work. Later that day, he gets closer to the surrogate, and a notification is sent to his phone, stating that his computation is ready. Upon opening the app, he can view a visualization of the regression function and the data points. Figure 2.6 shows an activity diagram for this scenario.

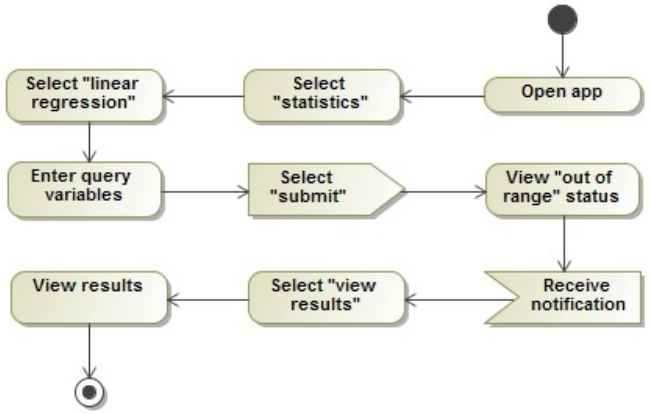

Figure 2.6: Scenario 2.3 Activity Diagram

Scenario S3.1 - Receive new weather forecasts

A farmer receives a notification on his mobile, informing him that new forecasts are available. He opens the app and selects the "Forecasts" icon. A list of dates combined with weather forecast data is shown and can be navigated. He selects tomorrow's date, after which a screen opens with a more detailed description of the forecast. Figure 2.7 shows an activity diagram for this scenario.

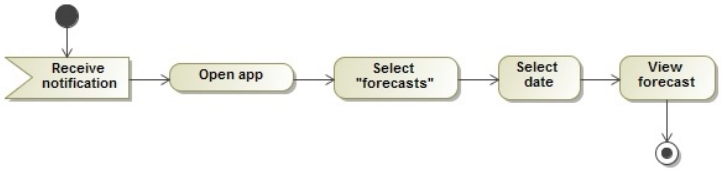

Figure 2.7: Scenario 3.1 Activity Diagram

Scenario S3.2 - Receive new weather forecasts, surrogate crashes during transfer

A farmer receives a notification on his mobile, informing him that new forecasts are available. He opens the app and selects the "Forecasts" icon. The app indicates it is trying to retrieve new forecast data. Eventually, an error message appears on screen, stating that the transfer failed because the surrogate cannot be reached. An icon is shown, to indicate that there is no connection to a surrogate. He presses the "Retry" icon. Because he has received instructions on the usage of the surrogate, after a few more tries he chooses to power the surrogate off and on again. A few minutes later, he

tries to retrieve the forecasts again. A list of dates combined with weather forecast data is shown and can be navigated. Figure 2.8 shows an activity diagram for this scenario.

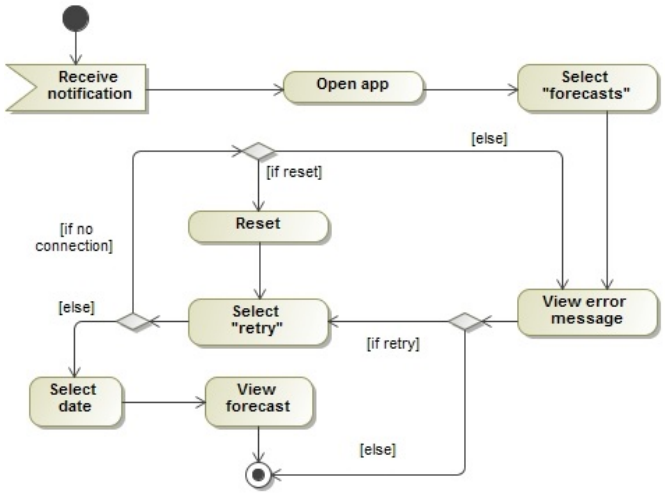

Figure 2.8: Scenario 3.2 Activity Diagram

Scenario S4 - Setting up a surrogate

An NGO employee is going to install a new surrogate in a village. At the NGO office, he powers it on after connecting a keyboard, mouse, and video screen to it. After the surrogate has booted up, a window containing a form opens. In this form, basic information about the village such as name and location can be entered. He enters all required data, after which he clicks the "Save" button. The surrogate now shows a confirmation of the data he entered, and the fact that the surrogate is ready for use. He powers off the surrogate and disconnects all peripherals.

2.9 shows an activity diagram for this scenario.

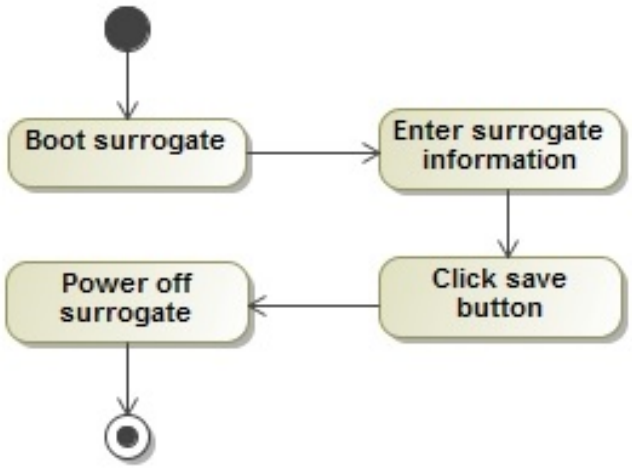

Figure 2.9: Scenario 4 Activity Diagram

Scenario S5 - Mobile hub synchronizes with surrogate
An NGO employee drives into the village with a vehicle that carries the mobile hub. The hub detects the surrogate's broadcast, and sends a request for storage of new data. The surrogate accepts the request and the latest forecasts and data for this region are transferred to it. When this transfer is done, the hub sends a request to receive new data. The surrogate accepts, and transfers newly saved weather data to the mobile hub. Because the mobile hub visits the village only about once a week, this procedure has high priority.

Scenario S6 - Mobile hub connects to the internet
An NGO employee returns the mobile hub to the office, where a wireless internet connection is available. The mobile hub connects to the internet, where it stores the data it has collected on its last roundtrip. Subsequently it pulls data from a weather API, based on the locations of the surrogates that are registered with it.

Scenario S7 - Retrieve stored weather data
A farmer wants to retrieve all weather data saved on the village surrogate last week. He selects the "Get Weather Data" button. He then inputs the start and end dates for his request, after which he selects the "Submit" button. After a few seconds, the requested weather data is shown on screen.
Figure 2.10 shows an activity diagram for this scenario.

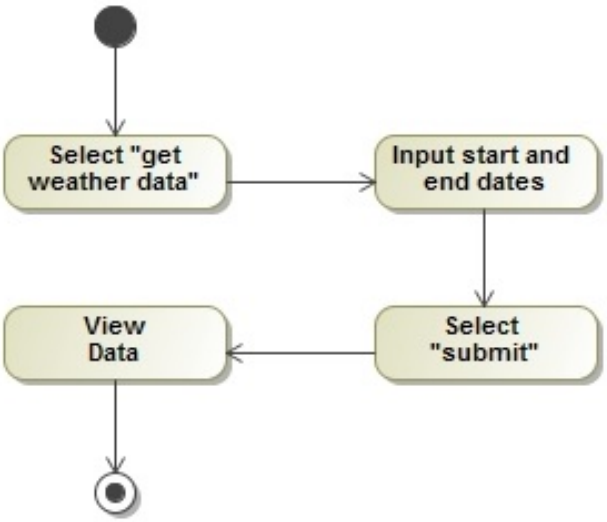

Figure 2.10: Scenario 7 Activity Diagram

Scenario S8 - Surrogate registration at the mobile hub

An NGO employee drives into the village with a vehicle that carries the mobile hub. The hub detects the surrogate's broadcast, and since it is the first time the hub has made contact with the surrogate, it sends a request for information. The surrogate replies with a message containing its unique identifier and location. The mobile hub now registers the surrogate data, adding the surrogate to the system.

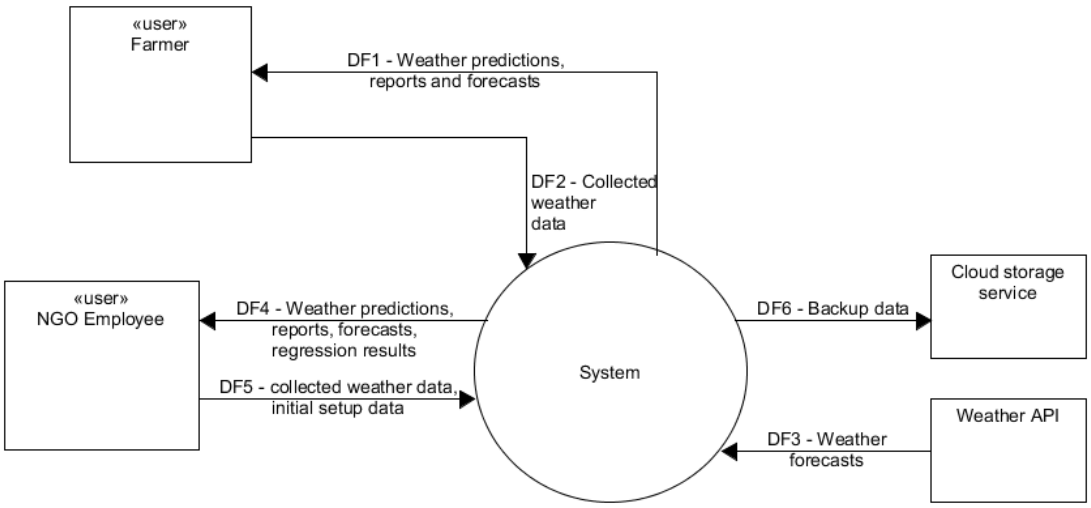

Figure 2.11: System Context Diagram

In the system context diagram (illustration 2.11), a high level overview of the data flows going into and out of the system are shown. Table 2.1 connects the scenarios to these data flows.

	S1	**S2.1**	**S2.2**	**S2.3**	**S3.1**	**S3.2**	**S4**	**S5**	**S6**	**S7**	**S8**
DF1					X	X				X	
DF2	X										
DF3									X		
DF4		X	X	X	X	X				X	
DF5	X						X				
DF6								X			

Table 2.1: mapping data flows to scenarios

2.3 Requirements

In this section, the requirements of the system will be listed. The functional requirements will be listed in table 2.2, non-functional requirements in table 2.3, and constraints, as well as assumptions for the system are shown in table 2.5.

2.3.1 Functional requirements

ID	Name	Description	Related scenarios	Priority
FR1	Store weather data	NGO employees and farmers shall be able to store weather data related to a certain area in the system through a mobile app.	S1	Must have
FR2	Retrieve weather data	NGO employees and farmers shall be able to retrieve weather data related to a certain area on a mobile app. This data is derived from earlier reports (FR1), as well as from a third party weather API accessible through the Internet.	S7	Must have
FR3	Perform regression on weather data	NGO employees shall be able to select a weather information dataset and perform regression on it through the mobile app. A visualization of the results will be available to the user afterwards.	S2.1, S2.2, S2.3	Must have
FR4	Predict future weather data values	NGO employees and farmers shall be able to receive predictions of future values of variables related to the weather. The predictions will run up to a week in the future.	S2.1, S2.2, S2.3	Must have
FR5	Surrogate setup	Surrogates will support the mobile devices used in the system. These surrogates are tied to a certain region and as such need a setup procedure that enables NGO employees to enter the correct settings before using it.	S4	Must have
FR6	Forecast delivery	Weather forecasts related to the region the user is in will be made available to retrieve at the mobile app.	S3.1, S3.2	Must have
FR7	Integration with cloud based storage systems	The system shall store all collected data in a cloud based system such as ERS (8).	S6	Should have

Continued on next page

Continued from previous page

ID	Name	Description	Related scenarios	Priority
FR8	Voice interface	The user interface thst farmers will interact with shall be supported by voice instructions to help users navigate through the app.	S1	Should have
FR9	Synchronize weather data	Periodically, the latest weather forecasts and data for relevant regions are retrieved from a third party weather API on the Internet. These will be stored at the surrogates eventually.	S5, S6	Must have
FR10	Surrogate registration at mobile hub	Whenever new surrogates are added to the system and are operational, their identification and location information (as provided in FR5), should be stored on the mobile hub. This way, the mobile hub can start collecting relevant data for this surrogate (FR9).	S8	Must have

Table 2.2: Functional requirements

2.3.2 Non-functional requirements

ID	Name	Description	Related scenarios	Priority
NFR1	Fault tolerance and reliability	The system should be able to recover from failures such as crashes and loss of connection between mobile devices and surrogates. In the regions where the system will be used, there is expected to be a low number of persons who are proficient in IT. -Surrogates should be able to detect failures in the services they offer and restart them accordingly. -Losing connection during interaction between surrogates and mobile hubs as well as surrogates and mobile devices must not cause the services running on the surrogates to stop functioning. -It is expected that mobile app users will regularly be moving in and out of range of surrogates during use of the system. This should not cause users to lose results of completed computations, or lose data they have stored on the mobile app. - Illustration 2.12 shows a basic overview of the data flows between the three main hardware elements of the system. Each of these interactions is initiated by a request (which is not modeled). Table 2.4 shows when data should be cached, and what should happen in other cases.	S2.2, S2.3, S3.2, S7	Must have

Continued on next page

Continued from previous page

ID	Name	Description	Related scenarios	Priority
NFR2	Easy deployment	The system should be easy to deploy, meaning that: -The mobile app can be installed through an app store and does not have to be configured. It should detect and connect to surrogates automatically. -Surrogates have to be configured locally (FR5), and this process should be able to be performed by NGO personnel with only basic knowledge of information technology. It should be a simple process, comparable to entering data in a form and confirming. -Active surrogates should register to the mobile hub automatically on first connection.	S4, S8	Must have
NFR3	Usability	Literacy among users of mobile devices will vary. Most end users will have low technical knowledge as well. The interfaces to the functionality they will use should be understandable to them. -Text in English, including voice explanations. -Text in French, including voice explanations (one of the target languages, but will not be implemented in the demonstrator).	S1, S2.1, S2.2, S2.3, S3.1, S3.2, S7	Should have

Continued on next page

Continued from previous page

ID	Name	Description	Related scenarios	Priority
NFR4	Extensibility	Developing new functionality and adding it to the system should be supported and made easy. -A standard format for services that perform either computation offload or data staging should be available to future developers, including documentation and an example.		Should have
NFR5	Energy efficiency	The mobile device and surrogate systems will run in an energy challenged environment. Access to energy is limited and not always available. -Energy use on mobile devices should be minimized. -Energy use on surrogates should be minimized, energy efficiency for mobile devices should have priority however.	S2.1, S2.2, S2.3	Must have
NFR6	Capacity	Mobile phones have low storage capacity, so storage should for the most part be the responsibility of the surrogates and mobile hubs. The surrogate should be able to provide offloading and data staging capabilities to multiple users at the same time. -Storage used on phones should be kept under 100 MB, not counting results for calculations that the user has saved. -Surrogates should be able to run 10 instances of services at the same time.	S1, S3.1, S3.2	Could have

Continued on next page

Continued from previous page

ID	Name	Description	Related scenarios	Priority
NFR7	Availability	Capabilities provided by surrogates should in principle be available 24 hours a day. However, because surrogates will run on solar energy, it is expected that they can run out of energy during heavy use, especially during periods with no or little sunshine. -Every 24 hour period, the surrogate should be able to deliver services amounting to 4 hours of surrogate activity. This does not give guarantees about availability loss due to crashes (which are discussed in NFR1 and NFR9). -When energy availability drops below 10% of the battery's capacity, computations that will take longer than 5 minutes should be queued until the battery is charged up to above 15% again.	S1, S2.1, S2.2, S2.3, S3.1, S3.2, S7	Should have

Continued on next page

Continued from previous page

ID	Name	Description	Related scenarios	Priority
NFR8	Performance	There are no strong performance requirements, except for the transfer of data between mobile hub and surrogate. This is because the window during which there is opportunity to interchange data is short and infrequent. -The transfer of data between mobile hub and surrogate should be prioritized over other offloaded computations or data staging operations the surrogate is performing. The only operation with higher priority is the registration of a new surrogate. -The mobile hub should check for a surrogate broadcast signal at least 10 times per second, as long as it is not interacting with one already. -The surrogate should broadcast it's presence at least once per minute.	S5	Should have
NFR9	Recovery	-When a surrogate has crashed, resetting the hardware should get it operational again within 10 minutes. -When a mobile hub has crashed, resetting the hardware should get it operational again within 10 minutes.	S3.2	Must have
NFR10	Data integrity	-When weather data is entered on the mobile app, it should be checked for valid values. Example: temperature values between certain limits. -The same applies to setup data during the setup process.	S1	Could have

Table 2.3: Non-functional requirements

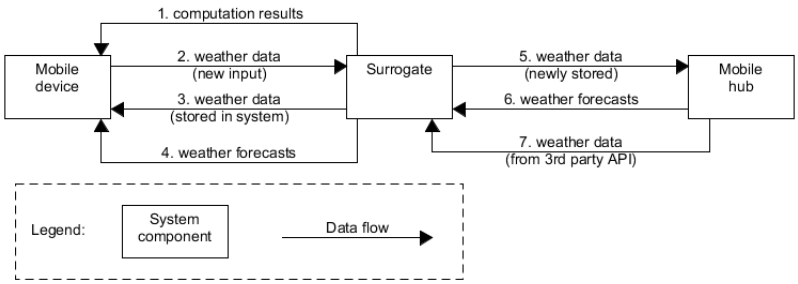

Figure 2.12: Data Flows Between System Nodes

Data flow	Initiator	Connection loss during request, or no connection available	Connection loss during operation
1	Mobile device	Retrieving results for computations happens periodically and is invisible to the user. If retrieval fails, the app will simply retry at a later time.	Request is cached at surrogate. Retry on reconnection.
2	Mobile device	Data is cached at mobile. Retry on reconnection.	Data is cached at mobile. Retry on reconnection.
3	Mobile device	Inform user of failure/lack of connection, user can retry. Inputs are retained for reuse.	Inform user of failure, user can retry. Inputs are retained for reuse.
4	Mobile device	Inform user of failure/lack of connection, user can retry. Inputs are retained for reuse.	Inform user of failure, user can retry. Inputs are retained for reuse.
5	Mobile hub	Data is permanently stored at surrogate. Retry on reconnection.	Data is permanently stored at surrogate. Retry on reconnection.
6	Mobile hub	Data is cached at mobile hub. Retry on reconnection.	Data is cached at mobile hub. Retry on reconnection.
7	Mobile hub	Data is cached at mobile hub. Retry on reconnection.	Data is cached at mobile hub. Retry on reconnection.

Table 2.4: Connection loss

2.3.3 Constraints and assumptions

ID	Name	Description
C1	Use of cyber-foraging tactics	One of the main purposes of this demonstrator is to showcase and test the usefulness of cyber-foraging tactics for an information system in resource challenged environments. Therefore, a range of these tactics as described in (3) should be applied, based on their usefulness in realizing the requirements of this system.
C2	Low cost infrastructure and hardware	End-users will mostly use low-end mobile devices, while the rest of the system will be deployed on hardware locally, for which the cost should be as low as possible.
C3	Use of FIrefoxOS	FirefoxOS is the mobile OS of choice for the developers. It is open source, based on standard Web API's, and targeted at low end smartphones and developing markets.
C4	Use open standards	Preferred use of open source components and open standards where possible.
A1	Concurrent access to multiple surrogates	It is assumed that the surrogate signals do not overlap as there is a maximum of one surrogate per village. This means the mobile devices and mobile hub can connect to different surrogates, but never at the same time.

Table 2.5: Constraints and assumptions

2.3.4 Mapping requirements to tactics

In table 2.6, a mapping from requirements to tactics that will be used to realize them is shown. The first four tactics are prerequisites for other tactics and as such are used in most cases.

	6.1 Computation offload	6.2 Data staging	6.2.1 Pre-fetching	6.3 Surrogate provisioning	6.3.1 Preprovisioned surrogate	6.4 Surrogate discovery	6.4.3 Surrogate broadcast	7.2 Fault tolerance	7.2.3 Cached results	7.3 Scalability and elasticity	7.3.1 Just-in-Time containers
FR1		X		X	X	X	X	X	X	X	
FR2		X	X	X	X	X	X			X	
FR3	X	X		X	X	X	X	X	X	X	X
FR4	X	X		X	X	X	X	X	X	X	X
FR5				X	X						
FR6		X	X	X	X	X	X	X			
FR7		X		X	X	X	X	X		X	
FR8											
FR9		X	X	X	X	X	X	X		X	
FR10				X	X	X	X				
NFR1								X	X		
NFR2				X	X	X	X				
NFR3											
NFR4											
NFR5	X			X	X	X	X			X	X
NFR6	X			X	X	X	X			X	X
NFR7	X	X	X	X	X	X	X	X	X		
NFR8	X			X	X	X	X				
NFR9											
NFR10											

Table 2.6: Tactic mappings

6.1 Computation offload

Regression and extrapolation (FR3 and FR4) will be initiated by the user on the mobile device, but the computations will be offloaded to the surrogate. Data sets on which these operations are performed will be located at the surrogates and can be reasonably large, while input for the operations are small sets of variables of simple data types. The view taken will therefore be that the surrogate offers services which are consumed by the mobile device, making the offload take place at the service level, with a payload consisting of parameters. Furthermore, offloading energy-intensive computations to the surrogate is the main method to minimize energy consumption on the mobile device (NFR5). Lastly, by offloading from low-end mobile devices to surrogates with more computational capacity, performance with regards to those calculations is improved (NFR8), even though there are no strong performance requirements on performing calculations.

6.2 Data staging and 6.2.1 Pre fetching

Data collected on mobile devices (FR1) will be stored locally until it has been successfully transferred to a surrogate. The surrogates will store this data indefinitely, both to make it accessible to mobile users in the future, but also to make it available to the mobile hub, which will collect all data eventually. This data will not be saved on the mobile device after it has been successfully transferred to the surrogate, since storage is limited for phones. This means mobile devices requesting this data (FR2) will always get it from a surrogate where this data is staged, unless it has been explicitly saved on the mobile device by the user. The same is true for additional data retrieved from the Internet (FR9) and forecasts (FR6), which are pre-fetched from third party API's and distributed to the correct surrogates, all based on surrogate locations. Availability (NFR7) of data for each region is thus realized by making sure all collected data is stored at the related surrogate, and capacity constraints (NFR6) are met because this minimizes storage space used on mobile devices. The mobile hub will eventually be able to store new data that was entered on the mobile device in the cloud when it connects to the Internet (FR7). This is another instance of data staging, where data is first staged at the surrogates, then at the mobile hub itself.

6.3 Surrogate provisioning and 6.3.1 Pre-provisioned surrogate

As noted, these tactics are basic prerequisites to set the system up as a cyber-foraging system. All required functionality will be available on the surrogate from the start. They will all use the same OS image (for example: Raspberry Pi with cloned SD card),

because they should all provide the same functionality. The only initial difference between surrogates will be the location and identification settings as provided during the setup procedure (FR5).

6.4 Surrogate discovery and 6.4.3 Surrogate broadcast
As noted, these tactics are basic prerequisites to set the system up as a cyber-foraging system. Mobile device users should be able to make use of system functionality as soon as they install the app and get in range of a surrogate (NFR2). To increase the ease of deployment, surrogates broadcast their presence and mobile devices in need of surrogate services can pick up on these broadcasts. The surrogate broadcasting its presence is also vital to the automatic registration of newly deployed surrogates at the mobile hub as soon as they get in communication range (FR10). Lastly, because the opportunities for interaction between surrogates and the mobile hub are scarce, both the surrogate broadcasting its presence continuously and the mobile hub continuously trying to discover surrogates are vital to the system's performance (NFR8).

7.2 Fault tolerance and 7.2.3 Cached results
Each of the interactions between system nodes is susceptible to loss of connection. This means a tactic for fault tolerance will have to be used. Illustration 2.12 and table 2.4 clarify what should happen when connection losses occur during different interactions (see also NFR1). When computation offload (FR3, FR4) has been correctly initiated, but the mobile user moves out of range of the surrogate during the computation, results should be cached so they can be sent to the user as soon as the mobile device connects to the surrogate again. Caching is used here to realize availability (NFR7). When entering weather data (FR1) without an available connection, the data is cached on the mobile device, which will periodically try to resend the data. In this case, caching is used to prevent breaking up the user experience and enable users to keep on working with the app, saving new readings and not having to worry about the data being saved correctly.

7.3 Scalability and elasticity and 7.3.1 Just-in-time containers
FR3 and FR4 have in common that they offload large computations that will be used infrequently. Therefore, as opposed to the other services offered by surrogates, these services are better suited to run in their own containers, such that small operations will not get queued behind these large computations. To be able to handle multiple computation offload requests at the same time, as well as to not let these (large) computations cause small data transfers to have to wait for them (see also NFR6, capacity), each

time a request for a computation offload is received at the surrogate, a container with the necessary functionality is created. The mobile app (or mobile hub) then interacts directly with this container to consume the service. As requests for computation offload will be infrequent, often with long periods of time between them, only creating containers for their respective capabilities when they are needed is a tactic that will save energy (NFR5). During periods of inactivity, only a gateway process has to run.

3

Architecture and design

3.1 System components

The component diagram for the system is shown in illustration 3.1. The components were initially derived from the tactics that have been chosen in the previous section. The mappings will be discussed in 3.3. Components for system requirements that were not covered by the tactics were added afterwards.

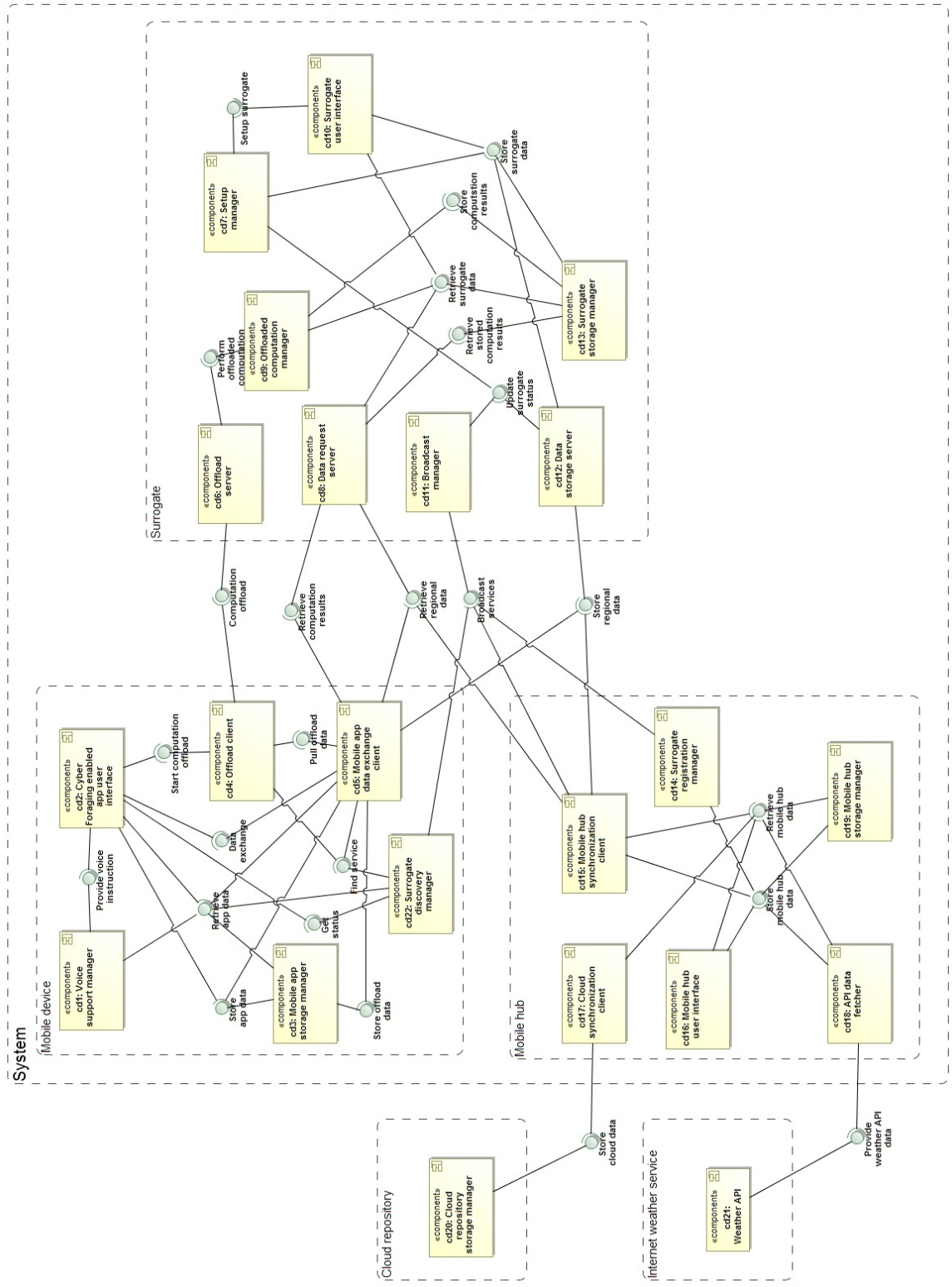

Figure 3.1: System Component Diagram
Note: to avoid clutter, high level components have been visualized as dashed line boxes

A mapping of the functional requirements to the components is provided in table 3.1. The system consists of the following components:

3.2 Mobile app components

CD1: Voice Support Manager
This component handles the voice snippets that support the user interface.
CD2: Cyber-foraging Enabled App User Interface
Handles user input and output of responses.
CD3: Mobile App Storage Manager
Manages storage of all permanent data and user settings on the mobile app except for data that is being staged before moving to the surrogate. Storing and retrieving data is done through its interfaces, *Store app data* and *Retrieve app data*.
CD4: Offload Client
This component handles computation offload from the mobile app to the surrogate, initiated through CD2.
CD5: Mobile App Data Exchange Client
This component handles staging data and transferring it from the mobile app to the surrogate after it has been entered through CD2. It also handles requesting and receiving data from the surrogate.
CD22: Surrogate Discovery Manager
Tasked with finding available surrogate services.

3.2.1 Surrogate components

CD6: Offload Server
Handles requests for computation offload from mobile devices at the surrogate.
CD7: Setup Manager
Implements the setup process for newly deployed surrogates. Offers the interface *Setup surrogate*, which is used by CD10 when the setup process is started.
CD8: Data Request Server
Handles requests for data stored on the surrogate from mobile devices as well as from the mobile hub.
CD9: Offloaded Computation Manager
This component creates containers that run offloaded computations and makes sure results are consequently stored at the CD13 component.

CD10: Surrogate User Interface
Handles user input and output of responses, used when a screen and mouse/keyboard are connected to the surrogate (for example during the setup process, or to check console output).
CD11: Broadcast Manager
Broadcasts the presence of the surrogate and its capabilities, through the interface *Broadcast services*. Vital for all requirements in which interaction between the surrogate and other system nodes is involved.
CD12: Data Storage Server
Handles requests for storing data on the surrogate from mobile devices as well as the hub.
CD13: Surrogate Storage Manager
Manages storage of all permanent data, computation results and settings on the surrogate. The interfaces for data retrieval include the possibility to have the data removed after a successful transmission, such that computation results and data that is in transit can be removed correctly.

3.2.2 Mobile hub components

CD14: Surrogate Registration Manager
This component handles the registration of surrogates that are new to the system. It does this by picking up broadcasts from the CD11 component and storing new surrogate data at CD19.
CD15: Mobile Hub Synchronization Client
Manages synchronization of data between the mobile hub and the surrogate.
CD16: Mobile Hub User Interface
Basic user input and output functionality for the hub is handled here.
CD17: Cloud Synchronization Client
This component makes sure data stored in the system is backed up to a cloud repository, works with CD20.
CD18: API Data Fetcher
This component retrieves weather data from a third party API and stores it on the mobile hub through component CD19. It also periodically checks whether the surrogate list stored by this component has new entries.
CD19: Mobile Hub Storage Manager

Handles storage and retrieval of data on the mobile hub. This includes settings, staged as well as permanent weather data and the list of known surrogates.

3.2.3 Components implemented on third party systems

CD20: Cloud Repository Storage Manager
Combined with CD17, this component makes sure data stored in the system is backed up to a cloud repository.
CD21: Weather API
This component provides weather data and forecasts through a (REST) interface.

	CD1	CD2	CD3	CD4	CD5	CD6	CD7	CD8	CD9	CD10	CD11	CD12	CD13	CD14	CD15	CD16	CD17	CD18	CD19	CD20	CD21	CD22
FR1		X			X			X			X	X	X		X				X			X
FR2		X	X					X			X		X		X			X			X	X
FR3		X	X	X		X		X	X		X		X									X
FR4		X	X	X		X		X	X		X		X									X
FR5							X			X			X									
FR6		X	X					X			X	X	X		X			X	X		X	X
FR7											X				X		X		X	X		
FR8	X	X	X																			
FR9											X	X	X		X			X	X		X	
FR10											X			X								
NFR1			X	X	X	X	X	X	X	X	X	X	X	X	X	X						X
NFR2							X			X	X			X								X
NFR3	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X			X
NFR4																						
NFR5	X	X	X	X	X	X	X	X	X	X	X	X	X									X
NFR6			X			X		X	X			X										
NFR7									X													
NFR8											X			X	X							
NFR9						X	X	X	X	X	X	X	X	X	X	X	X	X	X			
NFR10		X					X															

Table 3.1: mapping components to requirements

3.3 Mapping tactics to components

The basis for the system component diagram were components derived from the tactics selected from (3). In this section, a mapping of the tactics to the system components will be provided, showing how the tactics will be realized in the system. In the resulting models, stereotypes are used to show how the components map to the components from the original tactic descriptions.

Tactic 6.1 - Computation Offload

Figure 3.2 shows the model based on the computation offload tactic, described in (3, pp. 171).

The offload is done at the parameter level, and as opposed to the original model, the "Execute" call is implicit in the offload of the computation. The surrogate will perform the computation as soon as possible after receiving the parameters. Another difference from the original model is the absence of app metadata, which are not necessary for this application's offload functionality.

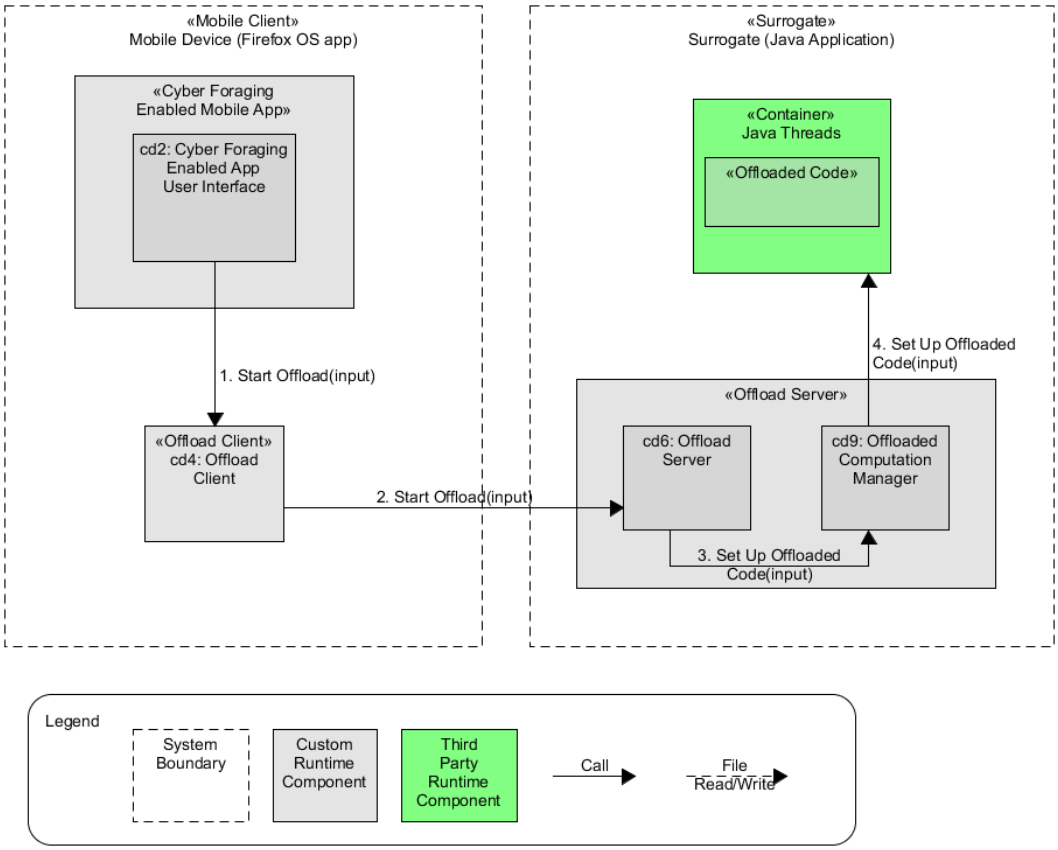

Figure 3.2: Components Realizing Tactic 6.1

Tactic 6.2 - Data Staging and 6.2.1 - Pre-Fetching

Figure 3.3 shows the model based on the data staging and pre-fetching tactics. The model focuses specifically on retrieving forecasts from the Weather API. The tactics are described and modeled in (3, pp. 172).

Since this data will have to move through the mobile hub as well as the surrogate before reaching the mobile device, the component with the "Data Staging Manager" stereotype covers subcomponents from both the surrogate as well as the hub. The data push from hub to surrogate (4 in the model) is modeled as asynchronous since it can only take place when mobile hub and surrogate connect, and there is no guarantee when this will happen.

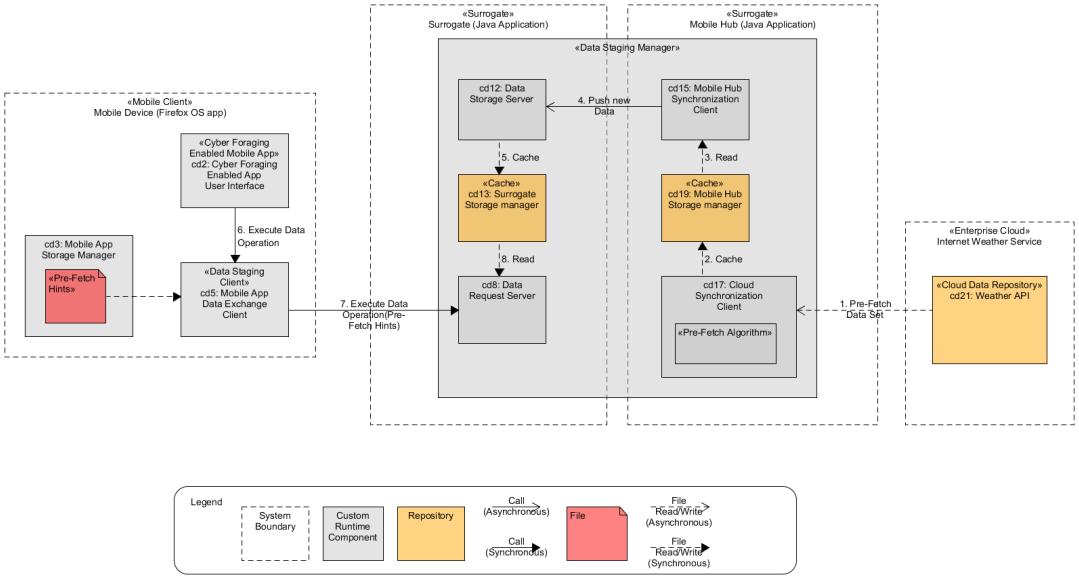

Figure 3.3: Components Realizing Tactics 6.2 and 6.2.1

Tactic 6.3 - Surrogate Provisioning and 6.3.1 - Pre-Provisioned Surrogate
Figure 3.4 shows the model based on the surrogate provisioning and pre-provisioned surrogate tactics, as described in (3, pp. 174).
As opposed to the original tactic, in the model, there is no component with the "Remote User Interface" stereotype, as a remote user interface for the surrogate is not part of the system specification. Furthermore, the "Capabilities Repository", "Capability Metadata" and "Capability Registry" are shown as subcomponents of the "Surrogate Manager" component, as they will not be true repositories (in the sense of a database or file). Rather, the data necessary to run the "Data Staging Manager" and "Offload Server" components will be part of the Surrogate Manager, and it will pass this data to them when it initializes them. After all components handling the Cyber-Foraging services have been initialized, the "Surrogate Discovery" component can be initialized, during which the Surrogate Manager can provide it with a list of capabilities based on the services it started earlier.

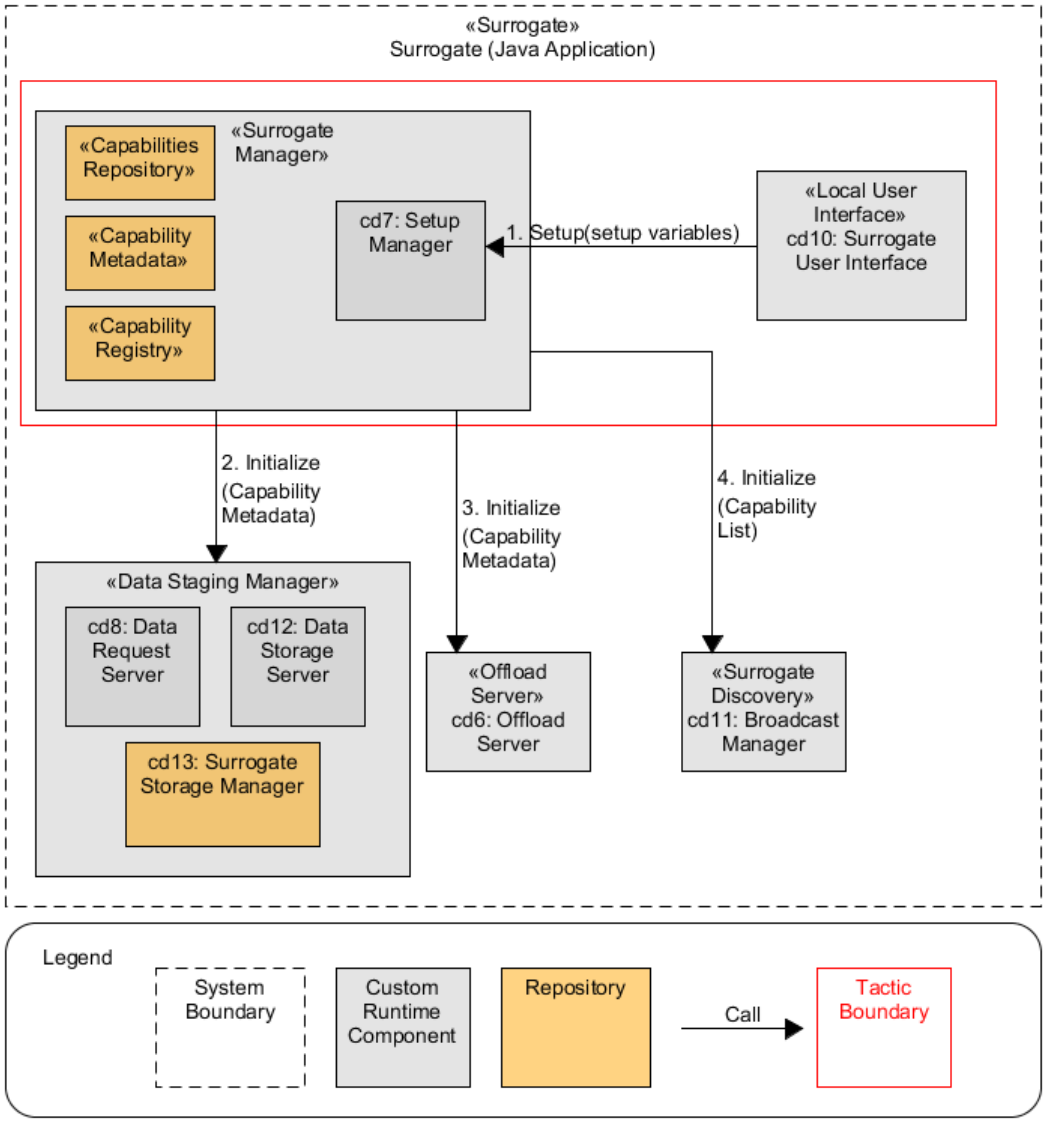

Figure 3.4: Components Realizing Tactics 6.3 and 6.3.1

Tactic 6.4 - Surrogate Discovery and 6.4.3 - Surrogate Broadcast

Figure 3.5 shows the model based on the surrogate discovery tactic as described in (3, pp. 176), and the surrogate broadcast tactic as described in (3, pp. 178).

The call from CD4 to CD22 (2 in the model) returns the relevant connection details if a surrogate is available. It is assumed in the design of the system that there is always either one or no surrogate available. This explains the difference in the number of

"Surrogate" components between this model and the one proposed in the tactics paper.

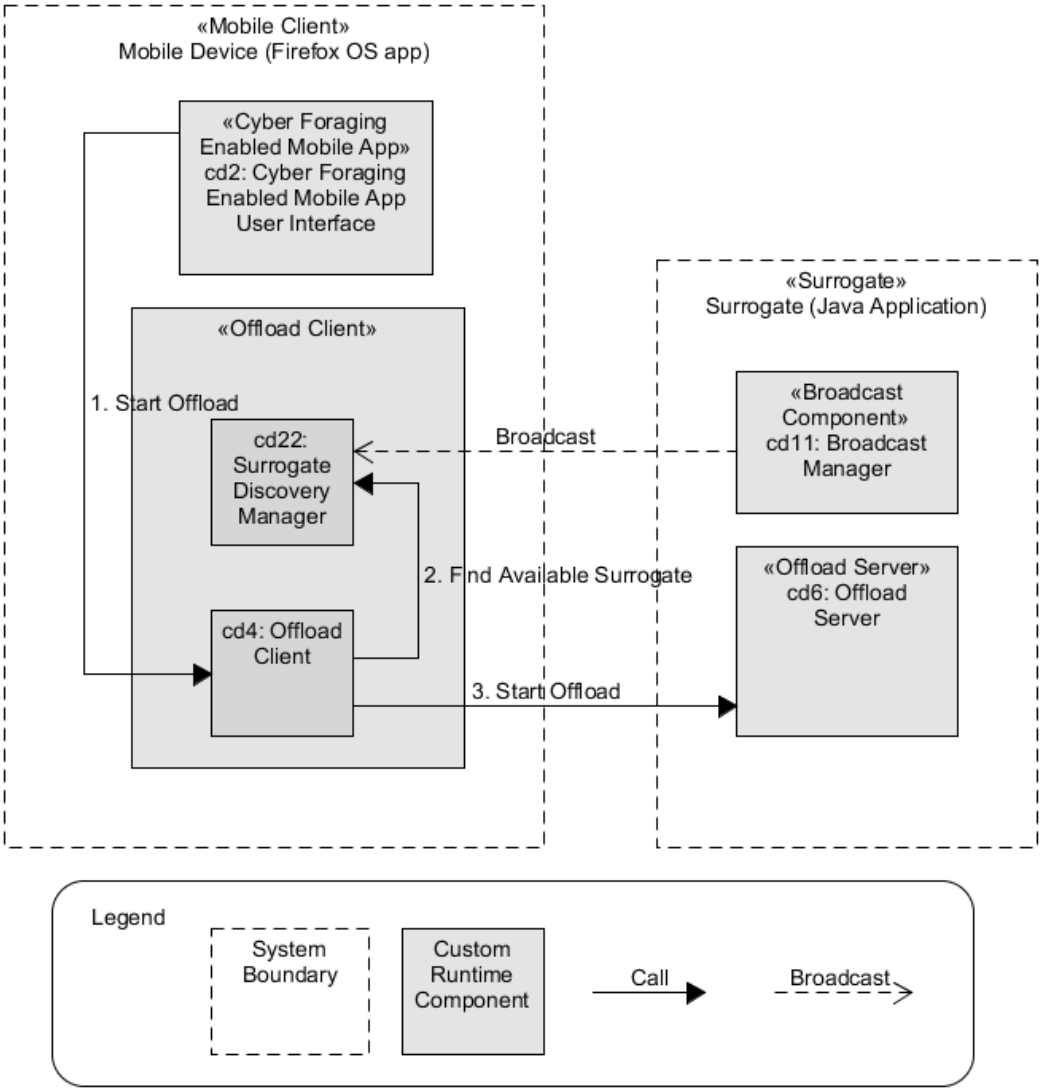

Figure 3.5: Components Realizing Tactics 6.4 and 6.4.3

Tactic 7.2 - Fault Tolerance, 7.2.3 - Cached Results, 7.3 - Scalability/Elasticity and 7.3.1 - Just-in-Time Containers

Figure 3.6 shows the model based on the fault tolerance, cached results, scalability/elasticity and just-in-time containers tactics. They are described in (3, pp. 180–182). In contrast with the model from the paper, the "Results Cache" stereotype/component

is outside of the created container in the form of the surrogate's storage manager. Containers for offloaded code are produced by component CD9 in the form of Java Threads, represented in the model by a call to the container (numbered 6). The creation of containers happens as soon as possible after offload, and the container is discarded after the computation has been completed (call 8) as per the Just-In-Time Containers tactic.

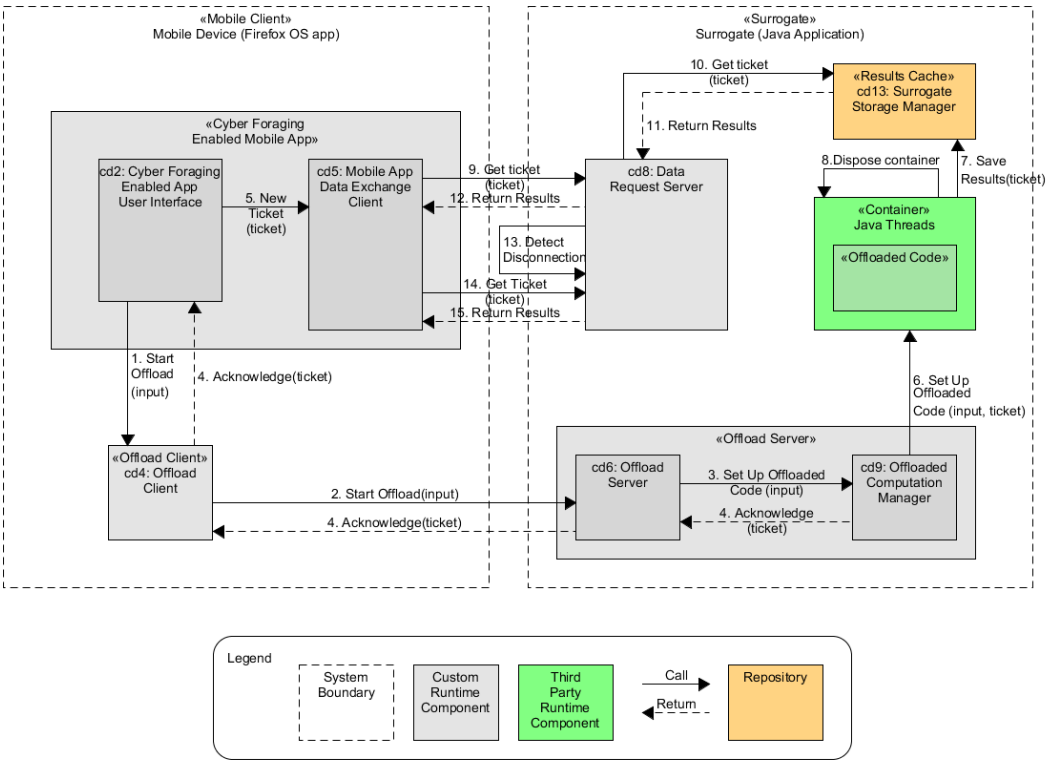

Figure 3.6: Components Realizing Tactic 7.2, 7.2.3, 7.3 and 7.3.1

3.4 Component interaction

In this section, the interaction between components that takes place during the usage scenarios (section 2.2) will be visualized using sequence diagrams. The message flows in the diagrams will be described. Each of the participants belonging to different high level components of the system (Mobile hub, Mobile device, Surrogate) will be colored differently, as can be viewed in figure 3.7.

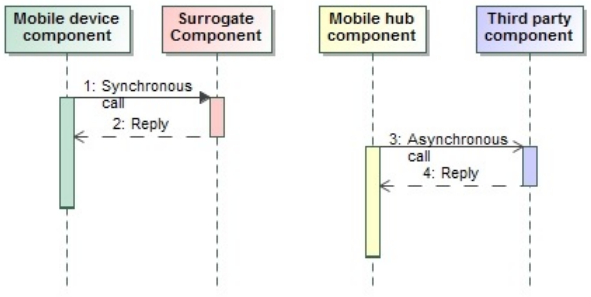

Figure 3.7: Sequence Diagram Legend

Scenario S1

Figure 3.8 shows the sequence diagram related to scenario S1. It is assumed that voice support is turned on, and user input is valid throughout the process.

Upon opening the mobile app, voice clips for menu items are played (1). When the user enters data to be saved, this data is packed in a message and sent to component CD5, tasked with queuing and sending the data to the surrogate (2, 3).

Periodically, this component will try to establish a connection to a surrogate through component CD22 (4, 5, 6). In this scenario, it is assumed that there is a surrogate with the needed service within range, so the connection details of the surrogate are returned to CD5 (7). Component CD5 can now hand off the data to component CD12 (8, 11), which stores the data at the Surrogate Storage Manager (9, 10). If there is no more data to be handed off by the mobile device, a message is sent to the UI to be able to inform the user of this fact (12), and another voice clip playback should be started (13).

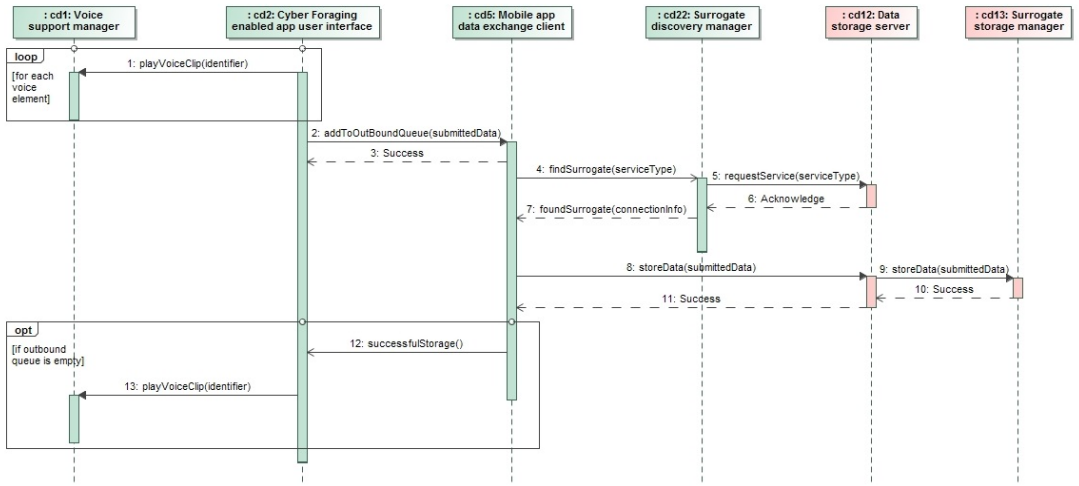

Figure 3.8: Scenario S1 (Collect temperature data, surrogate connection available), Sequence Diagram

Scenario S2.1, **Scenario S2.2**, and **Scenario S2.3**

Figure 3.9 shows the sequence diagram related to scenario S2.1, S2.2 and S2.3. It is assumed voice support is turned off and user input is valid throughout the process.

When the user enters the offload parameters, they are sent to the Offload Client on the mobile app (1, 2). The Discovery Manager is invoked to get the connection details to the surrogate entry point for the service (3). If successful, the connection details are returned (4, 5, 6). In the case that there is no surrogate available at that point, as it is in scenario S2.2, failure messages will be returned (32, 33) so that the user can be informed of this fact. Assuming success, the regression parameters are sent to the Offload Server (7). Assuming the parameters are correct, this component creates a unique identifier (named ticket in the diagram), and forwards the computation request plus ticket to the component that will perform the regression (8, 9). The ticket is also returned to the Offload Client (10), as it is used to retrieve regression results later. The actual computation is performed, after which the results are stored at the Surrogate Storage Manager (11, 12).

On the mobile device, the ticket is added to a queue on the Data Exchange Client (13, 14). This component will periodically try to retrieve outstanding tickets from the Data Request Server, which it finds through the Discovery Manager (15). When there is no connection at that point, as is the case in scenario S2.3, the Data Exchange and UI component are informed (28, 31), and the ticket is stored so it can be retrieved

at a later point in time (29, 30). Alternatively, when a connection is available, the connection details are returned (16, 17, 18). The Exchange Client requests the ticket results from the Request Server, which will in turn check for the ticket results on its Storage Manager. The results are then returned (19, 20, 21, 22). In this case, it is assumed that the regression results have been stored before the mobile app checks for the ticket. Results that have been successfully retrieved are stored on the Mobile App Storage Manager (23, 24), after which the surrogate can be informed and remove the results (25, 26). The user is informed through the UI component after it receives confirmation from the Data Exchange Client (27), and can fetch the results (28, 29).
Figure 3.10 shows an additional sequence diagram relating to scenario S2.2. Since the user has unsuccessfully tried to use a service that requires a connection to a surrogate, the discovery manager will increase the frequency of periodical checks for surrogates. This is done by trying to connect to the surrogate broadcast manager. At some point in time, it finds a broadcast manager (1, 2) and can inform the app UI component that a surrogate is available (3). The user can now be alerted that a surrogate is available through a notification.
Figure 3.11 shows an additional sequence diagram relating to scenario S2.3. In this scenario, at some time after the ticket for an offloaded computation has been received, but before the Data Exchange Client tries to connect to the surrogate to get the results, the user moves out of range and the mobile device cannot connect to the surrogate anymore. This results in the Discovery Manager not being able to find a surrogate. The ticket for the offloaded computation is stored at the Storage Manager, and the Data Exchange component will periodically try to retrieve the list of outstanding tickets (1, 2). As the user has moved into range once more, a connection to the surrogate is realized (3, 4, 5, 6), after which the results for the tickets for this surrogate are retrieved (7, 8, 9, 10), and stored on the mobile device (11, 12). The surrogate is informed that the requested results have been received and can remove the stored results (13, 14). The UI can then inform the user of the newly available data (15).

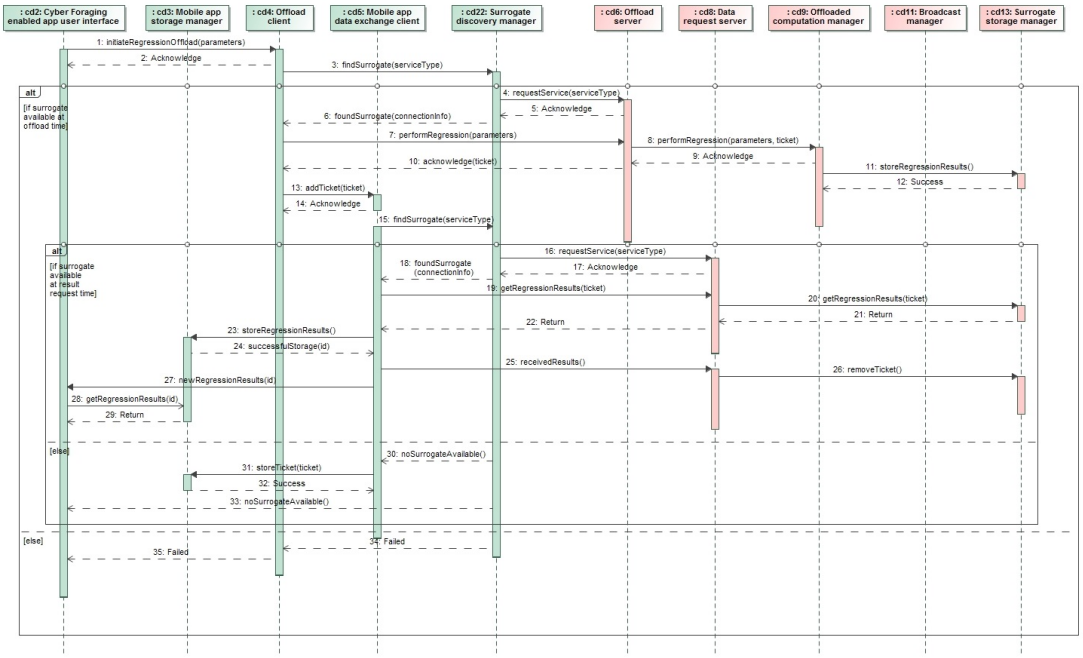

Figure 3.9: Scenario S2.1 (Perform regression on rainfall data, expensive computa- tion, surrogate connection available), S2.2 (Perform regression on rainfall data, expensive computation, no surrogate connection) and S2.3 (Perform regression on rainfall data, expensive computation, connection breaks during offloaded computation), Sequence Diagram

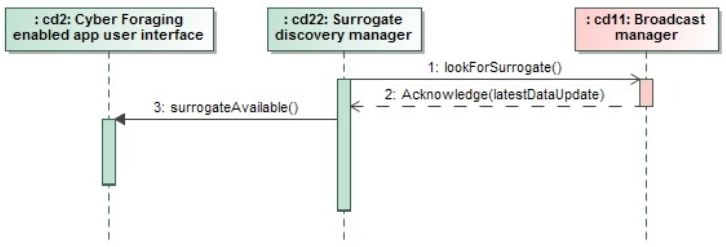

Figure 3.10: Scenario S2.2 (Perform regression on rainfall data, expensive computation, no surrogate connection), Additional Sequence Diagram

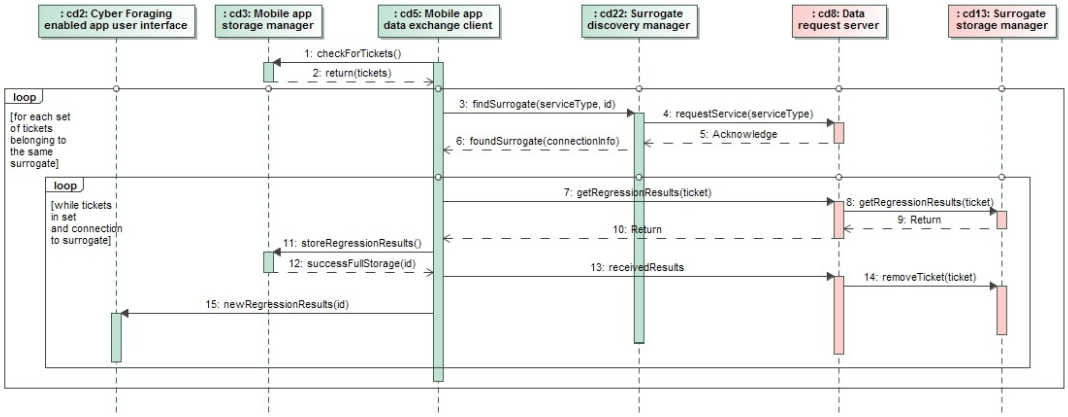

Figure 3.11: Scenario S2.3 (Perform regression on rainfall data, expensive computation, connection breaks during offloaded computation) sequence diagram), Additional Sequence Diagram

Scenario S3.1 and **Scenario S3.2**

Figure 3.12 shows the sequence diagram related to scenarios S3.1 and S3.2. It is assumed voice support is turned off.

Periodically, the Surrogate Discovery Manager checks whether there is a surrogate available (1). In this case, there is, and so the Broadcast Manager returns an acknowledgement (2) containing information about data that could be of interest to the mobile app. In this case, the latest date of the available forecasts is included. The Discovery Manager informs the UI component that there is a surrogate available (3), and checks whether the forecasts have been stored before, which in this case they haven't (4, 5). This also triggers the storage component to remove old forecasts (6). The UI component is informed that there are available forecasts, such that the user can be notified.

After the user sees the message, he selects the forecasts button, causing the UI component to ask the data exchange component to fetch the forecasts (8, 9). The Discovery Manager is used to get the service connection details (10, 11, 12, 13). The Data Exchange Client can now request the new forecast info (14, 15) and this data can consequently be stored through the Storage Manager (16, 17). The UI is informed that new forecasts have been fetched successfully (18), and can now request them to be able to show them to the user (19, 20).

In the case of scenario S3.2, retrieving the forecasts fails and the UI is informed (21), and can now show the "no connection available" icon. After the surrogate has been restarted, the user retries and succeeds (14 through 20).

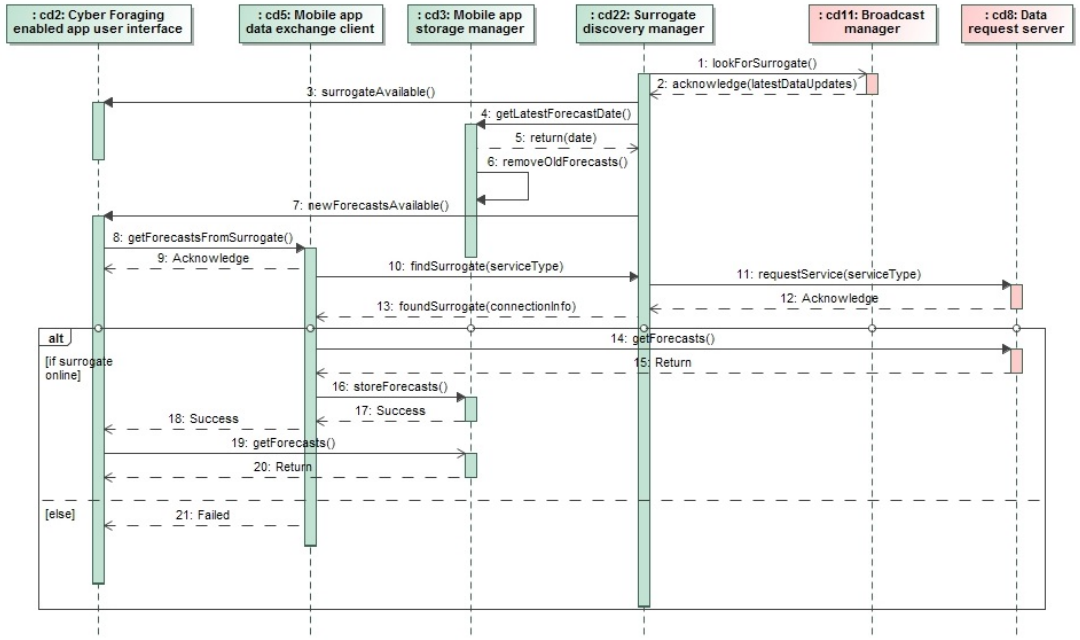

Figure 3.12: Scenario S3.1 (Receive new weather forecasts) and S3.2 (Receive new weather forecasts, surrogate crashes during transfer), Sequence Diagram

Scenario S4

Figure 3.13 shows the sequence diagram related to scenario S4.

As the surrogate is started by the user, the Setup Manager checks whether setup data for the surrogate has been stored before (1). If it has (2), the surrogate can proceed to boot into normal usage mode (which is not modeled here). If no setup data is available (3), the setup process has to be completed before the surrogate can work. Therefore, the setup procedure has to be started on the UI component (4), and the data filled in by the user has to be returned (5), and if it is valid input data, stored (7, 8). If the input is not valid, the user should be alerted so he can try again (6). When storage is successful, the user can be informed that the surrogate is now active through the UI (9).

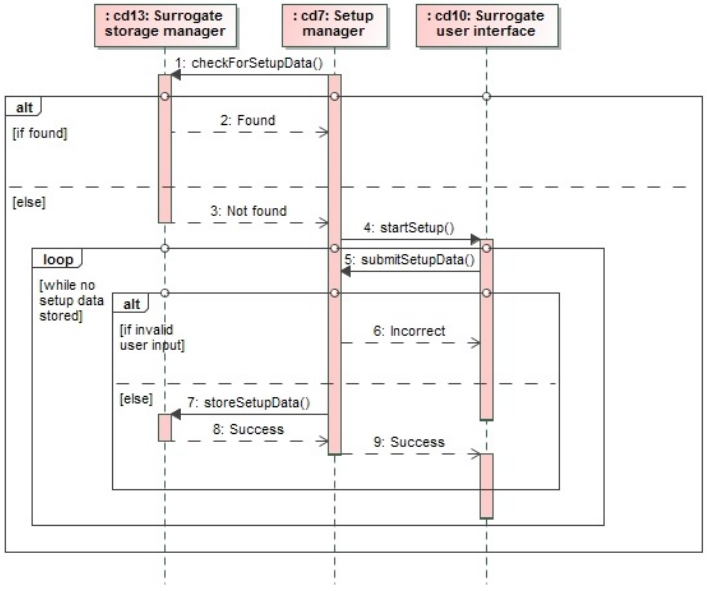

Figure 3.13: Scenario S4 (Setting up a surrogate), Sequence Diagram

Scenario S5

Figure 3.14 shows the sequence diagram related to scenario S5. The mobile hub UI component is left out of this diagram.

The Synchronization Client on the mobile hub picks up the surrogate broadcast (1), and sends a request to store new data to the Storage Server (2). After accepting (3), the data is fetched from storage (4, 5), transferred (6), and has to be stored by the Surrogate Storage Manager (7, 8), before a success message can be returned to the mobile hub (9). When this message is received, the mobile hub sends a request to retrieve new data to the data request server on the surrogate (10). Included in this request is the latest time that the mobile hub received data from the surrogate. The data has to be fetched from the Storage Manager (11, 12), and is eventually returned to the mobile hub (13), where it is stored (14, 15).

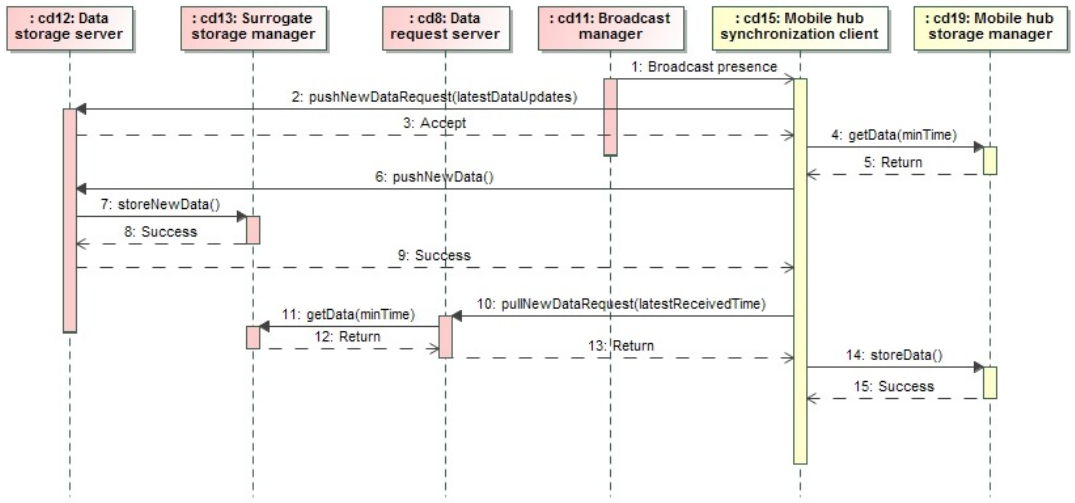

Figure 3.14: Scenario S5 (Mobile hub synchronizes with surrogate), Sequence Diagram

Scenario S6

Figure 3.15 shows the sequence diagram related to scenario S6. The mobile hub UI component is left out of this diagram.

When the mobile hub connects to the Internet, the Cloud Synchronization Client tries to connect to the cloud storage and sends a request for storage (1) which is accepted (2), after which the data to be sent to the cloud storage is fetched (3, 4). It is then successfully stored (5, 6).

The API Data Fetcher requests the latest data from an online weather API (7, 8), which is then stored on the mobile hub as well (9, 10).

Note: the two processes depicted by messages (1) through (6) and and (7) through (10) are asynchronous with respect to each other. They are both triggered on connection to the Internet.

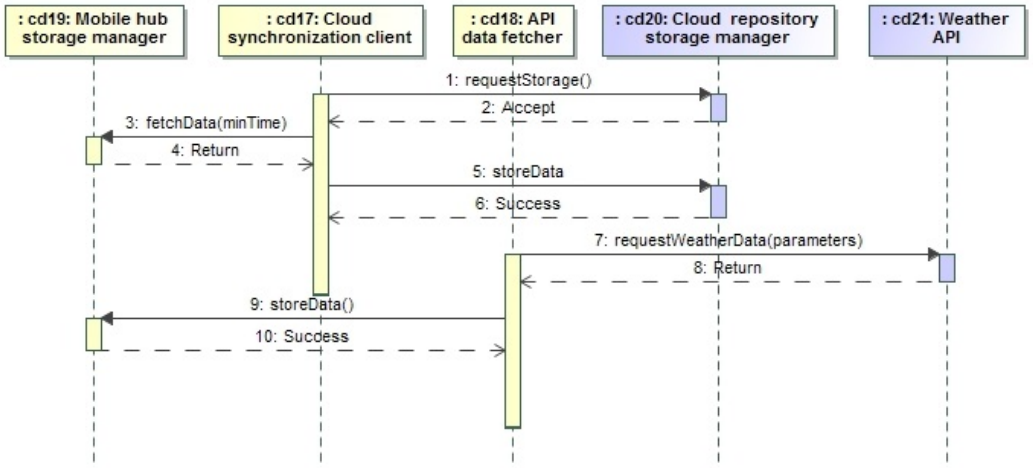

Figure 3.15: Scenario S6 (Mobile hub connects to the Internet), Sequence Diagram

Scenario S7

Figure 3.16 shows the sequence diagram related to scenario S7. It is assumed that the requested data is not saved and a connection to a surrogate is available at all times.
A user requests weather data (1), after which an acknowledgement is shown on screen (2). The Exchange Client requests the Discovery Manager to find a surrogate that provides the service (3). The Discovery manager connects to the surrogate, and returns the active connection to the Exchange Client (4, 5, 6), and the actual data request can now be sent (7). The request is forwarded to the Surrogate Storage Manager (8), and eventually the requested data is returned to the UI so it can be shown to the user (9, 10, 11).

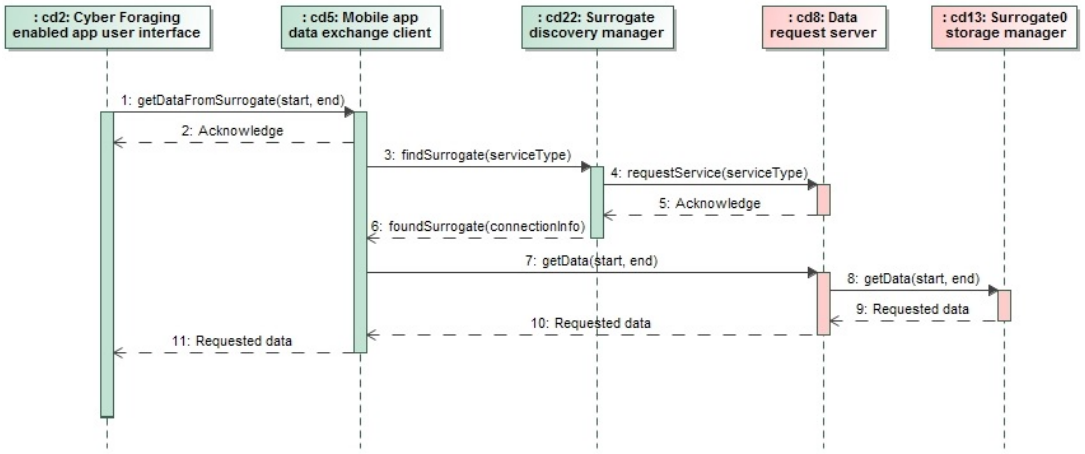

Figure 3.16: Scenario S7 (Retrieve stored weather data), Sequence Diagram

Scenario S8

Figure 3.17 shows the sequence diagram related to scenario S8.
The Surrogate Registration Manager on the mobile hub picks up the surrogate broadcast (1), and sends a request for surrogate information to the Broadcast Manager (2). The reply (3) contains the requested data, which is saved on the mobile hub (4).

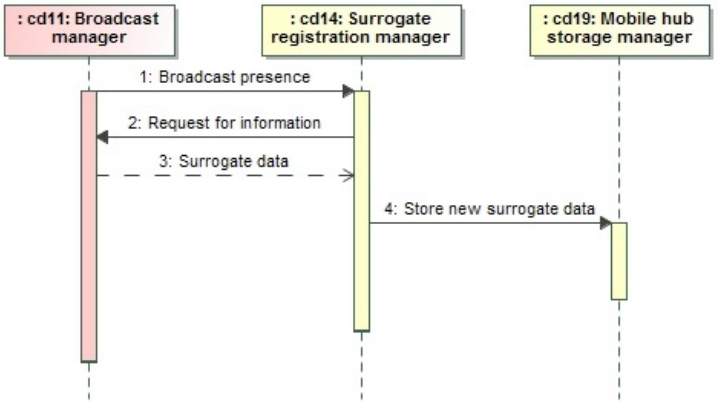

Figure 3.17: Scenario S8 (Retrieve stored weather data), Sequence Diagram

4

Implementation

4.1 Demo implementation details

4.1.1 Introduction

A demo implementation was created under the name AgroTempus. A GitHub Webpage for AgroTempus is available at (9). On this page, the demo project is available as a download. In Appendix A, a user guide is available, as well as installation and setup instructions. Due to time constraints, only part of the system was been implemented. Selection of implemented components was based on implementing as many of the selected tactics as possible, as well as delivering an application with working end-user functionality. This led to the choice of implementing the app and surrogate main components first. Figure 4.1 shows the system component diagram, with the components that have been implemented in the demo in green, and components that have not been implemented in red.

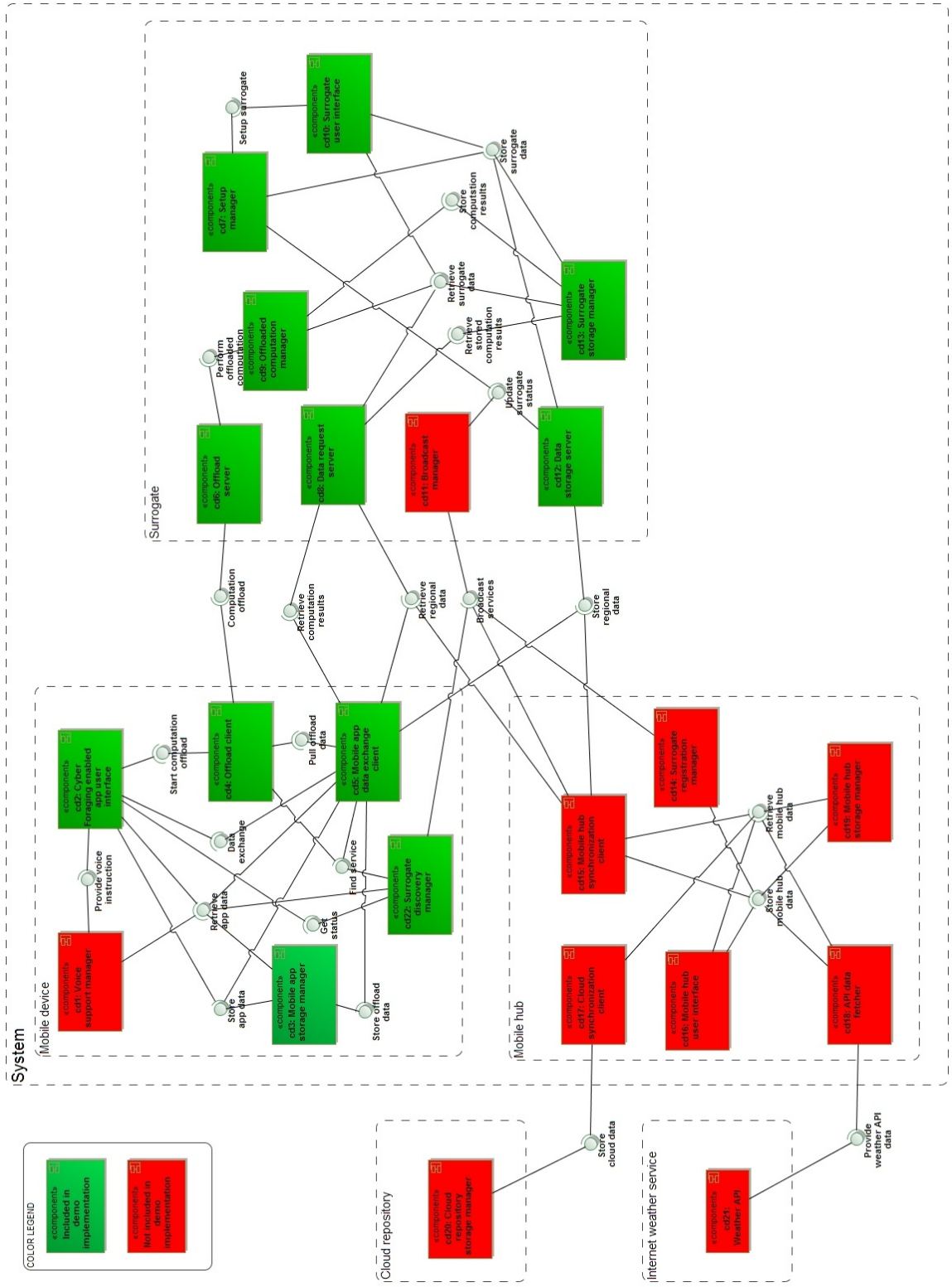

Figure 4.1: System Component Diagram - Implementation overview
Note: to avoid clutter, high level components have been visualized as dashed line boxes

4.1.2 Changes and notes

Change from Surrogate Broadcast to Local Surrogate Directory tactic
At implementation time, no working ad-hoc networking library was found for Firefox OS. Therefore, the *Surrogate broadcast* tactic could not be used for surrogate discovery in the mobile app. It was replaced by the *Local surrogate directory* tactic (tactic 6.4.1 in (3)). A list of surrogates, including connection details, is maintained on the mobile app. This way, whenever a surrogate service is needed, the mobile app can try and connect to each surrogate one by one until it can make a connection to a surrogate that provides the needed capabilities.

No persistent storage
The demo implementation does not yet have a persistent database for the surrogate. This means data on the surrogate is lost when the surrogate software is closed. To support testing of the application, dummy data (forecasts and local weather data is loaded during initialization.
The outgoing queues for computation result requests as well as weather data storage requests on the app are not yet persistently stored.

4.1.3 Technologies used

The mobile app ("Mobile device" in the component diagram 4.1) , covering components CD2, CD3, CD4, and CD22, is a Firefox OS app. This is in line with constraint C3. A Firefox OS app is essentially a Web app, consisting of HTML pages, CSS style sheets and Javascript code. Most of the app logic is written in plain Javascript, with minimal use of the JQuery library (10).
The surrogate, covering components CD6, CD7, CD8, CD9, CD10, CD12 and CD13, was implemented in Java as a multi-threaded application. The component performing regression and prediction (CD9) makes use of the free Java chart library JFreeChart (11), which offers tools to perform regression on data sets as well as to generate plot images to visualize the results in common image formats. The same component also makes use of the Apache Commons Codec (12) to convert generated images into Base64 binary string format.
For communication between components residing on different nodes, JSON (13) was chosen as the standard message and data storage structure. This format is used by (free) weather APIs like OpenWeatherMap (14), and works well with Javascript. To be able to use JSON objects in the surrogate code, the system makes use of the JSON

Simple library (15).
The surrogate software was tested on a Raspberry Pi 2 model B, using a TL-WN722N wireless adapter. The Operating system used was Raspbian, a Linux distribution optimized for Raspberry Pi (7).

4.1.4 Software project structure

On the GitHub Webpage for AgroTempus (9) the project can be downloaded as a packaged file, or cloned. As the demo was developed using the Eclipse IDE (16), the code is contained in an Eclipse project. The project is divided into three subprojects which can be found in the folders with their respective names, *AgroTempus-app*, *AgroTempus-hub*, and *AgroTempus-surrogate* for each of their corresponding components.

App components
CD2 code is contained in *app.js*, *app.html* and *style.css*, which also is the main app program.
CD3 in *storage.js*.
CD4 in *offload.js*.
CD5 in *dataexchange.js*.
CD22 in *discovery.js*.

Surrogate components
The main surrogate program is contained in *Surrogate.java*.
CD6 in *OffloadServer.java*, *OffloadServerWorker.java*.
CD7 in *SetupManager.java*.
CD8 in *RequestServer.java* and *RequestServerWorker.java*.
CD9 in *OffloadComputationManager.java*, *OffloadComputationWorker.java* and *ComputationRequest.java*.
CD10 in *SurrogateUI.java*.
CD12 in *StorageServer.java* and *StorageServerWorker.java*.
CD13 in *StorageManager.java*, *ComputationResultRequest.java*, *RegionalRequest.java* and *ForecastRequest.java*.

4.2 Requirements and tactic implementation

4.2.1 General comments

All of the requirements that involve the surrogate naturally make use of the Surrogate Provisioning and Preprovisioned Surrogate tactics. These will not be considered further in the following discussion of the requirements. As noted before, the Surrogate Discovery tactic has been replaced with the Surrogate Directory tactic in the demo implementation.

4.2.2 Functional requirements

The functional requirements that have been implemented in the demo are FR1 (Store Weather Data), FR2 (Retrieve Weather Data), FR3 (Perform Regression on Weather Data) , FR4 (Predict Future Weather Data Values), FR5 (Surrogate Setup) and FR6 (Forecast Delivery). Unimplemented requirements FR7, FR9 and FR10 all depend on the Mobile Hub component, which was not implemented.
Requirement FR8 (Voice Interface) does not relate to any tactic, and was therefore treated as a low-priority item.
The implementation does cover all selected tactics related to the functional requirements (see table 2.6), except for the Pre-Fetching tactic. Part of the pre-fetching process is simulated by the surrogate by loading dummy data at startup.
Requirements FR1, FR2, FR3, FR4 and FR6 make use of the Data Staging tactic. Out of the model 3.3 derived from this tactic, only the surrogate and app parts were used. The model only covers functionality where data is moving from the surrogate to the app, which was implemented as modeled. Components CD2, CD5, CD12 and CD13 are also used for moving data from the app to the surrogate, which is another instance of making use of data staging which was not explicitly modeled.
Ignoring the requirements related to the mobile hub and the Voice Interface requirement, all six functional requirements where realized by using the models based on the cyber-foraging tactics. The result is a functioning application that offers all the capabilities described in the selected requirements.

4.2.3 Non-functional requirements

NFR1 - Fault tolerance and reliability
Because service instances run in separate threads after the initial connection, a failed service thread won't influence the main service thread. Passing data between threads hap-

pens through thread-safe queues (*java.util.concurrent.ConcurrentLinkedQueue*). Furthermore, the main surrogate process periodically checks whether all service threads are alive. Crashed threads are restarted. There are no checks on the main process in the demo yet.
When a computation is successfully offloaded, the results are stored (at this moment indefinitely while the surrogate is running) until the app requests the corresponding ticket number. The results are only deleted when a message confirming reception is returned by the app (see also sequence diagram 3.9). Other connection loss scenarios are handled as per table 2.4, for example, when weather data has been stored on the mobile app but the connection fails, it is queued to be sent later.
This storage of data at system nodes when a connection fails is based on the Fault Tolerance and Cached Results tactics. Client-side Data Caching is also used, since caching happens at mobile devices as well.
NFR1 has been fully realized as far as the implemented components go.

NFR2 - Easy deployment

The mobile app cannot be installed through an app store yet. No configuration is needed after installation. The process of setting up a surrogate amounts to filling in a form and submitting it. Component CD22 (the Surrogate Discovery Manager) detects and connects to surrogates whenever the app tries to invoke a service.
Automatic connection is based on the Surrogate Discovery tactic as well as the Local Surrogate Directory tactic. This part of the implementation deviates from the models, since it originally was planned to be the Surrogate Broadcast tactic. The Surrogate Discovery Manager component now functions as the Surrogate Directory component (3, pp. 176–177), and is responsible for setting up the initial connection to the surrogate as well. This is different from the model described in the tactic, the reason being that this component was already coded to be used for the Surrogate Broadcast tactic, and partly functional. Updating it with a directory was the fastest way to realize the new tactic without having to radically change the existing codebase.
Another difference to the tactic is the absence of the Surrogate Directory UI component. Due to time constraints, the list of surrogates was hardcoded in CD22. A way to populate the directory should be present in a fully functional version of the application.
NFR2 has been mostly realized for the implemented components, except for the app being available in an app store. This is however not a critical feature for the demo.

NFR3 - Usability

At present, application text is in English, other languages have not yet been implemented.

No tactics are related to this requirement.
NFR3 has not been realized except for text in English. However, this requirement is not related to the tactics or cyber-foraging and as such is not critical for the demo implementation.

NFR4 - Extensibility

A description of the surrogate service structure and adding new services will be available on the project Webpage (9). Adding a new service is not optimized yet, and as such extensibility is still lacking. Java classes like the server workers should have interfaces, for example, which would make adding new services much easier. The Javascript files should also be organized better, grouping code that belongs to the same services.
No tactics are related to this requirement.
NFR4 has not been realized fully, however it is not critical for the demo version and the research.

NFR5 - Energy efficiency

The app only tries to connect to surrogates when necessary, either when a service is invoked or when there are items in an offload queue. The expensive regression computations are offloaded.
Worker threads are only created on the surrogate when a queued task has to be performed or when connections are made by the mobile app, and workers are destroyed after the task has been performed.
Offloading was based on the Computation Offload tactic, with no notable changes to the model.
The use of workers implements the Just-in-time Containers and Scalability/Elasticity tactics. The availability of standard libraries for threads and data structures supporting easy communication between threads in Java influenced the choice to implement this tactic in the chosen manner.
NFR5 has been realized. By using offload of expensive computations the energy use of the mobile device is minimized. Surrogates only create worker threads when needed, saving energy. There is some room for optimization by looking into the sleep time of the non-worker threads running on the surrogate.

NFR6 - Capacity

The mobile app size is under 10 MB, not counting stored computation results (which can be deleted by the user). The surrogate can handle multiple service requests simultaneously.
The Just-in-time Containers and Scalability/Elasticity tactics are at the basis of the surrogate being able to handle multiple service requests at a time, realized by the im-

plementation of services through the creation of worker threads. The need to handle multiple requests was therefore another factor leading to choosing Java threads for the surrogate implementation.
NFR6 has been fully realized.

NFR7 - Availability

Availability testing has not been performed yet, because not all hardware components (notably the solar battery component) were available at implementation time. Checking battery life and computing whether an offloaded computation will take long have not been implemented yet for the same reason.
NFR7 has not been realized.

NFR8 - Performance

The surrogate does not yet broadcast its presence because the mobile hub component has not been implemented yet. All other aspects of this requirement have to do with mobile hub functionality.
NFR8 has not been realized, it is not related to the implemented components.

NFR9 - Recovery

A startup script will be included in the surrogate OS image. No tactics are related to this requirement.
NFR9 will be realized.

NFR10 - Data integrity

Input checks were not given high priority in the demo implementation, some are present however. No tactics are related to this requirement.
NFR10 was not fully realized. It is however not critical the demo nor for the cyber-foraging tactics.

5

Conclusion and Discussion

5.1 Conclusion

This document has covered the usage scenarios and requirements for an agricultural knowledge exchange system that can be used in resource-challenged regions. A set of cyber-foraging tactics was selected based on those requirements, and consequently used to create an architecture for the system.
The demo implementation does not cover all requirements, but in conjunction with the architecture and design models nonetheless shows that the selected tactics can be used to successfully create the architecture that in turn is used to realize the functional as well as non-functional requirements of this kind of system.

5.1.1 Research results

RQ1: What are the usage scenarios for an agricultural knowledge exchange system to be used in a resource-challenged region?
Results: A system that can support exchanging weather information was used as the use case for this research. Section 2.2 describes a base set of usage scenarios for this type of system.
RQ2: Which of the proposed cyber-foraging architectural tactics can be used in the development of the system and how do they map to the system requirements?
Results: The system requirements, based on the scenarios, can be found in section 2.3. The tactic selection, along with the mapping of tactics to these requirements, can be found in section 2.3.4. The tactics that were selected are Computation Offload, Data staging, Pre-fetching, Surrogate Provisioning, Preprovisioned Surrogate, Surrogate Dis-

covery, Surrogate Broadcast, Fault Tolerance, Cached Results, Scalability and Elasticity and Just-in-Time Containers.

RQ3: What system architecture and design follows from using the selected tactics?

Results: By using the descriptions and example models of the selected tactics, a base set of 19 system components could be created, combined with a base set of component interactions. The final number of components was 22, with 3 components based on requirements that were not related to cyber-foraging. Therefore, the majority of the system architecture and design was directly derived from the tactics.

The full architecture and design can be found in chapter 3.

RQ4: Does the developed system based on these tactics meet all its functional and non-functional requirements?

Results: A demo version of the application was created, which implements two out of three of the main components (Mobile app and Surrogate). These components cover seven out of ten functional requirements. Six of those requirements were implemented. The unimplemented component was not related to or based on cyber-foraging tactics, and was therefore not as relevant for this research.

Out of the ten non-functional requirements, five were realized, of which one has no relation to cyber-foraging. Three were partly realized, two of which have no relation to cyber-foraging. Three non-functional requirements were not implemented, of which one had no relation to cyber-foraging, one was dependent on the unimplemented component (the mobile hub), and one was dependent on unavailable hardware.

It can be concluded that the tactics were successfully used to create an architecture and implementation of an application for agricultural knowledge exchange that fulfills most of its relevant requirements.

Main Research Question: What cyber-foraging architectural tactics can be used to develop an agricultural knowledge exchange system to be used in a resource-challenged region?

Results: The following tactics from (3) can be used: Computation Offload, Data staging, Pre-fetching, Surrogate Provisioning, Preprovisioned Surrogate, Surrogate Discovery, Surrogate Broadcast, Fault Tolerance, Cached Results, Scalability and Elasticity and Just-in-Time Containers.

5.2 Discussion

Although a working implementation and a design document were created based for the greater part on the tactics listed in the conclusion, other tactics could possibly be used, either for adding new functionality or implementing parts of the system differently. Firstly, one of the selected tactics wasn't implemented yet and should still be tested, namely the Pre-Fetching tactic. Secondly, different software engineers might prefer to use different tactics for similar functionality based on their knowledge and experience. More experimenting and comparing results would help to get more insight.
Furthermore, this study did not compare the tactics-based approach to other architectural approaches. This study can hopefully be used for comparisons in this aspect. The method did prove to be a reasonably straightforward as well as a very structured way of building an application.
The use of the Surrogate Broadcast tactic would have been preferred over the Surrogate Directory tactic, because it would remove the need for a way to propagate the connection information of surrogates that are added to the system later to all system nodes, which is more complex than each surrogate simply broadcasting a signal. The use of QR codes on surrogates to scan for the connection data on the mobile device was considered but not realized since there were no (open source) libraries that could be used for QR code scanning on Firefox OS at implementation time.
In a future implementation, this issue should definitely be revisited. Surrogates broadcasting their presence also seems more in line with the idea of mobile devices opportunistically making use of available resources.
For this project, the usage scenarios were created through discussion with people who were involved with projects in resource-challenged regions before. The ideal way to create a set of scenarios would evidently have been to gather them from the would-be users of such a system.
The entire project, including this document and the software component, will be made available through the Github page as an open-source project. Aspects that have less to do with the cyber-foraging tactics aspect of the project such as the voice interface (FR8), user interface and selection of proper services based on relevant scenarios could possibly merit their own research and projects.

6

Appendices

6.1 Appendix A: Running the application

The AgroTempus software is available at (9).

6.1.1 Installation

Surrogate A Raspbian image with the surrogate component installed is available at the project page (9). This image was tested on a Raspberry 2 model B with a TL-WN722N wireless adapter connected. The surrogate can also be ran or exported as a runnable Jar from the Eclipse workspace. A runnable Jar file is also included in the *Runnable_ AgroTempus_ surrogate* folder. If the surrogate is not started from the provided image, make sure to read the setup instructions below.

App The app can be installed on a Firefox OS phone, version 1.3 or higher, or ran on a simulator. Running it on a phone is preferred, as visual artifacts occur when running the app on the standard simulator. Installation is done through the WebIDE component of the Firefox Web browser, which can be started by navigating to *Tools* > *Web Developer* in the menu. If a Firefox OS phone is connected to the computer, it can be selected after clicking the "Select runtime" menu. Simulators can also be installed and selected here.
When a runtime has been selected, navigate to *Project* > *Open Packaged App*, and select the app project folder, *AgroTempus-app*. The app can now be installed and used.

6.1.2 Setup

Surrogate To be able to load dummy data when running the surrogate from the Jar file, make sure the data files (*DUMMY_FORECASTS.json* and *DUMMY_REGIONAL_DATA.json*) are in the *surrogate/data* folder, where "*surrogate*" is the folder in which the Jar is located.
The first time the surrogate is started, location data should be entered through a dialog. The application should be allowed to write a setup file to the folder where the Jar is located.

App To be able to use app functionality, Wi-Fi should be be enabled on the phone. If the standard surrogate setup (using the image for RaspBerry Pi with a TL-WN722N Wi-Fi adapter) is used, the correct connection data will be available in the app and no further setup will be needed. To be able to interact with the surrogate software if it is ran from another type of computer, the surrogate's connection information should be added to the *storage.js* file in the *AgroTempus-app* folder of the project. At the "SURROGATE CONNECTION DATA" comment, the first entry can be edited with the connection data for the system the surrogate will be running on. The app needs to be redeployed from the WebIDE console after these changes.

6.2 Appendix B: Usage guide

Surrogate The surrogate has a user interface that prints system messages to the screen. From the user interface, the surrogate can be closed, or the user interface can be closed while keeping the surrogate running. Entering "ui" on the program's STDIN will relaunch the user interface.

App A description of all app screens is provided in this subparagraph. Figure 6.1 shows a selection of app screenshots.

- **Main menu** The first screen (1) is the main menu, which is visible when the app starts.
- **Store data** To store weather data, the "Submit data" button is used to go to the data submission screen (2). Here, data can be entered and submitted.
- **Get data** To retrieve weather data, the "Get data" button is used. It will take the user to the data retrieval screen where a start and end date can be entered to get all data between these two dates for the surrogate that is currently in range.

- **Forecasts** The "Forecasts" button will take the user to the forecasts screen, the app will automatically try to load the latest forecasts.
- **Prediction** The "Prediction" button opens the prediction screen (3), where a variable to predict can be chosen. The app will try to connect to the surrogate to offload this operation. If successful, a "new message" icon will appear at the top of the screen. Tapping this will take the user to the received items screen.
- **Regression** Likewise, the "Regression" button opens the regression screen (4). When a query is submitted, an offload is attempted. If it is successfully performed, the "new message" icon will appear.
- **Received items** At the received items screen, all computation results that have been received from surrogates are listed. Tapping the ticket number will open the details for each item.

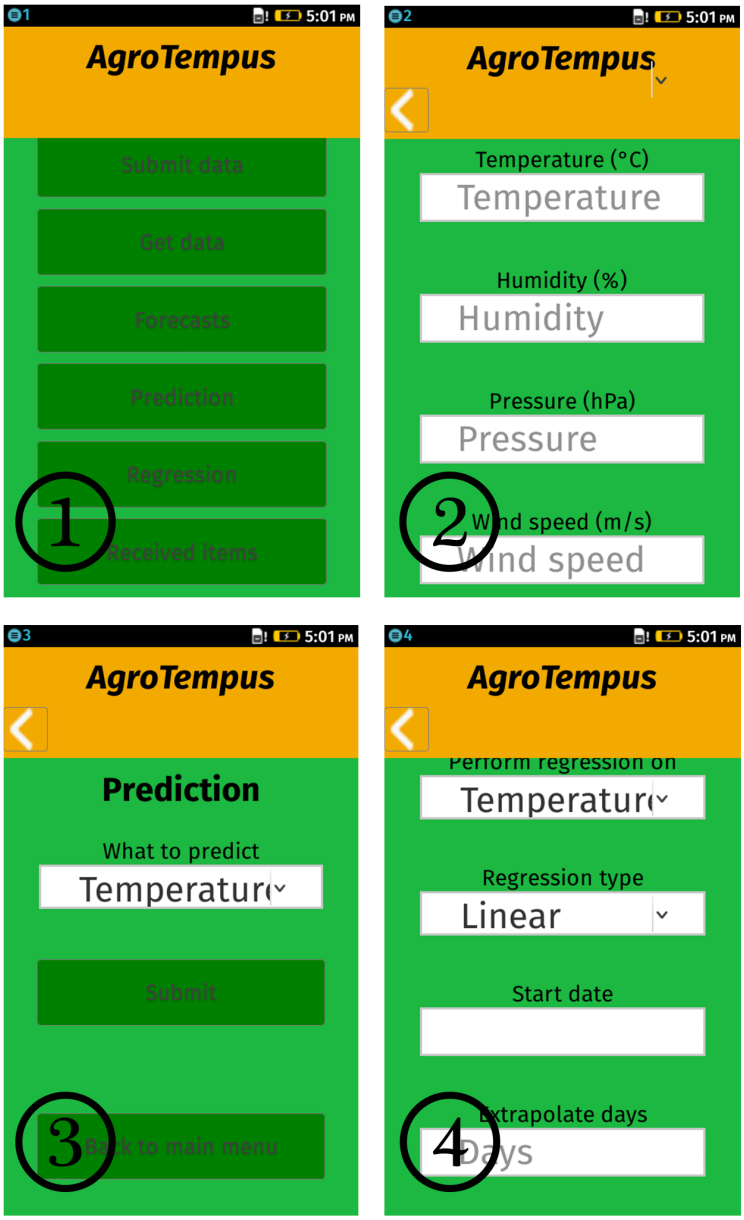

Figure 6.1: App screenshot

References

[1] **Voices project**, 2015. 1, 2

[2] **Firefox OS Phones for under 50 dollars**, 2015. 1

[3] Grace Lewis and Patricia Lago. **Architectural tactics for cyber-foraging: Results of a systematic literature review**. *Journal of Systems and Software*, **107**:158–186, 2015. 1, 21, 33, 34, 36, 37, 51, 54, 58

[4] Gu Zhang. **From feature phones to smartphones, the road ahead**. 2015. 3

[5] Ericsson AB. **Ericsson Mobility Report Appendix, Sub-Saharan Africa**. 2013. 3

[6] Alex Sandy Pentland, Richard Fletcher, and Amir Hasson. **Daknet: Rethinking connectivity in developing nations**. *Computer*, **37**(1):78–83, 2004. 3

[7] **Raspberry Pi website**, 2015. 3, 52

[8] Marat Charlaganov, Philippe Cudré-Mauroux, Cristian Dinu, Christophe Guéret, Martin Grund, and Teodor Macicas. **The Entity Registry System: Implementing 5-Star Linked Data Without the Web**. *arXiv preprint arXiv:1308.3357*, 2013. 13

[9] **AgroTempus GitHub project page**, 2015. 49, 52, 55, 61

[10] **jQuery Website**, 2015. 51

[11] **JFreeChart Website**, 2015. 51

[12] **Apache Commons Codec Website**, 2014. 51

[13] **JSON Website**, 2015. 51

[14] **OpenWeatherMap**, 2015. 51

[15] **JSON Simple Google Code page**, 2015. 52

[16] **Eclipse IDE Website**, 2015. 52

Declaration

I herewith declare that I have produced this paper without the prohibited assistance of third parties and without making use of aids other than those specified; notions taken over directly or indirectly from other sources have been identified as such. This paper has not previously been presented in identical or similar form to any other Dutch or foreign examination board.

The thesis work was conducted from 2-1-2015 to 10-1-2015 under the supervision of Grace Lewis at Vrije Universiteit Amsterdam.

Amsterdam, 9-1-2015