Project Distributed Systems 2nd Term

Tom De Schepper and Steven Latré

2016-2017

1 Introduction

This document discusses the project assignment for the Distributed Systems course in second term. The goal of the project is to create a distributed application representing a smart home environment of the future. This assignment needs to be completed **individual**. In this document we will provide you with the list of the features and requirements that need to be implemented, some details about the evaluation and some other practical aspects.

2 Background

Over the last few years there has been a dramatic increase in the number of devices that are connected to the internet. For instance, in 2013 there were almost 10 billion devices connected and it is predicted that the number of 50 billion will be reached in 2020. A large majority (more than 80 %) will use wireless technologies, like Wi-Fi or Bluetooth Low Energy (BLE), to connect to the internet. Examples of such devices are, among others, parking sensors, home automation (= domotics), robots and intelligent cars. This (r)evolution of a traditional internet with computers to an globally interconnected network of all kinds of every day devices, is known as the Internet of Things (IoT).

This new environment of public accessible and with the internet connected devices has opened the door for many new applications and possibilities. Key domains within the IoT are, for instance, smart cities, health care, logistics and smart homes. This project is situated in the latter domain. Here the focus lays on the automation and control of different aspects in a home environment. For instance, connecting different types of sensors (e.g., temperature sensors and security cameras) and devices (e.g., lights, air conditioning and smart fridges) to one control system that interacts with the user. An example of such a system is shown in Figure 1.

3 Goal

The goal of this project is to develop a distributed system that represents a smart home environment. This distributed system will have at one side a controller (also called server) that manages the entire system. At the other hand, there are a series of clients that can interact with the controller or with other clients. There will be four different types of clients: first of all, there is a user that represent an occupant of the smart house. Furthermore, the other clients are devices or objects in the home environment: a temperature sensor, a smart fridge and a light. Multiple occurrences of each type are possible in a home setup. Each client will be able to perform certain actions and communicate with the other devices. For instance, a user can enter or exit the system or the house, can place something in the fridge, get a list of all the devices or the other

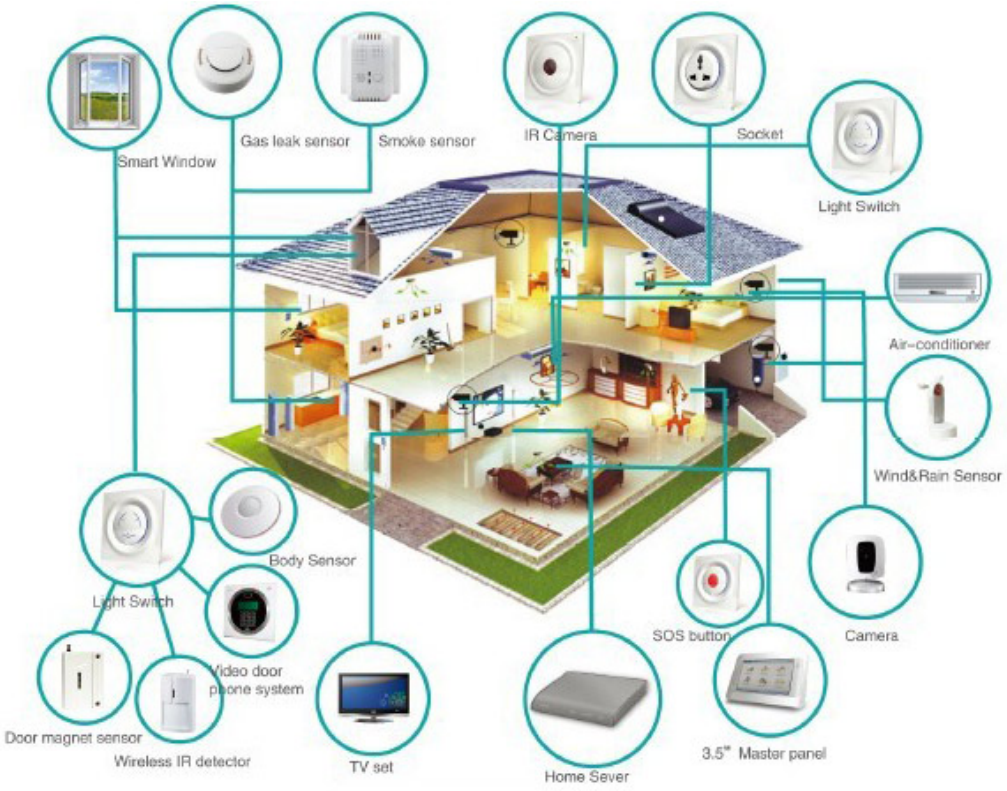

Figure 1: An example of a smart home environment

persons in the house, etc. Besides the different devices with their respective actions, also some basic aspects of fault tolerance and replication will need to be implemented. Since this is the version of the project for a group of three students, additionally a time synchronization algorithm will need to be implemented. The Apache Avro library will be used for the communication between the server and the clients and the mutual communication between the clients.

4 Features

In this section we will list the different features and components that will need to be implemented in the project.

4.1 Controller/server

The server should implement the following functionality:

- All clients (both devices and users) should be able to register with the server and will have a unique ID or username.
- The server will maintain a list of all the connected clients, both devices and users. For the users there should also be an indication if the user is in the house/system or not.
- If requested by the user it should be able to answer the request for a status overview of all the lights.

- If requested by the user it should switch the state of a specific light.
- The controller should handle the entering or exiting of users correctly: the users remain registered in the system and when a user enters or leaves the house, a information message will be transmitted to all other users in the house. After the last user leaves the house, all the lights will be turned off in order to save energy. However, when the first user enters the house afterwards, the state of the lights should be restored as it was before. In other words, when the last user leaves the house, the controller should remember the state of all the lights.
- If requested by the user it should be able to provide the user with a list of the inventory of a specific fridge.
- If requested by the fridge, the controller will inform all clients that the fridge is empty.
- If requested by the user, a fridge should be opened. This means that afterwards the controller will **not** handle anymore the communication between the user and fridge.
- The controller handles all communication, except for the direct connection between a user and a fridge after the fridge has been opened.
- The controller keeps the most recent x measurements for every temperature sensor. For x a practical value should be chosen.
- If requested by the user the controller will give the current temperature in the house (most recent measurement). If multiple temperature sensors are present, an average value should be calculated.
- If requested by the user the controller will give a history of the temperature in the house, using the stored most recent x measurements. If multiple temperature sensors are present, the measurements should be averaged.

4.2 Clients/devices

There are four different types of clients: a user, a temperature sensor a smart fridge and a light. Make sure that is possible to have multiple instances of each type within a scenario. An example setup that should be possible is a controller with the following devices connected to it: 2 users, 1 fridge, 3 temperature sensors and 3 lights.

4.2.1 User

We expect that following features have been implemented for a user:

- Connect to the controller and have a unique name or ID.
- A user should be able to enter or exit the system. This responds to the action of leaving or entering the house. Default, when a user is connected for the first time to the system, you can assume that he or she is in the house. For testing purposes, when a user enters or leaves the house, this action is printed in the terminal window.
- A user can ask the controller for a list of all devices and other users.
- A user can ask the controller for an overview of the state of all the lights (on or off).

- A user can ask the controller to switch a specific light to another state.
- A user can ask the controller for an overview of the inventory of a fridge
- A user can ask the controller to open a fridge. Afterwards it can add or remove items from the fridge. This communication will happen directly between the user and the fridge. A user can close the fridge when he or she is done adding or removing items. During the time the fridge is opened, the user cannot interact with an other device or the the controller. When the fridge is closed, the user connects back with the controller.
- A user can ask the controller for the current temperature in the house.
- A user can ask the controller for the a history of the temperature in the house.

4.2.2 Temperature sensor

We expect that following features have been implemented for a temperature sensor:

- Connect to the controller and have a unique name or ID.
- When a temperature sensor is launched, an initial start temperature should be provided (not hardcoded).
- Every sensor has an internal clock that is initially zero when the sensor is started. Every second in real time, the clock of a sensor also progresses by one. To simulate the drift behaviour of real clocks,every time the clock is increased by one, a drift value will also be added. For simplicity, we will assume that the drift remains constant over time. This drift value is provided at the start of the sensor (not hardcoded) and could, for instance, be 0,1 or 0,25.
- Every x seconds, the temperature sensor sends a new measurement to the controller. For x a practical value should be chosen. The value of this measurement is created by adding a random value between -1.0 and 1.0 degrees to the previous measurement, starting from the initial start temperature.

4.2.3 Smart fridge

We expect that following features have been implemented for a fridge:

- Connect to the controller and have a unique name or ID.
- Have a list of all the items that have been stored in the fridge. Items are represented by a string. For testing purposes, assume that every item has a unique name. The list can, for instance, look like this: [”apple”,”cheese1”,”milk”,”cheese2”,”butter”].
- It should be possible to ask the fridge for a list of all the items it contains.
- It should be possible to add or remove an item to or from the fridge. If the last item is removed from the fridge, the fridge should contact the controller to inform all users that the fridge is empty. Adding or removing an item can only be done after a the fridge has been opened by a user. Only that specific user will be able to add or remove items from the fridge. The communication between user and fridge will occur directly in this case and will not pass by the controller. After the user is done, the fridge will be closed again.

4.2.4 Light

We expect that following features have been implemented for a light:

- Connect to the controller and have a unique name or ID.
- Switch the device from one state to the other. There are only two states possible: on or off. For testing purposes, the new state should be printed in the terminal window.
- Ask the device for its current state.

4.3 Communication

In the theory classes and the introduction to Apache Avro we mentioned that there are two kinds of communication possible between the client and the server: synchronous and asynchronous. We expect that you understand these methods and that you can use and implement them in the correct situations within this project.

You should also take into account that the Avro RPC protocol is basically client-driven (i.e., pull-based communication), as explained during the introduction. In this assignment you will also have to use push-based communication (the invocation of methods of the client). To allow this, the clients should start their own server, in order to make a Avro protocol object available for invocation.

Furthermore, we expect that a server and a client can function on different physical machines and communicate with each other. This can be done by letting the server and clients ask for the correct ip addresses and ports, when they are launched. You should take into account that multiple `SaslSocketServers` on the same machine need different ports.

4.4 User interface

It is not necessary to implement a GUI (graphical user interface) for the interaction with the user. Instead you can use a terminal window to input commands. This can be done by means of the *Cliche* library, a Java library that allows the creation of a CLI (command-line user interface) in a simple manner. Documentation can be found on the following page: `https://code.google.com/p/cliche/wiki/Manual`. Note that other libraries are also allowed to use.

Although it is not mandatory, you are allowed to implement a simple GUI, instead of using a CLI. Some examples of libraries that can be used for this are AWT, Swing, JavaFX or QT Jambi.

4.5 Fault tolerance

We expect the implementation of a basic version of fault-tolerance in order to avoid crashes of the application:

- If the server goes down, a new controller should be elected from a set of clients. Clients that can be promoted to a (temporal) controller are the users and the smart fridges. The temperature sensors and lights are not available for election. To elect a new controller, the Ring algorithm of Chang-Roberts (seen in the theory part of this course) should be implemented. After a new controller has been elected, the distributed system should function as before. For sake of easiness, the new

controller should only function as a controller and not as a client. For example, a smart fridge that is elected as controller, will not act as a smart fridge during the time it is a controller. If the original controller would come back online at some point after the election, the elected controller will return to its previous function and the original controller will once again be the only controller in the system.

- We expect that the list of all connected devices to the system remains accurate at all times. If a device has gone offline, the list should be updated.
- At all times, all exceptions (e.g., when a certain connection is closed) should be handled appropriately. An exception should never be shown in the terminal window. An example of one of the exception that should be taken into account is, for instance, when a user crashes while communicating with a fridge (adding or removing items), the fridge should be closed.

4.6 Replication

In order to smooth the process when a new controller is elected after the old one has gone offline, the data of the server should be replicated upfront. The controller should replicate all of its data to all possible candidates (users and smart fridges) during its lifetime. To this extend, you should implement one of the consistency models that were studied during the theory classes. You should be able to motivate your choice during the final evaluation.

4.7 Time synchronization

In order to synchronize the clocks of all temperature sensors a time synchronization algorithm will need to be implemented. It is up to you to decide which algorithm, from those seen in the theory classes, you will implement. You should motivate your choice and relate it to the benefits and disadvantages from each algorithm. Make sure to test the implementation with different values for the drift parameters. Furthermore, make sure that the value of the clock is outputted at regular intervals, to be able to test if the algorithm is working.

4.8 Other considerations

During implementing the different features or components you should try to use the most efficient solution possible and always leverage different options. During the final evaluation we will ask you to explain certain design choices. Furthermore, an overview should of your architecture should be provided in the manual.

5 Evaluation

5.1 Minimum requirements

There are some minimum requirements that your solution should be comply to, in order to pass for this part of the course. If this requirements are not fulfilled, it is not possible to pass:

- The code should compile on the referential platform. This also means that all the necessary libraries should be present.
- Your implementation should be able to run on the reference platform. To make this possible we expect that you write a manual on how we can run your solution.

- Since this is a project for the course Distributed Systems, your solution should run on multiple machines. In other words, the server(s) and clients should be able to communicate with each other from different computers.
- For the communication between the different components you should use Apache Avro and not another technology.
- Every submission should have a manual containing at least instructions on how to use the program and a description of the architecture.

Remark: if your solution complies to these minimum requirements, this does not mean that you will certainly pass this assignment.

5.2 Other important requirements

If your solution complies to the minimum requirements, the next aspects will define your grade:

- We will mainly look towards the implemented functionality. So we will check if all the necessary functionalities have been implemented and if they work correctly. Some very important aspects of the assignment are fault tolerance, replication, time synchronization and the architecture of your solution.
- Furthermore, it also important that you show insight in the used technologies and concepts. You should be able explain the different design choices you have made.

6 Other practical guidelines

6.1 Individual project

In contrast with the project first term, you will have to complete this project individual. You are allowed to start from the code that was submitted in January for the first term evaluation. Notice that we will check for fraud or plagiarism.

6.2 Reference platform

The reference platform are the **8 computers in the middle of the computer lab on the second floor** of the library (M.G.234). At the evaluation we expect that your code compiles and executes on these machines. **You are only allowed to demonstrate your project on these machines.** These computers are pairwise connected with each other, so every group can test their application using two computers. More information on how to use these computers can be found on Blackboard.

7 Submission and deadlines

7.1 Submission

We expect that you add a manual or README with every submission that clearly indicates how we should use your code in order to demonstrate the functionality of your solution. We also expect that you add a description of your architecture and the used technologies and libraries.

Your solution should be handed in by WeTransfer to my email address: **tom.deschepper@uantwerpen.be**. Your solution should be zipped and have the following name: **lastname-firstname.zip** Make sure that all necessary scripts, files

and libraries are present, because only this code will be used on the evaluation. Make sure to test everything upfront.

7.2 Deadline

The final deadline for the project will be communicated after the second semester. Most likely the final deadline will be at the start of the 2nd term in August. During the examination period, in August or September, an evaluation moment will be scheduled where you can demonstrate your project.