Exercise 2: Quality Attributes and Architectural Design of the HS07 System

Anders H Poder, Jesper Dalberg, Lars Kringelbach

Department of Computer Science, University of Aarhus
Aabogade 34, 8200 Århus N, Denmark

Group 11 - Kilo
19951439, 20074976, 20074842
{ahp, jdalberg, u074842}@daimi.au.dk

2008-02-25

Abstract

The HS07 system implements a closed-loop control of the heating in a private home. It monitors thermometers in the home, and based on measurements HS07 adjusts radiators in the home. This report gives a quality attribute analysis and discussion and a architectural design of the HS07 system. The techniques used for both quality attributes and architectural design are taken from [Bass et al., 2003].

1 Introduction

Figure 1 shows a schematic overview of HS07 in a home. The home may be accessed by the home owner from the outside through the HS07 gateway. The HS07 gateway also monitors and controls the home.

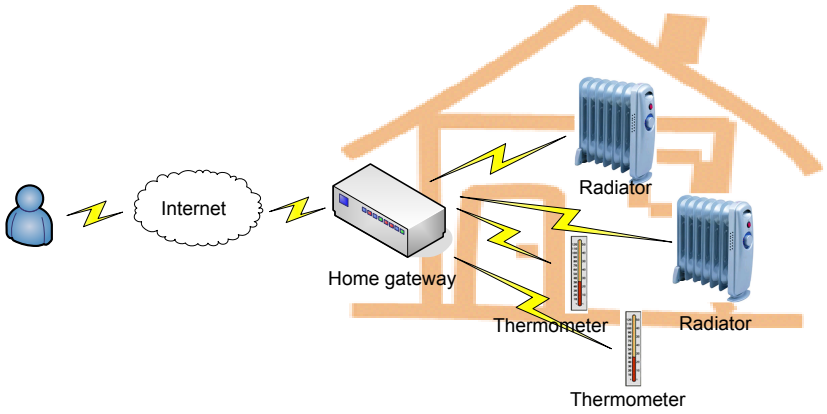

Figure 1: HS07 in a home

HS07 includes sensor and actuator hardware which runs on an embedded Java virtual machine with standard software.

An architecture of the HS07 system was proposed in [Kringelbach et al., 2008]. In this report we will elaborate on the quality attributes of the system, define some quality attribute scenarios and propose a new architectural design to fulfill these scenarios.

The proposed design has been implemented as part of exercise 2.

2 Quality Attributes in the HS07 system

2.1 Availability

As the HS07 relay on distributed components receiving and transmitting information, as well as remote user access over unreliable communication media outside of the systems control, availability has a high priority. Timely availability Hard requirements as to timeing of responses from the individual services are not that important in the HS07 system. Whether an actuator responds in 1 second or 5 seconds is not that important, just like the internet link in the gateway will not sease to function if its response time is e.g. doubled.

2.2 Usability

The HS07 system must be used by ”ordinary” users (i.e. non-technical), and a high degree of usability must therefore be inherent in the architecture to support rapid user interface modification as well as implementing functionality that may aid the user during use.

2.3 Performance

Performance and availability are related in the sense that increasing performance may increase timely availability, yet as the timely availability was deemed of low priority, this part of the performance quality attirbute is also of little importance. Another part of performance is related to footprint and memory usage rather than execution speed, and as the system is based on smaller sensors and actuators, this part of performance becomes very important. It is also related to another important quality; cost. The system must be able to handle enough sensors and actuators to control a full house.

2.4 Modifiability

As the system is based on dynamically replacable sensors and actuators of varying type and specification, it is imparative that the architecture supports these varying specification, and modifiability therefore become vital for the system.

2.5 Security

There are always at least two levels to security; percieved security and actual security. As long as the sensors and actuators are controlling non-vital functions, e.g. heating, lights, etc. it may be sufficient with perceived security as an attacker may not endanger safety and have no financial motive. On the other hand if the system is extended to control door-locks and house survailance, then actual security becomes a must. Perceived security is important to dissuade the fun-time intruders and reassure the customer.

2.6 Testability

As the system is highly modifiable, regression-testing is highly-important. Also the requirement of availability creates extra demands on testability, as it is required to test many fault scenarios to ensure propper recovery (if this is the fault-handling mechanism of choice).

2.7 Quality Attribute Selection

The major driving quality attributes of the HS07 system are:

- *Performance.* HS07 should be performant so that a large number of thermometers and radiators may be part of the system.
- *Modifiability.* It must be possible to modify HS07 to include new types of sensors and actuators.
- *Usability.* It must be easy to use the gateway user interface and include new types of sensors and actuators.
- *Availability.* The system must be responsive and available for both users and hardware elements.

3 Quality Attribute Scenarios

The following chapters documents the selection of the Quality Attributes that we select for the HS07 system. The selection of QA is split into a 3-part process.

- Brainstorming - we think up scenarios
- Selection - we chose which scenarios are interesting.
- Refinement - we refine the selected scenarios

The actual brainstorming and selection (voting) was done in Wiki in order to allow for a decentralised brainstorming and vote. The setup may be seen at:
http://code.google.com/p/pasoos2007hbc/wiki/SAiPM1H2.
All information has been moved to the report so it is only the use of Wiki as a meeting-place that may be of interest.

3.1 Brainstorming

In this part of the process we just attempt to think of as many possible scenarios for the HS07 system as we possibly can.

1. A sensor stops responding.
2. Someone leaves a window open in an ice storm, and the sensors makes the actuators overheat.
3. Someone accidentally puts their stiletto through the Ethernet switch, disabling networking parts of the system.

4. After an average user has used the system for less than one hour he or she has mastered the basic functionality.

5. The user adds a new actuator and the system configures to use the new hardware automatically within 1 minute.

6. The user logs on externally through the gateway and has access to the home control within 5 second of verifying 98,9% of the time (30 seconds 1% of the time, more than 30 seconds (unscheduled down-time) less than 0,1%) is connected and has access.

7. The user attempts to turn radiator above normal safety-level from outside the house and receive an error explaining the infraction. The system stores a log of the episode and refuse to execute the command.

8. An attacker attempts to log on using an invalid password and is thrown off. The next attempt will take longer (progressive log-on time). The system stores a log of the failed attempt.

9. Developer updates and tests the code base with a new device type in 1 week.

10. A technical installer installs a system at a customer in no more than 4 hours.

11. Developer updates the code base to a new requirement in less than 30 man-hours.

12. Configuration manager installs an upgrade remotely at a customer in no more than 30 minutes.

13. Tester/debugger extract debugging information directly from running system at customer.

14. The system handles 50 sensors and 50 actuators.

15. The system handles up to 50 rooms/groups with one or more sensors and actuators in each group.

16. The user changes the preferred temperature from a web page and the system adapts to it.

17. The customer requests that the system runs on other hardware (e.g. less power consuming hardware).

18. The customer requests that the system interfaces with other sensors/actuators.

19. The developer modifies the sensor poll interval at build time.

20. The system turns off all actuators when sensors cannot be probed.

21. The end user adds a sensor/actuator at runtime which registers automatically.

3.2 Selection

Here we put the scenarios to the vote. This is done by emulation of a real-life situation where the actual stakeholder would be part of the process. We take on the roles of the stakeholders in their absence, and chose which scenarios would be more interesting for us.

3.2.1 Stakeholders and interests

- User (homeowner)
  - Easy to use and configure
  - Always available
  - Safe (will not make an error and burn down the house)
  - Secure (other people cannot control my home)
- Producer
  - Cheap
  - Production line (modifiability for other uses)
- Technical installer
  - Easily deployable (installable)
- Developer
  - Easily adjustable to new requirements
  - Scalable
  - Remote update (upgrading online)
  - Remote maintenance (debugging online)
  - Automatically testable (regression test)
  - Developing in parallel
- Insurance company
  - Safe with respect to fires (radiators) and break ins (door), ...

3.2.2 Identical scenarios

Due to the distributed location of the participants in the brainstorming multiple ”similar” scenarios was created. In order to avoid having the same scenario represented twice with a slightly different wording, the below list show the scenarios which will be considered identical, meaning that their votes will be accumulated.

- 2 and 7
- 6 and 16
- 10 and 17
- 9 and 11 and 18
- 1 and 20
- 5 and 21

3.2.3 First vote

Person	Stakeholder Role	Vote 1	Vote 2	Vote 3	Vote 4	Vote 5	Vote 6
Jesper	User	4	5	6/16	20	21	2/7
Lars	Developer	19	19	5/21	9/11/18	9/11/18	13
Anders	Producer	2/7	9/11/18	9/11/18	12	10/17	10/17

3.2.4 Second vote

Person	Stakeholder Role	Vote 1	Vote 2	Vote 3	Vote 4	Vote 5	Vote 6
Jesper	Developer	9/11/18	9/11/18	9/11/18	19	13	6/16
Lars	User	1/20	4	6/16	15	4	5/21
Anders	Producer	9/11/18	6/16	5/21	5/21	12	10/17

3.2.5 Voting Results

This is the result of the vote in a prioritized order.

1. 9/11/18 (8 votes)
2. 5/21 (6 votes)
3. 6/16 (4 votes)
4. 19 (3 votes)
5. 4 (3 votes)
6. 13 (2 votes)
7. 1/20 (2 votes)
8. 2/7 (2 votes)
9. 10/17 (3 votes)
10. 12 (2 votes)
11. 15 (1 votes)
12. 3 (0 votes)
13. 8 (0 votes)
14. 14 (0 votes)

3.3 Refinement

The process of refining the chosen scenarios centers around specification of various elements of the scenario. Stimulus, Response, Response Measurement.

Table 1, 2 and 3 shows refined scenarios of the three quality attribute scenarios that were given the highest priorities in section 3.2.

Scenario(s):		Developer updates and tests the code base with a new device type in 1 week.
Relevant Quality Attributes:		Modifiability, Testability
Scenario Parts	**Source:**	The developer
	Stimulus:	Whishes to add a new device type
	Artifact	The code
	Environment:	At design time
	Response:	Modification is made with no side effects (all unit-/integrationtest pass)
	Response Measure:	Modification is made and tested within 1 week
Questions:		Does the device use implemented HTTP protocol or do we need an adaptor? Is it necessary to make structual modifications to the user interface? Does the change have architectural impact?
Issues:		Developer is assumed to know the system architecture

Table 1: Refined Scenario #1 - QA Scenario 9/11/18

Scenario(s):		The user adds a new actuator and the system configures to use the new hardware automatically within 1 minute
Relevant Quality Attributes:		Modifiability, Usability, Performance
Scenario Parts	**Source:**	The user
	Stimulus:	Attaches a new actuator
	Artifact	The system
	Environment:	Maintanance operation
	Response:	New hardware is automatically added to the system configuration and accessible remotely
	Response Measure:	Within one minute
Questions:		Is the user allwed to install now components himself?
Issues:		

Table 2: Refined Scenario #2 - QA Scenario 5/21

Scenario(s):		The user changes the preferred temperature from a web page and the system adapts to it
Relevant Quality Attributes:		Usability, Availability
Scenario Parts	**Source:**	The user
	Stimulus:	Remotely change the temperature
	Artifact	The system
	Environment:	Normal operation
	Response:	Acknowledge of operation from gateway
	Response Measure:	Within 5 seconds 98,9% of the time, within 30 seconds 1% of the time and more than 30 seconds 0,1% of the time (unshceduled down-time)
Questions:		When do the temperature adjustment begin? When is the command is received by the radiator?
Issues:		

Table 3: Refined Scenario #3 - QA Scenario 6/16

4 Evaluation

The following sections will elaborate on the refined scenarios described in section 3.3. The compliance of the Architectural Description of HS07 [Kringelbach et al., 2008] for the selected refined scenarios will be described and tactics that can be applied to the architectural design in order to fulfill the quality attributes are identified.

4.1 Refined Scenario #1

The new device type can either be a new thermometer/actuator or a completely new type of device, e.g. a door.

Adding new sensor/actuator:

- Currently the Gateway ↔ sensor/actuator architecture, being decoupled by Service, supports adding a new Sensor implementation as long as it meets the specifications. The sepcification lie only in semantics, and require that a method getTemperature exist for the sensors (and notify for actuators). There is no interface and therefore no syntax checking at compile time (only runtime).
- The present architecture of the sensor/actuator do not directly support replacing the implementation with a new as the Service new the Thermometer type directly (update by modification as opposed to update by addition). It also performs a call to a pure virtual method from a base-class constructor, which is so wrong it is unmentionable.

Adding different types:

- A new sensor/actuator type must only implement the getController method in order to be elligable for use with the Service. The architecture holds.

- For the Gateway there is a hard-coded reference to the Thermometer and Ragiator, in the sense that it has registerThermometer methods and what is notified to the oberservers are the temperature. In a general architecture with generic sensors and actuators this hard-coding will greatly limit the possibility of adding new types, and certainly require modifying the existing code (with the required regression test to boot) in order to do so.

We have chosen to focus in the integration of new types of thermometers and actuators and addition of different sensor and actuator types will not be described any further.

4.1.1 Modifiablity

- New Thermometer/Radiator implementation
  - In the present implementation the ThemometerService has a hard-coded reference to Themometer, and having a new operation in the implementation is bad design if modifiability is desired, as it promoted change by modification. Changing the ThemometerService to use e.g. AbstractFactory to create the given Themometer instance will allow a new Themometer to be implemented and deployed by merely adding a class and a factory, which is modification by addition and then writing a main for the device, which uses this factory and the generic ThermometerService. At present the getController returns an object and the AbstractFactory may be implemented to also return this. It would however be an advantage to define a Thermometer interface in order to ensure that the implementations meets the protocol (has getTemperature method). This imply several tactics, as it *localize modifications* to the temperature implementation. If a Temperature interface is created, then this uses *anticipate expected changes* as the interface must contain the methods, which is expected to be needed, not just now but in the future. *Generalize the module* is used in the fact that Objects are exchanged between Gateway and sensor/actuator, giving the Gateway to work on any instance of a class.
  - The use of HTTP between Gatwway, sensor and actuator removes the syntactical dependency replacing it with runtime checking (by the Invoker), with the advantages and disadvantages it has. There is also a strong *Location dependency* from all sensore/actuators to the gateway, as they must know where it is located (no discovery). It may be updated with a DNS-lookup to make this weaker.
  - As the Gateway must support all sensors/actuators in its run algorithm there is a *QoS dependency* that this must not take longer than the period by which the run-method is expected to iterate.
  - The Gateway already support *defer binding time* as the actuator/sensor registers with the Gateway at runtime.
- New sensor/actuator type
  - The current implementation of the Gateway supports *defer binding time*, yet as the Gateway run method has a hardcoded dependency on the semantics of the radiator and thermometer (through the getTemperature and

notifyObserver command syntax) it is not possible to simply register other types. The Gateway would have to use *Generalize the module* in order to allow more general actuators and sensors to register, as opposed to only those that have radiator and thermometer semantics.

4.1.2 Testability

Tactics for testability include *Test harness* which is essentially a needed setup for regression testing. The test harness for the HS07 system will consist of a series of JUnit test cases on a modular level. On a larger scale, the deployment on a PC, shown in /citeSAiP1H1 is one big test scenario where sensors and actuators a stubbed using visual inspection for verification. This harness will continue to be the main test setup for introduction into the gateway/model parts of the system.

4.2 Refined Scenario #2

Because of the Client/Server architecture of the gateway, adding new actuators is currently possible without architectural changes. With the current design, only actuators of the radiator type can be added.

4.3 Refined Scenario #3

In order to have the system encompass a user interface that allows a user to change the preferred temperature from a web page, the gateway subsystem needs to become the Model part of a MVC pattern.

This functionality is not part of the current architecture, even though it supports commands on the gateway. At least one new command needs to be implemented on the gateway.

Scenario #3 requires the HS07 system to provide a web interface that can be used to change the temperature in the house. The response requirements sets some requirements to the availability of the http service.

4.3.1 Availability

As the user may experience non-responsiveness 0,1% of the time it is believed that this can be done with a single hardware unit (no HW-reduadance, e.g. *voting*). At the same time the requirement of 30 seconds 1% of the time is sufficient time to restart the web-service, and it is therefore not required to have *active redundancy*. Finally it is believed that an availability of 5 seconds 98,9% of the times, require a form of redundancy, and for this reason *passive redundancy* is chosen. As an example of a design may be mentioned the Apache web-server design, which is redundant processes. A number of processes is spawned to handle HTTP/GET requests, and it uses some kind of fault detection scheme to detect if one become non-responsive. If this happens it is simply killed and an identical process is spawned in its place.

4.3.2 Usability

A web interface must be provided to support the requirement that the user can change the temperature from a web page. In order to maintain the modifiability of the system, a *Model-View-Controller* pattern will be used to separate the web interface from the

gateway. A web backend process will be added to limit the web access to the model thereby hiding irrelevant information. The model needs to maintain all information about the system that is relevant to the user - *Maintain a model of the system*.

5 Architectural Design

5.1 Module View

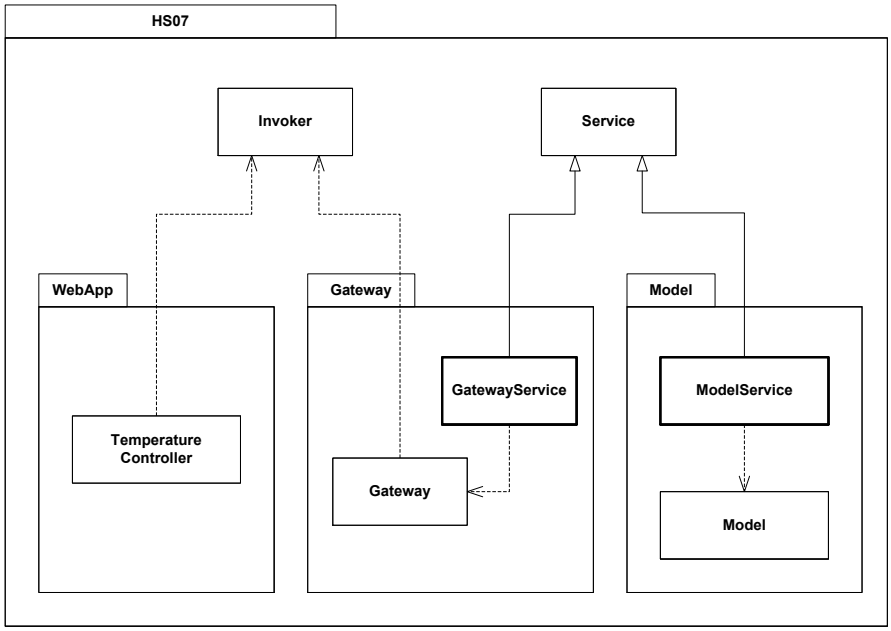

Figure 2: Module View of the web interface in the redesigned HS07 system

Figure 2 illustrates the new packages needed in HS07 to include a web interface for the user. A *Model-View-Controller* pattern has been applied in order to seperate the model from the view and the controller. The model package creates a service with which to get and set the values needed by the user to make a decision an any new target temperature, that is, the Model *maintains a model of the system*. Both the gateway and the WebApp uses their invokers to contact the model service for parameter manipulation.

The separation between the gateway and the model packages creates an opportunity for increase of security when that becomes a driving quality attribute, since the gateway information can be hidden within the system, letting the ModelService become the only exposed part.

In order to archieve the availability requirements, the Apache web server can be used as HTTP server for the WepApp. This is not depicted in the figure.

Figure 3 shows the classes involved and their interdependencies for implementing the *factory* pattern for the sensor module. Please refer to Figure 6 for its runtime usage.

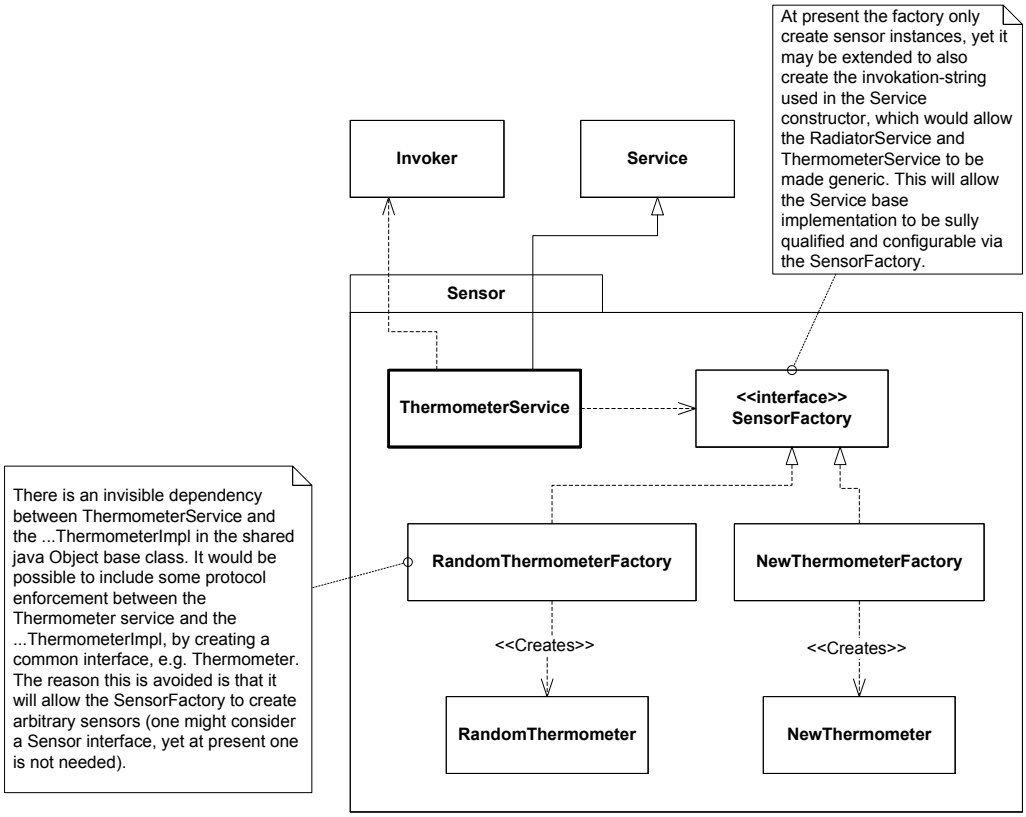

Figure 3: Module View of the abstract factory in the redesigned HS07 system

5.2 Component & Connector View

Figure 4 shows all the active components of the HS07 system. The new parts are the ModelService and the HTTP Server, needed to create a suitable web interface for the user. As can be seen on the figure, the HTTP server only accesses the model and not any internal parts of the system.

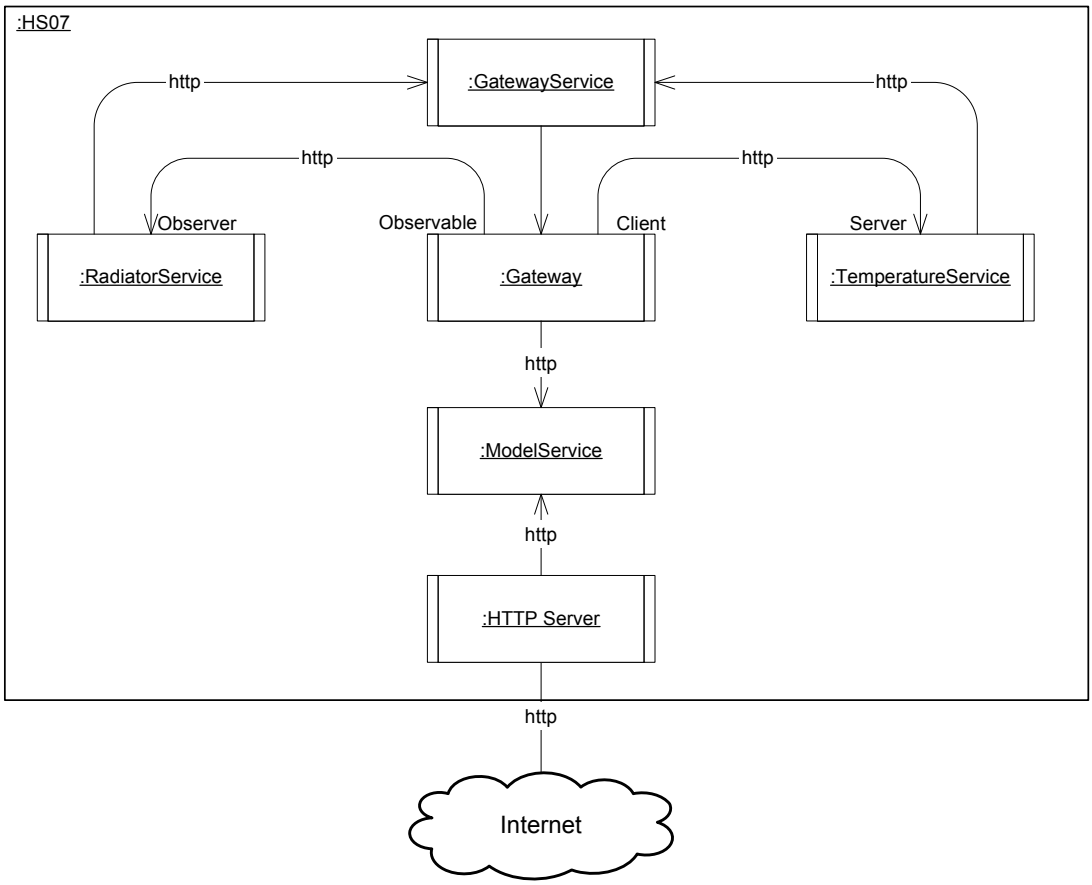

Figure 4: Active Objects in the redesigned HS07 system

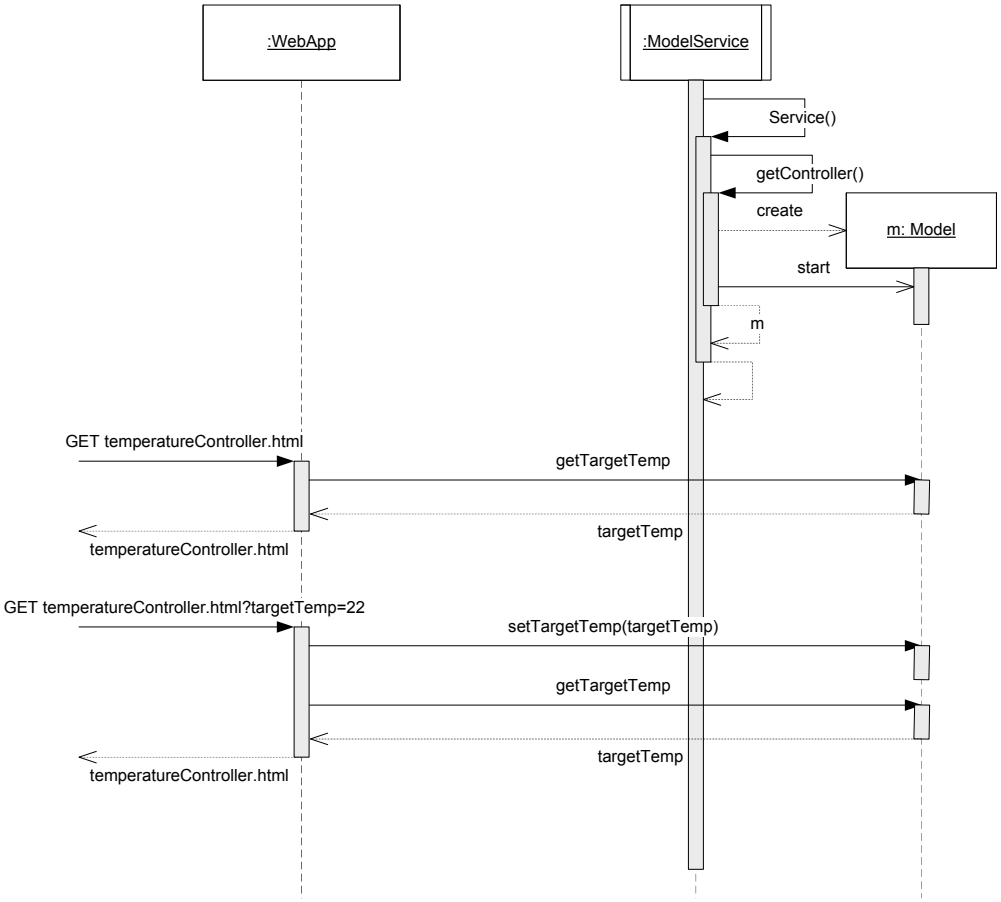

Figure 5: Sequence Diagram for the web requests in the redesigned HS07 system

Figure 5 shows the flow of contol when a web application uses the methods exposed through the model service. The amount of trafic on the model service will have no direct impact on the performance of the gateway service, and the response times on the web application will not be directly impacted when the gateway is polling the sensors.

The gateway uses the same http interface as the webapp to set the value of the current average room temperature, and to read the values of the target temperature requested by the user.

If the ModelService is non-responsive, it is important for the gateway service to continue to work, ”guessing” the target temperature either by cache or by default values.

Figure 6 shows initialization sequence when using the SensorFactory illustrated in Figure 3. The sequence shows the use of the factory to create a decoupling between the actual Thermometer implementation and the ThermometerService.

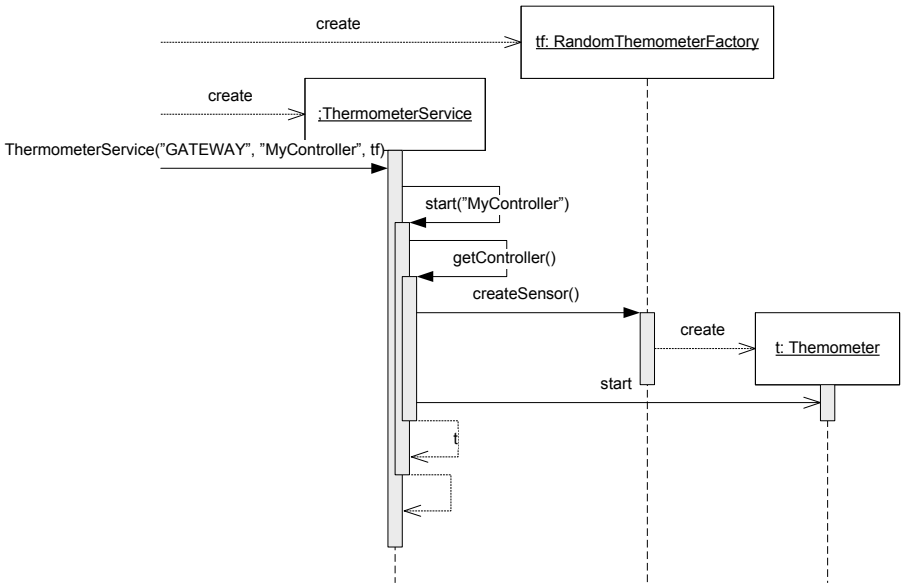

Figure 6: Sequence Diagram for the Abstract Factory in the redesigned HS07 system

5.3 Allocation View

Figure 7 shows how the HTTP Server and the ModelService can be deployed on the Home Gateway next to the GatewayService. As all tasks on the Home Gateway are communicating using HTTP, the HTTP Server or the ModelService can be deployed on different hardware if required. E.g. the HTTP Server can be moved to other hardware in order to minimize the impact on the more critical functions of the GatewayService when accessing the web interface.

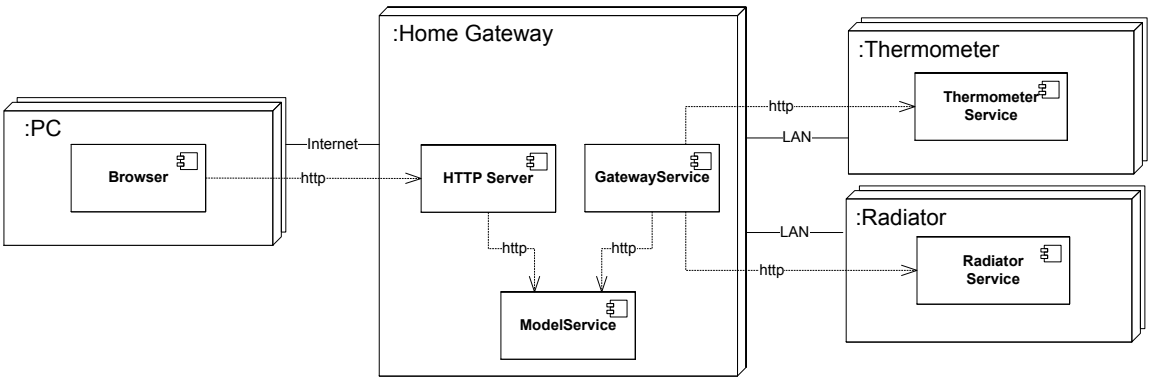

Figure 7: Deployment in the redesigned HS07 system

References

[Bass et al., 2003] Bass, L., Clements, P., and Kazman, R. (2003). *Software Architecture in Practice*. Addison-Wesley, 2 edition.

[Kringelbach et al., 2008] Kringelbach, L., Dalberg, J., and Poder, A. H. (2008). Exercise 1: Software architecture description of the hs07 system. Technical report, AU.