SOFTWARE ENGINEERING
(Week-7)

USAMA MUSHARAF

MS-CS (Software Engineering)

LECTURER (Department of Computer Science)

FAST-NUCES PESHAWAR

AGENDA OF WEEK # 7

4+1 View Model

Discussion on Uber Case Study (System Design)

Towards Object Oriented Design

4+1 VIEW MODEL

The 4+1 view is an architecture verification technique for studying and documenting software architecture design.

THE 4 +1 VIEW MODEL

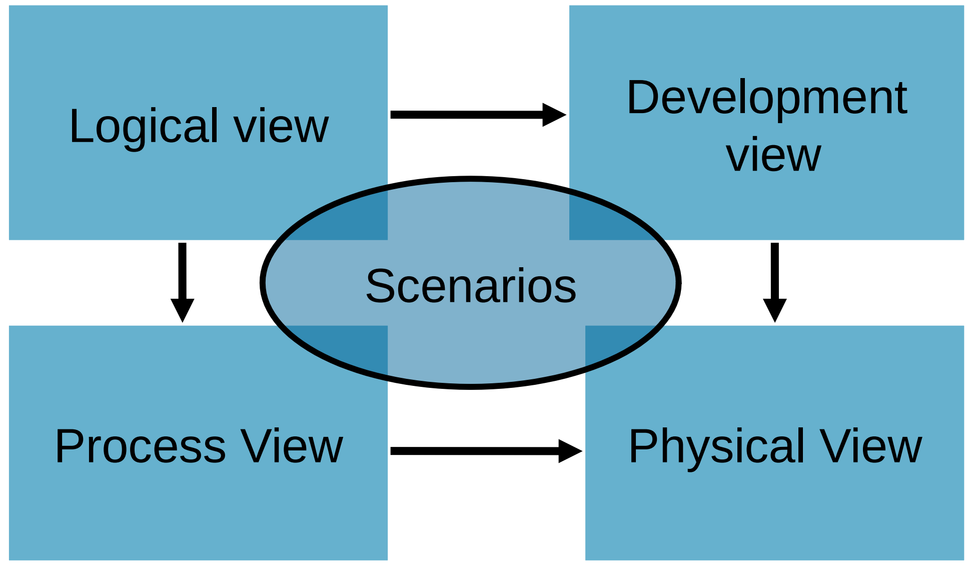
The 4+1 view model was originally introduced by Philippe Kruchten (Kruchten, 1995).

The model provides four essential views:

- the logical view,
- the process view,
- the physical view,
- the development view

and fifth is the scenario view

4+1 VIEW MODEL OF ARCHITECTURE



THE 4+1 VIEW MODEL

Multiple-view model that addresses different aspects and concerns of the system.

Standardizes the software design documents and makes the design easy to understand by all stakeholders.

THE SCENARIO VIEW- USE CASE VIEW

The scenario view describes the functionality of the system, i.e., how the user employs the system and how the system provides services to the users.

It helps designers to discover architecture elements during the design process and to validate the architecture design afterward.

THE LOGICAL OR CONCEPTUAL VIEW

The logical view is based on application domain entities necessary to implement the functional requirements.

The logical view specifies system decomposition into conceptual entities (such as objects) and connections between them (such as associations).

THE LOGICAL VIEW

The logical view is typically supported by

UML static diagrams including class/object diagrams and

UML dynamic diagrams, sequence diagram, state diagram.

THE DEVELOPMENT OR MODULE VIEW

The development view derives from the logical view and describes the static organization of the system modules.

UML diagrams such as package diagrams and component diagrams are often used to support this view.

THE PROCESS VIEW

The process view focuses on the dynamic aspects of the system, i.e., its execution time behavior.

This view maps functions, activities, and interactions onto runtime implementation.

The UML activity diagram support this view.

THE PHYSICAL VIEW

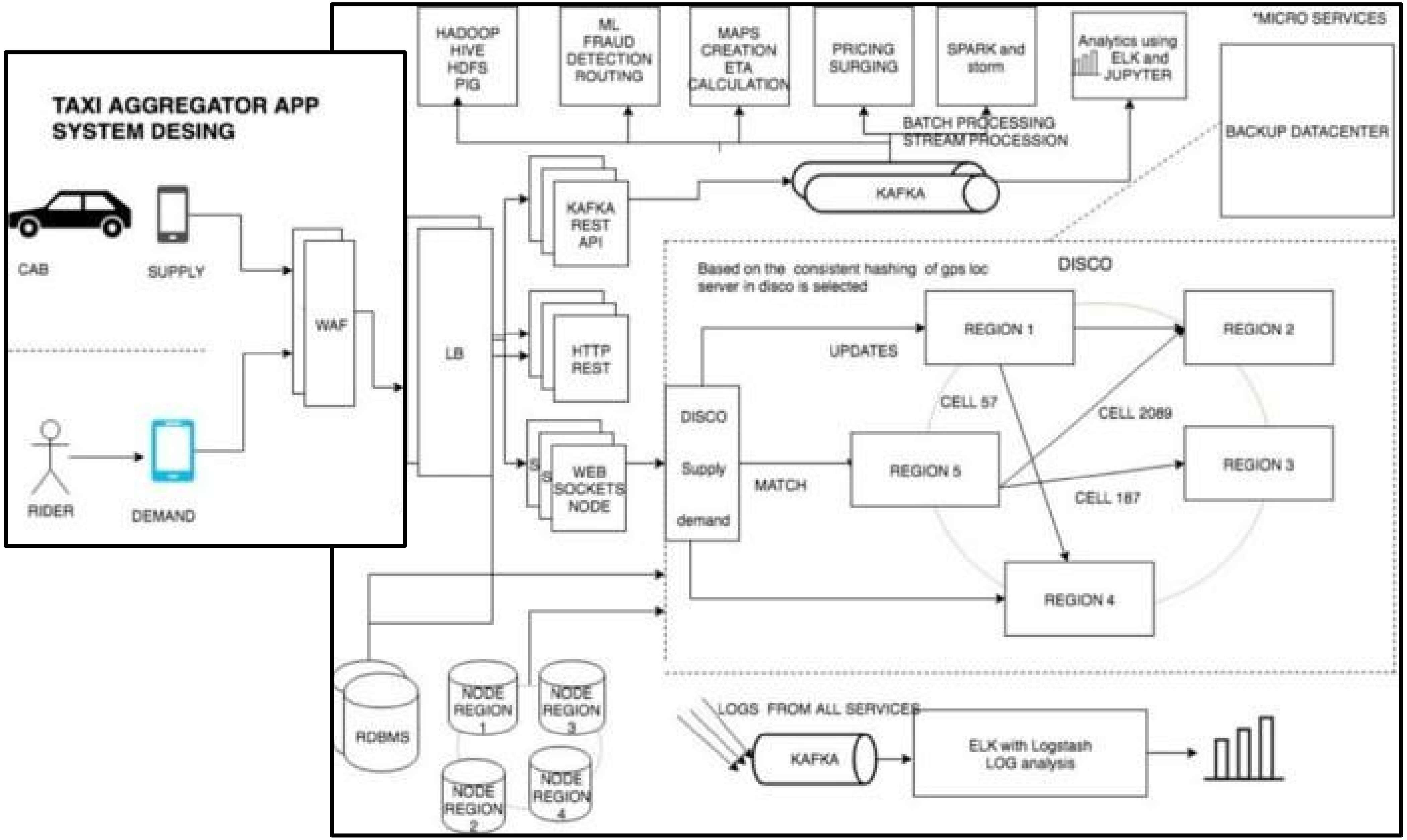
The physical view describes installation, configuration, and deployment of the software application.

It concerns itself with how to deliver the deploy-able system.

The physical view shows the mapping of software onto hardware.

The UML deployment diagrams are often used to support this view.

UBER CASE STUDY


TAXI AGGREGATOR APP SYSTEM DESING
CAB
SUPPLY
RIDER
DEMAND
WAF
LB
HADOOP HIVE HDFS PIG
ML FRAUD DETECTION ROUTING
MAPS CREATION ETA CALCULATION
PRICING SURGING
SPARK and storm
Analytics using ELK and JUPYTER
*MICRO SERVICES
BACKUP DATACENTER
BATCH PROCESSING STREAM PROCESSION
KAFKA
KAFKA REST API
HTTP REST
WEB SOCKETS NODE
DISCO
Based on the consistent hashing of gps loc server in disco is selected
DISCO Supply demand
UPDATES
MATCH
REGION 1
REGION 2
REGION 3
REGION 4
REGION 5
CELL 57
CELL 2089
CELL 187
RDBMS
NODE REGION 1
NODE REGION 2
NODE REGION 3
NODE REGION 4
LOGS FROM ALL SERVICES
KAFKA
ELK with Logstash LOG analysis

UBERS CASE STUDY

Uber’s technology may look simple but when A user requests a ride from the app, and a driver arrives to take them to their destination.

But Behind the scenes, however, a giant infrastructure consisting of thousands of services and terabytes of data supports each and every trip on the platform.

Like most web-based services, the Uber backend system started out as a “monolithic” software architecture with a bunch of app servers and a single database.

UBERS SERVICES

The challenging thing is to supply demand.

So we need two services

- Supply service
- Demand service

UBERS SERVICES

Supply service

The Supply Service tracks cars using geolocation (lat and lang).

Every cab which is active keep on sending lat-long to the server every 5 sec once.

Demand service

The Demand Service tracks the GPS location of the user when requested.

Now we have supply and demand. all we need a service which matches they demand to a supply and that service in UBER is called as *Dispatch Optimization.*

HOW DISPATCH SYSTEM WORKS? HOW RIDERS MATCH TO DRIVERS?

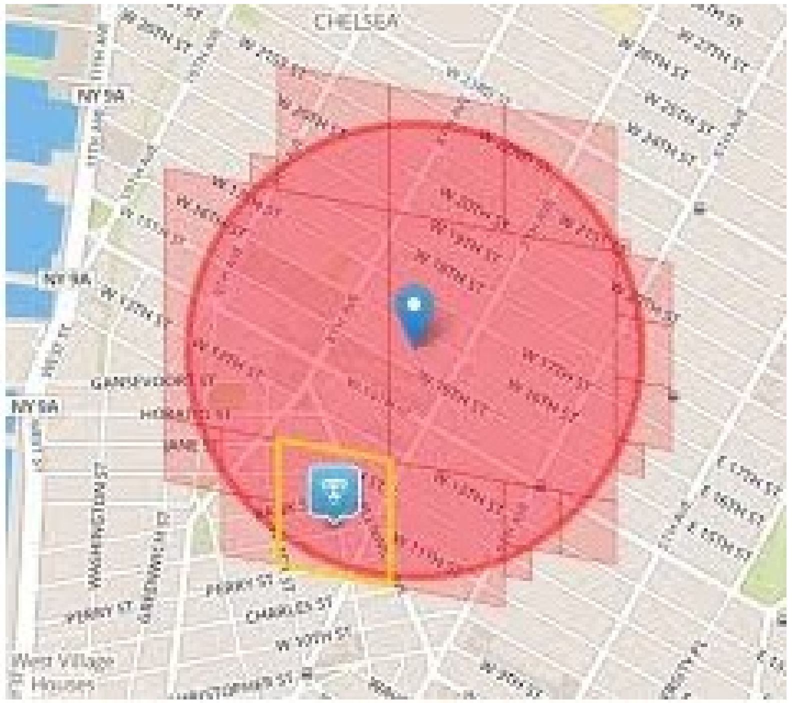
GPS/ location data is what drive dispatch system, that means we have to model our maps and location data.

The earth is a sphere. It’s hard to do summarization and approximation based purely on longitude and latitude.

HOW DISPATCH SYSTEM WORKS? HOW RIDERS MATCH TO DRIVERS?

So Uber divides the earth into tiny cells using the Google S2 library. Each cell has a unique cell ID.

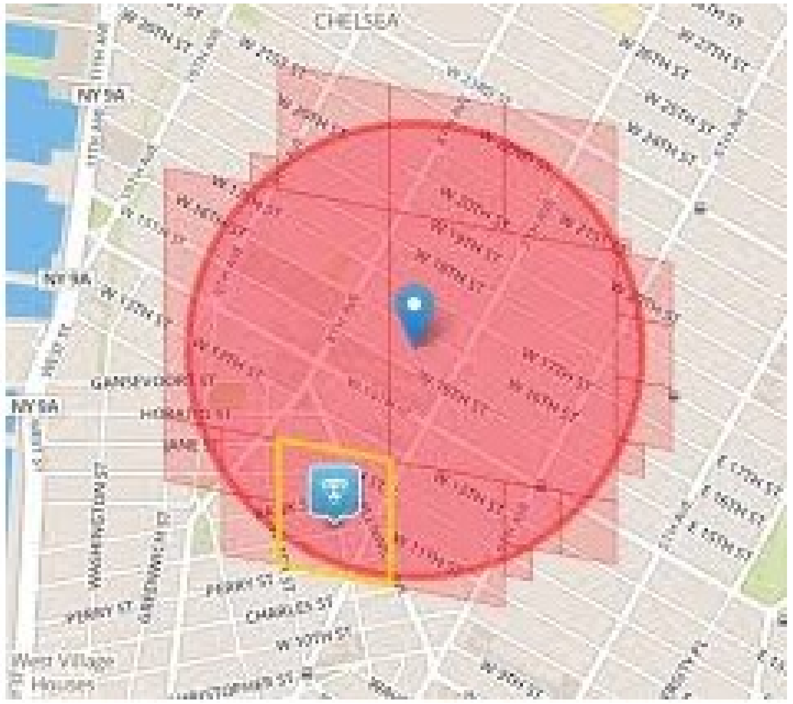
S2 can give the coverage for a shape. If you want to draw a circle with a 1km radius centered on London, S2 can tell what cells are needed to completely cover the shape.



HOW DISPATCH SYSTEM WORKS? HOW RIDERS MATCH TO DRIVERS?

ETA (Estimated Time to Arrival)

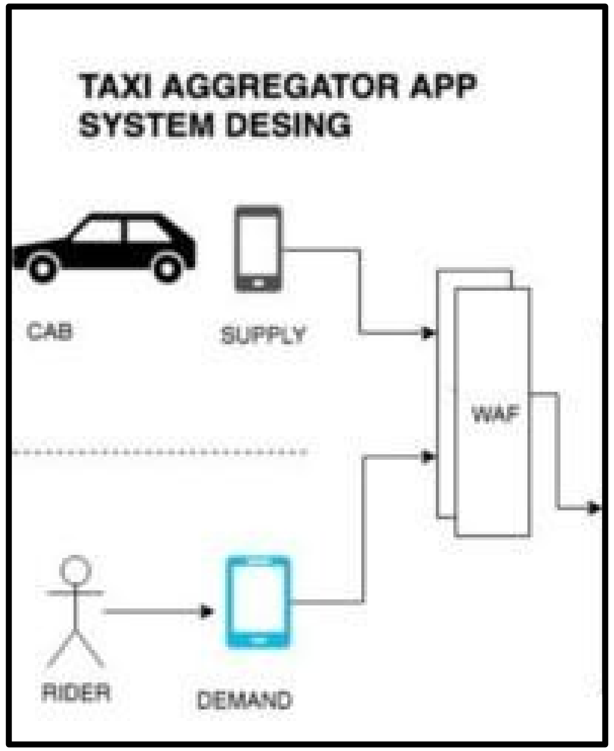
To compute the ETA of how nearby they are not geographically, but by the road system.



WEB APPLICATION FIREWALL

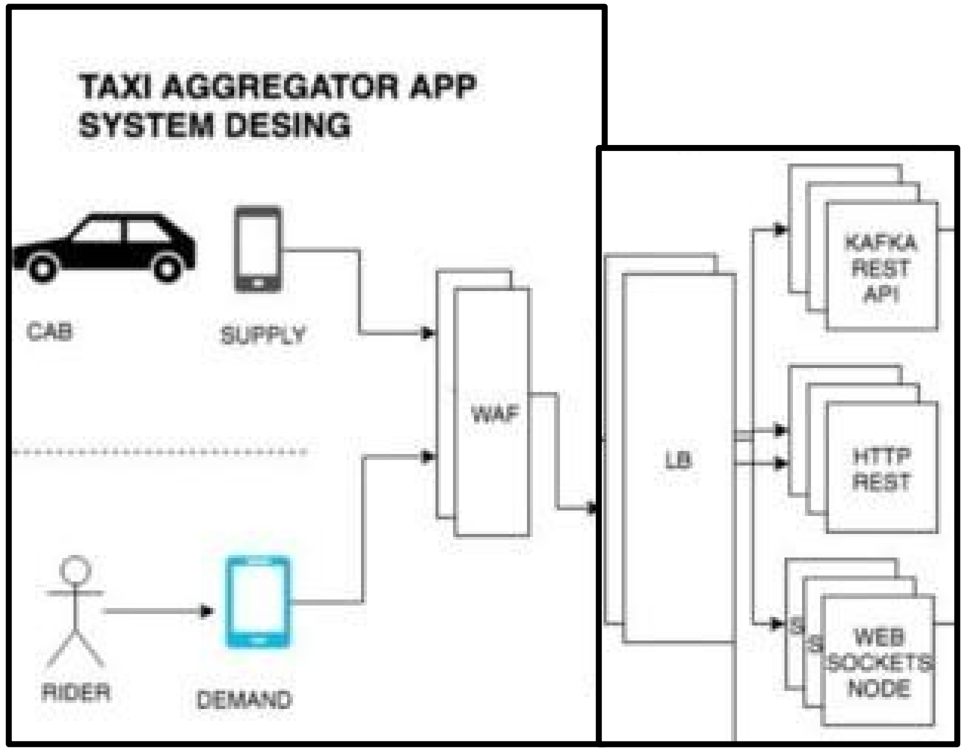
A web application firewall (WAF) is a firewall that monitors, filters and blocks requests from

- block IPs
- Bots
- Or where UBER service is not launched yet

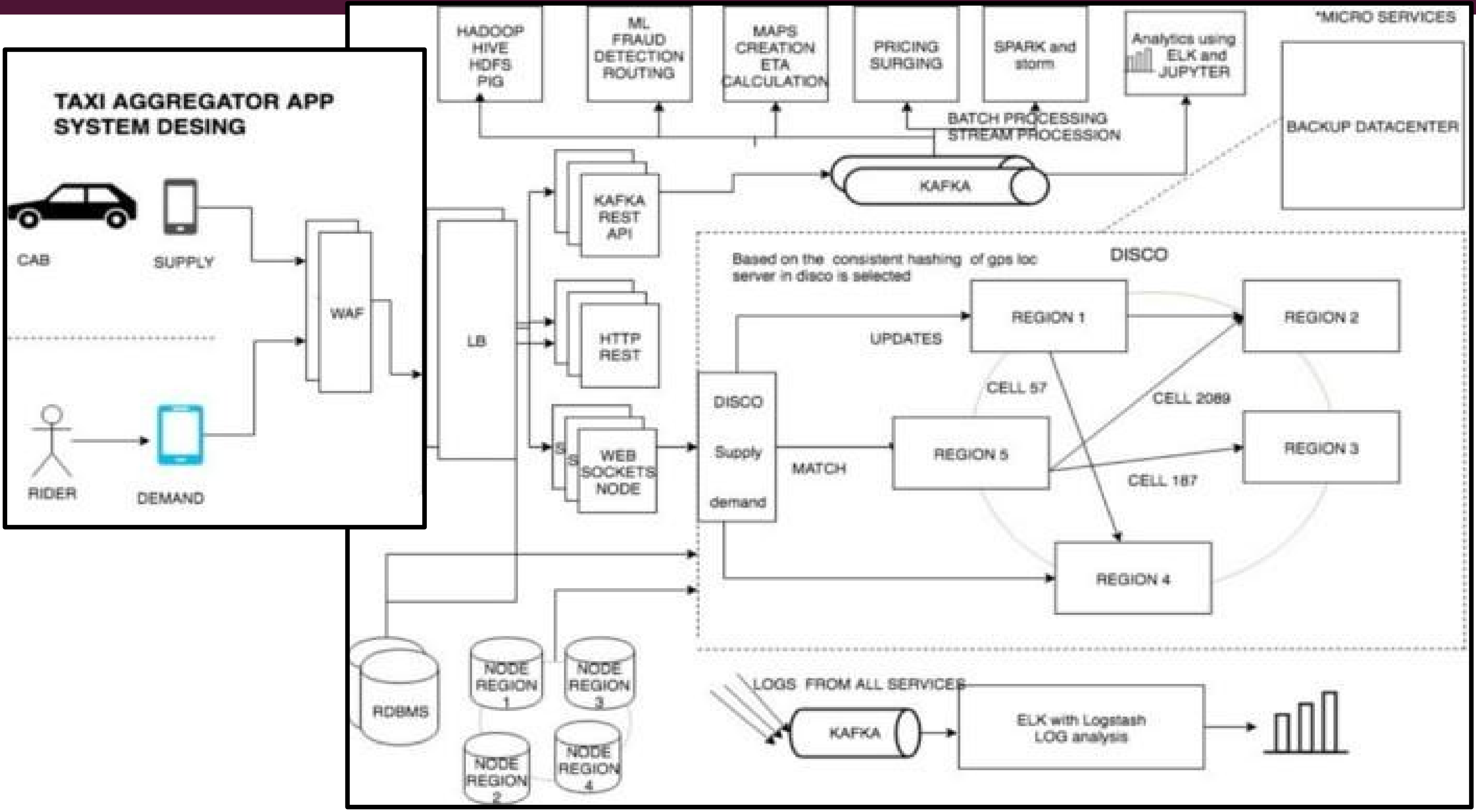


LOAD BALANCING

Load balancing refers to efficiently distributing incoming network traffic across a group of backend servers, also known as a *server farm* or *server pool.*

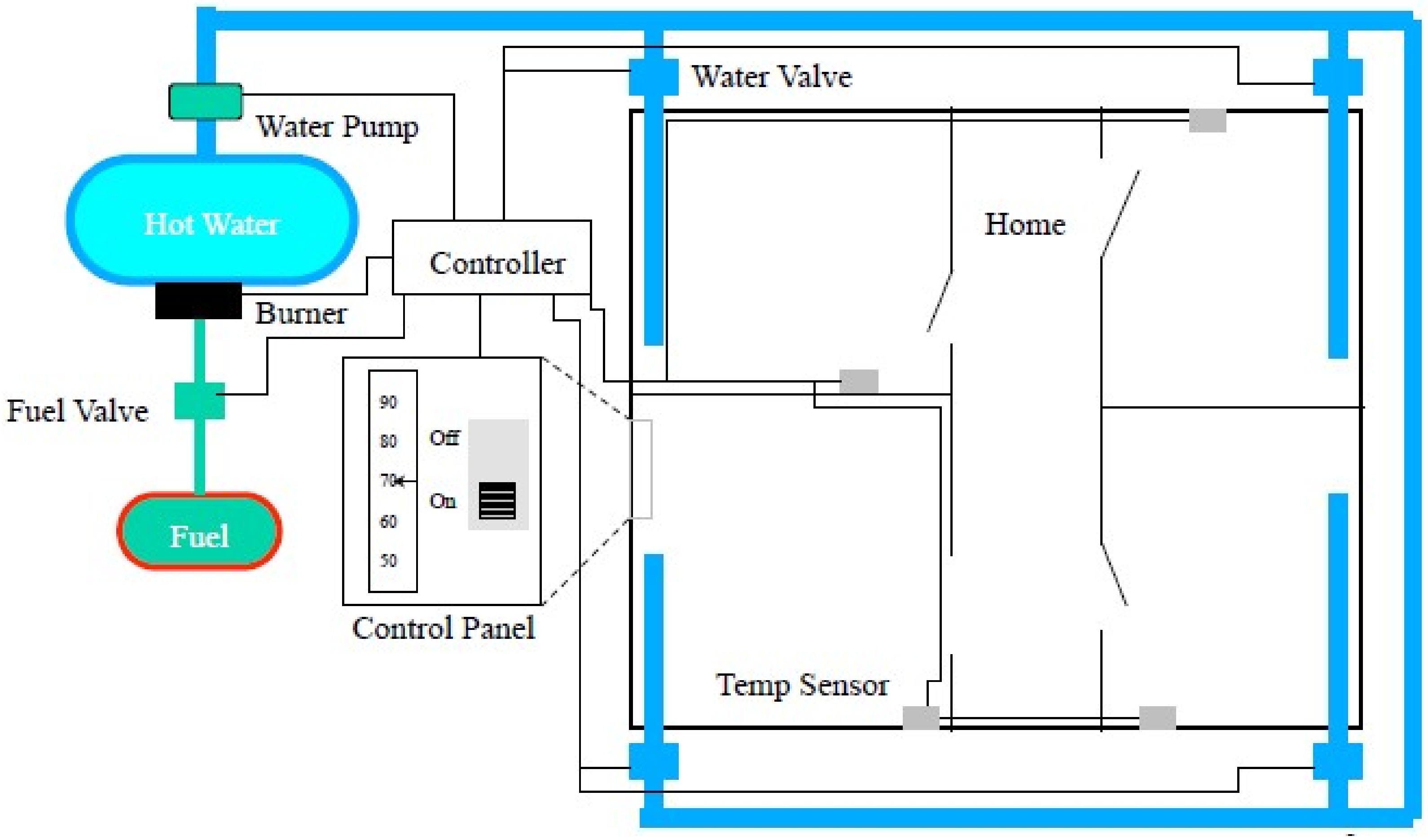


SUPPLY/ DEMAND



TOWARDS OBJECT ORIENTED DESIGN

THE HOME HEATING SYSTEM



HOME HEATING REQUIREMENTS

The purpose of the software for the Home Heating System is to control the heating system that heats the rooms of a house. The software shall maintain the temperature of each room within a specified range by controlling the heat flow to individual rooms.

HOME HEATING REQUIREMENTS

The software shall control the heat in each room

The room shall be heated when the temperature is 2F below desired temp

The room shall no longer be heated when the temperature is 2F above desired temp

The flow of heat to each room shall be individually controlled by opening and closing its water valve

The valve shall be open when the room needs heat and closed otherwise

The user shall set the desired temperature on the thermostat

The operator shall be able to turn the heating system on and off

HOME HEATING REQUIREMENTS

The <u>furnace</u> must not run when the system is off

When the furnace is not running and a room needs heat, the software shall turn the furnace on

To turn the furnace on the software shall follow these steps

– open the <u>fuel</u> <u>valve</u>

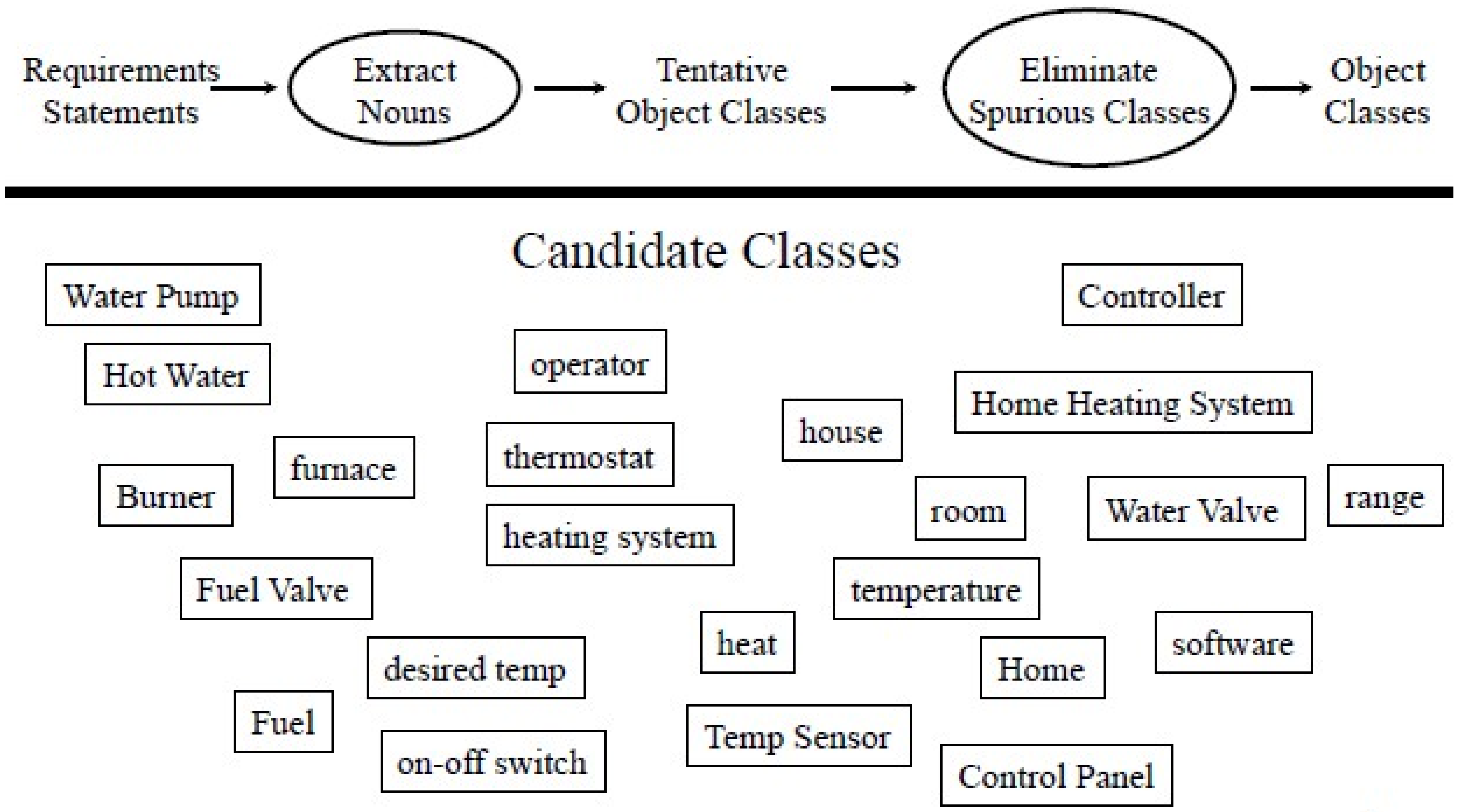
– turn the <u>burner</u> on

The software shall turn the furnace off when heat is no longer needed in any room

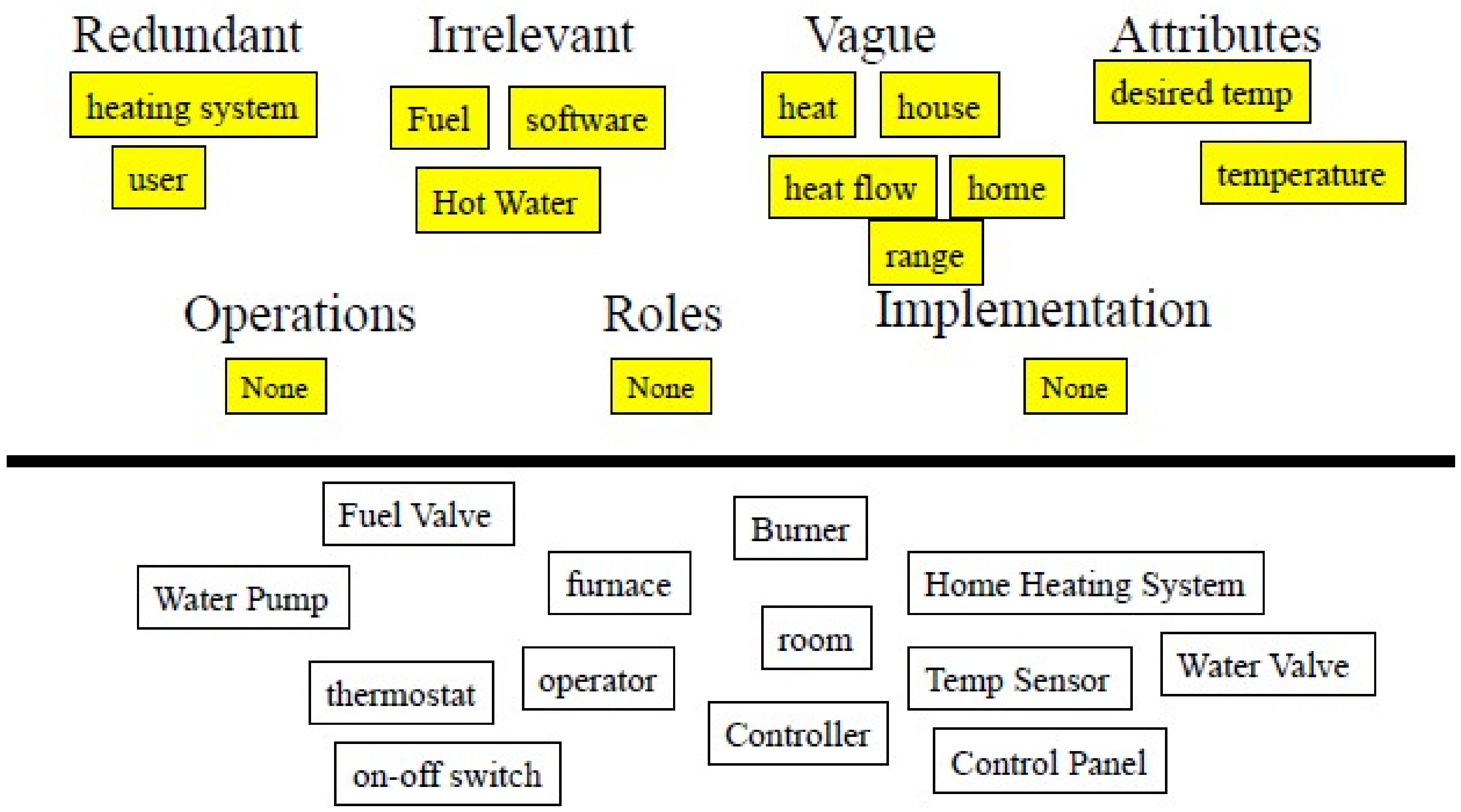
To turn the furnace off the software shall follow these steps

– close fuel valve

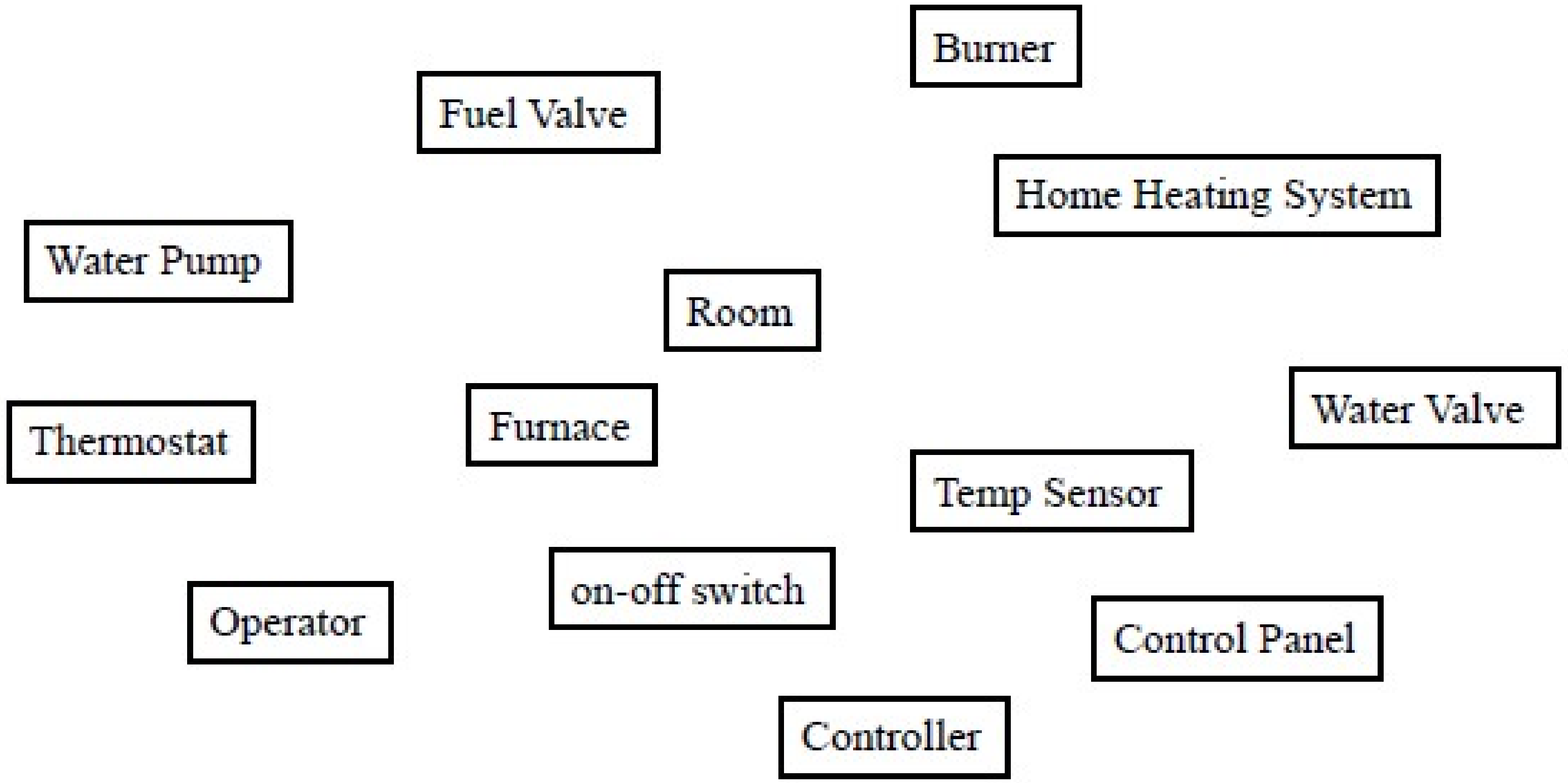
IDENTIFY OBJECT CLASSES



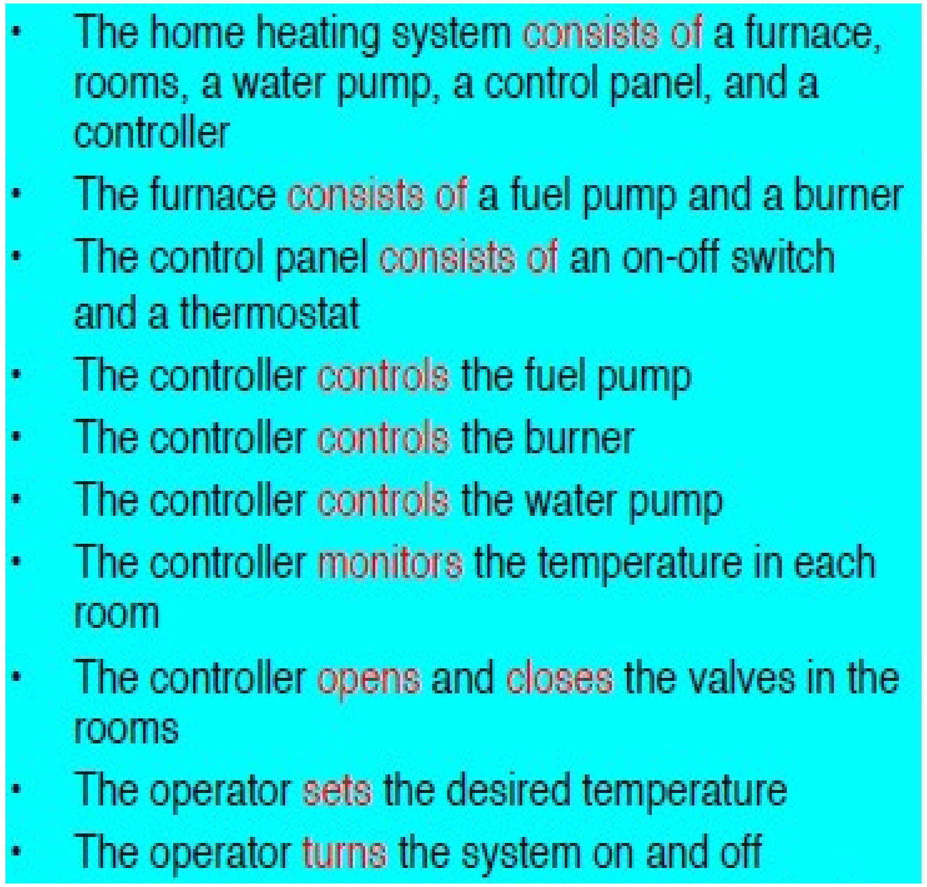
ELIMINATION



AFTER ELIMINATION

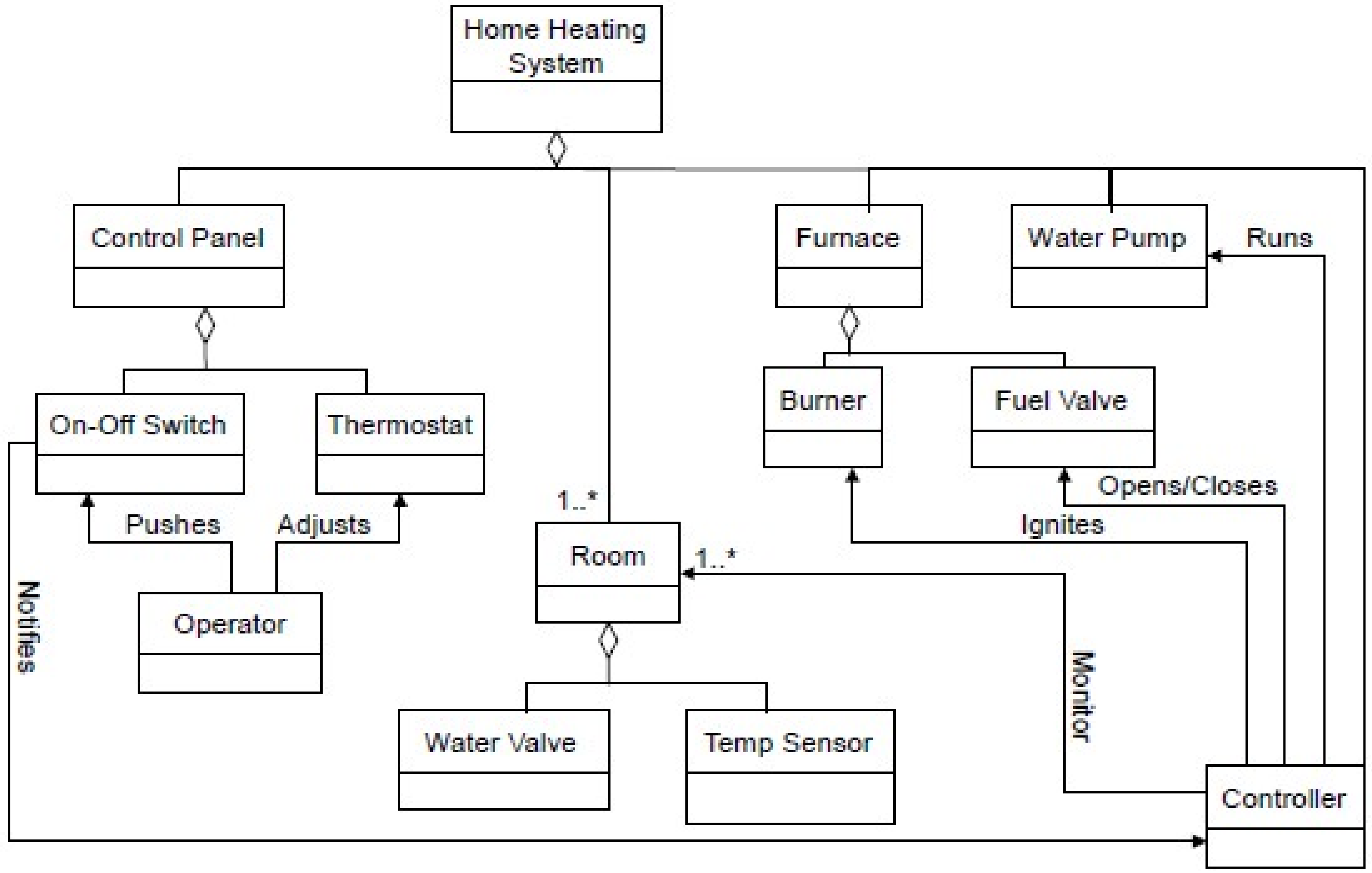


POSSIBLE ASSOCIATIONS

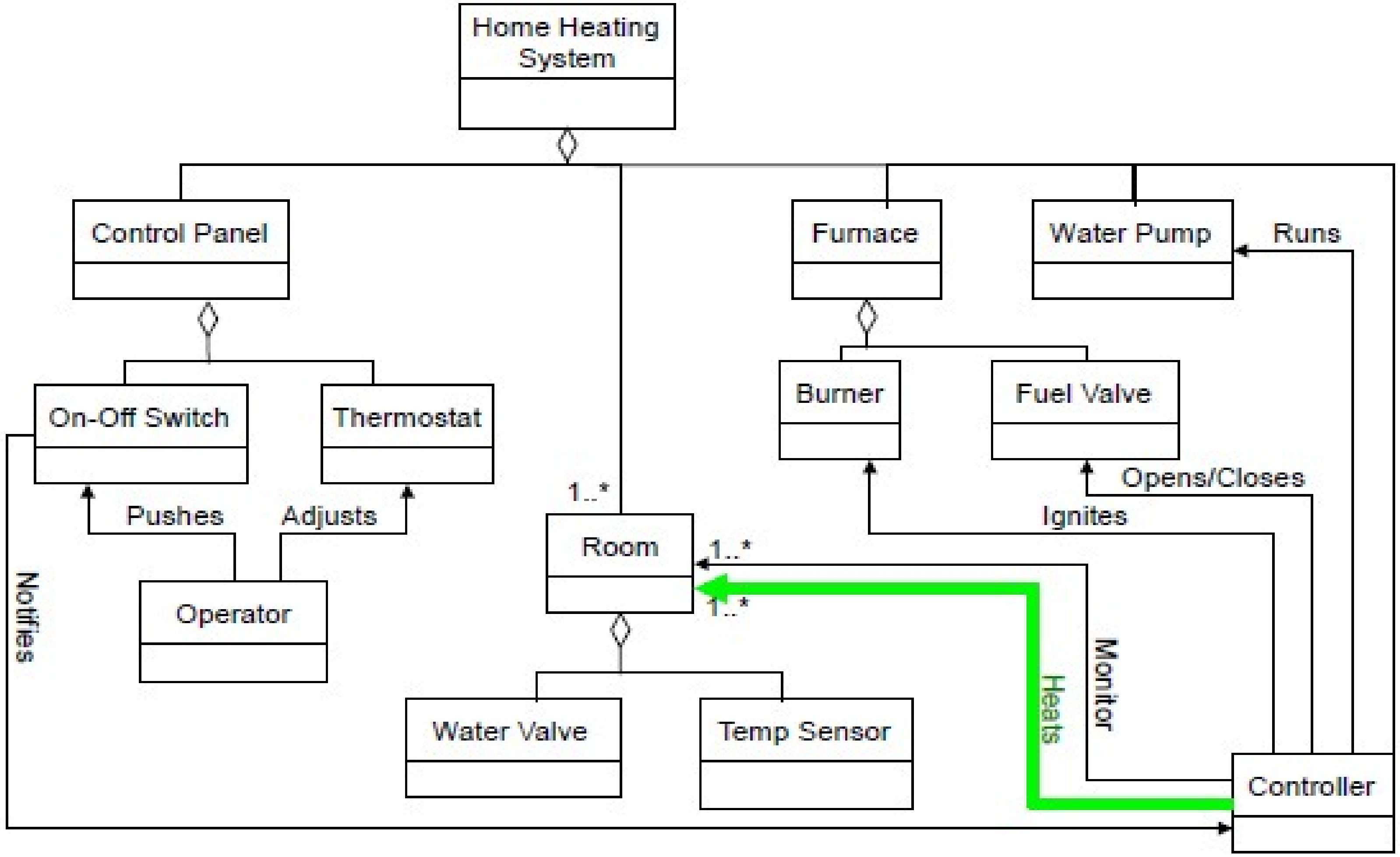


- The home heating system consists of a furnace, rooms, a water pump, a control panel, and a controller
- The furnace consists of a fuel pump and a burner
- The control panel consists of an on-off switch and a thermostat
- The controller controls the fuel pump
- The controller controls the burner
- The controller controls the water pump
- The controller monitors the temperature in each room
- The controller opens and closes the valves in the rooms
- The operator sets the desired temperature
- The operator turns the system on and off

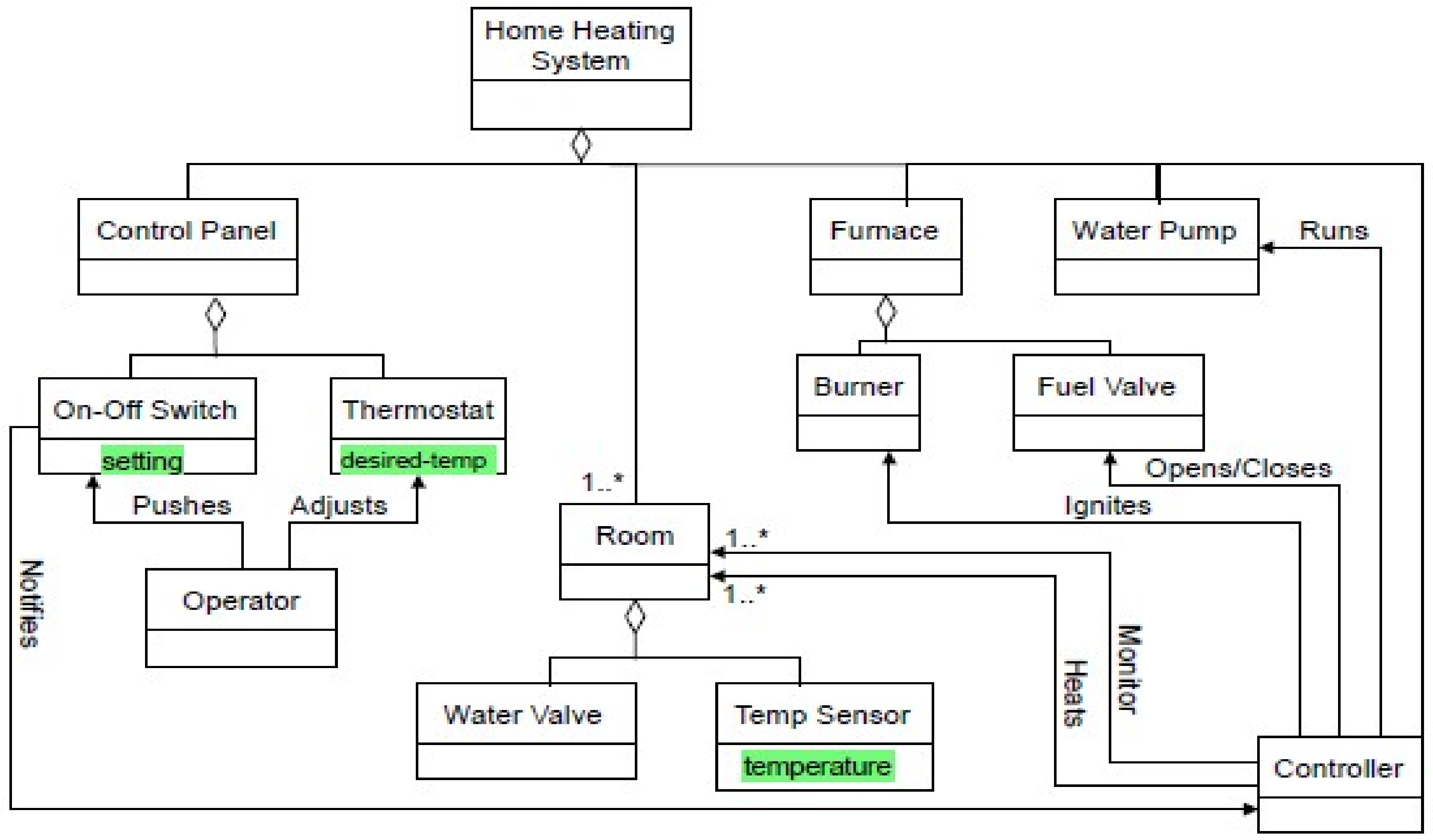
OBJECT MODEL



OBJECT MODEL - MODIFIED

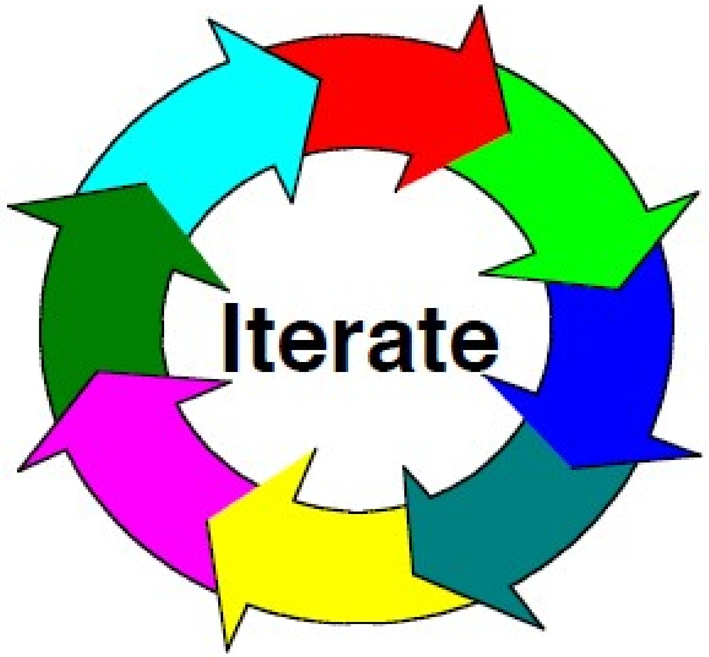


OBJECT MODEL: ATTRIBUTES

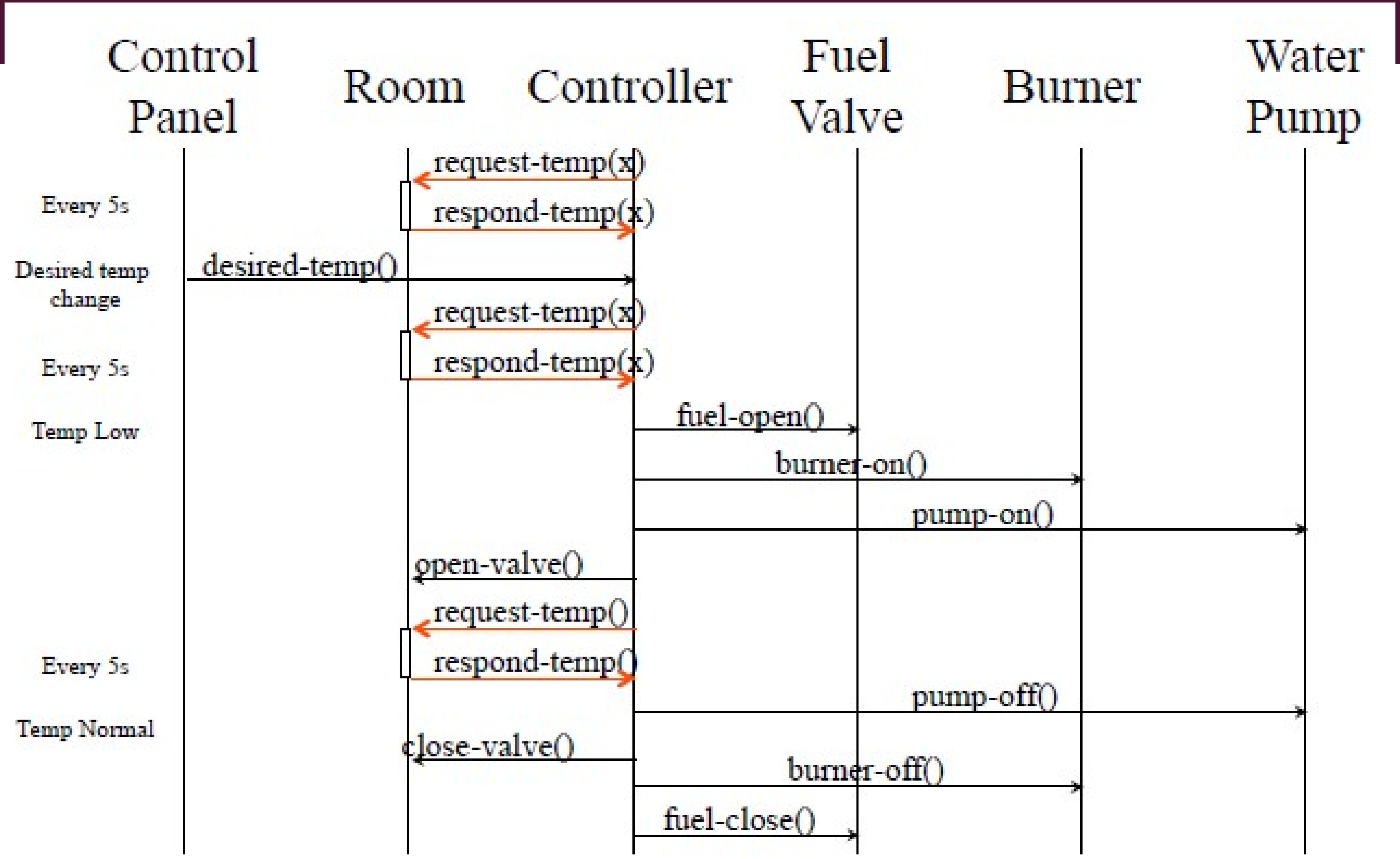


ITERATE THE MODEL

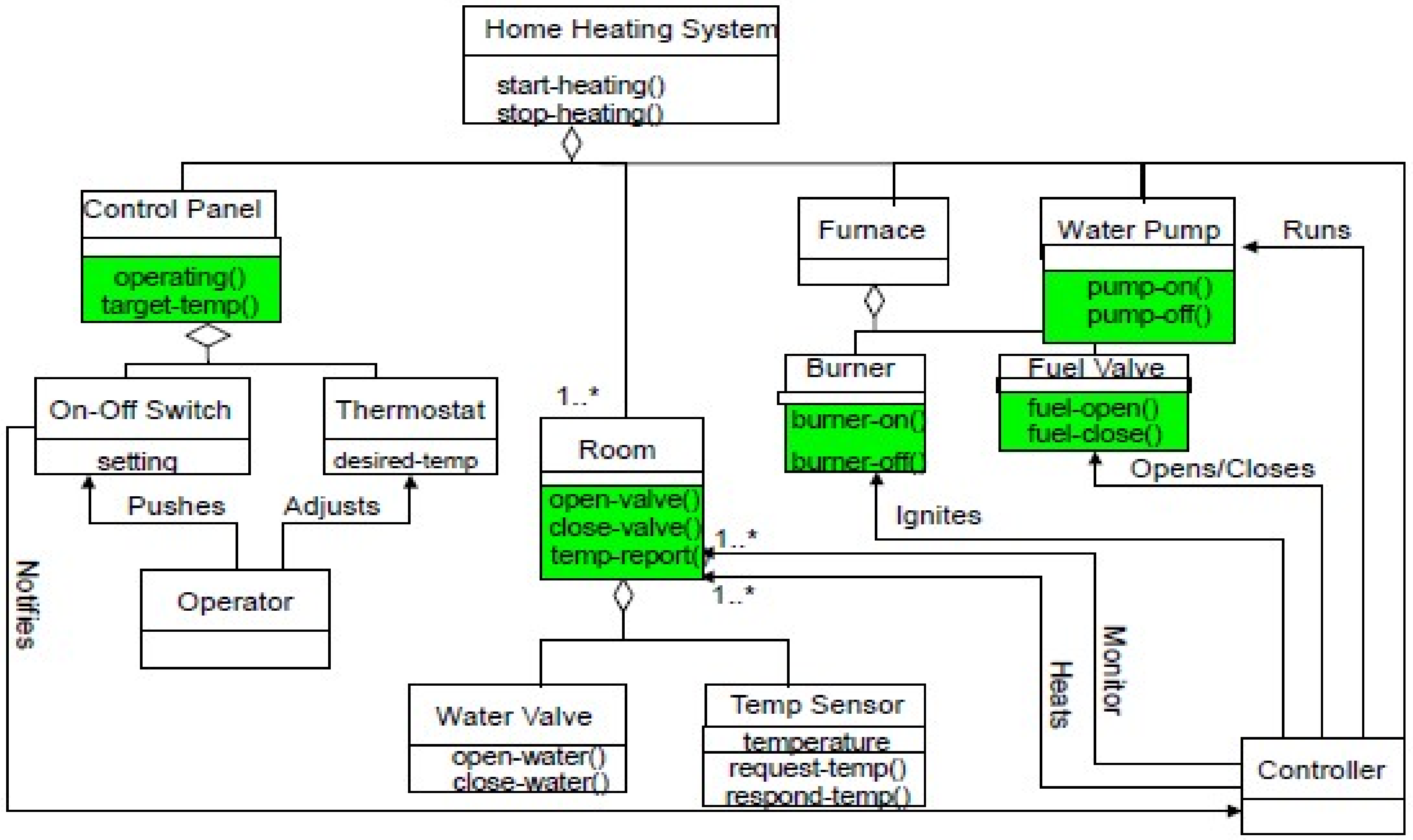
Keep on doing this until you, your customer, and your engineers are happy with the model



SEQUENCE DIAGRAM

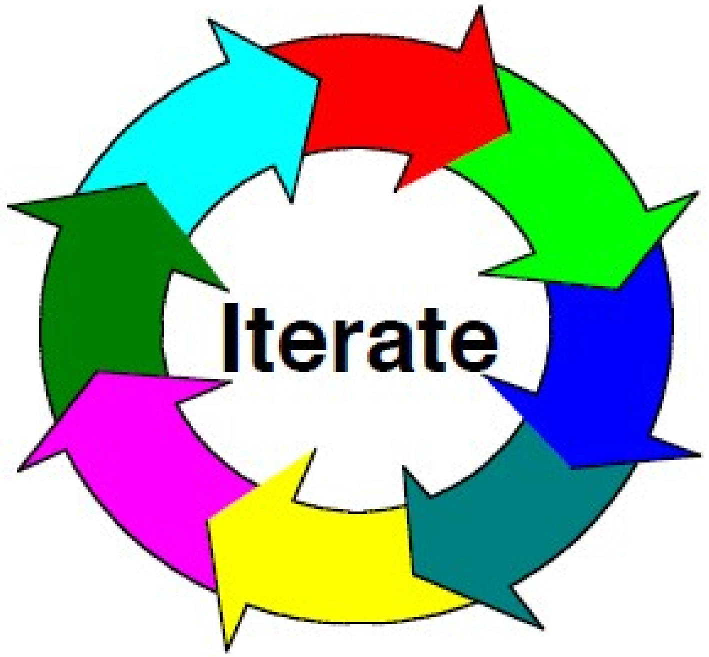


OO MODEL – MODIFIED AGAIN!



ITERATE THE MODEL

Keep on doing this until you, your customer, and your engineers are happy with the model



HAVE A GOO DAY!