INF2120 ± SPRING OF 2005

BY GROUP 6:

danielyc
sverrekt
isiry
yun
asli

TRAFIKANTEN PLUS [GROUP 6]

¼welcome to a project assigment in INF2120 maked in the spring of 2005¼

I N D E X :

1. PROJECT DESCRIBING

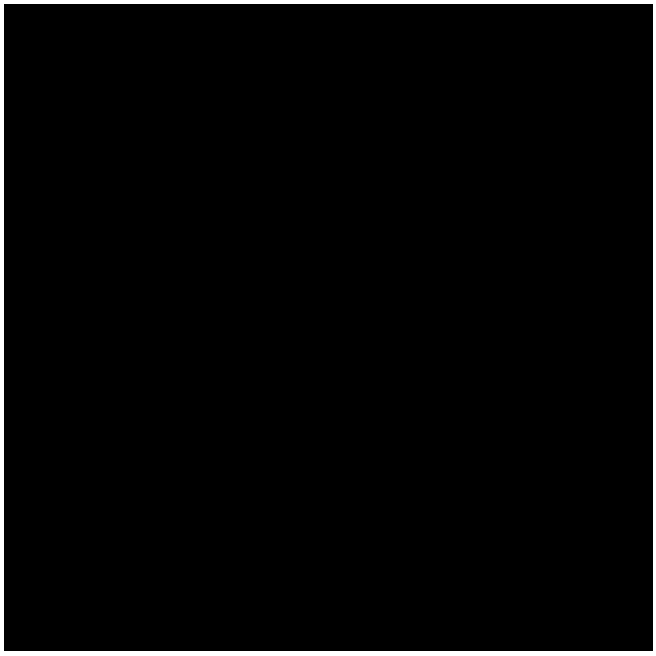
1.1 PROJECT DESCRIPTION

We took over this project from group 4 after they had made the system specification. The use-case we chose to design and implement is `Get route-time via SMS'. A SMS from a user comes in to our system via PATS, with a syntax like this: '37 helsfyr', '37 nydalen' or just `37'. In the last case we will give the user the next departure from his closest stop that bus 37 bus passes, in both directions.

1.2 CHANGES MADE FROM THE ORIGINAL SPECIFICATION

The specification group 4 had made, was way too big and complex for us to design and implement with our background in education and time to disposal. We chose to design one use-case that had to use both the static timetable, dynamic timetable from Trafikanten and the positioning that PATS offers us.

For us to design that use-case, we had to make some changes in both the class and sequence diagrams.



OLD CLASS DIAGRAM

NEW CLASS DIAGRAM

Group 4 had not thought about the fact that one line could have several routes (departures), so we fixed that by add a class called staticTimetable that had all the lines, with their routes and stops under. We also had to remove all the administrator stuff that they had, and fix a new way to handle the databases. Since we got new information about how we achieve dynamic information about the whereabouts of the transports, we added a simple realTimeDatabase class that symbols the XML document we have to parse from the Trafikanten web pages. The last bit that's new is the session class. We were notified about the multi user issue after drop one, and chose to handle that with one session object for each user.

1.3 TOOLS USED

In this system we used Eclipse as the platform, with IBM Rational Modeller. The sequence diagrams are made in MS Visio. Later we will use Eclipse to do the Java programming. The document is written in MS Word.

1.4 PRE ASSUMPTIONS

We assume that:

1. all the methods in our system always know what time it is.

2. SPECIFICATION: USE CASE, SEQUENCE- AND CLASS DIAGRAM

2.1 USE CASE DIAGRAM

We will in our system start gently with only four use cases. This is because we don't know how much work it will be to implement. If the workload is too small, we will add functionality to the system. But since we have to program everything our selves now, we want to keep things realistic. To start with we have four functions:

- Get Route Time via SMS
  - The user sends a simple message to the system, where he/she asks for the next departure for a specific route. The system finds the users position and the nearest stop, and then the next departure there. If the bus is delayed, the user will be notified.

2.2 USE CASE DESCRIPTIONS

Get Route Time via SMS	
ACTOR:	Service User
PRE-CONDITION:	• User's mobile has GSM function • Id of transport is unique • All stations are in the system • Dynamic positions of all transports are in the system • Timetable of all transport are in the system
POST-CONDITION:	• User get a message from the system about next arrival time of the transport to the nearest station
TRIGGER:	System got an asking message from user about next departure times for a specific route on the nearest stop

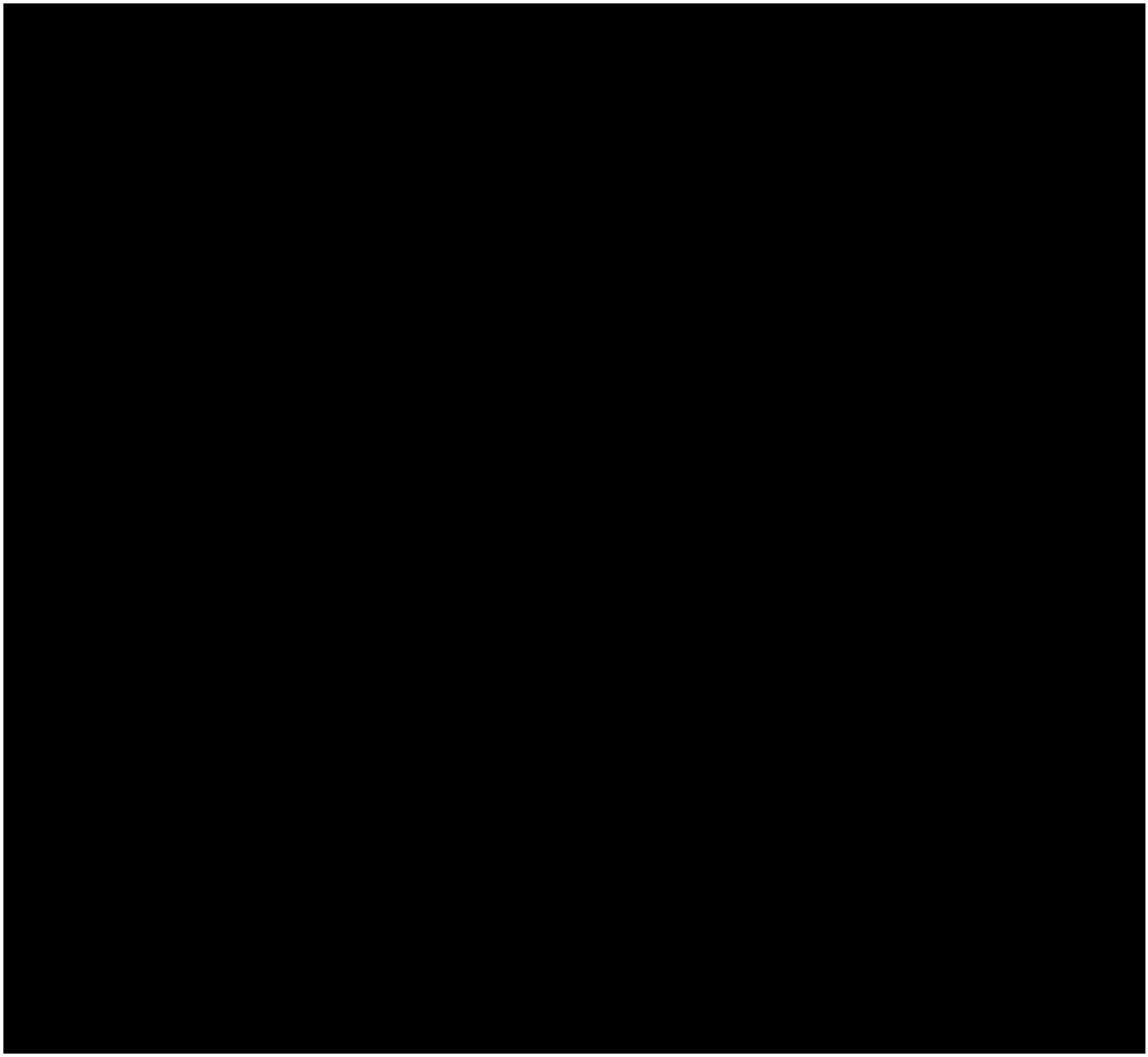
NORMAL COURSE OF EVENTS

1. User provides desired route to system.
2. System locates user and nearest stop.
3. System finds next departure from static database
4. System locates where the transport is and calculate any delay.
5. System sends an SMS with next departure and delay (if any).

VARIATIONS

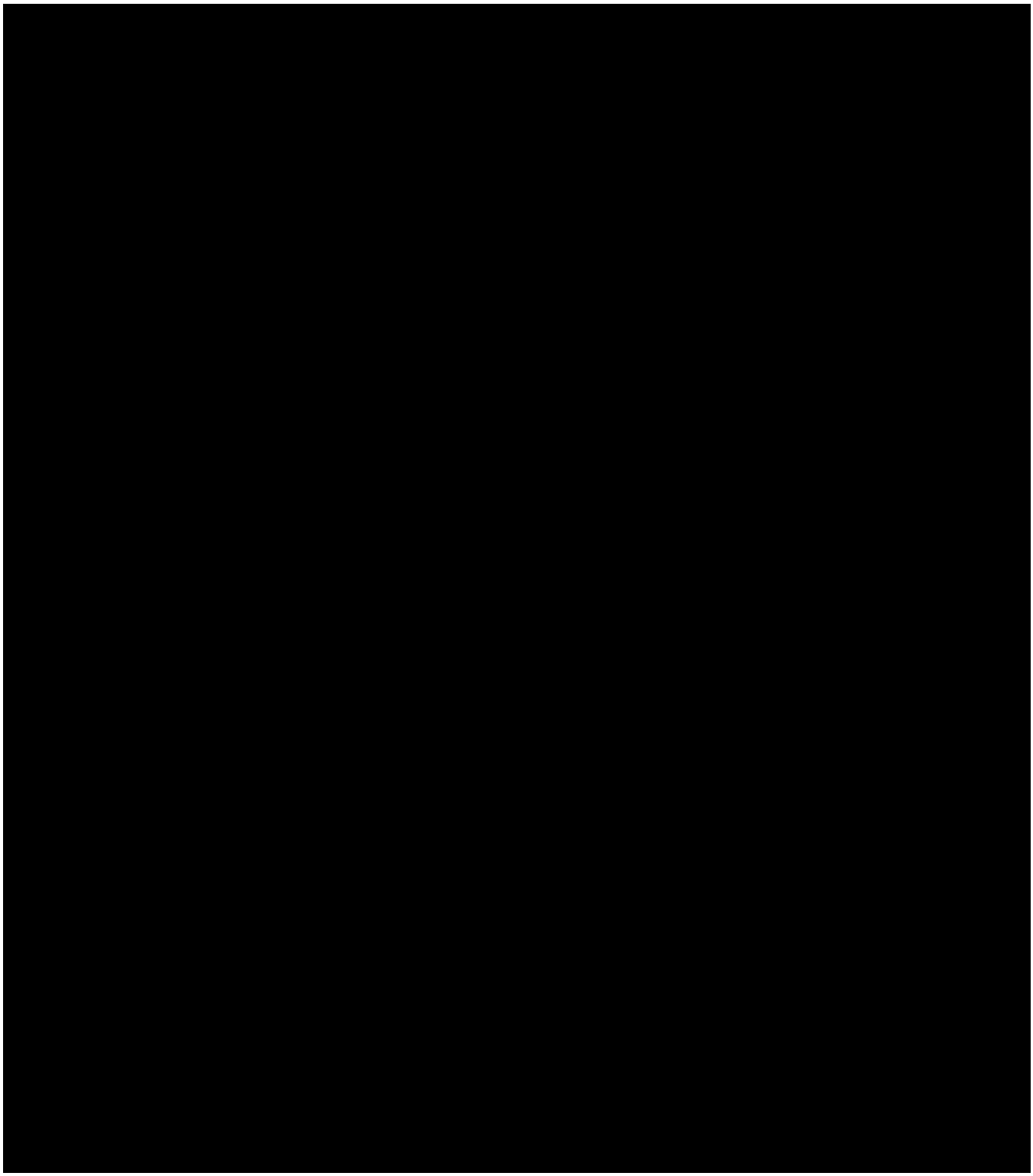
1a. The route provided is not valid.
2. System returns error message.

2.3 UPDATED CLASS DIAGRAM

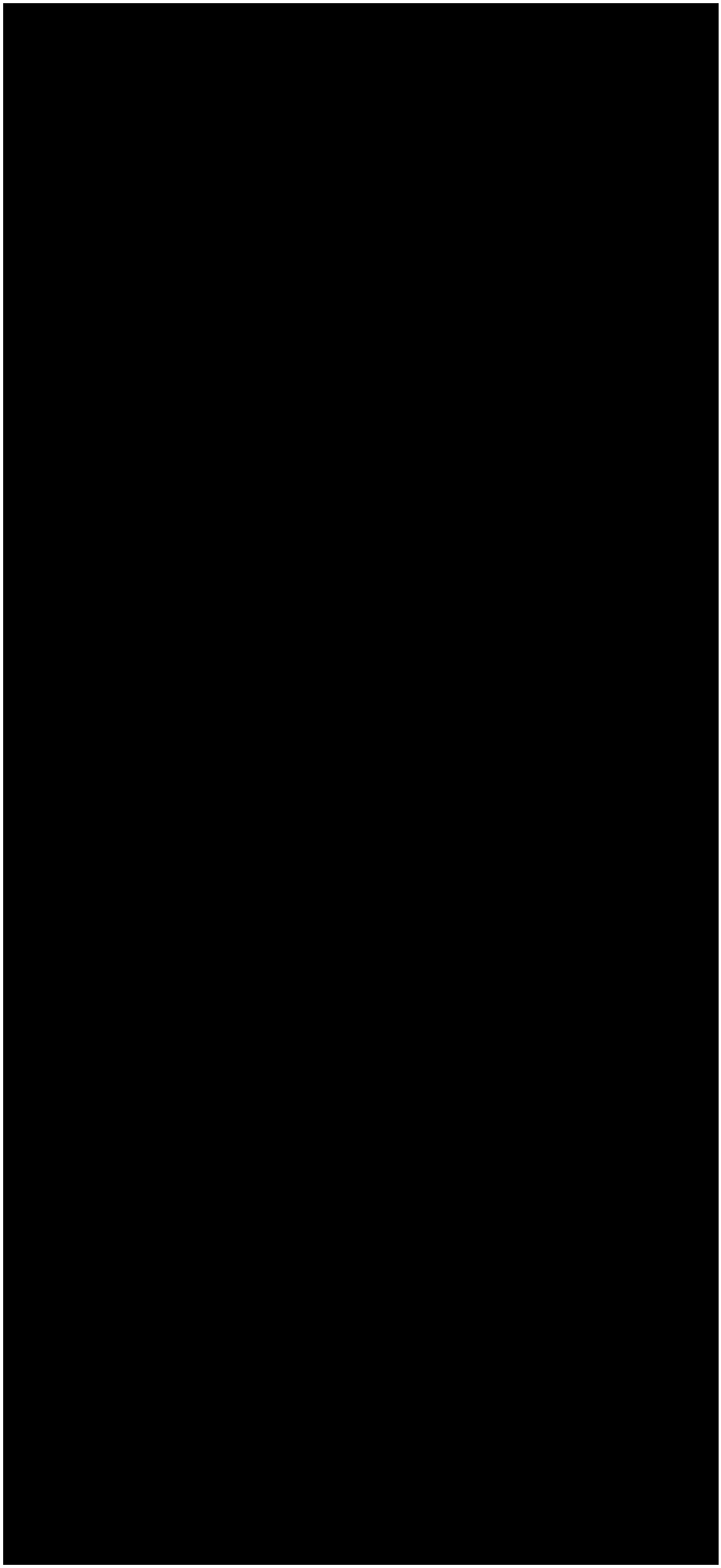


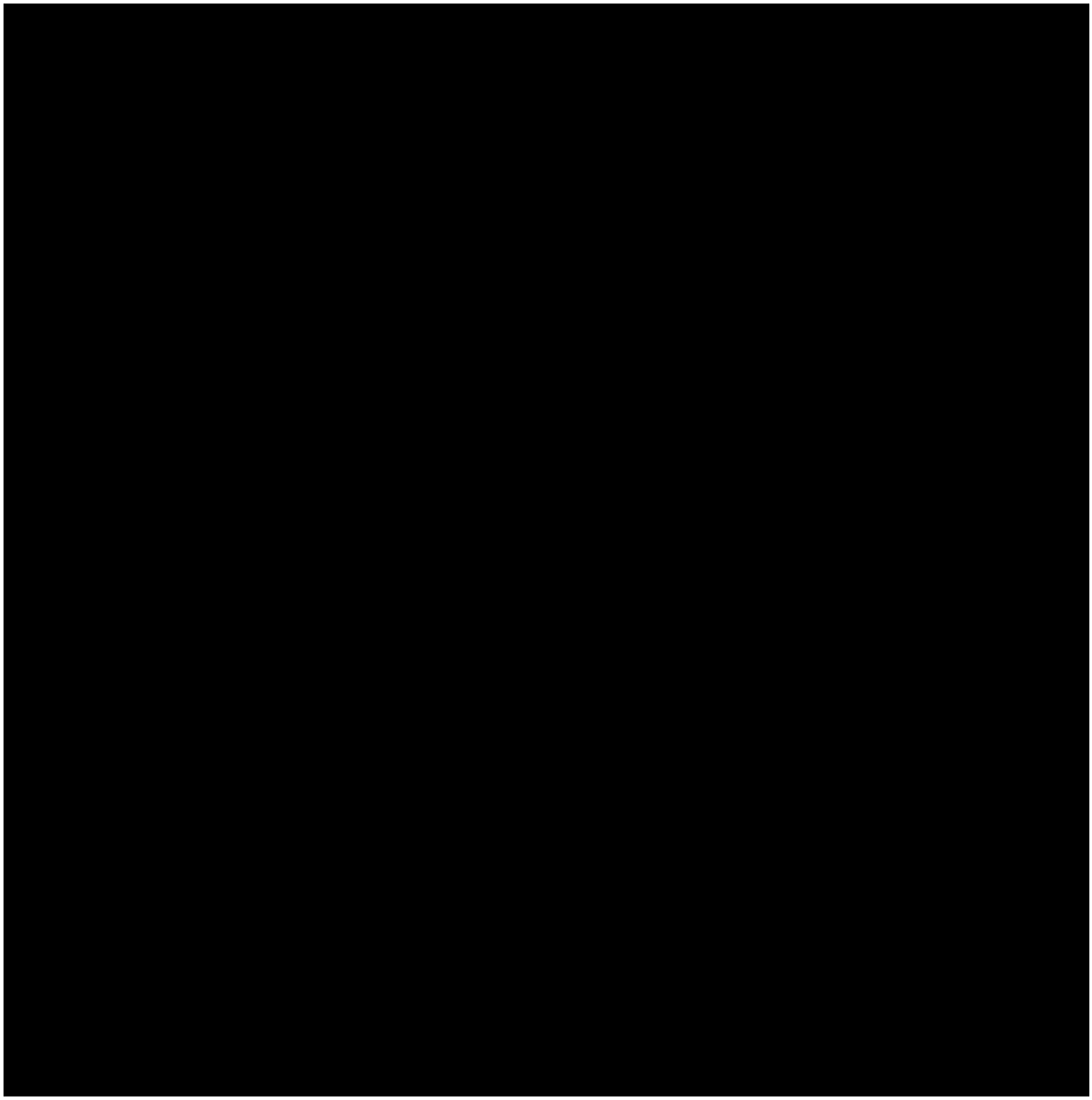
As explained above, this is the updated class diagram. RealTimeDatabase symbolizes the XML document Trafikanten offers us. Static Timetable will be responsible for importing and creating the data structure for all the static information about lines, routes and stops. The session object will get and fill out all information needed to send a SMS, so that the system can handle several users simultaneously. From PATS we get a library of functions we use to send and receive messages with. System plays the role as the central for everything in our system. It also has control over all sessions made.

2.4 SEQUENCE DIAGRAMS



Here is a sequence diagram that covers three user examples, two with correct SMS format, and one with wrong format. You may also notice that the session who asks for information first, isn't necessary the one that gets the answer first. The session object terminates when all the information is gathered. In the next pages you will see decomposition of session and static time table.





3. DESIGN: COMPOSITE STRUCTURE AND STATE MACHINES

3.1 COLLABORATION DIAGRAM

Here we see all the main parts of the system, and their communication directions. The information flow goes something like this:

1. The user sends a SMS to PATS and the system gets the mission from PATS .
2. The system gets the data which the user ask for, from the static time table, and compare the data from real time table database and calculate if the transport is delayed.
3. When all the information is gathered, PATS gets the required information from the system and PATS send the information back to the user.

3.2 COMPOSITION DIAGRAM

The diagram shows what parts from the collaboration diagram that has sub-classes.

3.3.1 COMPOSITE STRUCTURE: SYSTEM

The system coordinates with all the interactions between PATS, staticDB, realTimeDB and the session through the ports.

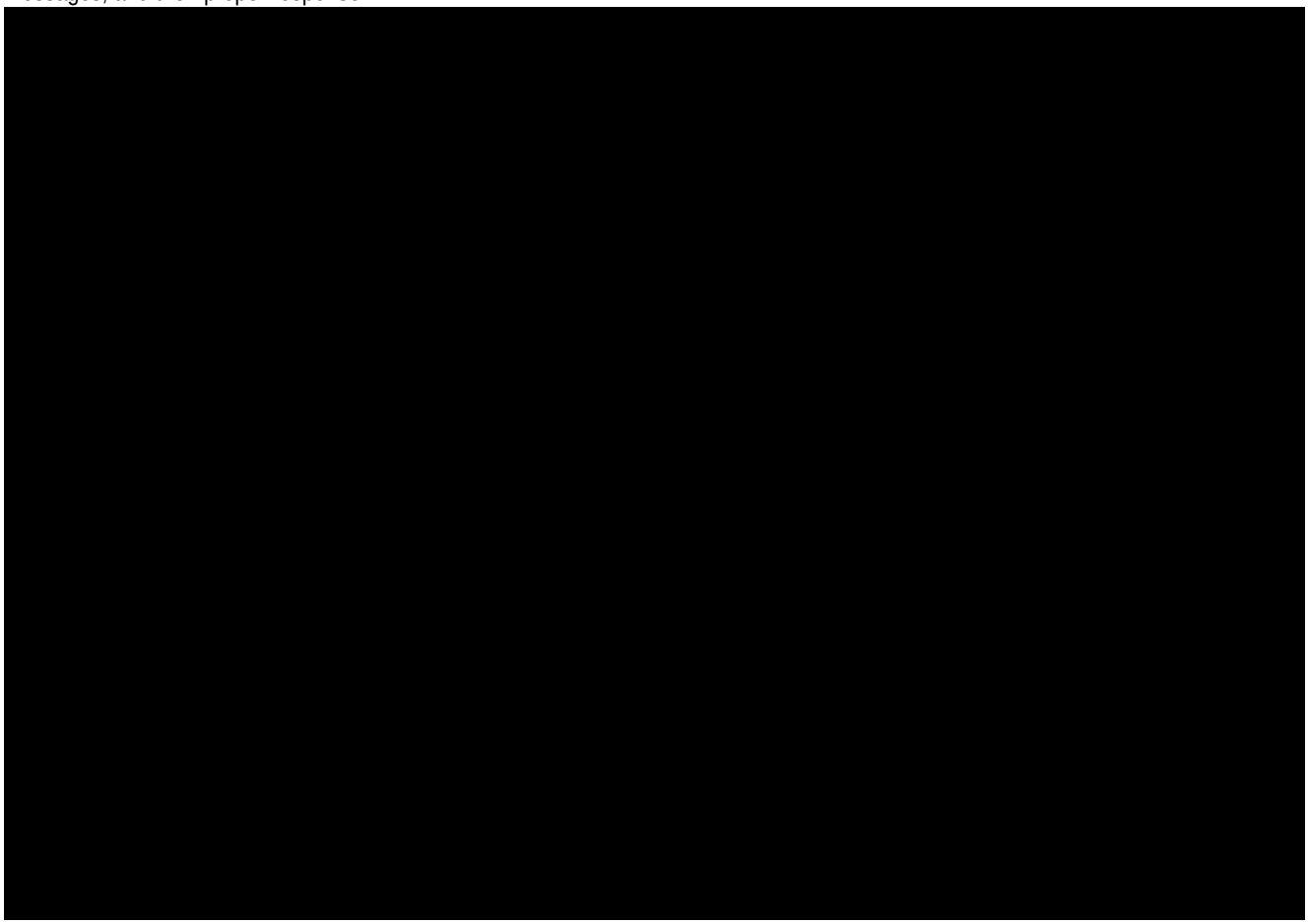
3.3.2 COMPOSITE STRUCTURE: STATIC TIMETABLE

3.4 STATE MACHINES

Session is created when the system gets an SMS message from PATS. If the format is wrong, or the line does not exist, an error message is sent, and session terminates. In the other case, all the information is gathered, and an SMS will be sent to PATS. The entry points show where the states gets its information needed to proceed to the next states in the machine.

We could have written code to such methods as validateLine and getPosition, but we chose not to. This is because the methods would have been pretty straight forward, and the diagram would have been harder to follow. Since drop 3 is implementation, we had to write the code manually anyway, so that's the reasoning behind it all.

The system is always in an idle position because it is receiving and sending information back and forward from the static database, real time database and PATS. The system needs to be able to handle incoming messages at any time. We here show all possible incoming messages, and their proper response.



Here the line is in an idle position waiting request from the static database.

Here the route is in an idle position waiting for request from the line.

Here the stop is in an idle position waiting for request from the line.