Chapter 1

Prototype: DDS

1.1 Design and architecture

Setting up 2 publishers and 2 subscribers will give an idea of how a DDS system can be used. In this example the two cities Gotham and Metropolish acting as publishers, have citizens reporting different kind of emergencies. Citizen can, as topic, report a bar fight, a blood clot, a break in and a paper cut. The Police Station and the Medic Station acts as subscribers to the events but will not respond to any call.

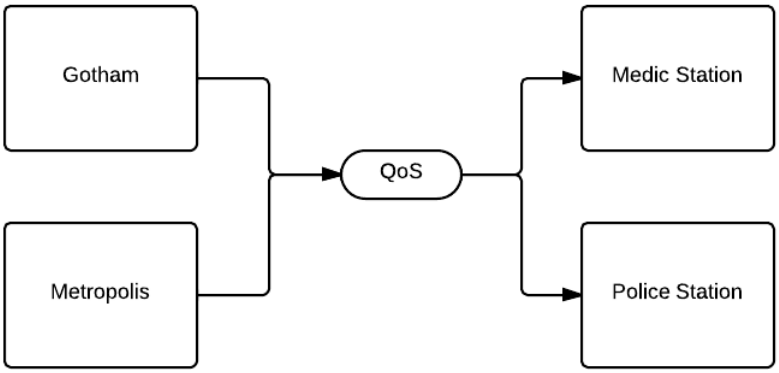

Figure 1.1: text

The QoS registers all emergencies and the sends the messages to the stations as shown in Table 1.1

	Medic Station	Police Station
Paper cut	-	-
Break in	-	+
Bar fight	+	+
Blood clot	+	-

Table 1.1: Stations will move out to '+' but not '-'

As shown in Table 1.1 the Medic Station will move out to bar fights and blood clots but no paper cuts and break ins. The Police Station will however move out to break ins and bar fights but not blood clots and paper cuts.

The program first start the two subscribers and afterwards the two publishers. Typing in a category, a message and a key in the publisher will send it to the QoS and let it show in the subscribers if they subscribe to the category as shown in Figure 1.2.

Attached to the document is the project files, where all the `.sh`-files are compiled. Running the 2 subscribers will let them start listing for topics. Running the 2 publishers let the user decided which topics they will publish. After the topics have been selected the publishers can sends message's key and the message's actual message.

When the message is sent it will written in the console of the subscriber(s).

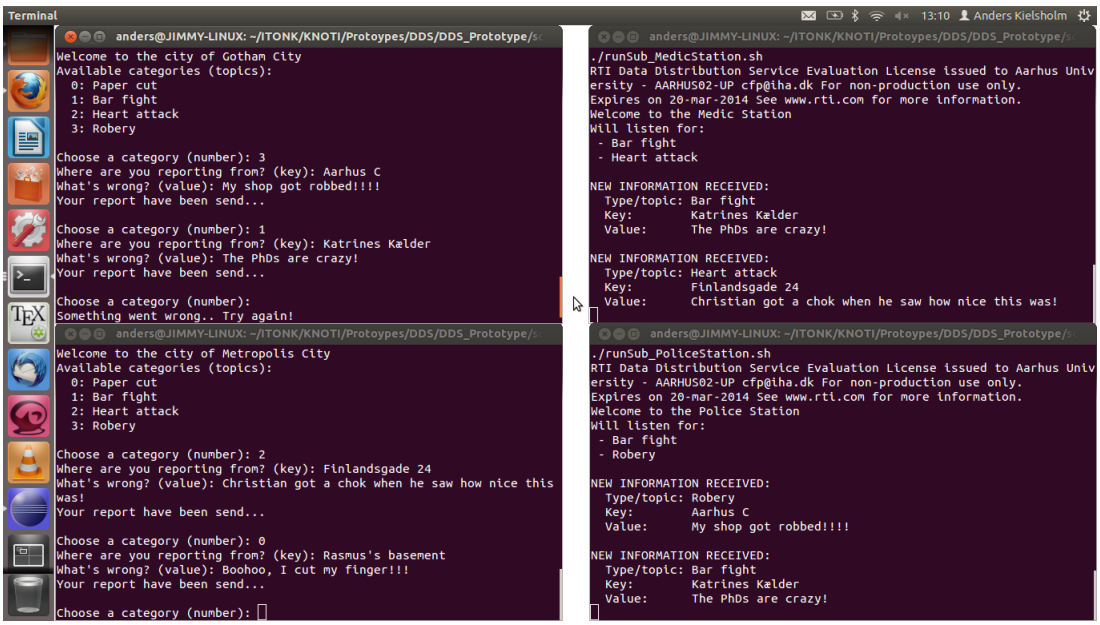

Figure 1.2: text

1.2 The implementation

The implementation is split up in to four files, each providing its own service for the program to start.

- **RTIHelper.java**
  Provides different helper methods for both the publisher and the subscriber
- **ReportListener.java**
  Provides the listener functionality to the middleware to send the data for the subscriber.
- **ReportSubscriber.java**
  Takes a name for a station and optional numbers of topics to subscribe to as argument. Will call the `ReportListener.java` to attach it self.
- **CityPublisher.java**
  Takes one argument as a name of the city. Creates a topic and a datawriter from `ReportListener.java` and reads user input for messages.

The `RTIHelper` contains four methods:

Code snippet 1.1: RTIHelper: getDomain()

```
static DomainParticipant getDomain() {
  return DomainParticipantFactory.get_instance().create_participant(
      0, // Domain ID = 0
      DomainParticipantFactory.PARTICIPANT_QOS_DEFAULT,
      null, // no listener
      StatusKind.STATUS_MASK_NONE);
}
```

Code snippet1.1 creates a new domain participant which is the reference to the domain, here with the `ID` predefined to '0'. Furthermore it specifies which QoS must be attached to the participant though the `PARTICIPANT_QOS_DEFAULT` is chosen here for simplicity. The domain has no listeners and therefore the mask is `STATUS_MASK_NONE`.

Code snippet 1.2: RTIHelper: createTopic

```
static Topic createTopic(DomainParticipant domain, String topicStr) {
  return domain.create_topic(
      topicStr,
      KeyedStringTypeSupport.get_type_name(),
```

```
      DomainParticipant.TOPIC_QOS_DEFAULT,
      null, // no listener
      StatusKind.STATUS_MASK_NONE);
}
```

Code snippet1.4 creates a new topic that can be used for both publishers and subscribers. The only thing that differs the different topics in the prototype are their names, defined for the first parameter. This is the one equal to e.g. "Robery" or "Bar fight". Via the last parameters it is told that the topic is bound to the KeyedString type, the default QoS because of simplicity is used for creating the topic and that there like for the DomainParticipant won't be any listeners.

Code snippet 1.3: RTIHelper: createKeyedStringWriter

```
static KeyedStringDataWriter createKeyedStringWriter(DomainParticipant domain, ↩
    Topic topic) {
  DataWriter writer = domain.create_datawriter(
        topic,
        Publisher.DATAWRITER_QOS_DEFAULT,
        null, // no listener
        StatusKind.STATUS_MASK_NONE);
  return (KeyedStringDataWriter) writer;
}
```

Code snippet 1.4: RTIHelper: createKeyedStringReader

```
static KeyedStringDataReader createKeyedStringReader(DomainParticipant domain, ↩
    ReportListener listener, Topic topic) {
  DataReader reader = domain.create_datareader(
        topic,
        Subscriber.DATAREADER_QOS_DEFAULT,
        listener,         // Listener
        StatusKind.DATA_AVAILABLE_STATUS);
  return (KeyedStringDataReader) reader;
}
```