Events API based on Broker/Channel API

Specification

Overview

A `MessageQueue` is a peer-to-peer communication established between two tasks.

Binding to another broker

Signature: `boolean bind(int port, AcceptListener listener)`

First, you have to create a listener geared to listen to accepting incoming brokers:

```
interface AcceptListener {
void accepted(MessageQueue queue);
};
```

This listener sends a signal entitled `accepted` which returns back a `MessageQueue` used to establish the peer-to-peer connnection with the other remote `MessageQueue`. To reach this, you have to pass an instance of `AcceptListener` into the `bind` method from your task's broker. Here is an example of usage:

```
class AListener implements QueueBroker.AcceptListener {
    @Override
    public void accepted(MessageQueue queue) {

        System.out.println("I well received the connected queue");

    }
}
// Create one instance of the accepting listener...
AListener l = new AListener();
// ... and send it to the QueueBroker
Task.getQueueBroker().bind(PORT, l);
```

Cancelling a binding request

Signature: `boolean unbind(int port)`

When you have waited a little too long and want to cancel a earlier binding request on a certain port number, you can call the method `unbind` to cancel any binding request you sent earlier on the same broker.

Connecting

Signature: `boolean connect(String name, int port, ConnectListener listener)`

First, you have to create a listener geared to listen to connecting incoming brokers:

```
interface ConnectListener {
void connected(MessageQueue queue);
void refused();
};
```

This listener can send two different signals: `connected` and `refused`. The second signal is invoked when the remote broker is neither available nor connected. The first signal returns back a `MessageQueue` used to establish the peer-to-peer connnection with the other remote `MessageQueue`. To reach this, you have to pass an instance of `ConnectListener` into the `connect` method from your task's broker.

Design

This application is based on the Broker/Channel API lastly programmed in SAR. To mimic the behavior of an event pump, each method from this application launches a new thread doing operations while the main thread is back free to the user. Once a thread finishes to process, it calls a function from the given listener, as simply as it, no need to implement an event pump? Here is a global layout of the design:

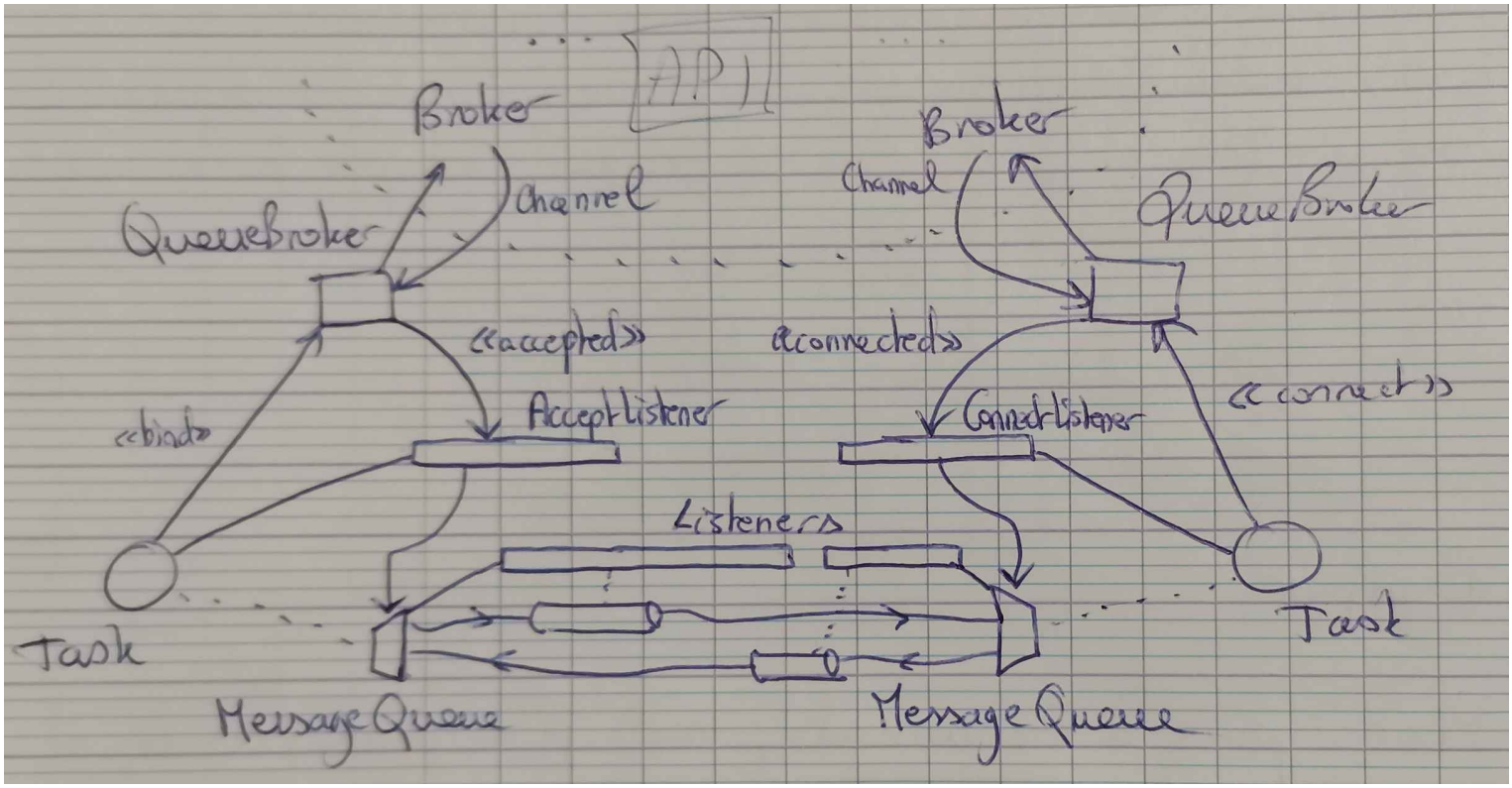

Figure 1: Design