RobMoSys-1FORC

RoQME

Dealing with Non-Functional Properties Through Global Robot Quality of Service Metrics

RoQME DDS data space

THIS PROJECT HAS RECEIVED FUNDING FROM THE *EUROPEAN UNION'S HORIZON 2020 RESEARCH AND INNOVATION PROGRAMME* UNDER GRANT AGREEMENT NO. 732410, IN THE FORM OF FINANCIAL SUPPORT TO THIRD PARTIES OF THE ROBMOSYS PROJECT

Content

IMPORTANT: The RoQME DDS support libraries are part of the RoQME distribution, which is available in GitHub as a free and open source licensed software. It can be downloaded at: https://github.com/roqme/robmosys-roqme-itp

Introduction

This document is available as a reference for the use of the DDS topics defined as part of the RoQME data space. For further information regarding the DDS distribution used in RoQME and how to install it, please refer to the installation of RoQME itself, available in GitHub.

RoQME DDS topics

The topics used within the RoQME data space are defined in a corresponding IDL file. Basically, two topics are used, one for the propagation of context values (RoqmeContext) and one for the observations in CEP (RoqmeCEPResult).

```
module RoqmeDDSTopics
{
    typedef sequence<string> contextData;
    enum contextType {
              NUMBER_TYPE,
              BOOL_TYPE, ENUM_TYPE, EVENT_TYPE, NUMBER_V_TYPE, BOOL_V_TYPE, ENUM_V_TYPE,
       EVENT_V_TYPE}; //The XXX_V_TYPE are vectors

    struct RoqmeContext
    {
        string name;
        contextType type;
        contextData data;

    };
#pragma keylist RoqmeContext name

    enum effectType {UNDERMINES, REINFORCES};
    enum impactType {HIGH, LOW, VERY_HIGH, VERY_LOW};


    struct RoqmeCEPResult
    {
        string property;
        effectType effect;
        impactType impact;
    };
#pragma keylist RoqmeCEPResult property
};
```

Using the RoqmeDDS library

Class `RoqmeDDSManager` is responsible for the creation of writers and readers (both for contexts and CEP observations):

```
        void createWriterContext(...); //creates a publisher for contexts
        void createWriterCEPResult(...); //creates a publisher for CEP observations
        void createReaderContext(...); //creates a subscriber for contexts
        void createReaderCEPResult(...); //creates a subscriber for CEP observations
```

The API has the same interfaces for Java and C++, so we are going to illustrate its use through an example of publisher/subscriber implemented in Java.

```
import Roqme.RoqmeDDSException;
import Roqme.RoqmeDDSManager;
import RoqmeDDSTopics.RoqmeContext;


public class RoqmePublisher {
    public static void main(String[] args){
        /* RoqmeDDSManager is part of the Roqme package, whereas topics are part of the RoqmeDDSTopics package*/

        try{
                RoqmeDDSManager manager = new RoqmeDDSManager();
                manager.createWriterContext();

                int i = 0;
                while(true){
                  System.out.println("Sending message..." + i++);

                // Data topic creation
                RoqmeContext cxtmsg = new RoqmeContext();
                cxtmsg.name = "sensor";
                cxtmsg.type = RoqmeDDSTopics.contextType.EVENT_TYPE;
                cxtmsg.data = new String[] {"alarm"}; /*this is a vector of strings. However, because
                it was defined as of type EVENT_TYPE, it contains only one element */
                //context publication
                manager.write(cxtmsg);
                Thread.sleep(1000);
            }

         }
         catch(RoqmeDDSException e){
             e.printStackTrace();
         }
         catch (InterruptedException e) {
             e.printStackTrace();
         }

         System.out.println("Bye!");
    }
}
```

The try/catch blocks are important to get errors due to invalid operations. For instance, a write() attempt without a prior creation of the corresponding writer throws a RoqmeDDSException.

The subscriber for a RoQME topic is created first using the `RoqmeDDSManager`, and then providing it with a `listener` implementing `RoqmeDDSListener<T>` (with T being a roqme defined topic):

```
import Roqme.RoqmeDDSListener;
import RoqmeDDSTopics.*;


public class EsperListener implements RoqmeDDSListener<RoqmeContext>{
    @Override
    public void onDataAvailable(RoqmeContext msg){
        System.out.println("--------------");
        System.out.println("Received:");
        System.out.println("\tname:" + msg.name);
        System.out.println("\ttype: " + msg.type.value());
        int i = 0;
        for(String str : msg.data){
            System.out.println("\tdata["+ (i++) + "]:" + str);
        }
    }
}


import Roqme.RoqmeDDSException;
import Roqme.RoqmeDDSManager;


public class RoqmeSubscriber {
    public static void main(String[] args){
        try{
              RoqmeDDSManager manager = new RoqmeDDSManager();
              //Asynchronous reading
              manager.createReaderContext(new EsperListener());
              Thread.sleep(1000000);
        }
        catch(RoqmeDDSException e){
            e.printStackTrace();
        }
        catch(InterruptedException e){
            e.printStackTrace();
        }
        System.out.println("End!");
    }
}
```

Advanced configurations

Configuring the RoqmeDDSManager

By default, the number of elements stored in a writer/reader queue are unlimited. However, this can lead to memory issues and exhausted resources without a proper implementation of control flow policies in the sender and receiver applications. Therefore, the RoqmeDDSManager allows developers to configure the queue maximum data length. For instance, it we want to store a maximum of one hundred topic data, we will instantiate our manager as:

```
RoqmeDDSManager manager = new RoqmeDDSManager(100);
```

Defining partitions

The DDS specification allows developers the creation of partitions, which are differentiated data spaces to share topics without collisions. They are useful is many situations and our DDS library supports them. In order to create a writer or reader that publish or subscribes data using a specific partition, you must provide an optional argument with the partition name as a text string when you call the createXXX() methods in RoqmeDDSManager:

```
RoqmeDDSManager roqme;
roqme.createReaderContext(new MyEsperEvents(),”contextsPartitionName”);
roqme.createWriterCEPResult(“CEPobservationsPartitionName”);
```