INtegrated TOol chain for model-based design of CPSs

INTO-CPS External Data Broker FMU (RabbitMQ FMU)

Version: 0.01

Date:

The INTO-CPS Association

http://into-cps.org

Contributors:

Kenneth Lausdahl, AU
Casper Thule, AU

Editors:

Casper Thule, AU
Kenneth Lausdahl, MjInformatics A/S

Document History

Ver	Date	Author	Description
0.01	January 13, 2019	Casper Thule	Initial Version.

Abstract

TBD

Contents

1 Introduction

The RabbitMQ FMU is an implementation of an external data broker based on RabbitMQ. The RabbitMQ FMU subscribes to a RabbitMQ queue and publishes the message content via its outputs. This way, the RabbitMQ FMU brings data from any source into an FMI simulation context. Note, that this document assumes general knowledge of the Functional Mock-up Interface.

The approach is depicted in fig. 1, where the content within the *System Specific* frame will vary based on the system providing the data. The Log-Translator entity translates the system-specific log messages to RabbitMQ messages and publishes them to a RabbitMQ queue, which RabbitMQ FMU is configured to subscribe to. This means that RabbitMQ FMU has some system-specific configuration, however this is contained to the static model description file. RabbitMQ FMU then publishes the RabbitMQ messages via regular FMI outputs.

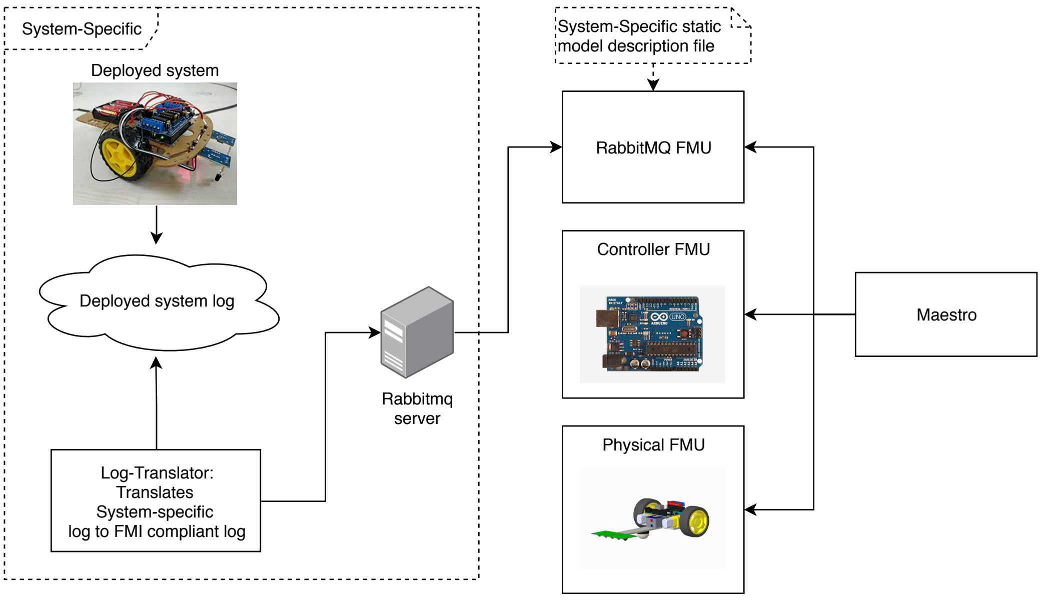
System-Specific
Deployed system
Deployed system log
Log-Translator: Translates System-specific log to FMI compliant log
Rabbitmq server
System-Specific static model description file
RabbitMQ FMU
Controller FMU
Physical FMU
Maestro

Figure 1: Interfacing Overview

This approach generalises the FMI enabling entity, the RabbitMQ FMU, such that it can be used for different kinds of systems with different kinds of logging facilities. The motivation behind this approach is that tools of the INTO-CPS Association should be generally applicable.

The following chapters describes the following in order:

Time Handling How system-time is mapped to time in an FMI-simulation context

Configuration How to configure the FMU via the ModelDescription File

Example - Line Following Robot This example demonstrates transmission of data from a deployed line following robot that is available in the co-simulation of its digital twin and how the values differ.

Conclusion A brief conclusion on the contribution.

2 Time Handling

This section concerns the digital twin and how data time is handled with relation to FMI. The description below is accompanied by fig. 2.

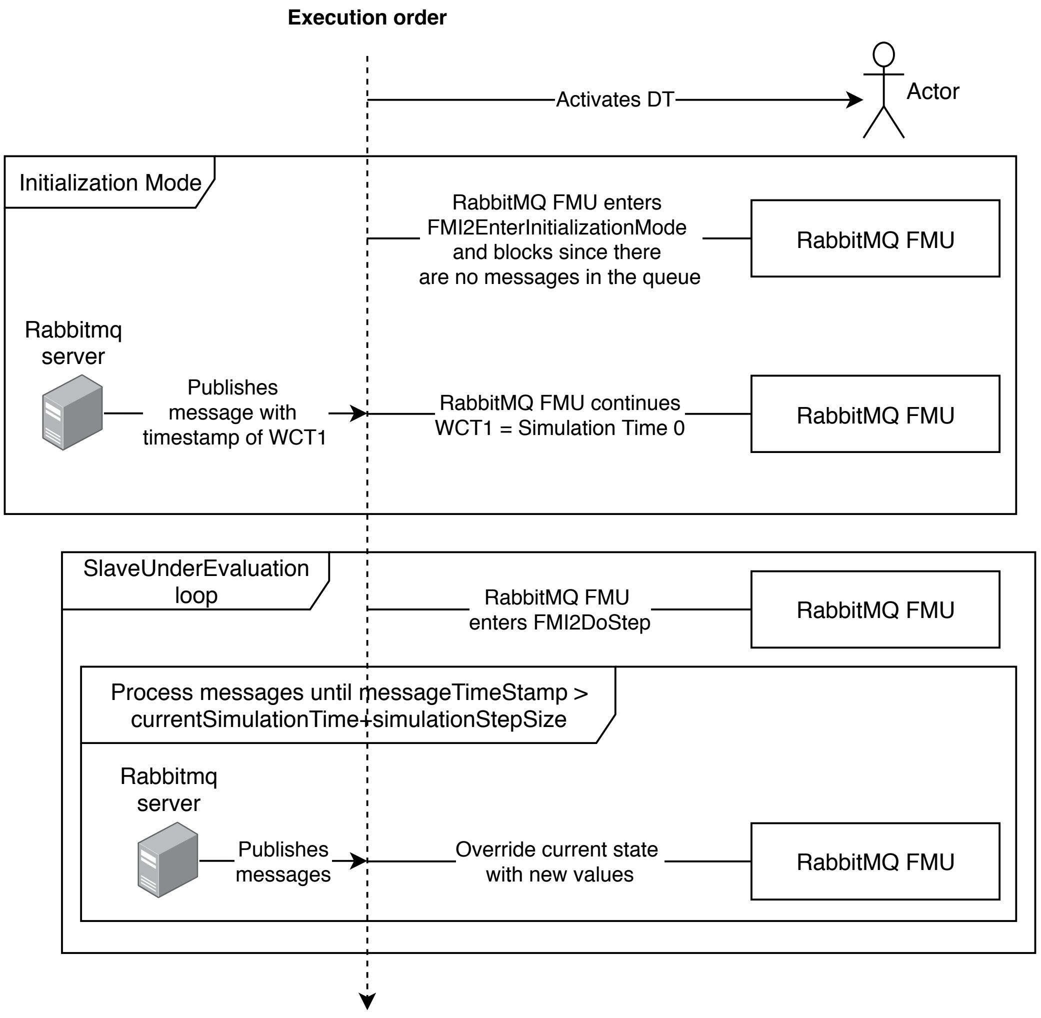
Execution order
Activates DT
Actor
Initialization Mode
RabbitMQ FMU enters FMI2EnterInitializationMode and blocks since there are no messages in the queue
RabbitMQ FMU
Rabbitmq server
Publishes message with timestamp of WCT1
RabbitMQ FMU continues WCT1 = Simulation Time 0
RabbitMQ FMU
SlaveUnderEvaluation loop
RabbitMQ FMU enters FMI2DoStep
RabbitMQ FMU
Process messages until messageTimeStamp > currentSimulationTime+simulationStepSize
Rabbitmq server
Publishes messages
Override current state with new values
RabbitMQ FMU

Figure 2: Time Handling in Simulation View

Invoking the function `fmi2EnterInitializationMode` on EDB FMU causes it to block until a message is available.
The time stamp of the first message ($WCT1$) received defines $simulationTime0$ and EDB FMU continues. Thus $WCT1 = simulationTime0$ and a mapping between WCT and simulation time is establishedcreated. Thus, any subsequent timestamps has $WCT1$ subtracted in order to map them to $simulationTime$.

This also implies that any message with a time stamp of $WCT0 < WCT1$ are ignored.

Invoking the function `fmi2DoStep` on EDB FMU causes it to process messages and keep executing until there is a message with a time stamp defined by $messageTimeStampInSimulationTime >= currentSimulationTime + simulationStepSize$. Such a message is stored in order to use it for the subsequent `fmi2DoStep` operation.

3 Data Handling

The data handling occuring within `fmi2DoStep` of EDB FMU is described below. All values of messages with time stamps (converted to simulation time) within the time interval $]currentSimulationTime, currentSimulationTime + simulationStepSize]$ overrides the current state of values in the received order.

An example is given in fig. 3. First, `message x` is received and overwrites the value of `a` in the EDB FMU state.
Afterwards, `message y` is received and overwrites the value of `b` in the EDB FMU state.
Lastly, `message z` is received and overwrites the value of `a` in the EDB FMU state.
The example above implies that the value of `a` in `message x` is never outputted from the EDB FMU, since it has been overwritten by the value of `a` in `message z` within the same `fmi2DoStep` execution.

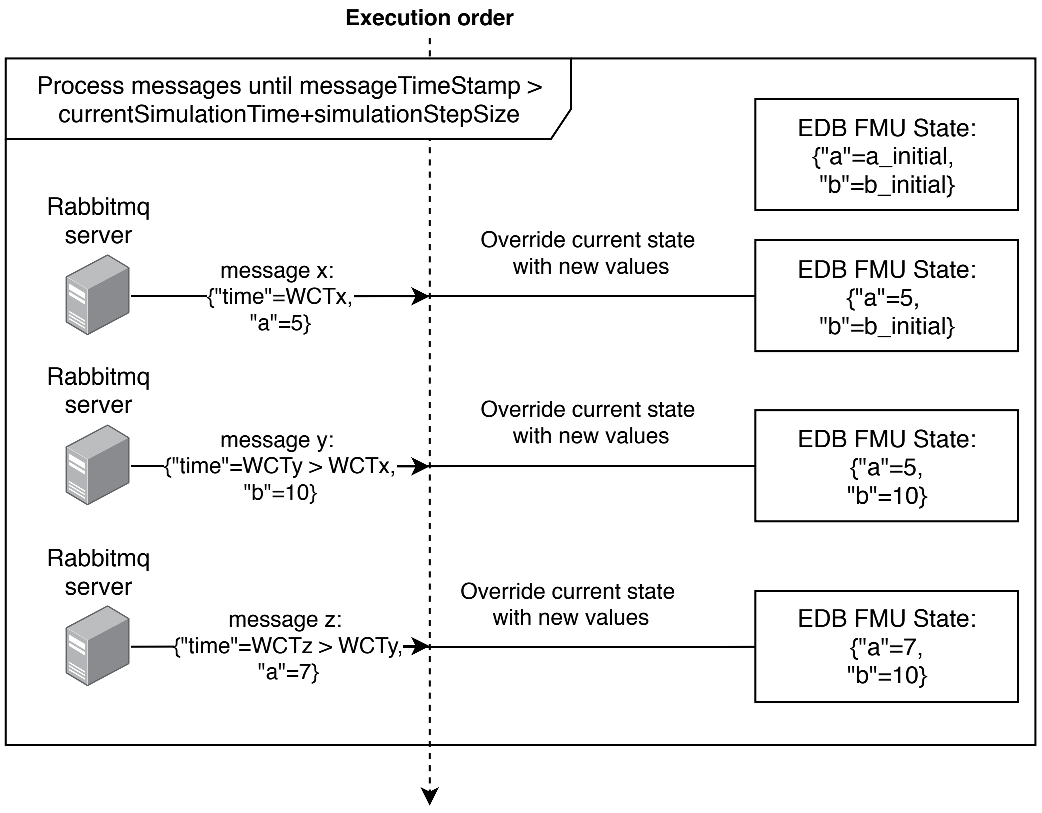
Execution order
Process messages until messageTimeStamp > currentSimulationTime+simulationStepSize
EDB FMU State: {"a"=a_initial, "b"=b_initial}
Rabbitmq server
message x: {"time"=WCTx, "a"=5}
Override current state with new values
EDB FMU State: {"a"=5, "b"=b_initial}
Rabbitmq server
message y: {"time"=WCTy > WCTx, "b"=10}
Override current state with new values
EDB FMU State: {"a"=5, "b"=10}
Rabbitmq server
message z: {"time"=WCTz > WCTy, "a"=7}
Override current state with new values
EDB FMU State: {"a"=7, "b"=10}

Figure 3: Data Handling in `fmi2DoStep`

References

A List of Acronyms

XML	Extensible Markup Language