

FMUs for runtime monitoring in RTLola

Masters defence by: Selleban Mohamud Farah

Supervised by: Volker Stolz



Table of contents

01

Introduction

Introduce thesis topic

02

Background

Going over concepts and technologies

03

Use case and Requirements

Integration use cases and requirements to fulfill them

04

Implementation

Method and technicalities of implementation

05

Testing and Results

Cover testing conducted on solution, demo

06

Discussion and future works

Evaluate results, discuss usefulness of the solution

Research Goals



Goal 1

Enable runtime monitoring and verification of RTLola within Functional Mock-up units in its use in Co-simulation



Goal 2

Integrate it through building RTLola support for the Functional mock-up interface, allowing it to be incorporated into FMI compliant tools



Goal 3

Build baseline for further development and use in building and monitoring Digital twins

Background

FMI and FMU

- Functional mock-up interface and Functional Mock-up Unit.
- A standard that defines an interface used in computer simulation, to simulate complex cyber-physical systems.

RTLola

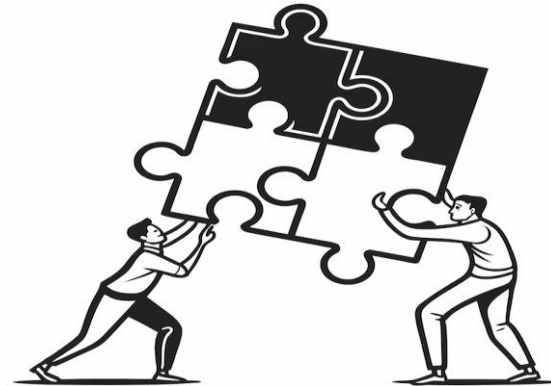
- Stream based real-time runtime verification tool
- Well suited for cyber-physical systems
- Rich specification language

Digital twins

- Digital model that is a 1-to-1 representation of a physical system
- optimize, test, analyze and predict behavior of a system

The Functional Mock-up Interface

- FMI standard defines a standardized interface that allows Functional Mockup Units (FMUs) to exchange models and interact within a simulation environment.
- Streamlines simulation integration
- Built and maintained by modelica association
- Several interface types
- Co-simulation, coupling of subsystems
- Build simulation models for use in various industries
- Supported by various simulation environments



The Functional Mock-Up Unit

- self-contained, modular representation of a dynamic system or subsystem, designed for use in simulations
- It the implementation FMI condensed and packaged into a module
- Importer load and run these modules
- Zipped into .FMU file exstension format

FMU Build up



XML files

All static data related to FMU functionality e.g variable names, interface type , dependencies

Binaries/Libraries

Required for importer to access and execute FMI functions, external functionality through shared libraries

FMI API

The standardized FMI functions that drive the simulation

Model Algorithm

The code that dictates models logic, in C code

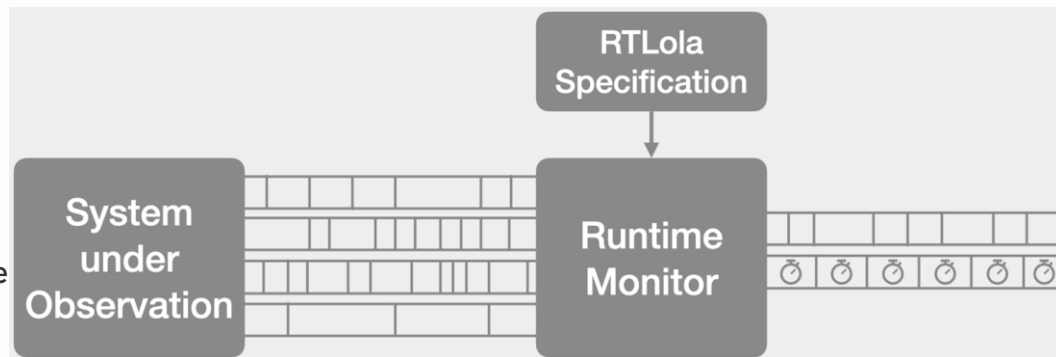
RTLola

- Real-time runtime monitoring tool
- Used to monitor real-time cyber-physical systems
- Stream based
- Specification language
- Turns input streams into output streams
- Handles timed and asynchronous inputs
- Built in Rust
- Frontend and Interpreter
- Precompiled executable



RTLola - Specification Language

- High level specification language
- Stream equations
- define input and output streams
- Easy to implement
- Highly expressive specification language
- Reusability
- Strict typing, but basic to use
- Handle timed and asynchronous inputs
- Variable-rate inputs
- Offset
- Aggregation



Specification Example

```
input temperature: Float64
input pressure: Float64

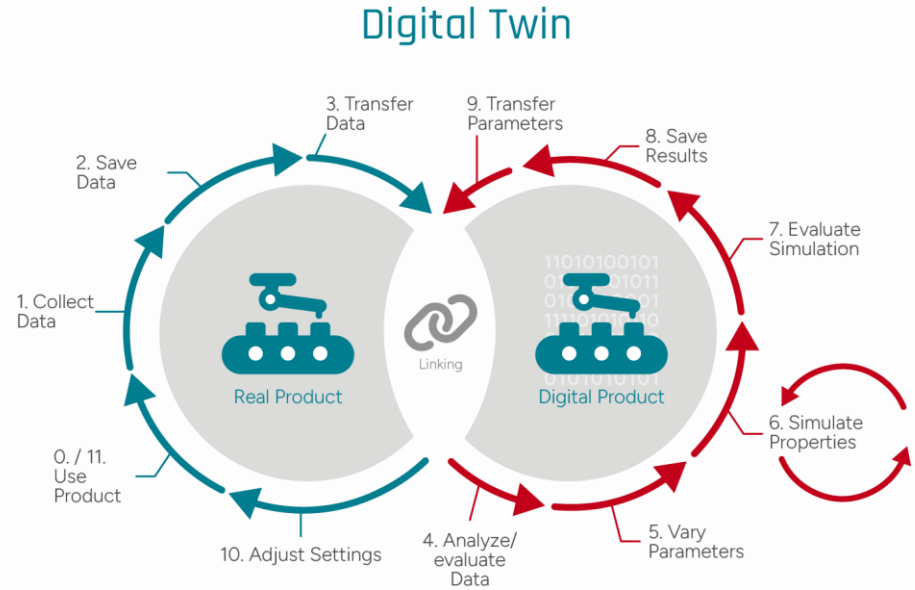
output temp_threshold: Bool @temperature := temperature > 75.0
output temp_press_ratio: Float64 @(temperature ^ pressure) := temperature / pressure
output pressure_threshold: Bool @pressure := pressure > 200.0
output critical_condition: Bool @(temperature ^ pressure) := temp_threshold && pressure_threshold

trigger @temperature temp_threshold "Warning: Temperature exceeds safe limit!"
trigger @pressure pressure_threshold "Warning: Pressure exceeds safe limit!"
trigger @(temperature ^ pressure) critical_condition "Critical: Both temperature and pressure are above safe limits!"

output avg_temp_last_10s : Float64 @1Hz := temperature.aggregate(over: 10s, using: avg).defaults(to: 0.0)
trigger @1 Hz avg_temp_last_10s > 70.0 "Alert: Average temperature over the last 10 seconds is too high!"
```

Digital twins

- Represents real-world physical product
- 1-to-1 reflection of real world counterparts
- Encapsulates all of the physical components features
- Direct communication channels
- Processes, Systems, Components
- Test, monitor and analyze



Motivation - Why build RTLola support ?

- Integration will allow runtime monitoring of cyber-physical systems defined through FMUs
- Take advantage off the expressive high level specification language
- Monitor large complex simulations
- Adapting to FMUs will extend RTLola capabilities to the various FMI compliant tools
- Explore the usefulness in development off digital twins

Use case and Requirements

Use cases

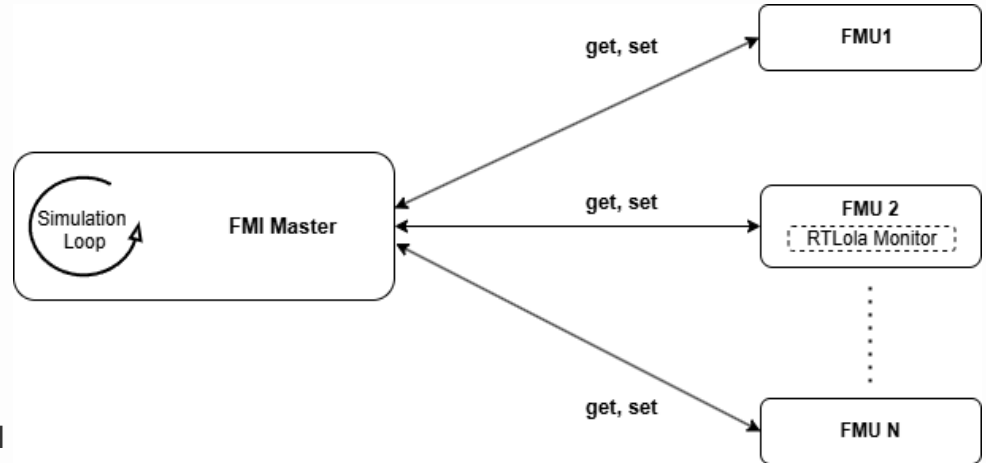
- Portable simulation model
- Monitor diverse co-simulation environments and cyber-physical systems
- Monitor changing environments to fit potentially varying needs
- Simulation behavior influenced by monitoring evaluations, calculations and triggers
- Efficient and correct monitoring

Requirements

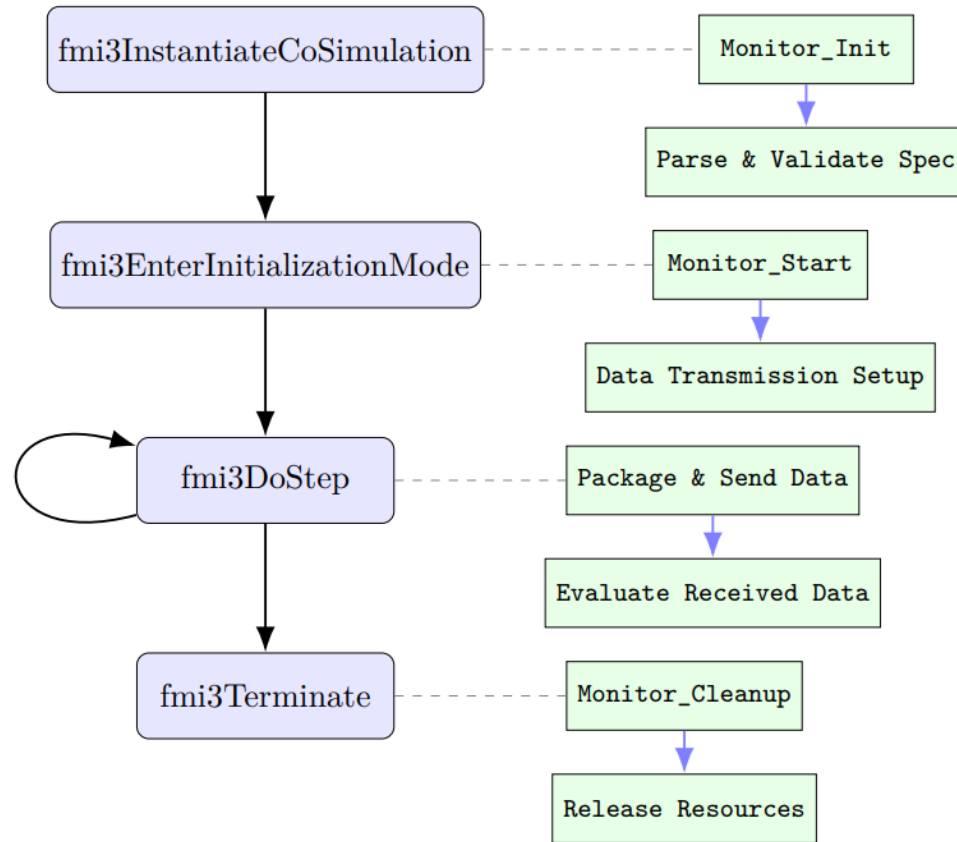
- Embed and packaged RTLola within FMU
- Translate FMU output data to input stream for RTLola evaluation
- Data and type integrity
- Access RTLola through FMI
- Fast evaluation response
- Dynamic specification loading and switching
- Solution integrable and compliant on different FMI versions and systems

Implementation methodology

- FMU lifecycle
- Bake into the FMI framework
- Invoke RTLola within FMI
- In larger Co-simulation environment
- Runtime verification paper 2022
- DejaVu for runtime monitoring in FMI based Co-simulation
- Two approaches to integration



FMU Lifecycle

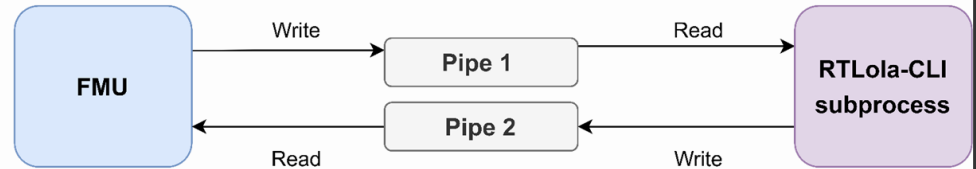


Inner workings

- Shared C structures
- Manages configuration, runtime state, and integration handles
- Specification parsing
- Type Checking & Conversion Ensures safe data exchange (FMI types → RTLola types) via runtime checks
- Variable mapping, matching of FMU variable names and types with RTLola input streams
- Modify getters and setters

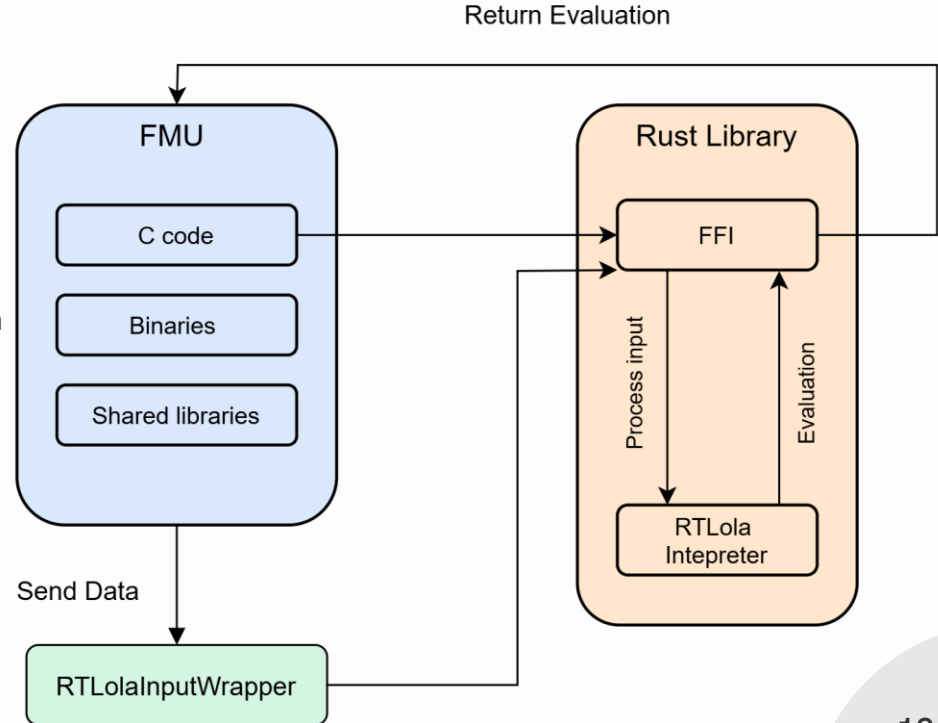
RTLola-CLI subprocess and POSIX Pipes

- Uses interprocess communication for real-time monitoring
- FMU spawns a RTLola-CLI subprocess
- Communicates through two unidirectional pipes
- FMU sends timestamped CSV data to RTLola after each step
- FMU data evaluated
- Output stream read back into FMU memory
- CLI tool bundled with FMU package
- Separation of concerns



Native rust through Foreign Function Interface

- Custom Rust library wraps RTLola interpreter
- Part of the shared library
- Exposes C-compatible interface via FFI
- Called from FMUs internal C code
- Instantiate and start a monitor instance
- FMU passes output data to RTLola monitor each simulation step
- Monitor computes periodic streams and checks against specification
- Return output streams directly



	Requirement Goal	IPC Pipes	Native FFI	Notes	
	Proper FMU output to RTLola Input for evaluation	✓	✓	Both approaches	
	Proper FMU deployment	✓	✓	RTLola-CLI executable needs to be bundled, causes large file sizes	
	Specification validation on start up	✓	✓	Both solution validate spec and maps to FMU variables	
	Error handling	✓	✓	Error handling up to developer when using native FFI	
	Specification switching	Slow	Fast	Reinstating pipes takes up more time	
	Full RTLola typing support	✓	✓	Full coverage of the RTLola basis types	
	Latency	Slow	Fast	IPC struggles handling of high frequency of inputs	
	Cross platform deployment	Partial	Good	IPC method would need to fit system/environment or	

Testing

- Test environment that load and run the RTLola-FMU integration
- Test done to assert FMUs ability to monitor specific properties using RTLola
- Solution adapts BouncingBall FMU
- Test lateral details and functionality
- Test efficiency
- Display the integration usefulness in monitoring Co-simulation environments

Discussion and Evaluating results

- Successfull conversion that allows running off RTLola through FMUs
- Improve FMU reliability and adaptability during simulation
- Native FFI approach generally outperforms IPC subprocess method
- Integration enables advanced monitoring in complex co-simulation environments
- Some limitations due to test scope and setup
- Approach is theoretically portable to any FMI-compliant toolchain
- FMUs as a standalone RTLola monitor
- Supports digital twin development via FMU supporting digital twin technology

Metric	Baseline	FFI	Overhead (%)
Step Time (ms)	0.0029	0.0756	2,506.9
Total Time (s)	0.0273	0.1327	386.5
Memory (MB)	43.06	49.33	14.6

Metric	Baseline	IPC	Overhead (%)
Step Time (ms)	0.0032	6.6559	210,146.4
Total Time (s)	0.0243	3.3876	13,838.7
Memory (MB)	43.06	43.53	1.1

Future works

- Testing and use in large more complex co-simulation environments
- Adapt further on the implementation
- Further exploration of RTLola functionalities for FMU monitoring
- Sliding window aggregation
- Further exploration for use in development and use in digital twins
- Development for digital twins using SMOL
- Potentially expand on FMI compliant tools

Thank you for your time!

Do you have any questions?

