NTNU

Norwegian University of Science and Technology

FMU4cpp: A C++ Framework for Creating FMI-Compliant Functional Mock-up Units

Lars Ivar Hatledal, PhD
Associate Professor
Department of ICT and Natural Sciences
Faculty of Information Technology and Electrical Engineering

Problem formulation

- A large number* of *modelling tools* supports exporting their models as FMUs.
  - This is great! But what if we need custom logic?
- Several frameworks enabling programming languages to be directly used also exists.
  - Compared to specialized models exported from simulation tools, such FMUs can do pretty much *anything*.
    - I.e., Custom logic, Networking, Computer vision, AI, etc.

* https://fmi-standard.org/tools/

Problem formulation cont.

- Non-native (interpreted or managed) programming frameworks available:
  - JavaFMI, FMI4j, PythonFMU and more.
  - Benefit: High-level languages that can do a lot.
  - Downside: Typically slower, requires a runtime, might expose IP.
- Native frameworks available:
  - Modelica Reference-FMUs, CPPFMU & fmicpp and more.
  - Benefit: High performance, hides IP, no runtime required.
  - Downside: Building native FMUs is still to involved.
    - **This is what FMU4cpp tries to resolve.**

Value proposition

- FMU4cpp brings the benefits of native FMUs, while:
  - Being very easy and intuitive to setup/use.
  - Solves cross-compilation issues.
  - Provides integrated compliance checks.

In a limited test-case, a fmucpp system of 4 connected FMUs simulated ~10x quicker than the same PythonFMU setup.

What is fmu4cpp?

And why we want it

- Fmu4cpp is a combination of:
  - A C++ library for creating FMUs.
  - A CMake utility function for packaging the content of an FMU.
  - A Continuous Integration (CI) pipeline for validating and cross-compiling authored FMUs.
- Supports FMI2 and **FMI3** for Co-simulation.
  - Any number of FMUs can be built and packaged during the same build step.
  - When linking against shared-libraries, these are automatically packaged with the FMU.
  - Generates the modelDescription.xml.

```
set(dest "${CMAKE_SOURCE_DIR}/fmus/fmu4cpp")

generateFMU("cpp_propeller"
        SOURCES
        "prop_model.cpp"
        FMI_VERSIONS
        "fmi2"
        "fmi3"
        DESTINATION
        ${dest}
)

generateFMU("simplified_motor"
        SOURCES
        "motor.cpp"
        FMI_VERSIONS
        "fmi2"
        "fmi3"
        DESTINATION
        ${dest}
)

generateFMU("tyristor_soft_starter"
        SOURCES
        "tyristor.cpp"
        FMI_VERSIONS
        "fmi2"
        "fmi3"
        DESTINATION
        ${dest}
)

generateFMU("variable_frequency_drive"
        SOURCES
        "vfd.cpp"
        FMI_VERSIONS
        "fmi2"
        "fmi3"
        DESTINATION
        ${dest}
)

generateFMU("reduction_gear"
        SOURCES
        "reduction_gear.cpp"
        FMI_VERSIONS
        "fmi2"
        "fmi3"
        DESTINATION
        ${dest}
)
```

Workflow

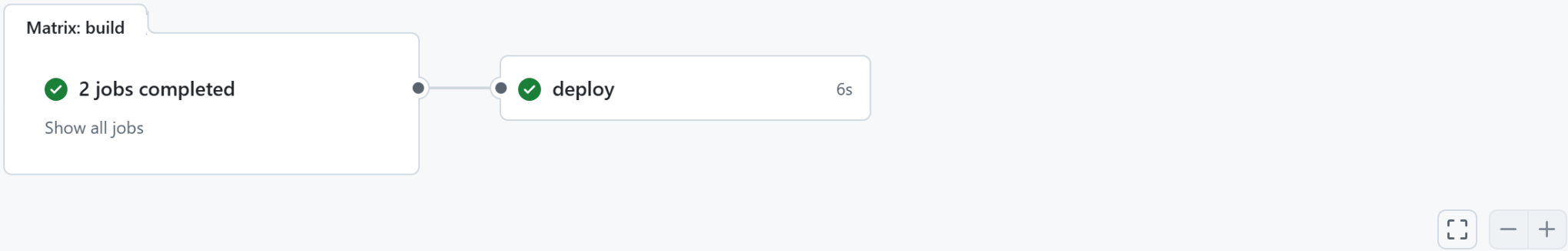
Standalone project

1. Clone the repository using the "Use this template" button.
2. Browse the example models from the examples *branch*.
3. Implement your own model by inheriting from the provided *fmu_base* abstract class.
4. Create an FMU target by using the provided *generate_fmu* CMake function.
5. Build! Models for your platform are by default located in a */models* folder within the build folder.
6. Push your changes to GitHub to trigger cross-compilation.
7. Download cross-compiled FMUs from the Actions tab.

Triggered via push 3 days ago	Status	Total duration	Artifacts
markaren pushed 7f935e8 master	Success	1m 20s	3

cross-compile.yml

on: push



Artifacts

Produced during runtime

Name	Size	Digest
models-merged	2.72 MB	sha256:c94de40cadae5fb6439e89a598bc8986d4f4c28cb88...
models-ubuntu-22.04	2.29 MB	sha256:d738bcb3938aab7694b6a083c217ea46c87f6cba2ce...
models-windows-2022	621 KB	sha256:4c66633e4f8ee88ad3fd1536d30bd52588cee2dc655...

Alternate workflow

Existing project integration

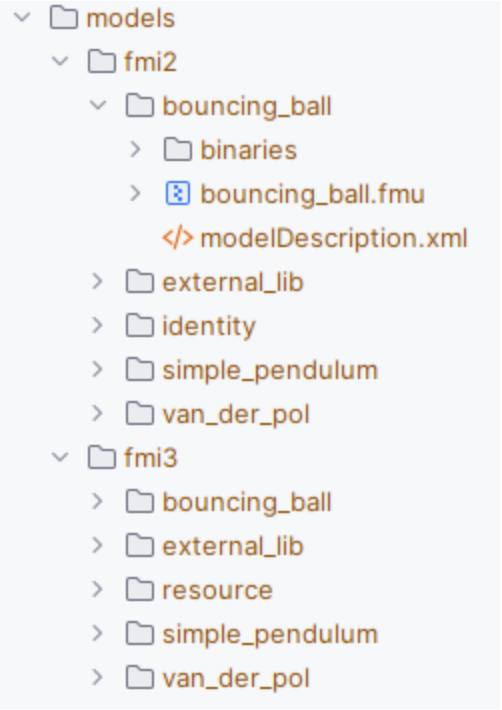
1. Use CMake FetchContent to get access to fmu4cpp in an existing CMake project.
2. Implement your own model by inheriting from the provided *fmu_base* abstract class.
3. Create an FMU target by using the provided *generate_fmu* CMake function.

```
include(FetchContent)
FetchContent_Declare(
        fmu4cpp
        GIT_REPOSITORY https://github.com/Ecos-platform/fmu4cpp.git
        GIT_TAG dccd641e558df304fcbc8e516a3dcd8802939e6c
)
FetchContent_MakeAvailable(fmu4cpp)
```

Example code: BouncingBall

```cpp
class BouncingBall : public fmu_base {
public:
    FMU4CPP_CTOR(BouncingBall) {
        register_variable(real("height", &h).setCausality(OUTPUT).setVariability(CONTINUOUS).setIr
        register_variable(real("velocity", &v).setCausality(LOCAL).setVariability(CONTINUOUS));
        register_variable(real("gravity", &g).setCausality(PARAMETER).setVariability(FIXED));
        register_variable(real("bounceFactor", &b).setCausality(PARAMETER).setVariability(FIXED));
        reset();
    }

    bool do_step(double dt) override {
        v += g * dt; h += v * dt;
        if (h <= 0.0) { h = 0.0; v = -v * b; }
        log(fmiOK, fmt::format("h={:.2f}, v={:.2f}", h, v));
        return true;
    }

    void reset() override { h = 10; v = 0; g = -9.81; b = 0.6; }

private:
    double h{}, v{}, g{}, b{};
};

model_info fmu4cpp::get_model_info() {
    return {"BouncingBall", "A bouncing ball model"};
}

FMU4CPP_INSTANTIATE(BouncingBall);
```

```
generateFMU("bouncing_ball"
        SOURCES
        "bouncing_ball.cpp"
        FMI_VERSIONS
        "fmi2"
        "fmi3"
)
```



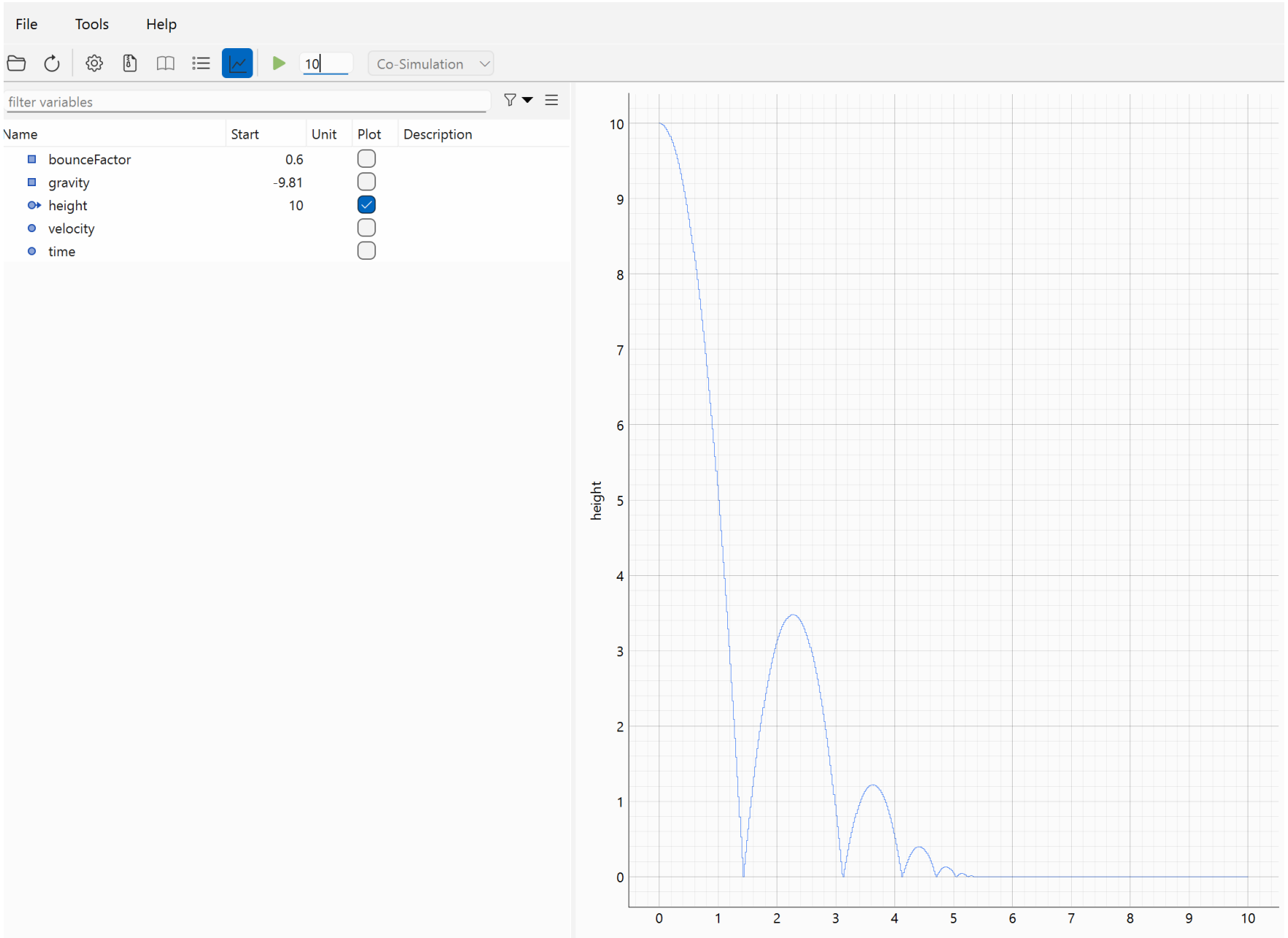
Generated modelDescription

FMI3

```xml
<?xml version="1.0" encoding="UTF-8"?>
<fmiModelDescription fmiVersion="3.0"
    modelName="BouncingBall"
    instantiationToken="16308446847443284887"
    generationTool="fmu4cpp v0.5.0"
    generationDateAndTime="2025-11-10T21:07:03Z"
    description="A bouncing ball model"
    author=""
    variableNamingConvention="structured">

    <CoSimulation
        needsExecutionTool="false"
        modelIdentifier="bouncing_ball_fmi3"
        canHandleVariableCommunicationStepSize="true"
        canBeInstantiatedOnlyOncePerProcess="false"
        canGetAndSetFMUstate="false"
        canSerializeFMUstate="false"
        providesDirectionalDerivatives="false"
        providesAdjointDerivatives="false"
        providesPerElementDependencies="false"
        providesEvaluateDiscreteStates="false"/>

    <ModelVariables>
        <!--index=1-->
        <Float64 name="time" valueReference="0" causality="independent" variability="continuous"/>
        <!--index=2-->
        <Float64 name="height" valueReference="1" causality="output" variability="continuous" initial="exact" start="10"/>
        <!--index=3-->
        <Float64 name="velocity" valueReference="2" causality="local" variability="continuous"/>
        <!--index=4-->
        <Float64 name="gravity" valueReference="3" causality="parameter" variability="fixed" start="-9.81"/>
        <!--index=5-->
        <Float64 name="bounceFactor" valueReference="4" causality="parameter" variability="fixed" start="0.6"/>
    </ModelVariables>

    <ModelStructure>
        <Output valueReference="1"/>
    </ModelStructure>

</fmiModelDescription>
```

FMI2

```xml
<?xml version="1.0" encoding="UTF-8"?>
<fmiModelDescription
    fmiVersion="2.0"
    modelName="BouncingBall"
    guid="{6339824621610749220}"
    generationTool="fmu4cpp v0.5.0"
    generationDateAndTime="2025-11-10T21:38:21Z"
    description="A bouncing ball model"
    author=""
    variableNamingConvention="structured">

    <CoSimulation
        needsExecutionTool="false"
        modelIdentifier="bouncing_ball_fmi2"
        canHandleVariableCommunicationStepSize="true"
        canBeInstantiatedOnlyOncePerProcess="false"
        canGetAndSetFMUstate="false"
        canSerializeFMUstate="false"
        canNotUseMemoryManagementFunctions="true">
    </CoSimulation>

    <ModelVariables>
        <!--index=1-->
        <ScalarVariable name="time" valueReference="0" causality="independent" variability="continuous">
            <Real/>
        </ScalarVariable>
        <!--index=2-->
        <ScalarVariable name="height" valueReference="1" causality="output" variability="continuous" initial="exact">
            <Real start="10"/>
        </ScalarVariable>
        <!--index=3-->
        <ScalarVariable name="velocity" valueReference="2" causality="local" variability="continuous">
            <Real/>
        </ScalarVariable>
        <!--index=4-->
        <ScalarVariable name="gravity" valueReference="3" causality="parameter" variability="fixed">
            <Real start="-9.81"/>
        </ScalarVariable>
        <!--index=5-->
        <ScalarVariable name="bounceFactor" valueReference="4" causality="parameter" variability="fixed">
            <Real start="0.6"/>
        </ScalarVariable>
    </ModelVariables>

    <ModelStructure>
        <Outputs>
            <Unknown index="2"/>
        </Outputs>
    </ModelStructure>
```


File
Tools
Help
10
Co-Simulation
filter variables
Name
Start
Unit
Plot
Description
bounceFactor
0.6
gravity
-9.81
height
10
velocity
time
height
0
1
2
3
4
5
6
7
8
9
10

Future work

- Source code FMUs.
  - Soon.
- Wider feature compliance.
  - FMI3 binary and dynamic arrays
- Built-in state handling (for rollbacks)
- More examples.
- Compile Source code FMUs?

Closing remarks

- Neither the FMI 3.0 nor the FMI 2.0 specs are fully implemented.
  - But good enough for common use-cases.
- Only 64-bit Windows and Linux are *officially* supported.
- The source code is Open-source and licensed under a permissive MIT license.

https://github.com/Ecos-platform/fmu4cpp

Thank you!

Contact: laht@ntnu.no