FMI1.0
FMI for Co-Simulationについて

1. FMI for Co-Simulationの概要
2. FMUを作って使ってみる
3. Tool CouplingタイプのFMUを試作する。

第５７回オープンCAE勉強会@関東（流体など）

finback

2016/08/05 改訂

1. FMI for Co-Simulationの概要

1. FMI-1.0 FMI for Co-Simulation の概要

(Functional Mock-up Interface) https://www.fmi-standard.org/start

様々なダイナミックシステムのシミュレーションツール間で Co-Simulation を実行するための規格である。

https://www.fmi-standard.org/tools (の一部)

サポートしている機能

Total numbers | Filter | ⇄ ModelExchange | CoSimulation

Tools supporting FMI	FMI Version	Export	Import	Slave	Master	Notes
OpenModelica	FMI_2.0	Available	Available	Planned	Planned	Open source Modelica environment from OSMC
	FMI_1.0	Available 3	Available	Planned	Available	
Ptolemy II	FMI_1.0				Planned	Software environment for design and analysis of heterogeneous systems.
PyFMI	FMI_2.0		Available 23		Available 32	For Python via the open source package PyFMI from Modelon. Also available as part of the JModelica.org platform.
	FMI_1.0		Available 63		Available 59	
RecurDyn	FMI_1.0	Planned	Planned	Planned	Available	High End Multi Flexible Body Dynamcis Software from FunctionBay
Reference FMUs	FMI_1.0	Planned		Planned		Reference FMUs supplied by enthusiasts and volunteers to show case specific FMU features
Scilab/Xcos FMU wrapper	FMI_2.0	Planned	Planned	Planned	Planned	FMI support for Scilab / Xcos a free and open source software for numerical computation.
	FMI_1.0	Available	Available	Available	Available	

OpenModelica, JModelica, PyFMI, Scilab/Xcos FMU wrapper, FMUSDK, EnergyPlus など
オープンソース系ツールも多い。

1. FMI-1.0 FMI for Co-Simulation の概要

1-1 どんなモデルのCo-Simulationができるのか？
1-2 どのようなツールならCo-Simulationができるのか？
1-3 ソフトウェアの構成は？
1-4 FMUの中身は？

- XMLファイル
- ダイナミックリンクライブラリに実装される関数

仕様書　https://www.fmi-standard.org/downloads

FMI 1.0 FMI for Model Exchange 2010-07-26 MODELISAR
FMI 1.0 FMI for Co-Simulation　　2010-10-12 MODELISAR
FMI 2.0 FMI for Model Exchange & Co-Simulation 2014-07-25 Modelica Association

1-1 どんなモデルのCo-Simulationができるのか？

対象となるモデルの概要　次のようなサブシステムが連成するモデルのシミュレーションを行う。

サブシステムモデル(Subsystem Models)

入力 $u(t)$ と**出力** $y(t)$
で他のサブシステムモデル
と接続され、
シミュレーション中に
中間的な計算値が
の交換がなされる。

連成システムモデル(Coupled System Model)

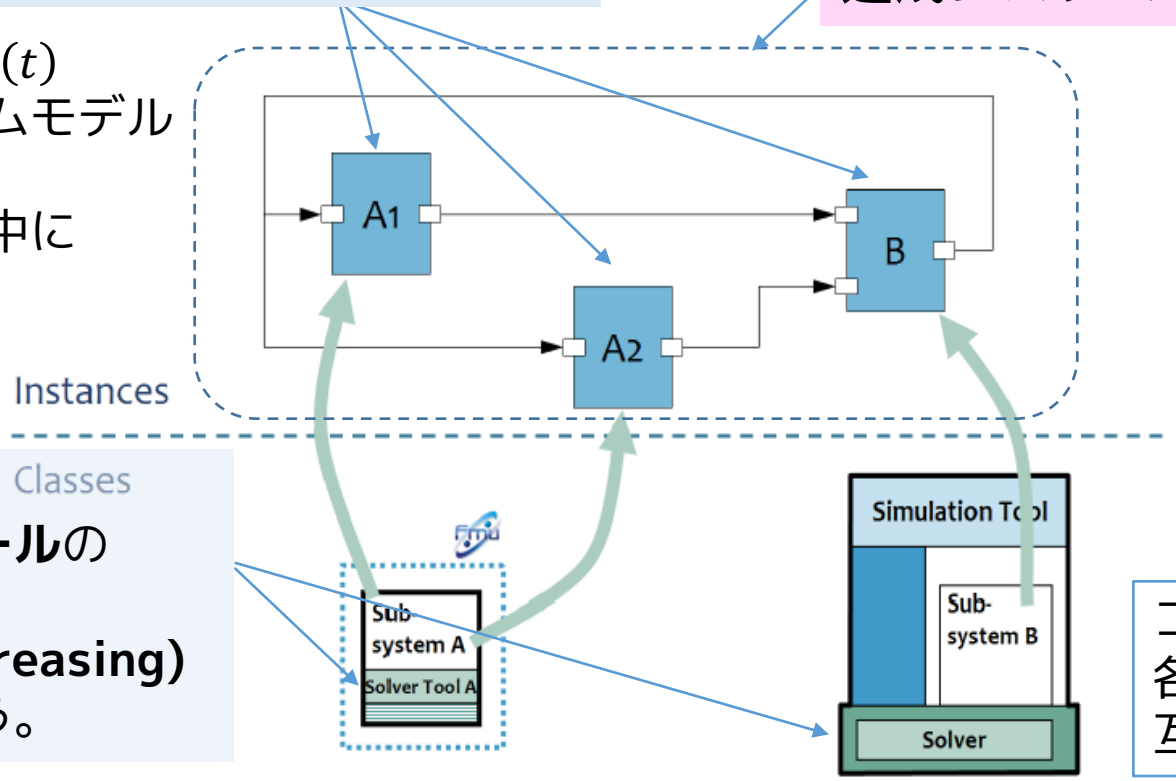


Figure 6: FMU Classifier/Instance Differences

計算値の交換を行うのは
コミュニケーションポイント tC_i と
呼ばれる離散的時刻に限定される。

$$t = tC_0, tC_1, \cdots, tC_i, \cdots, tC_N$$

コミュニケーションステップ

$$hC_i = tC_{i+1} - tC_i$$

互いに異なる
シミュレーションツールの
ソルバーが
時間前進的(time increasing)
なシミュレーンを行う。

コミュニケーションステップhC_i と
各サブシステムのソルバーの h_i は
互いに独立にとれる。

独立なタイムステップ

$$h_i = t_{i+1} - t_i$$

（仕様書 P.11より）

1-2 どのようなツールならCo-Simulationができるのか？

サブシステムのシミュレーションツール（スレーブツール）の機能的条件

時間依存変数 $v(t)$の $t_{start} < t < t_{end}$ の時間前進的 (time increasing) なシミュレーションを行うツールが次のような機能をもつとき連成(coupling)ができる。

基本的機能
- $t_{start} < tC_i < t_{end}$ となる任意の時間値（コミュニケーションポイント）tC_i を受け取ることができる。
- 計算時間が tC_i になるとシミュレーションを中断する（一時的に止める）ことができる。
- 中断している間に、入力変数 $\mathrm{u}(\mathrm{tC_i})$ を受信し、出力変数 $\boldsymbol{y}(tC_i)$ を送信することができる。
- 中断している間に、次のコミュニケーションポイント $\mathrm{tC_{i+1}}$ を受け取ることができる。

$hC_i = tC_{i+1} - tC_i \geq 0$：コミュニケーションステップ

オプション的機能（必ずしも必要ではない）
- $hC_i > 0$ を可変にできる。
- $hC_i = 0$ にできる。（イベントループのイタレーションができる）
- $[tC_i, tC_{i+1}]$ で入力変数 $\mathrm{u}(\mathrm{tC_i})$ を補間できる。($\dot{\mathrm{u}}(\mathrm{tC_i}), \ddot{\mathrm{u}}(\mathrm{tC_i}), \cdots$ を使用する)
- $[tC_i, tC_{i+1}]$ のシミュレーションを非同期的に実行することができる。

サブシステムモデルは
常微分方程式や
微分代数方程式
で表されるモデルで
無くてもいい。

1-3 ソフトウェアの構成は？

マスターツール (Master Tool)
FMUを読み込んで
サブシステム間の計算値の交換や同期、Co-Simulationの進行などを担当する。

スレーブツール(Slave Tool)
サブシステムのシミュレーションを実行する。
FMUを生成する機能をもつ

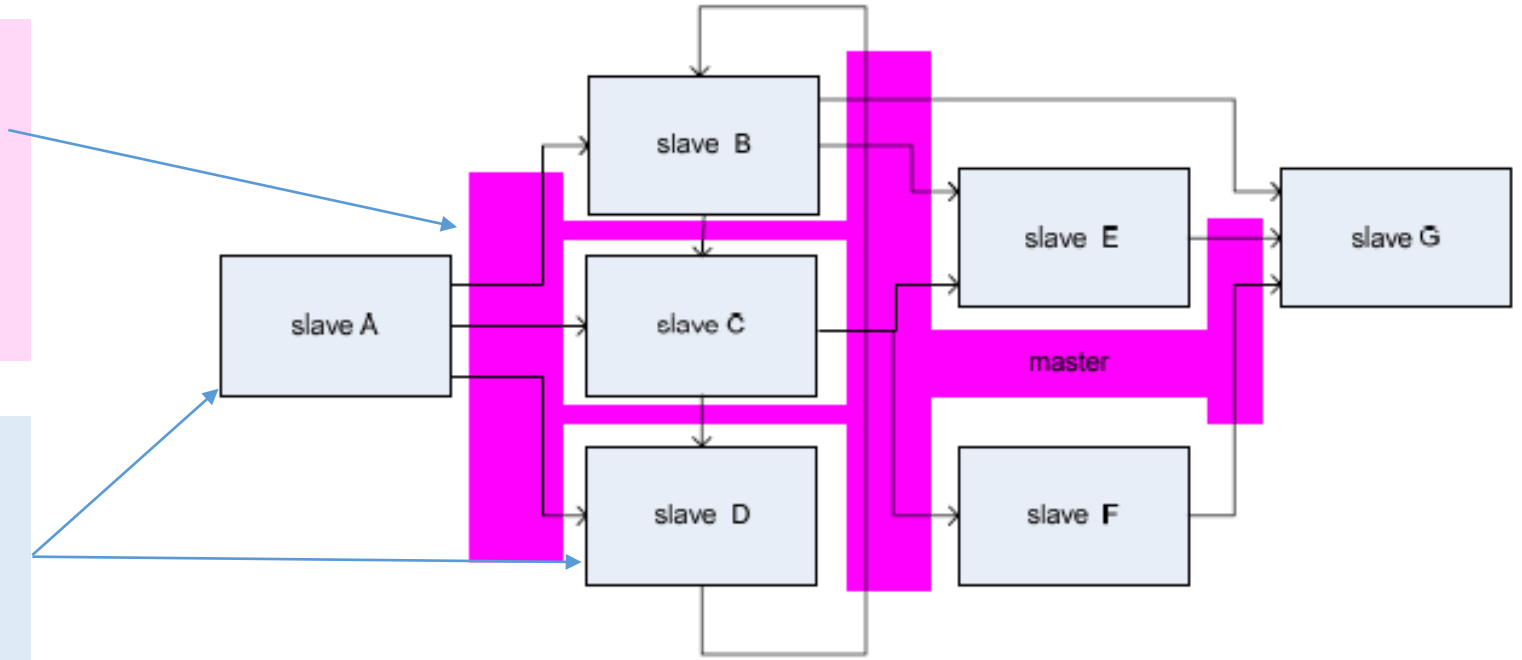


Figure 14: Master-Slave structure

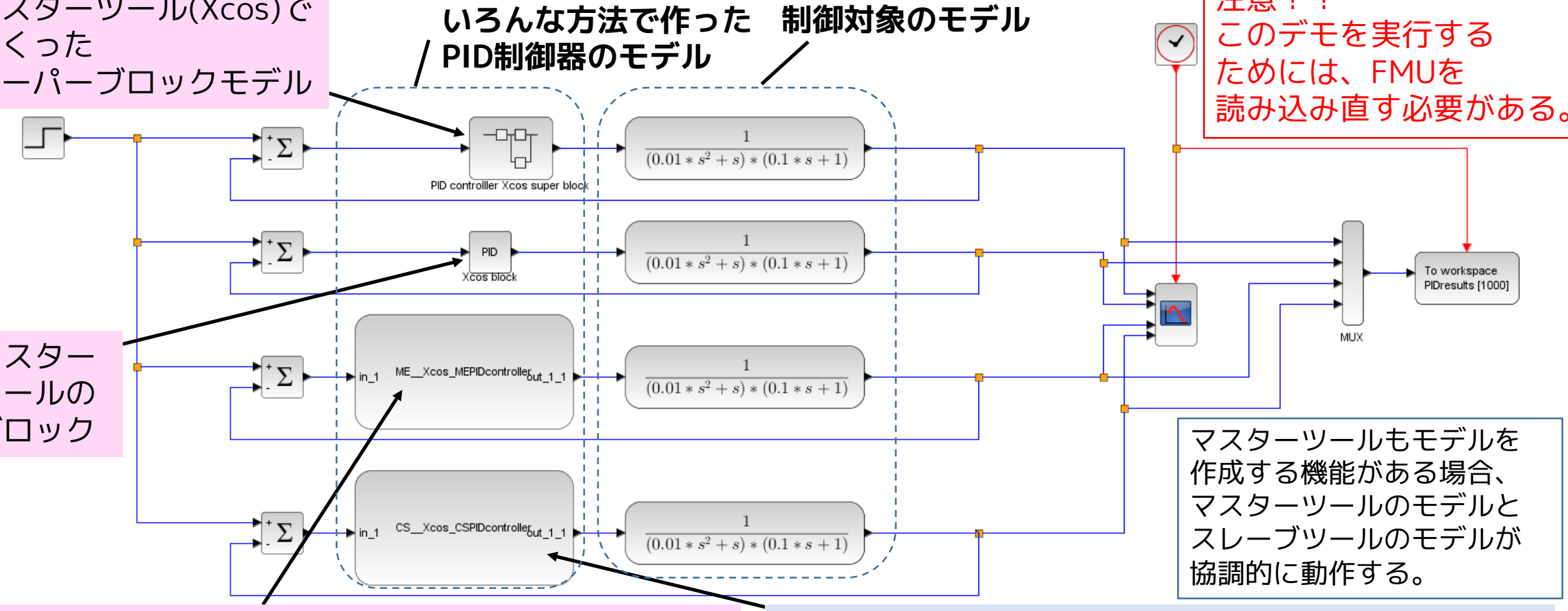
（仕様書 p.17）

FMU(Functional Mock-up Unit) for Co-Simulation
以下を含む zip ファイル（拡張子fmu）
- サブシステムモデルやソルバーに関する情報が記述されたXMLファイル
- マスターとスレーブのインターフェース関数を含むダイナミックリンクライブラリ
- Cで記述されたソースコード（必須ではない）

Co-Simulationの実例

Scilab/Xcos FMU wrapper のデモ
PID controller FMU Generated by Xcos
https://forge.scilab.org/index.php/p/fmu-wrapper/

マスターツール(Xcos)で
つくった
スーパーブロックモデル



マスター
ツールの
ブロック

注意！！
このデモを実行する
ためには、FMUを
読み込み直す必要がある。

マスターツールもモデルを
作成する機能がある場合、
マスターツールのモデルと
スレーブツールのモデルが
協調的に動作する。

FMI for Model ExchangeのFMUを読み込んだブロック
マスターツールのソルバーで動く。

FMI for Co-SimulationのFMUを読み込んだブロック
スレーブツールのソルバーで動く。

1-3 ソフトウェアの構成は？

マスターツールの役割

初期化 (initialization sub-phase)
- スレーブツールに関する情報を得る
- サブシステムの接続関係を解析する
- マスターアルゴリズムを選択する

マスターアルゴリズムは
FMI 1.0 for Co-Simulation
の仕様には含まれていない。

シミュレーション (simulation sub-phase)
- コミュニケーションステップ $\mathrm{hC_i} = \mathrm{tC_{i+1}} - \mathrm{tC_i}$ を計算する。
- サブシステムの入力値 $\mathrm{u}(\mathrm{tC_i}), (\dot{\mathrm{u}}(\mathrm{tC_i}), \ddot{\mathrm{u}}(\mathrm{tC_i}), \cdots)$を計算する。
- スレーブツールに $\mathrm{tC_i}, \mathrm{tC_{i+1}}, \mathrm{u}(\mathrm{tC_i}), \ (\dot{\mathrm{u}}(\mathrm{tC_i}), \ddot{\mathrm{u}}(\mathrm{tC_i}), \cdots)$ を送信する。
- スレーブツールに $[\mathrm{tC_i}, \mathrm{tC_{i+1}}]$ のシミュレーションを実行させる。
- 正常に計算が終了したら、出力値 $\mathrm{y}(\mathrm{tC_{i+1}}), (\dot{\mathrm{y}}(\mathrm{tC_{i+1}}), \ddot{\mathrm{y}}(\mathrm{tC_{i+1}}), \cdots)$ を受信する。
- 正常に計算が終了しない場合、オプションに応じて他の情報を送受信する。

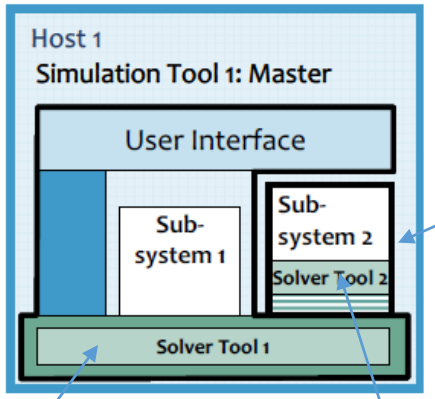
終了処理 (shutdown sub-phase)
- シミュレーションを完全に終了する。

1-3 ソフトウェアの構成は？

FMUの実装の種類

スレーブツールのソルバーがどこにあるか！

Cosimulation_StandAlone (Code Generation)
FMUのダイナミックリンクライブラリにスレーブツールのソルバーが含まれる



FMU

（仕様書 P.13）

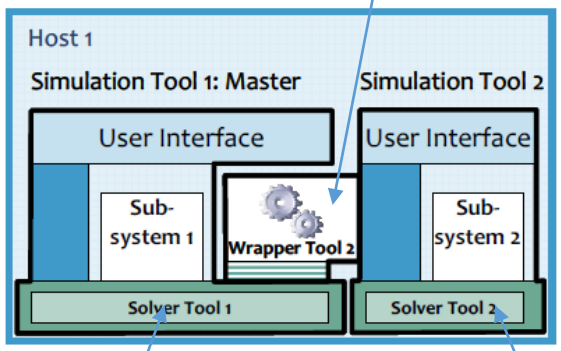
Figure 9: Co-simulation with generated code on a single computer

スレーブツールのソルバー

マスターツールのソルバー

Cosimulation_Tool (Tool Coupling)
FMUのライブラリにはスレーブツールと通信するラッパーツールが含まれる。

FMU

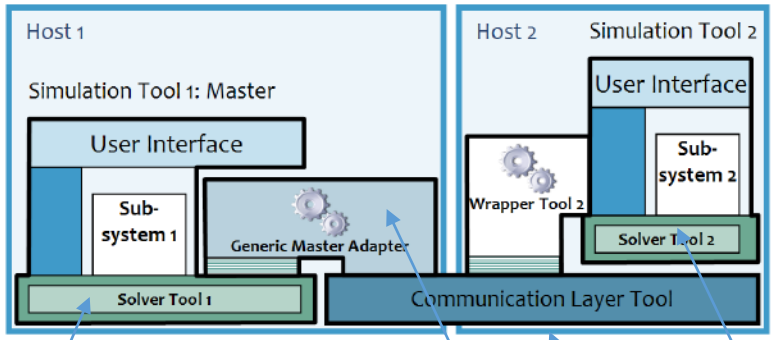


一台のコンピュータでマスターツールとスレーブツールが異なるプロセスで動作する場合。

Figure 10: Co-simulation with tool coupling on a single computer（仕様書 P.14）

マスターツールのソルバー

スレーブツールのソルバー



複数のコンピュータでマスターツールとスレーブツールが動作する場合。

Figure 11: Distributed Co-simulation Infrastructure（仕様書 P.14）

マスターツールのソルバー

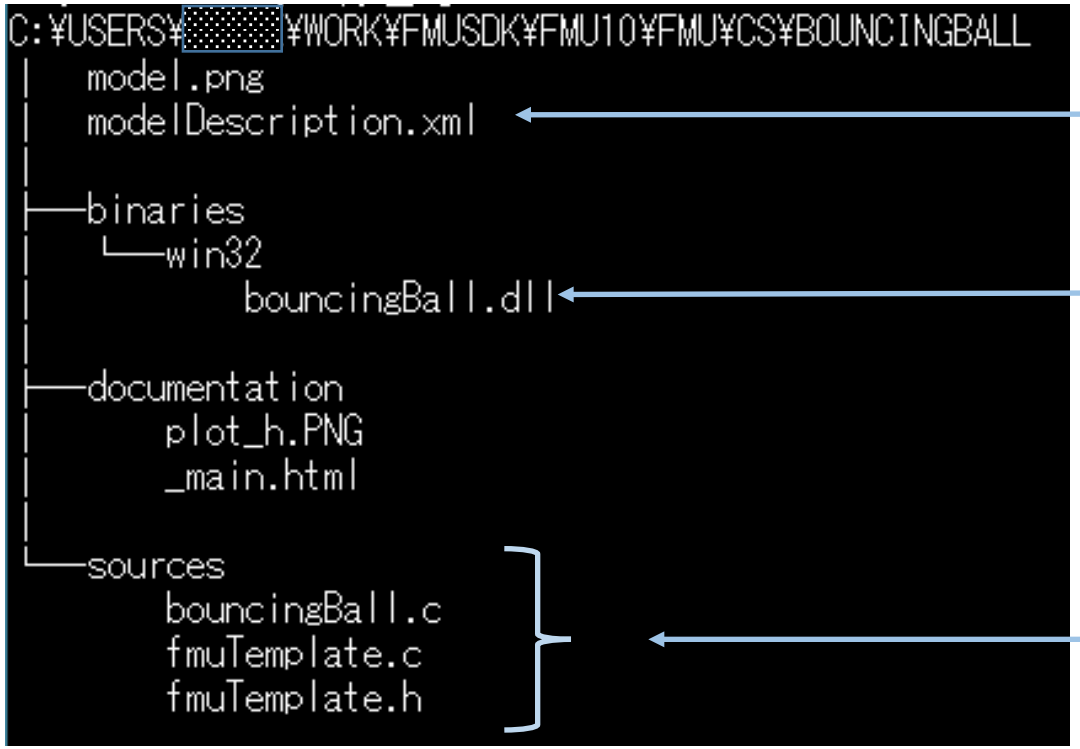
FMU

スレーブツールのソルバー

ネットワーク通信

1-4 FMUの中身は？

FMU(Functional Mock-up Unit) for Co-Simulation
以下を含む zip ファイル（拡張子fmu)
- サブシステムモデルやソルバーに関する情報が記述されたXMLファイル
- マスターとスレーブのインターフェース関数を含むダイナミックリンクライブラリ
- Cで記述されたソースコード（必須ではない）



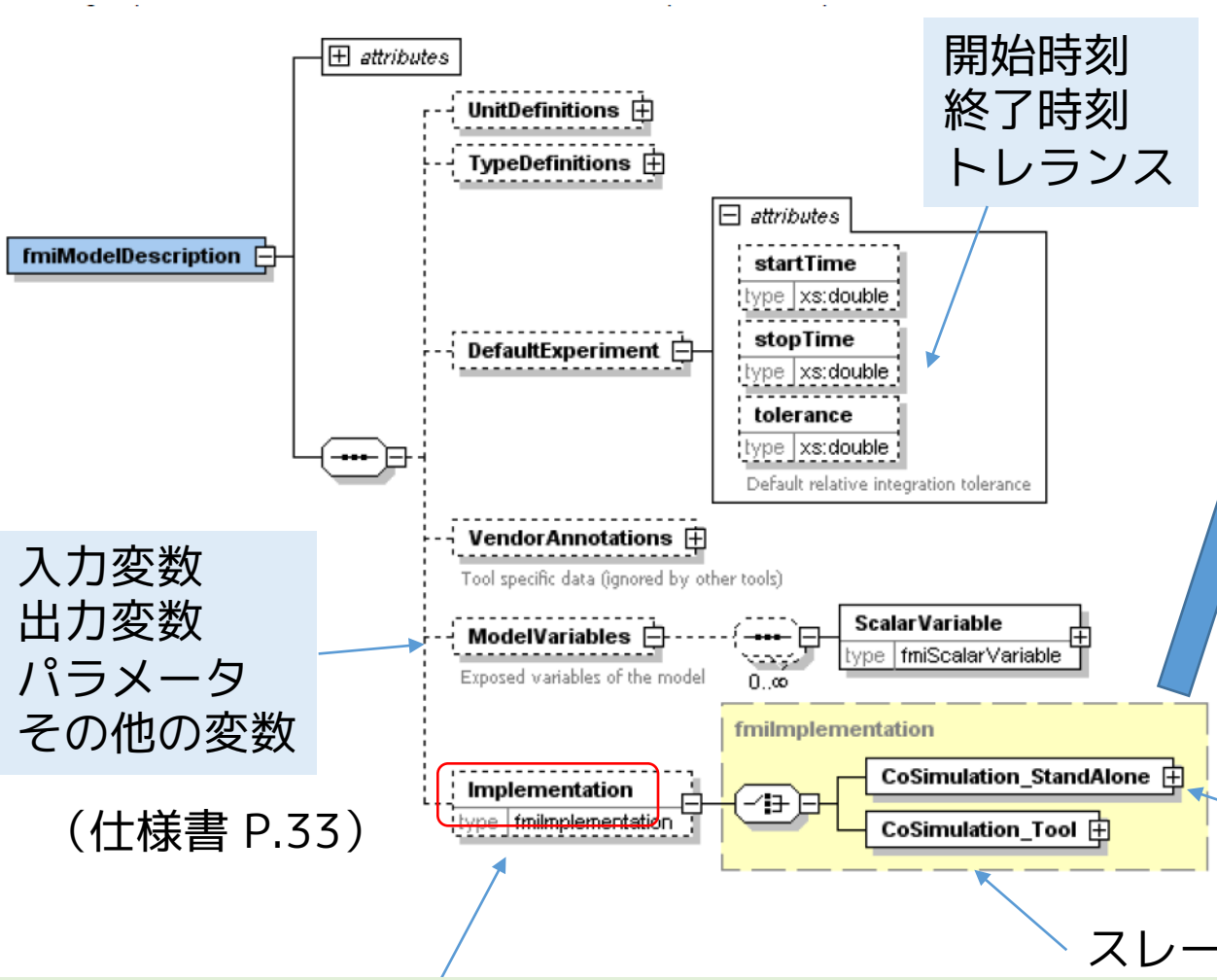
FMUSDKで作成した **bouncingBall.fmu** の内容

modelDescriptin.xml　XMLファイル

ダイナミックリンクライブラリ

Cで記述されたソースコード

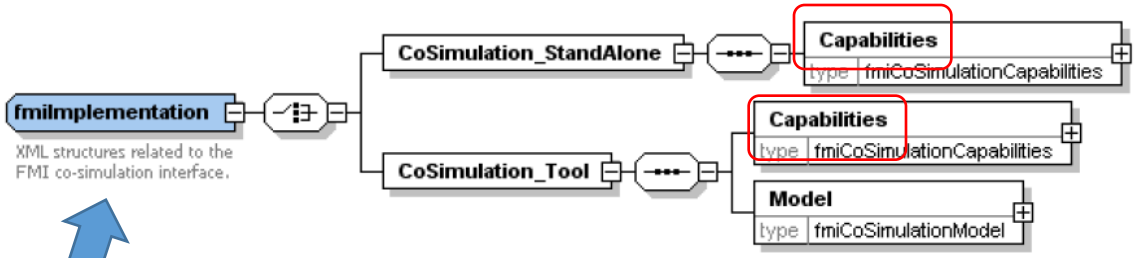
1-4 FMUの中身は？ XMLファイル　modelDescription.xml の内容



開始時刻
終了時刻
トレランス

入力変数
出力変数
パラメータ
その他の変数

（仕様書 P.33）



Capabilitiesはスレーブツールの（仕様書 P.34）
オプション的機能の有無について記述する。

- canHandleVariableCommunicationStepSize
- canHandleEvents
- canRejectsSteps

} $hC_i = 0$ が可能 ← hC_i 可変

- canInterpolateInputs
- maxOutputDerivativeOrder

} 入力変数の補間が可能

- canRunAsynchronously ← 非同期的実行が可能
- canSignalEvents
- canBeInstantiatedOnlyOncePerProces
- canNotUseMemoryManagementFunctions

（仕様書 P.36-37）

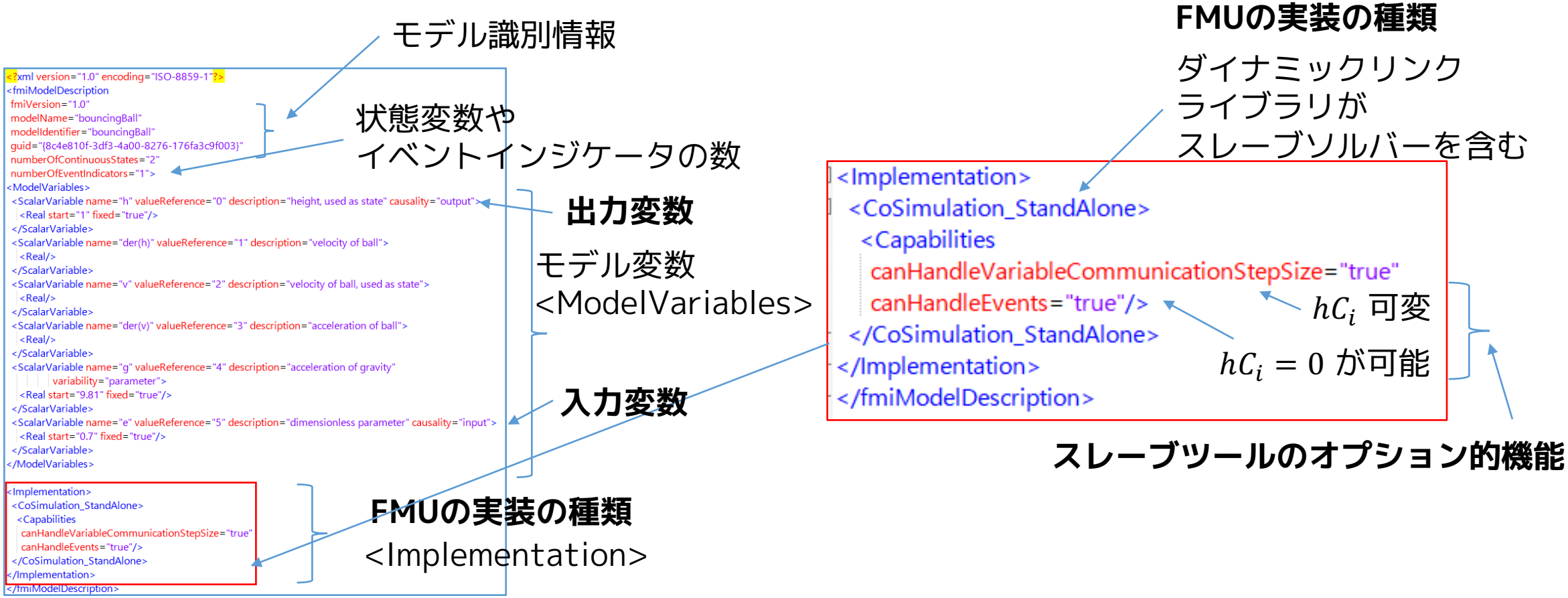
スレーブツールのソルバーを
ダイナミックリンクライブラリに含む

スレーブツールと通信するラッパーを含む

FMI for Model Exchange と比べて
FMI for Co-Simulation ではFMUの実装の種類に関する情報が追加されている。

1-4 FMUの中身は？　XMLファイル modelDescription.xmlの実例

FMU SDK（https://www.qtronic.de/jp/fmusdk.html ）
で作成したFMI 1.0 for Co-Simulation に対応した**bouncingBall.fmu**
modelDescription.xml（eを入力変数、hを出力変数に改変したもの）




```
<?xml version="1.0" encoding="ISO-8859-1"?>
<fmiModelDescription
 fmiVersion="1.0"
 modelName="bouncingBall"
 modelIdentifier="bouncingBall"
 guid="{8c4e810f-3df3-4a00-8276-176fa3c9f003}"
 numberOfContinuousStates="2"
 numberOfEventIndicators="1">
<ModelVariables>
 <ScalarVariable name="h" valueReference="0" description="height, used as state" causality="output">
   <Real start="1" fixed="true"/>
 </ScalarVariable>
 <ScalarVariable name="der(h)" valueReference="1" description="velocity of ball">
   <Real/>
 </ScalarVariable>
 <ScalarVariable name="v" valueReference="2" description="velocity of ball, used as state">
   <Real/>
 </ScalarVariable>
 <ScalarVariable name="der(v)" valueReference="3" description="acceleration of ball">
   <Real/>
 </ScalarVariable>
 <ScalarVariable name="g" valueReference="4" description="acceleration of gravity"
                 variability="parameter">
   <Real start="9.81" fixed="true"/>
 </ScalarVariable>
 <ScalarVariable name="e" valueReference="5" description="dimensionless parameter" causality="input">
   <Real start="0.7" fixed="true"/>
 </ScalarVariable>
</ModelVariables>

<Implementation>
 <CoSimulation_StandAlone>
   <Capabilities
    canHandleVariableCommunicationStepSize="true"
    canHandleEvents="true"/>
 </CoSimulation_StandAlone>
</Implementation>
</fmiModelDescription>
```

```
<Implementation>
 <CoSimulation_StandAlone>
   <Capabilities
    canHandleVariableCommunicationStepSize="true"
    canHandleEvents="true"/>
 </CoSimulation_StandAlone>
</Implementation>
</fmiModelDescription>
```

1-4 FMUの中身は？FMUのダイナミックリンクライブラリに実装される関数

FMI 1.0 FMI for Model Exchange と共通の関数

FMI のバージョンを返す
- const char* fmiGetVersion()

デバッグ用ログ出力の設定

fmiStatus fmiSetDebugLogging(fmiComponent c, fmiBoolean loggingOn)

スレーブツールに入力データを渡す。
- fmiStatus fmiSetReal(fmiComponent c, const fmiValueReference vr[], size_t nvr, const fmiReal value[])
- fmiStatus fmiSetInteger(fmiComponent c, const fmiValueReference vr[], size_t nvr, const fmiInteger value[])
- fmiStatus fmiSetBoolean(fmiComponent c, const fmiValueReference vr[], size_t nvr, const fmiBoolean value[])
- fmiStatus fmiSetString(fmiComponent c, const fmiValueReference vr[], size_t nvr, const fmiString value[])

スレーブツールから出力データを得る
- fmiStatus fmiGetReal(fmiComponent c, const fmiValueReference vr[], size_t nvr, fmiReal value[])
- fmiStatus fmiGetInteger(fmiComponent c, const fmiValueReference vr[], size_t nvr, fmiInteger value[])
- fmiStatus fmiGetBoolean(fmiComponent c, const fmiValueReference vr[], size_t nvr, fmiBoolean value[])
- fmiStatus fmiGetString(fmiComponent c, const fmiValueReference vr[], size_t nvr, fmiString value[])

1-4 FMUの中身は？ FMUのダイナミックリンクライブラリに実装される関数

FMI 1.0 FMI for Co-Simulationのみの関数(1) – 基本的機能に関するもの

ヘッダファイル fmiPlatformTypes.h の変数 fmiPlatform の文字列ポインタ("starndard32")を返す。
- const char* fmiGetTypesPlatform()

スレーブツールのインスタンス生成、初期化
- fmiComponent fmiInstantiateSlave(fmiString instanceName, fmiString GUID, fmiString fmuLocation, fmiString mimeType, fmiReal timeout, fmiBoolean visible, fmiBoolean interactive, fmiCallbackFunctions functions, fmiBoolean loggingOn)
- fmiStatus fmiInitializeSlave(fmiComponent c, fmiReal tStart, fmiBoolean StopTimeDefined, fmiReal tStop)

コミュニケーションステップのシミュレーションの実行
- fmiStatus fmiDoStep(fmiComponent c, fmiReal currentCommunicationPoint, fmiReal communicationStepSize, fmiBoolean newStep)

コミュニケーションポイント

コミュニケーションステップサイズ

スレーブツールの終了、リセット、メモリー解放
- fmiStatus fmiTerminateSlave(fmiComponent c)
- fmiStatus fmiResetSlave(fmiComponent c)
- void fmiFreeSlaveInstance(fmiComponent c)

FMU SDKの場合、関数 fmiDoStep に、コミュニケーションステップサイズの1/10のタイムステップでEuler法の数値積分を行うソルバーが組み込まれるようになっている。

1-4 FMUの中身は？ FMUのダイナミックリンクライブラリに実装される関数

FMI 1.0 FMI for Co-Simulationのみの関数(2) – オプション的機能に関するもの

入出力変数補間のため、入出力変数の時間微分を得る。

- fmiStatus fmiSetRealInputDerivatives(fmiComponent c, const fmiValueReference vr[], size_t nvr, const fmiInteger order[], const fmiReal value[])
- fmiStatus fmiGetRealOutputDerivatives(fmiComponent c, const fmiValueReference vr[], size_t nvr, const fmiInteger order[], fmiReal value[])

コミュニケーションステップのシミュレーションを非同期実行したときの処理

- fmiStatus fmiGetStatus(fmiComponent c, const fmiStatusKind s, fmiStatus* value)　スレーブの状態を調べる
- fmiStatus fmiCancelStep(fmiComponent c)　コミュニケーションステップのシミュレーションを中断（キャンセル）する。

コミュニケーションステップの途中までしかシミュレーションが成功しなかったときの処理。スレーブの状態を調べる

- fmiStatus fmiGetRealStatus(fmiComponent c, const fmiStatusKind s, fmiReal* value)
- fmiStatus fmiGetIntegerStatus(fmiComponent c, const fmiStatusKind s, fmiInteger* value)
- fmiStatus fmiGetBooleanStatus(fmiComponent c, const fmiStatusKind s, fmiBoolean* value)
- fmiStatus fmiGetStringStatus(fmiComponent c, const fmiStatusKind s, fmiString* value)

スレーブツールの状態遷移と
マスターツールからコールされる関数

関数が細かい文字で見えないと思うけど。。。

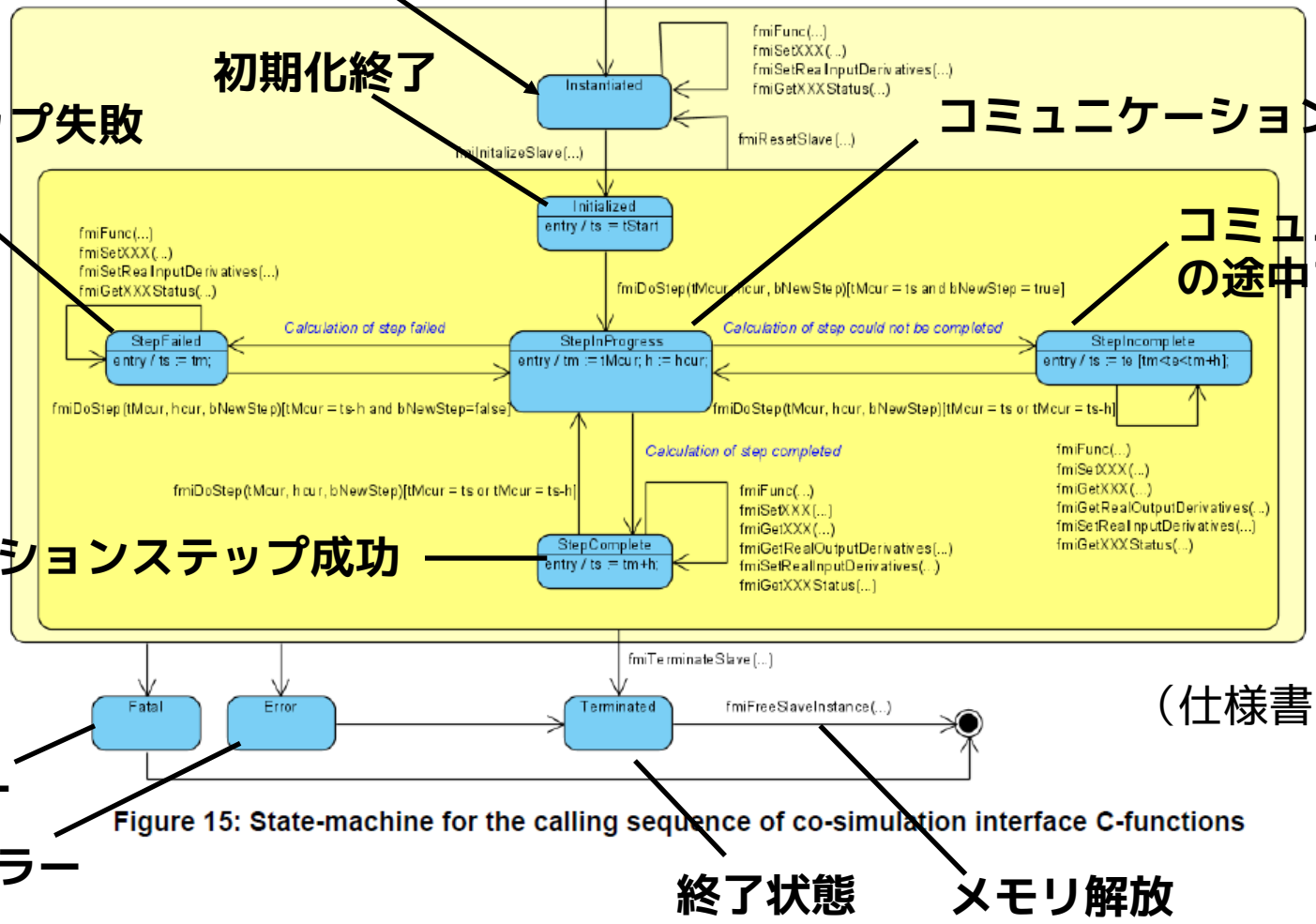


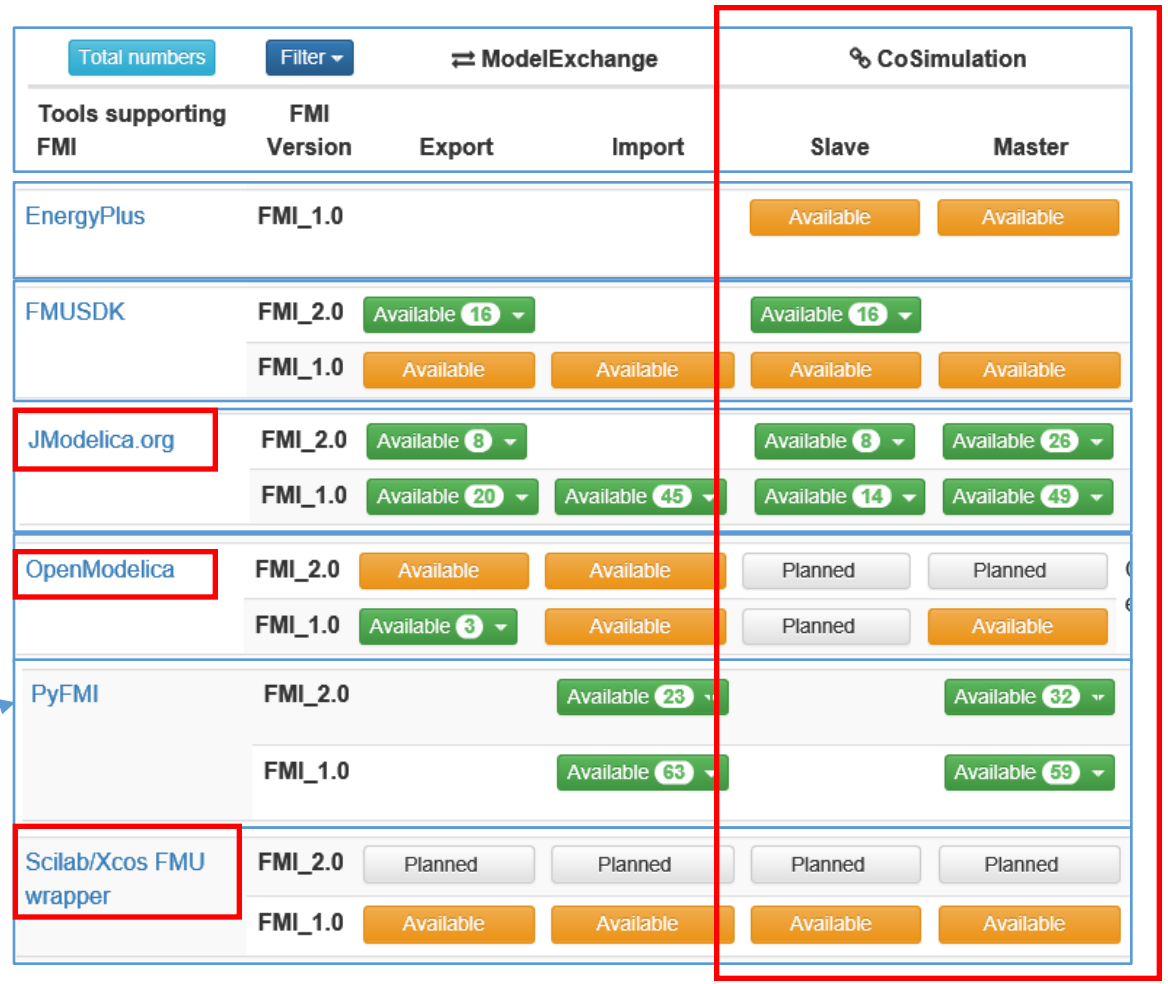
Figure 15: State-machine for the calling sequence of co-simulation interface C-functions

（仕様書 P.31）

2. FMUを作って使ってみる

2. FMUを作って使ってみる

オープンソースツールのFMIサポート状況（ https://www.fmi-standard.org/tools より抜粋）



JModelicaの
一部でもある

2. FMUを作って使ってみる

- 2-1 OpenModelica でサブシステムモデルをつくる。
- 2-2 JModelica.org でサブシステムモデルのFMUをつくる。
- 2-3 JModelica.orgでFMUを読み込む。
- 2-4 OpenModelica でFMUを読み込む。
- 2-5 Scilab/Xcos FMU wrapper でFMUを読み込む。

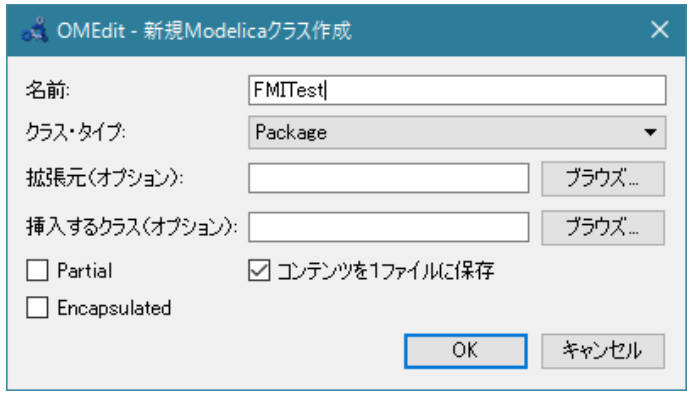
対象とするサブシステムモデル

跳ね返るボール（ Bouncing Ball ）
Modelica by Example
Discrete Behavior > Bouncing Ball
http://book.xogeny.com/behavior/discrete/bouncing/
をモデルの参考にして、e を入力変数とし、出力変数 y = h を追加したもの。

2-1 OpenModelicaでサブシステムモデルをつくる

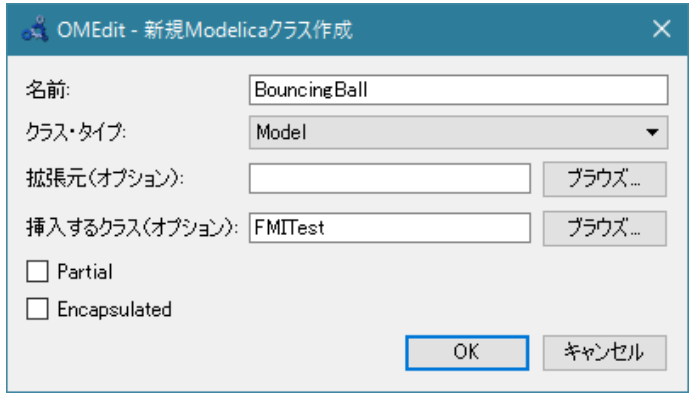
OpenModelica 1.9.6 Windows版を使用する。
(https://openmodelica.org/download/download-windows)

①ファイル>Modelicaクラス新規作成



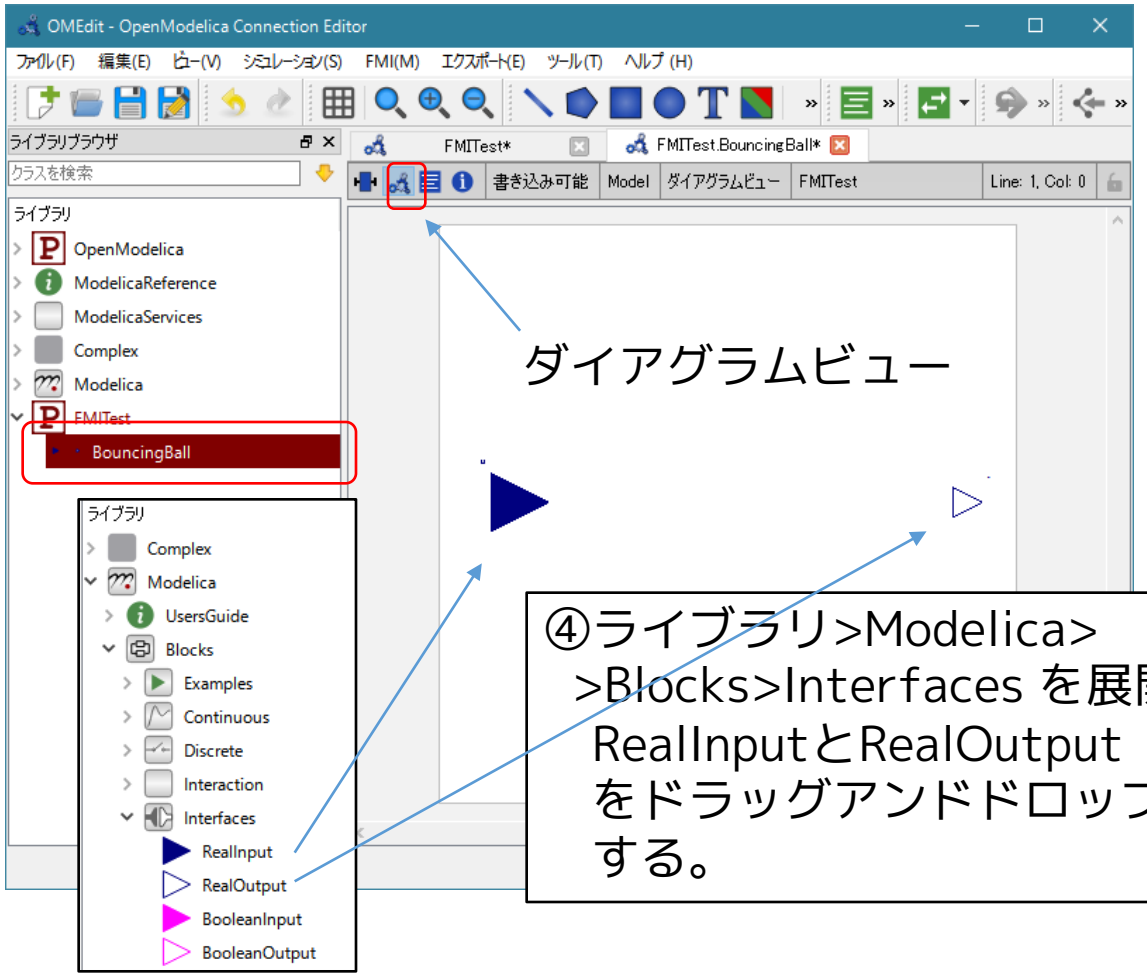
名前: FMITest
クラス・タイプ: Package

②FMITestを右クリック＞Modelicaクラス新規作成



名前: BouncingBall
クラス・タイプ: Model
挿入するクラス:
FMITest

③ライブラリブラウザで、BouncingBallを選択し、
ダイアグラムビューに切り替える。

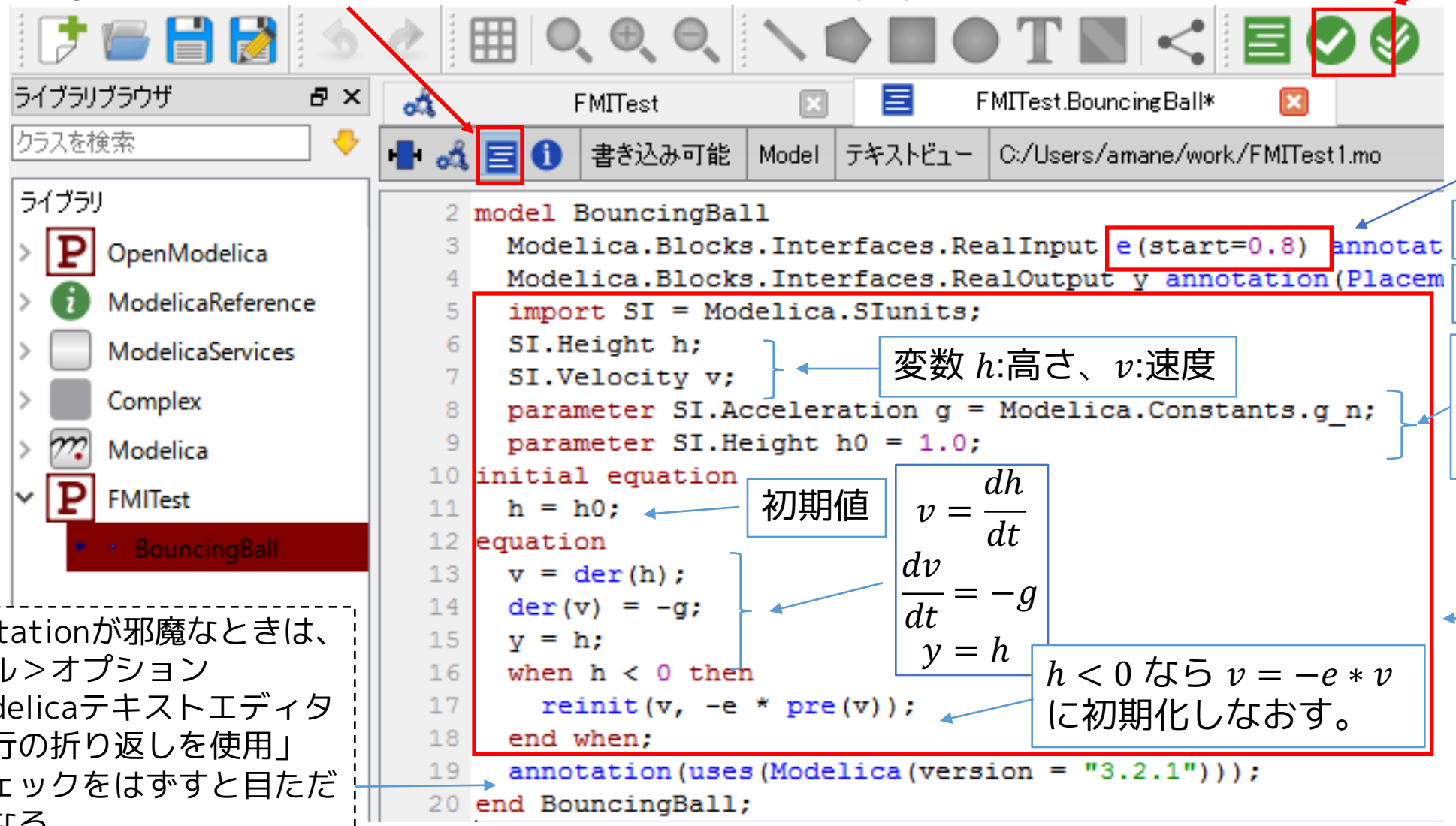


④ライブラリ>Modelica>
>Blocks>Interfaces を展開し
RealInputとRealOutput
をドラッグアンドドロップ
する。

2-1 OpenModelicaでサブシステムモデルをつくる

⑤テキストビューに切り替えて、ソースコードを編集する。

⑥編集が終わったら[モデルチェック]をクリックする。



変更部分

入力変数 e

出力変数 y

パラメータ
g：重力加速度
h_0: 高さの初期値

変数 h:高さ、v:速度

初期値

$$v = \frac{dh}{dt}$$
$$\frac{dv}{dt} = -g$$
$$y = h$$

$h < 0$ なら $v = -e * v$ に初期化しなおす。

追加するコード

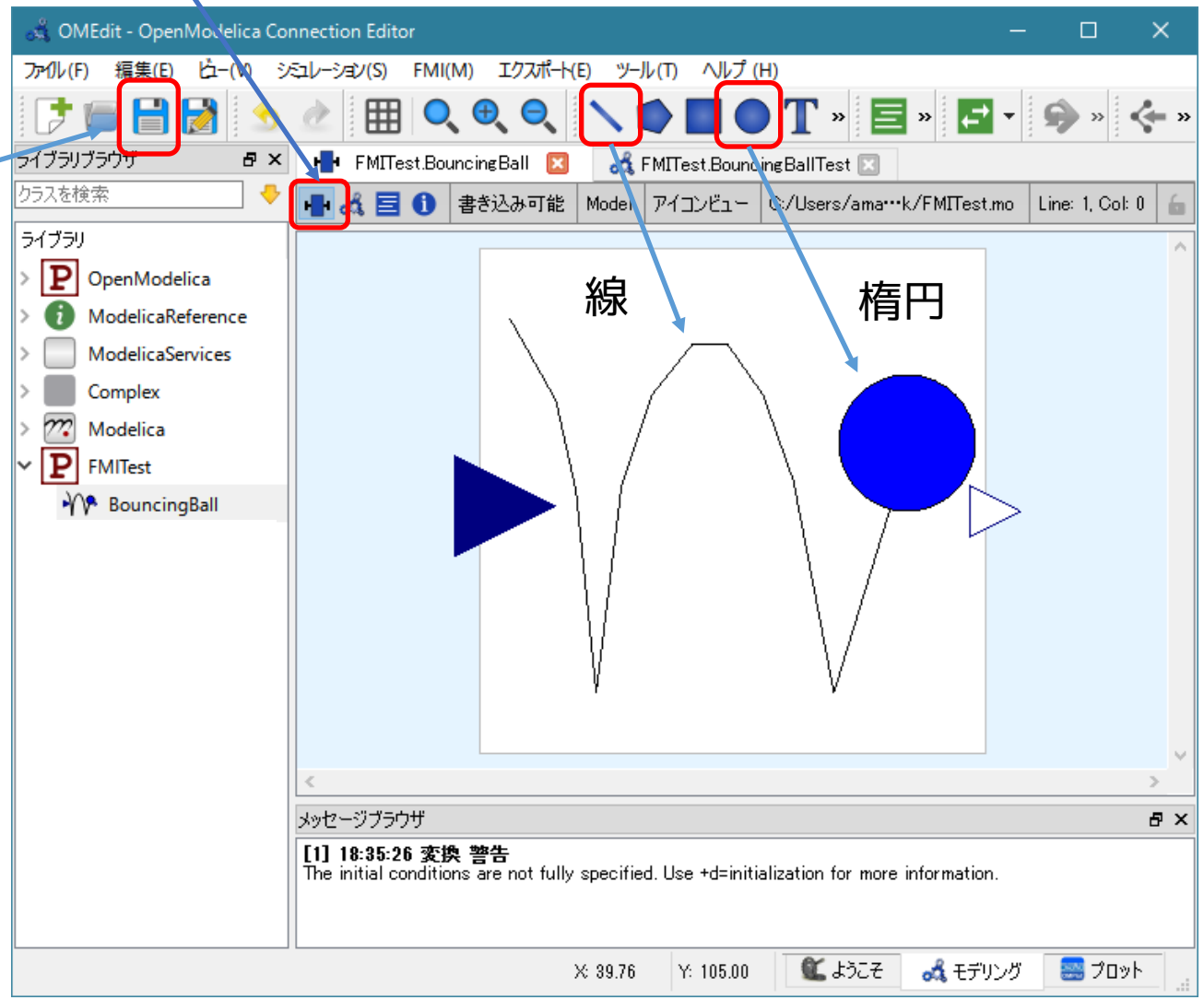
annotationが邪魔なときは、ツール＞オプション＞Modelicaテキストエディタで「行の折り返しを使用」のチェックをはずすと目ただなくなる。

2-1 OpenModelicaでサブシステムモデルをつくる

⑦アイコンビューに切り替えてアイコンを作成する。

⑧終わったら保存する。

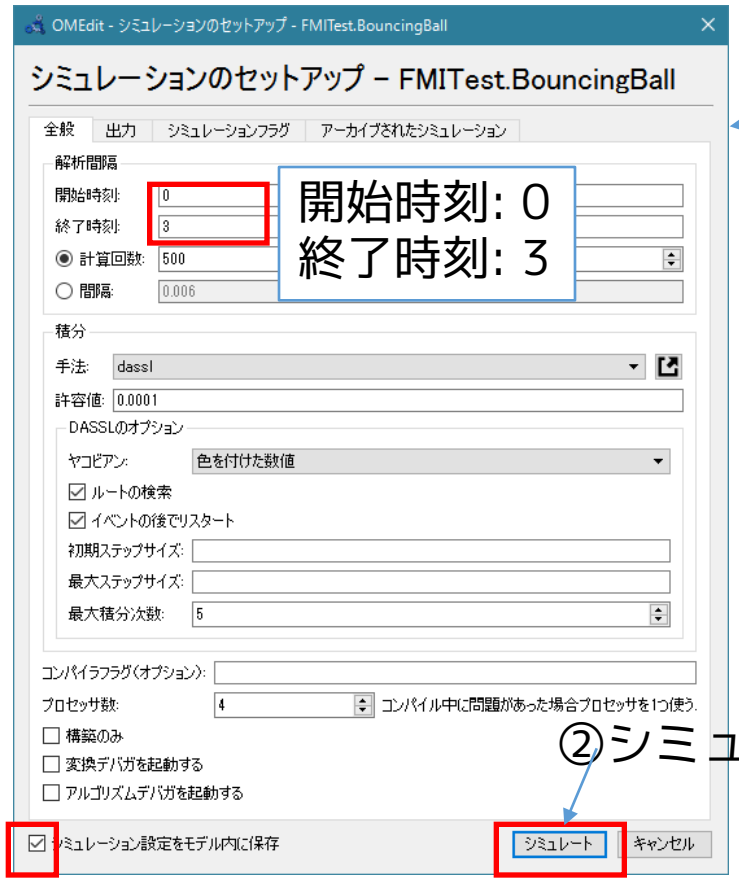
サブシステムモデル作成は
終了！
次は動作テスト
を行う。



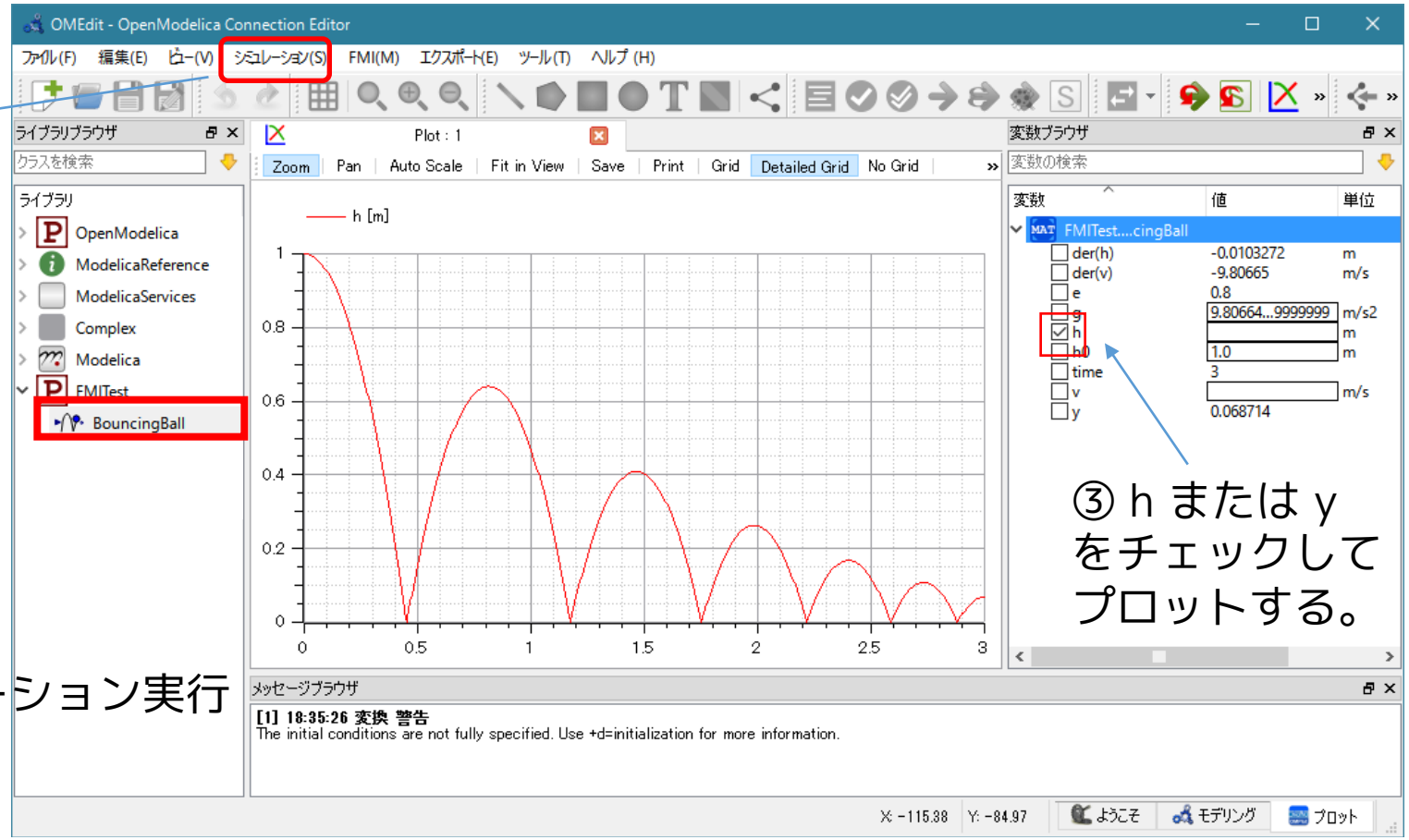
2-1 OpenModelicaでサブシステムモデルをつくる

テスト１：入出力なしのサブシステムモデル単体でのシミュレーション　　e=0.8で一定

①シミュレーション＞シミュレーションのセットアップ



開始時刻: 0
終了時刻: 3



③ｈまたはｙをチェックしてプロットする。

②シミュレーション実行

チェックする。（モデルの中に開始時間や終了時間に関する情報を入れる）

2-1 OpenModelicaでサブシステムモデルをつくる

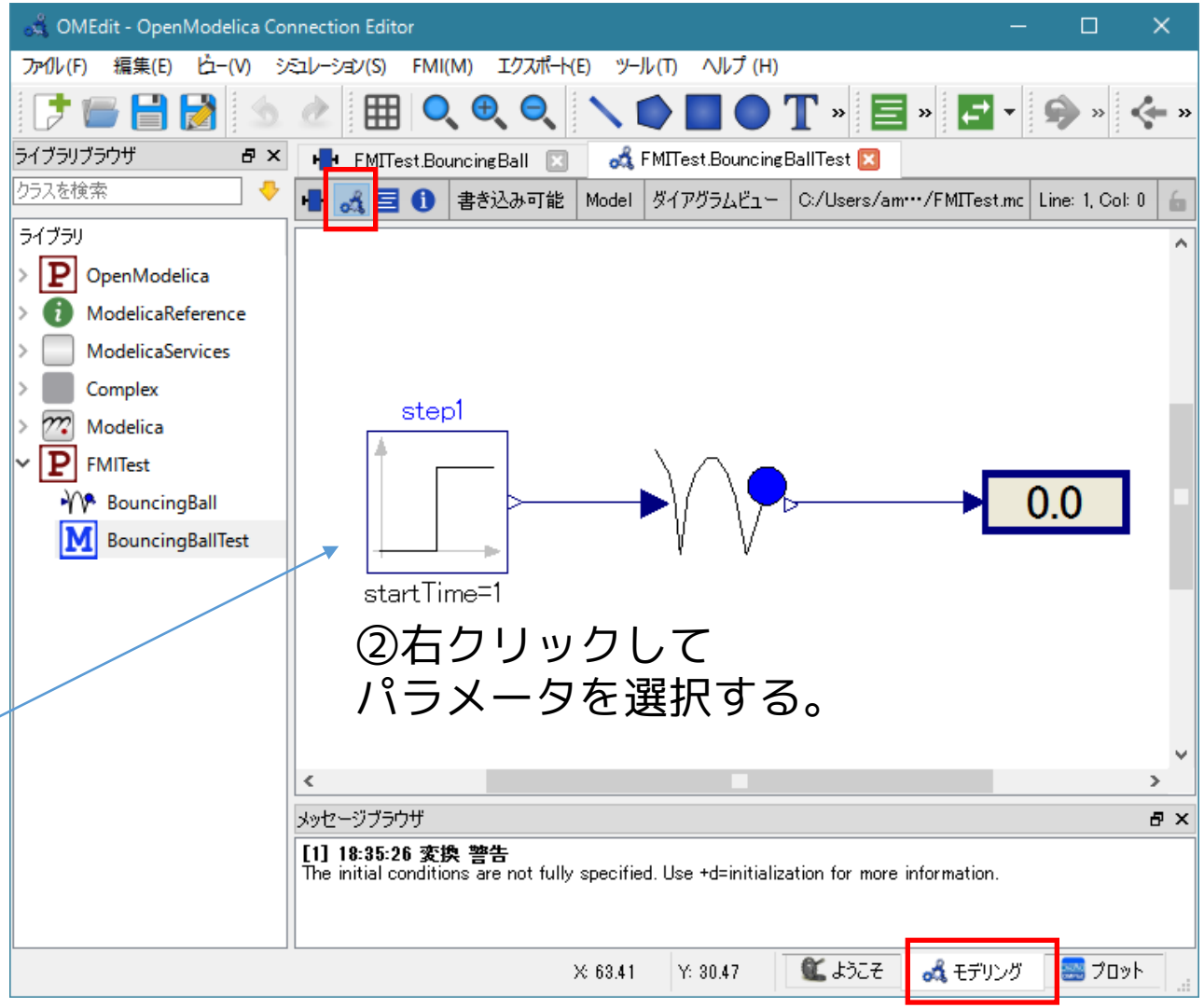
テスト２：入出力がある場合のサブシステムモデルのシミュレーション

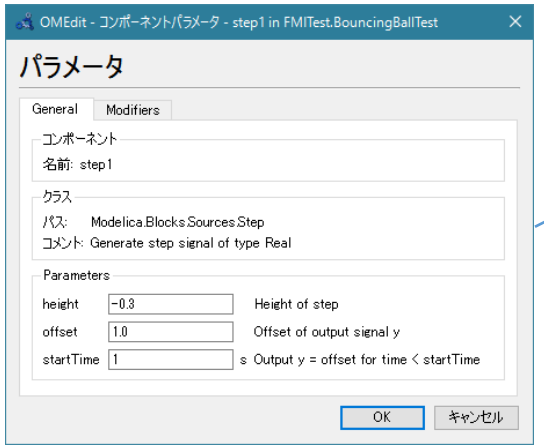
① FMITest右クリック＞Modelicaクラス新規作成

名前: BouncingBallTest
クラス・タイプ: Model

で新しいモデルを作成し、以下のモデルをライブラリブラウザからドラッグして配置し接続する。

- BouncingBall
- Modelica.Block.Sources.Step
- Modelica.Block.Interaction.Show.RealValue





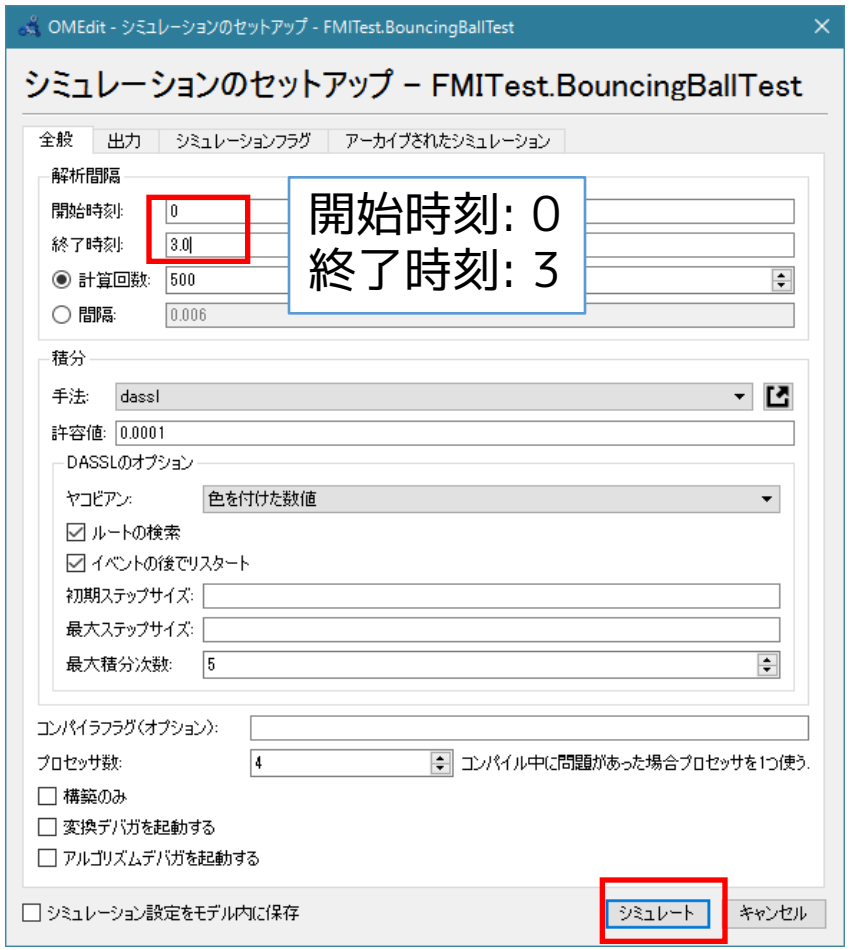
e を変化させる設定

height: -0.3
offset: 1.0
startTime: 1

2-1 OpenModelicaでサブシステムモデルをつくる

テスト２：入出力がある場合のサブシステムモデルのシミュレーション

③シミュレーション＞シミュレーションのセットアップ

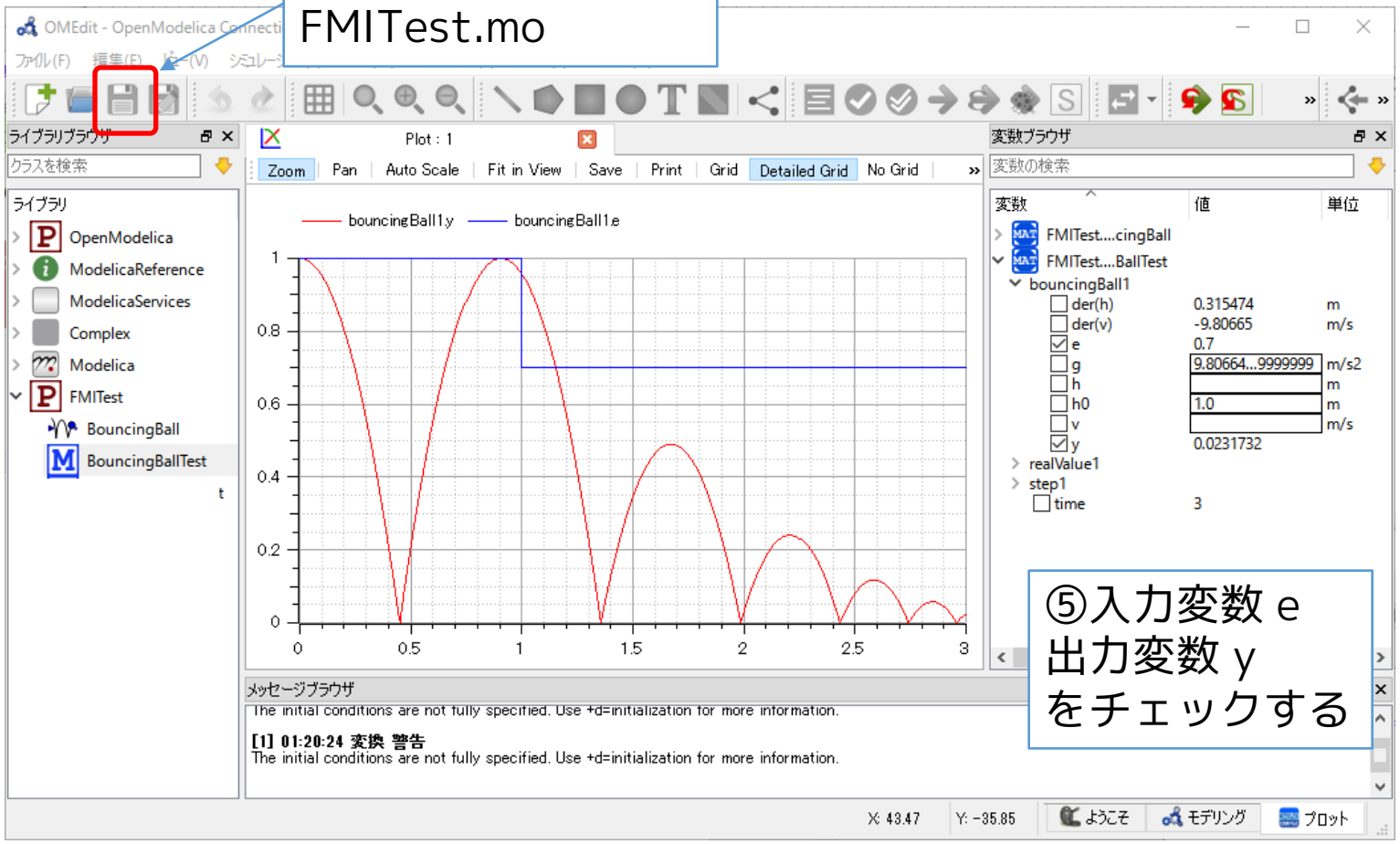


開始時刻: 0
終了時刻: 3

④シミュレーションの実行

⑥最後に保存する。
FMITest.mo

１秒までは e=1、その後　e= 0.7



⑤入力変数 e
出力変数 y
をチェックする

2-2 JModelica.org でCo-Simulation用のFMUをつくる

OpenModelica には、FMI 1.0 for Co-Simulation の仕様のFMUを生成する機能が無いので
JModelica.org を使用して作成する。

JModelica.org 1.17 windows版を使用する。
(http://www.jmodelica.org/)

① スタートメニューの JModelica.org-1.17の中の
IPython か Pythonのコンソールを起動する。
② OpnemModelicaで保存したファイルのある
ディレクトリをカレントディレクトリにして
以下を入力する。

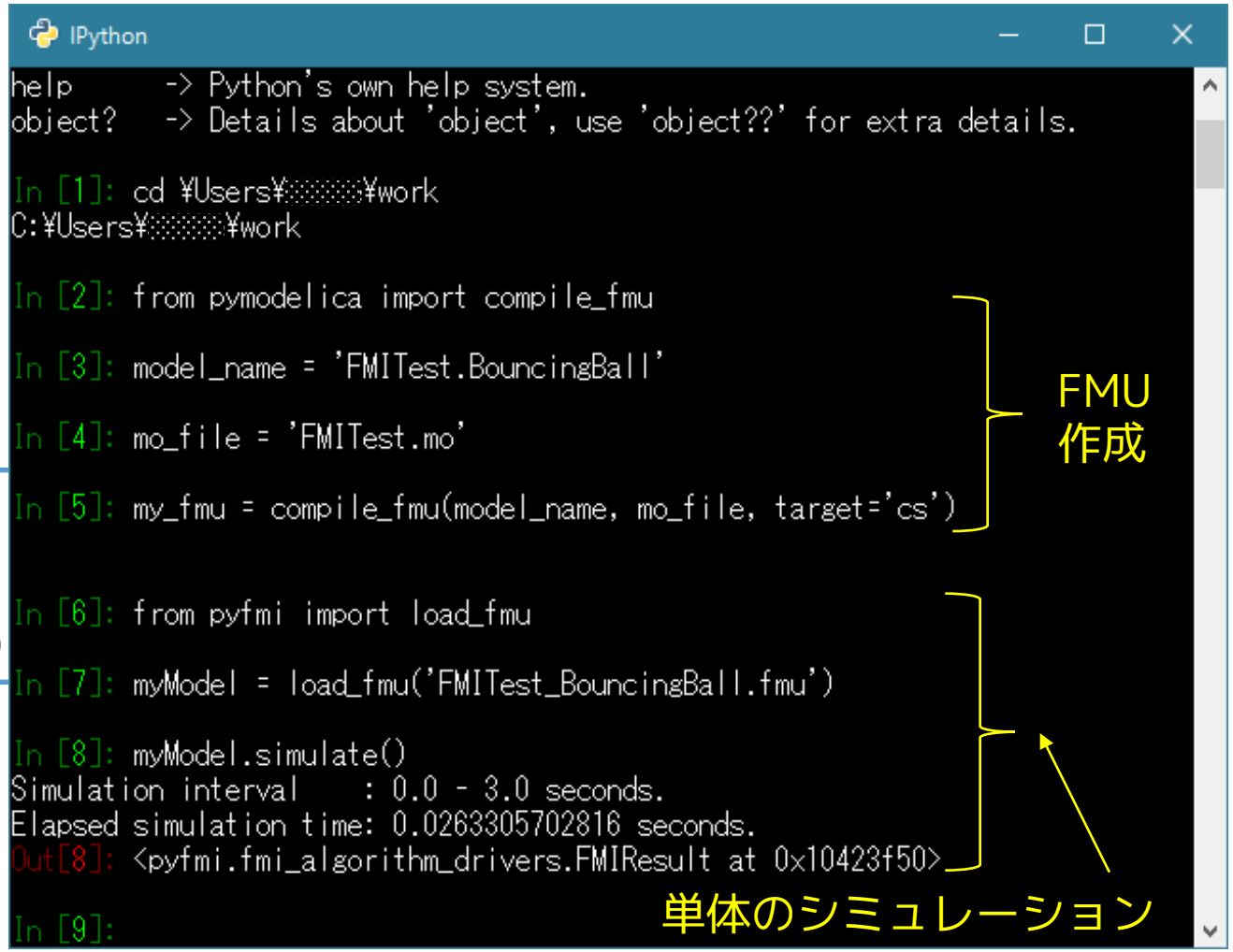
```
from pymodelica import compile_fmu
model_name = 'FMITest.BouncingBall
mo_file = 'FMITest.mo'
my_fmu = compile_fmu(model_name, mo_file, target='cs')
```

Co-Simulation用

FMITest_BouncingBall.fmu が生成される。

③ 以下を入力し、単体のシミュレーションを行う。

```
from pyfmi import load_fmu
myModel = load_fmu('FMITest_BouncingBall.fmu')
myModel.simulate()
```



JModelica環境の IPython コンソール

2-2 JModelica.org でCo-Simulation用のFMUをつくる

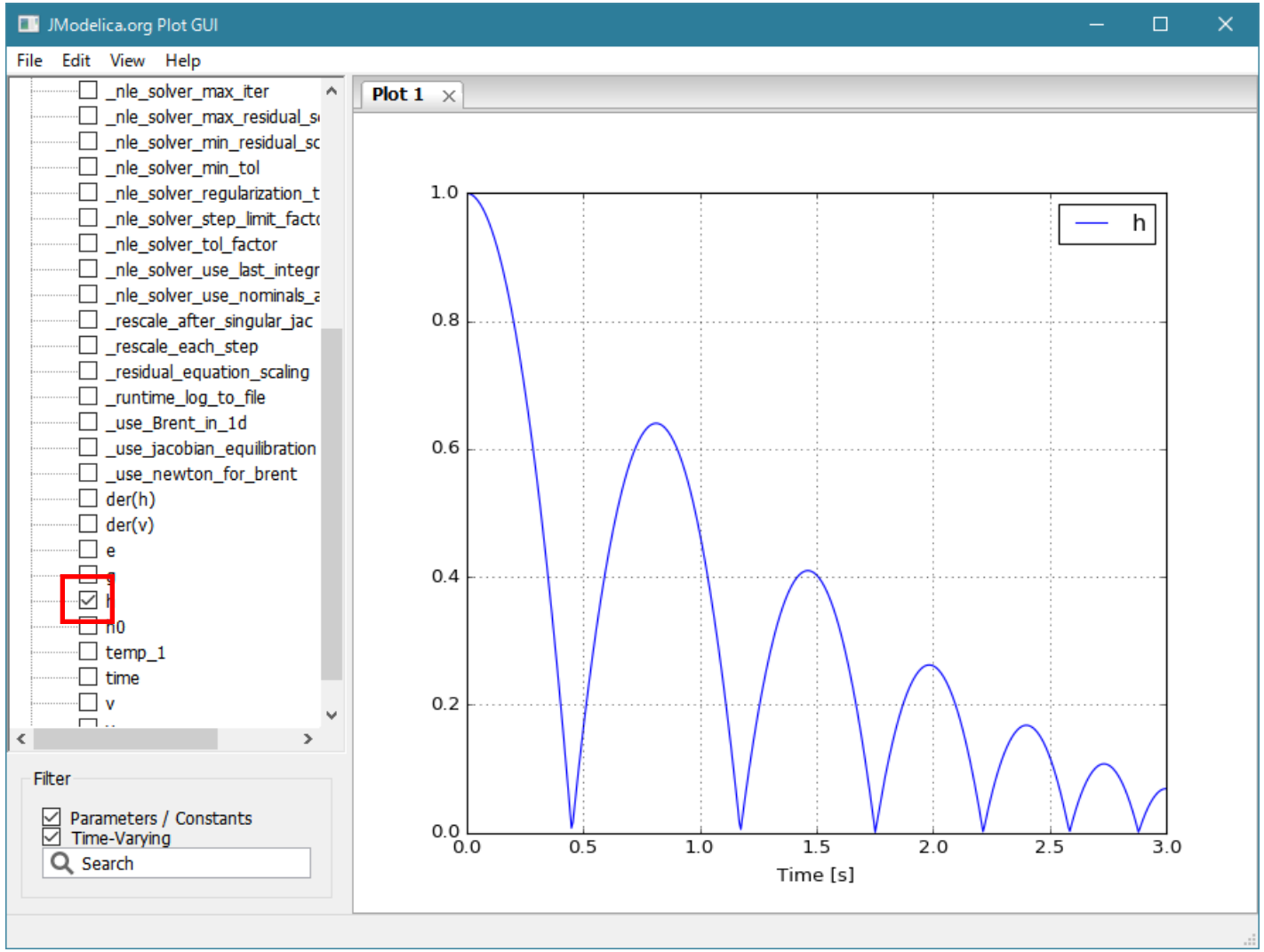
テスト１：入出力がない単体の
シミュレーション

JModelica.orgのplot-GUIで
結果を表示する。

④スタートメニューのJModelica-1.17
　の中の plot-GUI を起動する。

⑤File＞Openで
　FMITest_BoucingBall_result.txt
　を選択する。

⑥ツリーを展開し、
　h または y をチェックして
　プロットする。



2-3 JModelica.org でFMUを読み込む

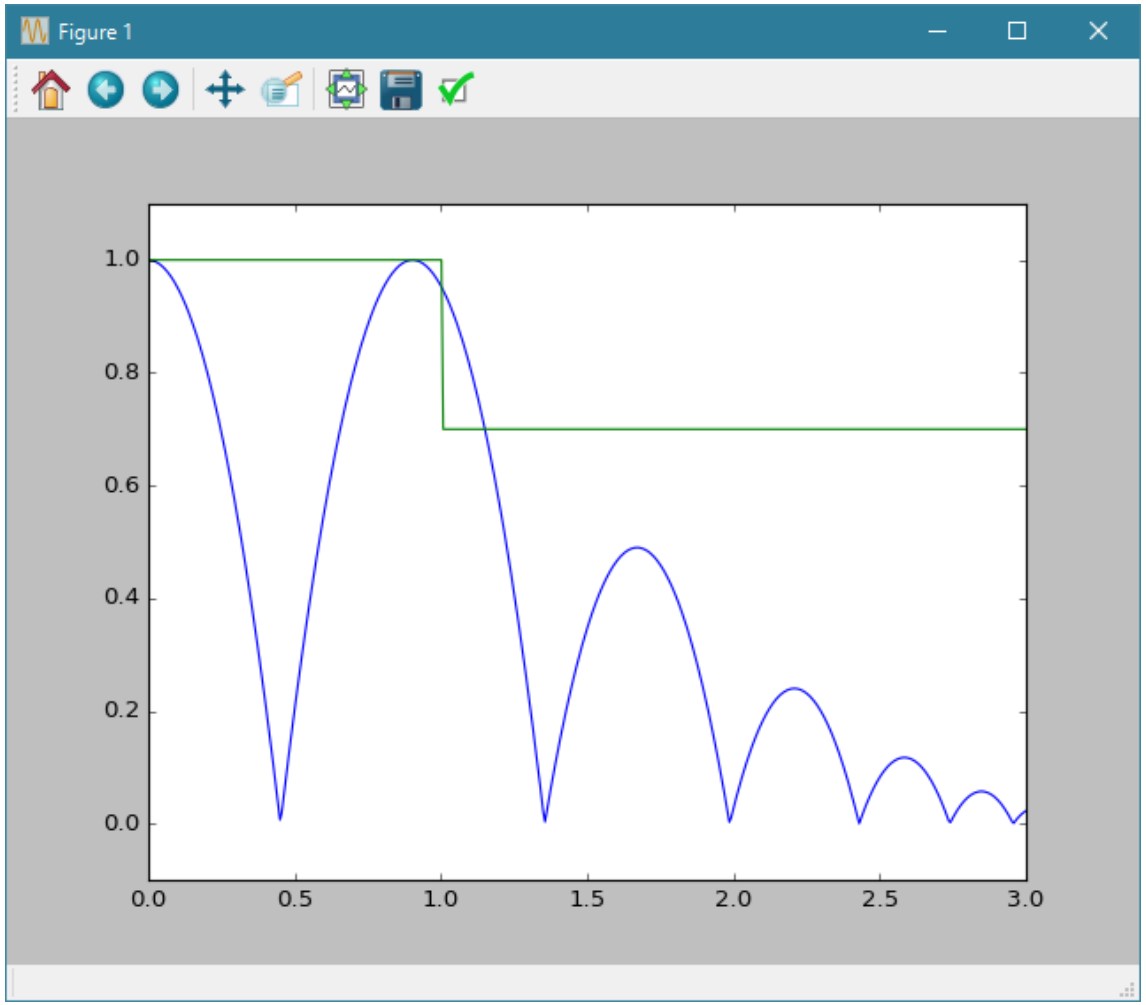
テスト２：入出力がある場合のシミュレーション

JModelica.org では GUI によるシステム
モデルの構築はできないので次のスライドに
示すPythonスクリプト
　bouncingBallTest.py
を実行することによって
入出力がある場合のテストを行う。

⑦ JModelica.org-1.17のIPython
　コンソールで以下を実行する。

```
run "bouncingBallTest.py"
```

図化に Matplotlib を使用した。
(http://matplotlib.org/)



boundingBallTest.py

```
# -*- coding: utf-8 -*-
"""
Created on Tue May 10 01:18:52 2016

@author: finback
"""

import pylab as P
import pyfmi

def stepf(t):
    startTime = 1.0
    if t < startTime:
        return 1.0
    else:
        return 0.7

model = pyfmi.load_fmu("FMITest_BouncingBall.fmu")
t_start = 0.0
t_end = 3.0
dt = (t_end-t_start)/500
t_res = []
e_res = []
y_res = []
t = t_start
e = stepf(t)
model.set("e",e)
model.initialize()
y = model.get("y")
t_res.append(t)
e_res.append(e)
y_res.append(y[0])
while t < t_end :
    status = model.do_step(t,dt)
    if status == pyfmi.fmi.FMI_OK:
        y = model.get("y")
        e_res.append(e)
        y_res.append(y[0])
        t += dt
        t_res.append(t)
        e = stepf(t)
        model.set("e",e)
    else:
        print "error"
        break

P.plot(t_res,y_res)
P.plot(t_res,e_res)
P.xlim(t_start,t_end)
P.ylim(-0.1,1.1)
P.show()
```

入力変数の変化を表す関数

モデルのインスタンス化

結果用配列

コミュニケーションステップサイズ計算

入力変数の初期設定とモデルの初期化

初期化された出力変数取得

コミュニケーションステップの実行

出力変数取得

次のコミュニケーションステップの計算

入力変数の計算と設定

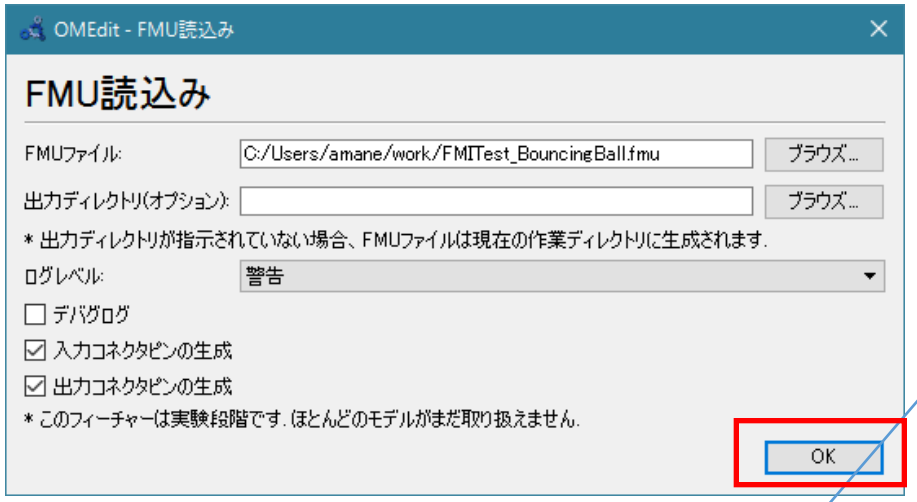
図化処理

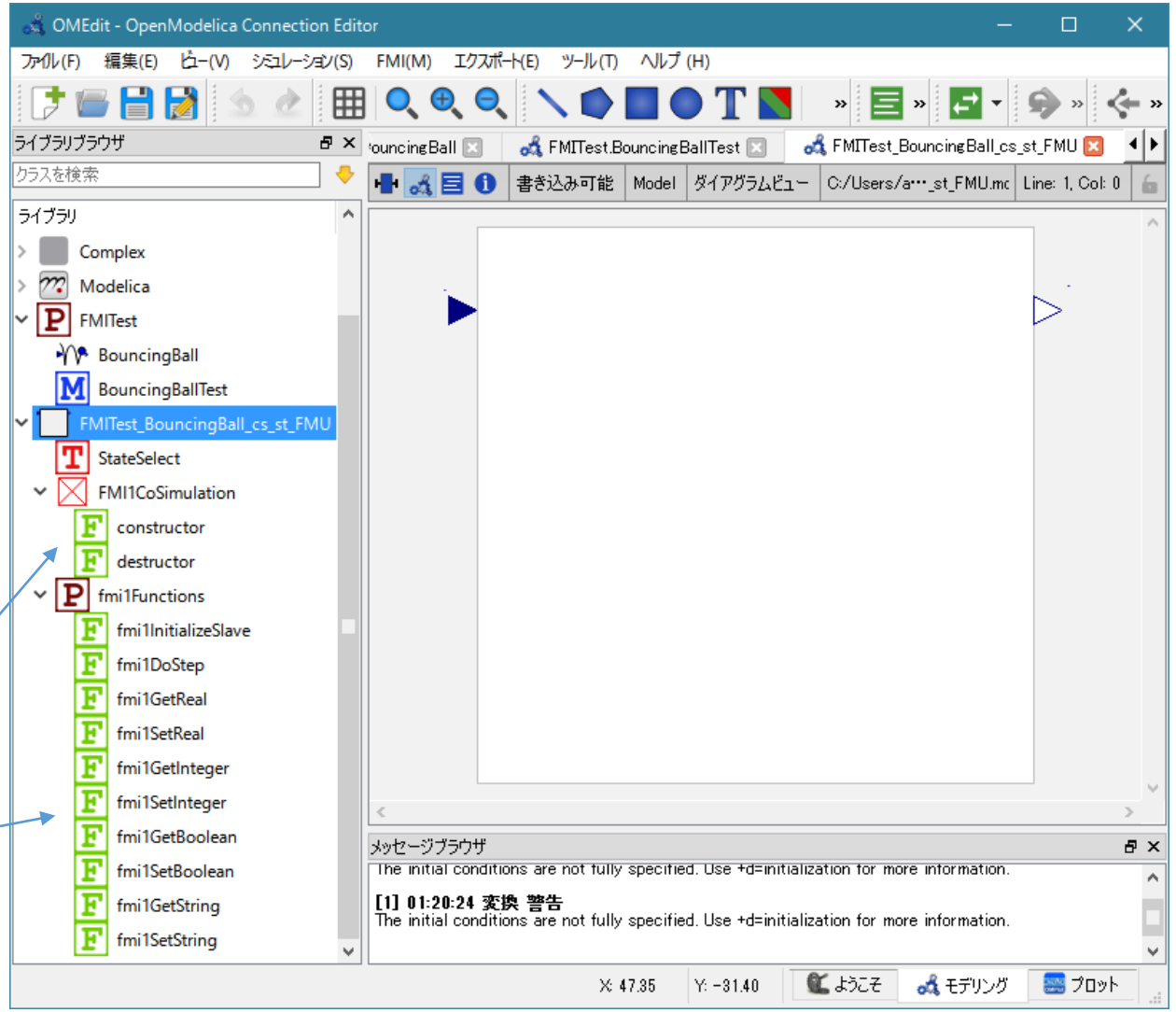
2-4 OpenModelicaでFMUを読み込む

JModelica で作成したFMUを
OpenModelica で読んでみる。

①FMI(M)>FMU読込み

FMIファイル：FMITest_BouncingBall.fmu



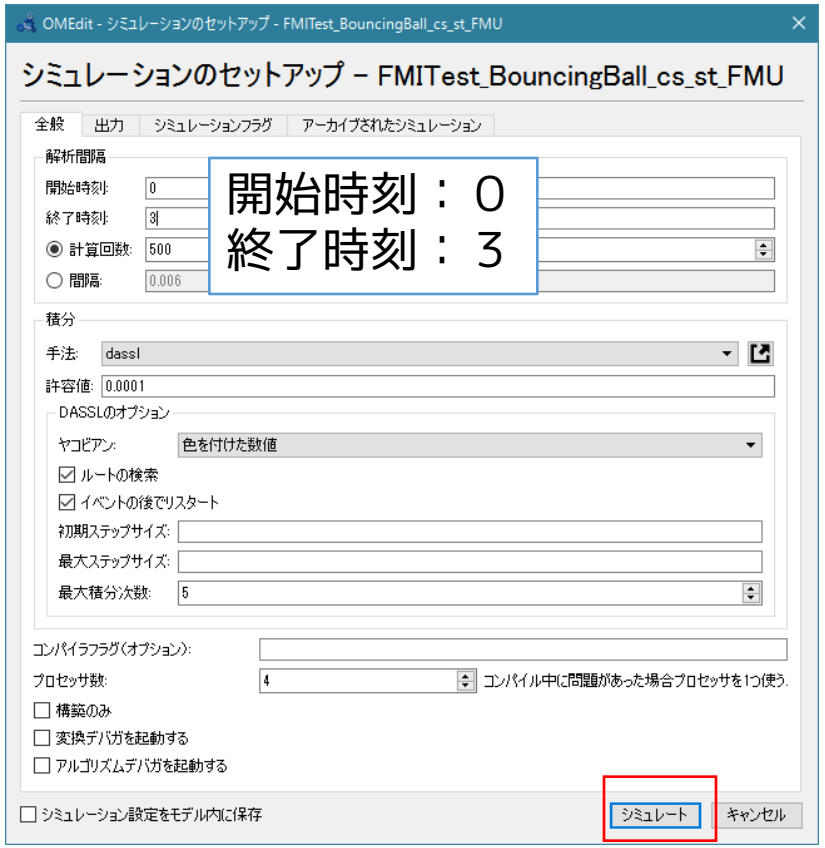


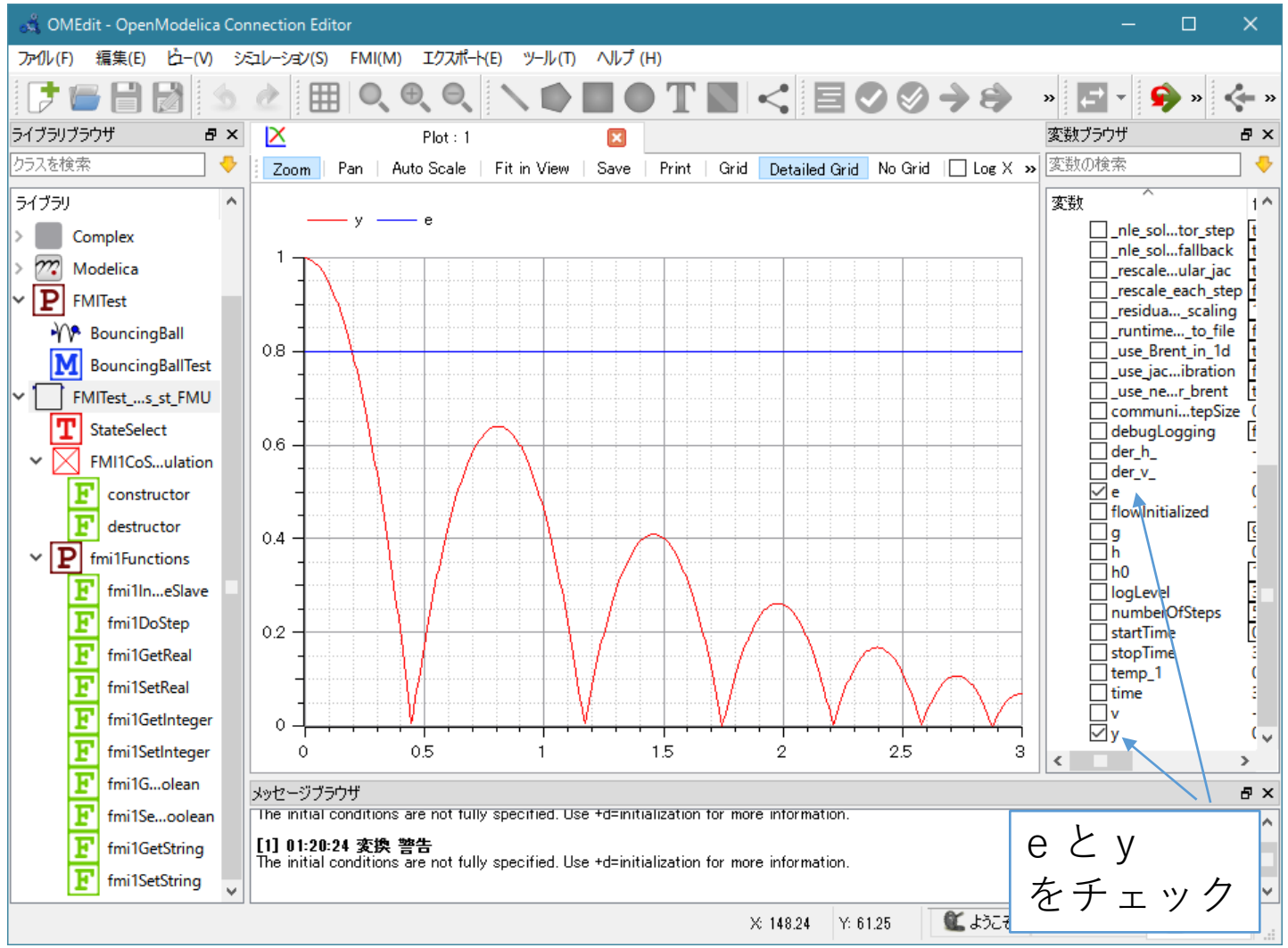
ツリーを展開すると
FMIのインターフェース関数に相当する
関数が表示される。

2-4 OpenModelicaでFMUを読み込む

テスト１：入出力のない単体の
シミュレーション

②シミュレーション
＞シミュレーションのセットアップ



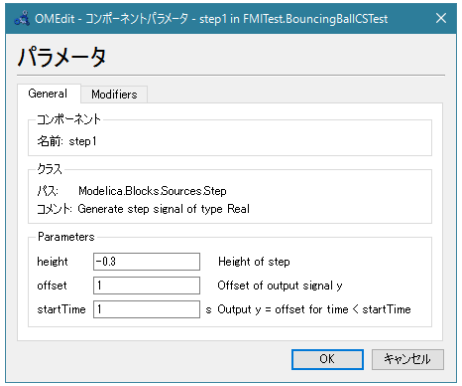


2.4 OpenModelicaでFMUを読み込む

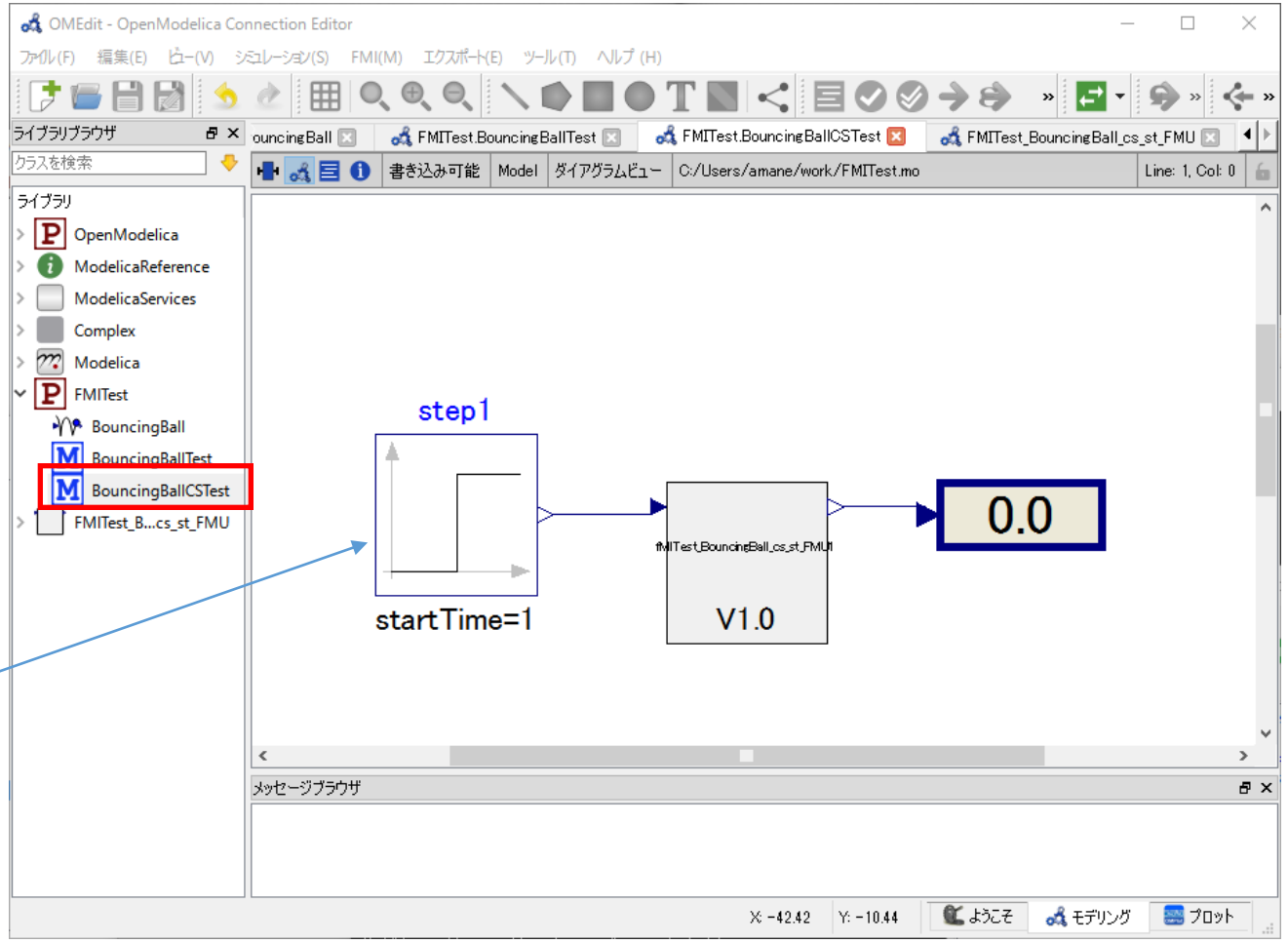
テスト２：入出力のある場合の
シミュレーション

③新しいモデル
BouncingBallCSTest
を作成し、以下をライブラリから
ドラッグ・アンド・ドロップし、
接続する。

- FMITest_BouncingBall_cs_st_FMU
- Modelica.Block.Sources.Step
- Modelica.Block.Interaction.Show.RealValue



height: -0.3
offset: 1.0
startTime: 1



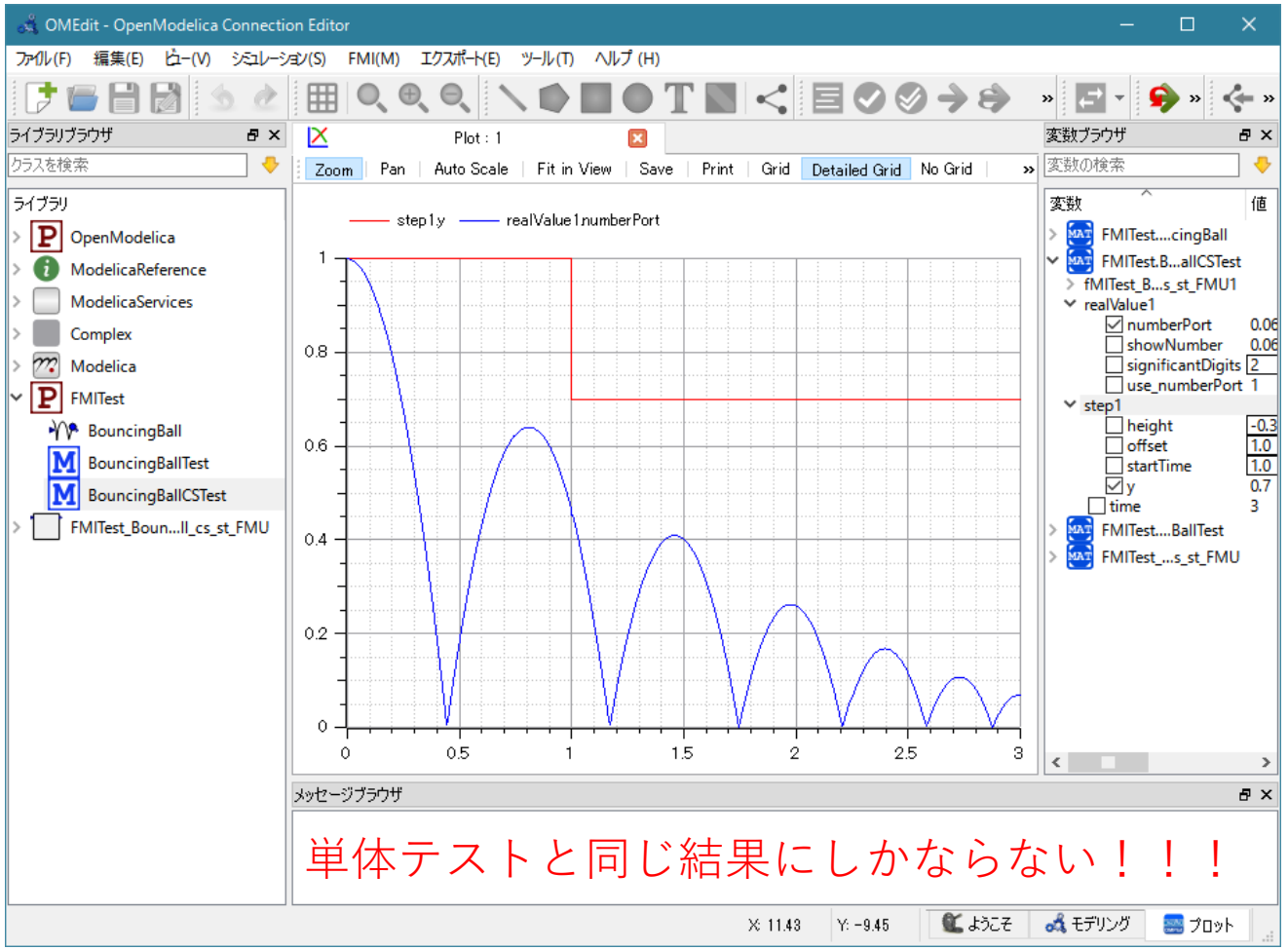
2-4 OpenModelicaでFMUを読み込む

④シミュレーション
　＞シミュレーションのセットアップ

開始時刻：０
終了時刻：３
でシミュレーションを実行する。

入力変数 e をステップ的に
変化させたのに、
出力変数 y の計算結果に反映
されない!!

ログファイルを見ると、
入力変数をセットする関数
fmiSetReal がコールされて
いないようである。

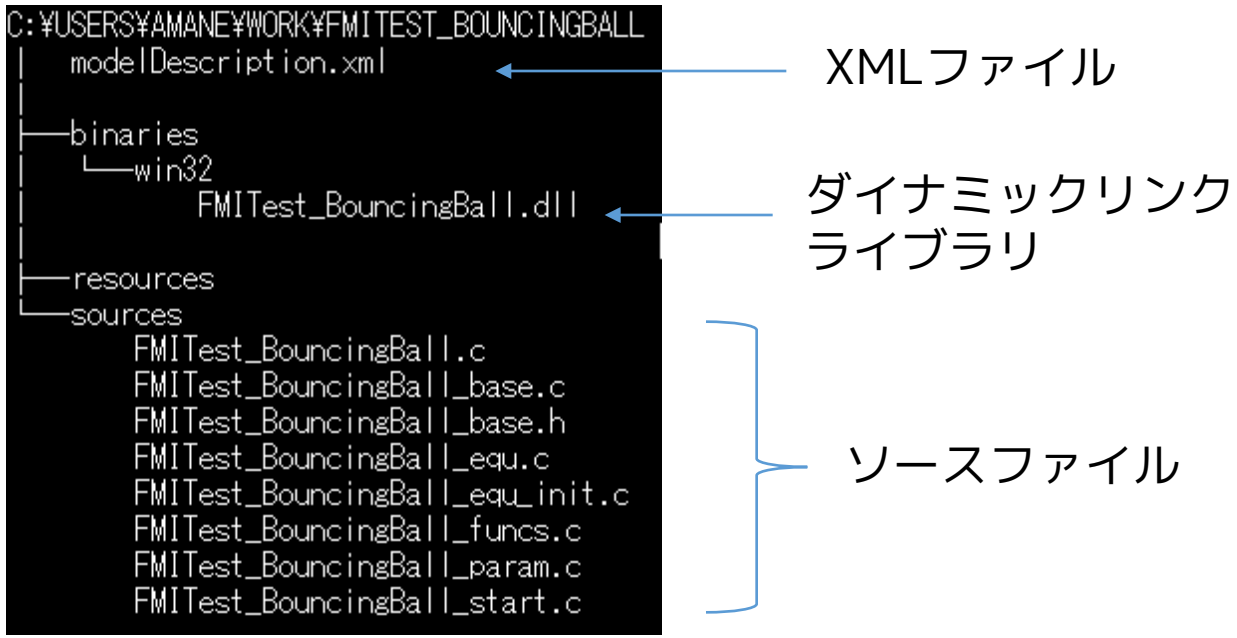


OpenModelica FORUM
でも報告されている。
https://www.openmodelica.org/forum/default-topic/1639-data-exchange-between-fmu-model-and-openmodelica-block

2-5 Scilab/Xcos FMU wrapperでFMUを読み込む

JModelica.org で作成した
FMITest_BouncingBall.fmu
をコピーして次のようにリネームする。
FMITest_BouncingBall.zip
解凍して中身を確認する。

Windows用32bitのダイナミック
ライブラリが生成されている。



Scilab/Xcos もWindows 32bit版を使用する。

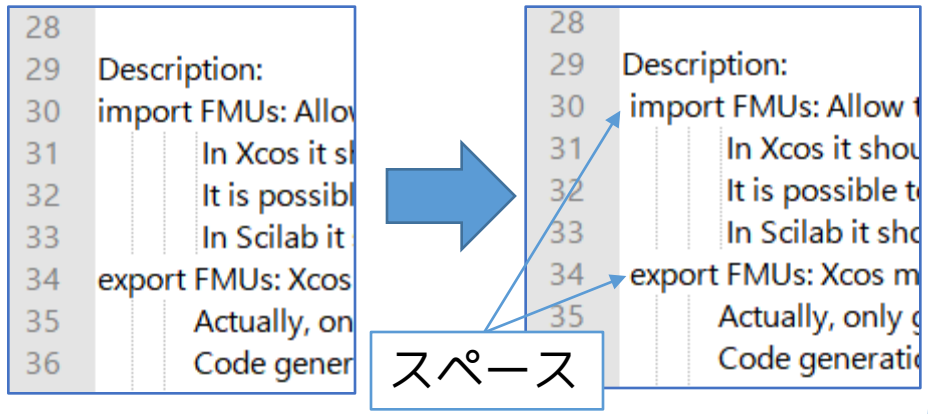
Scilab には ATOMS というモジュール管理システムがあり、atomsInstall というコマンドでインストールする。
atomsInstall で FMU wrapper をインストールするには、少し問題があり手直しが必要である。
次スライドの手順でインストールする。

2-5 Scilab/Xcos FMU wrapperでFMUを読み込む

Scilab/Xcos FMU wrapper (https://forge.scilab.org/index.php/p/fmu-wrapper/) のインストールに関するワークアラウンド

① fmu-wrapper-0.6.zip を解凍する。

② ファイル DESCRIPTION の次の行（３０行目）以降で、１桁目から始まっている全ての行の１桁目にスペースを挿入する。



③ etc¥fmu_wrapper.start の77行目の pwd() を root_tlbx に変更する

```
76    for i = 1:3
77      copyfile(root_tlbx+fullfile("¥src¥c")+filesep()+fmiHeader(i), TMPDIR);
78    end
79  // Load and add help chapter
```

④ 再び zip で fmu-wrapper-0.6.zip を圧縮する。

⑤ Scilab を起動し、ファイルブラウザを fmu-wrapper-0.6.zip のあるディレクトリに移動して以下を実行する。

```
atomsInstall('fmu-wrapper-0.6.zip')
```

⑥ ファイルブラウザを %userprofile%¥Roaming¥Scilab¥scilab-5.5.2¥atoms¥fmu_wrapper¥0.6に移動して以下を実行する。

```
exec builder.sce
```

⑦ Scilabを再起動する。

Scilabのインストール先を変えた場合は、Scilabのインストール先¥contrib¥fmu_wrapper¥0.6

https://forge.scilab.org/index.php/p/fmu-wrapper/issues/1462/ に情報あり。

2-5 Scilab/Xcos FMU wrapperでFMUを読み込む

Scilab によるスタンドアロンタイプのFMUの単体のシミュレーション
はサポートされていない。

https://forge.scilab.org/index.php/p/fmu-wrapper/issues/1548/

Scilab コンソール

```
-->fmu = importFMU("FMITest_BouncingBall.fmu")
 fmu  =

FMITest_BouncingBall FMU Model

-->out = fmiSimulate(fmu)
 !--error 999
fmu_call: Wrong type for input argument #1: FMU expected.
at line      11 of function fmiGetVersion called by :
at line      50 of function fmiSimulate called by :
out = fmiSimulate(fmu)
```

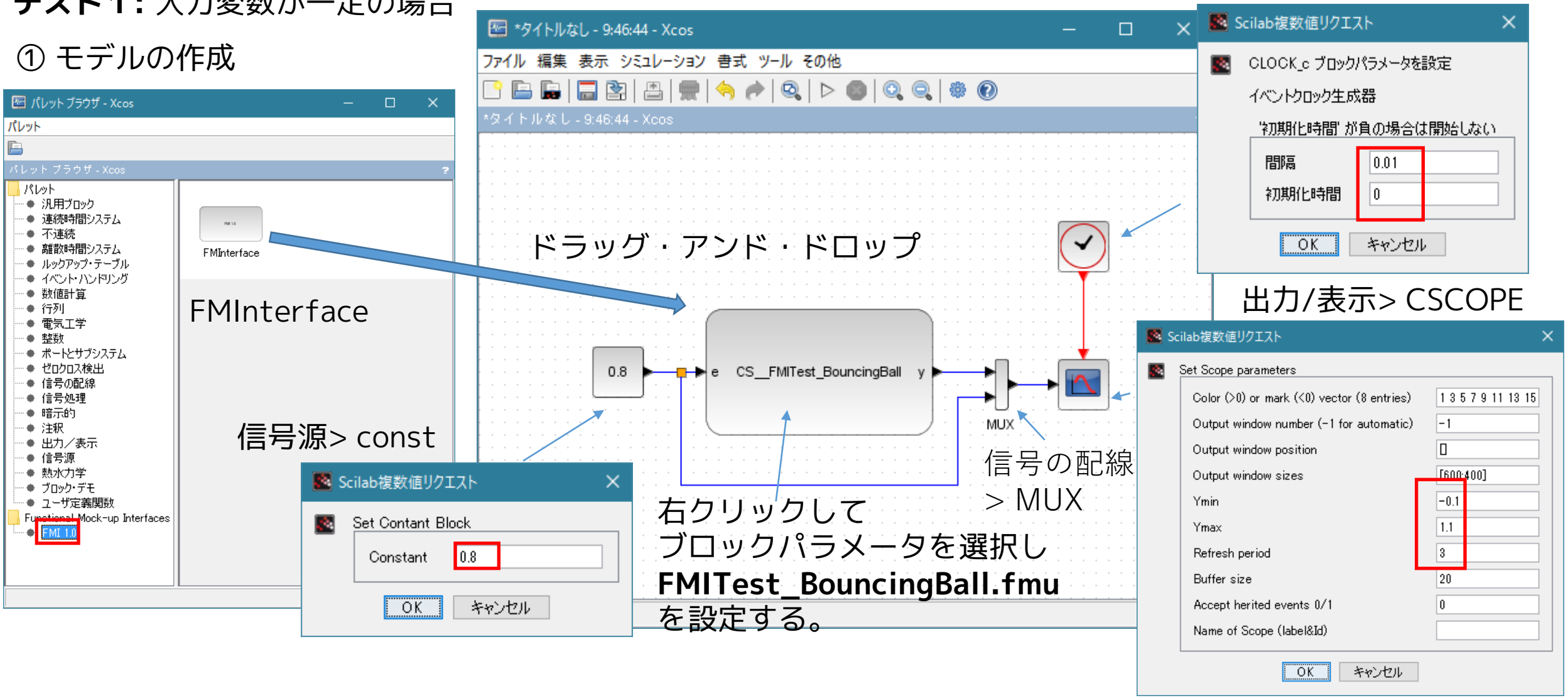
エラー

Xcosによるテストのみを行う。

2-5 Scilab/Xcos FMU wrapperでFMUを読み込む

テスト1： 入力変数が一定の場合

① モデルの作成

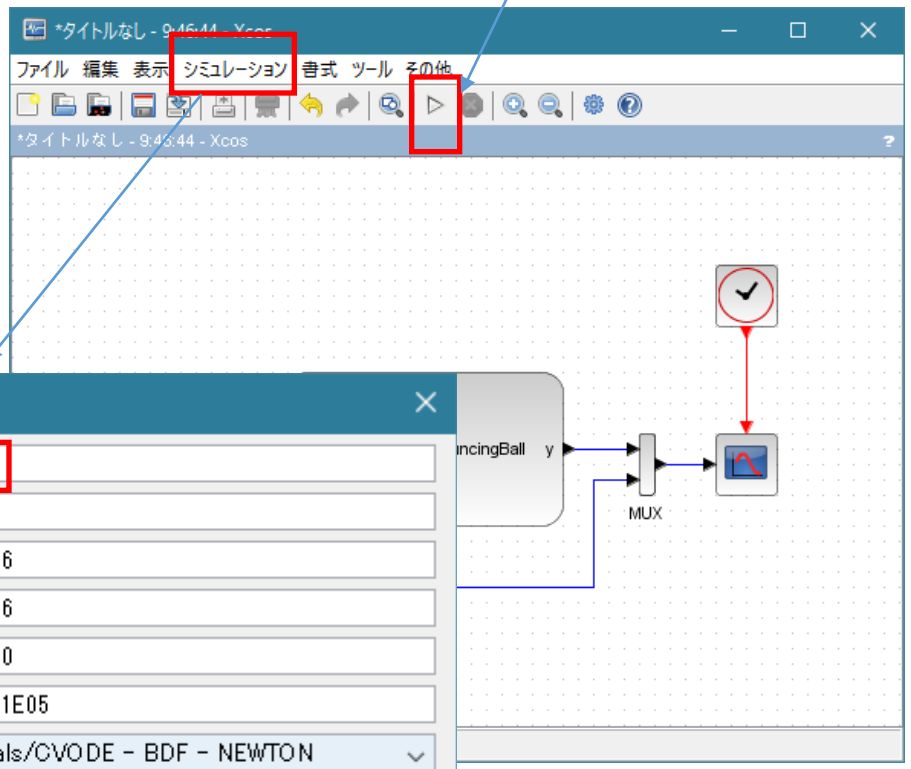


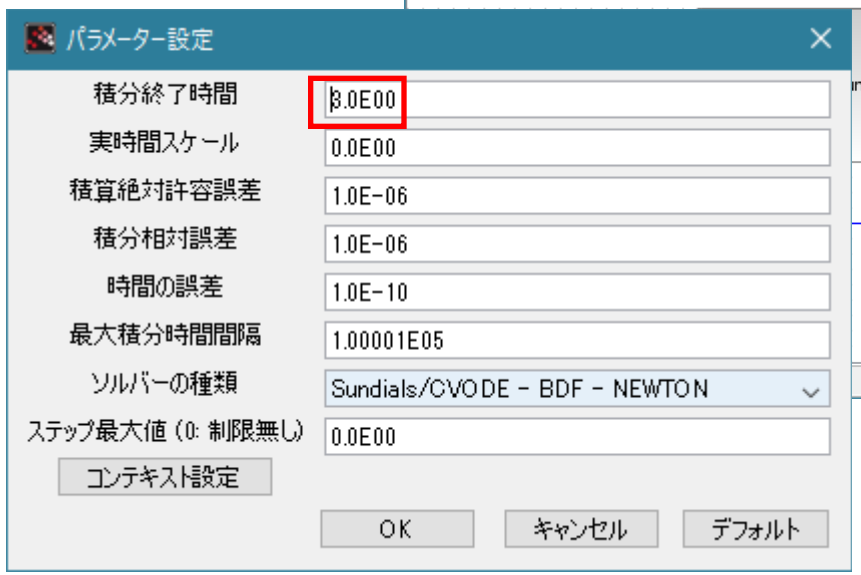
2-5 Scilab/Xcos FMU wrapperでFMUを読み込む

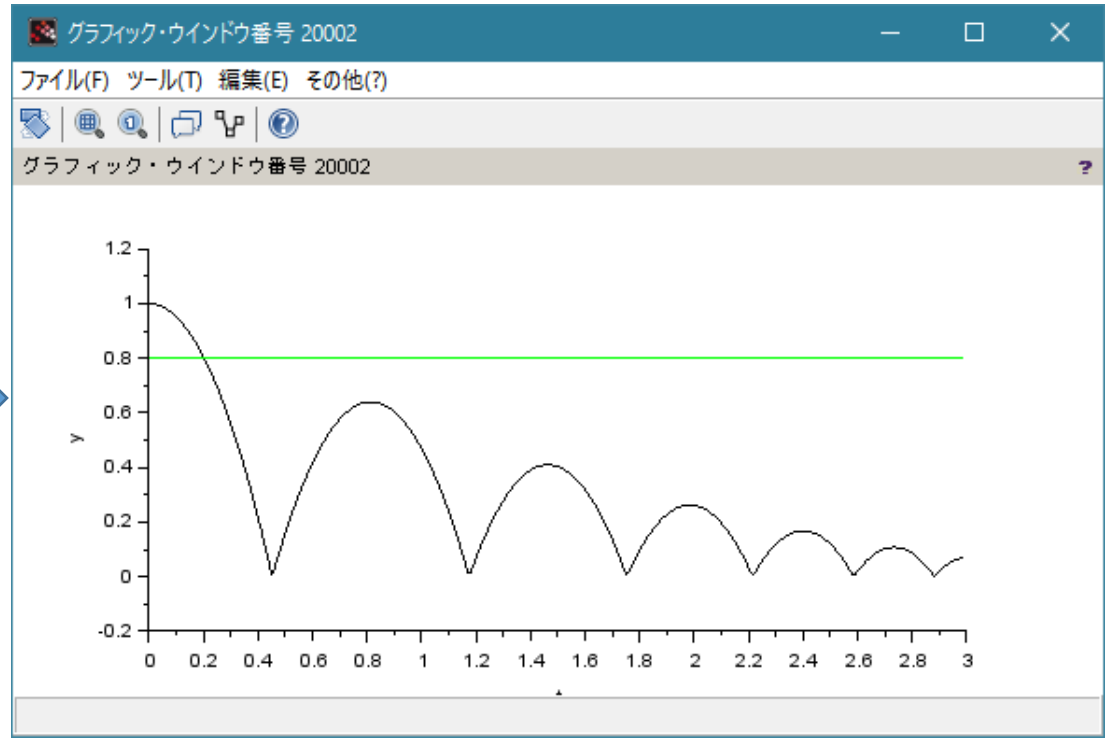
テスト1： 入力変数が一定の場合

② シミュレーション＞設定

③ 開始



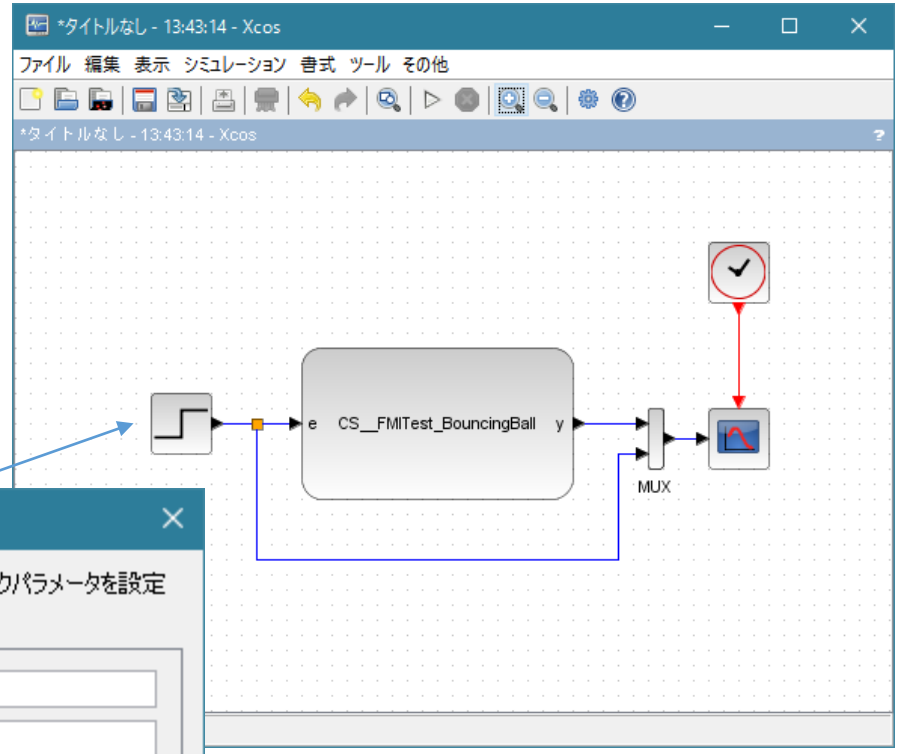


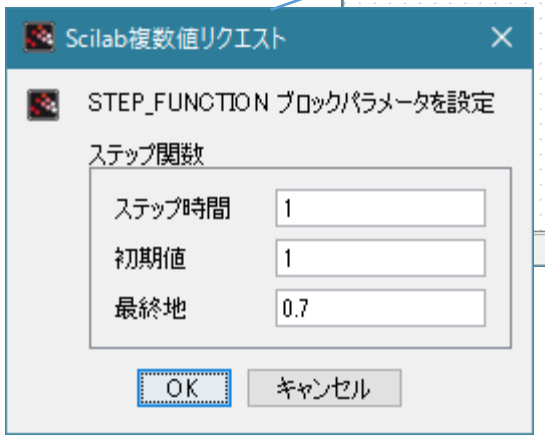


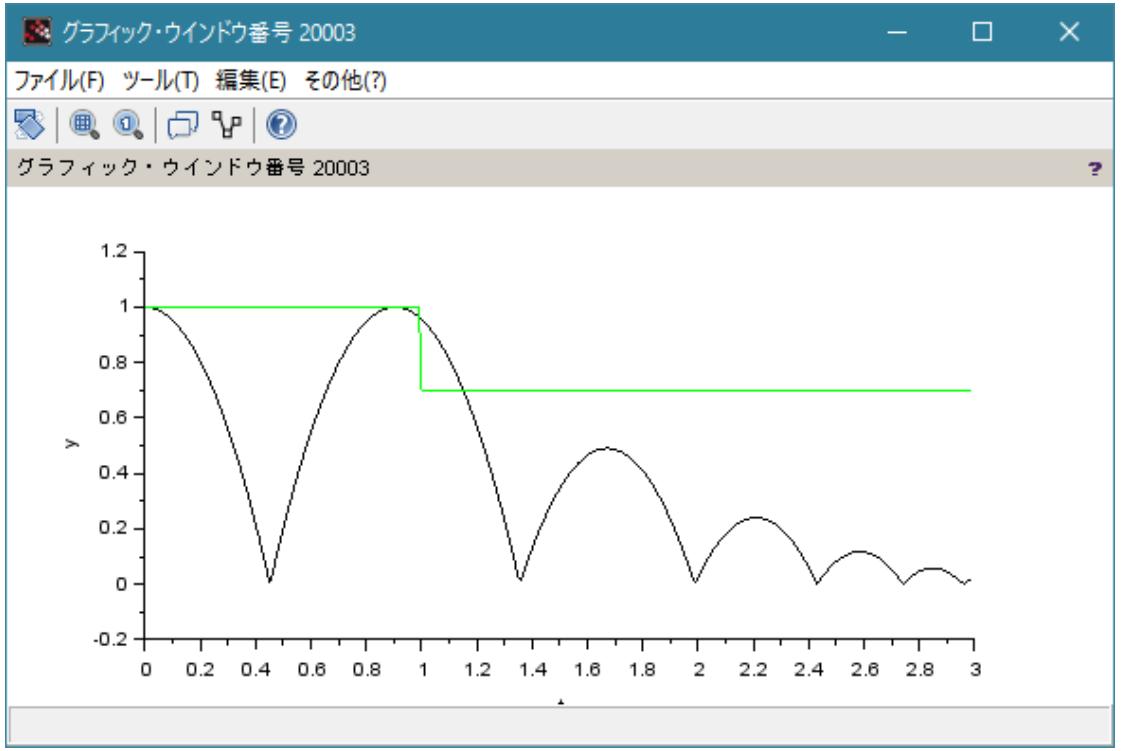
2-5 Scilab/Xcos FMU wrapperでFMUを読み込む

テスト2: 入力変数が変化する場合

入力信号をステップ関数に変えて実行する。





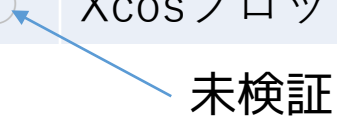


信号源
>STEP_FUNCTION

各ツールのFMI 1.0 FMI for Co-Simulation の対応状況のまとめ

FMI 1.0 FMI for Co-Simulation の FMUの作成　スレーブツール

ツール		モデル作成環境	ソルバー
FMU SDK	○	C言語	Euler法
OpenModelica	×		
OpenModelica+ JModelica.org	○	Modelica言語	Euler法, cvode
Scilab/Xcos + Xcos FMU wrapper	○	Xcosブロック線図	



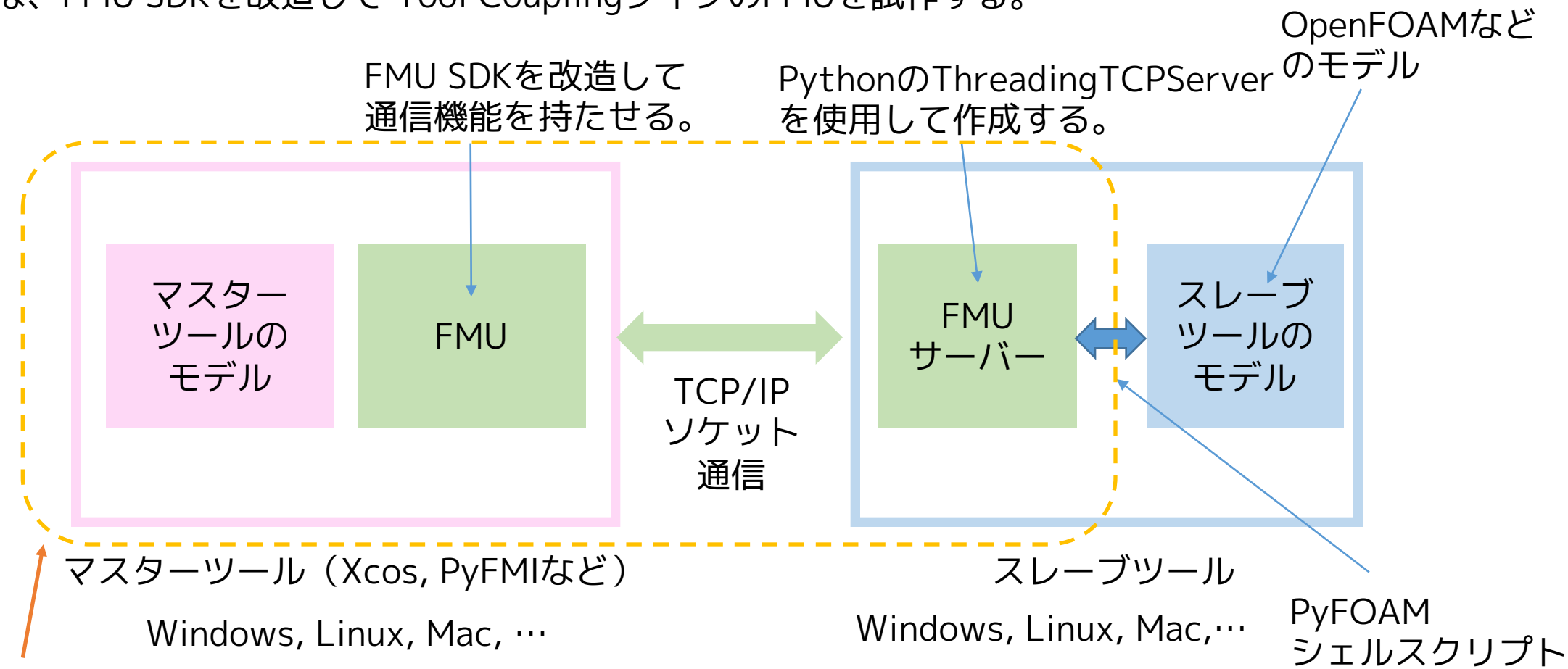
FMI 1.0 FMI for Co-Simulation の FMUの読み込み　マスターツール

ツール		モデル作成環境	備考
FMU SDK	△		入力変数なし
OpenModelica	△	Modelica言語	入力変数なし
JModelica.org (PyFMI)	○	Pythonスクリプト	
Scilab/Xcos + Xcos FMU wrapper	○	Xcosブロック線図	Euler法

3．Tool Coupling タイプのFMUを試作する

3．Tool Coupling タイプのFMUを試作する

FMU SDK, JModelica.org, Xcosで作成できるのは StandAlone タイプのFMUである。
ここでは、FMU SDKを改造して Tool Couplingタイプの FMUを試作する。



構想中のシステム

3．Tool Coupling タイプのFMUを試作する

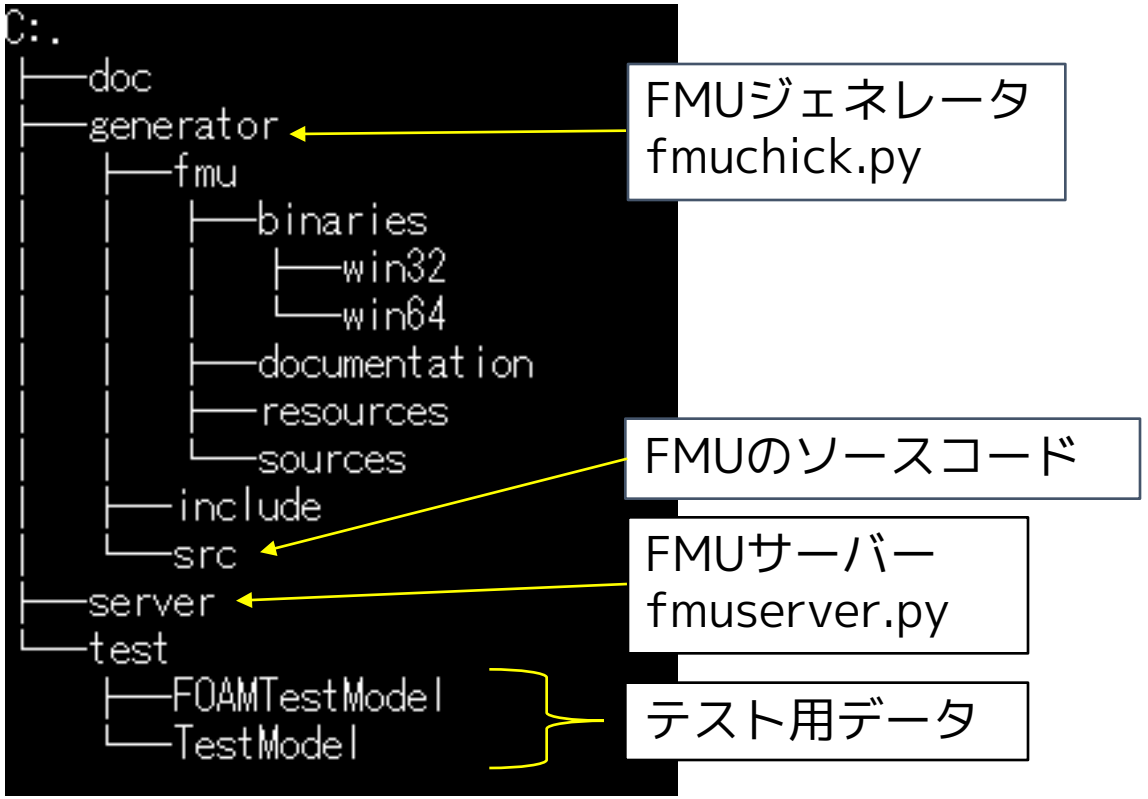
- 3-1 作成したもの
- 3-2 FMUジェネレータによる FMU の作成
- 3-3 Xcos による Co-Simulation モデルの作成
- 3-3 FMUサーバーのスレーブツール実行部分の編集
- 3-4 Co-Simulation の実行

3-1 作成したもの　fmuchick (https://github.com/finback-at/fmuchick)

FMUジェネレータ（Python)
- FMUのソースコード（**可変部、モデル固有の情報**を含む）を出力する。
- XMLファイルを生成する。
- バッチファイルを使用してコンパイルし、ZIP化を行う。

FMUのソースコード（不変部）(C言語)
- FMUサーバーと通信を行うクライアント。
- FMU SDKのCo-Simulation用のソースコードを改造し通信機能を持たせる。

FMUサーバ (Python)
- FMU本体の要求によってスレーブツールをコントロールするサーバ。
- Pythonの標準ライブラリを使用し、マルチプラットフォームを目指す。



開発およびテスト環境
- Windows 10
- Visual Studio 2012
- Python 2.7.11 (32bit)
- wxPython 3.0

3-2 FMUジェネレータによる FMU の作成

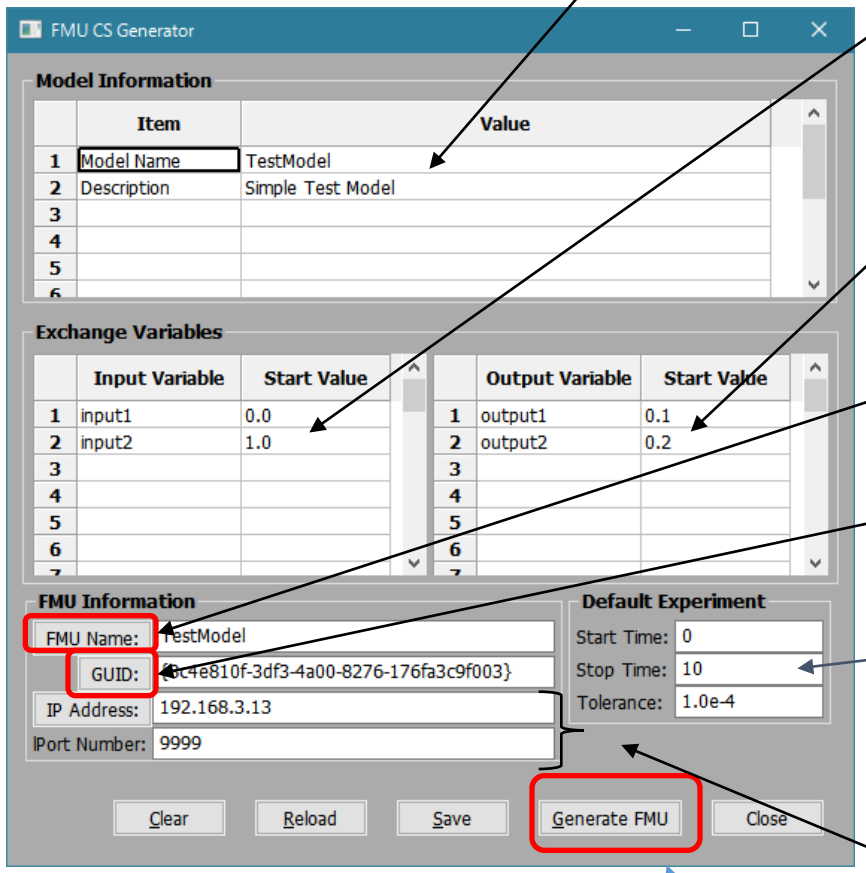
① ディレクトリgeneratorでコマンドプロンプトから
python fmuchick.py
を実行する。

② モデルデータを入力する。
- モデル情報
- 入力変数
- 出力変数
- ＦＭＵ名
- GUID
- サーバのIPアドレス
- サーバのポート番号

③ [Generate FMU]ボタンをクリックする。

④ FMU が生成される。

スレーブツールの実行に
必要なモデル情報など任意の情報
- Mode Name: TestModel
- Description: Simple Test Model



FMUジェネレータ

入力変数
input1: 0.0
input2: 1.0

出力変数
output1: 0.1
output2: 0.2

クリックするとModel Nameの値がコピーされる。

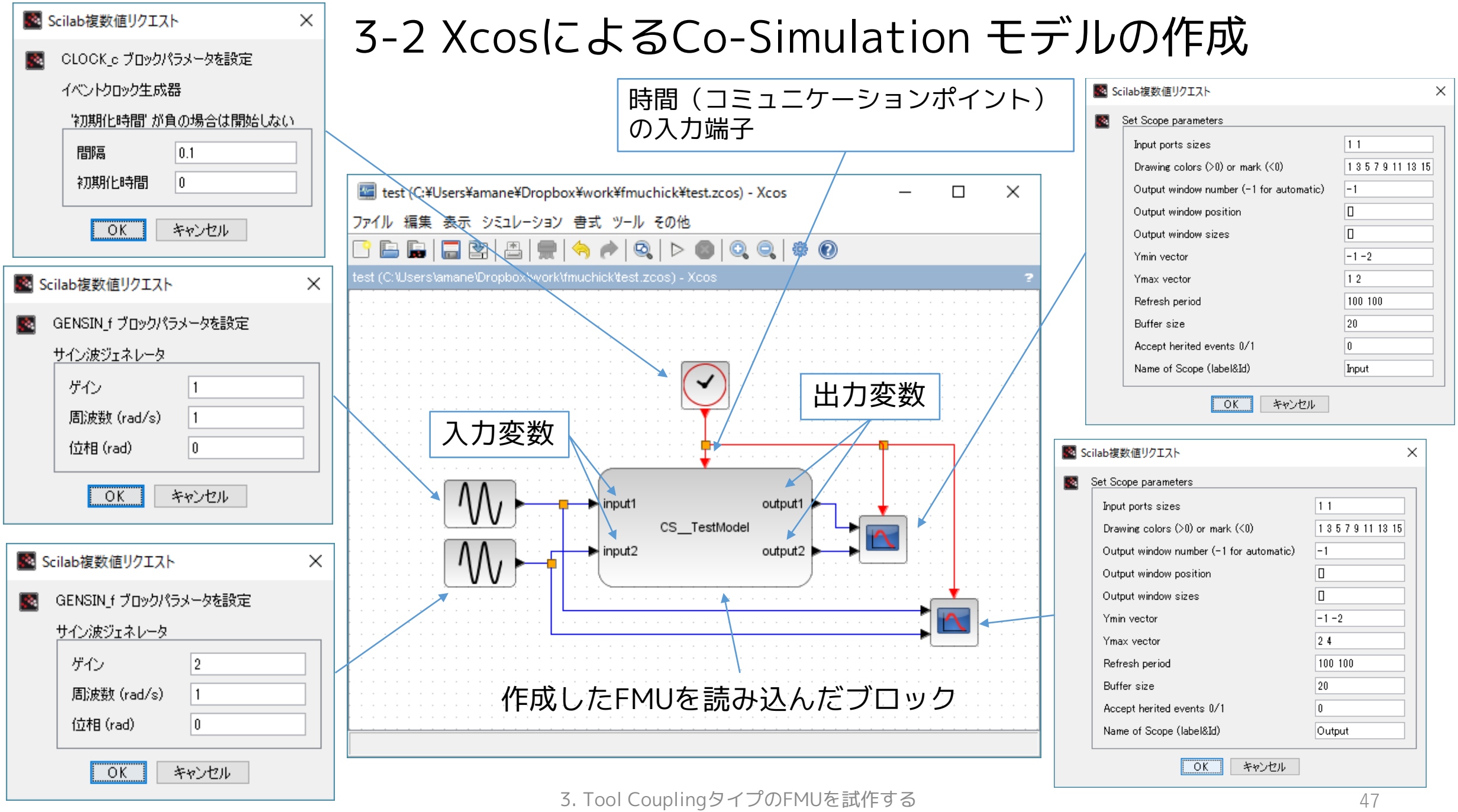
クリックするとGUIDが生成される。

Start Time: 0 s
Stop Time: 10 s
Tolerance: 1.0e-4

FMUサーバーを実行するマシンの
- IPアドレス
- 使用するポート番号

ボタンを押すとFMUができる。

3-2 XcosによるCo-Simulation モデルの作成



3-3 FMUサーバー (server/fmuserver.py) のスレーブツール実行部分の編集

```
def toolInitialize(self):
    for i in range(0,self.nitem):
        print self.itemName[i]+": "+self.itemValue[i]
    for i in range(0,self.ninput):
        print self.inputName[i]+" = "+str(self.itemValue[i])
    for i in range(0,self.noutput):
        print self.outputName[i]+" = "+str(self.outputValue[i])

def toolDoStep(self):
    print "t = "+str(self.ctime) + " dt = "+str(self.cstep)
    for i in range(0,self.ninput):
        print self.inputName[i]+" = "+str(self.inputValue[i])
    for i in range(0, self.noutput):
        if self.ctime <= 1e-6:
            self.outputValue[i] = self.inputValue[i]
        else:
            self.outputValue[i] +=  self.inputValue[i] * self.cstep
        print self.outputName[i]+" = "+str(self.outputValue[i])

def toolTerminate(self):
    print "Terminate tool!"
```

スレーブツールを起動して初期化する。
itemValue[], inputValue[], outputValue[]
が使用できる。

入力変数 inputValue[] を使って、
時刻 t から t+dt のシミュレーションを行い、
結果を出力変数 outputValue[] にセットする。

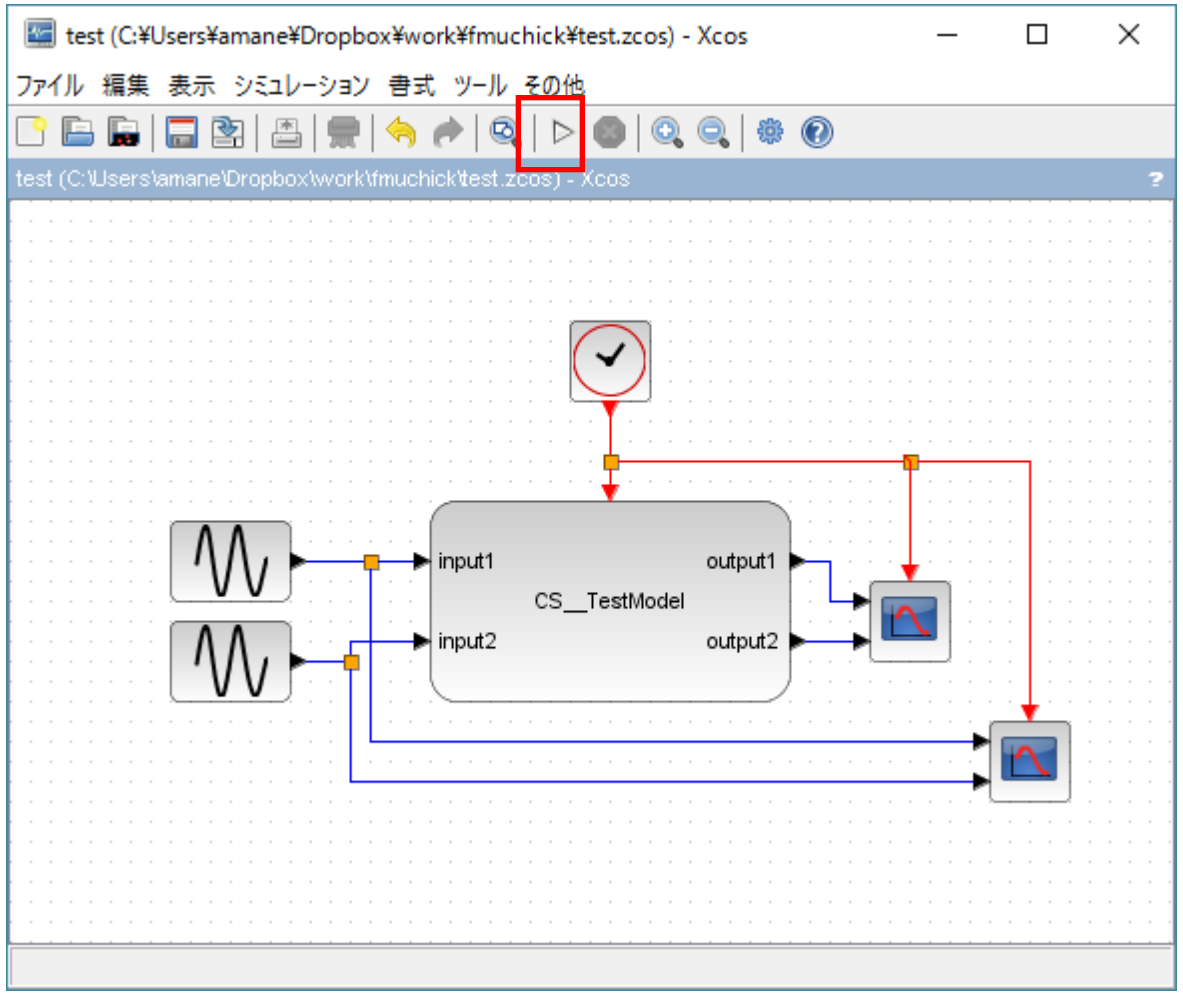
スレーブツールとして外部プログラムを実行する代わりに入力信号をEuler法で数値積分するコードを記述した。

シミュレーションを中止する。
abort, kill, …

上記をカスタマイズすることで様々なスレーブツールに対応することが可能。

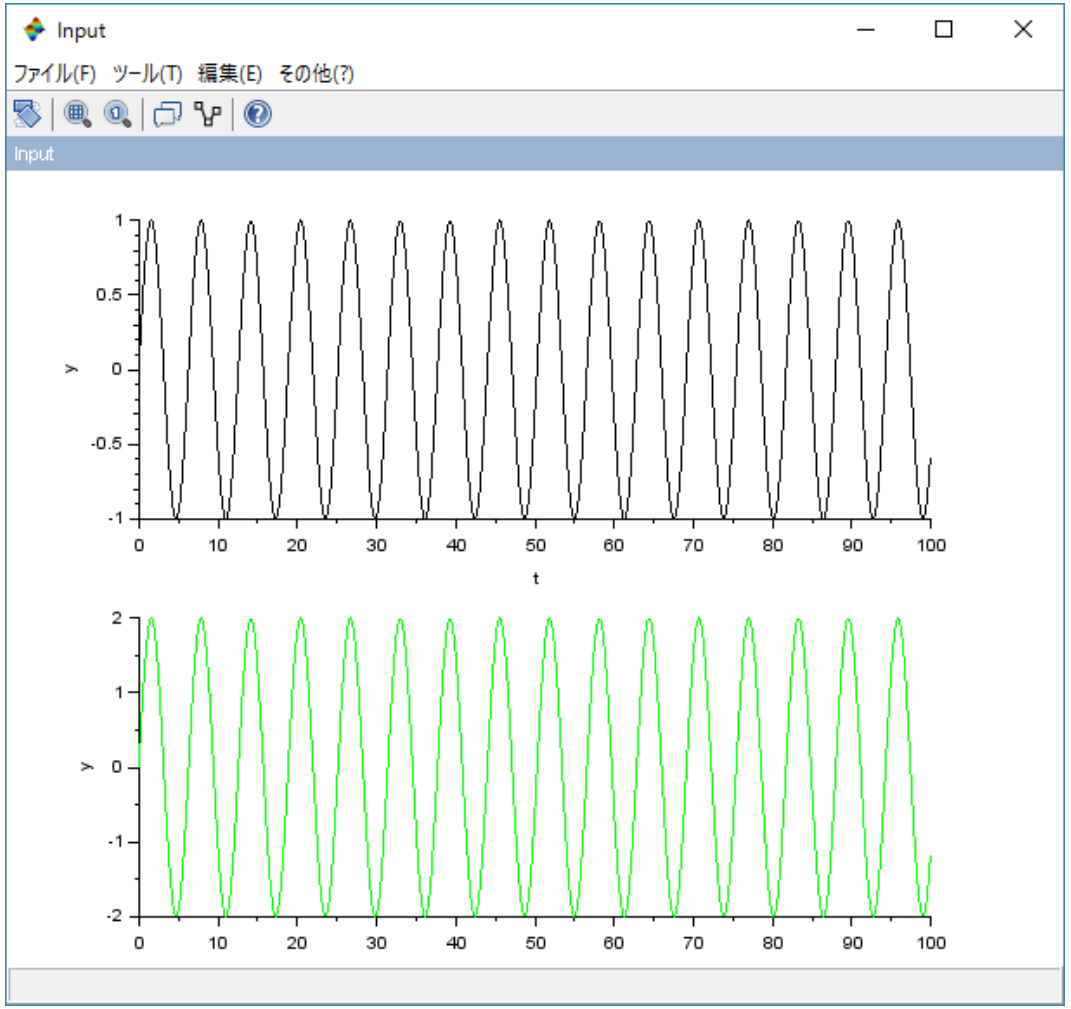
3-4 Co-Simulationの実行

① ディレクトリ server で
コマンドラインから、
python fmuserver.py
を実行する
② Xcos を実行する。

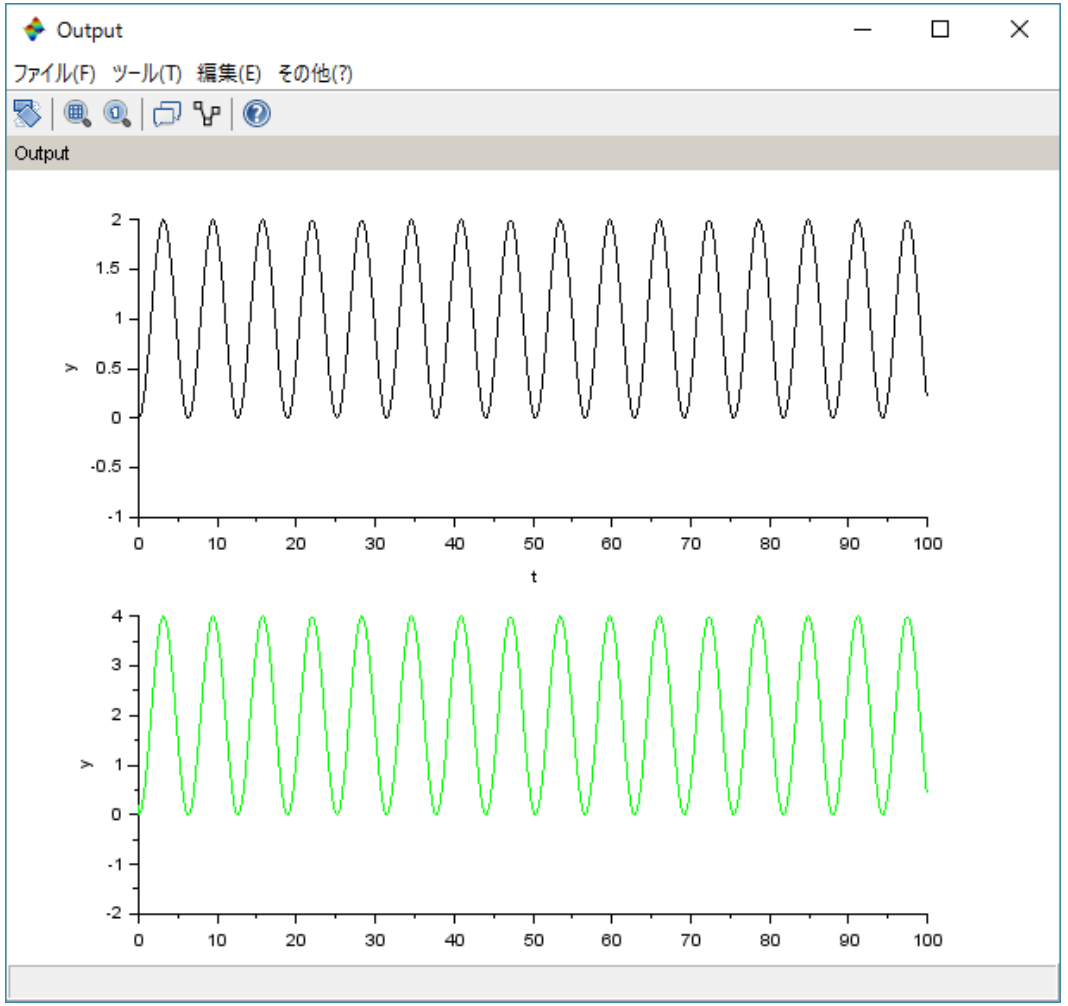


シミュレーションが終了したら、
Ctrl-C でサーバーを止める。

Co-Simulationのテスト結果



入力信号



出力信号（ダミーソルバーによる入力信号の積分）

まとめ

- Tool Coupling タイプのFMUを試作した。
- XcosでCo-Simulationテストモデルを作成した。
- スレーブツールとして入力信号をEuler法で積分するコードをFMUサーバに実装した。
- Co-Simulationテストモデルが正常に実行できることが確認できた。