

# 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		Notes
Tools supporting FMI	FMI Version	Export	Import	Slave	Master		
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 Dynamics 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	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の交換がなされる。

Instances

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)なシミュレーションを行う。

Classes

## 連成システムモデル(Coupled System Model)

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。

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

## コミュニケーションステップ

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

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。

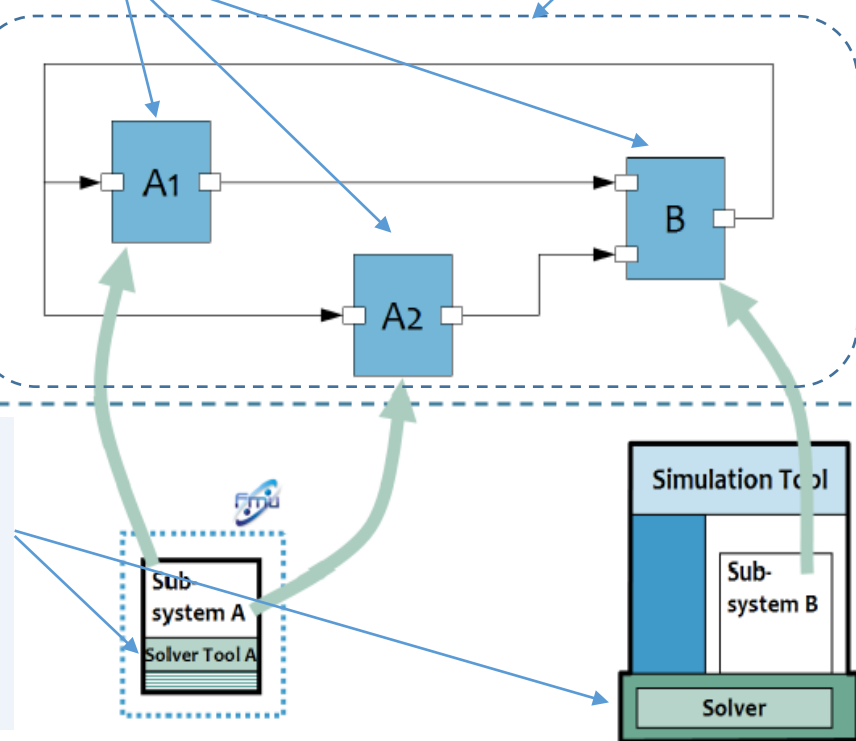


Figure 6: FMU Classifier/Instance Differences (仕様書 P.11より)

独立なタイムステップ  
 $h_i = t_{i+1} - t_i$

# 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  $u(tC_i)$  を受信し、出力変数  $y(tC_i)$  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  $tC_{i+1}$  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  $u(tC_i)$  を補間できる。(  $\dot{u}(tC_i), \ddot{u}(tC_i), \dots$  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

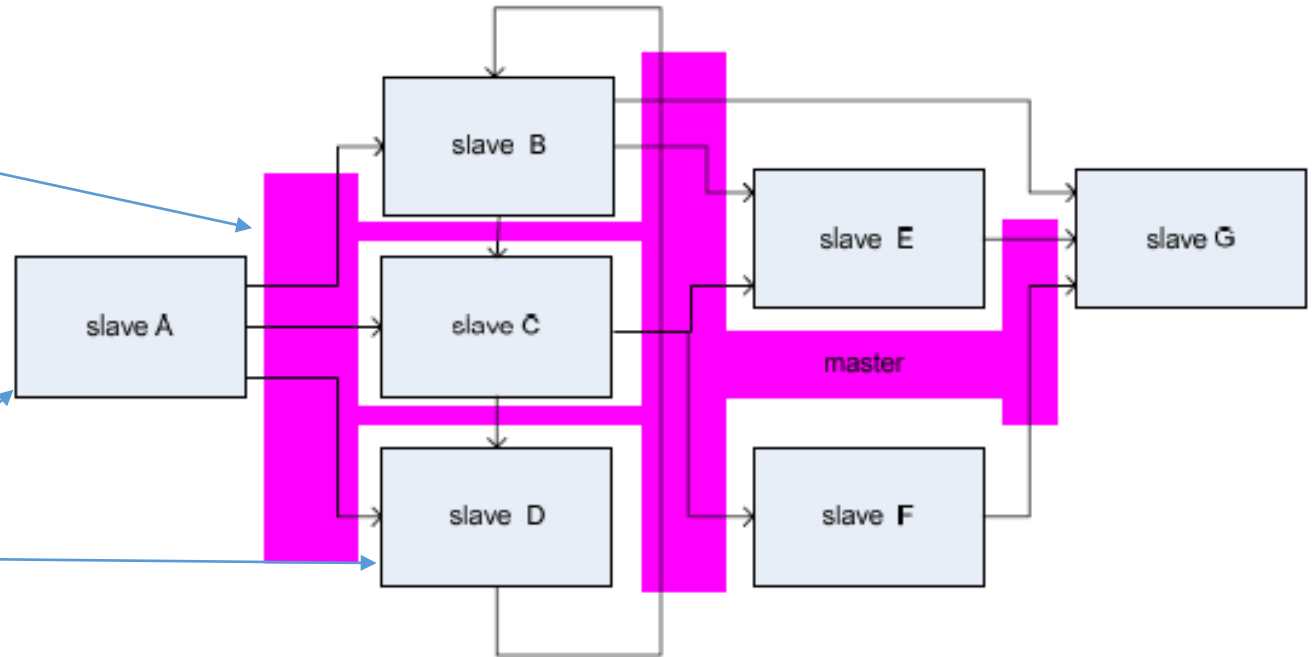


Figure 14: Master-Slave structure (仕様書 p.17)

## 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

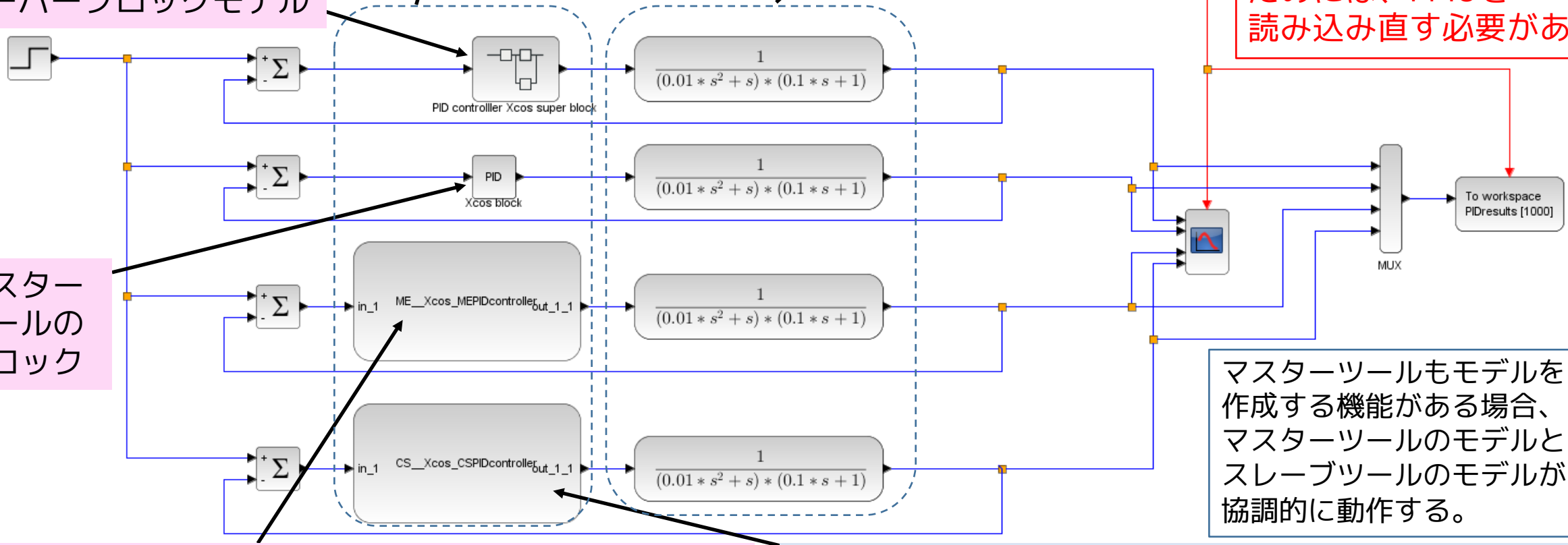
<https://forge.scilab.org/index.php/p/fmu-wrapper/>

マスターツール(Xcos)で  
 つくった  
 スーパーブロックモデル

いろいろな方法で作った 制御対象のモデル  
 / PID制御器のモデル

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  $h_{C_i} = t_{C_{i+1}} - t_{C_i}$  を計算する。
- サブシステムの入力値  $u(t_{C_i}), (\dot{u}(t_{C_i}), \ddot{u}(t_{C_i}), \dots)$  を計算する。
- スレーブツールに  $t_{C_i}, t_{C_{i+1}}, u(t_{C_i}), (\dot{u}(t_{C_i}), \ddot{u}(t_{C_i}), \dots)$  を送信する。
- スレーブツールに  $[t_{C_i}, t_{C_{i+1}}]$  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  $y(t_{C_{i+1}}), (\dot{y}(t_{C_{i+1}}), \ddot{y}(t_{C_{i+1}}), \dots)$  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

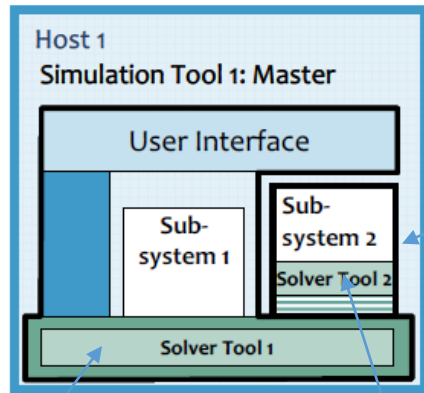


Figure 9: Co-simulation with generated code on a single computer

(仕様書 P.13)

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。

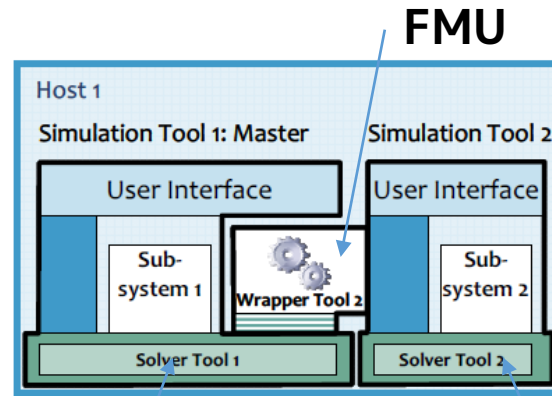


Figure 10: Co-simulation with tool coupling on a single computer (仕様書 P.14)

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。

マスターツールのソルバー

スレーブツールのソルバー

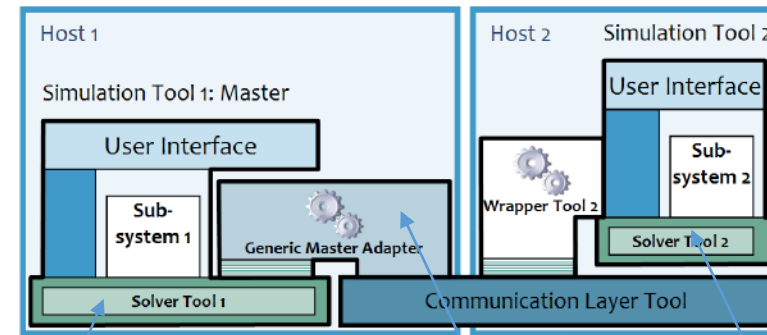


Figure 11: Distributed Co-simulation Infrastructure

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。

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)

```
C:\%USERS%\%USER%\%WORK%\FMUSDK\FMU10\FMU\CS\BOUNCINGBALL
├── model.png
├── modelDescription.xml
├── binaries
│   └── win32
│       └── bouncingBall.dll
├── documentation
│   ├── plot_h.PNG
│   └── _main.html
└── sources
    ├── bouncingBall.c
    ├── fmuTemplate.c
    └── fmuTemplate.h
```

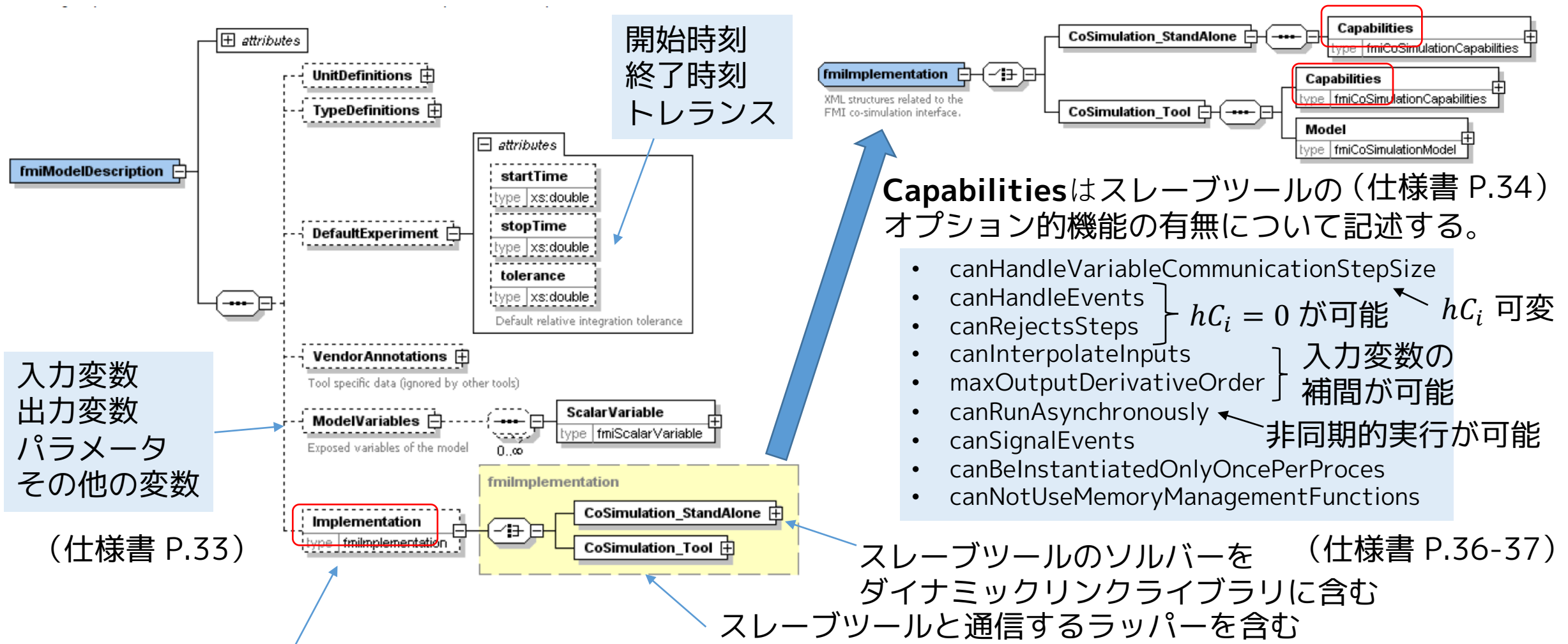
## FMUSDKで作成した bouncingBall.fmu の内容

modelDescription.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



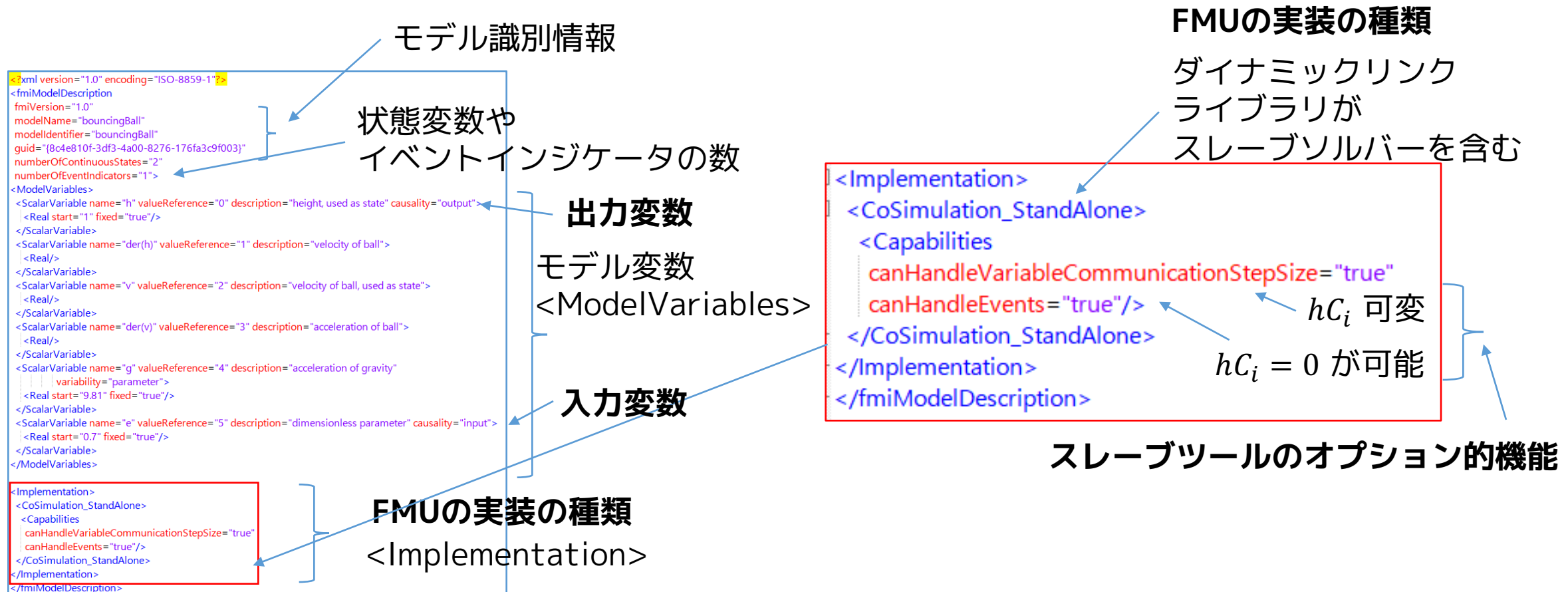
FMI for Model Exchange と比べて  
FMI for Co-Simulation ではFMUの実装の種類に関する情報が追加されている。

# 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)



# 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("standard32")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)`

### 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)`

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

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。。

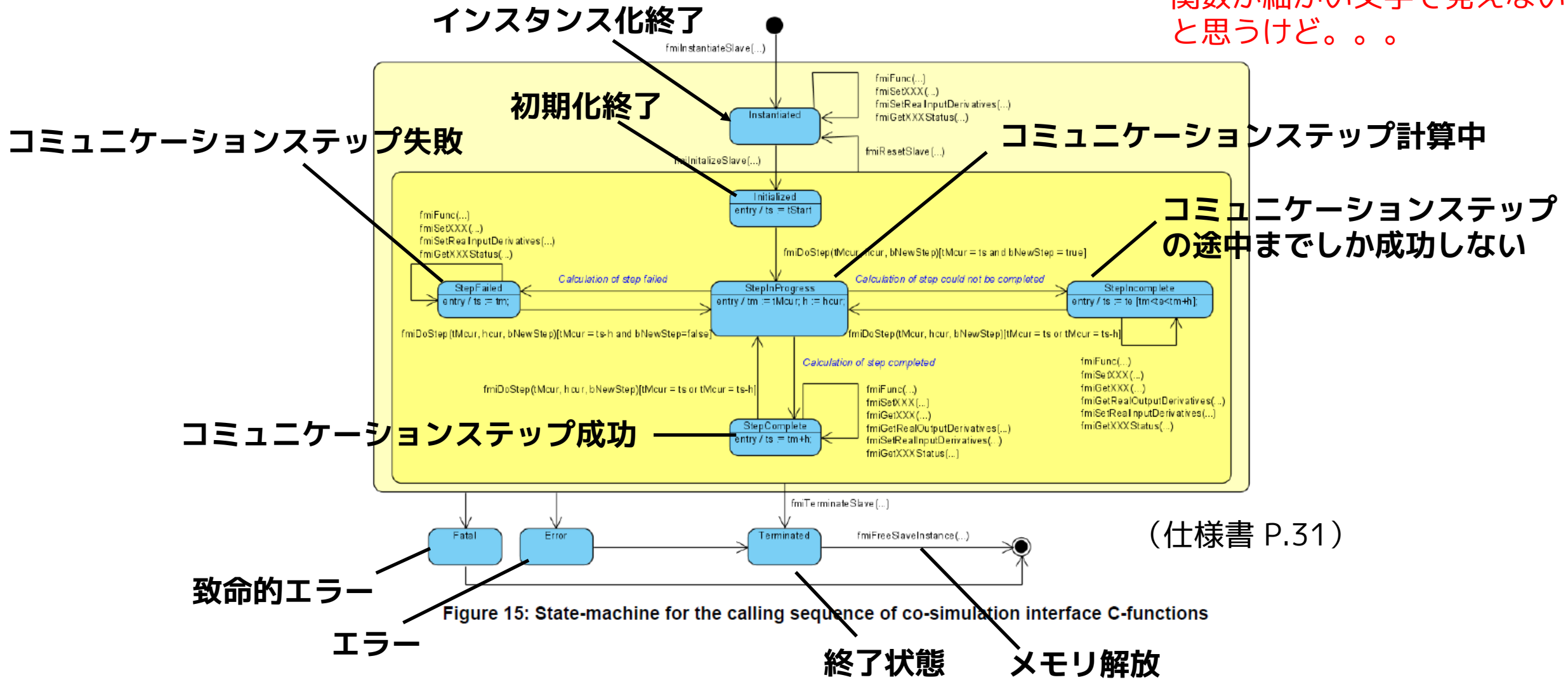


Figure 15: State-machine for the calling sequence of co-simulation interface C-functions

## 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)

Tools supporting FMI	FMI Version	ModelExchange		CoSimulation	
		Export	Import	Slave	Master
EnergyPlus	FMI_1.0			Available	Available
FMUSDK	FMI_2.0	Available 16		Available 16	
	FMI_1.0	Available	Available	Available	Available
JModelica.org	FMI_2.0	Available 8		Available 8	Available 26
	FMI_1.0	Available 20	Available 45	Available 14	Available 49
OpenModelica	FMI_2.0	Available	Available	Planned	Planned
	FMI_1.0	Available 3	Available	Planned	Available
PyFMI	FMI_2.0		Available 23		Available 32
	FMI_1.0		Available 63		Available 59
Scilab/Xcos FMU wrapper	FMI_2.0	Planned	Planned	Planned	Planned
	FMI_1.0	Available	Available	Available	Available

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

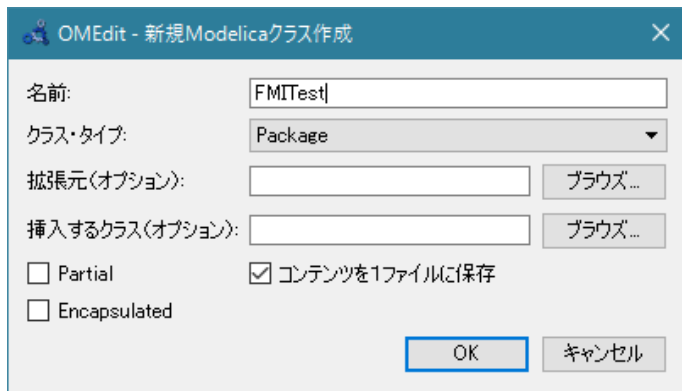
<http://book.xogeny.com/behavior/discrete/bouncing/>

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

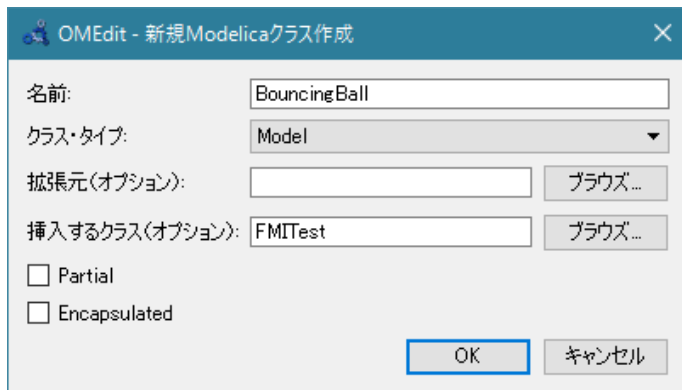
OpenModelica 1.9.6 Windows版を使用する。  
( <https://openmodelica.org/download/download-windows> )

## ①ファイル>Modelicaクラス新規作成



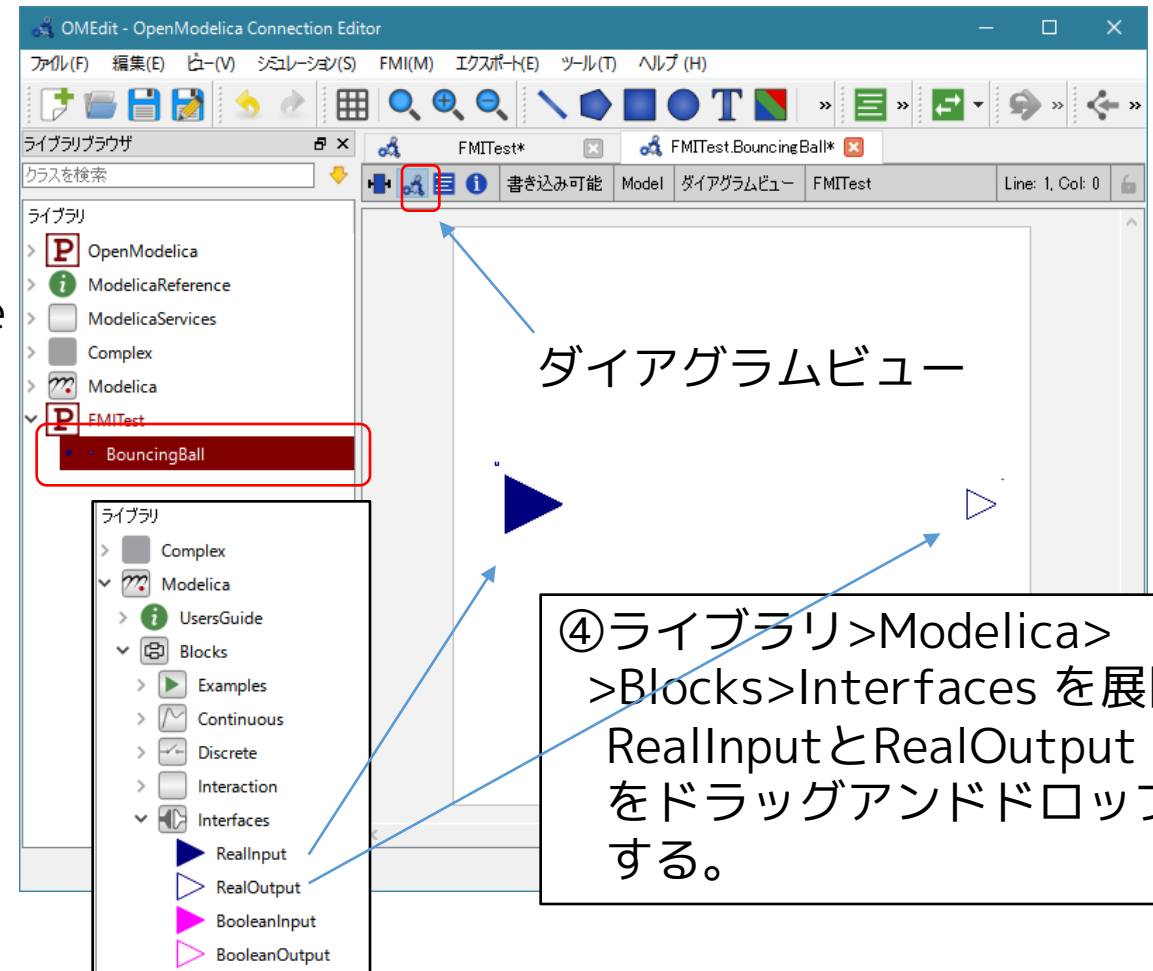
名前: FMITest  
クラス・タイプ: Package

## 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。



ダイアグラムビュー

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。

```

2 model BouncingBall
3   Modelica.Blocks.Interfaces.RealInput e(start=0.8) annotat
4   Modelica.Blocks.Interfaces.RealOutput y annotation(Placem
5   import SI = Modelica.SIunits;
6   SI.Height h;
7   SI.Velocity v;
8   parameter SI.Acceleration g = Modelica.Constants.g_n;
9   parameter SI.Height h0 = 1.0;
10  initial equation
11   h = h0;
12  equation
13   v = der(h);
14   der(v) = -g;
15   y = h;
16   when h < 0 then
17     reinit(v, -e * pre(v));
18   end when;
19   annotation(uses(Modelica(version = "3.2.1")));
20 end BouncingBall;
  
```

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

$$v = \frac{dh}{dt}$$

$$\frac{dv}{dt} = -g$$

$$y = h$$

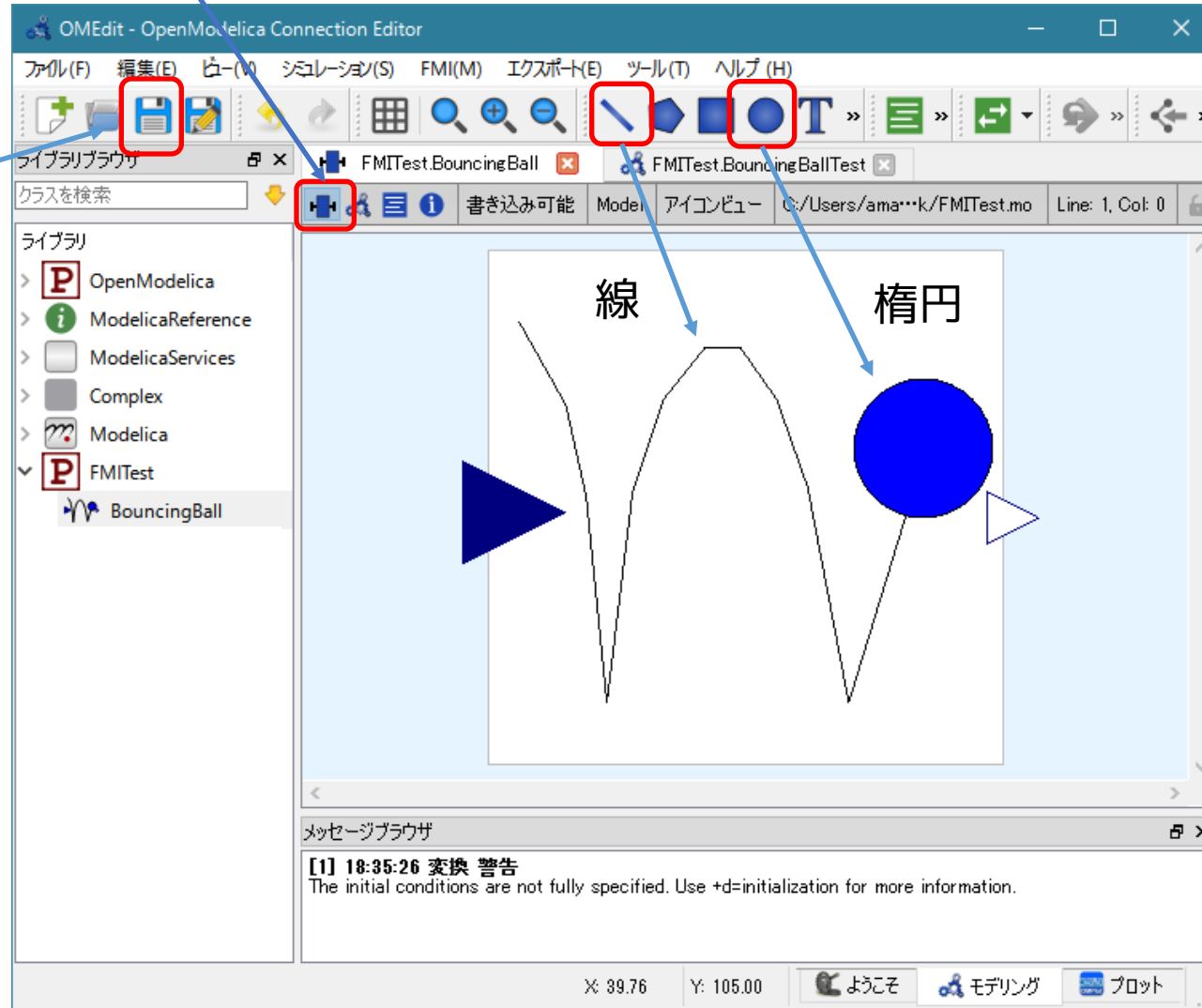
$h < 0$  なら  $v = -e * v$   
に初期化しなおす。

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  
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. 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

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

The image shows two windows from the OpenModelica software. The left window, titled "シミュレーションのセットアップ - FMITest.BouncingBall", contains simulation parameters. The "開始時刻" (Start time) is set to 0 and "終了時刻" (End time) is set to 3, both highlighted with red boxes. A text box next to them says "開始時刻: 0 終了時刻: 3". The "シミュレート" (Simulate) button is also highlighted with a red box. The right window, titled "OMEdit - OpenModelica Connection Editor", shows the simulation in progress. The "シミュレーション(S)" menu item is highlighted with a red box. The "ライブラリ" (Library) pane shows "BouncingBall" selected. The "変数ブラウザ" (Variable Browser) pane shows a list of variables, with "h" and "y" checked and highlighted with a red box. A text box next to it says "③ h または y をチェックしてプロットする。". The main plot area shows a graph of "h [m]" over time, with a red curve oscillating between 0 and 1.0. The status bar at the bottom shows coordinates and system information.

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

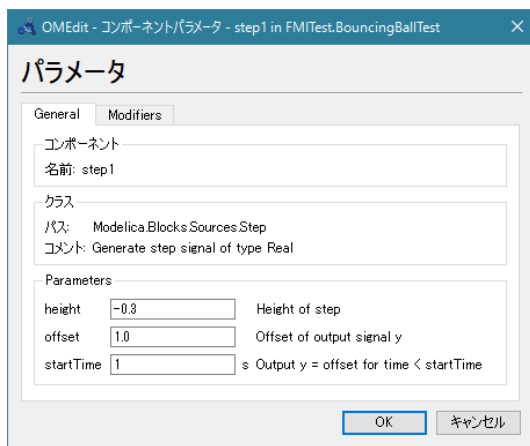
テスト2：入出力がある場合のサブシステムモデルのシミュレーション

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

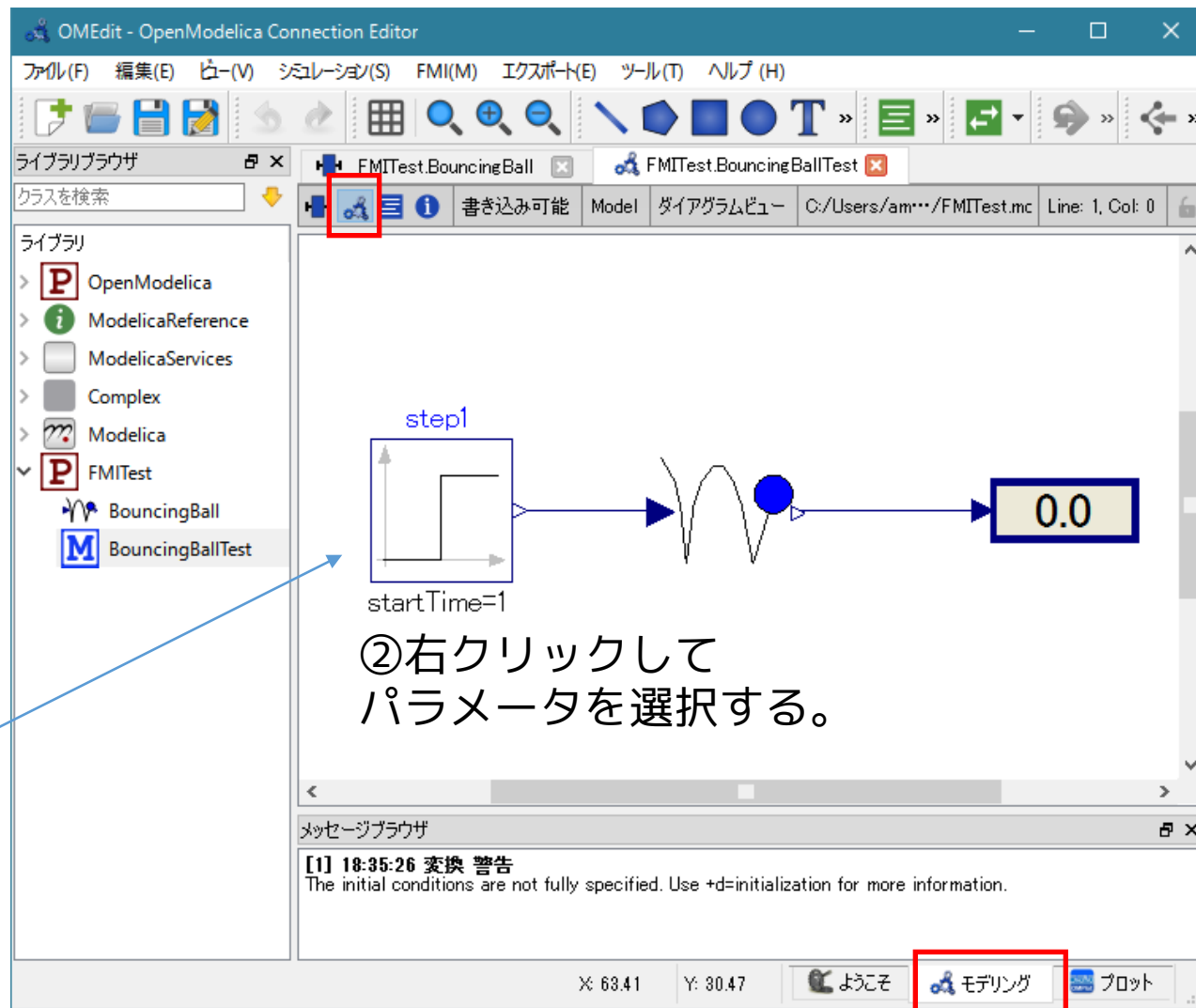
で新しいモデルを作成し、以下のモデルをライブラリブラウザからドラッグして配置し接続する。

- 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右クリックしてパラメータを選択する。

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

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

シミュレーションのセットアップ - FMITest.BouncingBallTest

全般 出力 シミュレーションフラグ アーカイブされたシミュレーション

解析間隔  
開始時刻: 0  
終了時刻: 3.0  
計算回数: 500  
間隔: 0.006

積分  
手法: dassl  
許容値: 0.0001  
DASSLのオプション  
ヤコビアン: 色を付けた数値  
 ルートの検索  
 イベントの後でリスタート  
初期ステップサイズ:  
最大ステップサイズ:  
最大積分次数: 5

コンパイラフラグ(オプション):  
プロセッサ数: 4  
 構築のみ  
 変換デバガを起動する  
 アルゴリズムデバガを起動する

シミュレーション設定をモデル内に保存

シミュレート キャンセル

開始時刻: 0  
終了時刻: 3

⑥最後に保存する。  
FMITest.mo

1秒までは  $e=1$ 、その後  $e=0.7$

ライブラリブラウザ  
クラスを検索

ライブラリ  
OpenModelica  
ModelicaReference  
ModelicaServices  
Complex  
Modelica  
FMITest  
BouncingBall  
BouncingBallTest

Plot: 1  
Zoom Pan Auto Scale Fit in View Save Print Grid Detailed Grid No Grid

変数ブラウザ  
変数の検索

変数	値	単位
FMITest....cingBall		
FMITest....BallTest		
bouncingBall1		
der(h)	0.315474	m
der(v)	-9.80665	m/s
e	0.7	
g	9.80664...9999999	m/s <sup>2</sup>
h		m
h0	1.0	m
v		m/s
y	0.0231732	
realValue1		
step1		
time	3	

メッセージブラウザ  
The initial conditions are not fully specified. Use +d=initialization for more information.  
[1] 01:20:24 変換 警告  
The initial conditions are not fully specified. Use +d=initialization for more information.

⑤入力変数 e  
出力変数 y  
をチェックする

④シミュレーションの実行

2. FMUを作って使ってみる

## 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/> )

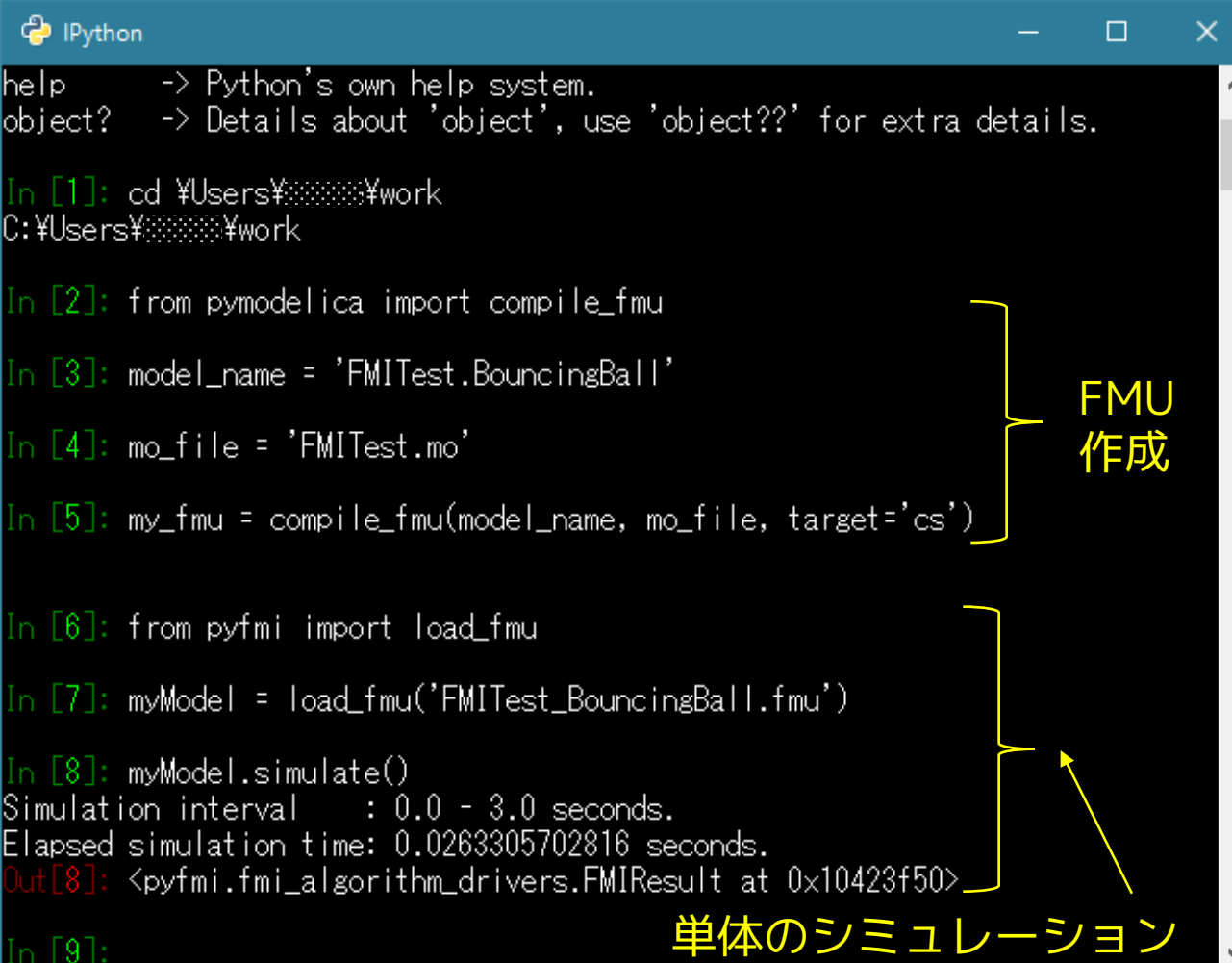
- ① スタートメニューの JModelica.org-1.17の中の IPython か Pythonのコンソールを起動する。
- ② OpnemModelicaで保存したファイルのあるディレクトリをカレントディレクトリにして以下を入力する。

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

**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()
```



```
IPython
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: cd %Users%[redacted]work
C:%Users%[redacted]work

In [2]: from pymodelica import compile_fmu

In [3]: model_name = 'FMITest.BouncingBall'

In [4]: mo_file = 'FMITest.mo'

In [5]: my_fmu = compile_fmu(model_name, mo_file, target='cs')

In [6]: from pyfmi import load_fmu

In [7]: myModel = load_fmu('FMITest_BouncingBall.fmu')

In [8]: myModel.simulate()
Simulation interval      : 0.0 - 3.0 seconds.
Elapsed simulation time: 0.0263305702816 seconds.
Out[8]: <pyfmi.fmi_algorithm_drivers.FMIResult at 0x10423f50>

In [9]:
```

FMU  
作成

単体のシミュレーション

JModelica環境の IPython コンソール

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

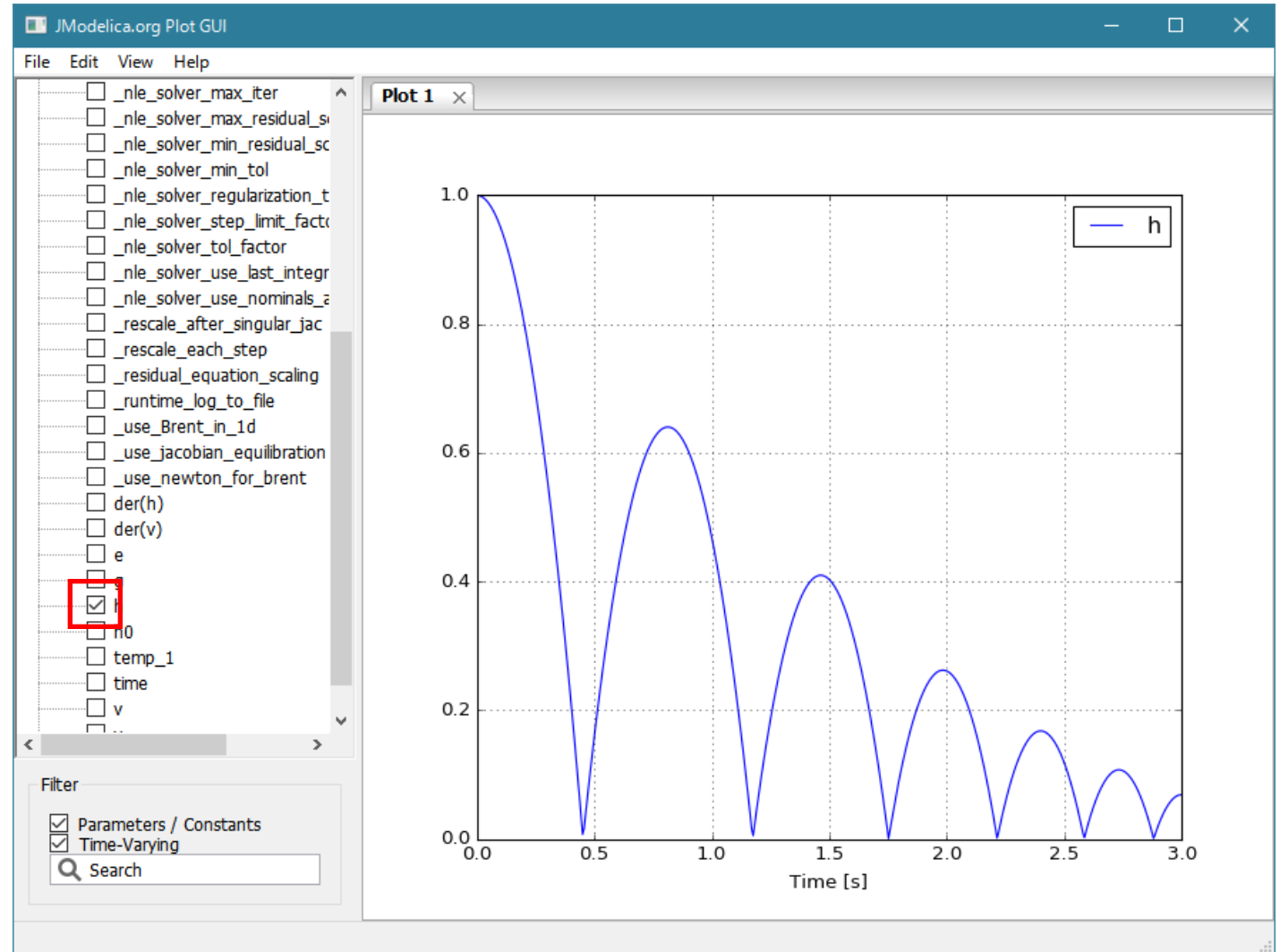
**テスト1**：入出力がない単体のシミュレーション

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\_BouncingBall\_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

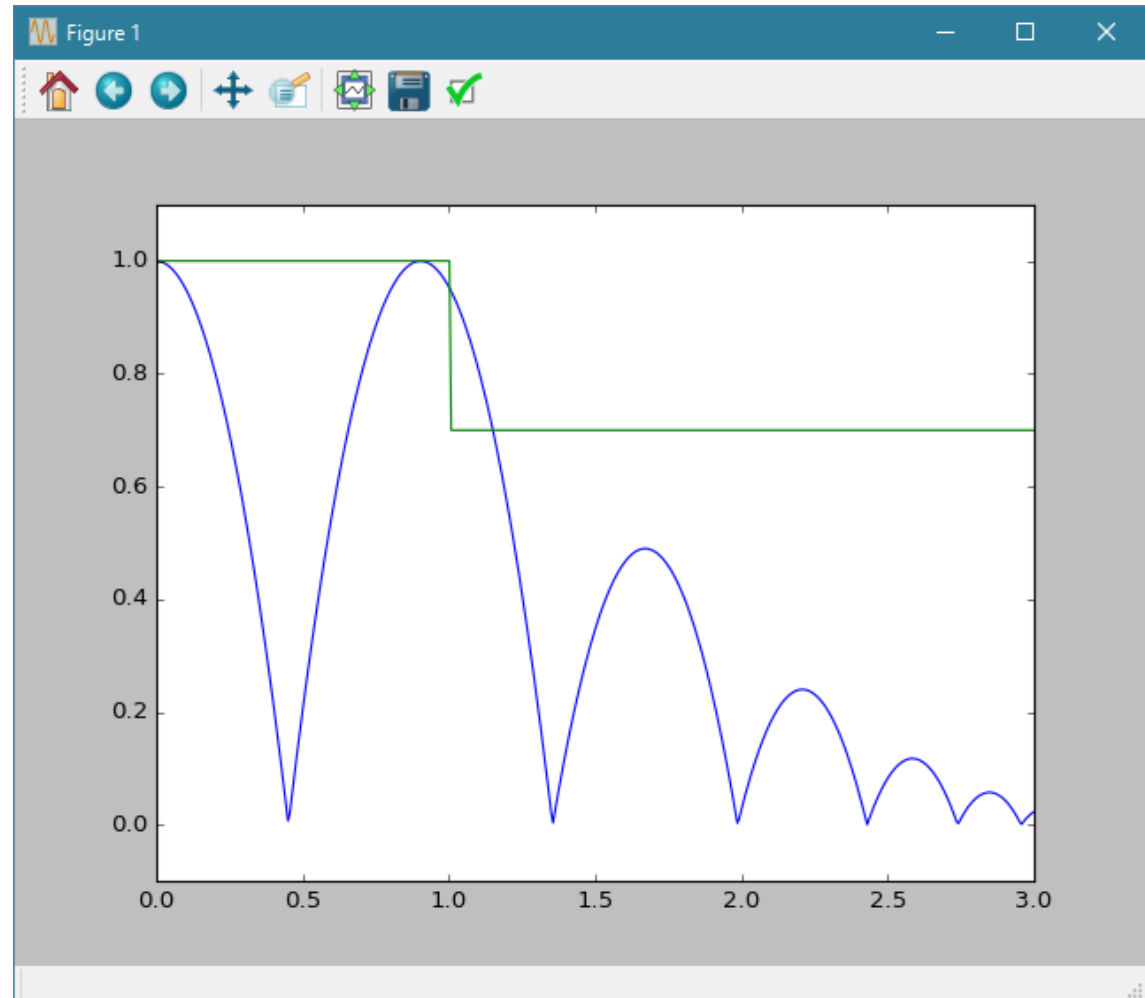
#### **bouncingBallTest.py**

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
```

入力変数の変化を  
表す関数

```
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()
```

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

```
y = model.get("y")  
t_res.append(t)  
e_res.append(e)  
y_res.append(y[0])  
while t < t_end :
```

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

```
    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])
```

出力変数取得

```
        t += dt  
        t_res.append(t)  
        e = stepf(t)  
        model.set("e",e)
```

次のコミュニケーション  
ステップの計算

```
    else:  
        print "error"  
        break
```

入力変数の計算と  
設定

```
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()
```

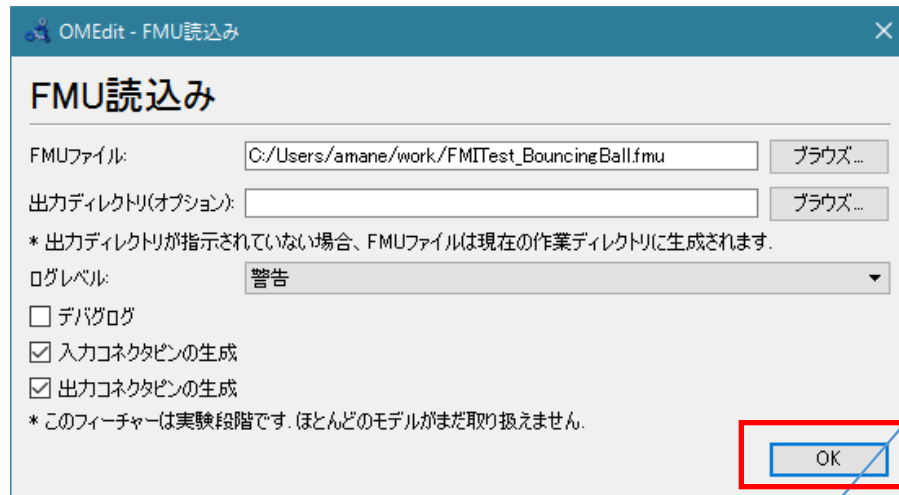
図化处理

# 2-4 OpenModelicaでFMUを読み込む

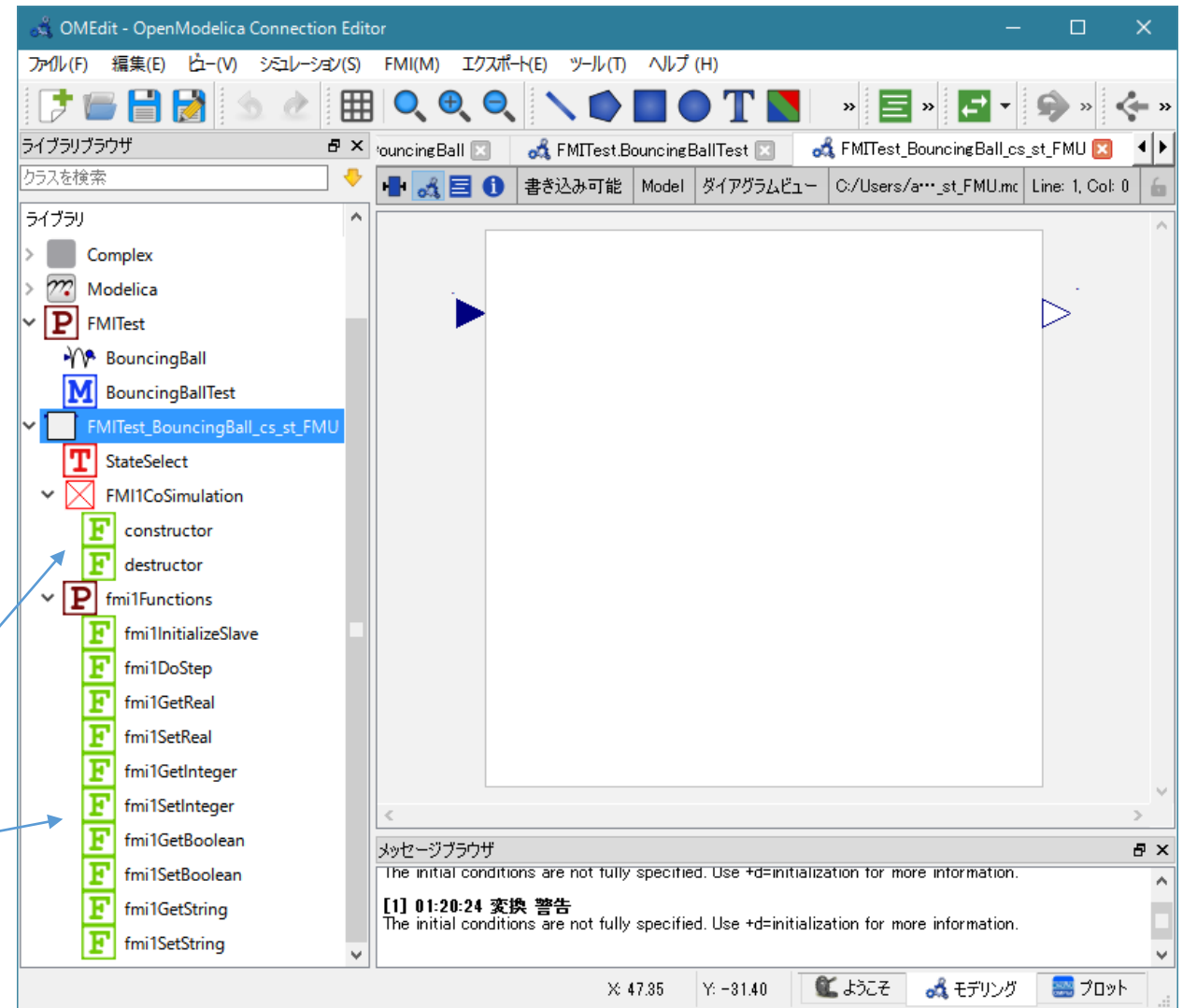
JModelica で作成したFMUを  
OpenModelica で読んでみる。

## ①FMI(M)>FMU読み込み

FMIファイル : FMITest\_BouncingBall.fmu



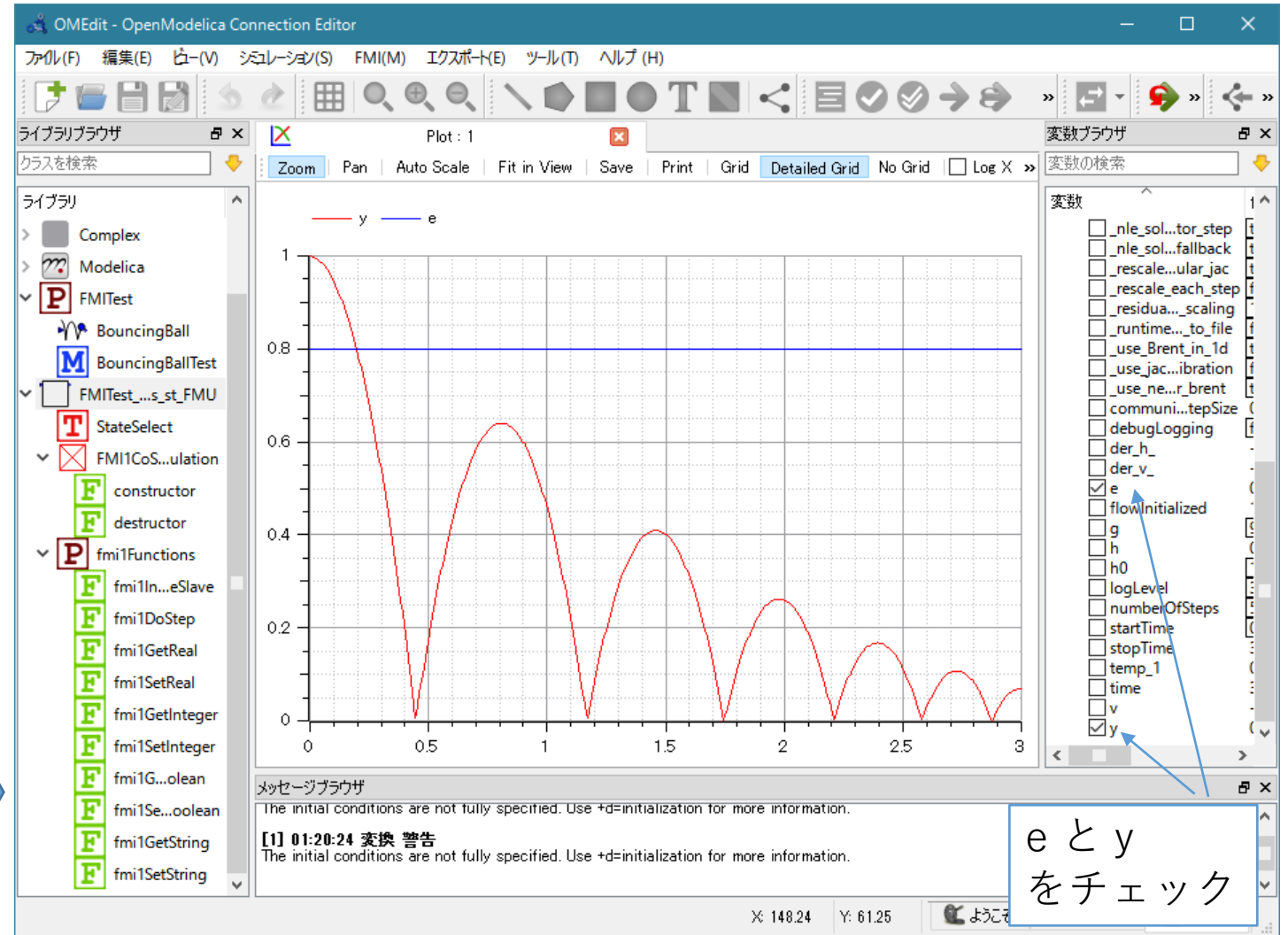
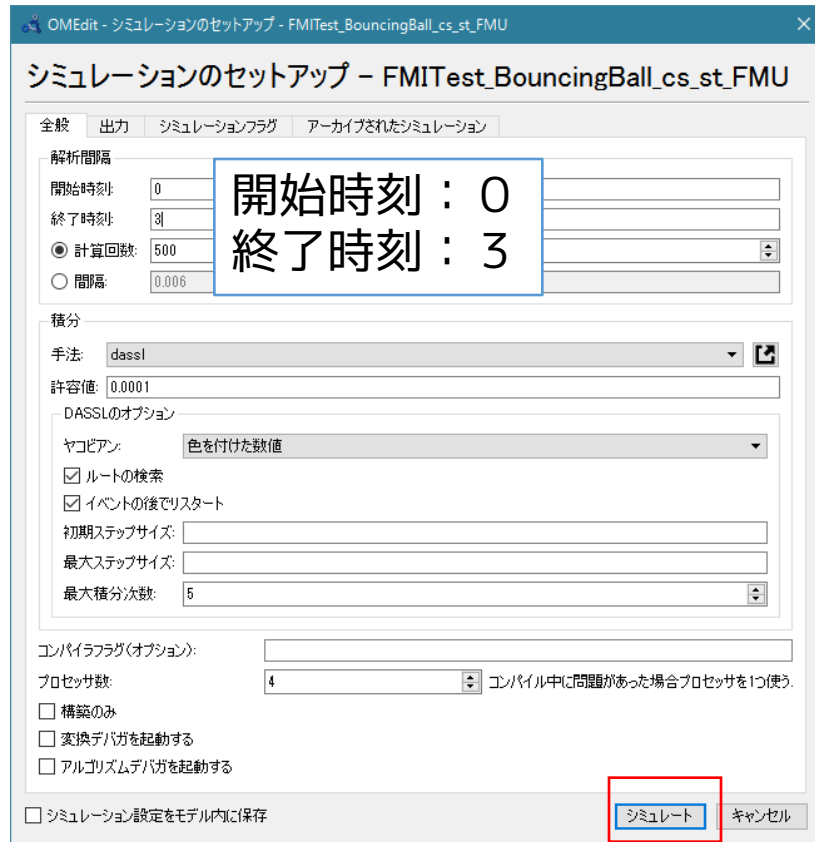
ツリーを展開すると  
FMIのインターフェース関数に相当する  
関数が表示される。



# 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. FMUを作って使ってみる



# 2.4 OpenModelicaでFMUを読み込む

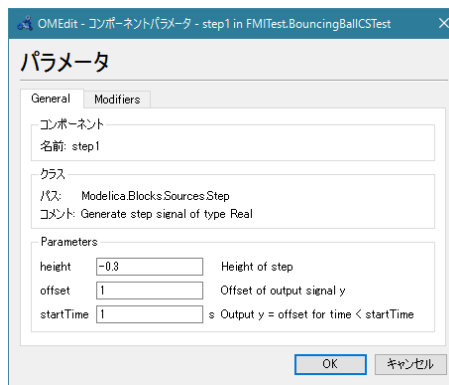
## テスト2：入出力のある場合のシミュレーション

### ③新しいモデル

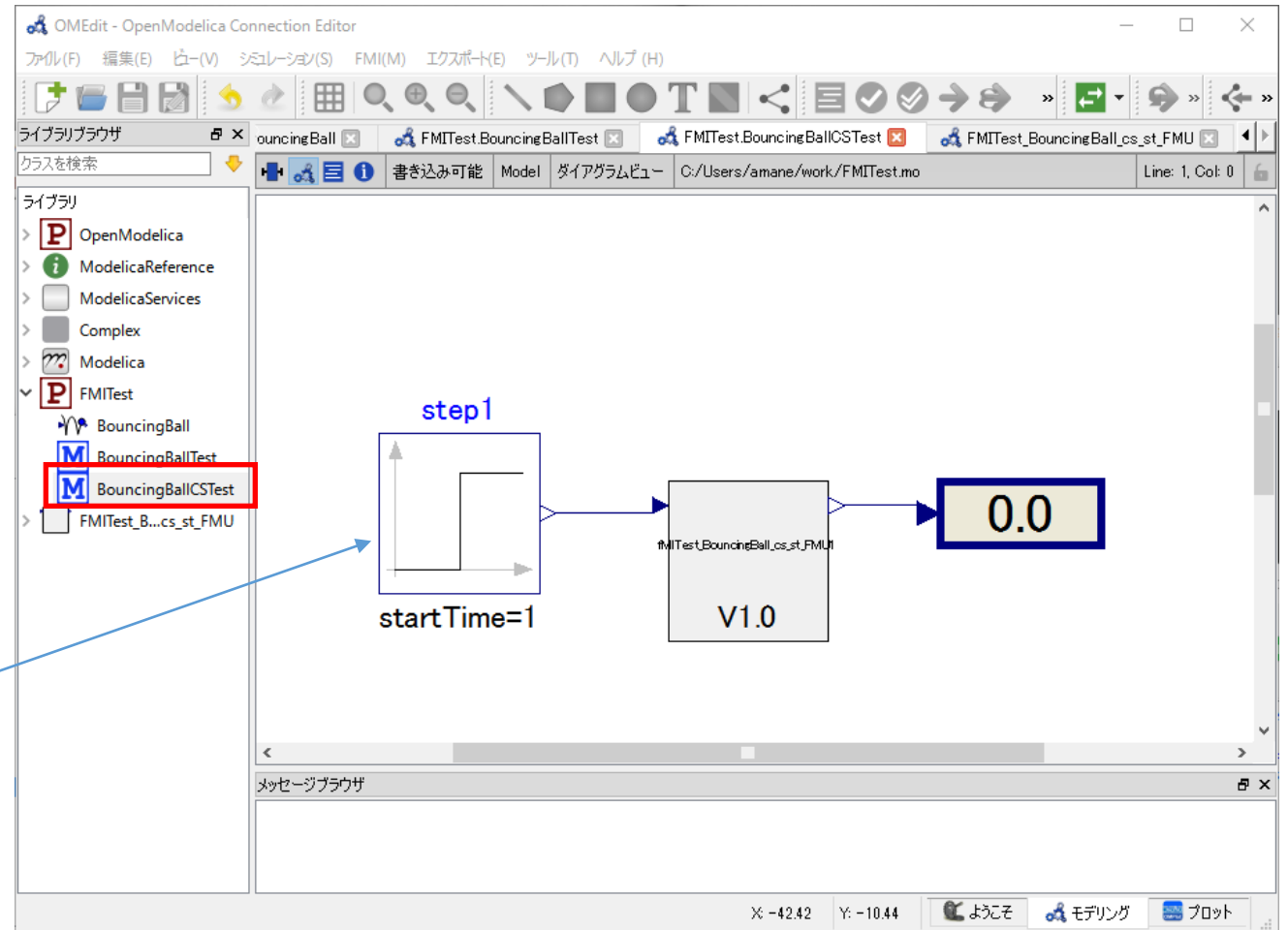
#### BouncingBallCSTest

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

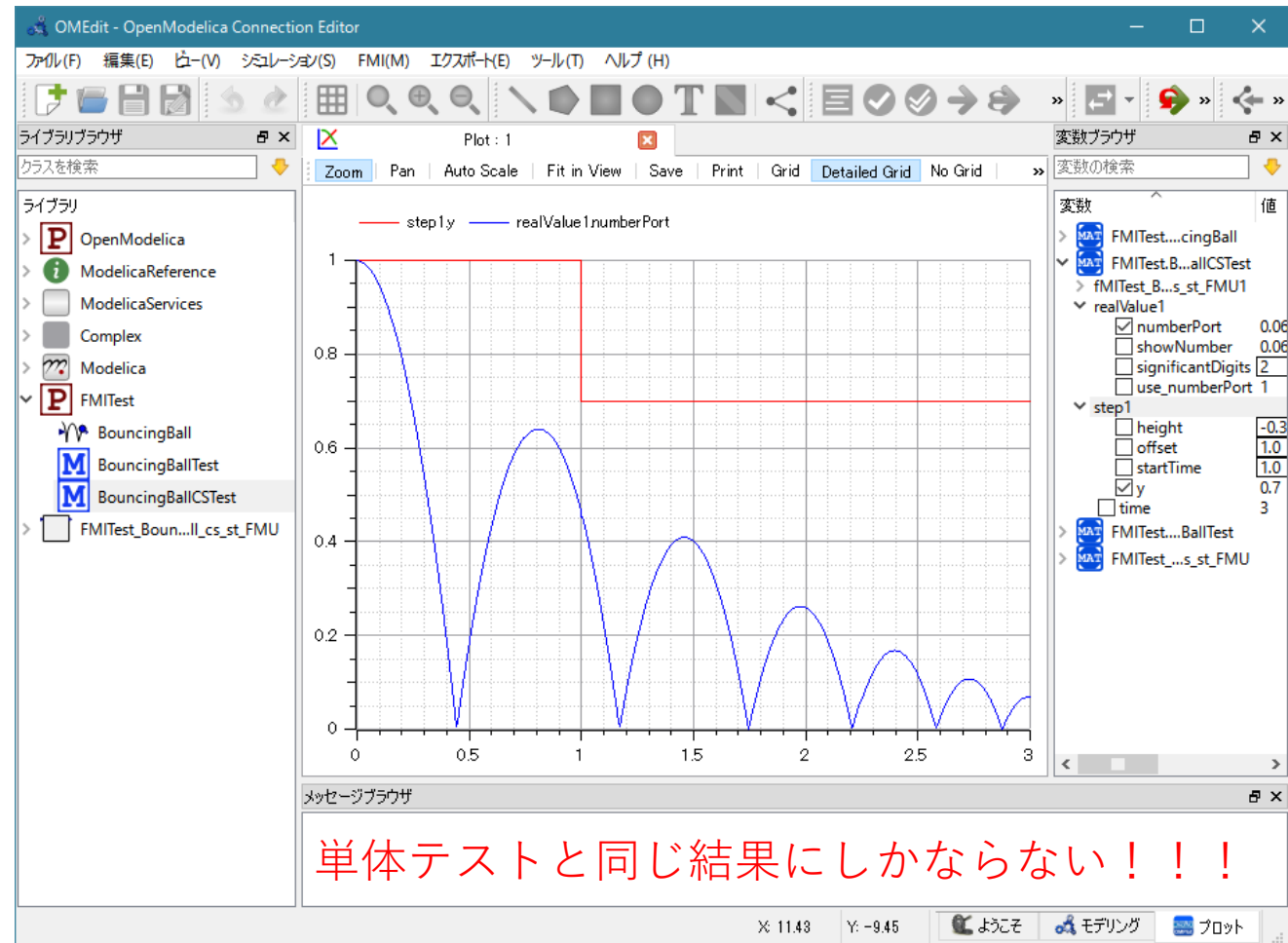
でシミュレーションを実行する。

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

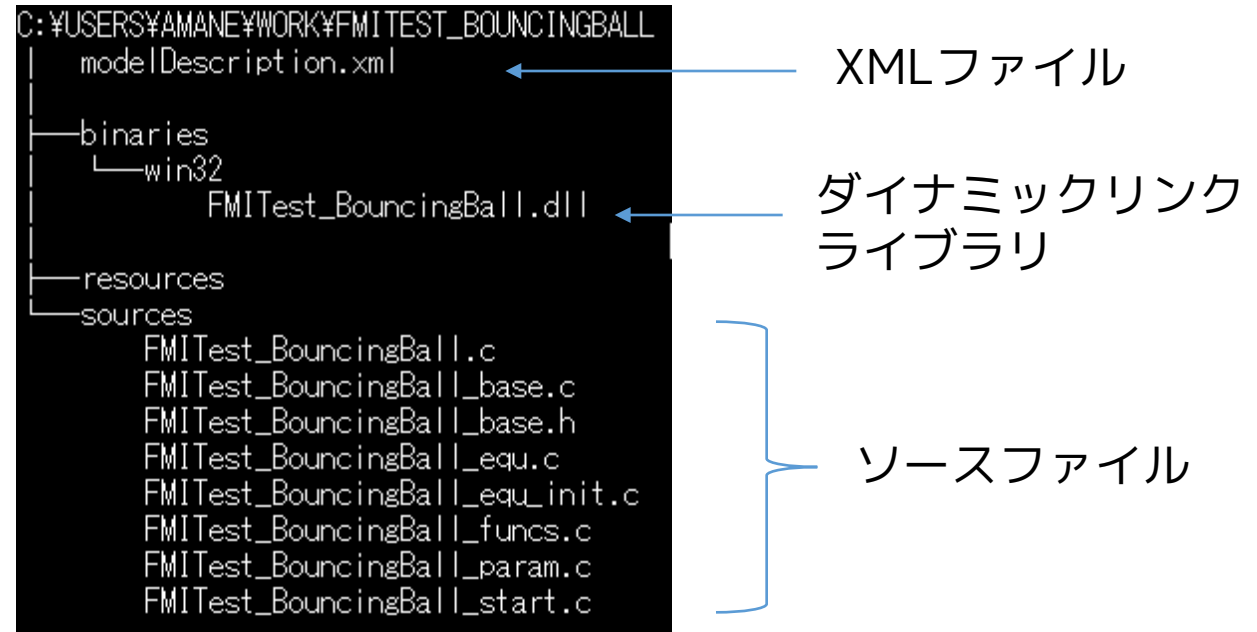
JModelica.org で作成した  
**FMITest\_BouncingBall.fmu**  
をコピーして次のようにリネームする。  
**FMITest\_BouncingBall.zip**  
解凍して中身を確認する。

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。

```
28  
29 Description:  
30 import FMUs: Allow  
31 | In Xcos it sh  
32 | It is possibl  
33 | In Scilab it  
34 export FMUs: Xcos  
35 | Actually, on  
36 | Code gener
```

```
28  
29 Description:  
30 | import FMUs: Allow t  
31 | In Xcos it shou  
32 | It is possible to  
33 | In Scilab it sho  
34 | export FMUs: Xcos m  
35 | Actually, only g  
36 | Code generati
```

スペース

- ③ etc¥fmu\_wrapper.start の 77 行目の pwd() を root\_tlbx に変更する

```
75 | ... | minmodetypes.m ],  
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

エラー

```
-->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)
```

**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

パレットブラウザ - Xcos

パレット

- 汎用ブロック
- 連続時間システム
- 不連続
- 離散時間システム
- ルックアップ・テーブル
- イベント・ハンドリング
- 数値計算
- 行列
- 電気工学
- 整数
- ポートとサブシステム
- ゼロクロス検出
- 信号の配線
- 信号処理
- 暗示的
- 注釈
- 出力/表示
- 信号源
- 熱水力学
- ブロック・デモ
- ユーザー定義関数
- Functional Mock-up Interfaces
  - FMI.1.0**

FMInterface

信号源 > const

ドラッグ・アンド・ドロップ

CS\_FMItest\_BouncingBall

MUX

信号の配線 > MUX

右クリックしてブロックパラメータを選択し **FMItest\_BouncingBall.fmu** を設定する。

Scilab複数値リクエスト

CLOCK\_c ブロックパラメータを設定

イベントクロック生成器

'初期化時間' が負の場合は開始しない

間隔	0.01
初期化時間	0

OK キャンセル

出力/表示 > CSCOPE

Scilab複数値リクエスト

Set Scope parameters

Color (>0) or mark (<0) vector (8 entries)	1 3 5 7 9 11 13 15
Output window number (-1 for automatic)	-1
Output window position	0
Output window sizes	[600;400]
Ymin	-0.1
Ymax	1.1
Refresh period	3
Buffer size	20
Accept herited events 0/1	0
Name of Scope (label&Id)	

OK キャンセル

# 2-5 Scilab/Xcos FMU wrapperでFMUを読み込む

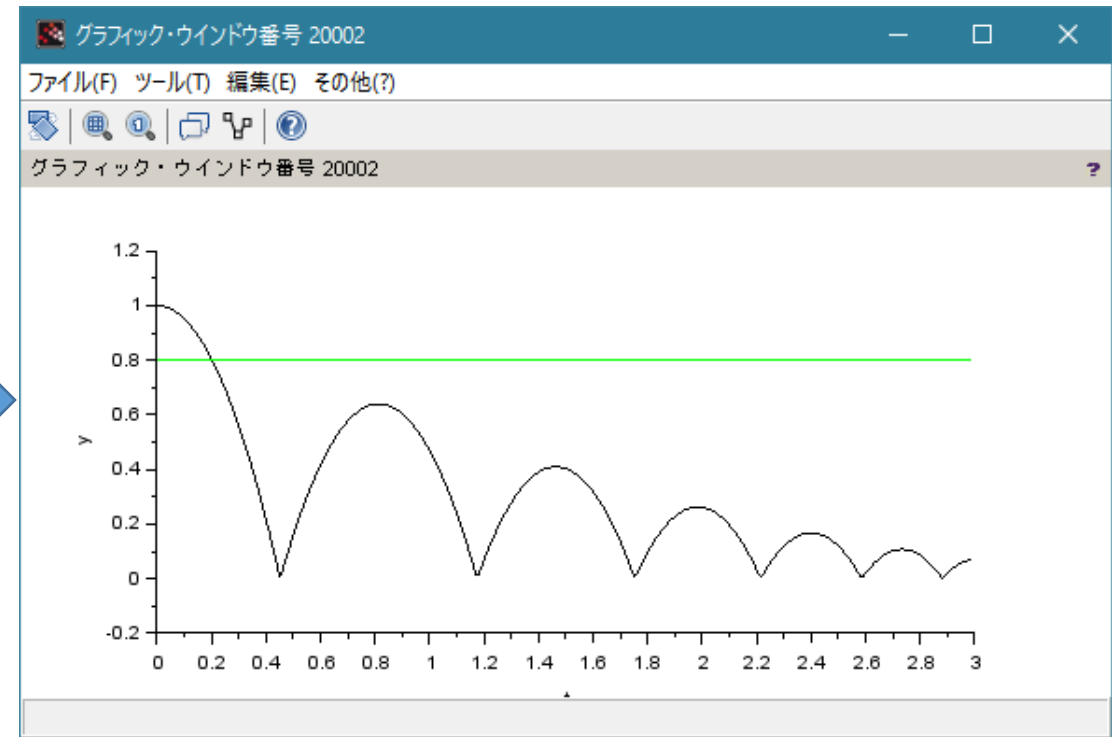
テスト1: 入力変数が一定の場合

② シミュレーション>設定

③ 開始

The screenshot shows the Xcos simulation environment. The 'パラメーター設定' (Parameter Settings) dialog box is open, with the '積分終了時間' (Integration End Time) field set to  $3.0E00$ . The simulation diagram includes a 'balancingBall' block, a 'MUX' block, and a scope icon. The 'シミュレーション' (Simulation) menu is highlighted, and the '開始' (Start) button is also highlighted.

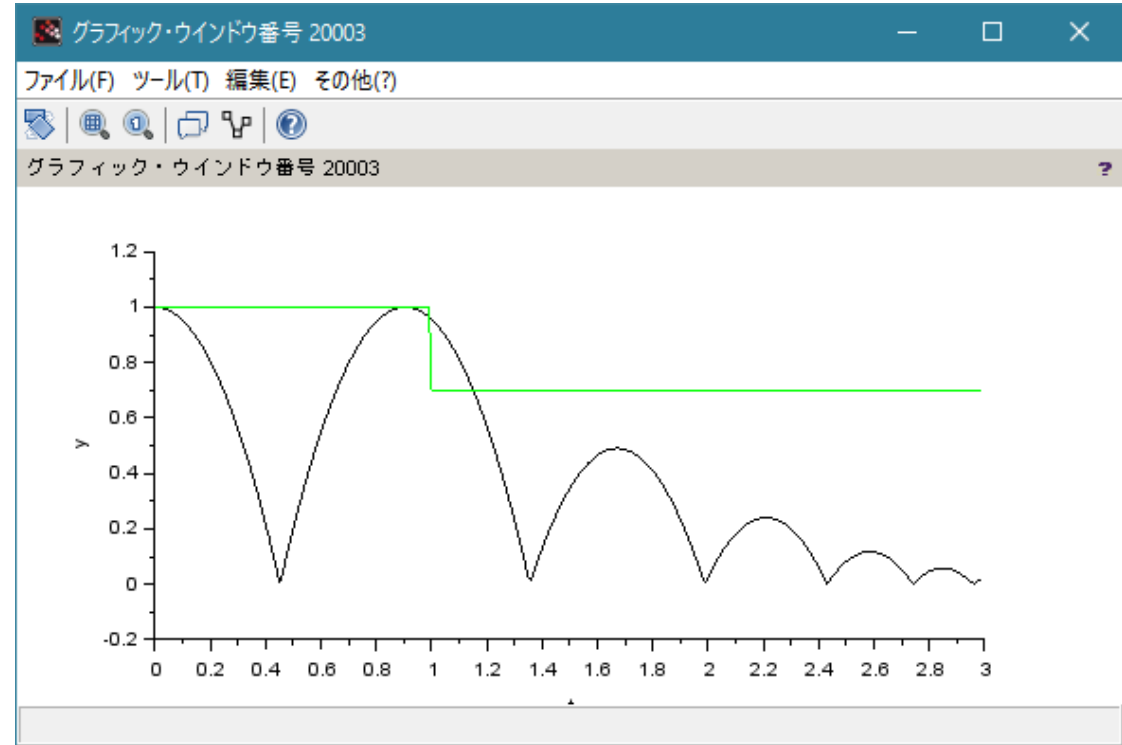
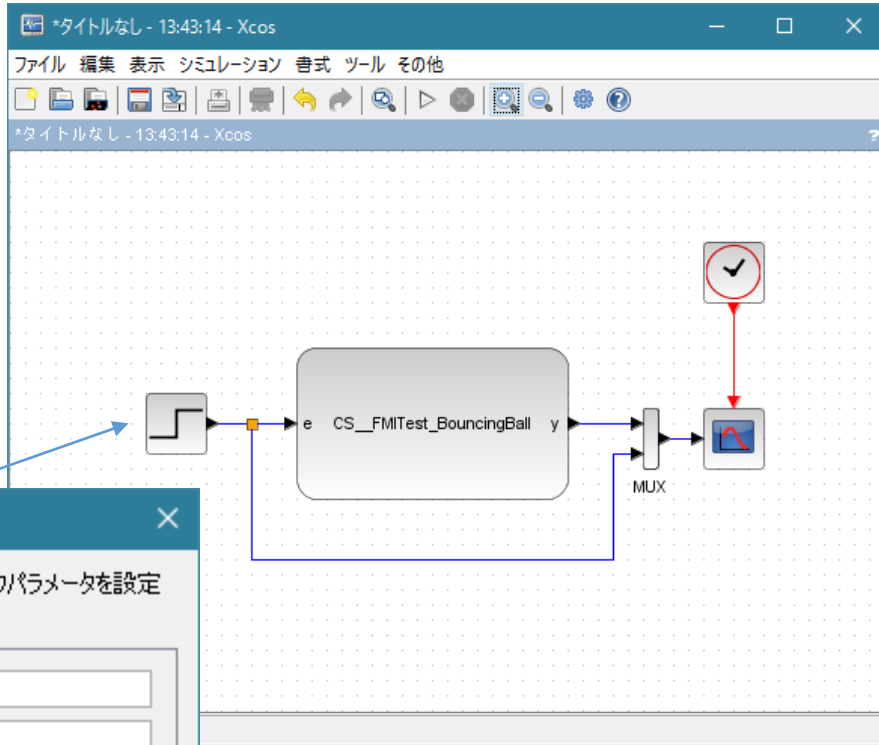
パラメーター	設定値
積分終了時間	3.0E00
実時間スケール	0.0E00
積算絶対許容誤差	1.0E-06
積分相対誤差	1.0E-06
時間の誤差	1.0E-10
最大積分時間間隔	1.00001E05
ソルバーの種類	Sundials/CVODE - BDF - NEWTON
ステップ最大値 (0: 制限無し)	0.0E00



# 2-5 Scilab/Xcos FMU wrapperでFMUを読み込む

## テスト2: 入力変数が増える場合

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	

未検証

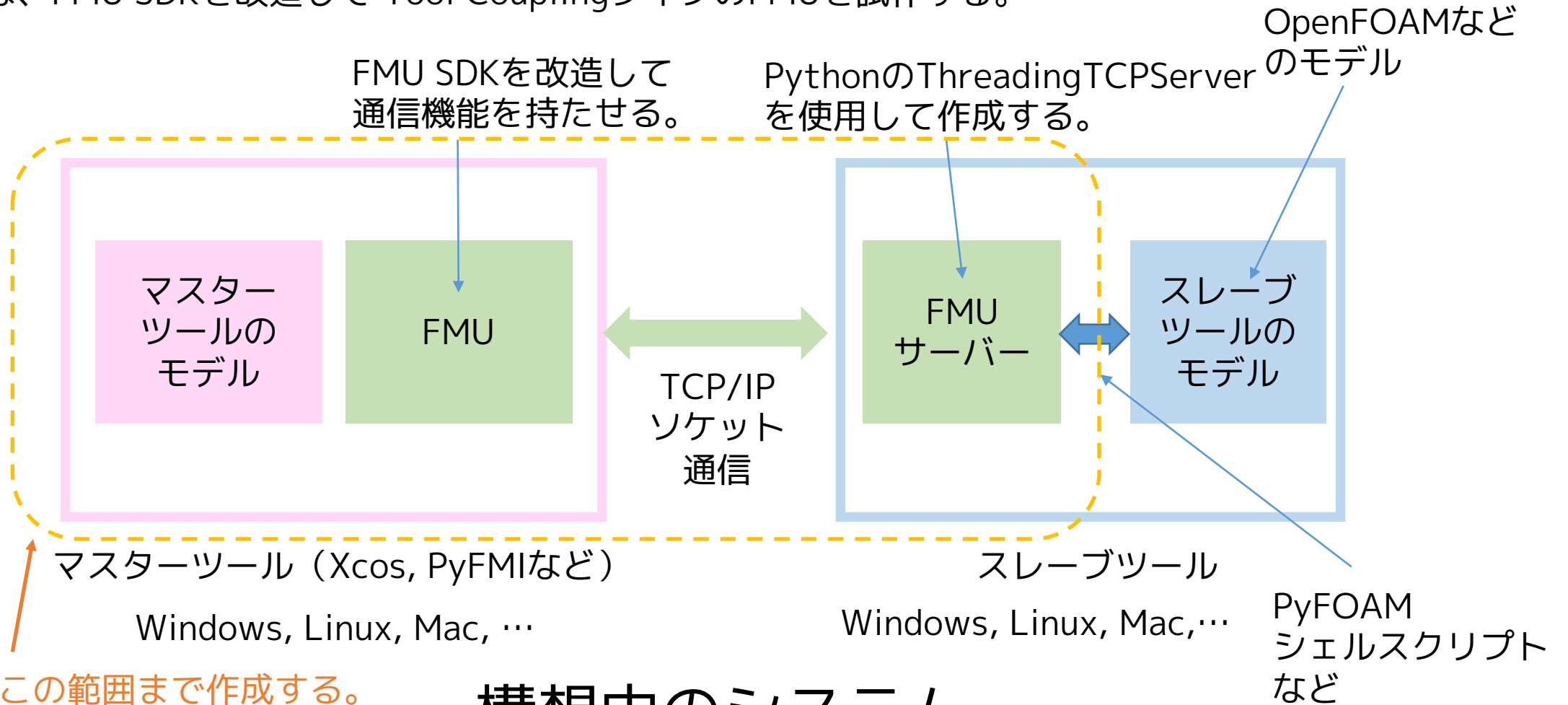
## FMI 1.0 FMI for Co-Simulation の FMUの読み込み マスターツール

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。



今回はこの範囲まで作成する。  
Windows版のみ

## 構想中のシステム

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

- 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)

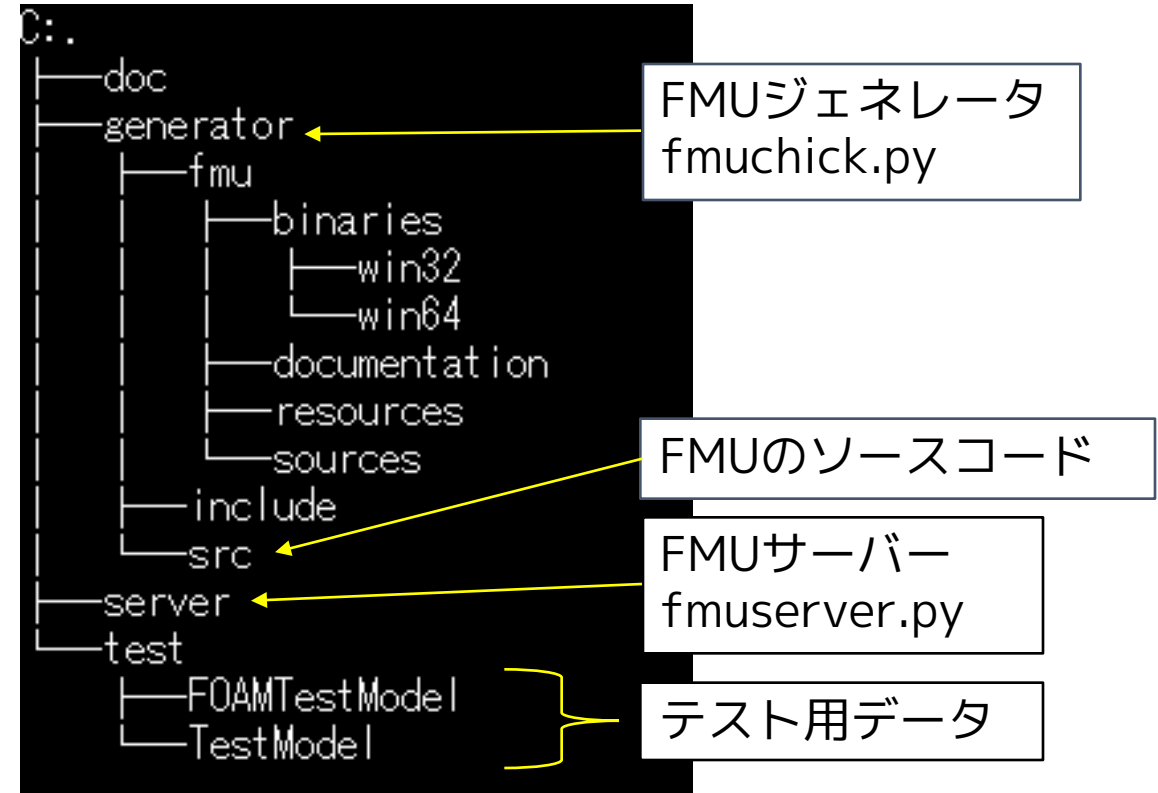
- FMUのソースコード (可変部、モデル固有の情報を含む) を出力する。
- XMLファイルを生成する。
- バッチファイルを使用してコンパイルし、ZIP化を行う。

## 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

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

The screenshot shows the FMU CS Generator application window. It contains several sections: 'Model Information' with a table for Model Name (TestModel) and Description (Simple Test Model); 'Exchange Variables' with two tables for Input and Output variables; 'FMU Information' with fields for FMU Name (TestModel), GUID, IP Address (192.168.3.13), and Port Number (9999); and 'Default Experiment' with fields for Start Time (0), Stop Time (10), and Tolerance (1.0e-4). At the bottom, there are buttons for Clear, Reload, Save, Generate FMU, and Close. The 'Generate FMU' button is highlighted with a red box. Arrows point from various text boxes to specific elements in the interface.

Item	Value
1	Model Name: TestModel
2	Description: Simple Test Model
3	
4	
5	
6	

Input Variable	Start Value	Output Variable	Start Value
1	input1: 0.0	1	output1: 0.1
2	input2: 1.0	2	output2: 0.2
3		3	
4		4	
5		5	
6		6	
7		7	

FMU Information	Default Experiment
FMU Name: TestModel	Start Time: 0
GUID: {c4e810f-3df3-4a00-8276-176fa3c9f003}	Stop Time: 10
IP Address: 192.168.3.13	Tolerance: 1.0e-4
Port Number: 9999	

FMUジェネレータ

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

Scilab複数値リクエスト

CLOCK\_c ブロックパラメータを設定

イベントクロック生成器

初期化時間 が負の場合は開始しない

間隔

初期化時間

OK キャンセル

時間（コミュニケーションポイント）  
の入力端子

Scilab複数値リクエスト

GENSIN\_f ブロックパラメータを設定

サイン波ジェネレータ

ゲイン

周波数 (rad/s)

位相 (rad)

OK キャンセル

Scilab複数値リクエスト

Set Scope parameters

Input ports sizes	<input type="text" value="1 1"/>
Drawing colors (>0) or mark (<0)	<input type="text" value="1 3 5 7 9 11 13 15"/>
Output window number (-1 for automatic)	<input type="text" value="-1"/>
Output window position	<input type="text" value="0"/>
Output window sizes	<input type="text" value="0"/>
Ymin vector	<input type="text" value="-1 -2"/>
Ymax vector	<input type="text" value="1 2"/>
Refresh period	<input type="text" value="100 100"/>
Buffer size	<input type="text" value="20"/>
Accept herited events 0/1	<input type="text" value="0"/>
Name of Scope (label&Id)	<input type="text" value="Input"/>

OK キャンセル

Scilab複数値リクエスト

GENSIN\_f ブロックパラメータを設定

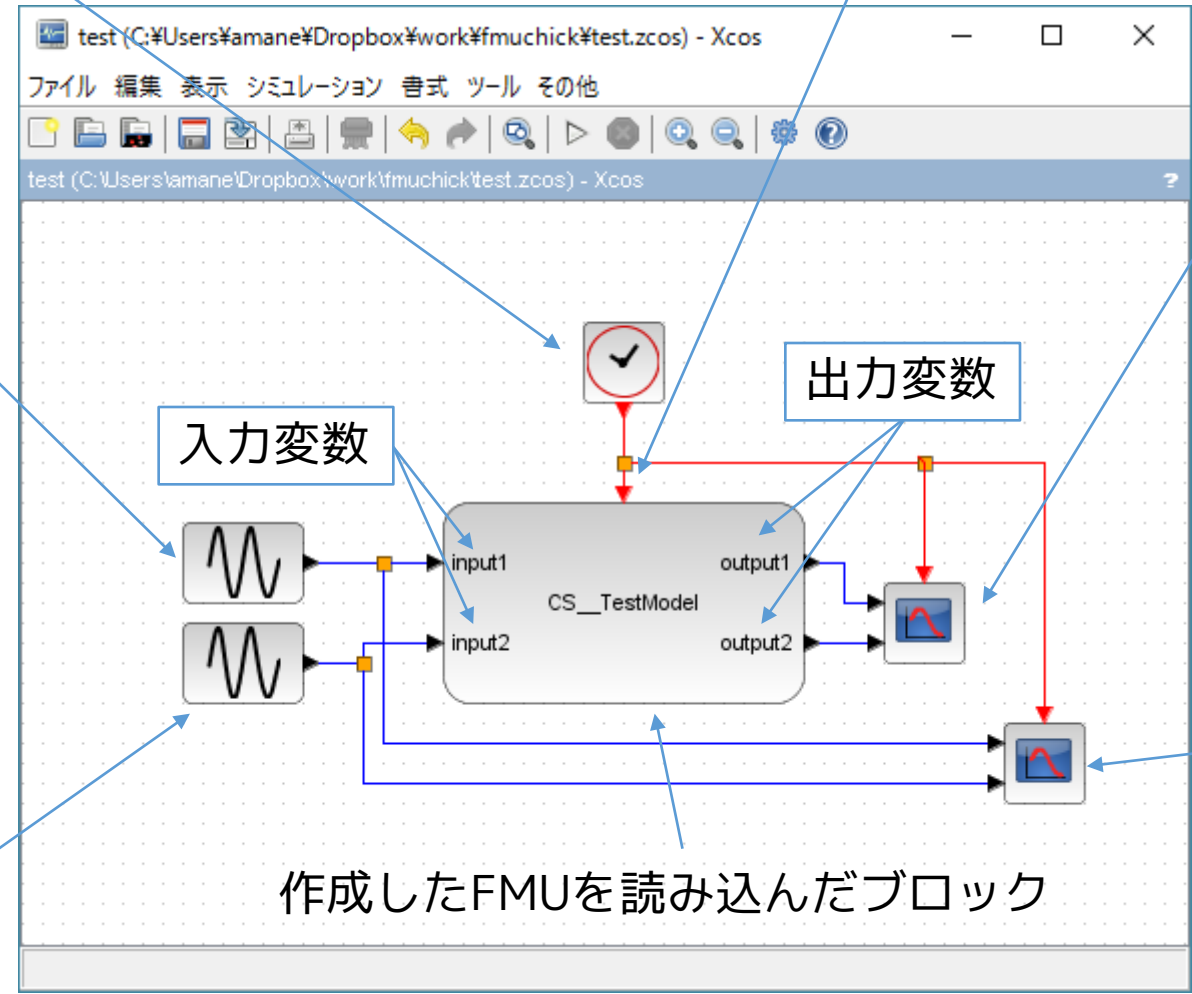
サイン波ジェネレータ

ゲイン

周波数 (rad/s)

位相 (rad)

OK キャンセル



Scilab複数値リクエスト

Set Scope parameters

Input ports sizes	<input type="text" value="1 1"/>
Drawing colors (>0) or mark (<0)	<input type="text" value="1 3 5 7 9 11 13 15"/>
Output window number (-1 for automatic)	<input type="text" value="-1"/>
Output window position	<input type="text" value="0"/>
Output window sizes	<input type="text" value="0"/>
Ymin vector	<input type="text" value="-1 -2"/>
Ymax vector	<input type="text" value="2 4"/>
Refresh period	<input type="text" value="100 100"/>
Buffer size	<input type="text" value="20"/>
Accept herited events 0/1	<input type="text" value="0"/>
Name of Scope (label&Id)	<input type="text" value="Output"/>

OK キャンセル

3. Tool CouplingタイプのFMUを試作する

### 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[]**  
が利用できる。

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, ...

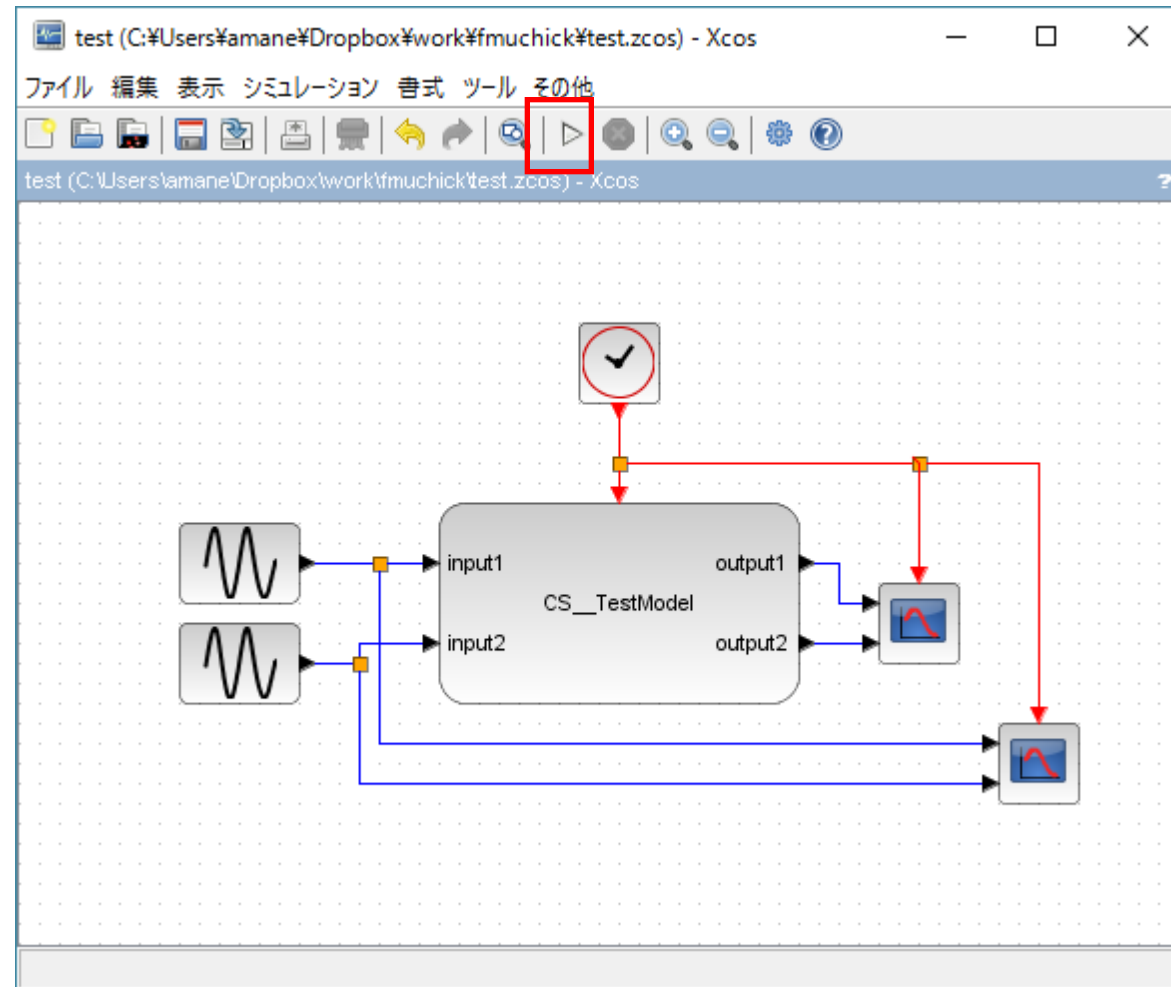
上記をカスタマイズすることで様々なスレーブツールに対応することが可能。



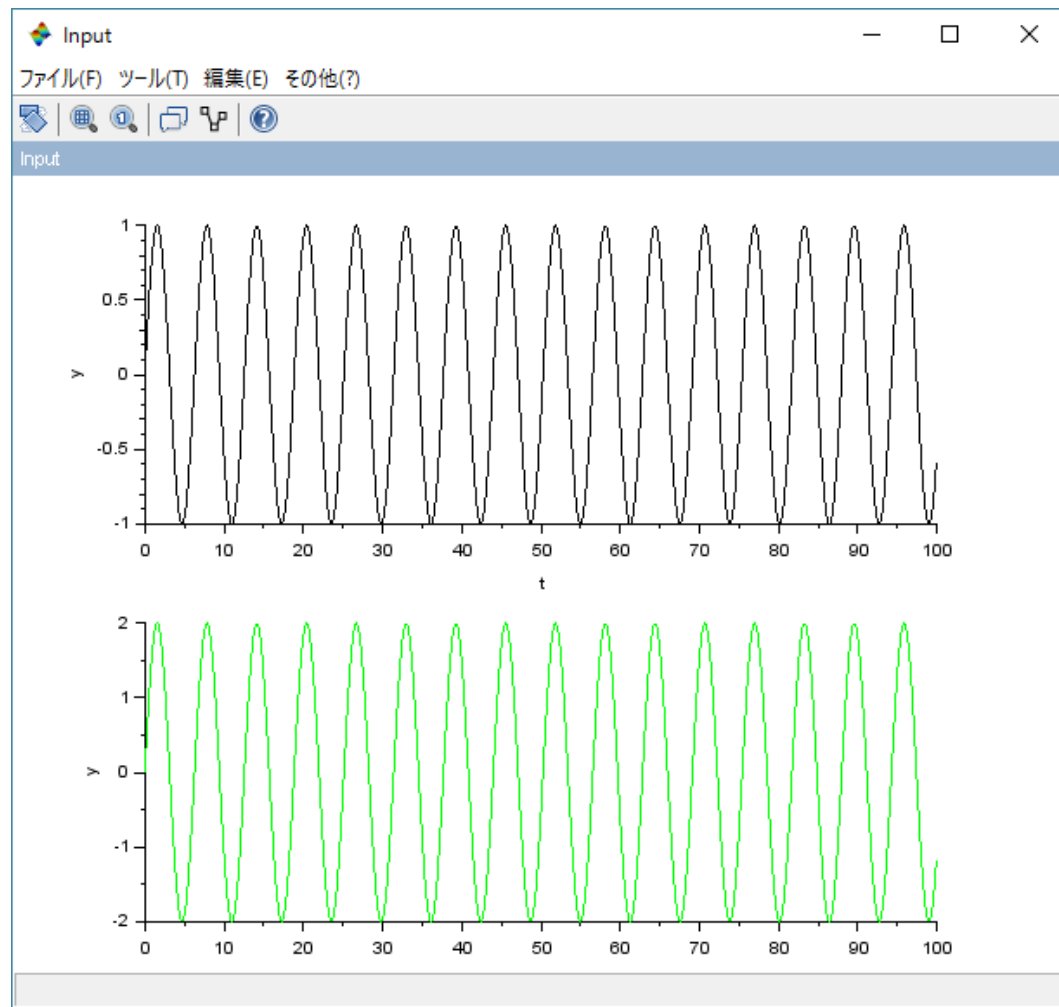
# 3-4 Co-Simulationの実行

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

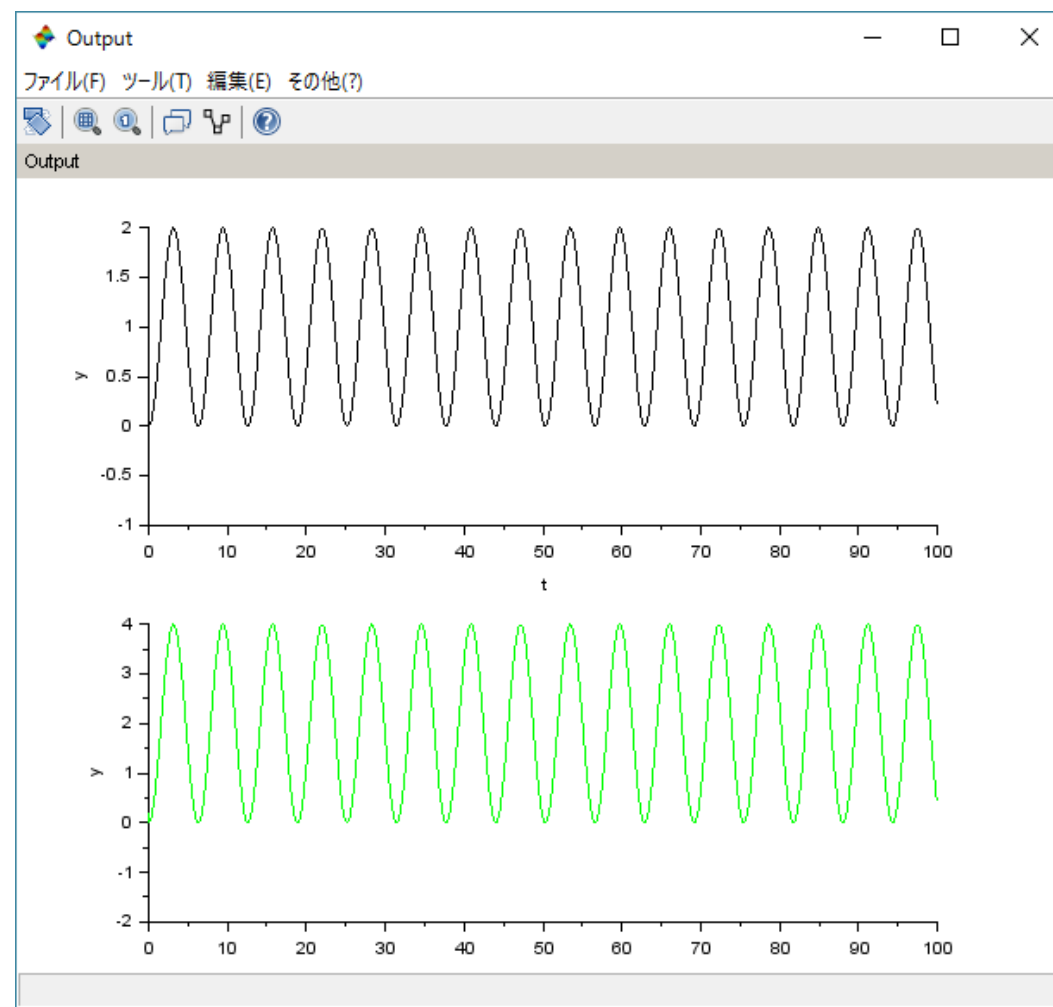
シミュレーションが終了したら、  
Ctrl-C でサーバーを止める。



# 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。