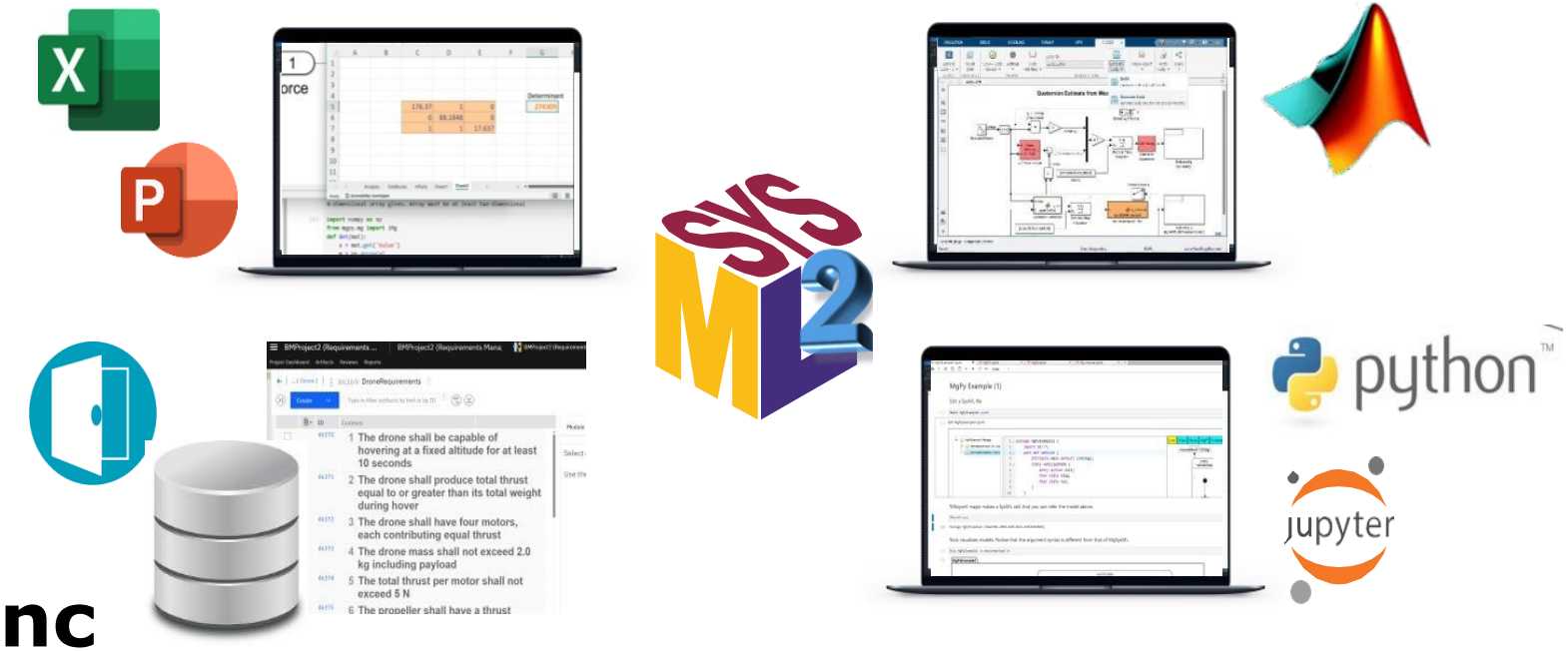
Mg SysML v2を基盤とした MBSE/MBDの統合プラットフォーム



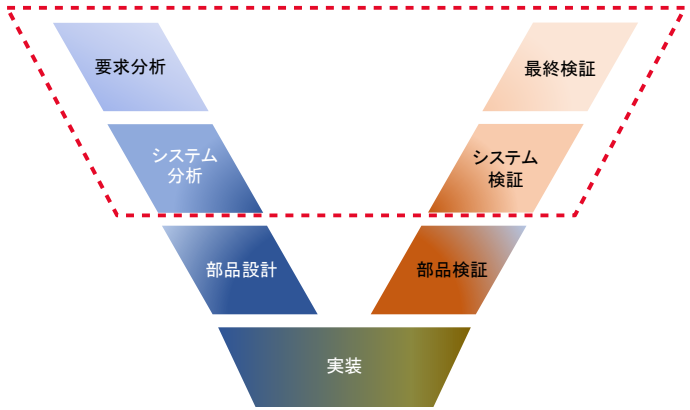
Mgnite Inc

2026/1/27

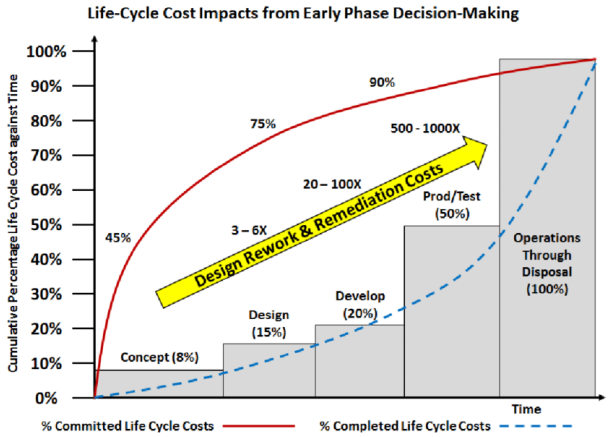
SysMLv2およびMgが求められている背景

複雑・高度化するシステム開発において、上位工程が致命的に重要になっていることは共通認識となっています

MBSEが求められる上位工程



システムが複雑になるほど上位工程がコストに致命的な影響を与えます



しかし、現状でもMBSEは生産性・相互運用性・再利用性に課題を抱えています

モデリング言語の課題

- 複雑で生産性が向上しない
- ツール間の相互運用性がない
- 設計資産の再利用が困難

ツール環境の課題

SysMLv2

- 複雑なUMLから脱却して再構築され、大幅に整理
- テキスト表記、再定義の大幅な強化、強力な形式化

Mg

- オープンソースのSysMLv2標準実装による相互運用性の担保
- Excel, PowerPoint, MATLAB/Simulink, Python, DOORS Nextなど、主要環境やツールとの高度な連携

SysMLv2とMgがもたらす解決

- 複雑で生産性が向上しない
- 設計資産の再利用が困難
- ツール間の相互運用性がない

2025年からの強化点

機能強化

MgSの機能強化

- Reverse機能が導入され、Simulinkでの接続線編集をSysMLに反映させられるようになりました
- SysMLとの対応が強化され、SysMLによる式をSimulinkのブロック線図に変換することができるようになりました

MgXの機能強化

- Excelからのモデル編集が行えるようになりました
  これによって、表形式で柔軟にモデルをExcelから編集することができます
- MgPyからExcelシートの操作が行えるようになりました

*Public versionには、まだ導入されていません

MgDNGの追加

- DOORS Next（DNG）のモデルとSysML間を相互に変換することができるようになりました
- SysMLのモデルライブラリによって、柔軟にDNGのArtifact Typeに対応できます

*Public versionには、導入予定はありません

MgPyの機能強化

- Jupyter上でデバッグが可能となりました
- モデル・編集操作や木構造操作のための多くのAPIが追加されました
  （Public versionには、まだ導入されていません）
- mgpy.cmdによって、コマンドラインから独立で起動可能になりました
- Jupyter Kernel Gatewayによって、MgPyをREST APIで操作することができるようになりました
  これによって、他のツールとの連携や、CI/CD連携が容易になります

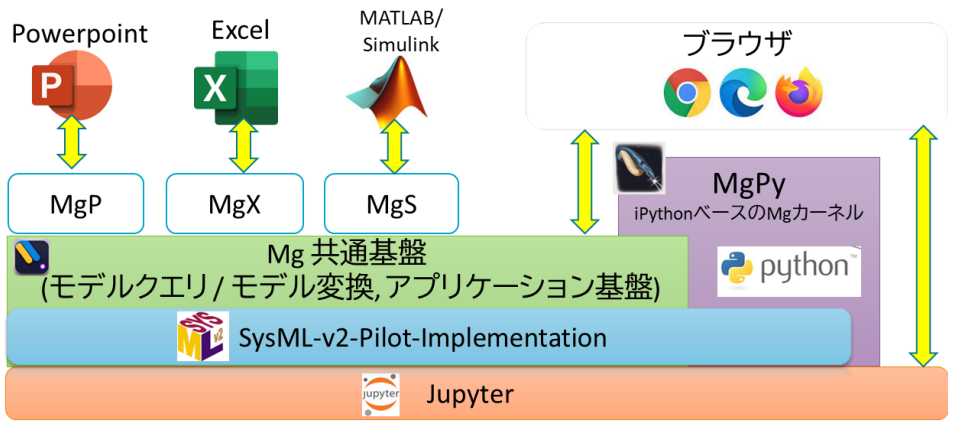
Mgの概要

■ Mgは、Mgnite社によるSysMLv2を基盤とした統合モデリング環境です

JupyterLab上で、SysMLv2のモデリング及び、PowerPoint, Excel, Python, MATLAB/Simulinkとの統合を提供します

- Python統合(MgPy)は、標準のMgによって提供されています
- PowerPoint統合は、MgPによって提供されています
- Excel統合は、MgXによって提供されています
- MATLAB/Simulink統合は、MgSによって提供されています

  - **現状では、Windows 11の64bit環境で動作します**



Mgによって以下のようなことができます

- SysMLv2でのテキスト形式でのモデリングとグラフィカル形式での同時可視化
- MgXによるExcel形式でのビューの提供 (0.5.xでモデル編集機能も提供される予定)
- MgPによるPowerPoint形式でのプレゼンテーションの作成
- MgSによるMATLAB/Simulinkの生成及びシミュレーションと結果取得
- MgPyによるPythonでのスクリプティング (Mgのほぼすべての機能はPythonからアクセスできます)
- JupyterLabによる分析・評価・自動化の統合環境

インストールおよび起動

■ Mg-win64-*{version}*.zipを適当な書き込み可能な場所に展開するだけです

- 例えば、ホームディレクトリ下(c:¥Users¥<ユーザー名>¥Mg)などが適切です

■ MATLAB/Simulinkとともに使う場合にはMATLAB.cmdのコメントをはずしてMATLAB.exeの場所を指定してください

rem set MG_MATLAB_EXE_PATH=c:/Program Files/MATLAB/R2021a/bin/matlab.exe →
set MG_MATLAB_EXE_PATH=c:/MATLAB/R2021a/matlab.exe

- なお、MATLABバージョンはR2023aを当面は推奨します

起 動

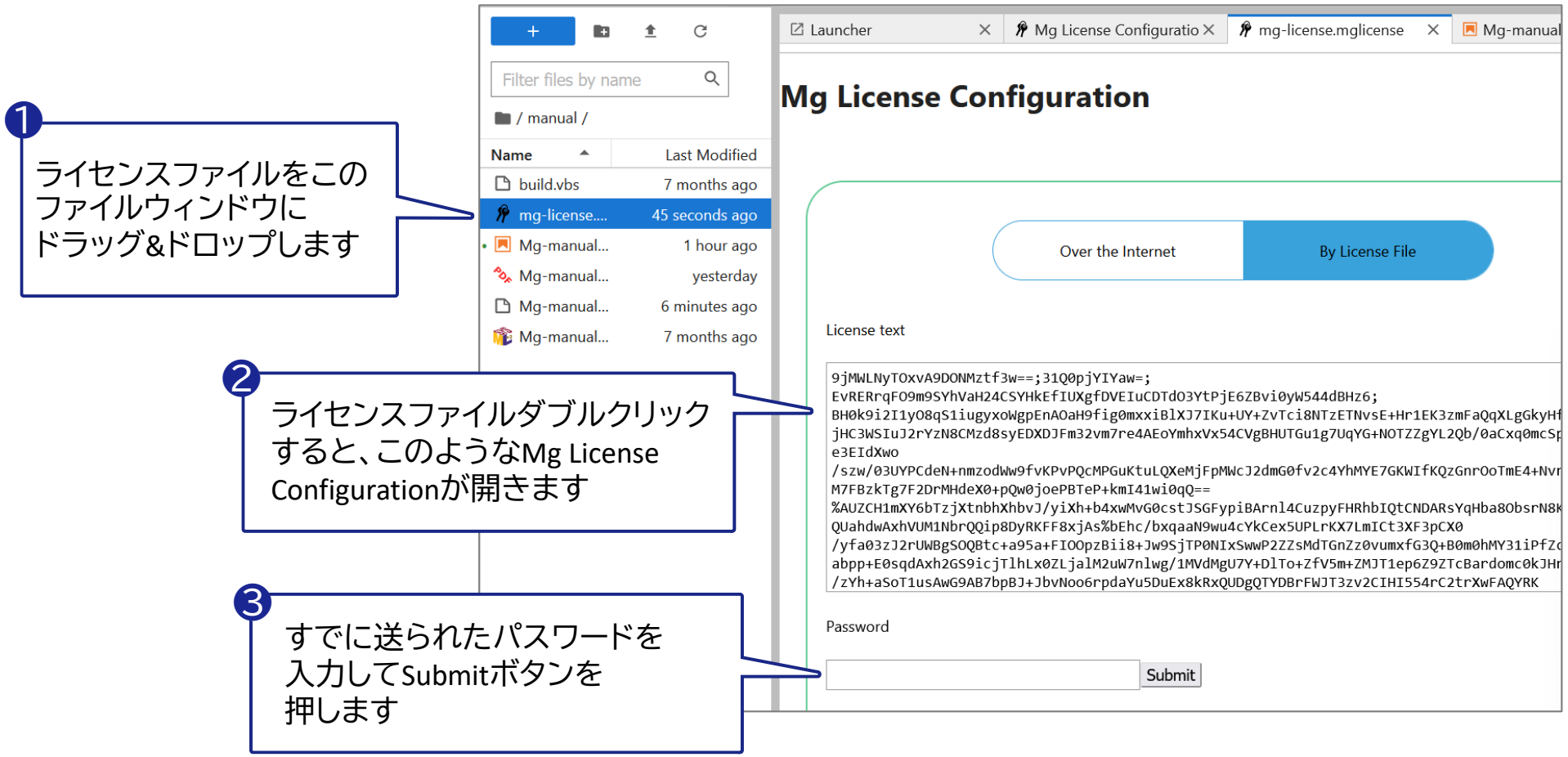
- 展開場所にあるstart.cmdをエクスプローラーでダブルクリックしてください
  - その場合には、展開場所のwsフォルダが作業フォルダになります
- launch.cmdを使って起動した場合にはカレントディレクトリはそのまま作業フォルダになります
- なお、Mgは複数起動することもできます。ただし、MgSにおけるMATLAB/Simulinkの起動は現状で一つに限られます
  - 複数のMgが一つのMATLAB/Simulinkと相互作用します
  - MgX, MgPについては、カーネルごとにExcel, Powerpointと相互作用します

終 了

- メニューのFileからShut Downを選んでください
  - Excel, PowerPointは終了になりますが、プロセスが残ることがあります。その場合にはタスクマネージャーからプロセスを終了させてください。将来のバージョンでは、強制終了のオプションを追加する予定です
  - MATLAB/Simulinkは起動したまま残りますので、別途終了させてください。

ライセンス登録

■ 御社にはライセンスファイルを別途お渡ししますので、以下の手順で登録をお願いします



EV2サンプルを試す（紹介する機能の目的とうれしさ）

■ EV2サンプルでは、Mgの以下の主要な機能を紹介します

Jupyterノートブックで一括してSysMLv2のモデリングと分析ができる機能

SysMLv2のテキスト表現のファイルを即座に可視化しながら編集できる機能

SysMLで記述された様々な車両諸元や実験条件から、Simulinkを生成・実行して分析できる機能

モデルや分析結果をExcelにて表示・加工することができる機能

モデルからPowerPointプレゼンテーションを作成することができる機能

生産性の高い SysMLv2統合環境の実現

MBSEの実現では、モデルを構築するだけではなく、分析などに活用できることが重要で、Mgは様々な用途に活用できるJupyterLabによる統合環境を提供します

MBSEによる MBDの効率化

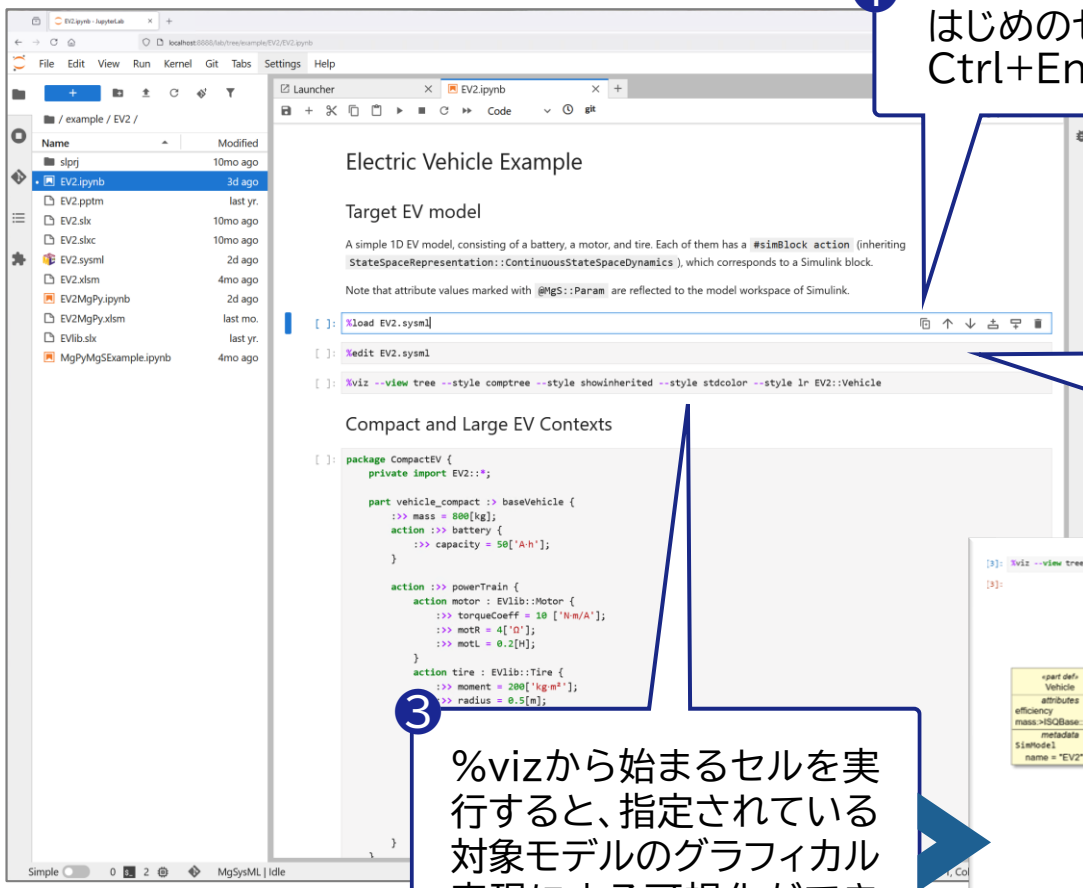
MBSEの価値を下流工程で活用するために重要とみなされているMBDに適用することは、生産性向上に向けて極めて重要になります。Mgによって、SysMLv2をもとに効率的にMBDの主要ツールであるSimulinkを活用できます

オフィス（Excel, PowerPoint）とのシームレスな連携

実際の特に上流工程では、コミュニケーションのためにExcelおよびPowerPointが多用されています。Mgは、ExcelやPowerPointと密接に連携することができ、SysMLv2の価値をオフィス製品にもたらすことで、飛躍的に生産性を向上させることを狙っています

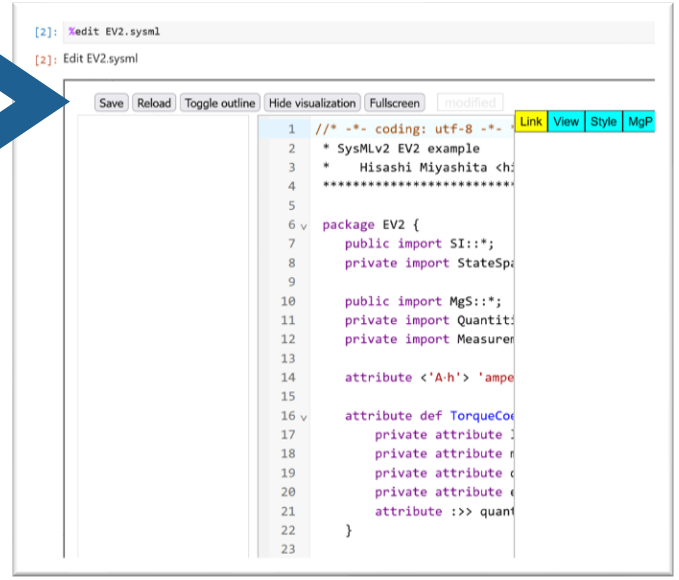
EV2サンプルを試す（1: ノートブックの実行）

■example¥EV2 フォルダにあるEV2.ipynb（Jupyterノートブックには*.ipynbの拡張子がつきます）をダブルクリックすると以下のようなページが開きます

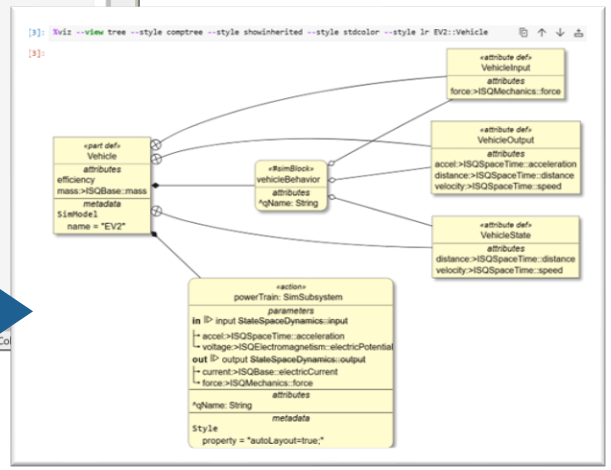


❶ はじめのセルから順番にクリックして選択し、Ctrl+Enterで実行していきます

❷ %editから始まるセルを実行すると、エディタが開き、対象ファイル（この場合はEV2.sysml）がその場で編集できます



❸ %vizから始まるセルを実行すると、指定されている対象モデルのグラフィカル表現による可視化ができます



Jupyterでは、ノートブックを開くと、カーネルが起動します。カーネルとは一連のコマンドを処理する個別のプロセスで、ノートブックでの評価内容は、そのカーネルの中だけで有効になります。カーネルを再起動する（メニューのKernel→Restart Kernel）と、最初の評価からやり直すことができます

EV2サンプルを試す（2: SysMLの評価）

■ 通常のセル（%ではじまらない）を実行する（Ctrl+Enter）と、内容はSysMLとして読み込まれます

■ 読み込まれたモデルは%editで開かれたモデルと同様に、カーネル内で共有されます

Compact and Large EV Contexts

[4]:

```
package CompactEV {
    private import EV2::*;

    part vehicle_compact :> baseVehicle {
        :>> mass = 800[kg];
        action :>> battery {
            :>> capacity = 50['A·h'];
        }

        action :>> powerTrain {
            action motor : EVlib::Motor {
                :>> torqueCoeff = 10 ['N·m/A'];
                :>> motR = 4['Ω'];
                :>> motL = 0.2[H];
            }
            action tire : EVlib::Tire {
                :>> moment = 200['kg·m²'];
                :>> radius = 0.5[m];
            }
            bind input.voltage = motor.input.voltage;
            bind motor.output.current = output.current;

            flow mt2tt from motor.output.torque to tire.input.torque;
            flow to2mf from tire.output.torque to motor.input.friction;

            bind input.accel = tire.input.accel;
            bind tire.output.force = output.force;
        }
    }
```

このセルでは、基本EVの記述からコンパクトEV用の差分を記述しています

車重とバッテリーサイズを再定義によって与えています

モーターやタイヤの諸元を変更しています

タイヤ、モーター、バッテリーを結合しています

SysMLv2では、この例に挙げたように、再定義によって極めて効率的に差分的に記述を行うことができます。Mgでも、ノートブック中に短く差分を記述したり、Pythonが差分を与えるなどの効率的な処理ができるように配慮されています

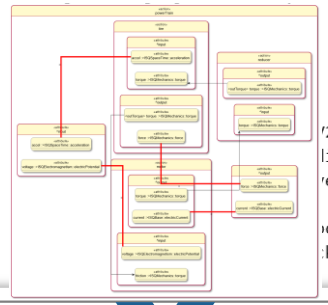
EV2サンプルを試す（MgSによる分析の実行）

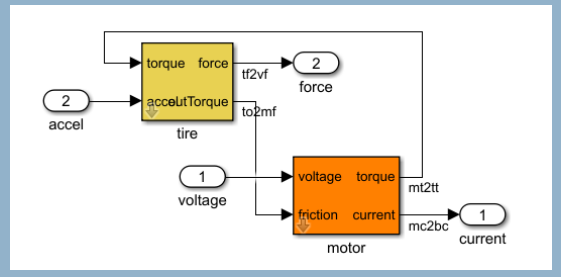
Analysis Evaluation

分析のリストを表示します

```
[7]: %mgs list
[7]: analysisCases, smallEVAnalysis, efficiencyAnalysisSmall, largeEVAnalysis, maxSpeedAnalysisSmall, rangeAnalysisSma
     lysisLarge, subAnalysisCases, efficiencyAnalysisLarge, maxSpeedAnalysisLarge
[8]: %mgs exec rangeAnalysisSmall
[8]: [Success] Binding: null is ma
     [Success] Binding: null is ma
     [Success] Flow connection: mt
     [Success] Flow connection: to
     [Success] Binding: null is ma
     [Success] Binding: null is ma
     pc2bc is matched with Line (pc2bc) |powerTrain/current| --> |EV2/battery|
     va2pa is matched with Line (va2pa) |vehicleBehavior/accel| --> |powerTrain/accel|
     pf2vf is matched with Line (pf2vf) |powerTrain/force| --> |vehicleBehavior/force|
```

SysMLv2での車両モデル及びパラメータはMgSによって、Simulinkに生成・反映され、分析条件に従って自動的に実行されます。実行結果はSysMLv2モデルと結びつけられ、MgXやMgPyなどで後ほど活用することができます





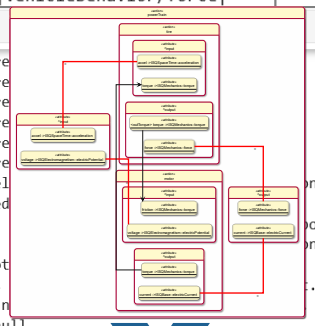
rangeAnalysisSmallによって生成されたSimulinkモデル

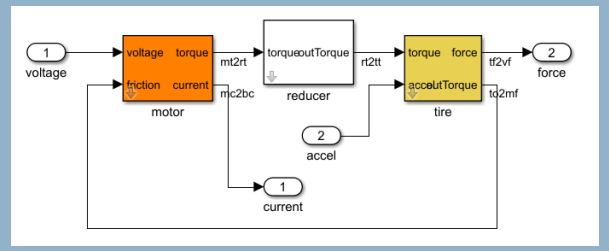
rangeAnalysisSmall分析を実行します

```
analysis rangeAnalysisSmall :> smallEVAnalysis : RangeAnalysis {
    :>> simCond { :>> toTime = 5000[s]; }
    requirement :>> rangeRequirement = rangeRequirementSmall;
    return simulatedRange = vehicle.vehicleBehavior.output.distance {
        @MgS::SimRet {
            ret = MgS::Retrieval::FINAL;
        }
    }
}
```

rangeAnalysisSmallは上記のようにモデル化されているSysMLv2による分析記述であり、コンパクト車(SmallEVAnalysisをsubset化)の5000秒までのシミュレーションを行い、距離の最終結果(MgS::Retrieval::FINAL)を結果として返すことが記述されています。

```
[6]: %mgs exec maxSpeedAnalysisLarge
     Flow connection: rt2tt has no corre
     Flow connection: to2mf has no corre
     Flow connection: bv2pv has no corre
     Flow connection: pc2bc has no corre
     Flow connection: va2pa has no corre
     Flow connection: pf2vf has no corre
     Line (va2ta) |vehicleBehavior/accel
     !!It had an MgS tag and was removed
     Line () |vehicleBehavior/distance|
             |vehicleBehavior/velocity|
     SLO:EV2/Accel[block] does not
     SLO:EV2/Distance[block] does
     SLO:EV2/Velociy[block] does n
     a new line for connection: null
     a new line for connection: null
```





maxSpeedAnalysisLargeによって生成されたSimulinkモデル

maxSpeedAnalysisLarge分析を実行します

maxSpeedAnalysisLargeは、大型車(LargeEV)を対象(subject)としているため、異なる構成と異なるパラメータ条件でのシミュレーションを実行し、速度の最終結果および電圧推移全体を返します

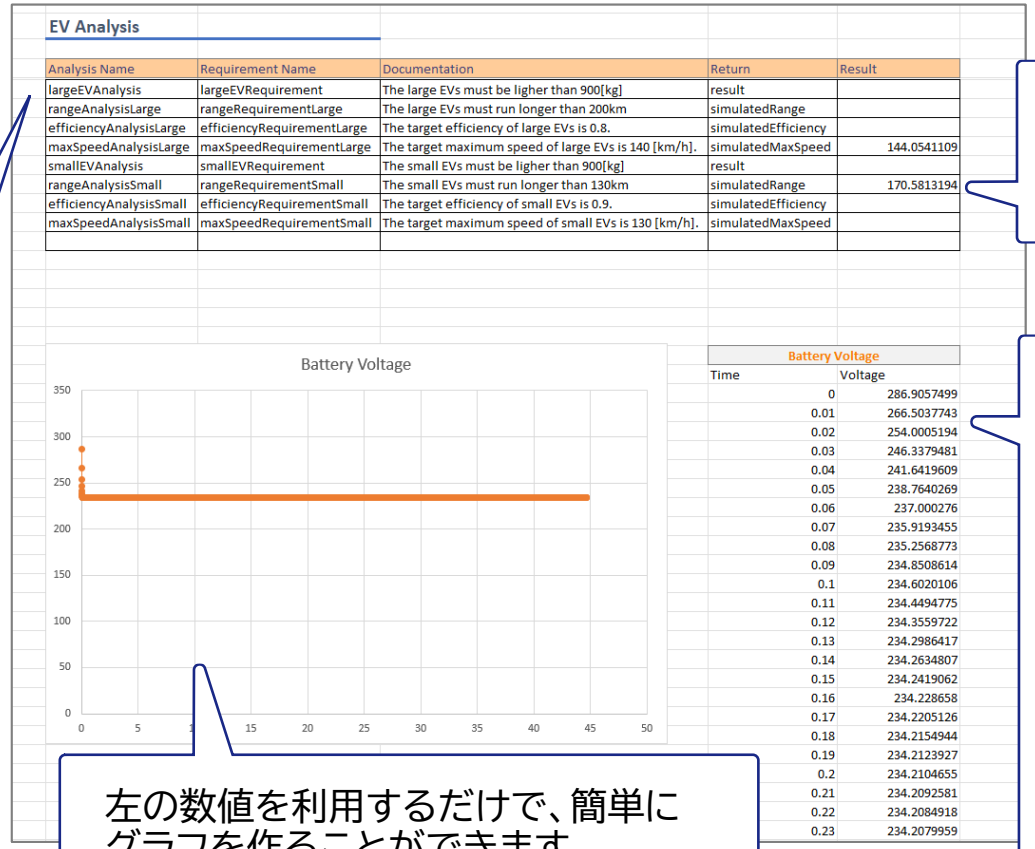
```
analysis maxSpeedAnalysisLarge :> largeEVAnalysis : MaxSpeedAnalysis {
    :>> simCond { :>> toTime = 700[s]; }
    requirement :>> maxSpeedRequirement = maxSpeedRequirementLarge;
    out voltage = vehicle.battery.output.voltage {
        @MgS::SimRet {
            ret = Retrieval::FULL;
        }
    }
    return simulatedMaxSpeed = vehicle.vehicleBehavior.output.velocity {
        @MgS::SimRet {
            ret = Retrieval::MAX;
        }
    }
}
```


EV2サンプルを試す（MgXによる結果の提示）

```
[12]: %mgx open EV2
[12]: Opened MgX template: .\EV2.xlsm
```

%mgx open <モデル名> で、対応するMgXテンプレートを開きます(通常は<モデル名> .xlsm)
ここでの<モデル名>は、EV2になります

Excelが起動後、EV2.xlsmが開かれます。一部のセルの数式が表示されないことがありますがF9キーを押下することで、再計算されて表示されます。



EV Analysis

Analysis Name	Requirement Name	Documentation	Return	Result
largeEVAnalysis	largeEVRequirement	The large EVs must be ligher than 900[kg]	result	
rangeAnalysisLarge	rangeRequirementLarge	The large EVs must run longer than 200km	simulatedRange	
efficiencyAnalysisLarge	efficiencyRequirementLarge	The target efficiency of large EVs is 0.8.	simulatedEfficiency	
maxSpeedAnalysisLarge	maxSpeedRequirementLarge	The target maximum speed of large EVs is 140 [km/h].	simulatedMaxSpeed	144.0541109
smallEVAnalysis	smallEVRequirement	The small EVs must be ligher than 900[kg]	result	
rangeAnalysisSmall	rangeRequirementSmall	The small EVs must run longer than 130km	simulatedRange	170.5813194
efficiencyAnalysisSmall	efficiencyRequirementSmall	The target efficiency of small EVs is 0.9.	simulatedEfficiency	
maxSpeedAnalysisSmall	maxSpeedRequirementSmall	The target maximum speed of small EVs is 130 [km/h].	simulatedMaxSpeed	

Battery Voltage

Time	Voltage
0	286.9057499
0.01	266.5037743
0.02	254.0005194
0.03	246.3379481
0.04	241.6419609
0.05	238.7640269
0.06	237.000276
0.07	235.9193455
0.08	235.2568773
0.09	234.8508614
0.1	234.6020106
0.11	234.4494775
0.12	234.3559722
0.13	234.2986417
0.14	234.2634807
0.15	234.2419062
0.16	234.228658
0.17	234.2205126
0.18	234.2154944
0.19	234.2123927
0.2	234.2104655
0.21	234.2092581
0.22	234.2084918
0.23	234.2079959

シミュレーション結果もモデルクエリで取得できます

表の一行目にはモデルをクエリするための数式が記述されてます。
（なお、その他の行はMgXが生成することができます）
例えば、一番左上のセルには、
=MgQ(“*.*.#{Part}.*.#{AnalysisCaseUsage}”)と記述されており、Partの下のAnalysisCaseUsageをクエリしています

シミュレーション結果が時系列データであっても取得することが可能です。例えば、
=TRANSPOSE(MgQ(“.prop(data).voltage”, C9))
と記述することで、C9セルに対応する分析の電圧結果を転置して取得することができます(なお、TRANSPOSEによって行方向に数字が配置されます)

MgXは、Excel上で数式によって柔軟にモデルをクエリすることができるため、自由度の高いExcelの活用を行うことができます。
近く、Excelによるモデル編集も可能になる予定です

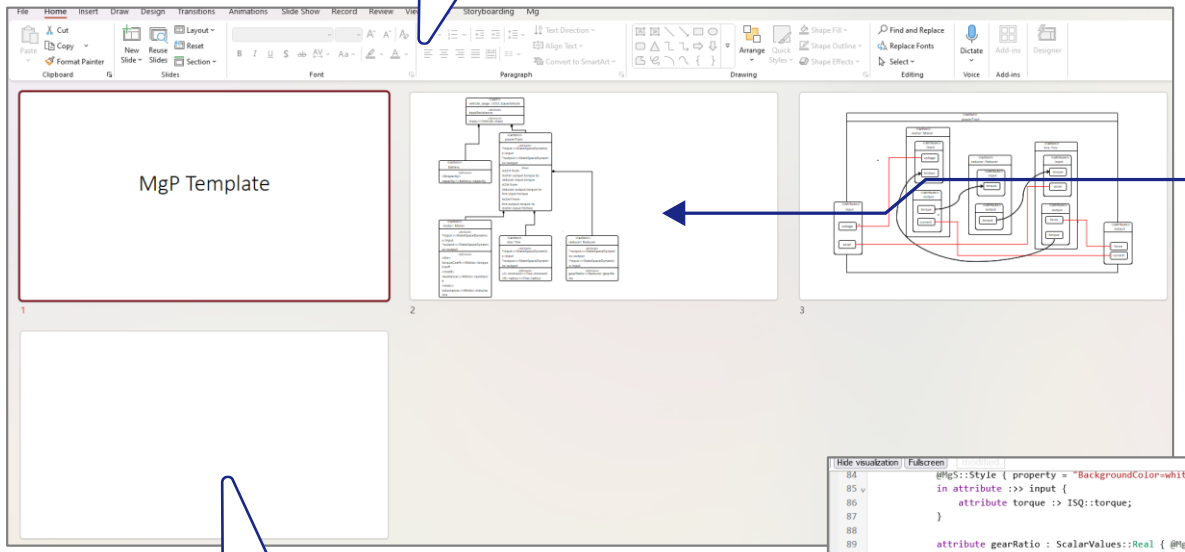
左の数値を利用するだけで、簡単にグラフを作ることができます

EV2サンプルを試す（MgPによるプレゼンテーション）

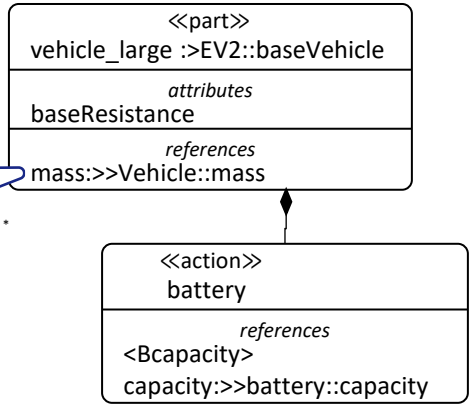
```
[13]: %mgp open EV2
[13]: Opened MgP template: .\EV2.pptm
```

%mgp open <モデル名> で、対応するMgPテンプレートを開きます(MgXと同様です)。この例では、<モデル名>は、EV2になります

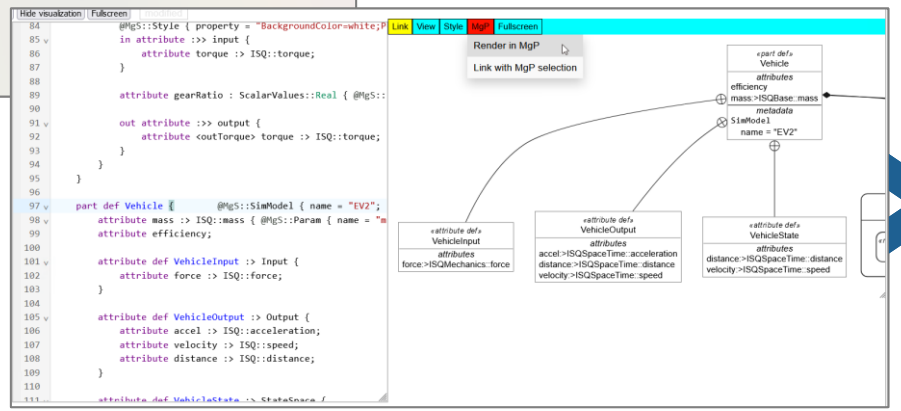
PowerPointが起動し、EV2.pptmが開かれます



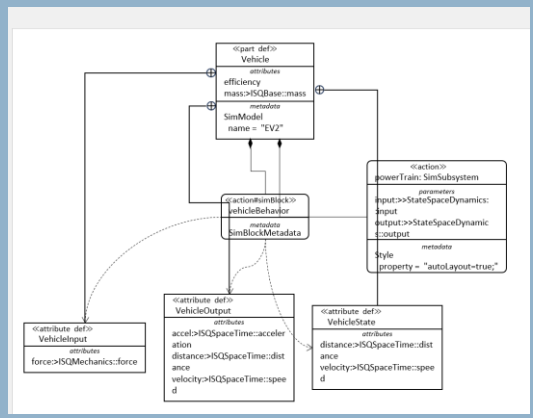
すでに描かれているPowerPointシェイプは、あらかじめ、MgPによって描画されたものです。
右の例のように、PowerPointシェイプとして自由に編集することができ、モデル要素としても対応付けられています



空白のスライドを選択し、Mg上のモデルエディタから、MgP → Render in MgPを選択することで、PowerPointシェイプによるモデルの描画を行うことができます。これらのシェイプもモデルと関連付けられています。



描画されたPowerPointシェイプ

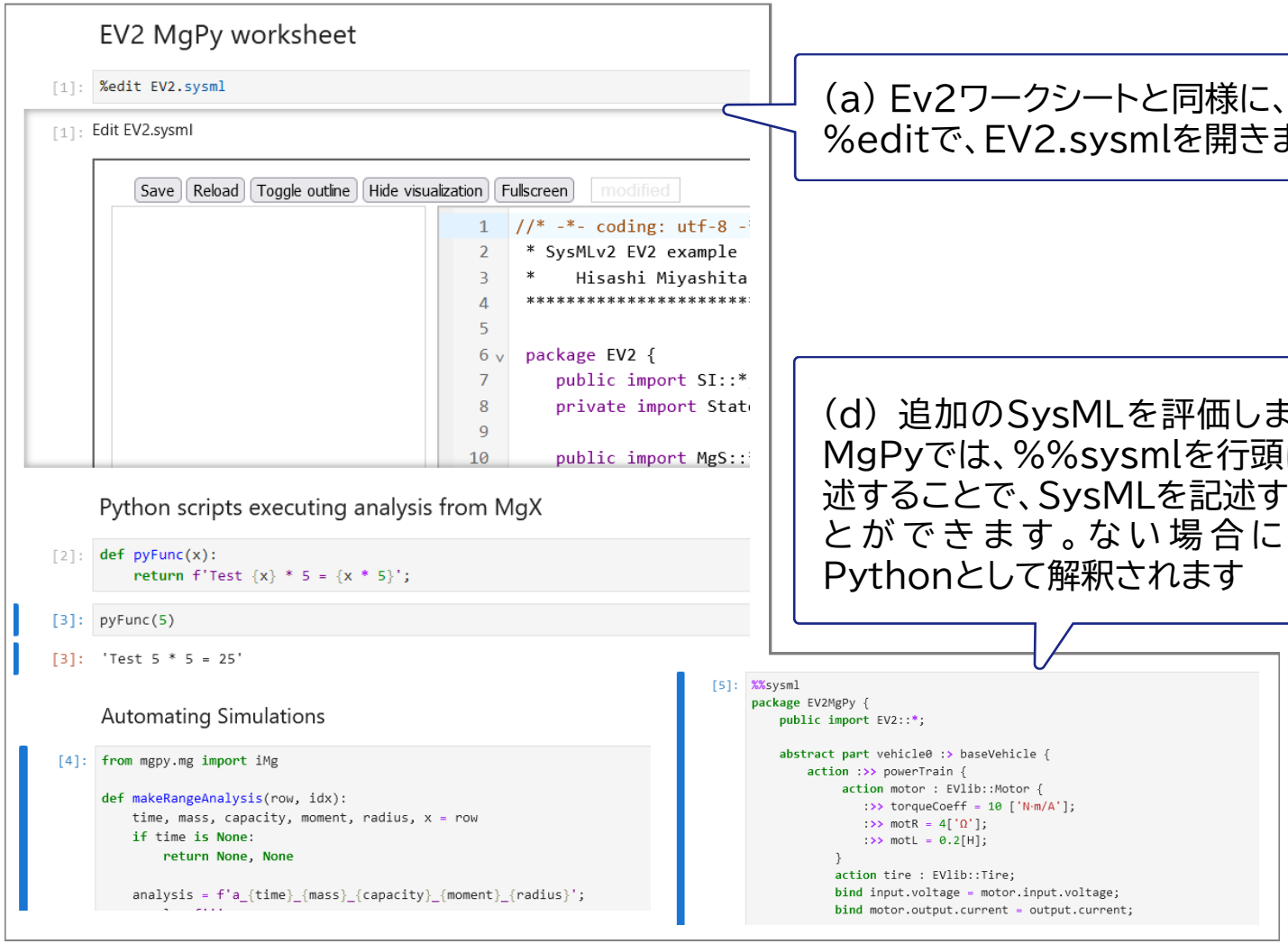


EV2-MgPyサンプルを試す（1: 準備）

このサンプルでは、Pythonによる処理の自動化のデモを行います。Mgでは、ほぼすべての機能は、Pythonによってコントロールすることができるうえ、SysMLv2のモデルにもPythonでアクセスすることができます

（b）簡単なPythonの関数を定義します。これは、数値をメッセージに変換する簡単な関数ですが、後にMgXによってExcelから利用します

（c）次にPythonによるシミュレーション自動化のためのスクリプトを評価します。これはExcel上でボタンを押すと実行され、Excelのセルから適切なSysMLを生成してMgSによってシミュレーションを実行します



（a）Ev2ワークシートと同様に、%editで、EV2.sysmlを開きます

（d）追加のSysMLを評価します。MgPyでは、%%sysmlを行頭に記述することで、SysMLを記述することができます。ない場合には、Pythonとして解釈されます

EV2-MgPyサンプルを試す（2: MgX, MgSの実行）

（e）%mgx EV2MgPyで、MgXを起動します

Launch MgX

```
[6]: %mgx open EV2MgPy
[6]: Opened MgX template: .\EV2MgPy.xlsm
```

（f）Pythonによる計算のサンプルです。=MgPyA(<Python関数名>, 引数…)で、ノートブックの関数を呼び出すことができます。この例では、セルの値を(b)で定義したpyFuncに呼び出して、メッセージを取得しています

（g）このボタンを押すことで(c）で定義したPythonを呼び出します。このPythonスクリプトは、下の表の値を受け取り、SysMLv2を生成して、MgSを起動して、結果をRangeセルに埋めます

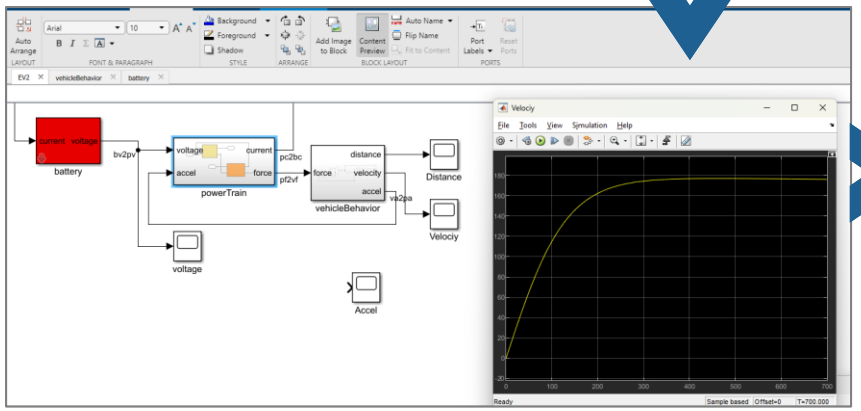
=MgPyA("pyFunc", VALUE(B3))

INPUT	Pythons Calcualtion
5	Test 5.0 * 5 = 25.0
100	Test 100.0 * 5 = 500.0

Analyze

EV2 Range Simulation

time[s]	mass [kg]	capacity [Ah]	moment [kg m2]	radius [m]	Range
500	800	50	200	0.5	
700	900	70	200	0.5	



time[s]	mass [kg]	capacity [Ah]	moment [kg m2]	radius [m]	Range
500	800	50	200	0.5	20.5266078
700	900	70	200	0.5	30.0427841

この例のように、Mgでは、様々なツールを高度に連携することができ、SysMLv2とPythonを中心として様々な設計作業を自動化することができます

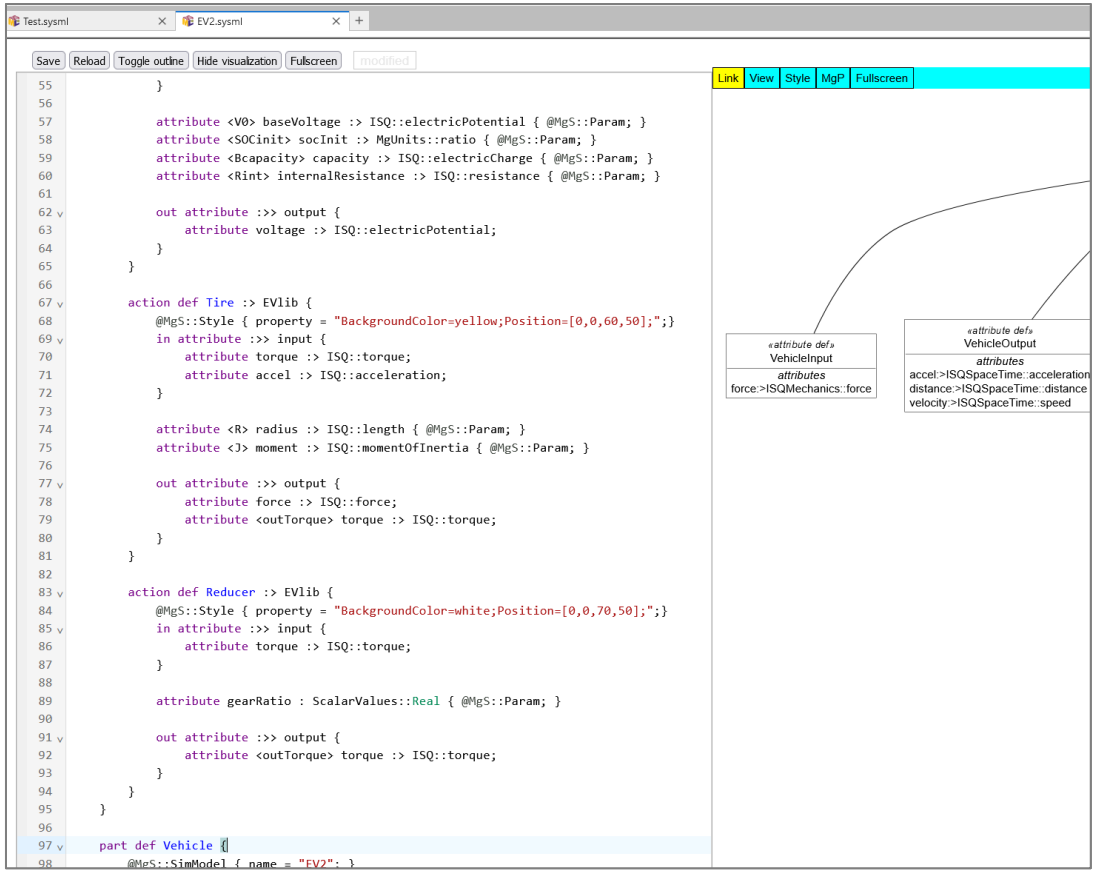
MgのSysMLv2エディタのうれしさ

■SysMLv2ではテキスト表記とグラフィカル表記の両方が標準化されており、MgのSysMLv2エディタは、テキストでの記入を即座にダイアグラムに変換して確認できます

これによって、どちらの良さも引き出してモデリングを行うことができます

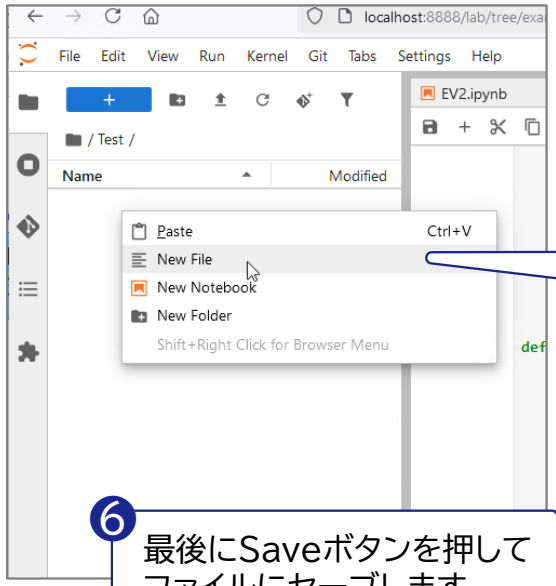
- 名前や数値や構造などはテキストでの入力の効率が勝ります
- 一方で、アクションやステートのつながりなどはダイアグラムで確認することで、素早く状況を把握することができます

また、Mgは複数ファイルもAutoloadingで効率良く取り扱うことができます
これは大規模プロジェクトを扱う際の効率性に大きく貢献します



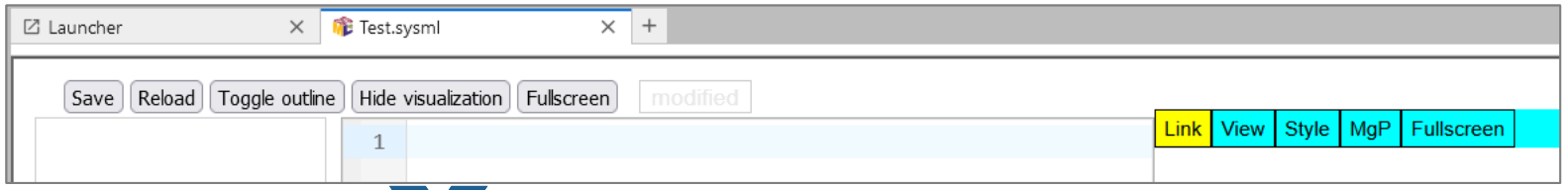
SysMLv2編集の基本（1: パッケージ、コメント, Part, Attribute）

ここで、SysMLv2について基本的な解説を行います。すでに見たようにSysMLv2は、グラフィカル表記だけでなく、テキスト表記も定義されており、柔軟性や開発効率性を大幅に向上させています

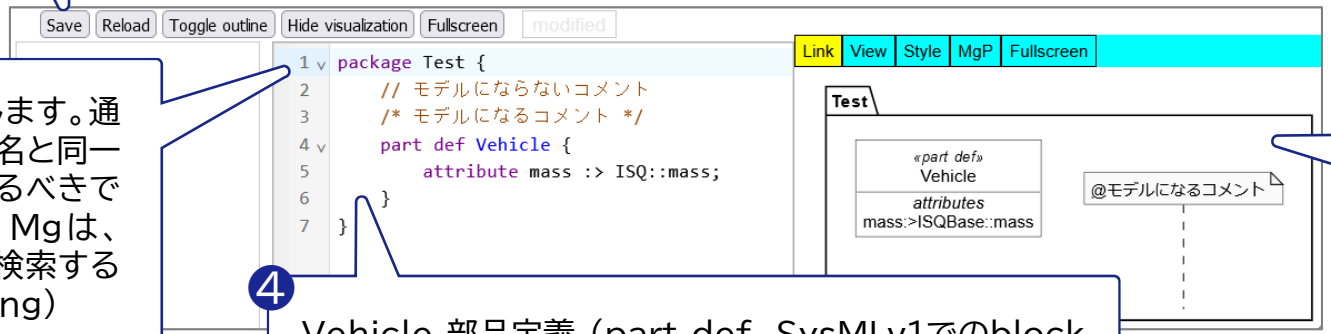


1 JupyterLab上でFileを作成するには左のペインを右クリックして、New Fileを選択すると、ファイルが出来上がるため、Test.sysmlと名前を変更します

2 Test.sysmlを開くとエディタが起動します



6 最後にSaveボタンを押してファイルにセーブします

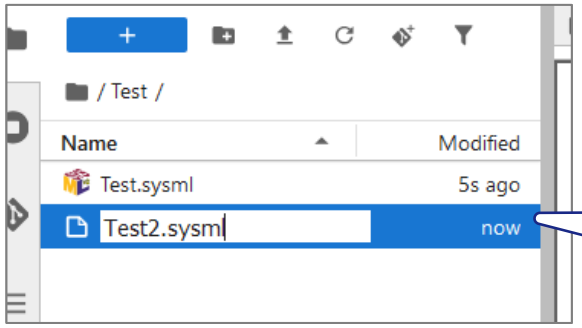


3 基本的なSysMLv2を入力します。通常、SysMLv2では、ファイル名と同一の名前のpackageを記述するべきです。このようにすることで、Mgは、パッケージ名からファイルを検索することができます（Autoloading）

4 Vehicle 部品定義（part def, SysMLv1でのblockに相当）を記述しています。
この例では、mass属性(attribute)も記述されており、100kgと値が与えられています。このように、SysMLv2では、物理量（ISQ::mass)が記述できます

5 編集内容は、即座に解釈され、MgによってSysMLv2のグラフィカル表記で可視化(Visualize)されます。この例にあるように、モデルにあるコメントのみが表示されますが、モデルにならないコメントはソースにのみ存在するため、表示されません

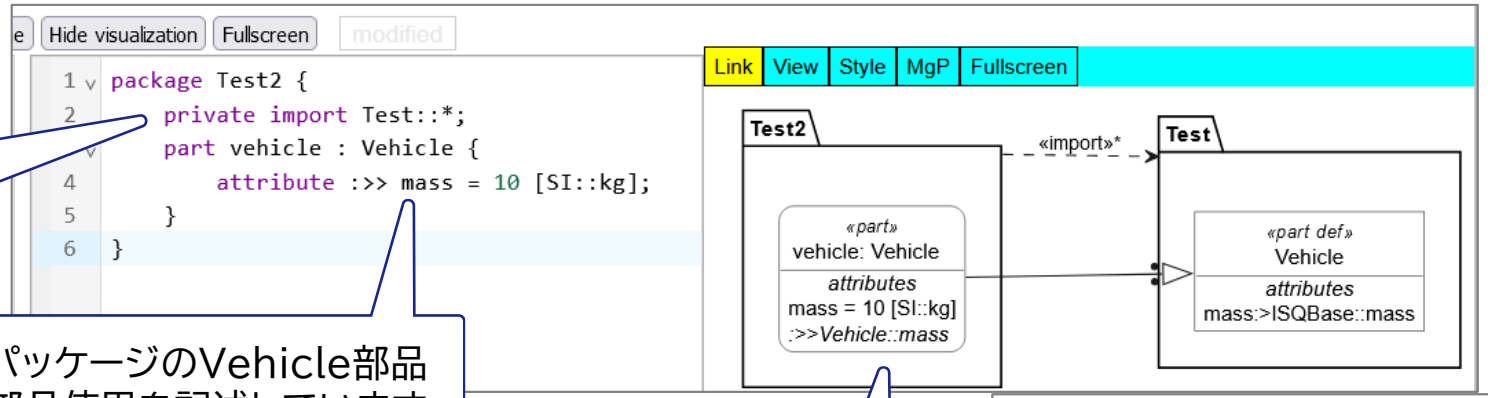
SysMLv2編集の基本（2: Import and Autoload）



次に、先ほどと同様の方法でTest2.sysmlファイルを作ります

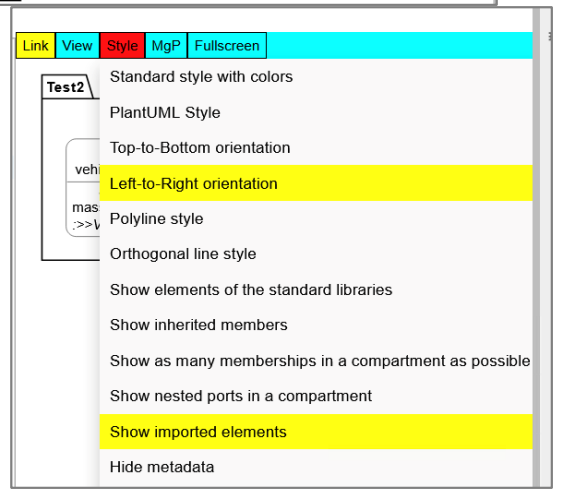
Mgでは複数ファイルを取り扱うためにAutoloadの機能を持っています。これは、ファイルをまたがった参照を自動的に解決するための機構です。

Test2パッケージを作り、Testパッケージをインポートします。これによって、自動的にTestファイルが読み込まれて参照が解決されます



インポートされたTestパッケージのVehicle部品定義を用いてvehicle部品使用を記述しています。ここで、massは10kgに再定義されています。SysMLv2では、単位系も定義されており、SI単位系はSIライブラリに用意されています

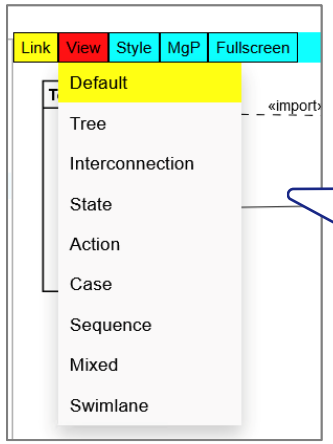
可視化された結果が表示されます。ここでは、インポートされたTestパッケージを表示し、水平方向に可視化しています。このようなスタイル設定は、上のメニューから右図のように行うことができます



Mgでは、このオートロード機能はカーネルごとに取り扱われるます。各カーネルでは、必要なSysMLv2ファイルしか読み込まないため、大規模なプロジェクトについても取り扱えるように設計されています

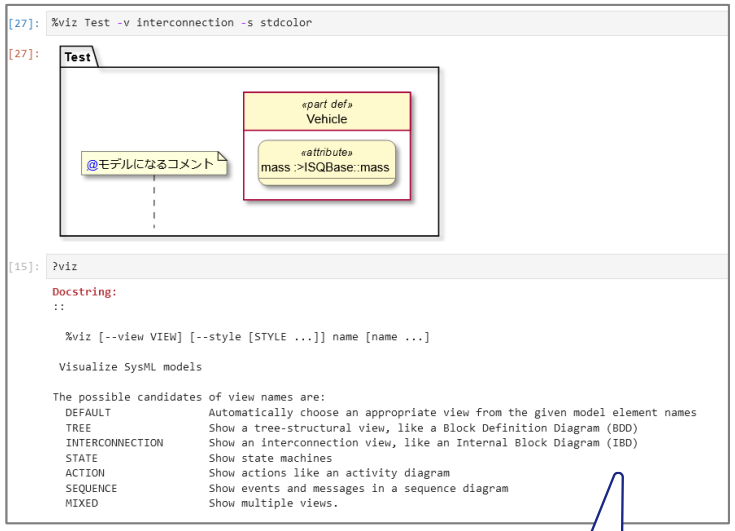
MgにおけるSysMLv2の可視化手法

すでに見たように、Mgは様々な可視化オプションを用意しており、ビューやスタイルはメニューやmagicのオプションで選ぶことができます

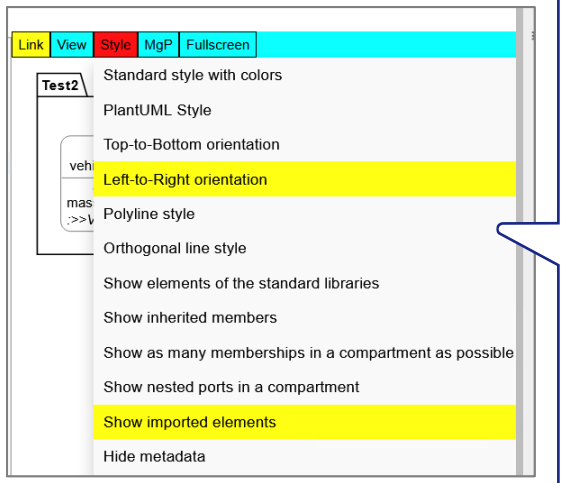


Viewはモデルをどのように可視化するのかを指定します。以下のうちの一つだけを選ぶことができます。

Default: モデルの種類によってビューを自動的に選択します
Tree: 基本的に階層上にモデルを表示します。コンパートメントも活用し、いわゆるSysMLv1で呼ぶBDD(Block Definition Diagram)に似た表示を行います
Interconnection: ノードを入れ子にして相互接続を表示するビューです。いわゆるSysMLv2で呼ぶIBD (Internal Block Diagram)に似た表示を行います
State: ステート図を表示するビューです
Action: アクション図を表示するビューです。SysMLv1でいうアクティビティ図に相当します
Case: ユースケースやアナリシスケースなどのケースを表示するためのビューです
Sequence: シーケンス図を表示するビューです
Mixed: 様々なビューを複合して表示するビューです
Swimlane: (実験中の機能ですが、スイムレーンを表示するビューです)



ノートブック中にコマンドライン(Jupyterでのmagic)で可視化を行うこともできます。ヘルプは、MgPyにおいては、?viz で表示されます。(MgSysMLで)は、%vizでヘルプが表示されます) なお、オプションの指定方法が異なっていることに注意してください。MgPyでは、可視化したい要素名から指定し、オプションはあとに配置することが推奨されますが、MgSysMLでは、オプションは前に配置されます



Styleは可視化の際の様式を指定します。以下のうちの複数を選ぶことができます。

Standard style with colors: カラーの標準スタイルです。(デフォルトでは白黒です)
PlantUML style : PlantUMLの独特な表示形式です
Top-to-Bottom orientation: 上から下に図形を配置します
Left-to-Right orientation: 左から右に図形を配置します
Polyline style: エッジを折れ線で表示します
Orthogonal line style: エッジを直角に折れ曲がる線で表示します
Show elements of the standard libraries: 標準ライブラリのモデル要素も表示します。通常は使いませんが、標準ライブラリの構造を知りたいときは有用です。
Show inherited members: 継承された要素も表示します。差分的に書かれたモデルを検討する際に極めて有用です。
Show as many memberships in a compartment as possible: できるだけ、コンパートメントに要素を表示します。コンパクトに表示する際には有用です
Show nested ports in a compartment: ネストされた構造もコンパートメントに表示します
Show imported elements: インポートされている要素も表示します。複数パッケージにまたがるモデルを表示する際に有用です
Hide metadata: メタデータを隠します。メタデータが煩雑に表示される場合に有用です

効率的な可視化はモデリングにおいて重要であるため、Mgでは今後も可視化手法について強化してく予定です。一方で、MgPによる可視化手法もありますが、それはMgPでの説明に譲ります

MgXのうれしさ

■MgXは、Excel上でSysMLv2モデルを操作することができます

- Excelの数式を用いて、柔軟にSysMLv2モデルを取得できます
- 取得したモデルを表形式に配置することも簡単にできます

これによって、Excelでありながら、オリジナルの情報はモデルで一元管理することができるようになります

- Excelだけで作業をした場合には、情報が様々なExcelファイルに散らばってしまい、トレーサビリティに重大な問題を引き起こすばかりでなく、他のツールや情報システムとの連携も困難になってしまいます

また、Excel上で動作するため、Excelのあらゆる機能を使うことができます

- 他のMBSEツールでも、Excelとのインポートやエキスポート機能が用意されていますが、MgXは、Excel上でモデルを直接操作することができます
- また他のMBSEツールでも表形式のユーザーインターフェースが用意されていることがしばしばありますが、Excelと同等の柔軟な処理ができるわけではありません

さらに、Mgは簡単でありながら強力なクエリ機能を持っており、モデルを簡単に参照することができ、これも生産性に大きく寄与します
なお、近い将来に、モデルの編集も可能になる予定です

MgXの使い方

現状ではMgXは基本的にはExcelにおける数式拡張として設計されており、それを土台として、表を構築するための機能を用意しています

MgXでは表の一行目に数式を記述し、参照を工夫することで表を構成することができます

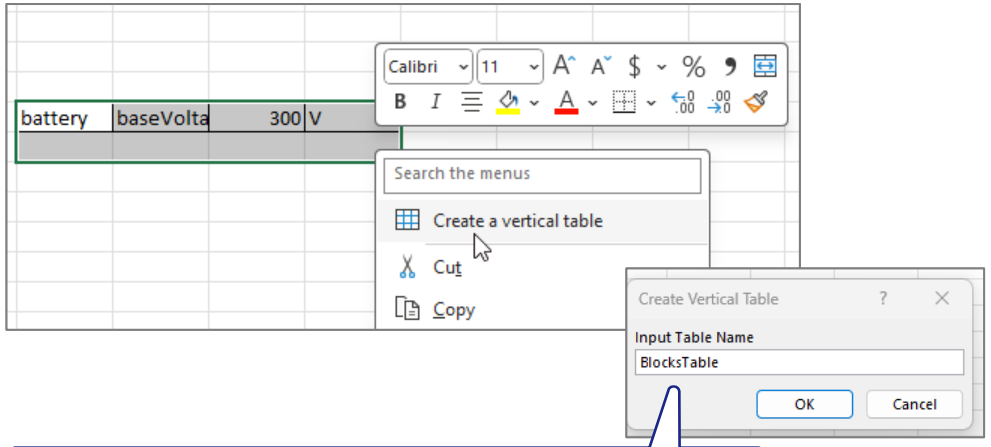
	D	E	F	G
6	SimBlock	Attribute	Value	Unit
7	=MgQ("EV2.baseVehicle.^*.#{ActionUsage}")	=MgQ(".*.#{^AttributeUsage}", D7)	=MgQ(".value()", E7)	=MgQ(".unit()", E7,,"")

EV2パッケージの下にあるbaseVehicle部品が持つ、継承分も含めたすべてのActionUsageを取得

左セル(D7, ActionUsageになるはず)の要素が持つ属性(AttributeUsage)を取得

左セル(E7,属性)の値を取得

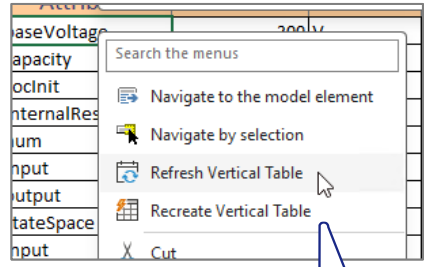
E7セルの属性の単位を取得



これらの数式をセルに記述して、選択し、右クリックメニューからCreate a vertical tableを選択し、テーブル名を入力すると、表を作ることができます

構築された表

SimBlock	Attribute	Value	Unit
battery	baseVoltage	300	V
	capacity	70	A·h
	socInit	0.8	
	internalResistance	1.8	Ω
voltage	num	1	
vehicleBehavior	input	0	
	output	0	
	stateSpace	0	
powerTrain	input	0	
	output	0	



表を更新するには右クリックして、Refresh Vertical Tableを選択します。モデル要素が追加・削除された場合には、Recreate Vertical Tableを行うと、表を再構築することができます

MgXによるExcelからのSysMLモデル編集機能

機能強化

- 非公開バージョンのMgでは、Excelから柔軟にSysMLモデルを編集することができます
- カスタマイズは柔軟にできるようになっており、Excelの編集を即座にSysMLに反映させることができます
  さらに、Pythonによって、複雑なExcelとSysML間の変換処理も行うこともできます
- 柔軟な処理が可能なSysMLv2設定及びPython用のAPIを用意しています
  詳細については、別途お問い合わせください

Excel

	System			Subsystem	
Name	Input	Output	Name	Input	
integrated thermal control	powertrain temp				
	battery temp				
	cabin temp				
	ambient temperature				
	newInput				
	required powera				
		air flow			
		target cabin temp			
		target battery temp			
			battery temp control	required power	
				battery temp	
					target heattransfer

入力がSysMLに反映

SysMLの内容を下に更新

MgX / SysML v2

```
1  package MgXEdit1 {
2      private import MgS::SimSubsystem;
3      private import Mg::MgView;
4      private import ScalarValues::*;
5
6      part sampleSystem {
7          @MgS::SimModel {name = "MgXEdit1";}
8          perform action 'integrated thermal control' : S
9              in :>> input {
                   attribute 'powertrain temp' : Integer;
                   attribute 'battery temp' : Integer;
                   attribute 'cabin temp' : Integer;
                   attribute 'ambient temperature' : Integ
                   attribute 'newInput' : Integer;
                   attribute 'required powera' : Integer;
16             }
               out :>> output {
                   attribute 'air flow' : Real;
                   attribute 'target cabin temp' : Real;
                   attribute 'target battery temp' : Real;
               }

23             /* Subsystems here */
24
25             action 'battery temp control' : SimSubsyste
26                 in :>> input {
```

Link | View | Style | MgP | Fullscreen

«part»
sampleSystem
metadata
SimModel
name = "MgXEdit1"

«perform action»
integrated thermal control: SimSubsystem
parameters
in input ~~StateSpaceDynamics::input~~
out output ~~StateSpaceDynamics::output~~

«action»
battery temp control: SimSubsystem
parameters
in input ~~StateSpaceDynamics::input~~
out output ~~StateSpaceDynamics::output~~

Mgにおけるクエリ（Dot Query）

■ MgX, MgP, MgPyでは、モデルを検索するために共通のクエリ言語（Dot Query）が用意されています
これは、`.’（ドット）で区切られたXPathに似たパス言語で、簡単にモデルを検索することができるように工夫されています

- **文字列:** モデル要素名、モデル要素短縮名（short name）にマッチします
- ***:** すべての直接存在する要素にマッチします
- ****:** 子孫を含めたすべての存在する要素にマッチします
- **^*:** 継承された要素にもマッチします
- **#{型名}:** 指定された形にマッチします
- **value():** 要素に与えられた値を取得します
- **text():** 要素に対応するテキストを取得します
- **eval():** 要素が評価して結果を取得します

例:

- ****.vehicle.*.#{Part}:**
  vehicleという名前の要素が直接所有する部品（Part）を検索します
- ****.vehicle.*.#{AttributeUsage}.value():**
  vehicleの持つ属性の値を検索します

MgPのうれしさ

■MgPは、PowerPoint上でモデルからプレゼンテーションを構築することができます

モデルを雛形(ステンシル)をもとに、PowerPointシェイプへと変換します

- 変換されたシェイプはモデルと関連付けられています
- すでにあるPowerPointシェイプをモデルと結びつけることもできます
- PowerPointシェイプから対応するモデルにたどることや、その逆(モデルからシェイプ)もできます

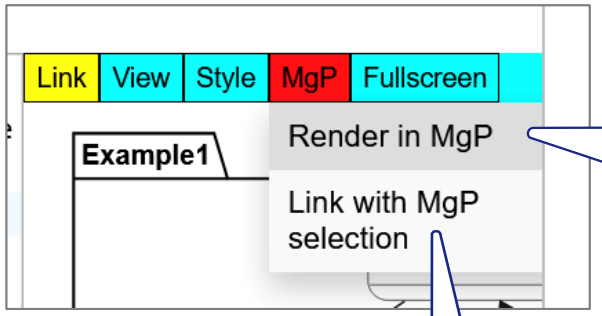
クエリを元に必要なモデル要素だけをPowerPointシェイプへと変換することもできます

これによって、PowerPointのプレゼンテーションを効率よく構築できるうえ、さらにモデルとのトレーサビリティを保つことができます

- 様々な設計ツールを使っていたとしても、結局はお客様や社内に対してもコミュニケーションのためにPowerPointが使われることが頻繁に起こります。その時、たくさんの整合性がなくトレーサブルでないプレゼンテーションファイルが乱立することが頻繁に起きます
- MgPは、SysMLv2とPowerPointを統合することで、このような問題を解決します

近い将来には、PowerPointからモデルの編集や、モデルとPowerPointとの差分や整合性確認もできるようにする予定です

MgP（1）モデルをプレゼンテーションに変換します

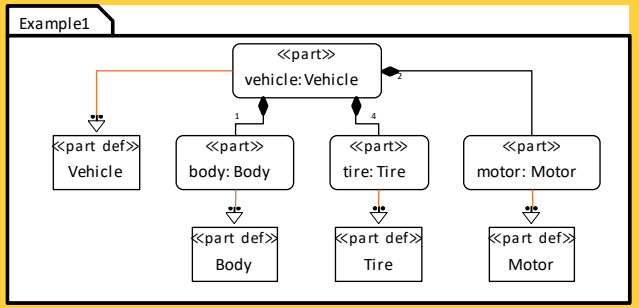


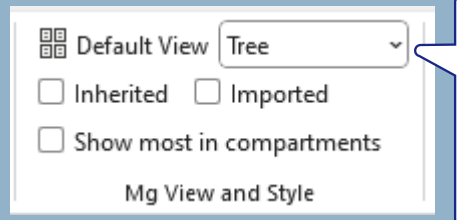
エディタで選択されている要素をMgPで描画します

このメニューを選択すると、Mgエディタ上での現在のカーソル位置に対応するPowerPointシェイプが選択されます

Mgエディタ側でのビューやスタイルの指定はできるだけ反映されて可視化されます
（まだ、すべてのビューやスタイルがサポートされているわけではありません）

PowerPoint
シェイプに変換



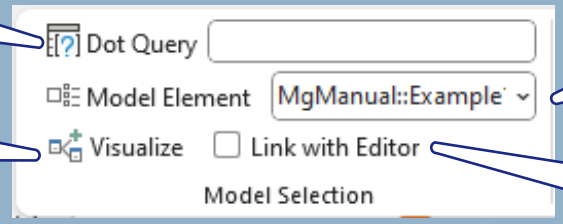


ビューとスタイルはPowerPointのリボンから選択できます

MgP View and Style リボン

Dot Query を入力すると、クエリ対象のモデル要素だけがPowerPointシェイプに変換されます

このボタンで、現在選択されているモデル要素をPowerPointシェイプに変換します



MgP Model Selection リボン

現在選択されているモデル要素が表示されます

このチェックボックスをONにすると、MgPで選択されたシェイプに対応するモデル要素にMgエディタのカーソル位置が移動します

MgP（2）シェイプの利用とステンシル

example > MgP > simpleStencilフォルダから、MgPSimpleStencil.ipynbノートブックを開きます

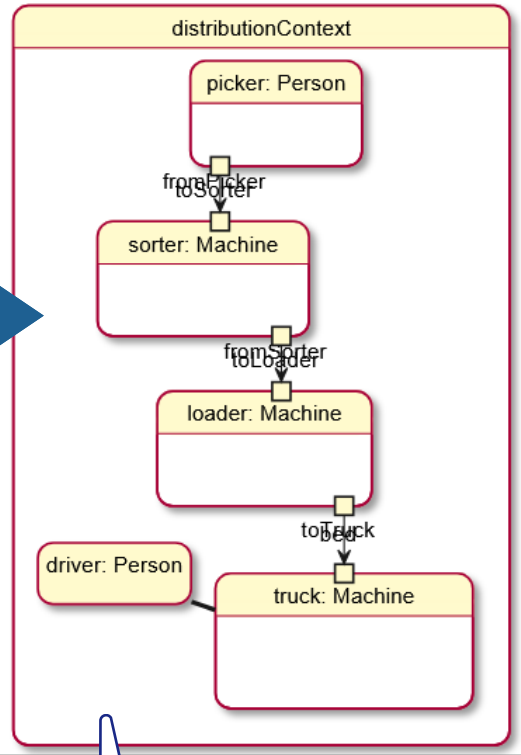
MgPはステンシルという雛形を用いてPowerPointシェイプを描画します。このステンシルをカスタマイズすることで、柔軟にプレゼンテーションを作ることができます

MgP Stencil Example

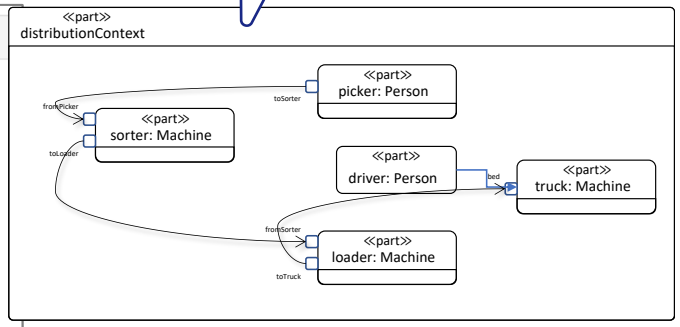
Write SysML model

```
[1]: %%sysml
package MgPSimpleStencil {
    part def Person;
    part def Machine;
    part distributionContext {
        part driver[1]: Person;
        connect driver to truck;
        part truck: Machine {
            port bed;
        }
        flow from loader.toTruck to truck.bed;
        part loader[2]: Machine {
            port fromSorter;
            port toTruck;
        }
        flow from sorter.toLoader to loader.fromSorter;
        part sorter[1]: Machine {
            port fromPicker;
            port toLoader;
        }
        flow from picker.toSorter to sorter.fromPicker;
        part picker[3]: Person {
            port toSorter;
        }
    }
}
```

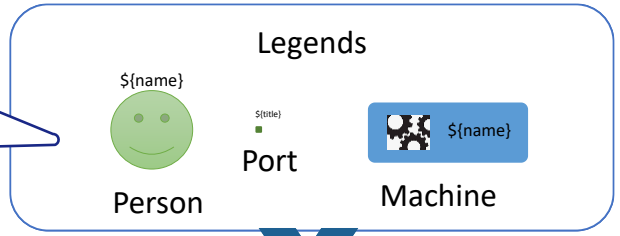
```
[2]: %viz MgPSimpleStencil::distributionContext --view interconnection --style stdcolor
```

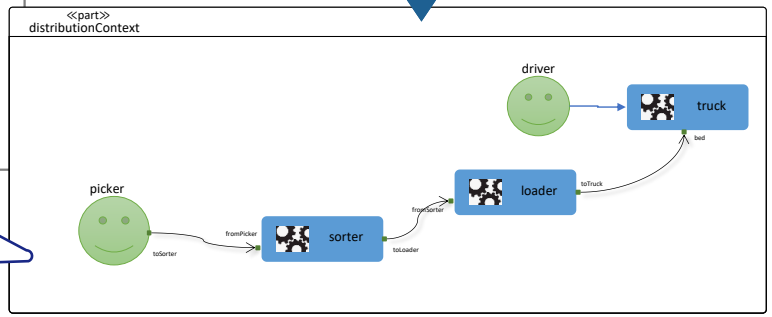


MgPでinterconnection view(相互接続図)で可視化すると、通常は以下のようなPowerPointシェイプを得ます



しかし、今回の例では、以下のようなステンシルがテンプレートに用意されています。これは、Person，Port，Machineにそれぞれ別のシェイプを割り当てています（Dot Queryで柔軟に割当は決定できます）





はじめのセルは、サンプルのSysMLで、配送コンテキストについてモデリングされています

これを%vizで可視化すると以上のような図を得ます

その結果として、右図のような状況にあった図を作ることができます。このようにMgPでは柔軟にプレゼンテーションを作ることができるように設計されています（右の図は一部手動でレイアウトをしています）

MgSのうれしさ

■MgSは、シミュレーション環境、特にMATLAB/Simulinkとの連携機能を提供します

SysMLv2の状態空間表現モデル（State Space Representation）をもとに、Simulinkモデルを生成したり、対応付を行うことができます

- MgSでは、SysMLv2やSimulinkの一部と整合性を持って対応付けることが可能なため、既存のSimulink資産を活用できるように考慮されています

また、SysMLv2で指定された分析条件でSimulinkを実行して、結果をSysMLv2モデルに結びつけることができます

結果は、MgXやMgPyから取得することができます

これによって、MBSEとMBDを融合させて分析や検証ができるようになります

- SysMLv2による上位階層のモデルをSimulinkによる実装レベルのモデルを柔軟に連携させることで、素早い検証によるフロントローディングを行うことができ、それによって生産性を向上させることができます
- また、膨大な量になりがちな条件の異なる様々なSimulinkファイルをSysMLv2によって大幅に整理することができるようにもなります
- さらに、Simulinkの実行もJupyterやPythonで自動化することで、作業効率を向上させることもできます

MgSと状態空間表現モデル

MgSは、SysMLの分析および状態空間表現モデルを元にSimulinkを生成・実行し、結果を取得することができます。これによって、シームレスにMBSE/MBDを連携させることを目的としています。

u (入力) → «action» 状態空間表現 (State Space Representation) → y (出力)

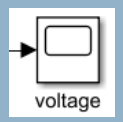
x (状態)

$$\dot{x} = f(x,u)$$
$$y = g(x,u)$$

状態空間表現とは、入力・出力・状態およびその変化が明示的に記述された振る舞いモデルです。SysMLv2では、状態空間表現モデルライブラリ(StateSpaceRepresentation)で記述することができます。Simulinkのブロックはこの状態空間表現モデルで形式化することができます。（なお、弊社がSysMLv2での状態空間表現モデルライブラリに貢献いたしました）

MgSは、このようにSysMLv2の強力な拡張機能を活用して、シームレスにSimulinkとの連携を可能にしており、基本的にはほぼすべての種類のSimulinkブロックを生成して、対応付けることができます。例えば、

```
action voltage : MgS::Scope {
    :>> num = 1;
}
```



だけでスコープを生成できます

SysMLv2で記述された状態空間モデルの例

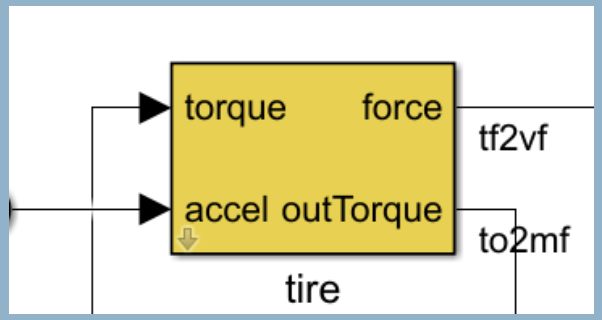
```
action def Tire :> EVlib {
    @MgS::Style { property = "BackgroundColor=yellow;Position=[0,0,60,50];";}
    in attribute :>> input {
        attribute torque :> ISQ::torque;
        attribute accel :> ISQ::acceleration;
    }

    attribute <R> radius :> ISQ::length { @MgS::Param; }
    attribute <J> moment :> ISQ::momentOfInertia { @MgS::Param; }

    out attribute :>> output {
        attribute force :> ISQ::force;
        attribute <outTorque> torque :> ISQ::torque;
    }
}
```

Simulink上で表現されるスタイルをメタデータで記述することができます

トルクと加速度の入力を表します

パラメータ(半径と慣性モーメント)を表します

力とトルクの出力を表します



MgSによって生成されるSimulinkブロック

MgSの機能強化

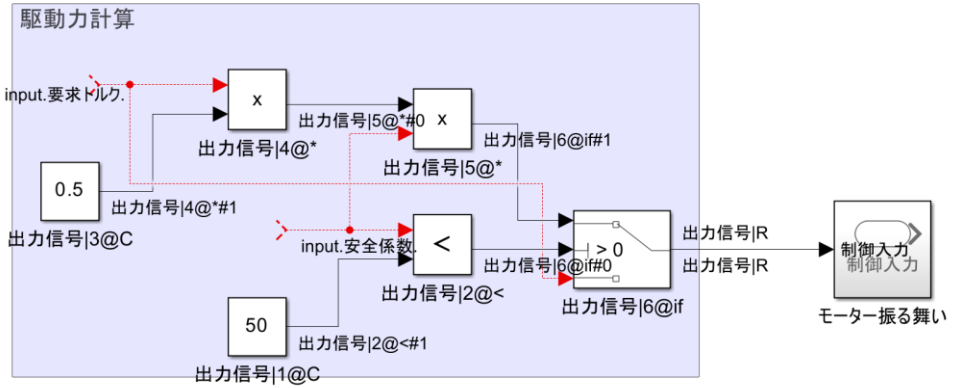
■SysML式（Expression）のサポート

機能強化

・SysMLで記述した式を、Simulinkに変換することができます
・サブシステムの入力を参照することも可能です
また、一時変数のように使える束縛（#simLet）をサポートしています

```
action '駆動力計算' : SimSubsystem {
    in :>> input {
        attribute '要求トルク';
        attribute '安全係数';
    }
    out :>> output {
        attribute '出力信号' = if input.'安全係数' < 50?
                input.'要求トルク' * 0.5 * input.'安全係数'
            else input.'要求トルク';
    }
}

action 'モーター振る舞い' : SimSubsystem {
    in :>> input {
        attribute '制御入力' = '駆動力計算'.output.'出力信号';
    }
}
```

これによって、Simulinkでは記述しにくい、
複雑になりがちな条件式を効率よく開発することができます



Reverse機能（逆変換: Simulink->SysML）の部分的サポート

- Simulinkでの接続線の編集をSysMLに反映させることができます
- 今後、より多くのSimulinkでの編集に対応する予定です

MgSでの記述方法の基本（1）— サブシステム

SCSK

機能強化

すでに述べたようにMgSは、SSR（状態空間表現）によるモデルをSimulinkに対応付けることができます

これを効率よく記述するためのモデルライブラリがMgS.sysmlによって提供されています

- サブシステムを記述するには、MgS::SimSubsystem 定義を用います
- MgS::SimSubsystemは、SSRであるContinuousStateSpaceDynamicsを継承しており、SSRの記述方法に沿って、サブシステムを記述します
- サブシステムには、入力、出力、内部の構造を記述します。MgSは、この内容をもとにSimulinkのサブシステムを生成します

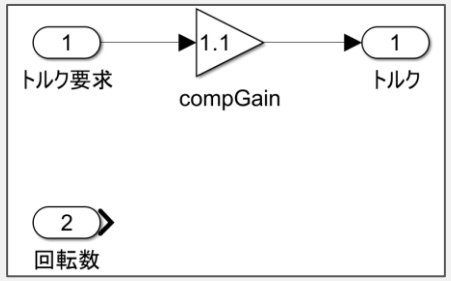
サブシステムの書式

```
action サブシステム名 : MgS::SimSubsystem {
    in :>> input {
        サブシステム入力1;
        サブシステム入力1;
        ...
    }
    out :>> output {
        サブシステム出力1;
        サブシステム出力2;
        ...
    }

    サブシステム内部の記述

}
```

使用例

```
action torqueGenerator : MgS::SimSubsystem {
    in :>> input {
        attribute ‘トルク要求’;
        attribute ’回転数’;
    }
    out :>> output {
        attribute ‘トルク’;
    }

    bind ‘トルク要求’.input = compGain.input.value;
    action compGain : MgS::Gain {
        :>> gain = 1.1;
    }
    bind compGain.output.value = output.’トルク’;
}
```

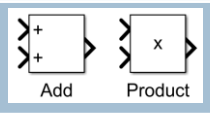
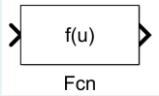
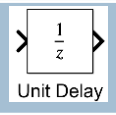
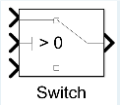
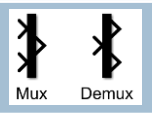
生成されるSimulinkモデル



MgSでの記述方法の基本（2）— 様々なSimulinkブロック

機能強化

MgSは、Simulinkの標準ライブラリのブロックを用いることができ、代表的なブロックを記述するためのモデルライブラリがMgS.sysmlによって提供されています
ここで示したブロックは代表例で、事実上、どの種類のSimulinkブロックもMgS::SimBuiltInBlock定義を継承することで利用することができます

MgSでのライブラリ（MgS::をつけること）	Simulink ブロック	説 明
Scope		結果を観測する代表的なブロックです。 input.i1..の入力を持ちます
Gain		定数倍の演算を行います 制御記述に非常によく用いられるブロックです
Add, Product		加減乗除に用いられます。Addの場合にはsignsで符号を, Productの場合にはopsで演算子を設定できます。
Fcn		exprに文字列で数式を記述することができます。ただし、入出力はおのおの一つだけになります
UnitDelay (Zinv)		1サンプルだけ出力を遅延させます 状態を記述するために多用されます
Switch		条件によって2つの入力を切り替えるifのようなブロックです
Mux, Demux		複数入力をベクトルにまとめる(Mux)、ベクトルを複数入力に分割する(Demux)ブロックです

MgSでの記述方法の基本（3）— 分析の記述

MgSで指定する対象はSysMLv2のanalysis usage（分析使用）です

機能強化

- 分析には、分析対象（subject）とパラメーターを指定できます
- 最低限subjectを指定すれば動作させることができます
- @MgS::Paramを属性に割り当てることで、モデルワークスペースに値を反映させることができます
- @MgS::SimRetを出力パラメーター（out）に指定することで、シミュレーション結果を取得できます
- さらに、MgS::Retrievalを指定することで、時系列のどのデータを取得するかを指定できます
  - FINAL（ディフォルト）: 終了値，AVERAGE: 時間平均値，MAX: 最大値，MIN:最小値，FULL:すべての値
- これにくわえて、MgSでは、分析条件（simCond）を指定できるように、MgS::SimAnalysis定義を用意しています
- 現状では、simCondには、シミュレーション終了時刻であるtoTimeのみが指定できますが、今後要望に応じて設定できる項目を増やす予定です

分析の書式

```
analysis 分析名 : MgS::SimAnalysis {
    :>> simCond = {
        :>> toTime = シミュレーション時間(秒);
    }
    subject 対象名 ::> 対象への参照;
    attribute 属性 = 属性値 {@MgS::Param; }
    …
    out 観測対象名 = 観測対象のポートへの参照 {
        @MgS::SimRet {
            ret = MgS::Retrieval::
                FINAL | AVERAGE| MAX | MIN| FULL;
        }
    }
    …
}
```

使用例

```
action maxSpeedAnalysis : MgS::SimAnalysis {
    :>> simCond = {
        :>> toTime = 500; // 500秒までシミュレーション
    }
    subject ev2Subject ::> ev2; // ev2を分析対象
    attribute mass = 2000[SI::kg]{ @MgS::Param; }
    out maxSpeed = ev2Subject.vehicle.velocity {
        //最高速度を取得します
        ret = MgS::Retrieval::MAX;
    }
}
```


MgSのSysML式（Expression）の変換

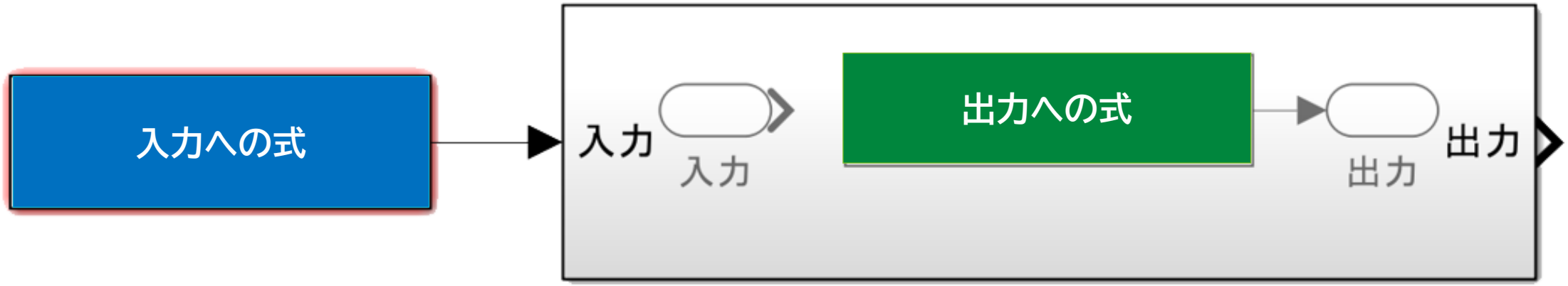
機能強化

MgSにおける、サブシステムの入出力に式を記述することができます

例

```
action 'サブシステム' : SimSubsystem {
    in :>> input {
        attribute '入力' = 「入力への式」
    }
    out :>> output {
        attribute '出力' = 「出力への式」
    }
}
```

- 「入力への式」は、サブシステムの外部に配置され、式の出力が、サブシステムへの入力になります
- 「出力への式」は、サブシステムの内部に配置され、式の出力が、サブシステムからの出力になります



一時的な束縛(#simLet)

機能強化

#simLetをattribute(属性)の前に記述することで、Simulink側で再利用可能な値として利用することができます

これは、実際には、Signal Specificationブロックに変換されます

なお、Signal Specificationブロックは、実際には何もしない(identity)ブロックです。

ただし、検証用の仕様を加えることができます

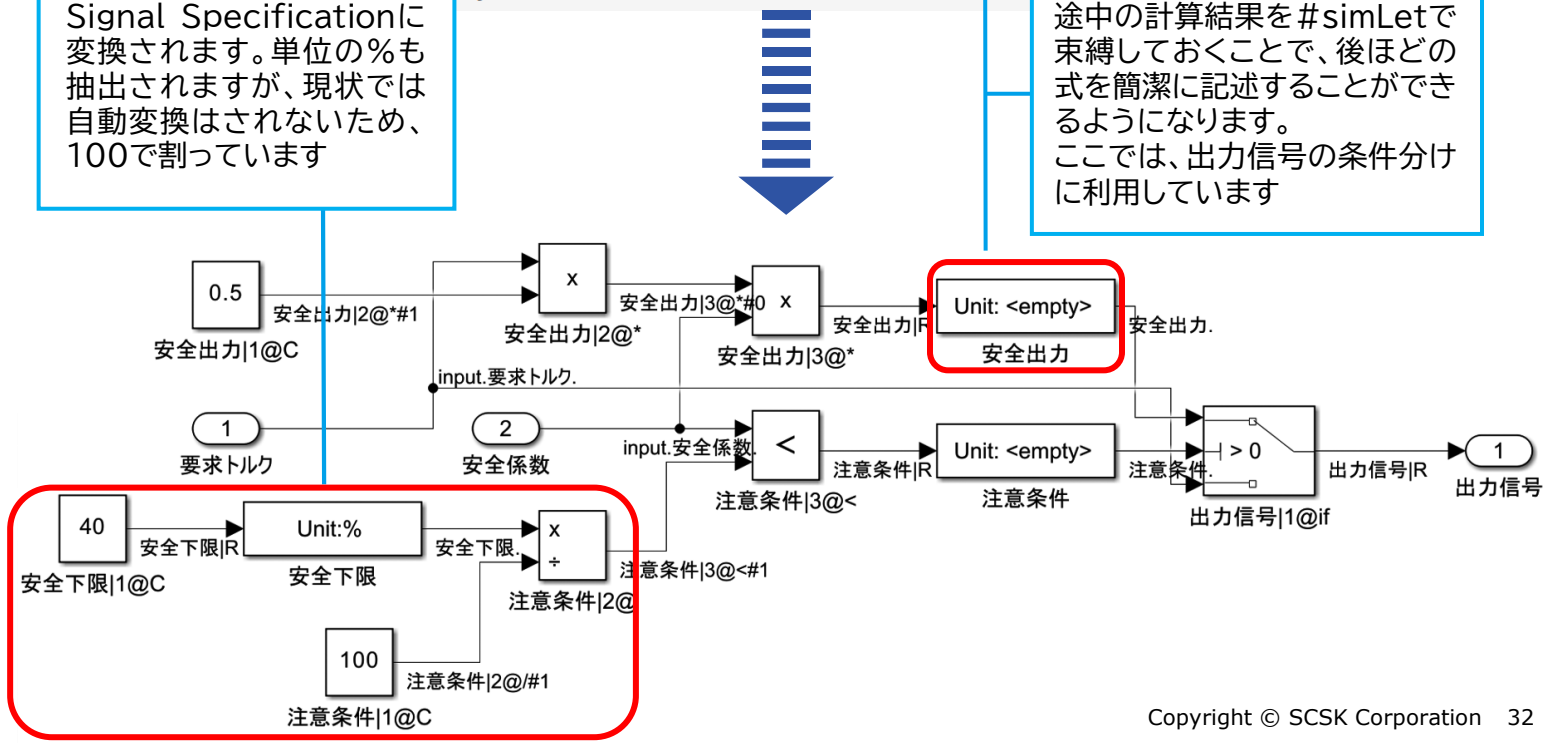
MgSでは、単位が記述されていた場合には、その名前をunitとして設定します

今後、より多くの仕様記述をサポートすることを検討してます

```
action '駆動力計算' : SimSubsystem {
    in :>> input {
        attribute '要求トルク';
        attribute '安全係数';
    }

    #simLet attribute '安全下限' = 40 ['%'];
    #simLet attribute '注意条件' = input.'安全係数' < '安全下限' / 100;
    #simLet attribute '安全出力' = input.'要求トルク' * 0.5 * input.'安全係数';

    out :>> output {
        attribute '出力信号' = if '注意条件' ? '安全出力' else input.'要求トルク';
    }
}
```

Simulinkから参照可能なSignal Specificationに変換されます。単位の%も抽出されますが、現状では自動変換はされないため、100で割っています

途中の計算結果を#simLetで束縛しておくことで、後ほどの式を簡潔に記述することができるようになります。
ここでは、出力信号の条件分けに利用しています



サポートしている演算子

■MgSは現状で以下のSysML演算子をサポートしています

機能強化

種 類	SysML演算子	Simulinkブロック	説 明
四則演算	+, -, *, /	Add, Subtract, Product, Divide	通常の四則演算です。SysML演算子は常に二項演算子ですので、ブロックも2入力のみになります
比較演算子	==, !=, <, >, <=, >=	==, ~=, <, >, <=, >=	比較用のブロックです。基本的には同等のSimulinkの関係演算に対応します
論理演算子	&, |, xor, not	AND, OR, XOR, NOT	論理演算も基本的に対応しますが、SysMLのand, orは、Simulinkに短絡評価機能がないため対応しません
if(三項演算子)	if 条件? True式 else False式	Switch	if式は、Switchに対応されます。SimulinkのIfブロックは、制御フローになるため、対応されません。

SysMLの式評価における注意事項

機能強化

01

SysMLの式は、評価可能であれば、事前に評価されてからSimulinkに変換されます

例

#simLet attribute a = 5 + 4;

この場合、aには9がConstantとして割り当てられます

02

現状では簡約化は行われていませんが、将来的に行うことを検討しています。

a + b + c

3入力のAdd, 3 * c

ProductではなくてGainブロック　などの最適化

03

関数呼び出し（Invocation Expression）など、その他の式については、引き続き対応を検討いたします

04

単位変換やチェックは重要であると考えており、今後のSysML式評価の物理量・単位に合わせて、対応を検討する予定です

MgSのreverse（逆変換）機能

機能強化

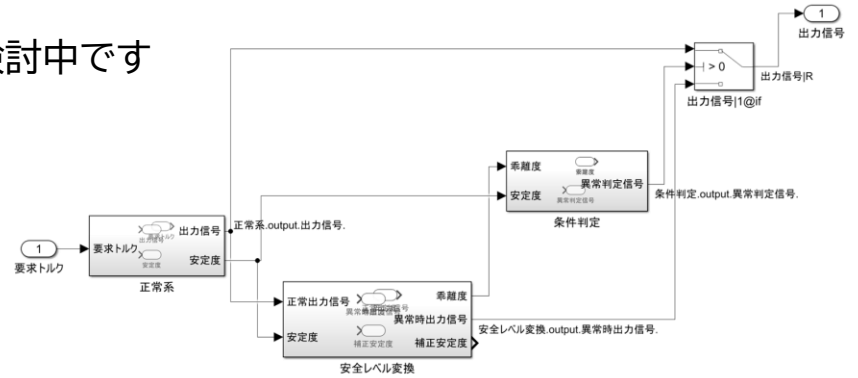
MgSは、SysMLの状態空間モデルをSimulinkに反映させる（reflect）ことで、シミュレーションを行う機構を取っていますが、Simulinkで編集した変更をSysML側に戻したいという要望がございました

MgSのreverse機能は、主にこの要求に答えるために提供されています

- 現状では、サブシステム間の接続のみをSysMLに変更させることができます
- サブシステムなどのブロック、システム=サブシステム間の接続についての対応は検討中です

```
action '駆動力計算' : SimSubsystem {
    in :>> input ...}
    bind input.'要求トルク' = '正常系'.input.'要求トルク';
    action '正常系' : SimSubsystem ...}
    action '安全レベル変換' : SimSubsystem ...}
    action '条件判定' : SimSubsystem ...}
    out :>> output ...}

    flow from '安全レベル変換'.output.'乖離度' to '条件判定'.input.'乖離度';
    flow from '正常系'.output.'出力信号' to '安全レベル変換'.input.'正常出力信号';
    flow from '正常系'.output.'安定度' to '安全レベル変換'.input.'安定度';
    flow from '正常系'.output.'安定度' to '条件判定'.input.'安定度';
}
```

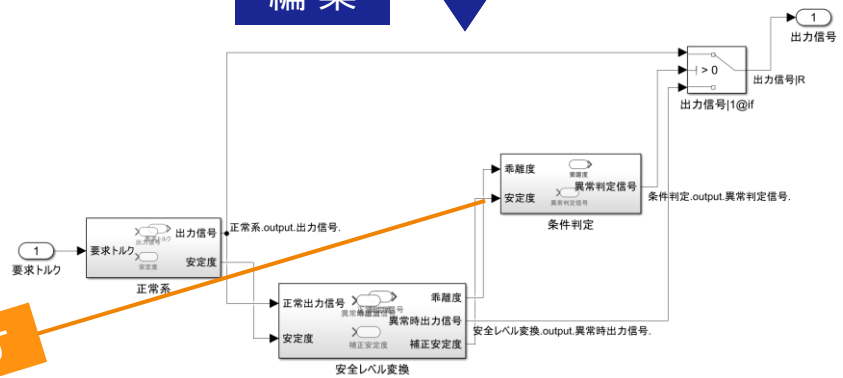
reflect



編 集

reverse

```
action '駆動力計算' : SimSubsystem {
    in :>> input ...}
    bind input.'要求トルク' = '正常系'.input.'要求トルク';
    action '正常系' : SimSubsystem ...}
    action '安全レベル変換' : SimSubsystem ...}
    action '条件判定' : SimSubsystem ...}
    out :>> output ...}

    flow from '安全レベル変換'.output.'乖離度' to '条件判定'.input.'乖離度';
    flow from '正常系'.output.'出力信号' to '安全レベル変換'.input.'正常出力信号';
    flow from '正常系'.output.'安定度' to '安全レベル変換'.input.'安定度';
    flow from '安全レベル変換'.output.'補正安定度' to '条件判定'.input.'安定度';
}
```



編集が反映されます

MgPyのうれしさ

■MgPyはPythonによってSysMLv2およびMgの機能にアクセスすることができる機能です

- SysMLv2のモデルをPythonで読み書きできます
- Mgの処理をPythonで自動化することで、EV2で見たようなアプリを構築することもできます
- 膨大なPythonやJupyterのソフトウェア資産を活用することができます

これによって、MBSE/MBDのプロセス改善に大きく寄与することができます
柔軟かつ開発効率の高いPythonによるMBSE/MBDの自動化が可能になります

- Pythonは、事実上標準的なスクリプト言語になりつつあります。特にAIやデータ解析では、Python + Jupyterが最も重要な開発環境です
- これによって、様々な開発資産や環境とSysMLv2によるMBSEを統合させることができます

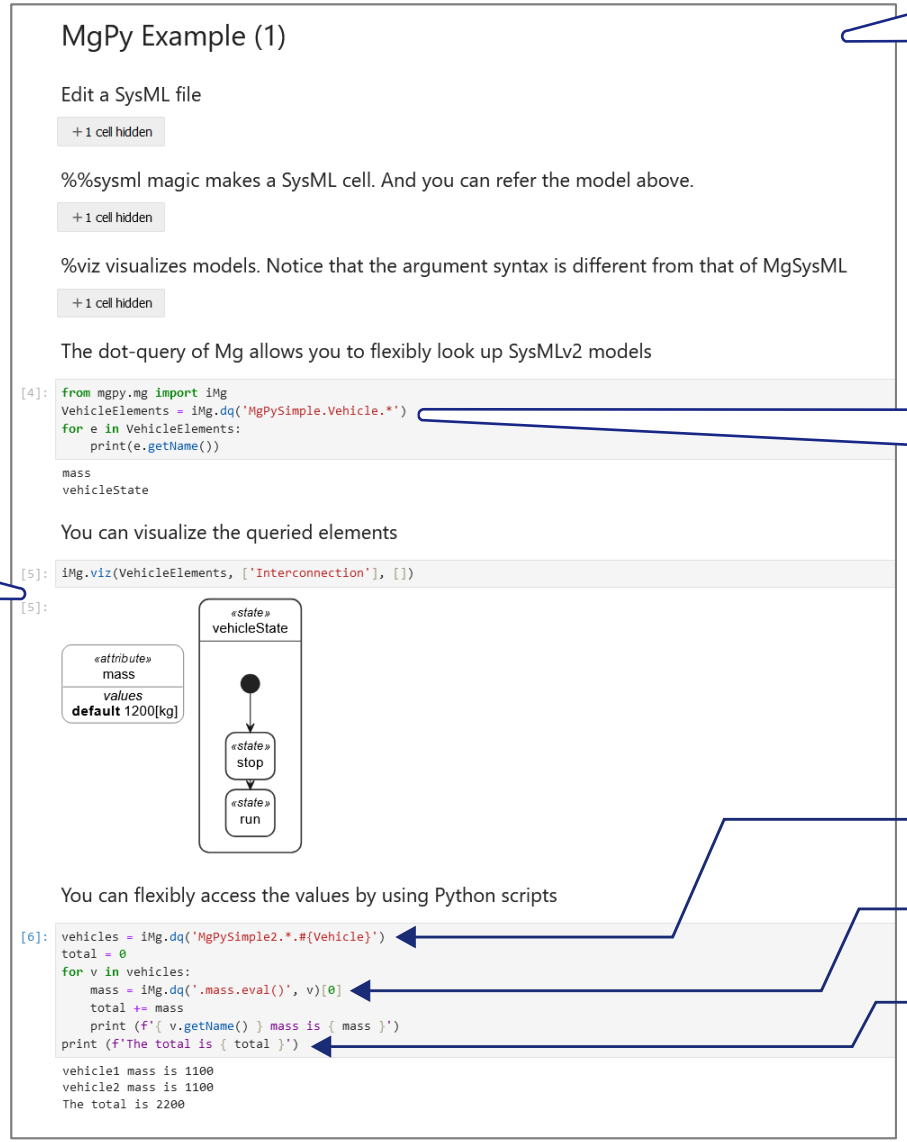
さらに、SimulinkやExcelへのアクセスが柔軟にできるようになっているため、保守が困難になりがちなMATLABやVBAスクリプトを置き換えることもできるようになります

今後も、MgPyによってほぼすべてのMgの機能にはアクセスできるように進めていく予定です
例えば、MgPはまだMgPyによる操作を可能としておりませんが、
順次対応を進めていきたいと考えております

MgPy の使い方

MgPyは、SysMLv2のみならず、Mgのほぼすべての機能をPythonから活用することができる環境です
JupyterLabの柔軟性と組み合わさり、強力な自動化機能を提供します

example > MgPy > simple フォルダから、MgPySimple.ipynbノートブックを開きます



この例にあるように、通常のセルはPythonとして解釈されます。Dot Queryを使うことで、簡単にSysMLv2要素にPythonからアクセスすることができます。この例では、MgPySimpleパッケージにあるVehicle部品定義が持つ要素を取得し、vehicleElements変数に格納して、名前を表示しています

Pythonに格納されたSysMLv2要素を可視化することもPythonによって行うことができます

この例では、Dot Queryを活用して値をPythonで計算しています

1. MgPySimple2パッケージにあるVehicleで定義された部品使用を取得してvehiclesに格納します
2. 各々の車両のmassを取り出して、値を評価(eval)してmass変数に格納します。
3. 計算された合計(total)を表示します

MgPyは、このようにSysMLv2とMgの機能を統合する強力な基盤となっています。ユーザーのみならず、Mgの機能強化にも大きく貢献しています

MgPyの機能強化

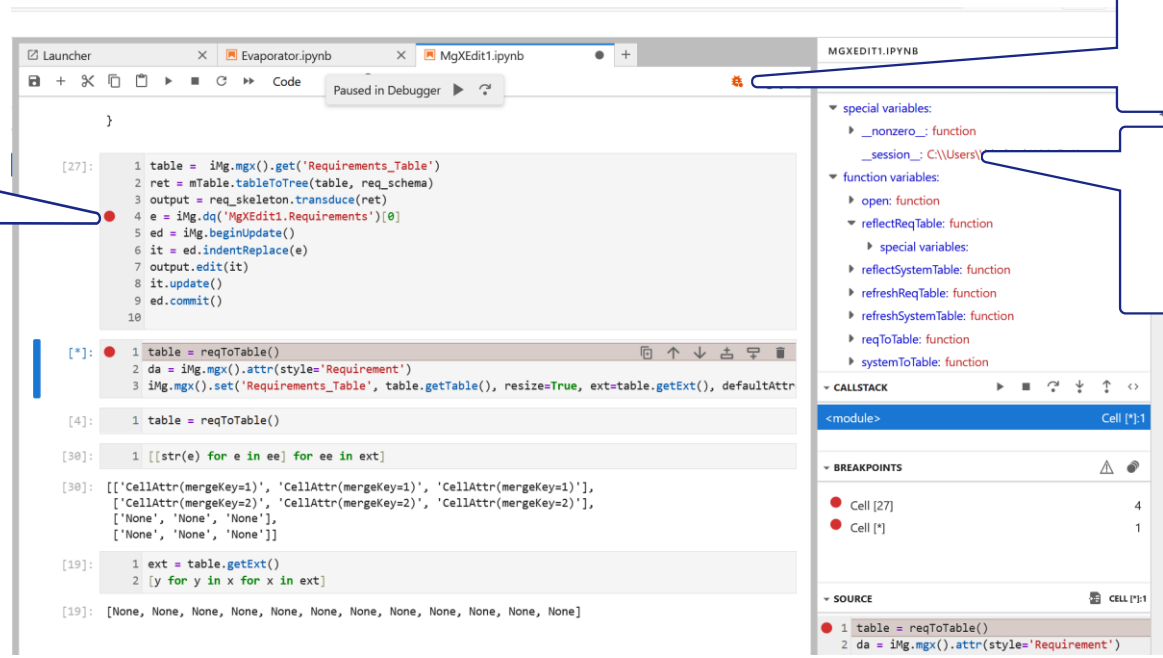
機能強化

■ MgPyは、Mgにおける基本プログラミング環境と言え、より機能性および生産性を向上させるために改善が行われてまいりました

Jupyter上のデバッグ環境を利用することができるようになりました
これによって、MgX, MgS, MgDNGなどのスクリプティングの開発が、より容易に行えるようになりました
また、pysysmlや、Mg自身の機能も、より多くAPIとして提供されており、SysML, Excel, Simulink, DOORS Nextをまたがった自動化の開発を強力にサポートします

- SysMLのリアルタイム編集機能（エディタ上で即座に更新を確認できます）
- 高度なExcelの操作機能（スタイルやドロップダウンリストなども操作可能）
- 高度なSimulink編集（あらゆるブロックや接続を編集できます）

デバッガを用いることで、実行時にブレークポイントを設定できます



虫ボタンを押すことで、デバッガを有効化できます

実行中の変数や、コールスタックなどを確認できます

MgPy-MgS Simulinkアクセス

MgPyの機能を活用することで、MATLABの代わりにPythonを用いてSimulinkを自在に操作するばかりか、SysMLに変換することすらできます

example > EV2フォルダから、MgPyMgSExample.ipynbノートブックを開きます

EV2 Simulinkモデル(EV2.slx)をロードします。
(MATLABが自動的に起動されます)
ev2変数にSimulinkモデルが格納されます

ev2にある最上位のブロックを取り出して名前を表示します

ev2にある最上位のラインを取り出して詳細を表示します

MgPyを活用することで、Simulinkの自動化をMATLABではなくてPythonで行うことができるようになります。
この機能はMBDにおける開発効率化、開発資産化、再利用化に大きく貢献します

MgPy-MgS Example

Load EV2 Simulink model

```
[1]: from mgpy.mg import iMg
     ev2 = iMg.mgs().load('', 'EV2')
```

get all of the blocks in EV2

```
[2]: for blk in ev2.getBlocks():
         print(blk.name())
```

```
Accel
Distance
Velociy
battery
powerTrain
vehicleBehavior
voltage
```

get all of the lines

```
[3]: for line in ev2.getLines():
         print(str(line))
```

```
SLO:EV2/[line]
SLO:EV2/[line]
SLO:EV2/pc2bc[line]
SLO:EV2/va2pa[line]
SLO:EV2/pf2vf[line]
SLO:EV2/bv2pv[line]
SLO:EV2/bv2pv[line]
SLO:EV2/bv2pv[line]
```

List all of the Gain blocks

```
[4]: for g in ev2.find(BlockType='Gain'):
         print(f'{g.name()} -- {g.get("Gain")}')
```

```
Gain -- airFrictionCoefficient
Gain1 -- 1/1000
Gain2 -- 3600/1000
Gain3 -- 1/m
```

Evaluate MATLAB scripts

```
[5]: list(iMg.mgs().eval('3+4'))
```

```
[5]: [7.0]
```

```
[6]: list(iMg.mgs().eval('[2 1 0 0] * [1; 0; 3; 4;]'))
```

```
[6]: [2.0]
```

Convert the Simulink model to SysML

```
[7]: from mgpy.mgs import slSubsystemToSysML
     slSubsystemToSysML(ev2, 'EV2SS.sysml')
```

Edit the generated SysML

```
[8]: %edit EV2SS.sysml
```

```
[8]: Edit EV2SS.sysml
```

Save | Reload | Toggle outline | Hide visualization | Fullscreen | modified

```
1  action 'EV2' : MgS::
2      flow from 'vehic
3      flow from 'power
4      action 'battery'
5      }
6      action 'powerTra
7          in attribute
```

ev2にあるすべてのゲインブロックを検索して詳細を表示します

MATLABスクリプトを評価することもできます。自動化のためにすでに構築されたMATLABを実行する必要があることがしばしばありますが、そのようなものにも対応できるようになっています

ev2にあるSimulinkのサブシステム構造をSysMLv2に変換します(slSubSystemToSysMLはMgSによって提供されているPython関数です)

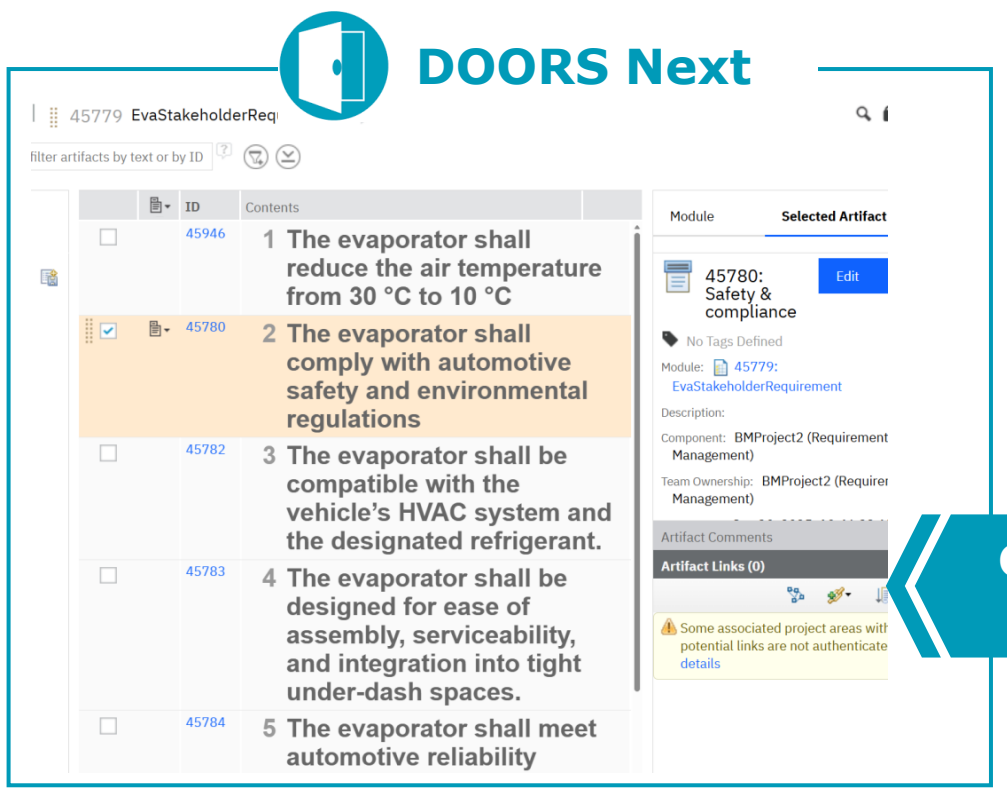
生成されたファイル(EV2SS.sysml)を開いて編集できます

MgDNGによるDOORS Nextとの連携

新機能

■ MgDNGは、DOORS Next（DNG: DOORS Next Generation）との連携を可能にする機能です

- SysMLv2と、DNGのArtifact（成果物）を双方向で変換することができます
  これによって、DNGの要求をSysMLv2や、MgXによってExcelから編集することができるようになります
- DNGのArtifact Typeに対応したSysML v2でモデルライブラリを用意することで、DNGのカスタマイズにも十分に対応できるようになっています
- MgDNGはMgPyベースで構築されており、直接DOORS NextにOSLC/RESTで通信を行います
  このため、カスタマイズや自動化が容易にできるようになっています



MgX + MgDNGによる、ExcelからのDNG編集

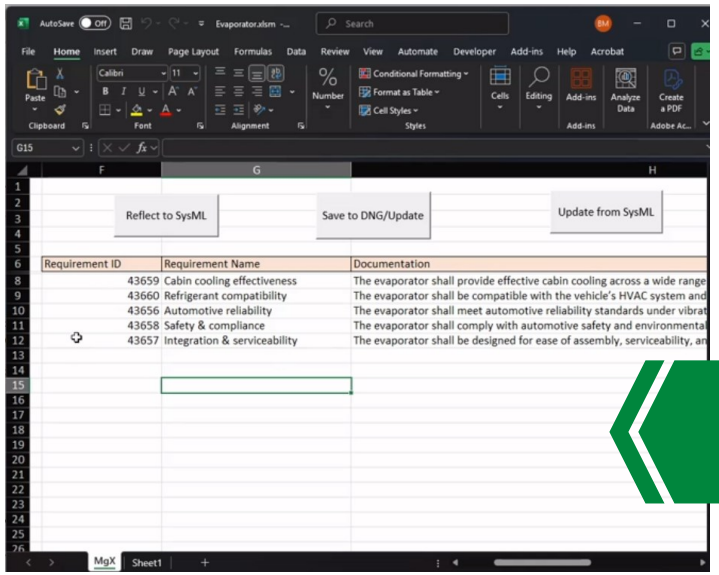
■ MgDNGをMgXを組み合わせることによって、ExcelからのDNG編集が可能になります

新機能

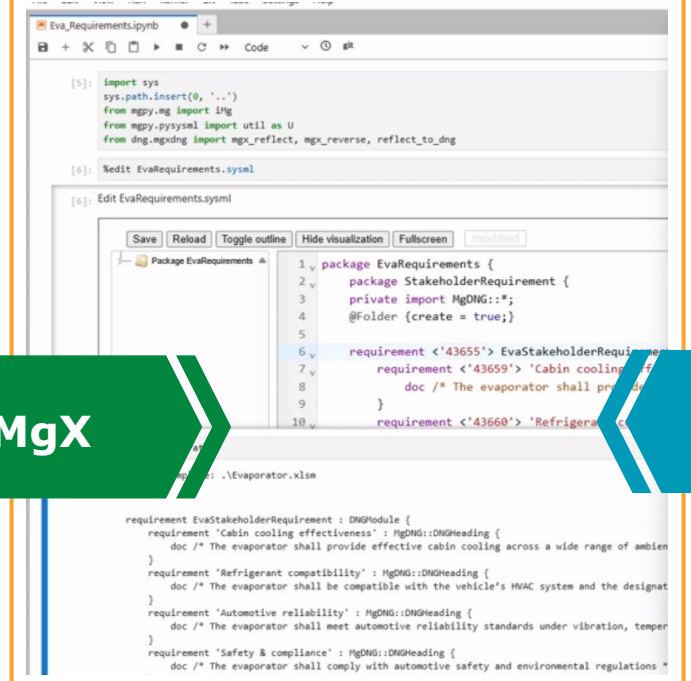
しかも、Pythonによるカスタマイズも柔軟に行うことができます

以下の例は、Excelから、DNGの要求をSysML経由で読み取り、Excelに反映させ、Excelで編集したものをSysMLに反映させて、DNGを更新する例ですユーザーからは、Excelでの操作として取り扱うことができます

Excel



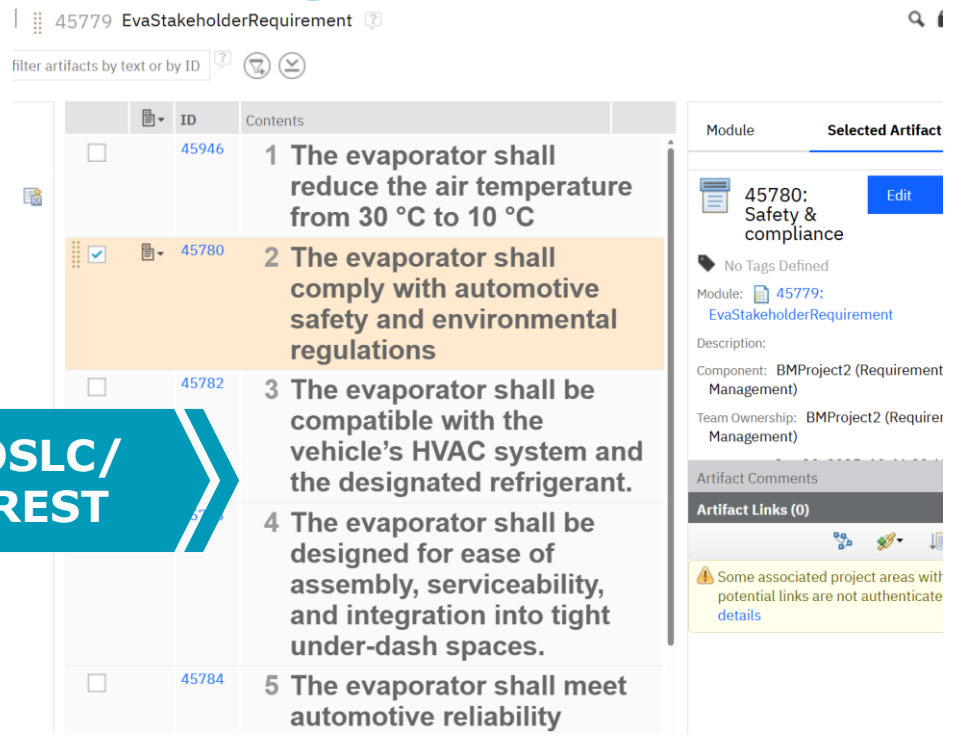
MgPy / SysML v2



MgX

OSLC/ REST

DOORS Next



Jupyter Kernel GatewayによるREST APIサポート

機能強化

■ Mg 0.5.8より、Jupyter Kernel Gatewayが導入されました。これによって、MgPy上で自由にREST APIによるサービスを提供することができます

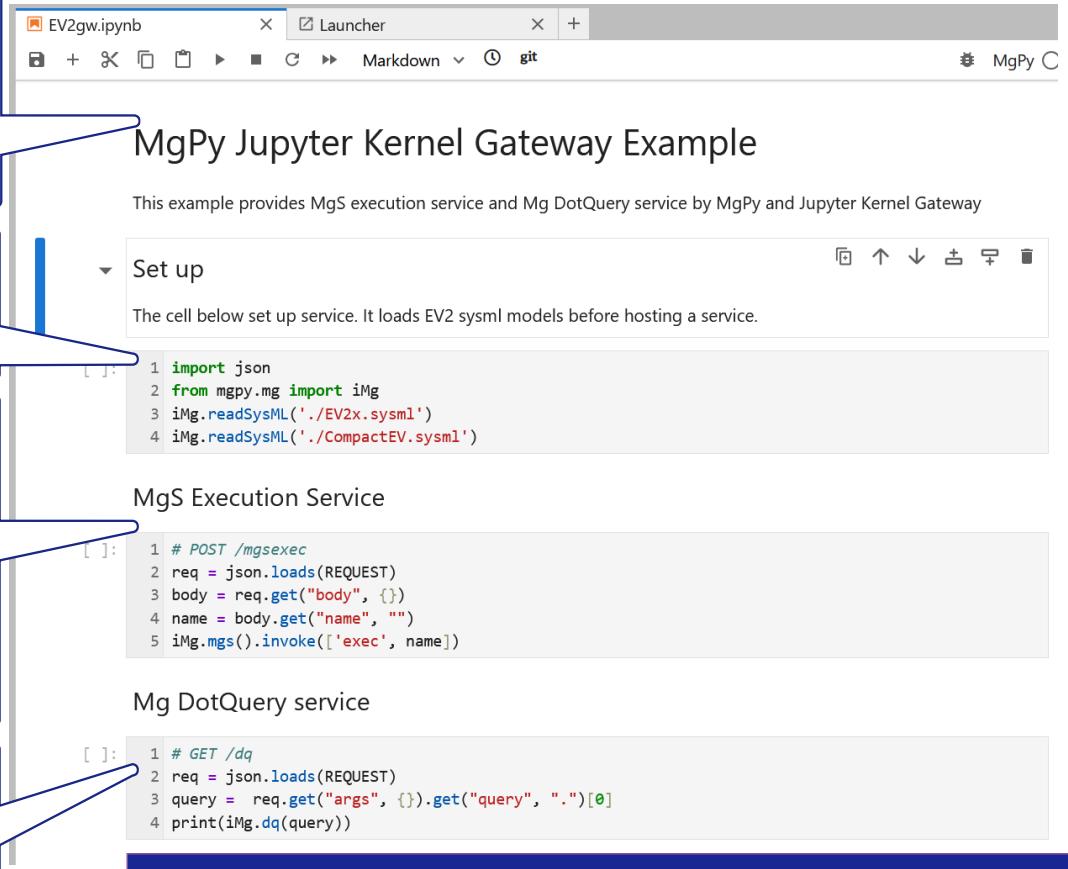
Jupyter Kernel Gatewayによって、Jupyter KernelをREST API(Websocketでのホストもありますが、Mgでは、まだサポートしておりません)によって、サービスを提供することができます。

MgPyカーネルによるノートブックを記述することでREST APIをデザインできます

通常のセルはサービス初期化に用いられます

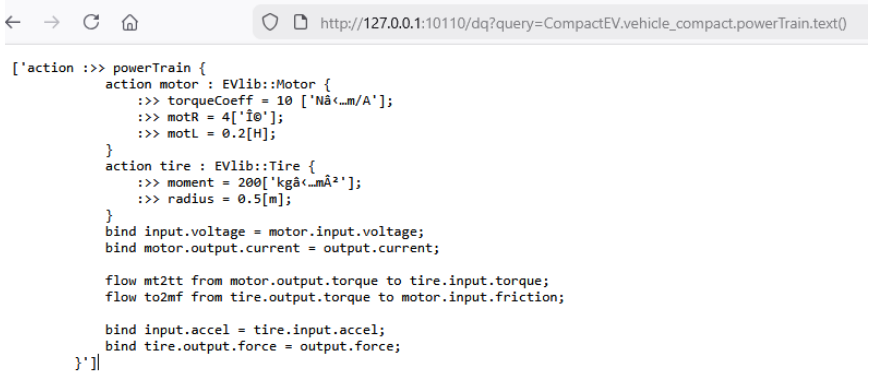
コメントを設定することで、RESTサービスを定義します。これは、/mgsexec に POSTすることで、MgSを用いてSimulinkを実行する例です

GETを用いるとブラウザから簡単に利用できるサービスも実現できます



作成後に、jkg <ノートブックファイル名>を起動することで、サービスが起動されます

この例ですと、ブラウザでlocalhost:10110/dqにアクセスすることで、簡単にモデルをクエリすることができます



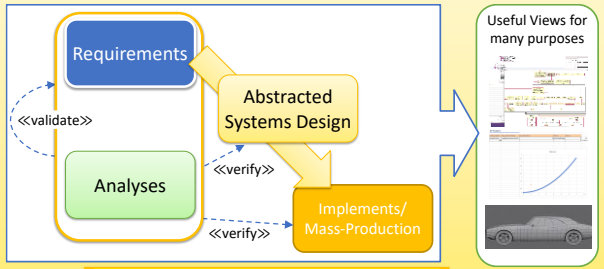
この機構によって、簡単にMgを外部サービスと連携できるようになり、CI/CDなどに活用することができます

日産様事例のご紹介

Nissan EV Thermal Management Study

EV Thermal Management System is critical since battery is thermal-sensitive and energy density is too limited

High-level System design and analysis are required



Useful Views for many purposes

SysMLv2 + MBD modeling and analysis by Mg

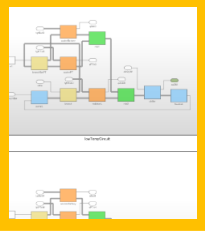
Testcases in Usecases

Formalized Requirements

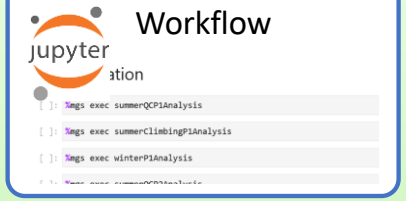
System Context covering many solutions

Analyses to verify solutions

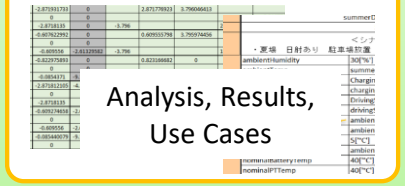
Simulink Thermal Model Library (MBD)



Automated Analysis Workflow



Excel Views in MgX



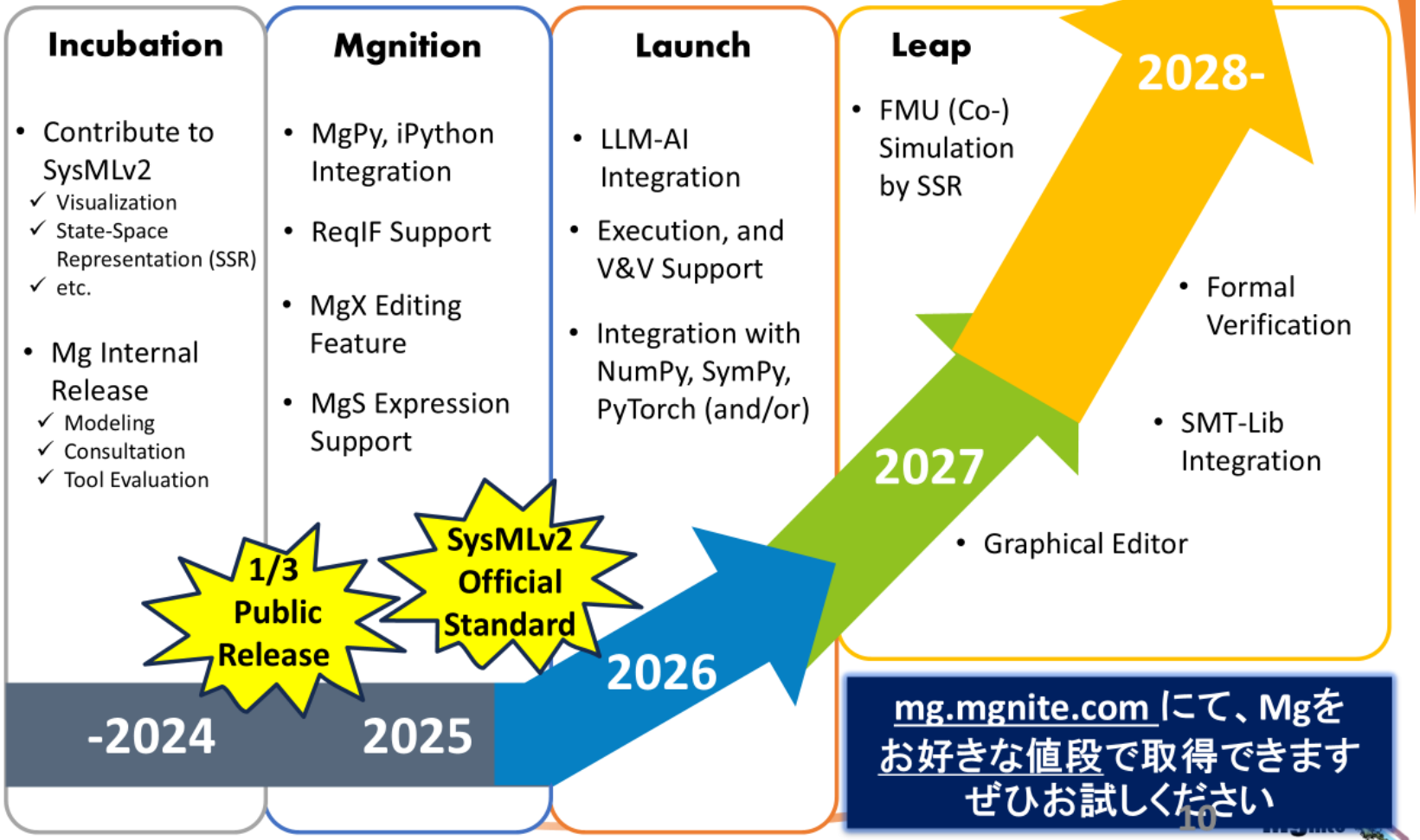
Analysis, Results, Use Cases

Holistic MBSE+MBD integrations were achieved

おわりに

今後に向けて

弊社は今後ともお客様の挑戦にフォーカスいたします



SCSK

夢ある未来を、共に創る。