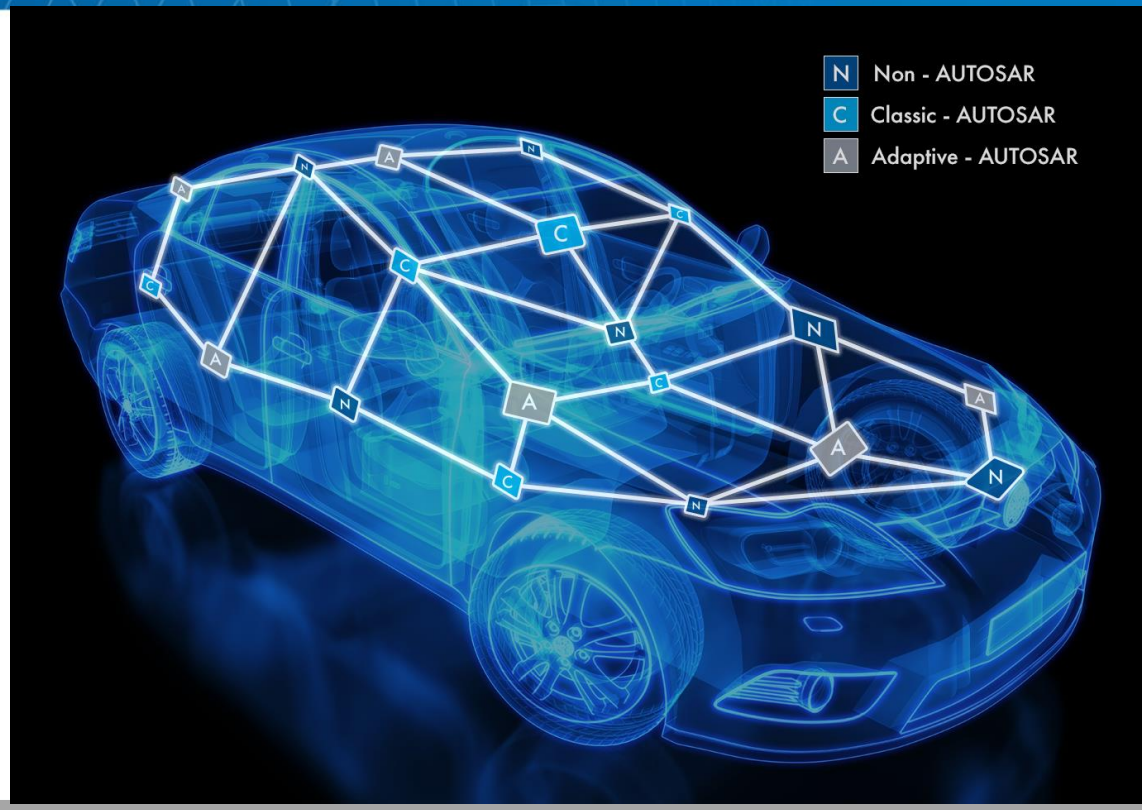
AUTOSAR ClassicおよびAdaptiveソフト開発のモデルベースデザイン



MathWorks Japan
アプリケーションエンジニアリング部
山本　順久

はじめに

ご紹介する内容

- AUTOSAR Classic PlatformとAdaptive Platformについて簡単に紹介します
- SimulinkモデルからAUTOSAR準拠コードを生成するためのモデリング、コード生成手順および必要製品について紹介します

本Webセミナーで得られる（期待される）効果

- モデルベースデザインでAUTOSARソフト開発を進めるための AUTOSARモデリングおよびコード生成に関する基本機能・手順を把握できます

AUTOSAR = 車載ソフト向け標準ソフトアーキテクチャ

Classic Platform：2004年～

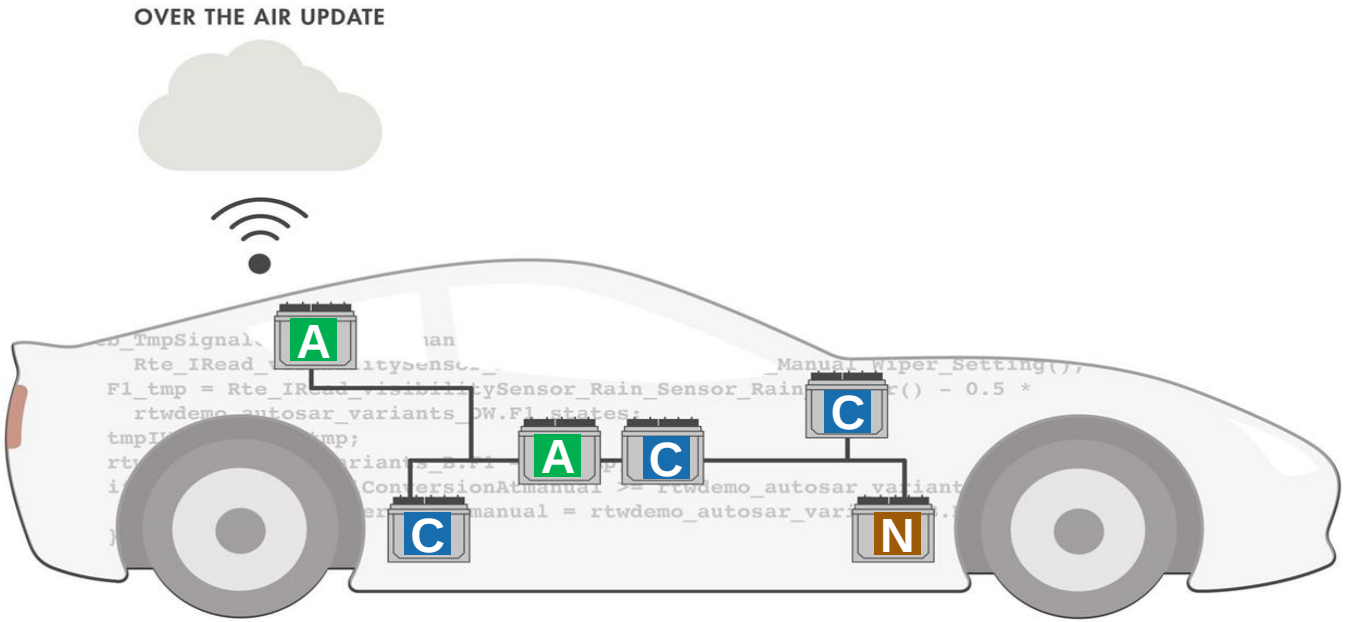
- 従来の制御用ECUで採用（C言語）
- 固定タスクでCAN等の車内通信のみ
- 例：パワトレ、ドラトレ、シャシー、EPS

Adaptive Platform：2017年～

- ADAS等の新しいニーズを元に策定（C++言語）
- 動的スケジューリングや車外との通信も想定
- 例：ITS、ADAS、ゲートウェイ／ドメインコントローラー

Non- AUTOSAR

- 例：インフォテイメント、メータークラスター、HUD等



AUTOSAR = AUTomotive Open System ARchitecture

各リリースでのAUTOSAR対応

R2018b以前

Embedded Coder Support Package for AUTOSAR Standard

- Embedded Coderライセンスがあれば無料で利用可能です
- Classic Platformのみに対応しています
- **モデルの編集のみでもEmbedded Coderが必要です**

C Classic Platform

R2019a以後

AUTOSAR Blockset

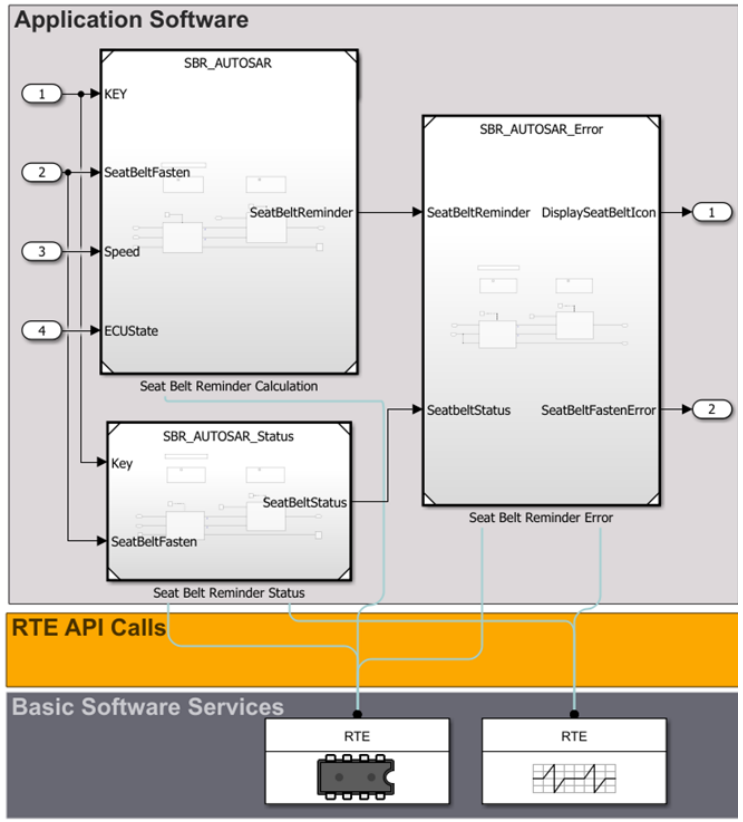
- Simulinkのアドオン製品です
- Classic Platformに加えてAdaptive Platformに対応しています
- **モデルの編集はSimulinkのみで可能です。**
- コード生成やSIL/PIL検証にはEmbedded Coderが必要です

C Classic Platform

A Adaptive Platform

AUTOSAR Blockset の概要

- AUTOSARソフトウェアのモデリング＆シミュレーション
- モデルからのAUTOSARコード＆ARXMLファイルの自動生成
- AUTOSARオーサリングツールと連携した開発が可能



AUTOSAR Classic (Cコード生成)

```
void UpdateOdometerRunnable(void)
{
  uint8 rtb_TmpSignalConversionAtPulseO;
  uint16 rtb_Sum;
  rtb_TmpSignalConversionAtPulseO =
    Rte_IrvIRead_UpdateOdometerRunnable_pulsedata();
  rtb_Sum = (uint16)((uint32)(uint8)(rtb_TmpSignalConversionAtPulseO -
```

AUTOSAR Adaptive (C++コード生成)

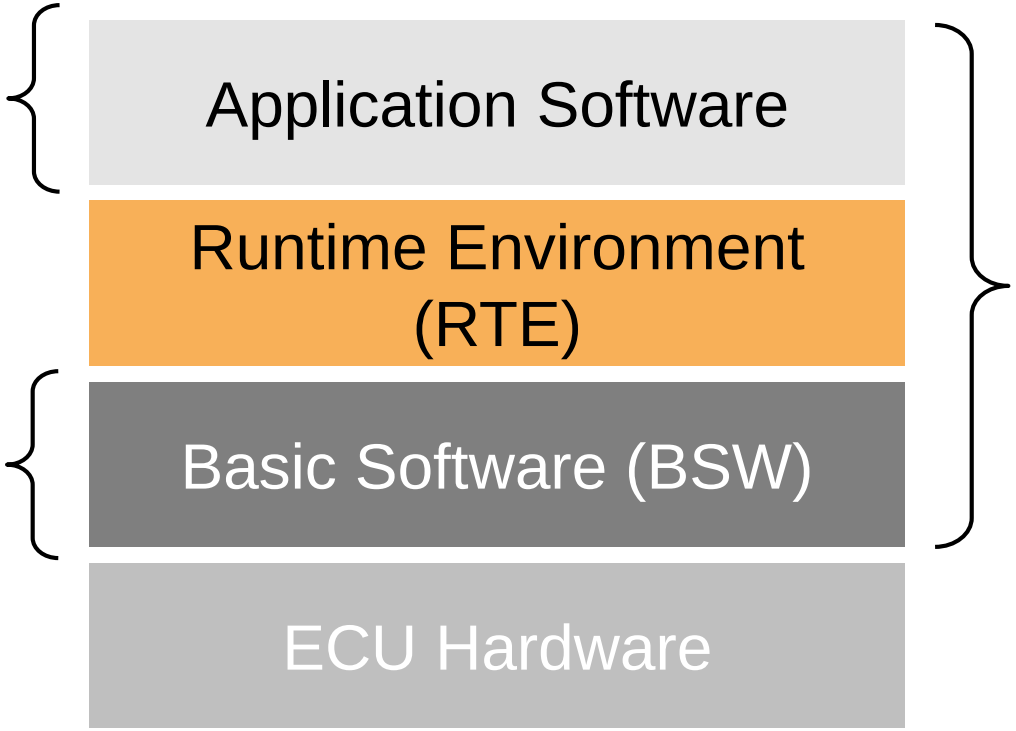
```
boolean_T mObjectDetectionModelClass::autosar_LaneGuidance_sf_msg_pop_EvtIn(void)
{
  boolean_T isPresent;
  const ara::com::SampleContainer< ara::com::SamplePtr< const real_T > >
    *sampleContainer;
  ara::com::SamplePtr< const real_T > samples;
  if (autosar_LaneGuidance_DW.EvtIn_isValid_i) {
    isPresent = true;
  } else {
```

マスワークス提供機能が対象としているAUTOSARソフト階層（CPの例）

マスワークス提供

- SWCモデリング
- シミュレーション
- コード生成
- SIL/PIL検証

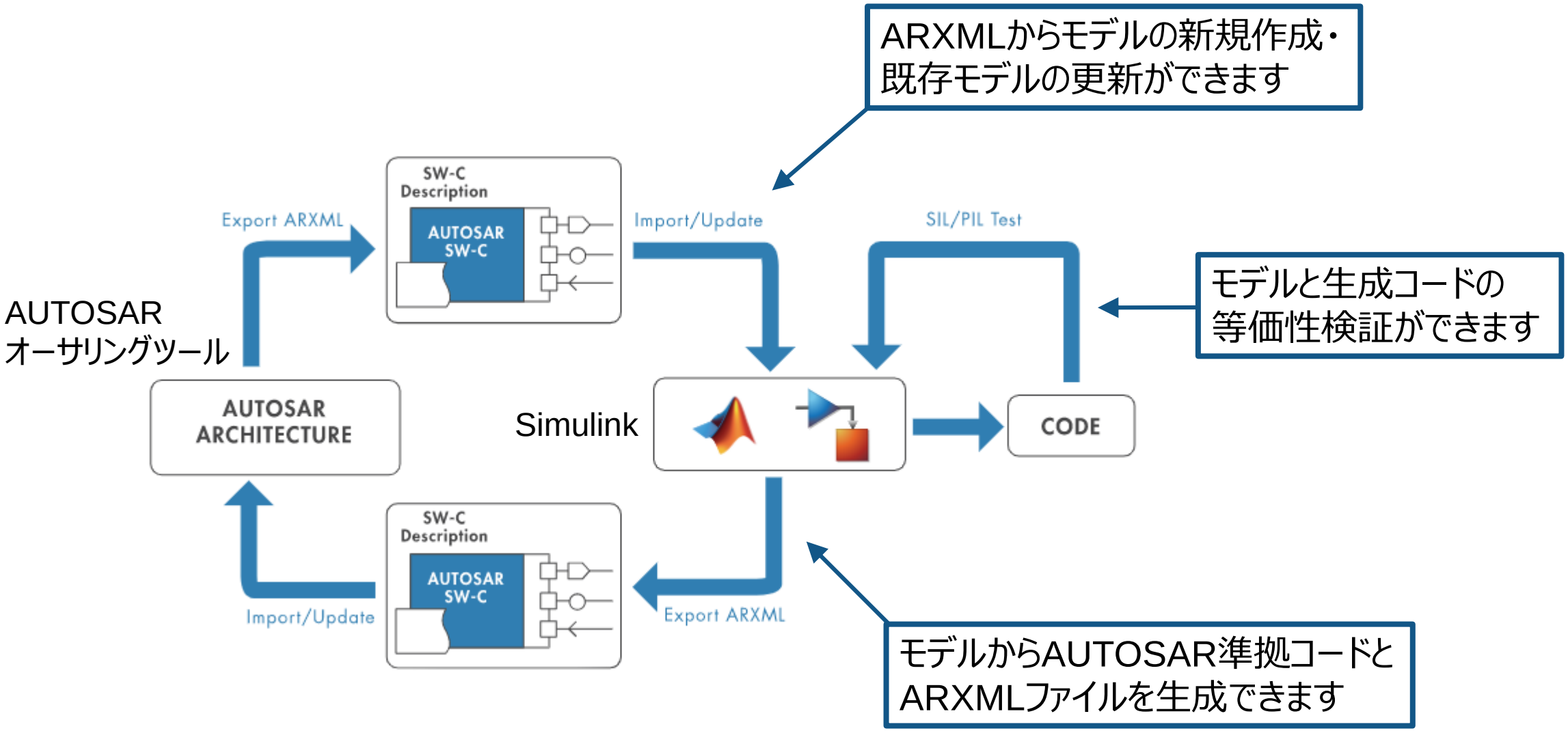
- BSWサービスブロック&シミュレーション模擬
  - 診断
  - 不揮発メモリ管理



マスワークス以外から提供

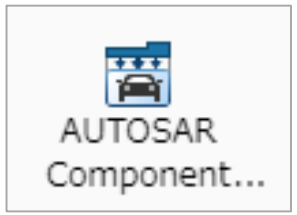
- AUTOSARオーサリングツール
  - BSWコンフィグ設定
  - RTE生成
  - ビルド
- MCAL
  - デバイスドライバ

モデルベースAUTOSARソフト開発フロー



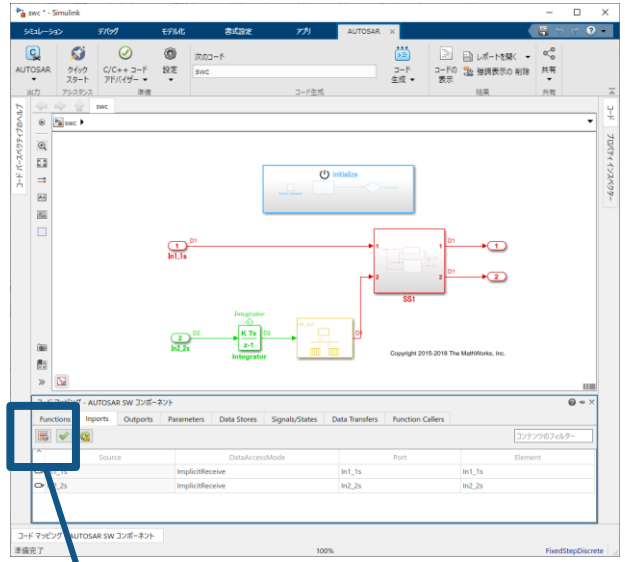
Classic Platform Cコード生成フロー

①アプリメニューからAUTOSAR設計ツールを起動します

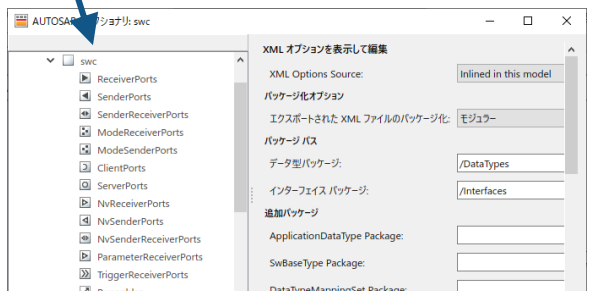


②AUTOSARを選択して[クィックスタート]を実行します

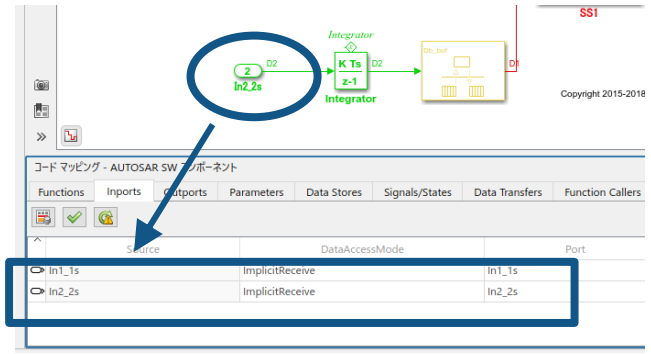
③モデルエディタのコードマッピングを開きます



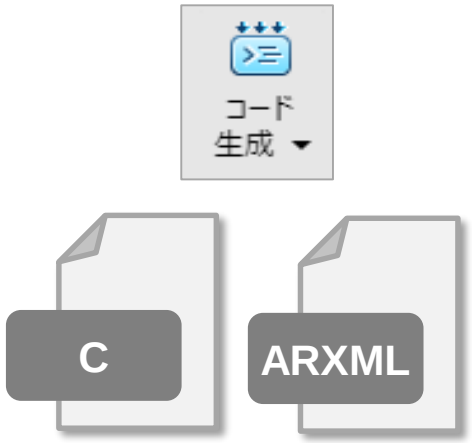
④AUTOSARディクショナリを確認・編集します



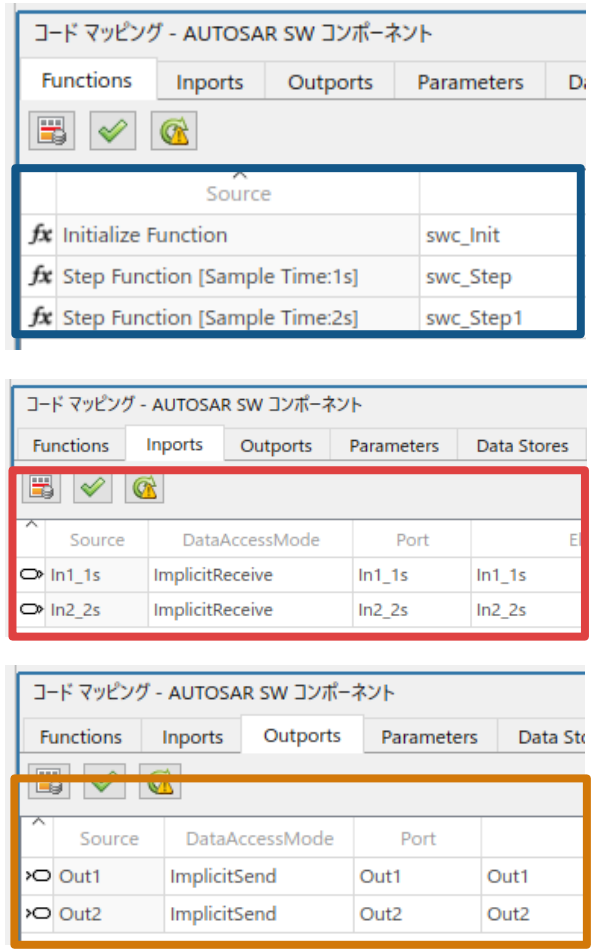
⑤AUTOSARプロパティをモデル要素にマッピングします



⑥コードを生成します

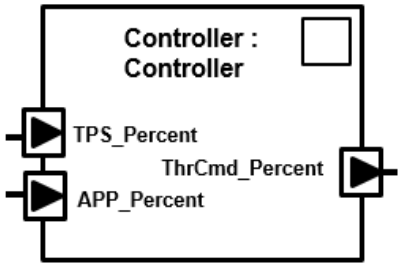


Classic C 生成コードの概要

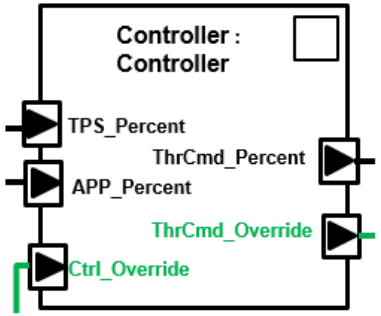



```
void swc_Step(void)      /* 1sec周期Runnable */
{
  …

  rtb_Sum_n = 2.0 * rtb_RateTransition + Rte_Iread_swc_Step_In1_1s_In1_1s();

  Rte_Iwrite_swc_Step_Out1_Out1(5.0 * (Rte_Iread_swc_Step_In1_1s_In1_1s() + 3.0 *
    rtb_RateTransition) + rtb_Sum_n);

  Rte_Iwrite_swc_Step_Out2_Out2(rtb_Sum_n);
}

void swc_Step1(void)     /* 2sec周期Runnable*/
{
  Rte_IrvIWrite_swc_Step1_IRV1(rtDW.Integrator_DSTATE);
  rtDW.Integrator_DSTATE += 2.0 * Rte_IRead_swc_Step1_In2_2s_In2_2s();
}

void swc_Init(void)      /* 初期化Runnable */
{
  rtDW.Integrator_DSTATE = 1.0;
}
```

ARXMLファイルを用いたモデルの新規作成、モデル更新による差分開発

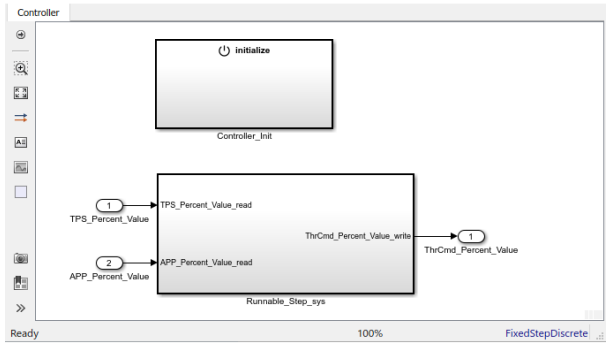


SWC修正

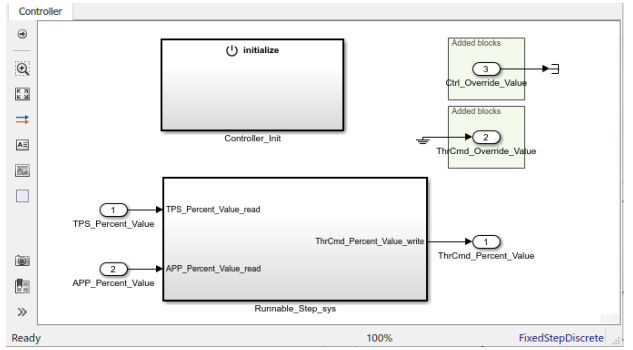


arxml.importer
createComponentAsModel

AUTOSARプロパティが
設定された空モデルを
作成できます



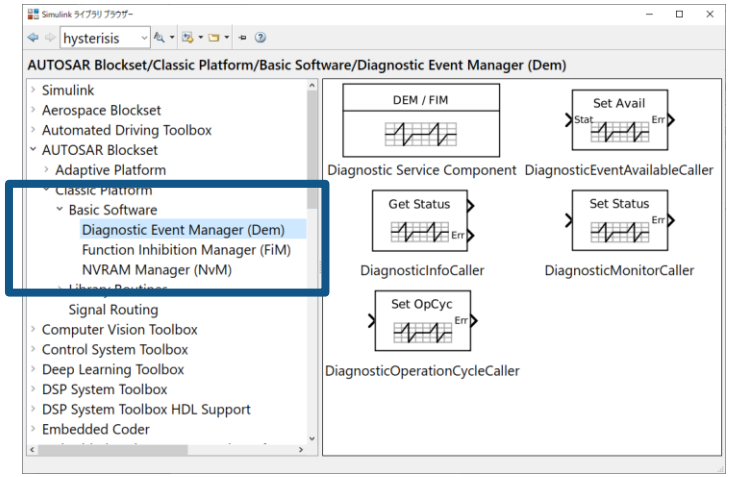
arxml.importer
updateModel

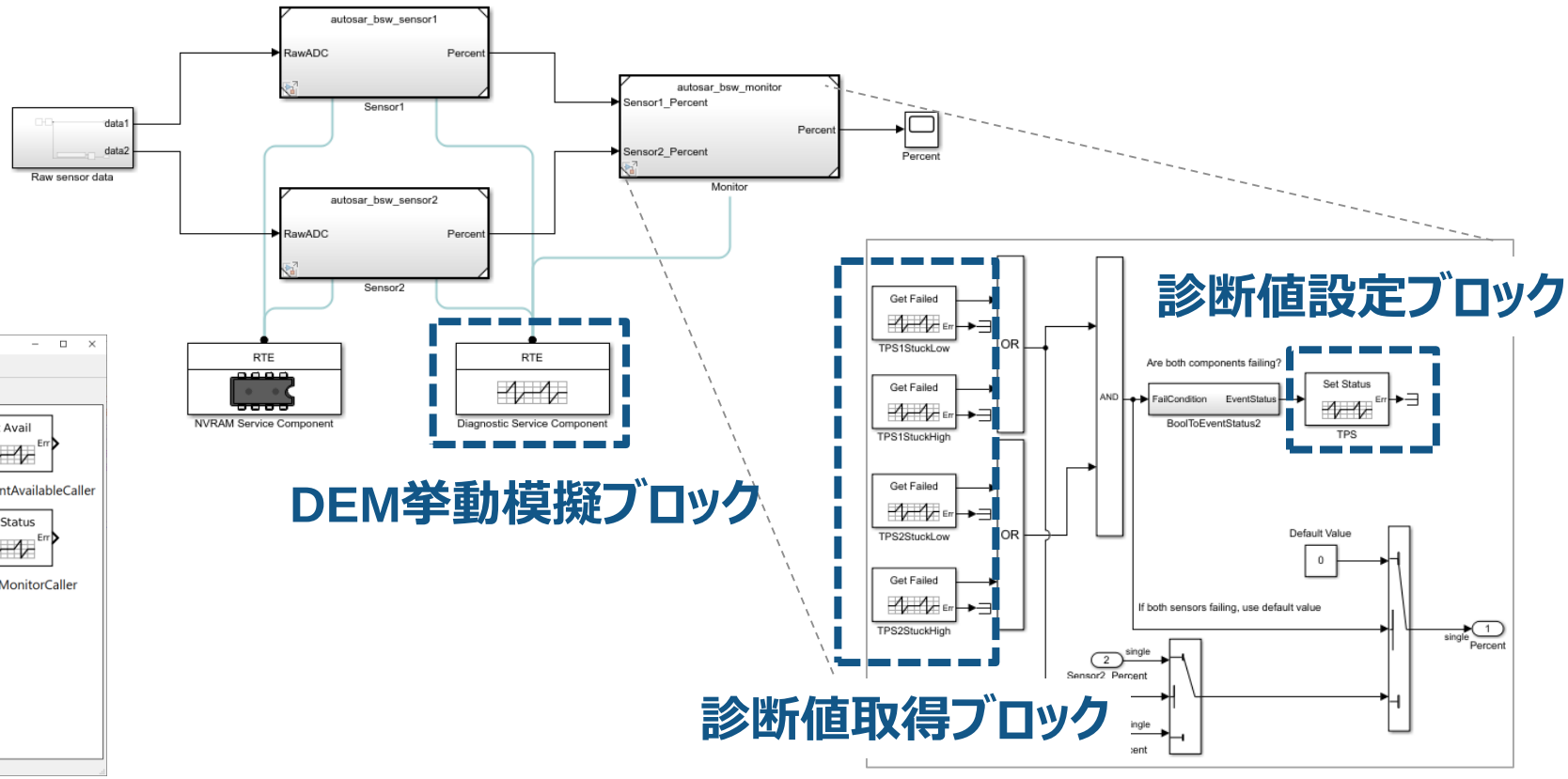


既存モデルの更新＆変更レポート
を作成できます

Classic Platform BSWサービスの利用

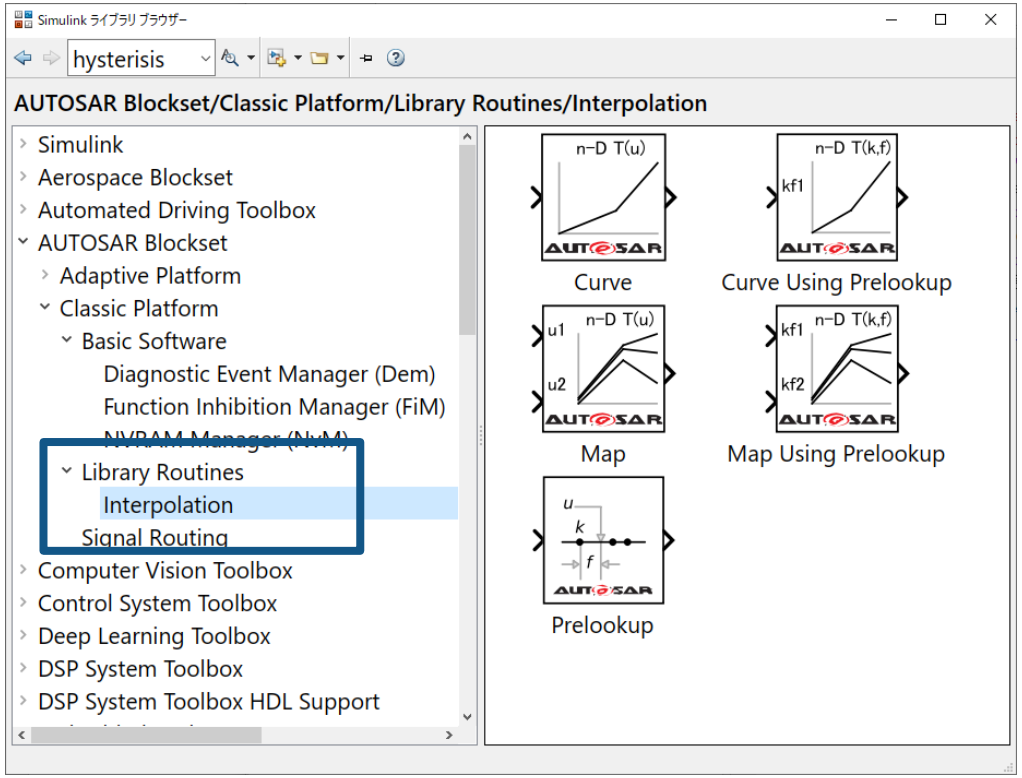
- Basic Softwareライブラリを用いてBSWサービスを呼び出すことができます
  - 不揮発メモリ(NVM) / 診断イベント(DEM) / 機能停止・解除（FiM）
  - 診断用カウンターの挙動を模擬してシミュレーションを行うことができます





AUTOSAR ルックアップテーブル関数の利用

- Library Routines/Interpolationに各種ルックアップテーブルが提供されています。
- コード生成するとIFL/IFXライブラリ関数コールが生成されます。

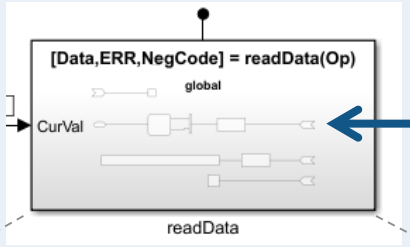



```
/* Model step function */
void Runnable_Step(void)
{
  Ifl_DPResultF32_Type rtb_Prelookup;

  /* PreLookup: '<Root>/Prelookup' incorporates:
   *  Inport: '<Root>/In2'
   */
  Ifl_DPSearch_f32(&rtb_Prelookup, Rte_IRead_Runnable_Step_In2_In2(),
                   (Rte_CData_Bp_4_single())->Nx, (Rte_CData_Bp_4_single())->Bp1);

  /* Outport: '<Root>/Out2' incorporates:
   *  Interpolation_n-D: '<Root>/Curve Using Prelookup'
   */
  Rte_IWrite_Runnable_Step_Out2_Out2(Ifl_IpoCur_f32(&rtb_Prelookup,
    Rte_CData_Lcom_4_single()));
}
```

サーバー/クライアントのモデリング

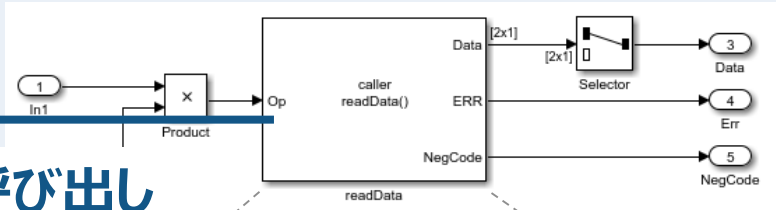
サーバー

Simulink Functionブロックで記述します



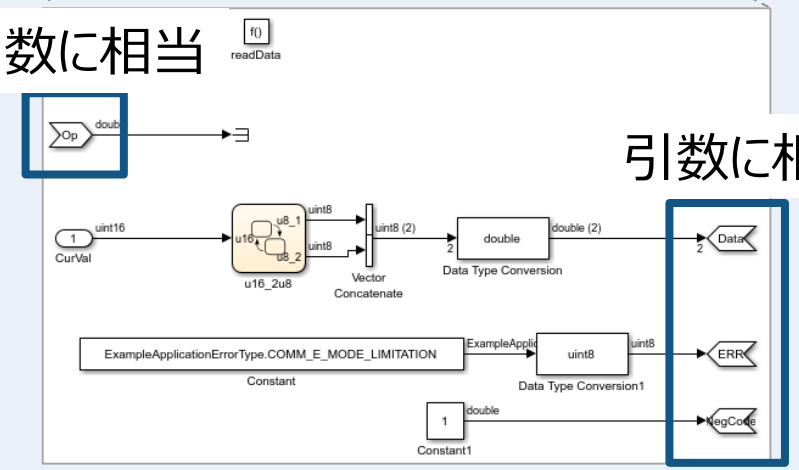
クライアント

Function Callerブロックからサーバー Simulink Functionを呼び出します

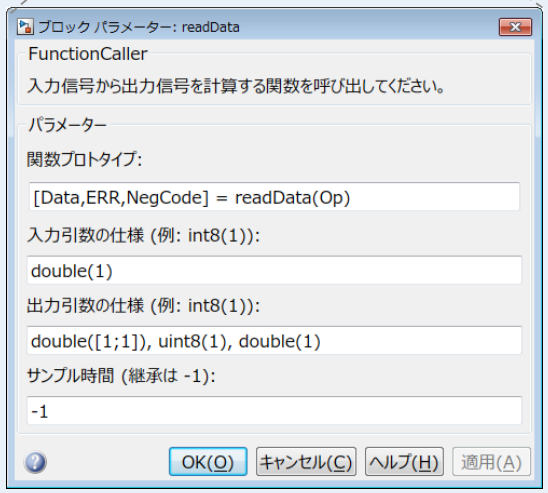


サーバー呼び出し

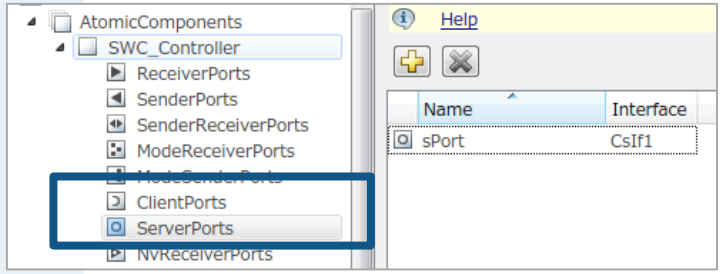
引数に相当

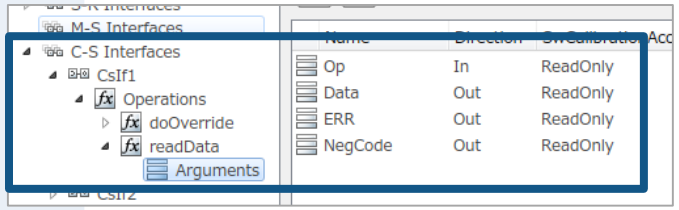


引数に相当



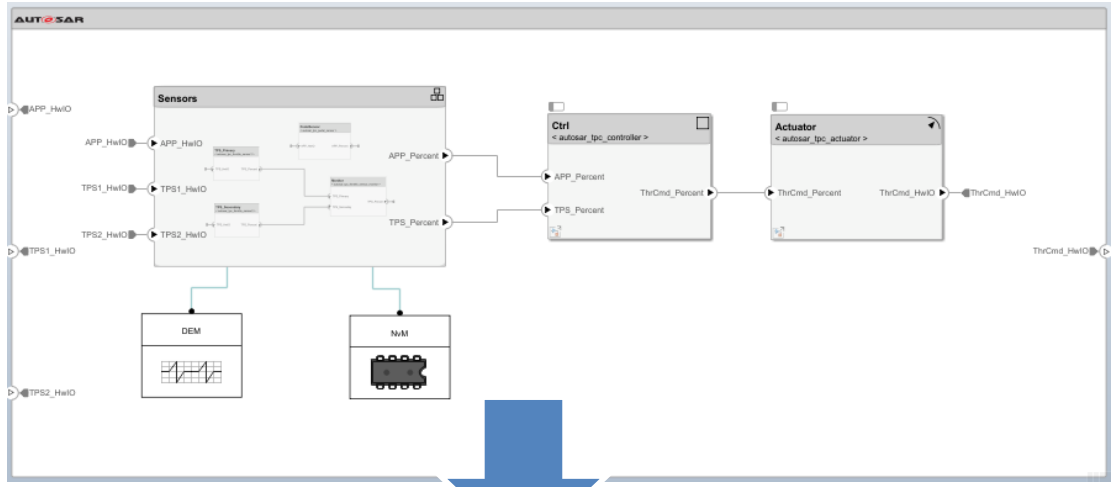
サーバー/クライアント
ポート・インターフェース設定

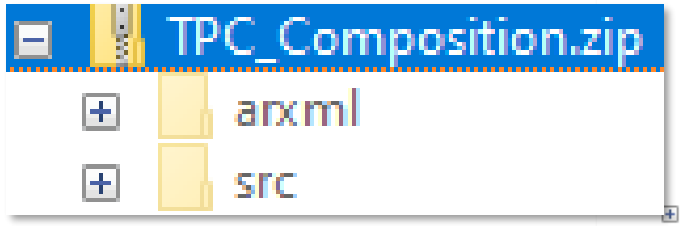


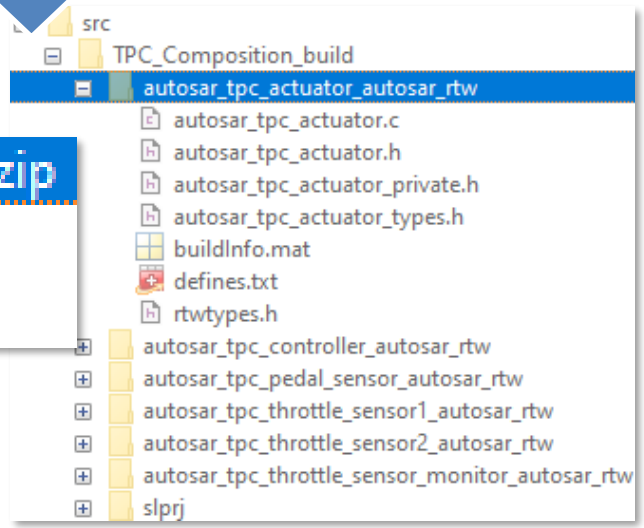


System Composerを用いたSWCコンポジション設計

- SWCコンポジションアーキテクチャを
  グラフィカルに設計できます
  - System Composerライセンスが必要です
  - Classic Platformのみに対応しています
- 各SWCロジックをSimulinkモデルと
  関連付けて開発できます
- SWCコンポジションを含むCコード
  およびARXMLファイルを生成できます



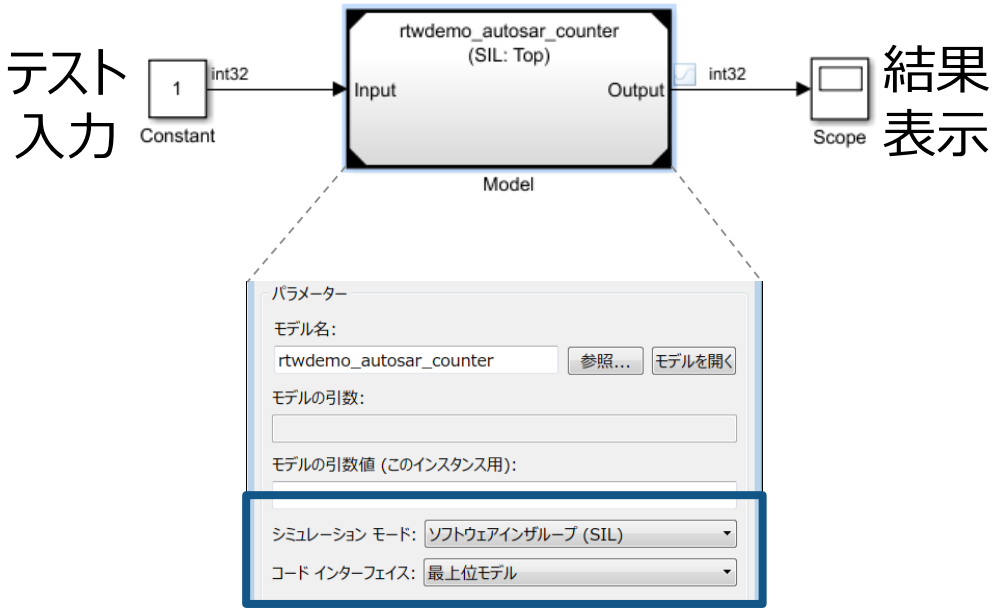




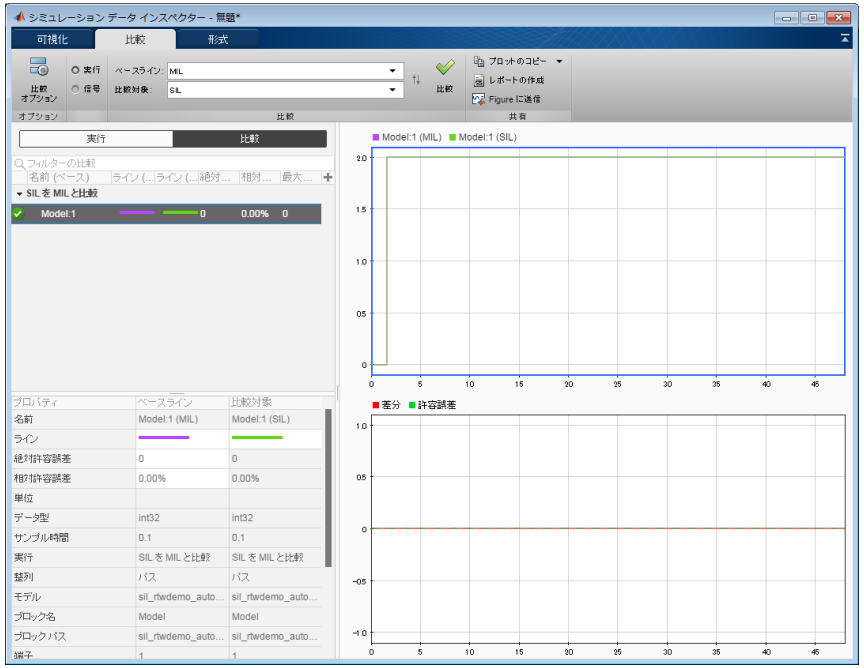
SIL/PILによるモデル・生成コードの等価性検証

- AUTOSARモデルのシミュレーションモードとしてMIL/SIL/PILを選択できます
  - 参照モデルの場合、コードインターフェースを最上位モデルに設定する必要があります
  - RTE/VFBはスタブとして実装、SWCロジック動作にフォーカスしたB2Bテストになります

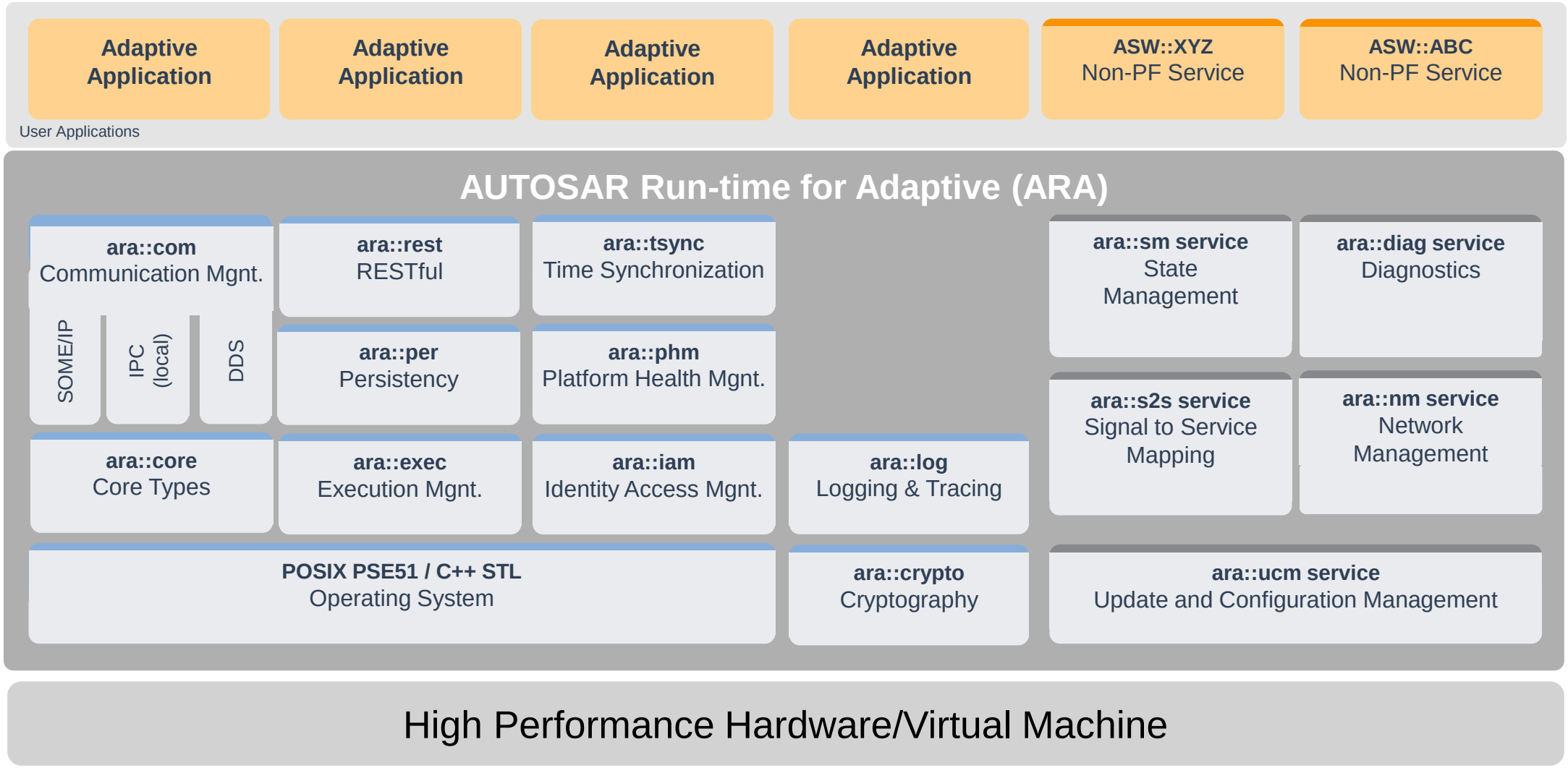
対象モデル



シミュレーションデータインスペクターを用いたMIL/SIL比較例

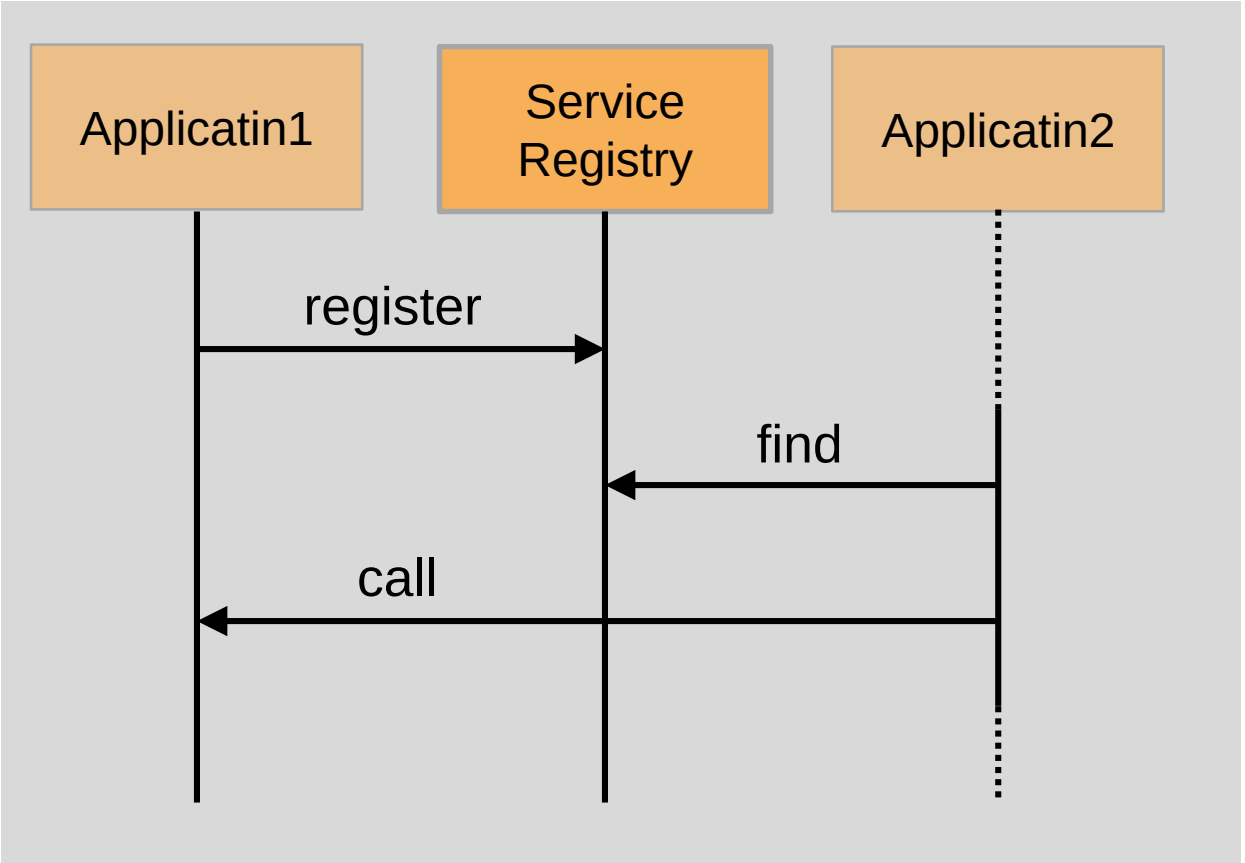


AUTOSAR Adaptive Platform ソフト階層



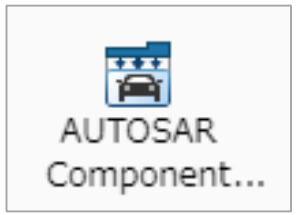
Adaptive Platform サービス指向通信

- インターフェースはサービスとAPI
- サービスは下記で構成
  - Methods (関数)
  - Events (メッセージ)
  - Fields (データ)

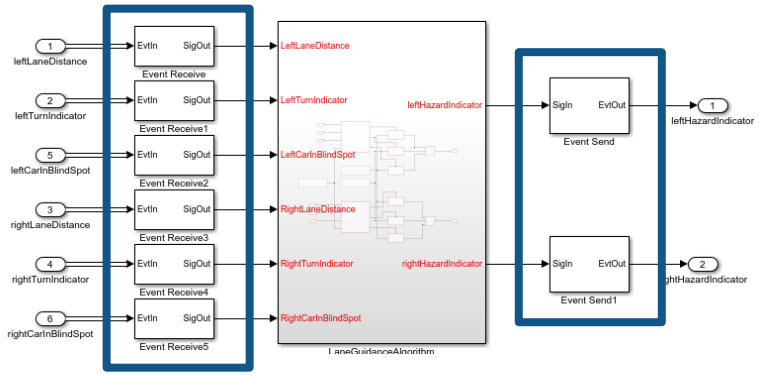


Adaptive Platform C++コード生成フロー

①アプリメニューからAUTOSAR
設計ツールを起動します

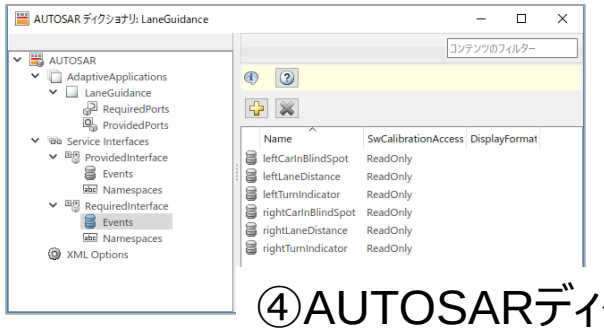


②入力にEvent Receive、
出力にEvent Sendブロックを挿入します

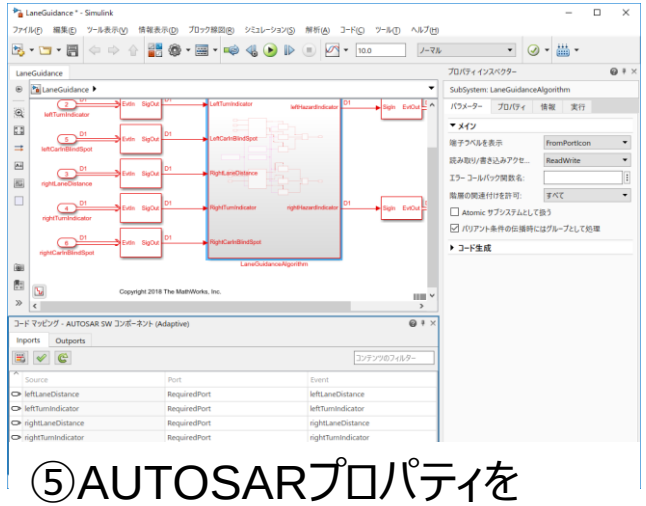


③ AUTOSAR Adaptiveを選択して
[クィックスタート]を実行します

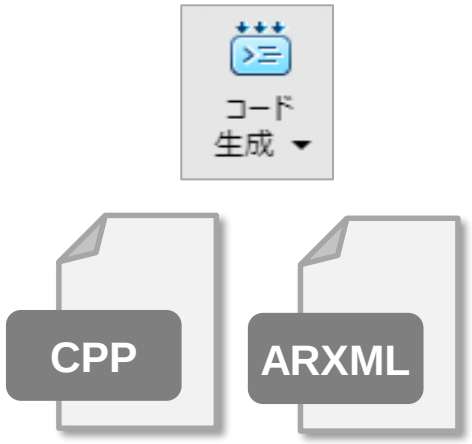
④AUTOSARディクショナリを
確認・編集します



⑤AUTOSARプロパティを
モデル要素にマッピングします



⑥コードを生成します



Adaptive C++
生成コードの概要

```
boolean_T ModelNameModelClass::ModelName_sf_msg_pop_EvtIn(void)
{
   // ARAミドルウェアを通じた入力端子相当データの受信
   // Update()でチェックしてGetCachedSamples()で取得
}

void ModelNameModelClass::ModelName_sf_msg_discard_EvtIn(void)
{
   // データ受信処理の終了フラグを設定
}
```

各入力に応じてメソッドが
個別に生成されます
（入力端子の数だけ生成）

①イベント経由で入力データを取得
②取得データを使って計算
③計算結果をイベント経由で出力

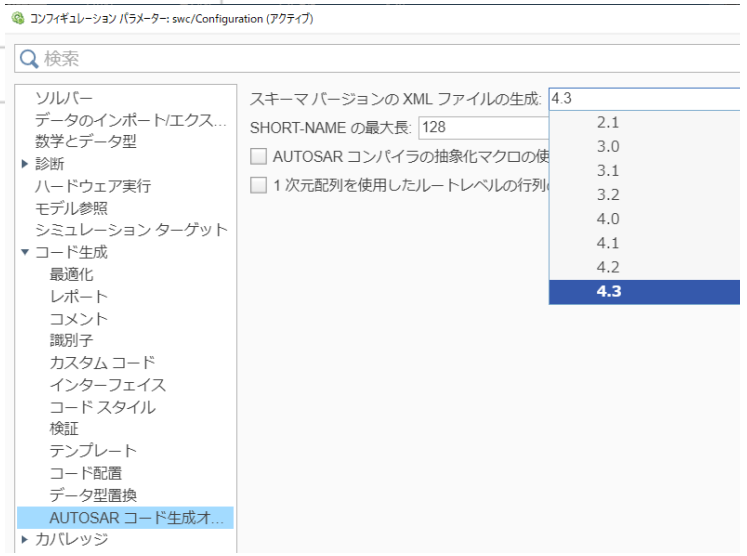
```
void ModelNameModelClass::step()
{
   // 入力データの受信
  if (ModelName_sf_msg_pop_EvtIn()) {
     ModelName_DW.SigOut = *(real32_T *)ModelName_DW.EvtIn_msgDataPtr_l;
   }
   ModelName_sf_msg_discard_EvtIn();

   // モデルロジック実行
  ModelName_DW.EvtOut_msgData = 1;

   // 出力データの送信
   ModelName_sf_msg_send_EvtOut();
}

void ModelNameModelClass::initialize()
{
   // 初期化処理
}

void ModelNameModelClass::terminate()
{
   // 終了処理
   ProvidedPort->StopOfferService();
}
```

- step：モデルロジック実行メソッド
- initialize：初期化処理メソッド
- terminate：終了処理メソッド

対応スキーマバージョン (R2020a時点)

Classic Platform

スキーマ バージョンの値	インポートでサポートされるスキーマ リビジョン	エクスポート スキーマ リビジョン
4.3 (既定値)	4.3.0、4.3.1	4.3.1
4.2	4.2.1、4.2.2	4.2.2
4.1	4.1.1、4.1.2、4.1.3	4.1.3
4.0	4.0.1、4.0.2、4.0.3	4.0.3
3.2	3.2.1、3.2.2	3.2.2
3.1	3.1.1、3.1.2、3.1.3、3.1.4	3.1.4
3.0	3.0.1、3.0.2、3.0.3、3.0.4、3.0.5、3.0.6	3.0.2
2.1	2.1 (XSD rev 0014、0015、0017、0018)	2.1 (XSD rev 0017)



Adaptive Platform

R18-10、R19-03

※MATLABリリースにより対応スキーマバージョンは異なります

さいごに　Call to Action

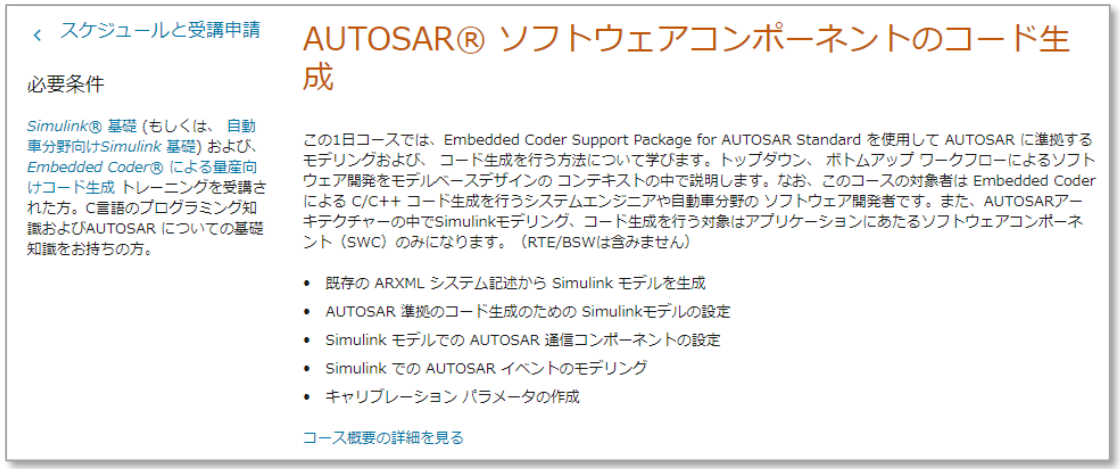
AUTOSARソリューションページ

- 参考情報が豊富に紹介されています
  - ユーザ事例
  - ホワイトペーパー

https://jp.mathworks.com/solutions/automotive/standards/autosar.html

AUTOSARトレーニング

- 実践的な例題でAUTOSARモデリング・コード生成について習得できます



https://jp.mathworks.com/training-schedule/code-generation-for-autosar-software-components

ご清聴ありがとうございました

Accelerating the pace of engineering and science