論文・解説

18

情報制御系ソフトウェアのモデルベース開発
Model Based Development of Infotainment Software

白　雪峰 *1　本城　創 *2　末冨　隆雅 *3
Xuefeng Bai　So Honjo　Takamasa Suetomi

要　約

CASE（Connected, Autonomous, Shared, Electric）を代表とする 100 年に一度の変革を迎え自動車のシステムはますます複雑となり，そのソフトウェアの規模は増大している。開発規模が増大し，開発が破綻しないためには開発の効率化が必須となっている。これまでエンジン制御や運転支援システムなどの動的制御系に対してはモデルベース開発を適用してきたが，今後もソフトウェア規模が増大するインフォテイメントなどの情報制御系において，モデルベース開発による早期の仕様検証と，自動コード生成によるソフトウェア実装の効率化に取り組んだ。本稿では，情報制御系のモデリング手法及びモデル交換や検証の開発環境について述べる(1)。

Abstract

In the age of CASE (Connected, Autonomous, Shared, Electric), automotive software has been growing in size with systems becoming more and more complex, making it crucial to improve development efficiency to avoid the collapse of development caused by expanded development scales and post-implementation reworks. So far, model based development has been applied to dynamic control systems such as engine controls and advanced driver assistance systems, and from now on, such development will also be required for information control systems, including infotainment systems that will further grow in the software scale to perform specification verifications at an early stage. Also we challenged efficient software implementation using automatic code generations. This paper describes the models of information control systems and the development environment of the model exchanges and verifications.

Key words : Model based development, Infotainment, SysML, MILS, SILS

1. はじめに

自動車の CASE（Connected, Autonomous, Shared, Electric）技術の進化に伴い，システムは複雑化し，ソフトウェアの規模は増大し，開発の難易度が上がってきている。また，CASE 技術は急速に進化しており，市場競争が激化している。自動車メーカーは常に最新の技術を採用する必要があるが，新しい技術はソフトウェアの規模を増大させるため，開発の失敗リスクが高まる。更に，自動車のソフトウェアは複数のパートナー企業によって開発される場合がある。例えば，自動運転の開発には，自動車メーカー（OEM），ソフトウェアメーカー，センサーメーカーなどが関わることがあり，コミュニケーションや調整不足の問題が生じる懸念が高まる。

以上のような要因により，自動車の CASE 技術に関するソフトウェア開発が破綻するリスクがあるといえる。OEM は，ソフトウェア開発の品質管理やリスク管理に十分な注意を払う必要がある。また，パートナー企業とのコミュニケーションや調整を円滑に行うことが重要であり，抜本的な開発の効率化が求められている。

2. ソフトウェア開発プロセス

2.1　自動車ソフトウェア開発プロセス概要

自動車ソフトウェア開発において，一般的な開発形態としてＶ字モデルの各プロセス内の作業が全て完了して，次プロセスに進むウォーターフォール型開発プロセスが採用されている。ウォーターフォール型開発プロセスは，ソフトウェア開発の上流活動から下流活動とテスト活動が行われ，段階的に進める手法である。

ソフトウェア開発の上流活動では，ハードウェアも含むシステムとしての要件定義やシステムアーキテクチャ設計が行われる。ユーザー要件を明確化し，システムの

*1〜3　統合制御システム開発本部
Integrated Control System Development Div.

機能や性能要件を洗い出す。非機能要件も加味しシステムのアーキテクチャやモジュールの設計を行い，要件に基づいたシステム仕様書を作成する。

下流活動では，上流活動で作成されたシステム仕様書で定義されたソフトウェア要求に基づいて，ソフトウェアの詳細設計が行われ，分割したモジュールやコンポーネントの詳細な設計を行い，実装するソースコードを開発する。

テスト・結合活動では，おおよそ単体テスト，結合テスト，システムテストの 3 つのレベルでソフトウェア及びシステムの検証が実施される。

単体テストでは，下流活動で作成された各モジュールやコンポーネントに対して，個々のモジュールが正しく動作し，要件を満たしていることを確認する。結合テストでは，単体テストが完了したモジュールやコンポーネントを結合し，システム全体の動作，モジュール間の相互作用やインターフェースの正常性を確認する。システムテストでは，システムが要件を満たしており，ユーザーの期待に沿った動作をすることを確認する。実際の環境でのテストやユーザビリティテストを実施し，システムが要求された機能を適切に提供することを確認する。

2.2　ソフトウェア開発のワークシェアパターン

プロジェクトや対象システムによって OEM とサプライヤーでの作業分担は異なる。以下に一般的なワークシェアパターンを示す（Fig. 1 参照）。

①インハウス開発パターン：OEM が自社内の開発チームをもち，ソフトウェア開発を自社で行うパターンである。OEM はソフトウェアの設計，開発，テスト，保守などの全てのプロセスを自社内で管理する。メリットとしては，直接的なコントロールが可能であり，セキュリティや知的財産の保護が容易である。

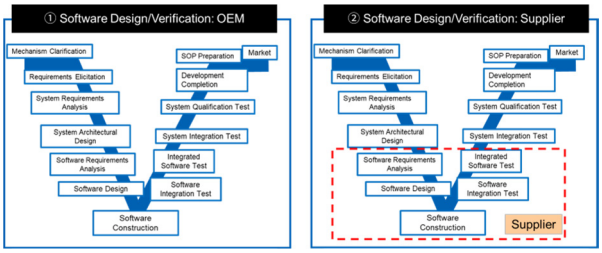


Fig. 1　Work Sharing Patterns in Software Development

②ソフトウェアサプライヤーへの委託パターン：OEM がソフトウェア開発をサプライヤーに委託するパターンである。サプライヤーはソフトウェアの開発，テスト，保守などを担当する。メリットとしては，OEM 内部の開発コストやリソースの削減が可能であり，外部の専門知識や技術を活用できる。

2.3　現状の開発プロセスの課題

自動車ソフトウェア開発における現状の問題として，手戻りの発生による開発効率の低下がある。特に，ソフトウェア要求定義に関連する手戻りが多いことが社内調査で判明している。ソフトウェア要求の正確性や一貫性は，ソフトウェア開発全体の品質に大きな影響を与えている。要求定義の不備や不明瞭さが後の開発段階で判明し，手戻りや修正が発生することはよくある問題である。

3. 目指す開発の姿

3.1　次世代の開発手法

従来の開発手法では要求，要件を自然言語による文章の仕様書に記載しており，内容の確認を人手によるレビューで行ってきた。このような従来の開発では仕様書の記載内容に曖昧さや間違いが混入しやすいため，Fig. 2 に示すとおりソフトウェア実装後にテストで要求と異なる動きが発覚し，元の要求の間違いに気づき大きな手戻りとなることがある。

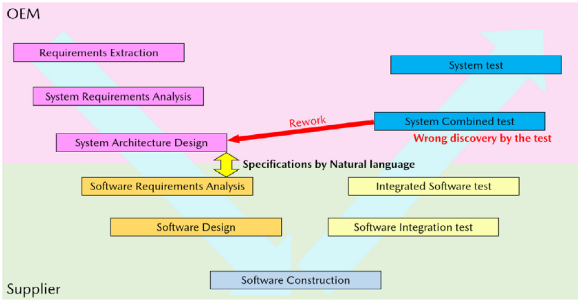


Fig. 2　Development Method (As is)

そこで次世代の開発手法として Fig. 3 に示すとおり，各工程ごとに検証を実施した後に，次工程を行うことで手戻りを少なくできると考える。また，要求や要件を曖昧さがない形式的に記述することで機械的にモデル化及びコード生成することでき，人手による間違いの混入を防ぐことができる。先行してエンジン制御開発では，ソフトウェア設計の工程において形式的なモデルによるソフトウェア開発を行っており，品質改善や効率化で大きな成果を上げている[(2)]。

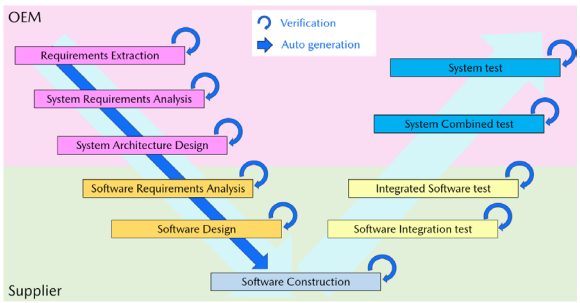


Fig. 3　Development Method (To be)

3.2　モデルでの事前検証（モデル検査，MILS）

各工程おいてモデルで記述した要求，要件，仕様を検証するために，MILS（Model in the Loop Simulation），及び形式検証の一つであるモデル検査による検証がある。シミュレーションは，モデルを実行した結果が入力に対して期待される出力が得られることを検証する。一方，モデル検査は，要求されていない状態（出力）となる入力の有無を網羅的に検査し，そうなる条件を検出，もしくは要求されていない状態にはならないことを保証するものである。Fig. 4 にシステムアーキテクチャ設計における MILS 及びモデル検査の概要を示す。

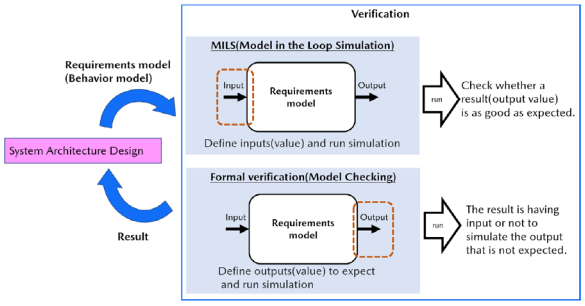


Fig. 4　MILS and Model Checking Overview

3.3　モデルからコード生成

モデルはシミュレーション検証環境へのコード実装が可能であると同時に，実機環境へのコード実装も可能である。これによりモデルで定義された情報から機械がソフトウェアの詳細設計を実施し自動でソフトウェアコードを生成することで大幅な効率化と実装期間短縮を期待できる。

4. 情報制御系モデル

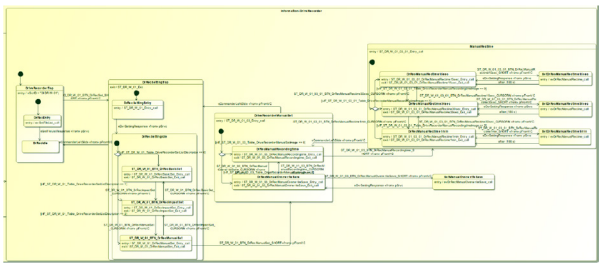
4.1　情報制御系の特性

エンジン制御や ADAS 等の動的制御系モデルは，エンジンや車両など動的な制御対象をプラントモデルとして，それを制御するソフトウェアをモデルで開発している。センサーなどの情報を入力として，望ましい状態にできるアクチュエータ制御操作量を決定する演算を繰り返し行う。

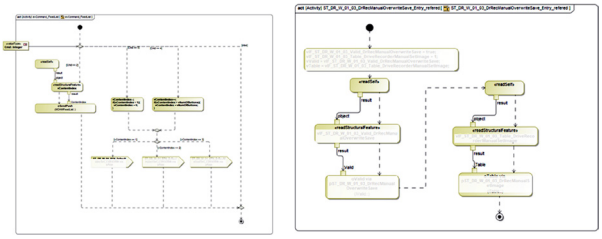
一方，情報制御系は，通信装置や表示装置などソフトウェアで動く制御対象をソフトウェアで制御することになり，ソフトウェアモジュール間のメッセージなどのイベントで次の処理が駆動され，他のモジュールとのやり取りは定められたシーケンスで行うなど，処理の手順を決定する。また，そこで伝えられる情報は動的制御系のような固定長のデータではなく，状況に応じて長さの変わるデータとなる。

4.2　モデリング言語

動的制御系モデルでは，データの流れが主となるため，Mathworks 社の Simulink/Stateflow が制御モデリング言語として自動車では標準的に用いられているが，情報制御系では，処理の手順が主となるため，状態遷移，メッセージのやり取りを定義できる，オブジェクト指向開発で使われる OMG UML（Unified Modeling Language）をシステム設計に拡張した OMG SysML（System Modeling Language）を採用した。Fig. 5 に SysML で（a）状態の遷移を表す状態機械図，（b）処理の流れを表すアクティビティ図の一例を示す。



(a) State Machine Diagram



(b) Activity Diagram

Fig. 5　Example of SysML Model Representation

4.3　モデル規約

OEM であるマツダは車に求められる要求からシステム要件を定義し，それをアーキテクチャ設計と検証をしながら，ユニット，部品に要件を分解／詳細化していくシステム設計に適したモデリングを行う。一方，サプライヤーでは，品質がよく，少ない計算リソースで効率的に処理を実行するソフトウェアソースコードを生成することのできるモデルが必要となる。

SysML でも，システム設計に記述するモデル要素と，効率的なコード生成を行うモデル要素が異なるため，両目的で共通に使うモデル記述を定義し，システム設計からコード生成まで利用できるモデル要素を定義したモデル規約を定めた。このモデル規約を OEM とサプライヤーでモデル交換ガイドラインとして定め，これに従ったモデルの作成と，5 章以降に述べるモデル変換ツールの開発を行った。

5. モデル自動変換

5.1　モデル変換概要

4 章で定義したモデルを後工程で利用するために，①シミュレーションのためのモデル変換（SysML モデルか

ら MATLAB Simulink モデルへの変換）と②実装コード生成のための変換（SysML モデルからモデル交換ガイドラインに準拠した SysML モデルへの変換）の 2 つの変換と，変換に伴うモデルの振舞いの等価性確認を行った。Fig. 6 にシステムアーキテクチャ設計にて実施したモデル変換と検証の概要を示す。

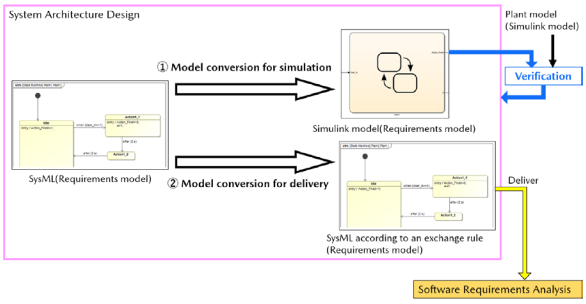


Fig. 6　Overview of Model Transformation

5.2　検証用モデル変換と等価性確認

シミュレーションのためのモデル変換（SysML モデルから MATLAB Simulink モデルへの変換）について述べる。

SysML モデルから Simulink モデルへ変換できないモデルの要素に対しては，変換可能なモデル要素を用いた SysML モデルを生成できるモデリングルールを作成した。Table 1 にモデリングルールの一例を示す。

Table 1　Example of Modelling Rules

Modeling rule	Reason
When plural transition exists from one state of the state machine diagram, Define the transitional priority on SysML.	Because a drive form of the simulation is different from SysML model in the Simulink model, a turn may be mixed up about the transition that a transition turn is not defined.
Use JavaScript for a programming language.	Because, by the program by JavaScript, We were able to confirm that simulation changed definitely.
Do not use Signal Event for a transition condition and use Change Event or Time Event.	Because, in Signal Event, the conversion by the conversion function of the SysML modeling tool is impossible.

作成したルールに従い SysML モデルを構築すると同時に，変換前後での等価性を確認するためのシミュレーションを実行した。Fig. 7 に実行した SysML モデルを示す。ここでのテストケースの SysML モデル（Fig. 7 中の左側）とは，例えばユーザー操作の様な要求の SysML モデル（Fig. 7 中の右側）を実行するための一連の入力である。

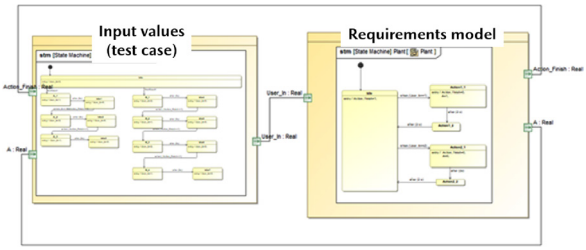


Fig. 7　SysML Simulation Execution Model

Table 1 に示すモデリングルールでは対応できなかった一部の変換については Simulink モデル変換後に手動でモデルを修正する必要があった。Table 2 にモデルの修正例を示す。

Table 2　Examples of Manual Modifications After Simulink Model Transformation

Modified item	Contents	Method to Modify
Evasion of the Algebraic Loop	The concept of the Algebraic loop does not exist by the simulation of SysML.	Insert a delay block in the point that is an Algebraic Loop.
Correction of the garbled text	The converted Simulink model may include "null" in a description.	Delete "null".
Designation of the model step time	When the simulation step time of the Simulink model after the conversion is longer than time of Time Event setting in SysML model, it is for an error at the time of simulation.	Set below the simulation step time of the Simulink model at time of Time Event.

ここまでの手順で，要求とテストケースの SysML モデルを Simulink モデルに変換し，シミュレーション可能となった。次に，要求の SysML モデルと Simulink モデルの等価性を確認するため，同じテストケースを用いて Simulink モデルのシミュレーションを実行し，SysML モデルのシミュレーション結果との比較を行う。この比較には MATLAB Simulink TEST を用いた。Simulink TEST では，対象モデル，入力，評価する出力，評価基準を登録したテストを作成し実行することで，入力に対して指定した出力が評価基準を満たしているかを自動判定できる。そこで，対象モデルを変換後の Simulink モデル，評価基準を SysML モデルのシミュレーション結果との各値の変化の順番の一致とし，全てのテストケースに対して出力が評価基準を満たしていれば変換後の Simulink モデルは SysML モデルと等価であるといえる。Fig. 8 に Simulink TEST による評価結果を示す。等価性があれば Fig. 8 中赤枠の表示が緑色になる。

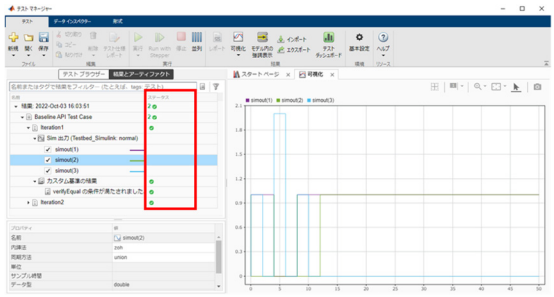


Fig. 8　Evaluation Results from Simulink TEST

ここまでのモデル変換及び等価性確認における SysML モデルの構築ルールや Simulink モデルでのモデル修正については人手によるミスの混入を回避するためにそれぞれ SysML モデルを作成する Dassault Systems 社の Cameo Systems Modeler（以降，CSM）のプラグインと MATLAB プログラム（m スクリプト）により自動修正を行う。以上でシミュレーションのためのモデル変換は完了であり，他の車両モデルと接続した MILS シミュレーションやモデル検査による検証が可能となる。

5.3　実装コード生成用モデル変換と等価性確認

次に実装コード生成のためのモデル変換について述べる。

これは要求の SysML モデルを 4 章で述べたモデル交換ガイドラインに沿った SysML モデルに変換する。Fig. 9 に変換の一例を示す。これは Change Event（値の変化をトリガとするイベント）を Signal Event（信号の受信をトリガとするイベント）に変換する例である。本変換による SysML モデルの検証は 5.2 節で述べた変換と同様の手順で実施した。具体的には Excel VBA を用いたテスト結果の比較ツールを構築し変換前後の SysML モデルの等価性を確認した。このモデル変換と等価性確認により実装コード生成可能な SysML モデルへの自動変換が可能といえる。

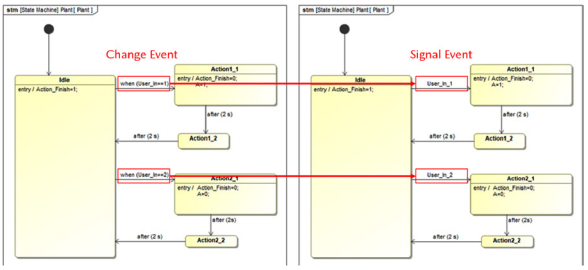


Fig. 9　Example of Model Transformation for Implementation Code Generation

6. MILS-SILS 統合検証環境

6.1　MILS-SILS 統合検証環境の概要

3.1 節で述べたとおり，従来の開発では，ソフトウェア詳細設計の結果であるソフトウェアソースコードが ECU に実装された後に検証が実施されるため，万が一要求，要件に間違いがあった場合，大きな開発の手戻りを発生させてしまう。

ここでは生成後のコードの検証環境である，MILS と SILS（Software in the Loop Simulation）を統合した検証環境（以下，MILS-SILS 統合検証環境）について述べる。

ECU への実装前のソフトウェアソースコードを SILS によるソフトウェアの単体検証の完了後に，MILS-SILS 統合検証環境を構築し，実装前に要求，要件の検証を行う方法を構築した。Fig. 10 に MILS-SILS 統合検証環境の概要を示す。

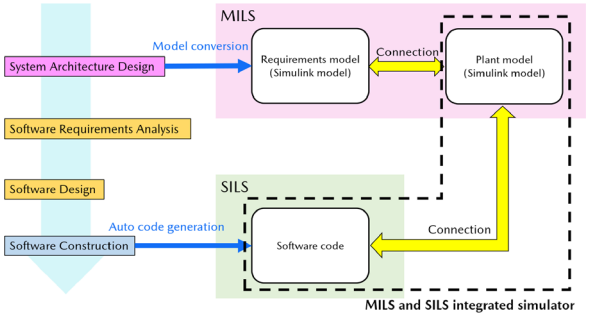


Fig. 10　MILS-SILS Integrated Simulator

6.2　MILS-SILS 統合検証環境の詳細

MILS-SILS 統合検証環境を構築するに当たり問題になったのが，プラントモデルの MILS と情報制御モデルの SILS の検証環境としての性質の違いである。

MILS は，自動車のエンジン制御の検証にも用いられており，熱や運動に関する連続系モデルを周期的な時間同期でシミュレーションを駆動するという性質をもつ。一方 SILS は，情報処理領域を離散事象系としてモデル化しており，非同期でシミュレーションを駆動するという性質をもつ。MILS，SILS を連携するに当たり，同期，非同期で駆動される両検証環境の信号を相互変換するインターフェース（以下 MILS I/F）を構築し，この MILS I/F を介して両検証環境を連携させた。Fig. 11 に MILS I/F の概要を示す。今回，MILS I/F を TCP（Transmission Control Protocol）通信により実現し，MILS と SILS の信号の相互変換と受け渡しを行った。

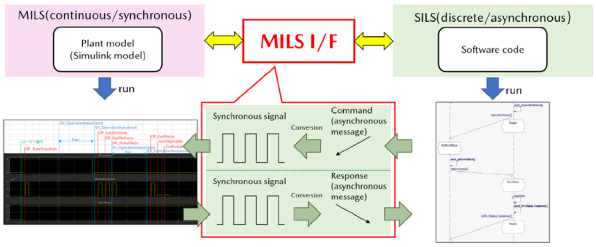


Fig. 11　MILS I/F

7. おわりに

OEM／サプライヤー間での仕様の曖昧さによる手戻り工数増大に対し，曖昧さの要因となる文書ベースの仕様書をモデルに置き換え，開発工程の各段階で仕様をモデルで詳細化し，検証して後工程に品質の高い仕様を提供することを可能とする，ソフトウェア開発へのモデル適用を行った。

OEM／サプライヤー間で，ツール間でのモデル流通を保証するためモデル交換ガイドラインを定義し，離散事象系（MDD）／連続系（MBD）をつなぐための I/F を定義し，OEM とサプライヤーのツール間を連携させたシミュレーション環境を構築し，シミュレーション実行可能なモデルを共有するなど共通基盤を構築した。本手法を用いて CX-60 以降に搭載されたマツダコネクトのドライブレコーダー連携機能の実装を行った。モデルをシミュレーション実行することでユーザビリティ改善案を仕様に織り込む等，特に SW 詳細設計，実装，テストでの工数削減の効果があった。

さらなる効率化のためには，モデル開発環境の整備や，業界内でのモデルの標準化，モデル設計・ソフトウェア設計を推進する人材育成が必要と考えている。

なお，本技術開発は，パナソニックオートモーティブシステムズ(株)と共同で開発したもので，関係諸氏にお

礼を申し上げる。

参考文献

（1）末冨隆雅，齋藤雅彦：OEM／サプライヤ間に跨るモデルベースシステムズエンジニアリングの実践と評価，2022 年度 MBD 推進センター（JAMBE）年度末報告会（2023/3/15）
（2）臼田ほか：SKYACTIV の MBD 検証環境について，マツダ技報，No.31，pp.48-53（2013）

■著　者■


白 雪峰


本城 創


末冨 隆雅