組込みソフトウェア開発課題への挑戦 ～統合化～

渡辺 政彦†

組込みシステムの規模増大により、コードに代わってモデルが多用されるようになってきている。様々なドメイン、工程で作成されるモデルが散乱することは望ましくない。そこでモデルの統合化をどうすべきかを提言する。本提言では次の３つの統合化モデルについて述べる。（１）制御系モデルと組込み系モデル　（２）構造モデルと振舞モデル　（３）バリエーションモデルと状態遷移モデル

Challenge to embedded software development problem -Integrated Modeling-

MASAHIKO WATANABE†

The model comes to be used in place of the code because of the scale increase of the embedded system. It is not preferable to scatter the models developed in various domains and the processes. Then, it proposes how to be integrated of the models. The following three models of integration are described in this proposal. (1) Control system models and Embedded system models (2) Structural models and Behavior models (3) Variation models and State transition models

1. はじめに

組込みソフトウェアの規模が急拡大している（図1）[1]。規模の拡大に対するソフトウェア工学がとるアプローチの１つが抽象化である。組込みソフトウェアの規模が大きくなり、マイコンが解読する機械語から、人が解読できるアセンブラ言語へ抽象度を上げた。さらにソフトウェアの規模が拡大すると、アセンブラ言語から高級プログラム言語であるC、C++、そしてJAVAなどへ抽象度を上げてきた。そして、現在、高級プログラム言語で記述する行数が急増している。

このため、さらなる抽象化として登場するのがモデルである。モデルとは現実世界の問題領域を抽象化し、なんらかの記述体系で表されたものである（図2）[2]。大辞林で抽象の意味をひくと、『事物や表象を、ある性質・共通性・本質に着目し、それを抽（ひ）き出して把握すること。その際、他の不要な性質を排除する作用（＝捨象）をも伴うので、抽象と捨象とは同一作用の二側面を形づくる』とある。

すなわち、組込みシステムアーキテクチャの問題領域から構造の性質を抽出し、なんらかの記述体系で示されたものが「構造モデル」となる。

構造モデルを表現する何らかの記述体系には、タスク関連図、モジュール関連図、そしてブロック関連図などがある。構造モデルは構造に対する性質・共通性・本質に着目し、振舞いに対する性質を排除する。

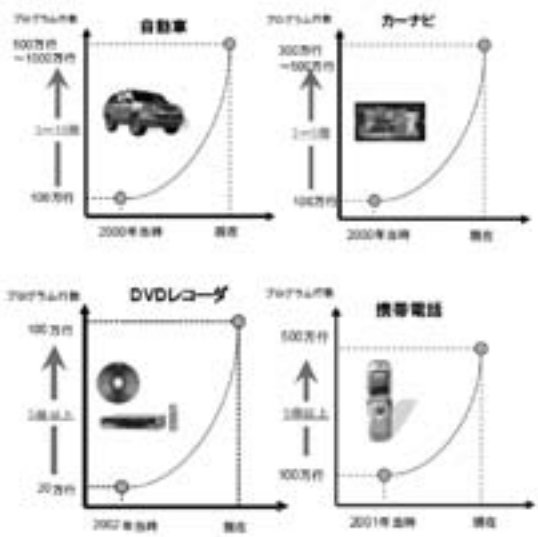

図１　急拡大する組込みソフトウェア

† キャッツ組込みソフトウェア研究所、 CATS Embedded Software Laboratory

また、振舞いモデルを表現する何らかの記述体系には、状態遷移図、状態遷移表、シーケンス図、そしてタイミング図などがある。振舞モデルは振舞いに対する性質・共通性・本質に着目し、構造に対する性質を排除する。

さらに、組込みシステム要求仕様の問題領域からバリエーションの性質を抽出し、なんらかの記述体系で示されたものがバリエーションモデルとなり、フィーチャー図などの記述表現がある。

組込みシステムの問題領域から抽出すべき性質はたくさんあるので、上記の3つだけでなく、数多くの種類のモデルが必要となるであろう。

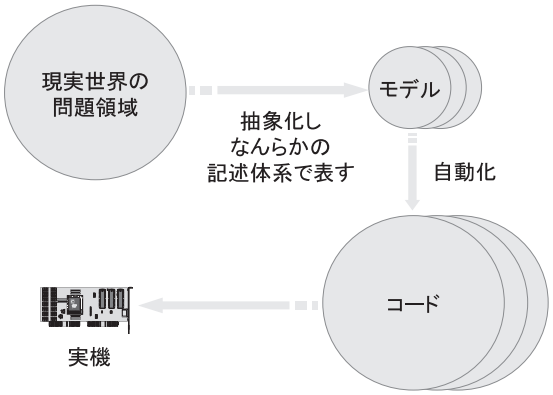

図2　モデルとは

2. 制御系モデルと組込み系モデル

2.1. 位置付け

本稿における制御系モデルと組込み系モデルの位置付けを図3に示す。当然ではあるが、図に示すモデルは1つの例であり、対象とする問題領域ごとにモデルの構成は変わる。

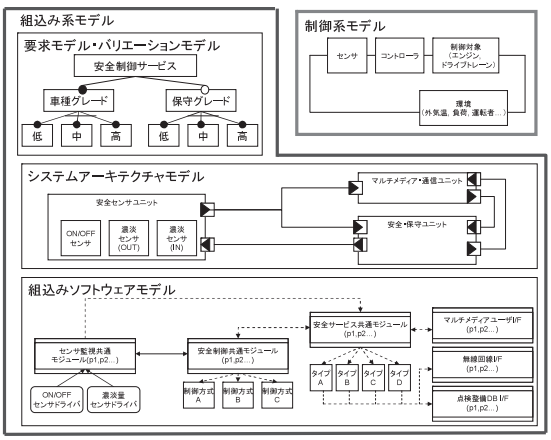

図3　制御系モデルと組込み系モデル

制御系設計では、モデルは一般的にプラントモデルを示す［3］。本稿では制御対象を表すものをプラントモデル、制御則を表すものをコントローラモデルと呼ぶ。制御系設計では、制御対象、制御目標、制御量を明確にしたのち、どのようなセンサとアクチュエータをどこに配置するかを検討する。

組込み系モデルは、上流工程の要求から下流工程の実装まであるプロセスのアクティビティごとにモデルが存在する。

どのような製品を開発するかが要求モデルに記述される。例えば自動車や家電機器には、多品種多オプション製品の仕様がある。これを上手く整理しないと、似て非なるソフトウェアを大量に開発しなければならない。多品種多オプションの問題領域を抽象化するのがバリエーションモデルである。

近年の組込みシステムはネットワークやバスによって構築される分散システムであることが多い。このため、アーキテクチャの良し悪しが、システム全体の性能に影響する。システムの物理構造性質に着目したモデルをシステムアーキテクチャモデルと呼ぶ。

物理構造とは切り離して、機能やサービスを抽象化したものが論理構造モデルである。コンポーネント、オブジェクト、プロセス、スレッドもしくはタスクなどの配置を示すものが組込みソフトウェアモデルである。

2.2. 課題

現在、組込み系モデルが抱える課題の1つに、モデル間における相互作用の複雑化がある。

本来、モデルは、機能独立性を有しているものである。機能独立性とは1970年初頭からParnas、Wirth、Stevens、Myers、そしてConstantineらの研究により確立された抽象化と情報隠蔽を意味する基本概念である。機能独立性とは、モジュールを「ただ1つの目的を持った」機能を実現するように開発し、他のモジュールとの過度の相互作用を「回避」することによって実現される［4］。

機能独立したモデルを組合せることで、システムの機能や振舞いを実現するのである。さらにシステムは固定化されたものではなく、常に進化する。

このため、どんなに機能独立性を高めても、より高度なシステムを実現するために、独立性を高めたモデルを複雑に相互作用させることになる。

自動車では車載LANを伝送路として、機能独立性が高いECUが相互作用し、統合安全機能などの高次元なサービスを提供する。家電機器も同様にホームネットワークに接続することで、独立していた家電機器の機能に相互作用を起こし、新しい付加価値を創出する。

組込みシステムでは、高付加価値のある機能の提供に加えて、低コストでそれらを実現することが求められる。そのためにモデルの個別最適化からシステム全体からの最適化によって個別のモデルを洗練する必要がある。

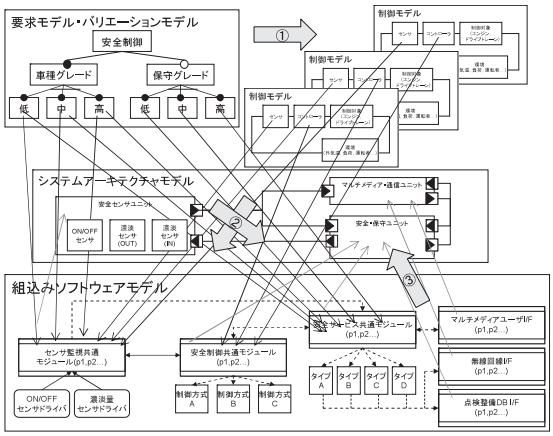

図４　モデル間相互作用

本稿で提言するモデル間で起こる3つの相互作用を図4に示す。

(1) バリエーションモデル－制御系モデル
(2) バリエーションモデル・制御系モデル－組込みソフトウェアモデル
(3) 組込みソフトウェアモデル－システムアーキテクチャモデル

モデルの種類はたくさんあるが、上記3つに関する相互作用課題への解決方法は、モデルの種類に関係なく広く適用できると考える。

2.2.1. バリエーションモデル－制御系モデル

単一の制御系モデルから、多品種多オプションの制御系モデルへの変化は、バリエーションモデルと制御系モデルとの間で起こる相互作用の管理が課題となる。

バリエーションモデルと制御系モデルの相互作用について図5に示す。車種グレードを3種類、「低」、「中」、「高」と設定することで、それぞれの車種グレード向けの制御系モデルを作成する。

「低」車種グレードでは、制御はON/OFFのみのフィードバック制御で、制御粒度は荒く、制御速度は遅くて良いが、実現コストは安価であることが設計条件である。「中」車種グレードでは、制御は濃淡と量のフィードバック制御で、制御粒度は細かいが、制御速度は遅くて良く、実現コストは高価ではないことが設計条件である。「高」車種グレードでは、濃淡量センサを採用し、センサからフィードバック制御することで、濃淡の量の制御で、制御粒度は細かく、かつ制御速度は速いが、実現コストは高くても可である。

制御系モデルで複数のバリエーションを扱うことは、制御系モデルが複雑化することを意味する。一方で、バリエーションをバリエーションモデルで扱い、制御系モデルは、あくまで単一バリエーションにすることで、制御系モデルの複雑化を防止できる。この場合、バリエーションモデルと制御系モデルの間に依存関係が生じ、この関係をどのように管理するかが課題となる。

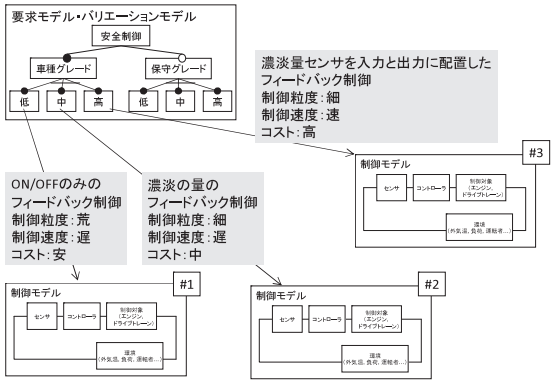

図５　バリエーションモデル―制御系モデル

2.2.2. バリエーションモデル・制御系モデルー組込みソフトウェアモデル

組込みソフトウェアモデルには、下流工程に実装を意識したモデルがある。一般的に実装モデルでは、ROM/RAMリソース削減や、テスト工数削減、保守性向上のために、モジュールの共通化を行う。実装上の制約から制御系モデルの構成要素が、組込みソフトウェアでは同じ構成とはならない場合が多くある。このため、制御系モデルと組込みソフトウェアモデル間の依存関係が複雑化し、追跡可能性が取り難くなる。同様に、バリエーションが実装モデルの広範囲に影響を与えることから、バリエーションモデルと組込みソフトウェアモデル間の依存関係が複雑化し、追跡可能性が取り難くなる。

バリエーションモデル・制御系モデルと組込みソフトウェアモデルの例を図6に示す。コントローラモデルは、プラントモデルを制御対象として、制御目標を達成するための制御量を計算する。組込みソフトウェアでは、コントローラモデルの機能に加えて、センサへのアクセス方法、故障の監視や故障時の対応などが含まれる。このため、制御系モデルで記述された１ブロックは、組込みソフトウェアモデルでは複数のモジュールに分散することが多い。

同様にバリエーションモデルは、コード一元管理下、 #ifdef文がコード中に散在する。構造化プログラミングによって、goto文スパゲティ構造を駆逐したが、#ifdef文スパゲティ構造が進行しているようだ。このため、コードレベルの#ifdef文によるバリエーション対応から抽象度の高いモデルレベルでのバリエーション対応が期待される。そこで、バリエーションモデルと他のモデルとの相互作用の明確化とその管理方法が課題となる。

2.2.3. 組込みソフトウェアモデルー システムアーキテクチャモデル

制御系設計では、コントローラモデルが安定した制御をするための制御則を設計する。制約を充足する計算式を算出するための開発プロセスは、イテレーティブ開発とインクリメンタル開発のどちらかまたは融合した開発プロセスとなる。

このプロセスを支援する開発環境に、MILS（Model In the Loop Simulation）とHILS（Hardware In the Loop Simulation）がある。MILSやHILSで、理想的な制御則を求めたとしても、あくまでも理論値であり、実装値ではない。制御系のシミュレーションは浮動小数点で行われるが、組込み系では、製品コストの制約から浮動小数点を固定小数点になる場合が多い。また、計算式のCPUリソース占有率が、故障診断等の制御系以外の機能実行に影響あるかを検討する。このため、組込みソフトウェアモデルからシステムアーキテクチャモデルにソフトウェアとハードウェアの最適配置の見直し要求という相互作用が起きる。

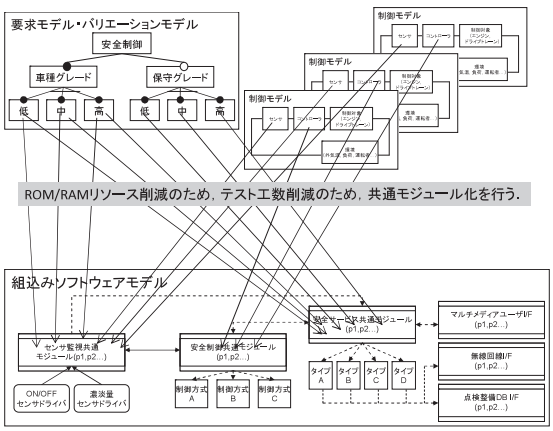

図６　バリエーションモデル・制御系モデル―組込みソフトウェアモデル

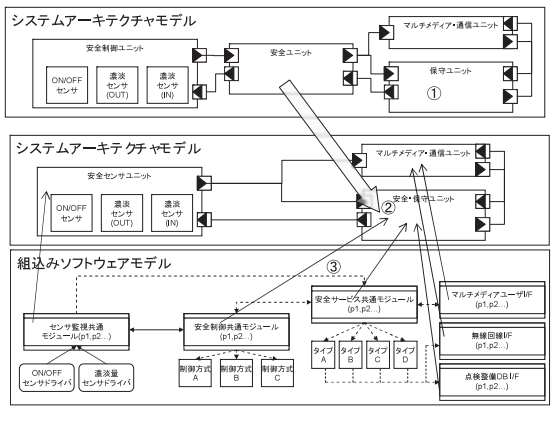

図７　組込みソフトウェアモデル―システムアーキテクチャモデル

例えば、保守ユニット（図7-①）がメンテナンス時にしか動作しないことから、安全制御機能を安全・保守ユニット（図7-②）に搭載し、ECUの数を減らすことでコストを削減する。またセンサ系に制御ロジックを搭載しないことで安全センサユニットのコストも削減する（図7-③）。

2.2.4. モデル間相互作用

制御系モデルと組込み系モデルを少数で開発していれば、さほど問題にはならなかったが、近年、数十人から数百人規模で開発することが多くなってきている。こうなると、開発の成果物であるモデル間の相互作用が混沌とした状況に陥る。

複数の問題領域における相互作用問題の例を図8に示す。

(1) 要求モデル：
- 車種グレードに超低価格と超高級を入れたいが、誰に何をどう伝えるのか（図8-①）

(2) 制御系モデル：
- 新しい制御則（精度がでるが速度はやや遅い）を採用したいが、誰に何をどう伝えるのか（図8-②）

(3) システムアーキテクチャモデル：
- センサのスペックが変更されたが、誰に何をどう伝えるのか（図8-③）
- 競合がもっと安く、かつ機能が高いものを出した。どこを直すか即時検証をしたいが、誰に何をどう伝えるのか（図8-④）

(4) 組込みソフトウェアモデル：
- ここをもっとこうすれば、メモリを大量に使わずに済むのに、誰に何をどう伝えるのか（図8-⑤）
- バグがでた。誰に何をどう伝えるのか（図8-⑥）

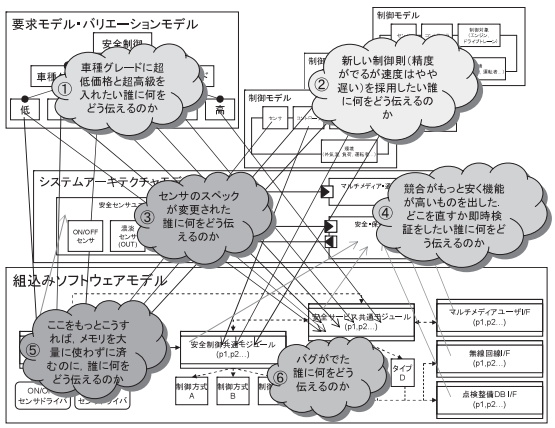

図８　モデル間その他の相互作用

2.3. 解決方法

モデル間の相互作用問題を解決するためには、モデル構成要素間の関連を把握できる仕掛けが必要である。モデル構成要素は、制御系モデルの場合は各ブロックである。システムアーキテクチャモデルでは、例えば車載用コンポーネントベース開発であるAUTOSAR［5］を採用した場合、SWC（ソフトウェアコンポーネント）モデルの各コンポーネント、トポロジーモデルの各ECUとなる。

モデル構成要素間の関連を把握できる仕掛けを実現する機構がリポジトリである。一般的にリポジトリの役割は広範囲で、モデルデータとツールの統合や、モデル同士の統合を実現するメカニズムとデータ構造から構成される［4］。

モデルデータとツールを統合するためにはメタモデルといった概念が必要になり、OMGのQVT（Query/View/Transformation）［6］、ISO/IEC 15474-1（2002）のCDIF（CASE data interchange format）フレームワークがある。相互作用問題を解決するには、モデルの交換形式より、モデルの依存関係を追跡可能性が必要になる。モデルの依存関係に特化したリポジトリを依存関係リポジトリと呼び、広義なリポジトリと区別する。

依存関係リポジトリを図9に示す。依存関係リポジトリと要求管理ツールとは、追跡性管理において似ている。本稿における提言は、統合するモデルを扱うツール同士を統合し、設計対象が異なるツールでも、同じ操作感でモデル構成要素の依存関係をリポジトリに登録できることである。これにより、膨大なモデル間の相互作用による依存関係を設計作業中に登録できる。設計完了後に相互作用の依存関係をあらためて

登録することは実務上難しいと考える。設計時にリポジトリに入力できることで開発者の作業負担を軽減できる。

統合化の対象となるツールは数多くあるがが、例えば、制御設計を支援するツール（CACSD: Computer Aided Control System Design）とソフトウェア工学を支援するツール（CASE: Computer Aided Software Engineering）を統合化し、依存関係リポジトリを構築することで、両領域モデルで起こる相互作用の煩雑さを低減する。

統合化CASEツールを4年間利用した場合、1ドルあたりの投資対効果は25ドルになるというデータがある［7］。CACSDツールとCASEツールとの統合化は、さらなる投資対効果を期待できる。

CACSDツールとCASEツールの詳細な連携については、別稿で述べる。

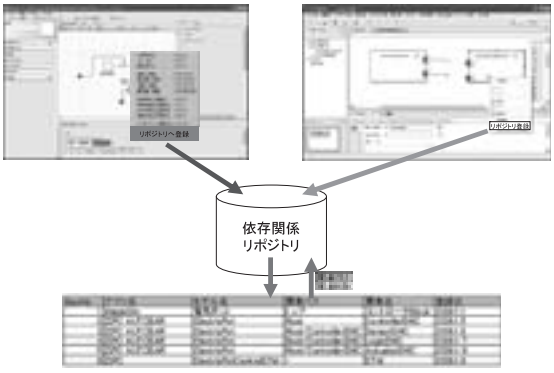

図 9　依存関係リポジトリ

3. 構造モデルと振舞モデル

構造モデルは、実現すべき複雑なシステムから、構造だけを抽出する。そうすることで、複雑なシステムの構造に関して、抽象度が上がり、適切な設計対象になる。しかしながら、実現すべきシステムは、構造だけでは動作しない。 機能が「いつ」、「どのようなときに」実行されるかについては、構造モデルでは捨象される。

振舞モデルは、システムから動的な性質を取りだす。動的な性質とは構造モデルで捨象された「いつ」、「どのようなときに」である。「いつ」とは事象であり、「どのようなときに」は状態となり、そして「次のどのような状態に変化するか」は遷移になる。事象、状態、そして遷移を表現する表記法として状態遷移図・表がある。本稿では状態遷移図・表で表記されたモデルを状態遷移モデルと呼ぶ。

組込み系モデルで良く用いられる表記法は、状態遷移図、フローチャート、状態遷移表、タスク関連図、そしてタイミングチャートである（図10）［8］。エンタープライズ系における同様な調査では、組込み系で下位に位置するUMLとDFDが上位になる［9］。組込み系では上位に位置する状態遷移図、状態遷移表、そしてタイミングチャートは、エンタープライズ系ではその他の設問に置かれる扱いで、かつその使用率は非常に低い。

どのようなモデル表記法を利用するかで、組込み系とエンタープライズ系の違いが分かるようで興味深い。最近のエンタープライズ系の不具合は、状態遷移モデルによる分析、設計をしていないからなのだろうかといった推測をしたくなる。

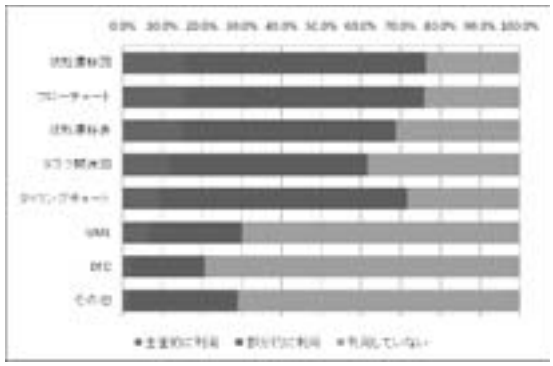

図 10　モデリング手法の利用状況

3.1. 課題

異なるモデル間での依存関係を管理する方法として2章で述べたリポジトリがある。モデルベースツール間を統合し、依存関係リポジトリへの登録方法を効果的にする提言をした。しかしながら、登録する依存関係そのものが属人的で、開発者によってその依存関係の定義が多種多様な混沌とした依存関係を管理することになる課題がある。

3.2. 解決方法

構造・振舞モデルの相互作用、および機能・振舞モデルの抽象度に関する相互作用に対して、設計手法を設けることで、各モデルにおける依存関係を整理することを提言する。

設計手法により、属人性を排除することは、モデル依存関係を自動的にリポジトリで管理する統合化モデルベース開発環境の構築に寄与する。

3.2.1. システム

システム振舞モデルとシステム構造モデルの相互作用を図11に示す。

システムの振舞いを状態遷移モデルで記述したシステム振舞モデルが用意されている。システム振舞モデルからアクションとアクティビティリストを取りだし、アクション・アクティビティリストを作成する（図11-①）。列挙したアクションおよびアクティビティを「いつ」、「どこで」呼び出し、その後「どうなるか」といった有限状態機械を動作させるFSMコントローラを用意する。

機械的に抽出されたアクション・アクティビティリストのアクションやアクティビティに対して共通化や洗練化を行い、システム構造モデルを導出する（図11-②）。モジュールの共通化、洗練化によって、実装時のリソース効率性、試験容易性、保守性等を向上する。

機能の依存関係は結線で示す。機能の実行順位や呼出関係が明確であれば、矢印を用いて示す。例では、3つのモジュールがFSMコントローラから呼び出される関係で、それぞれのモジュール間の依存関係をなくした機能独立性の高い構造となる。

システム構造モデルで作成されたモジュールをシステム振舞モデルに再配置する（図11-③）。これにより、システム構造モデルで洗練された機能モジュールとシステム振舞モデルとの依存関係が明確化できる。

アクション・アクティビティリストとシステム構造モデルとを対応付ける作業を手作業で行う必要はあるが、それができれば、後はシステム振舞モデルとシステム構造モデルとの依存関係を自動的にリポジトリに登録することが可能である。

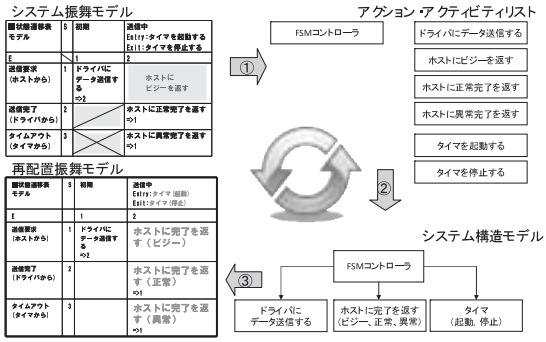

図 11　システム　振舞ー構造

3.2.2. システムーモジュール

システムとモジュールという抽象度の異なるモデル間における「振舞－構造」の相互作用を図12に示す。

システム構造モデルの構成要素であるモジュールごとに、振舞モデルを作成する。作成したモジュール振舞モデルから先ほどと同様に、アクション・アクティビティリストを作成する（図12-①）。抽出したアクションおよびアクティビティを有限状態機械として動作させるFSMコントローラを用意する。

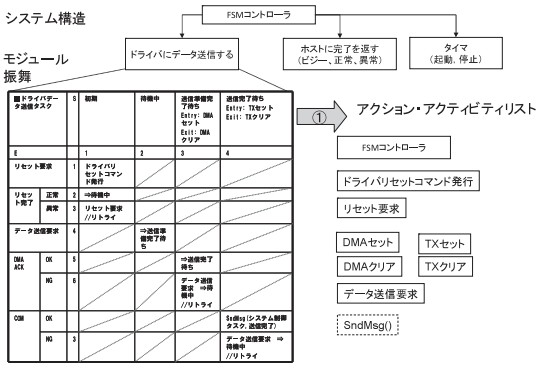

図 12　システムーモジュール

モジュール振舞モデルから機械的に生成できるアクション・アクティビティリストに掲載されたモジュールに対して共通化を行い、実装用モジュールを導出する。 実装モジュール構造モデルで作成したモジュールをモジュール振舞モデルに再配置する（図13-①）。

ここまで、システム振舞モデルから取り出されたアクション・アクティビティが、システム構造モデルのモジュールとなり、そのモジュールの振舞モデルから実装用のモジュールを導出できることを示した。

実装モジュール構造モデルに示されるモジュールには、ミドルウェアや再利用モジュールなどが配置される。外部モジュールは破線で示す。図12、13のSndMsgがその例である。

実装モジュールは、プログラミングによってプログラムコードを作成する（図13-②）。実装モジュールの規模がまだ大きければ、さらにモデリングを行い、その後、プログラミングか自動コード生成を行い、プログラムコードを作成する。

事象を解析し、当該状態下でモジュールを呼び出し、状態を遷移させる有限状態機械としてのメカニズムであるFSMコントローラのコードを自動生成することが可能である（図13-③）。

自動生成コードにより生産性の向上の他に品質の向上についても期待する声は大きい。JEITAでの自動生成コードに関する調査では、図14に示すように高い評価結果となっている［10］。

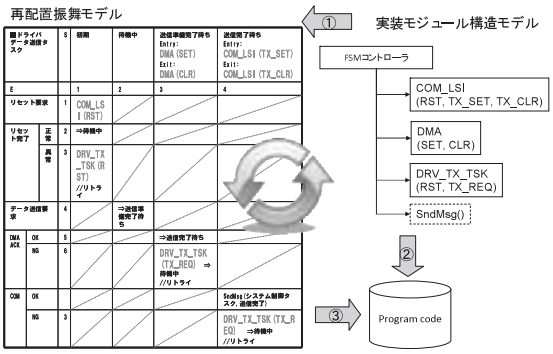

図 13　システムーモジュール　振舞ー構造

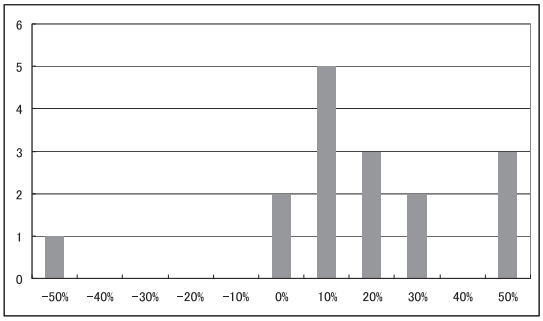

図 14　ソースコード自動生成ツールによる品質向上率

3.2.2.1. FSMコントローラ

FSMコントローラをシステムごと、モジュールごとに用意するのか、１つで共通化するかは、アーキテクチャ設計で決定する。 FSMコントローラをイベント待ちにして、イベント駆動型アーキテクチャにする。FSMコントローラをmain関数として、永久ループ構造から各モジュールを呼び出すアーキテクチャにする場合もある。FSMコントローラを状態スケジューラ［11］とすることで、並列状態をシングルコアで実現したり、マルチコアで実現したりするメカニズムを局所化することができる。FSMコントローラを必要としない場合もある。状態遷移表モデルでアクションが斜めに配置される場合（図15-①）、アクションはシーケンシャルに実行されることが多い。 この場合、各モジュール間で呼出関係を定義することで、FSMコントローラは不要となる（図15-②）。

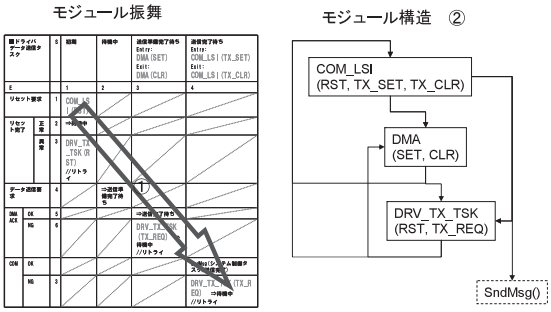

図 15　FSMコントローラ不要アーキテクチャ

3.2.3. 組込み系と制御系

組込み系モデルと制御系モデルの依存関係を明確にする設計方法を定義することで、モデル間の相互作用を自動的にリポジトリに登録できると考える。では、組込み系モデルと制御系モデルの依存関係における設計方法論とはどのようなものであろうか。

組込み系モデルから制御系モデルを呼び出す方式を図16に示す。近年、システムの複雑化に伴い、制御系モデル内部にある状態遷移モデルが複雑化している（図16-①）。

そこで、制御系の状態遷移モデルの大部分を組込み系モデルに持たせて、制御系モデルが持つ状態遷移モデルを極力単純化する。制御系モデル内部にある状態遷移モデルを外部の組込み系モデルの状態遷移モデルと融合させる（図16-②）。当然ながら、制御系の全ての状態遷移モデルが外部に出せることはない。

組込み系モデルの状態遷移モデルから制御系モデルを呼び出す構造にする（図16-③）。こうすることで、システムの変化を組込みモデル側に持たせて、制御系モデルの複雑化を防止する。4章で紹介するプラントテーブル方式がこれに近い方式である。

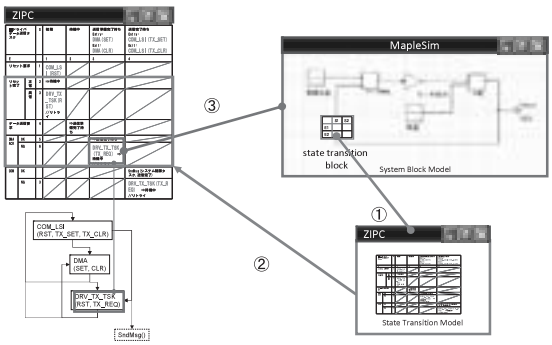

図 16　組込み系ー制御系

もう１つの方式は、制御系モデルから組込み系モデルを呼び出す方式である（図17）［12］。

コントローラモデルは、ECU上で実行される組込みソフトウェアとして実現される。コントローラモデルはプラントモデルを制御対象として、制御目標を達成するための制御量を計算する機能である。

しかしながら、組込みシステムとしては、故障時の対応などが要求される。このため、組込みソフトウェアには、故障を検知し、処理するためのウォッチドッグ監視機能（図17-①）に、それを実現するためのタイマやUI制御のためのIO制御機能（図17-②）が必要になる。機能の配置は静的モデルで示し、機能の振舞いは動的モデルで示す。静的モデルではタスク関連図やコンポーネント図など、動的モデルでは状態遷移モデルやアクティビティ図などによって表記される。

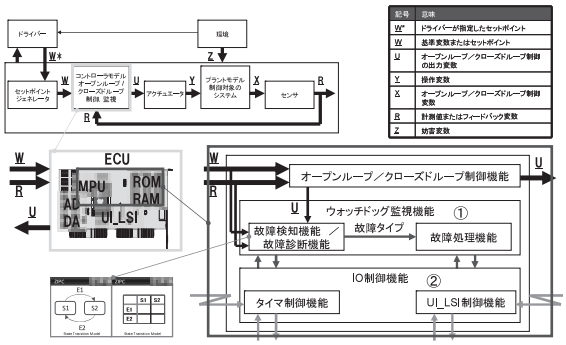

図 17　制御系ー組込み系

構造モデルや振舞モデルに関する表記法の統一は、UMLによって進められている。しかしながら、多種多様な問題領域をモデリングするには問領域に特化した言語が必要としてDSLが出現した。このため構造モデルではUML以外に、EAST-ADL、SysML、MARTEなどが出現した。AUTOSARでは、ハードウェア（物理）構造モデルであるトポロジーモデルにソフトウェア（論理）構造モデルとの関係や、バスに流れるシグナル設計を行い、後工程を支援するECUコンフィグレーションツールへ構造モデルのファイル（ARXML）を渡す設計手法を示す（図18）。しかしながら、UMLや他のDSLと同様にAUTOSARは、振舞モデルと構造モデルに関する相互作用や依存関係ついては定義していない。

各モデルの依存関係を設計手法により明確化することは、はじめは主観的な考えが支配的になる。しかしながら、実績を重ねていくことで、定量的データが蓄積され、より客観的な議論ができる。

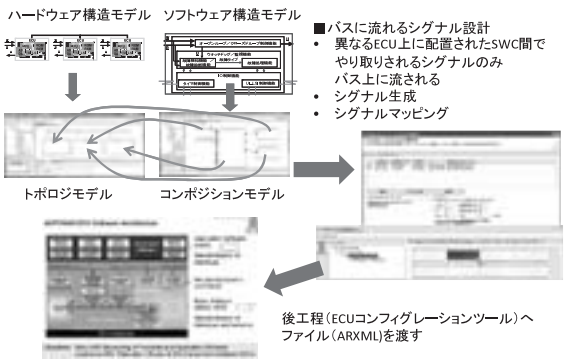

図 18　AUTOSAR

構造モデルと振舞モデルの詳細な連携については、別稿で述べる。

4. バリエーションモデルと状態遷移モデル

組込みソフトウェアを新規に開発することは少ない。既存の組込みソフトウェアに追加、変更を加えていく。こうした開発を派生開発、差分開発または流用開発などと呼ぶ。ソフトウェアプロダクトラインでは共通性・可変性を分析し、追加、変更される部分と、共通利用されるコア資産を蓄積することで、高生産、高品質な開発を実現する。

国内で初のソフトウェアプロダクトライン殿堂入りを果たした東芝のプラントテーブル方式（図19）は、発電所向け監視制御システムの開発に30年以上、国内外150以上納入実績を持つソフトウェアプラットフォーム構築技術である［13、14］。制御系モデルをコア資産化し、可変性の高い部分をテーブルにすることで高生産、高品質な開発を実現する。テーブルは1プラントあたり2000枚から5000枚程度だという。

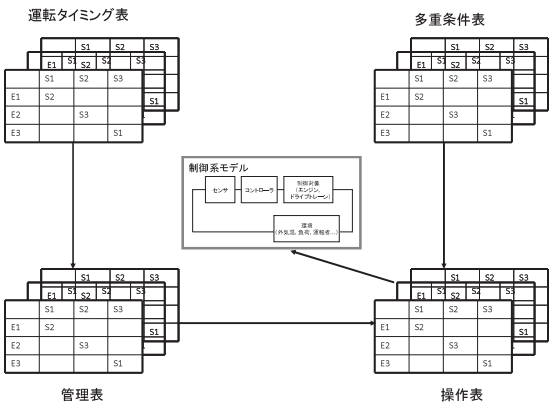

図 19　プラントテーブル方式

4.1. 課題

モデルベース開発において、差分開発をする場合に、可変性をどのように取り扱うかが課題である。C言語であれば、#ifdef文により、ソースコードの一元管理ができる。また、ソースコード差分検出ツールによって、差分箇所を表示することもできる。ところが、モデルにおいては、差分そのものはフィーチャモデルで可視化できるが、フィーチャモデルと他のモデルとの依存関係は明確にされていない。 本稿では、フィーチャモデルと状態遷移モデルに限定して述べる。

4.2. 解決方法

状態遷移モデルにC言語と同様に#ifdef文を採用する。状態遷移モデルの事象、状態、そしてアクションに#ifdef文を記述することで、バリエーションに応じた振舞いを行う状態モデルを確定できる。バリエーショモデルと状態遷移モデルにおける相互作用を図20に示す。

プロダクトのバリエーションをフィーチャモデルで指定し、指定された状態遷移モデルを確定する（図20-①、②）。遷移先や処理の変更はアクションセルに#ifdef文で記述する（図20-③）。

状態遷移モデルに可変性が存在し、状態遷移モデルから呼び出されるモジュールに共通性が存在すると仮定した場合、状態遷移モデルが図19で示したプラントテーブルに相当し、アクション・アクティビティが制御系モデルに相当する。

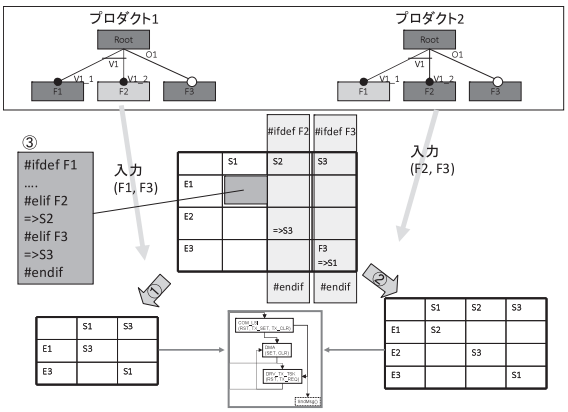

図 20　バリエーションモデルと状態遷移モデル

実際の車載オーディオにおける状態遷移モデルの差分箇所を調査したものが表1である［15］。オリジナルモデルから変更された個所が可変性個所であり、バリエーションとなる。11枚の状態遷移表モデルから構成される状態遷移表モデルのどの部分に差分が起こるかを調査した結果、事象の変更が1か所、状態の変更が1か所、アクションの変更が104か所であった。

この結果から、状態遷移モデルのバリエーションは、事象と状態よりは、アクションに多く

発生することが分かった。

バリエーションモデルと振舞いモデルの詳細な連携については、別稿で述べる。

表１　車載オーディオ状態遷移表差分

	original				update				diff		
stm no	E	S	A	total	E	S	A	total	E	S	A
1	4	26	104	134	4	26	104	134	0	0	3
2	8	17	136	161	8	17	136	161	0	0	6
3	2	3	6	11	2	3	6	11	0	0	0
4	15	37	555	607	15	37	555	607	0	0	31
5	3	32	96	131	3	33	99	135	0	1	8
6	10	27	270	307	10	27	270	307	0	0	41
7	11	20	220	251	11	20	220	251	0	0	9
8	12	9	108	129	12	9	108	129	0	0	0
9	13	12	156	181	13	12	156	181	0	0	0
10	11	19	209	239	11	19	209	239	1	0	6
11	3	8	24	35	3	8	24	35	0	0	0
total	92	210	1884	2186	92	211	1887	2190	1	1	104

5. むすび

モデリング表記法は、UML vs. DSLとなり、SysML、MARTE、EAST-ADL、AUTOSAR、TECSなど数多くのモデリング表記法が生み出されている。このような状況で統合化モデルの重要性は高まるばかりではある。

このため、斬新で、萌芽的なアイディアを地道に実践し、その結果のデータを収集、分析することで　定量化することが重要である。

参考文献

［1］ 経済産業省商務情報政策局情報処理振興課長八尋俊英、組込みソフトウェア産業の課題と政策展開、2008年11月19日 http://imyme.chicappa.jp/ET2008/conference/image/ET2008_S-2.pdf

［2］ 玉井哲雄、ソフトウェア工学の基礎、岩波書店、2004。

［3］ 村松鋭一、山形大学工学部制御工学□テキスト、http://www13.plala.or.jp/control/control1/c1_chapter1.pdf

［4］ Roger S. Pressman著、飯塚悦功・西康晴監訳、ソフトウェア工学の伝統的手法、日科技連、2000。

［5］ AUTOSAR: http://www.autosar.org/

［6］ MOF QVT Final Adopted Specification, 2007, http://www.omg.org/docs/ptc/05-11-01.pdf

［7］ Capers Jones, Assessment and Control of Software Risks, Yourdon Press, 1994

［8］ 経済産業省:平成19年　2008年版組込みソフトウェア産業実態調査＜プロジェクト責任者向け＞

［9］ 小柴　豊、@IT読者調査結果：情報マネージャ／ITアーキテクト編（2）～モデリングの現状と課題とは？～、アットマーク・アイティ、2005/2/26, http://www.atmarkit.co.jp/news/survey/2005/02arcmng/arcmng.html

［10］社団法人　電子情報技術産業協会　ソフトウェア事業員会、平成19年度　ソフトウェアに関する調査報告書Ⅱ　組込み系ソフトウェア開発の課題分析と提言　～大規模化、複雑化、短納期化、他機種化の波にどのように立ち向かうべきか～、2008。

［11］渡辺政彦、拡張階層化状態遷移表設計手法Ver.2.0、東銀座出版社、1998。

［12］ヨーク・ショイフェレ、トーマス・ツラフカ共著、福田晃監修、オートモーティブソフトウェアエンジニアリング～原則，プロセス、手法、ツール～、日刊工業新聞社、2008年10月

［13］滝沢治、赤石富士雄、早瀬健夫、ソフトウェアプラットフォーム構築技術、東芝レビューVol.64、No.4、2009。
［14］中本泰発、河原春郎、前大道正博、小暮洋一郎、阿部正夫、50年形発電所計算機制御システム「COPOS」の開発、東芝レビューVol.29、No.10、1974。
［15］渡辺政彦、福田　晃、中西 恒夫、細谷伊知郎、城戸滋之、“状態遷移表差分抽出技術とツールの提案と評価”，組込みシステムシンポジウム2008（ESS1008）論文集、pp.79-87、2008年10月。

渡辺政彦
1984 年日本大学・文理学・英文科卒。2009年九州大学大学院システム情報科学府博士後期課程終了。
博士（工学）。情報処理学会会員。電子情報通信学会会員。
現在、キャッツ株式会社取締役副社長 兼 最高技術責任者 兼 キャッツ組込みソフトウェア研究所所長、九州工業大学大学院情報工学研究院客員教授。

WATCHERS Back Numberで振り返るユーザ会

「ZIPC誕生20周年記念　第14回　ZIPCユーザーズカンファレンス」では、キャッツの技術者によるワークショップ「～流れでおさえる～　キャッツのプラットフォームベース開発」を開催し、キャッツが考えているプラットフォームベース開発の流れをご説明しました。

基調講演では、財団法人北九州産業学術推進機構 丸田秀一郎先生より「学術研究都市におけるマルチコアプロセッサアプリケーション創出活動」と題し、「マルチコアプロセッサ」の現状、将来などの最新動向をご講演いただきました。また、九州大学 中西恒夫先生より「プロダクトラインとフィーチャモデリング」、国立情報学研究所 田口研治先生より「UPPAAL入門 －モデル検査による実時間システムの検証－」をご講演いただきました。

事例講演では、シチズン時計株式会社様、日本ビクター株式会社様、株式会社日立情報制御ソリューションズ様より適用事例、株式会社デンソー様より教育事例を発表していただきました。