Panasonic Technical Journal Vol. 56 No. 3 Oct. 2010

組込み分野へのモデル駆動型開発手法の適用

Model-Driven Development for Embedded Software Development

高 橋 知 伸* 南 光 孝 彦*
Tomonobu Takahashi Takahiko Nankou

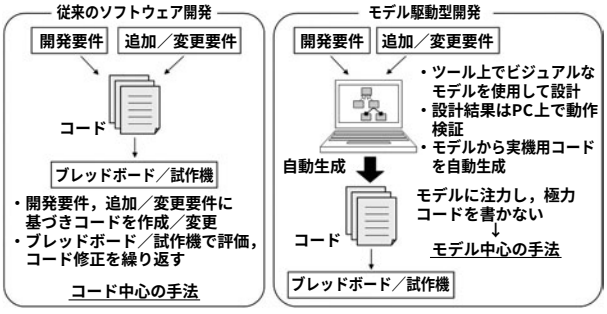
UML（統一モデリング言語）を用いて抽象度の高いソフトウェア設計を行い，作成されたソフトウェア設計図（モデル図）からコードを自動生成するモデル駆動型開発手法を組込み機器ソフトウェア開発へ導入した。視覚的に理解しやすいソフトウェアの図式表現とコード自動生成の効果により，ソフトウェア開発の効率化と品質向上が達成された。

We applied Model-Driven Development to Embedded Software Development for development efficiency and quality improvement. Model-Driven Development has 2 features. The first feature is that software structure is visually designed using Unified Modeling Language. Another feature is that the program code is automatically generated from the model designed using Unified Modeling Language. Visual design and automatic code generation lead to efficiency and quality improvement.

1. モデル駆動型開発

近年，機器のデジタル化・ネットワーク化が進むことにより，組込みソフトウェアの大規模化・多機能化・複雑化が急激に進行し，同時にソフトウェア検証も増大の傾向にある。加えて，開発機種数の増加，開発リードタイム短縮など，更なる開発の効率化が求められている。

このような状況に対する取り組みの1つとして，筆者らはモデル駆動型開発（以下，MDD：Model Driven Developmentと記す）の導入を進めている[1)]。MDDでは，UML（統一モデリング言語）によりソフトウェアを視覚的に記述したモデル図からプログラムコードを自動的に生成する（**第1図**）。従来開発ではコードが開発の中心であるのに対し，MDDはモデル図が開発の中心となる。



第1図 モデル駆動型開発の概要

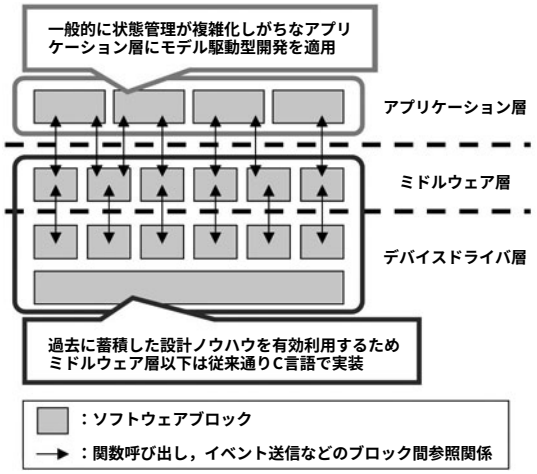
Fig. 1 Overview of model driven development

2. MDD導入のポイント

筆者らがMDDをAV機器の組込みソフトウェア開発に導入するにあたり検討した内容の一部を述べる。

2.1 MDDの適用範囲

筆者らは，システムの状態とイベントの組合せによるソフトウェアの振る舞いを状態マシン図で設計し，その設計内容を自動的にコードに変換できる点がMDDの有効性の1つと考えている。それゆえ，ソフトウェアの多機能化やユーザーインターフェース制御により一般的に状態管理が複雑になりがちなアプリケーション層にMDDを適用することを決定した（**第2図**）。一方，ミドルウェア層以下はこれまで蓄積してきた設計資産，ノウハウを流用することで既存機能の安定動作を保障した。



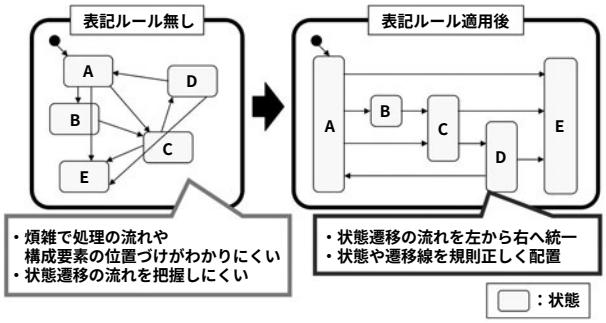
第2図 モデル駆動型開発適用範囲

Fig. 2 Scope of MDD

＊ AVCネットワークス社 技術統括センター
Technology Planning & Development Center,
AVC Networks Company

2.2 モデル表記ルールの策定

1章で述べたとおり，MDDはモデル図中心の開発である。ゆえに，開発者が設計内容や設計意図を理解しやすいモデルを書くこと，保守性を維持することが重要となる。そこで，筆者らはモデル図の属人性を減らすため，モデル図の表記ルールを策定した（**第3図**）。従来開発におけるコーディング規約と同様に，モデル表記に統一性を持たせることでモデルの可読性や保守性の向上を狙った。



第3図 モデル表記ルールの適用例（状態マシン図）

Fig. 3 Modeling rule example (State machine diagram)

3. MDD導入結果の考察

MDDをAV機器の組込みソフトウェア開発に導入し，その効果の検証を行った（**第1表**）。その結果について考察する。

第1表 MDD導入結果

Table 1 Result of MDD introduction

	実績	考察
生産性の向上	MDD導入部の工数は従来比78 %	・MDDキャッチアップ工数，設計変更工数などにより期待値である工数半減には届かず ・次機種開発では達成を目論む
上流工程での品質確保	システムテストまでに，全不具合の50 %以上を検出	・モデル図中心の設計により，設計レビューを効率化
不具合対策時間の短縮	従来は平均1～2件／日，MDD導入後は3～4件／日	・モデル図で不具合原因分析，修正時の影響範囲特定を行い，対策時間を短縮
手法適用規模	総規模の46 %	・今後適用範囲拡大を検討

3.1 生産性の向上

商品開発に費やした工数を分析した結果，従来開発と比較して20 %以上の工数が削減できた。

また，コード自動生成は単純なコーディング工数の圧縮だけでなく，コーディングミスの排除による後戻り工数の削減にも貢献していると考えられる。

3.2 上流工程での品質確保

開発中に検出された全不具合のうち，システムテスト開始前までに検出された不具合の割合は50 %以上と高い比率に達した。これは設計の見える化とコード生成の自動化の相乗効果によるものと考えられる。

モデル図は自然言語で書かれたドキュメントに比べて曖昧さが小さく，設計レビューの効率向上が期待できる。また，モデル自体を検証することができるため，設計工程の段階で早期に品質を作り込むことが可能である。さらに，モデル図でソフトウェア構造を見える化したことで全体を俯瞰（ふかん）して設計する視点が生まれ，結果として検討漏れによる不具合が減少したと考えられる。

また，モデル図からコードを自動生成するということにより設計内容と実装内容の一貫性が常時保持される。ゆえに，モデル図のレビューはコードレビューも兼ねることになる。

さらに，MDDでソフトウェアの振る舞いを変更するためにはモデル図の修正が必須であり，従来開発の設計工程に立ち戻ることを強制する。問題発生時は必ず設計工程に立ち戻ることで，場当たり的な対処やデグレードの頻度が低下し，設計品質の向上につながったと考えられる。

3.3 不具合対策時間の短縮

不具合1件あたりの原因分析から対策実施までの平均時間が大幅に短縮された。これもモデル図による設計の見える化の恩恵であると考えられる。

上述のとおり，モデル図とコードは常に一貫性が保持されている。モデル図とコードの対応は容易に確認でき，かつモデル図からは問題発生部分の関連処理や設計変更時の影響範囲を視覚的に特定しやすく，不具合対策時間が大幅に短縮された。

4. 今後の動向

MDD導入により，開発効率の向上，品質確保の効果が確認できた。今後も組込みソフトウェアへのMDD適用の範囲拡大を進めていく。

MDD導入時には，既存コードをモデル化する作業が必須である。また，モデル化は既存コードの外部的振る舞いを保ちつつ内部構造を改善するリファクタリングに等しい。MDD導入に際しては既存コードの設計，実装の問題点を抽出し，ソフトウェア構造や責務分担を見直した上でモデル化することが，保守性，拡張性の維持，向上への第一歩と考える。今後の組込みソフトウェア開発においては，トップダウンでソフトウェアの全体最適化設

計を行えるソフトウェア・アーキテクトの存在が重要となっていくと考える。

参考文献

1）IBM ProVISION Spring 2008 No.57 (IBM) pp.18-25 (2008).