社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2007－ＳＥ－156 (14)
2007－EMB－ 5 (14)
2007／5／29

MDDチャレンジにみる組込みソフトウェアモデル中心開発の工学と教育

鷲崎 弘宜[1] 久保秋 真[2] 小林 靖英[3]
渡辺 晴美[4] 小倉 信彦[5] 飯田 周作[6]

[1]国立情報学研究所 [2]サイバービーンズ [3]アフレル [4]東海大学 [5]武蔵工業大学 [6]専修大学

ソフトウェア工学研究会と組込みシステム研究会では，2004年より3年間にわたって，コンテスト形式で組込み開発におけるモデル駆動/中心型のソフトウェアシステム開発を競うMDDロボットチャレンジを開催してきた．本稿では，組込みシステム開発におけるモデル駆動開発や周辺のソフトウェア/システム工学技術の役割や可能性，および，それらの実践的な教育方法を，2006年を中心とした過去の開催結果より分析して報告する．また，分析結果を踏まえて今後の同種のチャレンジのあり方や組込み開発一般におけるモデル駆動開発の可能性を考察する．

Engineering and Education of Model-Driven Embedded Software Development in MDD Robot Challenge

Hironori Washizaki[1] Shin Kuboaki[2] Yasuhide Kobayashi[3]
Harumi Watanabe[4] Nobuhiko Ogura[5] Shusaku Iida[6]

[1]National Institute of Informatics [2]Cyberbeans, Inc. [3]Afrel, Inc. [4]Tokai University
[5]Musashi Institute of Technology [6]Senshu University

A robot software design contest, called the "Model-Driven Development (MDD) Robot Challenge", has been held once a year from 2004 to 2006. The challenge was intended to encourage research and education of model-driven development and related technologies in the field of embedded system development. Moreover, it is planed to hold the same challenge in the following years. In this paper, we consider the nature and education way of the model-driven development and related technologies in the field of embedded system development by analyzing the results of the challenge in 2006.

1 はじめに

情報処理学会ソフトウェア工学研究会と組込みシステム研究会では，過去3年間にわたり組込み開発におけるモデル駆動ソフトウェアシステム開発を競うMDDロボットチャレンジ（以降，チャレンジ）を開催してきた[1, 2, 3]．モデル駆動開発（Model-Driven Development: MDD）とは，属人性が排除された変換規則を用いて繰り返しモデルを変換していくことにより最終的にプログラムコードを導出する開発手法である．OMGのMDA（Model-Driven Architecture）[5]に代表されるMDDはその提案以来，Webサービスや金融システム等の主に一般的な情報システム開発について，支援ツールの拡充と事例（例えば[6, 7]）の蓄積が進められてきている．

一方，組込みシステムに対するMDDの研究・実践事例（例えば[8, 9]）も報告されつつあるが，MDA/MDDが組込み開発で果たす役割や有効性・必然性は依然として不明瞭である．そこで本稿では，チャレンジをリアルタイム組込みシステムの実践的開発機会と捉えて，同開発に特有な問題とMDDによる解決の対応，および，MDDの適切な適用にあたりエンジニアリングやマネジメントの観点から実施すべき事柄を考察する．さらにチャレンジを教育機会と捉えて，MDDや周辺技術およびチーム開発の実践的な教育方法を考察する．

2 チャレンジの概要

チャレンジにおいて参加者（チャレンジャ）は，小型飛行船を自動制御するシステムに組み込むソ

フトウェアをモデル駆動開発に従って開発する．用いるシステムの概要を図1に示す．システムは主に飛行船および基地局より構成され，飛行する飛行船を無線や超音波といった通信技術を利用して制御する．基地局PCは，各通信モジュールより得られる情報に基づき，飛行船に対して制御命令を発信する．チャレンジャは，基地局PCおよび機体搭載MPUに搭載されるプログラムを開発する．

チャレンジにおける競技は下記より構成され，各項について審査を行う．以下に示すように，チャレンジにおけるシステム構成および課せられる要求は非常に複雑なため，開発早期における諸問題の整理検討や段階的な洗練/合成を促すモデリング技術/MDD技術の適用が重要となる．

- 航法競技: 飛行船を出発地から離陸して立ち寄り点（ウェイポイント: WP）を通過しながら目的地に着陸させることの正確さを競う．概要を図2に示す．ウェイポイントは毎年のチャレンジによって異なり，例えば2006年では1つのバルーンが設定された．また，チャレンジャはスクリーンに基地局の処理状況を投影する．
- 規定競技: 垂直，水平，回転，停止など基礎的な動作について審査を行う．
- モデル審査: チャレンジャが作成して事前に提出したモデルそのものの内容を審査する．

3 組込み開発におけるモデル駆動

本節では，チャレンジにおける要求と達成手段の関係を分析することで，リアルタイム組込みシステム一般に共通する問題と，その解決に対するMDDの役割を考察する．

3.1 課題と達成手段

チャレンジにおけるソフトウェア開発は，分散・組込み・リアルタイムプログラムの実装という非常に高度な要求を満たさなければならない．さらに，運営側が用意するシステムの仕様は，チャレンジの開始時点においてセンサ類の細部に至るまでは確定していなかった．これらの要求と状況を，Chungらの非機能要求フレームワーク（Non-Functional Requirements Framework: NFR-FW）[4]の記法を用いて図3に示すように整理した．元来NFR-FWはソフトウェアプロダクトに対する非機能要求に着目したゴール指向要求分析手法・枠組みであるが，本稿では開発プロセス・手法に課せられる要求・制約と採用アプローチの関係整理に用いた．

図3において，チャレンジ達成のための開発上の課題・制約を以下にまとめる．

(a) 外部環境へのリアルタイム対応: システムを構成するセンサやモータを制御して，振る舞いを制御できない外部環境に対してリアルタイムに対応しなければならない．
この課題は，図4（文献[9], Figure 1の簡略化）に示すように一般化されたリアルタイム性を備える組込みシステムに共通する．

(b) 複雑なアプリケーション要求の満足: 複数プログラムの協調により，ウェイポイント経由で飛行船を目的地へ到着させなければならない．これは，一定規模のプログラムの必要性，および，効率のよい開発と保守のための一貫したアーキテクチャの必要性につながる．
この課題は，近年の組込みシステム開発一般に共通すると考えられる．例えば携帯電話では，ボタンセンサや通信回路等の制御の上に，「ボタンが一定時間押された場合にロックする」といった高機能アプリケーションサービスの構築が求められる．

(c) プラットフォーム仕様変化への対応: 開発中におけるプラットフォーム仕様の変化もしくは詳細決定に対応しなければならない．

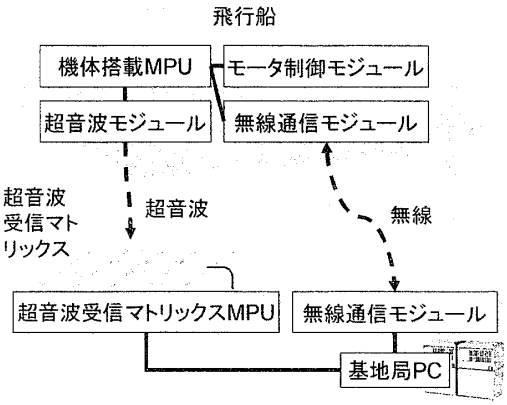

図1: チャレンジにおける共通のシステム

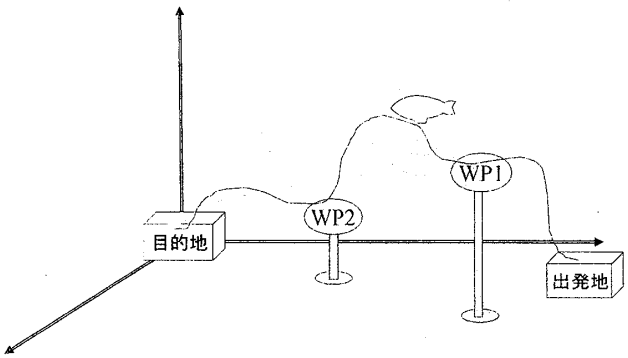

図2: チャレンジにおける共通の競技設計

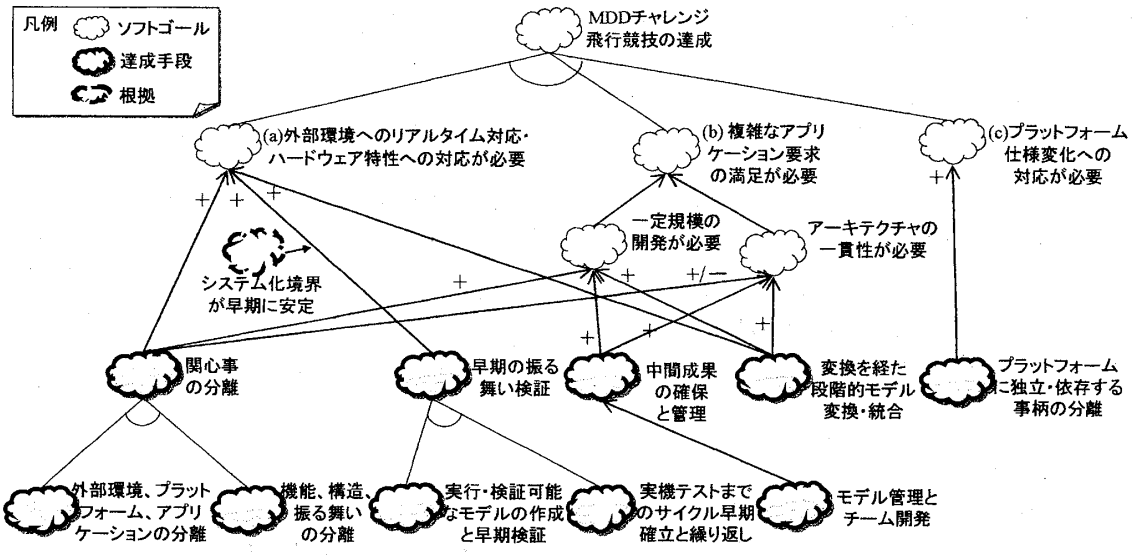

図 3: チャレンジにおける開発上の要求と達成手段の関係

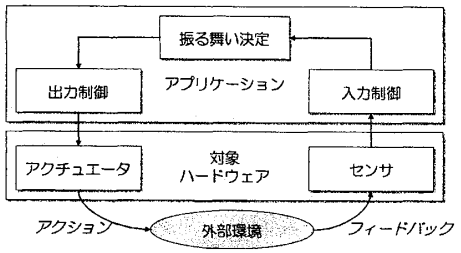

図 4: リアルタイム組込みシステムの仕組み（文献 [9], Figure 1 の簡略化）

上述の各課題・制約に対応する開発アプローチ（達成手段）を以下に挙げる．各アプローチはそれぞれ，図 3 において関連付けられた課題・制約の克服に対して正（もしくは負）の貢献をもたらす．

- 関心事の分離: プログラムを実装するうえで解決すべき種々の問題を分離してモデル化し別個に検討することで，(a)，および，(b) の一部である一定規模の効率的な開発を達成する．具体的には，外部環境・プラットフォーム・アプリケーションの分離，および，機能面，構造面，振る舞い面の分離により，複雑なアプリケーション要求とリアルタイム対応・制御要求の両方に貢献する．ただし関心事の過度の分離は，アプリケーションアーキテクチャ全体において一貫性を損なう可能性がある．

- 変換による段階的モデル変換・統合: 関心事の分離によって損なわれる可能性のある一貫性は，変換を経たモデル変換によって補うことができる．具体的には，属人性を排したモデル変換規則をメタモデル上で定義し一貫して適用することにより，同じような問題は同じように解決され，アーキテクチャの一貫性と一定レベルの品質がもたらされる．

- 中間成果物の確保と管理: 集団で意思疎通するための中間成果物を随時用意し，管理し，開発全体に対して計画的に取り組むことで (b) を達成する．具体的には，プログラムではなく図表記に基づくモデルを開発の中心におき，中間の成果物として常に管理し発展させる方法が考えられる．

- 早期の振る舞い検証: 競技会場における微風といった外部環境の振る舞いをシステムでは制御できないため，様々な状況に対応可能なように早期にシステムの振る舞いを検証し，機能性や信頼性を作りこむことで (b) を達成する．ここで，一般的なエンタープライズアプリケーション開発ではシステム化境界が開発の中盤まで不安定な傾向があるが，チャレンジのような組込みシステム開発では境界が早期に安定する．そこで，早期に実行・検証可能な外部環境・プラットフォーム・アプリケーションのモデルを作成し検証実行することが可能となる．

- プラットフォーム独立・依存の分離: プラットフォームへの独立性に応じて複数のモデルを段階的に作成することにより，同一問題領域におけるモデルの再利用性を向上させ，プラットフォーム変更時の影響をモデル間の変換作業に吸収させる．これにより (c) を達成する．

これらのアプローチ全てを包含する考え方が，MDA（Model-Driven Architecture）[5] に代表される MDD である．上述のように，組込み開発で出現

する種々の問題に対して，MDA/MDD が採用する開発アプローチは対応可能であり，さらに一部のアプローチは，特にリアルタイム性が求められる組込みシステム開発特有の問題解決に適している．

3.2 開発の流れとチャレンジ例

MDA に基づく開発の概要を図 5 に示す．MDA では，ドメインや要求を表す計算処理独立モデル（Computation Independent Model: CIM）を作成し，ツール等によって CIM を変換してプラットフォーム独立モデル（Platform Independent Model: PIM）を得た後に，ツール等によって PIM をプラットフォーム依存モデル（Platform Specific Model: PSM）に変換し，さらにその変換を経てプログラムコードを得る．用いるプラットフォームの情報を異なるモデル（プラットフォームモデル， Platform Model: PM）として記述し，PIM から PSM への変換時に用いることもある．

ただし図 5 は MDA/MDD の一般的な流れを示すものであるため，組込み開発における各モデルの詳細を検討する．組込み開発をモデル中心に進める場合に，性能実現のために物理現象や通信プロトコル規格などを連続系あるいは離散系モデルとして整理する Modeling by Rule（MbR）モデルと，アプリケーション機能実現のための構造や振る舞いを整理する Modeling by Analysis（MbA）モデルの 2 種が併用されることが知られている [10]．組込みシステムのモデル中心/駆動開発という文脈上で，それらの典型的なモデル群を整理した一例を図 6 に示す．

図 6 において，要求モデルやドメインモデルが CIM に相当する．その後は例えば，機能要求やドメインモデルに基づいて分析モデルを導出し，続いて，非機能要求・外部環境モデルおよび最下位のプラットフォームモデルに基づいて 設計モデル$_1$（アプリケーションアーキテクチャに相当）を導出し，以降は，より上位のプラットフォームモデル（ハードウェア，OS，…，フレームワーク）を順次参照して設計モデルを導出していく．それらの一連の導出手順は，設計モデル$_i$（あるいは分析モデル）を PIM，設計モデル$_{i+1}$（あるいは設計モデル$_1$）を PSM とするモデル変換とみなすことができる．

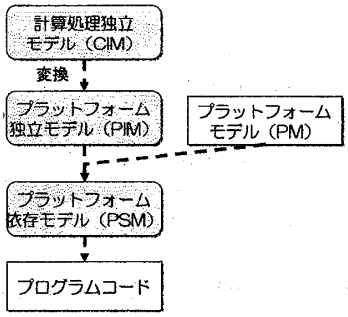

図 5: MDA における開発

例えば 2006 年のチャレンジにおいてチーム Tigermoth は，飛行競技の規約に基づいて要求を検討し，分析モデルから設計モデル・コードに至る逐次変換を実施していた．一方で同チームは並行して，座標系計算のための連続系の物理モデル上に図 7 に示すフィードバック制御方式を打ちたてて，信頼性の高いハードウェア制御のための基本的な動作を検討し，設計モデルへと反映させており，MDD 適用の好例と考えられる．

4 開発実践としてのチャレンジ

前節で述べたように，MDD の適用は飛行船制御というリアルタイム組込みシステム開発課題の達成に有効なことが期待される．しかし実際には大多数のチームについて，モデルの定性的評価結果と飛行船の競技結果が一致していなかった [3]．その一致のためには，モデリングの成熟とプロジェクト/プロセスのマネジメントの上で，モデル変換，モデル検証，および，飛行特性と対策のモデル追記といったモデル上での取り組みが不可欠と考えられる．

本節では以降において，それらの対策においてもっとも基本的なモデリング/MDD への取り組み，および，プロセス/プロジェクトマネジメントの現状と今後の期待を述べる．

4.1 モデリングへの取り組み

MDD ロボットチャレンジも 3 回を数え，チャレンジャが提出する資料にも MDD らしさがみうけられ

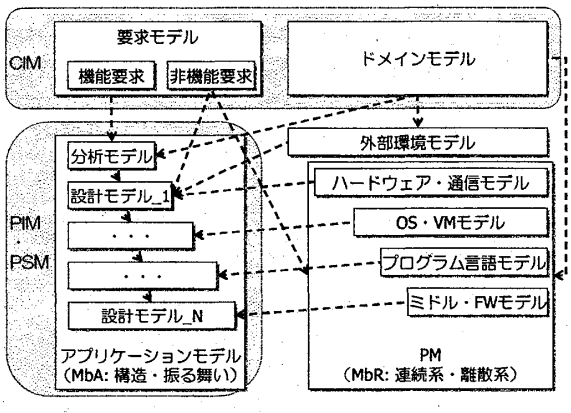

図 6: MDD に基づく組込み開発でのモデル間関係例

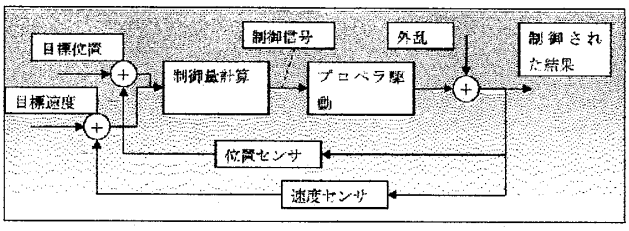

図 7: チーム Tigermoth: フィードバック制御方式

るようになってきた．本節では，チャレンジの成果を踏まえつつ，組込みシステム開発における MDD の活用可能性について述べる．

我々は，2005 年の審査員ワークショップで，2006 年のチャレンジへの期待として，a) 機器制御と競技をこなすことを分けて捉える意識を持つこと b)MDD のトライアルであるという認識の強化 c) チャレンジャに対する MDD を用いた開発への誘導を挙げた．まずこれらの点について振り返ってみよう．

まず c) の事前教育として，MDD の解説と体験チュートリアルを実施し，MDD によるチャレンジを促した．この結果，多くのチームが作成したモデルから規則に基づいてコードを得る取り組みを実施し，モデルから実装への追跡性が向上した．b) の MDD トライアルとしてみた場合に一定の効果があったといえる．また，a) として審査の際に機器制御の側面と競技をこなす側面を分離できているかを見たところ，多くのチームがこれらを分離したモデルを提出するようになってきていることがわかった．

しかしチャレンジャの MDD 実践はまだ始まったばかりである．たとえば，モデルからコードへの変換は，手で変換したところがほとんどである．これではスパイラルな開発スタイルに当てはめて，モデルを見直しては変換を繰り返すというのは困難である．また手作業の変換では，モデル上に振る舞いを記述してそれを変換するもも容易ではない．振る舞いを陽に記述したモデルの登場はこれからである．やはり，MDD のパワフルさを活用するにはツールの支援が欠かせない．今後は，より積極的な MDD の適用を促しつつ，ツール活用を前提とした教育や，容易に利用できるツール環境の整備を推進すべきではないかと考える．

われわれは，このチャレンジを通して，組込みシステム開発における MDD の役割は 2 つあるとみている．ひとつは安定したモデルやコードを効果的に再利用しながら効率よく機能開発をする役割であり，いまひとつは組込みシステムに固有の試行錯誤を効果的に支援する役割である．MDD ロボットチャレンジのワークショップで審査員の立堀氏が「試行錯誤万歳」というコメントをなさった．組込みの製品は，開発されるたびに小型化・多機能化・新しいデバイスの追加が発生するので，安定した制御の確保もまた重要な課題である．そして特性把握・性能確保のためには試作を繰り返す必要がある．氏の主張は，このような試行錯誤はあるものとして受け入れ，その代わり試行錯誤の多い状況に立った方法論や支援ツールを重視してはどうかということである．また，効率よい繰り返し開発には，支援ツールの役割は重要であると指摘している．

しかしながら，組込みシステム開発に適用する際の試行錯誤的側面に，MDD やその支援ツールが効果的かについては現段階では意見の分かれるところである．われわれは，MDD ロボットチャレンジやその他の MDD に関する教育や取り組みの中で，MDD の適用を促しつつ，MDD が組込みシステム開発につきまとう試行錯誤的側面に効果があることを確かめることで，組込みシステム開発の生産性の向上に寄与し，MDD の価値がさらに高まることを期待したいと考えている．

4.2 マネジメントへの取り組み

チャレンジャ各チームから提出されたモデル全体から，競技飛行を完成とする組込みシステム開発のスタイルを洞察している．提出モデルの内容としては，回を重ねるごとに開発モデルの網羅性が向上していることが特徴としてあげられる．これは，各チームの，チャレンジにおける必要な技術要素，問題解決領域の理解が高まってきたことであり，3 年目にしてチームごとの開発マネジメントも見えてくるようになった．つまり，実施当初は「とにかくやってみる」的な開発スタイルがありえたが，3 年目では全体網羅できてきたことで，開発の仕方，チーム内役割分担を決めた上での開発が可能となってきている．

各チームの開発プロセスはおおよそ,「ウォーターフォール的」「アジャイル的繰り返し型」「プロセスは見えない」の 3 つのパターンに分かれる．ウォーターフォール的と見られるチームは，モデルの網羅性が高いことが大きな特徴である．モデル上やシミュレータにより何らかの実行可能性の検証を行っている．その上で，チーム人数の多少に関わらず，サブシステム，ドメイン，メンバー特性といったことからチーム内役割分担開発を進めており，モデルがチーム内の情報整理共有・理解共有ツールとしても有効活用された結果であろうと見える．アジャイル的繰り返し型と見られるチームは，要素技術，サブシステムごとに実機テスト並行実施による開発スタイルとなっていることが特徴であり，テスト結果

から開発モデルを詳細化または拡大化している．

飛行競技の結果を見ると，ウォーターフォール的，アジャイル的繰り返し型とも，モデル上やシミュレーションまたは並行実機テストによる実現可能性検証に力を入れたチームが総じてよいように見え，これは組込みシステムの開発マネジメント・スタイルとして，「動かしてみてわかることが多い」ことへの対応方向性と考察される．モデルにおける実現可能検証は，このチャレンジのように実機利用制限ある中でも短期開発に有効であると思われ，組込み開発におけるMDDの取組みとして今後のチャレンジにも期待したい．

5 教育としてのチャレンジ

本稿ではここまで，リアルタイム組込みシステム開発の実践機会としてチャレンジを捉え，その性質とチャレンジの現状，今後の期待を考察してきた．一方でチャレンジは，特に学生にとってチームによるシステム開発の一通りを習得させる教育機会の側面を持つ．そこで以降において，教育機会としてのチャレンジの有効性および研究との関連を述べる．

5.1 チーム開発の教育

就職を希望する学生の多くは，就職に役立つよう即戦力が身に付く教育を望む．情報系教員の多くは，自発的に考えることができるようになって欲しい，コンピュータを理解しプログラムが組めるようになって欲しいと思っている．我々も，飛行船のように面白そうな題材ならば，興味をもって学習でき，社会人になってからも役立つ技術が身に付くのではと，MDDロボットチャレンジへ期待し，3年間取り組んできた．即戦力とは何か？という学生の率直な質問をそうした取り組みの中で考えることで，知識とそれを身につけるためのモチベーションに加えて，チームビルド，理科力を筆者は意識するようになった．以下，モチベーション，チームビルド，理科力の視点から，本チャレンジで得た知見について述べる．

(1) モチベーション

本チャレンジは，自律制御ロボット，飛行船，3次元航行制御を地上システムと協調して行う難易度の高い課題等の要素でモチベーションを与える試みである．飛行船を飛ばすということには，多くの学生が興味を持ち，研究室を希望する学生が増えた．難易度の高さの反面，最後までやり抜くことへの高いハードルとなっている．チャレンジに参加すると，夏休みもなく，アルバイトやサークル活動等，今までの学生生活がほとんどできない．飛行船を飛ばすためには，学生にとって想像以上に，学習を継続する強い心が求められる．MDDロボットチャレンジへ参加を始めた1回目は，面白い課題を与えれば，熱心に学習するようになることを期待していたが，このようなモチベーションの効果は数人の学生に限定された．むしろ，以下に記すチームビルディングや理科力の向上に対して効果があった．これらは，開始当初は想像していなかった事項である．

(2) チームビルディング

実際の開発では，部や課，プロジェクト等，チームで高い品質の製品の実現を目指す．MDDロボットチャレンジはチームでロボットを製作し，その品質の高さを競うことから，「チームで高い品質の製品を開発する意識」へと改革するのに非常に効果的である．

小学校から積み上げられてきた学習方法が「個人ができるようになること」を目指しているためか，学生達は，自分ができること，できた成果を見せることが優先される．「チームで高い品質の製品を開発する意識」が低い．従って，モデルやプログラムも，先生に診てもらい，成績を獲得することを目指した内容になってしまう．社会人になってからも役立つような即戦力へとつなげるためには，「個人⇒チーム」への意識改革が必要である．

MDDロボットチャレンジのようなコンテストでは，チームで勝つという強い心を持つこと無しには，勝つことはおろか，まともに動くシステムを作ることすら難しい．結果として，自然と「チームで高い品質の製品を開発する意識」が芽生える．チームによる議論や開発の様子を図8に示す．筆者も，本チャレンジに参加するまでは，ペアプログラミングやレビューを大学で行うことはかなり難しいと感じていた．現在では，ペアプログラミングとレビューが研究室の文化として完全に根付いた．1つの課題を2人で行わせると，片方が何もやらないという問題も起きなくなった．

(3) 理科力

ここで理科力とは，何かの問題について「仮説，

図 8: チームによる議論・開発の様子

計画，実験，考察」を繰り返し解決していく能力とする．セメスター制などにより，科目を自由に選択できることから，工学部においてもほとんど実験を経験したことのない学生が増えつつある．このような学生は，実験記録に実験時刻を記入することを知らないし，グラフや表でまとめる方法を知らない．また，授業で経験する実験では，仮説や計画は与えられているのが一般的である．考察でさえ，模範解答がある．組込み開発では，チューニングは不可欠であり，プロトタイプ開発を繰り返すことも多い．ところが，実験を繰り返し，解を発見する経験がない．従って，次の実験のための仮説を立てることも難しい．特に欠けていると感じたことは，計画を立てる力である．本チャレンジでは，考察で得られた気付きがなければ，次の実験のための仮説が立てられず進まない．このような実験の繰り返しにより解を見つける体験ができるという意義は大きい．

(4) 現在の状況

本チャレンジの経験から，大学1年生から長期的に取り組んでいかなければならないと感じた内容については，授業にもなるべく反映するようにしてきた．例えば，プログラミングの授業では，授業で毎回小テストを行い，変数の値を追跡するようにした．その授業を受けた学生達が，3〜4年生になり，成果を問われる時期になった．下記の事例により，その成果を簡単に紹介する．

2007年4月28日〜30日に，ETソフトウェアデザインロボットコンテストのための合宿を行った．このコンテストはMDDロボットチャレンジと同様に，モデリングとLEGOの走行を競う．合宿では，学生14人が3つのグループに分かれ，各々，テストの全体設計，個々のテスト項目の作成，テスト用プログラム作成・実施を行った．

設計の個々のテスト項目の内容を記すためのテスト計画書，テスト結果を記すためのテスト報告書のフォーマットを学生達で作成することができた．報告書には，LEGOの走行特有のテスト条件を記すようになっている．これらのテスト条件は，これまで学生がテストを行った際に，お互いに指摘しあってきた内容に基づいている．また，そのテスト計画書を作成した学生と別な学生がそのフォーマットに従い，プログラムを作成し，テスト報告書のフォーマットに従い結果をまとめるという作業を全員が取り組むことができた．尚，プログラミングについては，全員が基礎的な文法をすでにマスターしており，LEGO特有のモータやタイマー等の関数の使い方についても学生達で教えあうことができた．

自分達で考え，チームとして勝つために必要な計画を立て，必要な実験を遂行できたことは，継続してロボットチャレンジに取り組んで来た成果でもある．

(5) 今後の教育課題とまとめ

3年目が終了し，飛行船開発に必要な知識の与え方，すなわち，技術教育については，多少の課題はあるものの成熟してきたと考える．本チャレンジは，上記，理科力を高める題材として期待している．現在，計画を立てる力を養うために，週毎のメール数や構築したプログラムの大きさ等を計測させている．また，前節の事例で紹介したテスト計画書と報告書は，開発プロセスのドキュメント化とともに，理科力を高めるための活動の一環として行っている．テスト計画書と報告書は，大学の回路実験等をイメージさせ，これまでのレビューで問題となったことを加えることで作成した．今後，このような活動を公開していきたいと考える．

最後に，MDDロボットチャレンジに参加し，教育者として最も有益だと感じたことは，学生のいつわりのない姿を理解できたことである．本チャレンジに取り組む以前は，授業や研究室において，丁寧に個人指導を行うと，学生は理解してくれたと信じていた．この多くが自己満足であると反省する点が多く，教育者として成長できた．

5.2 教育と研究のバランス

本節では，MDDロボットチャレンジにおける教育と研究のバランスについて考察する．考察の対象として，専修大学・キャッツ株式会社・富士通デバイス株式会社・富士通の産学協同チーム (ねこねこ専FU) によって行われたエキシビション競技を取り上げる．

ねこねこ専FUは，これまで2004年度および2005年度のMDDロボットチャレンジにチャレンジャとして出場している．2006年度はエキシビション競技に参加し，コンテストのチャレンジとは離れて完全自律型の飛行船ロボットの作成を行った．2006年度のチーム構成は，学部3年次生が9人企業からの参加者が5人大学教員1人の計15人である．

(1) チャレンジ

ねこねこ専FUは，以下に挙げる点を今大会のチャレンジとして設定した．

- 学部生に与える課題は，ゴールが明確になるように設定する．
- 企業からの参加者は，既存の開発プロセスで使用するモデルとUMLをうまく融合させてMDDの適用を狙う．
- 開発の指針および方向性を失わないように，制約を出来るだけモデル化して共有する．

学部学生にとっては，MDDはおろか，ある程度の規模があるシステムを開発すること自体初めての

ことである．学生自らが考える部分と，エンジニアが持っている知識を伝達してもらう部分をうまく組み合わせながら適切なサブゴールを設定していくことが重要である．

組込ソフトウェア開発の現場では，数多くのモデルや仕様が使われているはずである．MDD を考える際に，そのことを無視しては上手くいかないのではないか．ねこねこ専FU が注目したのは，タスク図と状態遷移表である．分析モデルの段階では主にUML を使い，設計モデルの段階ではタスク図から状態遷移表を起こすという手順をとった．

組込みシステム開発は，制約によってドライブされる側面が強い．しかし，それをモデルに反映する，あるいは制約のモデルを積極的に使っていくことは，あまり進んでいない．そこで，開発を Mind Map で作成した制約モデルからスタートして，開発の過程でそれを常に参照しながら進めるという方法をとった．

(2) 何が難しいのか

MDD ロボットチャレンジにおいて教育と研究をバランスさせる難しさは以下のようにまとめられるであろう．

- MDD の有効性は，それ自体では測定できない: MDD を用いた開発が有功であることが判断できるためには，それ以外の方法との比較ができなければならない．企業で実際に開発に携わっているエンジニアは MDD 以外の方法での経験があるのでそれが可能であるが，学生にはそれができない．これが教育効果を大きく減じる結果になっている．
- 飛行船ロボットという問題自体が難しすぎる: 飛行船を上手く制御するためには，ソフトウェア開発技術以外に多くの知識・技術を必要とする．むしろソフトウェア開発技術の比重はかなり小さいと言える．そのような問題設定で MDD の有効性を考えるのは少し無理がある．
- チームや年代を超えた情報共有が難しい: 各チームには様々な問題や知見がたまっていくが，それをチーム外の人間に分かる形にまとめるのはなかなか難しい．そのため大会やその後のワークショップ等の短い時間ではなかなか情報交換が難しいと感じる．

(3) 今後の課題

MDD ロボットチャレンジが目指す教育と研究は，難しいながらも確実に成果を上げている．ここでは，議論を喚起するためにあえて今後の課題に絞って挙げた．

- 教材でない対象で教育することはなかなか難しい．MDD ロボットチャレンジの内容から教材的な内容をうまく浮かび上がらせることが重要である．
- これまで3年間実施してきて，大会全体として様々な知見が蓄積されてきている．この知見を再利用可能な形で整理していくことが必要であろう．複数年にわたって参加しているチームには内部にこのような資産があるが，大会レベルでも MDD という枠組みの中で語ることができるような再利用可能な資産がアクセス容易な状況で積み上がっていくと，研究として進展があると思われる．

6 おわりに

本稿では，主に 2006 年のチャレンジの成果を分析することで，組込みシステム開発におけるモデル駆動開発の役割と可能性，および，周辺技術を含む MDD 等の実践的な教育方法について考察した．

チャレンジは 2007 年も引き続き開催される予定であり，本稿における考察を踏まえたうえで，チャレンジを題材とした組込み MDD の追求，および，チーム開発の教育の実践が期待される．

参考文献

[1] 二上貴夫, 渡辺晴美編: MDD ロボットチャレンジ 2004 産学連携による組込みソフトウェア開発の実践, 情報処理学会, 2005.

[2] 二上貴夫 編: MDD ロボットチャレンジ 2005 産学連携によるモデルベース組込み開発の実践, 情報処理学会, 2006.

[3] 二上貴夫ほか: MDD ロボットチャレンジ 2006 開催報告, 情報処理学会研究報告, 2007-SE-156/2007-EMB-5, 2007.

[4] Lawrence Chung, et al.: Non-Functional Requirements in Software Engineering, Kluwer, 1999.

[5] Object Management Group (OMG), MDA Guide Version 1.0.1, 2003, http://www.omg.org/docs/omg/03-06-01.pdf

[6] David Frankel and John Parodi: Using MDA to Develop Web Services, IONA Technologies PLC White paper, http://www.iona.com/archwebservice/WSMDA.pdf

[7] Paloma Caceres, et al.: A MDA-Based Approach for Web Information System Development, Proc. Workshop in Software Model Engineering (WiSME) in UML'03, 2003.

[8] Jean-Louis Houberdon and Jean-Philippe Babau: MDA for embedded systems dedicated to process control, Proc. Workshop on MDA in SIVOEES in UML'03, 2003.

[9] Julien DeAntoni and Jean-Philippe Babau: A MDA-based approach for real time embedded systems simulation, Proc. 9th IEEE International Symposium on Distributed Simulation and Real-Time Applications, 2005.

[10] 渡辺政彦: 組み込みソフトウェア向け開発支援環境, 情報処理, Vol.45, No.1, 2004.