MDA による言語独立な生成系の設計と実現
～自動販売機制御ソフトウェアを例として～

2001MT073 中川 敬悟　　2002MT069 奥村 麗人　　2002MT082 田島 典子
指導教員　野呂 昌満

1　はじめに

本研究室では，組み込みソフトウェアのアスペクト指向ソフトウェアアーキテクチャスタイル (以下，E-AOSAS) を提案している．E-AOSAS では，組み込みソフトウェアアーキテクチャは並行に動作する状態遷移機械の集合と規定している．並行に動作する状態遷移機械は，並行処理，状態遷移，アプリケーションロジックのアスペクトで構成される．

一方，組み込みソフトウェアは，多様なプラットフォームで実現されている．組み込みソフトウェアが実現されるプラットフォームのひとつとして，言語があげられる．

E-AOSAS に基づく組み込みソフトウェアは，プラットフォームごとに実現方法が異なるので，手作業で実現する必要がある．プラットフォームに依存しないソフトウェアの開発を行うための手段として，MDA[5] がある．

本研究の目的は，E-AOSAS に基づく組み込みソフトウェアの開発に，MDA を用いるという開発方法の，実開発への応用可能性の考察である．本研究では，組み込みソフトウェアを MDA を用いて開発するさいのプラットフォームを，並行処理ライブラリなどの実行モデルと言語とする．実行モデルや言語によって，並行処理の実現方法は異なる．E-AOSAS に基づいて開発する組み込みソフトウェアは，並行処理を行う．MDA を用いることでプラットフォームの変更に対応できると考える．本研究では組み込みソフトウェアとして自動販売機制御ソフトウェアを例に用いる．

研究手順を以下に示す．

1. PIM を作成
2. PIM にマッピングルールを適用し PSM に変換
3. 実現
4. 実開発への応用可能性の考察

田島はアーキテクチャの構築，中川は MDA の適用，奥村は実現を担当した．

2　組み込みソフトウェアのアスペクト指向ソフトウェアアーキテクチャスタイル

E-AOSAS では，組み込みソフトウェアのアーキテクチャは並行に動作する状態遷移機械の集合と規定している．複数の状態遷移機械が互いにメッセージを送ることで協調動作をし，並行に動作する．状態遷移機械には，待機状態と活性状態があり，システム起動時には待機状態である．待機状態のときに，他の状態遷移機械からメッセージを受け取ると活性状態となる．活性状態になると，処理を開始し，処理の終了時に待機状態に戻る．並行に動作する状態遷移機械は，アスペクトとして，並行処理，状態遷移，アプリケーションロジックから構成されている．

3　MDA(Model Driven Architecture)

MDA は，PIM(Platform Independent Model) から PSM(Platform Specific Model) へ変換するアーキテクチャである．PIM はプラットフォームに独立なモデル，PSM はプラットフォームに依存したモデルである．MDA におけるプラットフォームとは，ソフトウェアの実行環境や，開発環境である．プラットフォームの例として言語，OS，ミドルウェアなどがある．PIM に，実現したいプラットフォームに対するマッピングルールを適用することで，PSM を作成できる．マッピングルールとは，PIM から PSM への変換手順を示した規則である．MDA では，PSM から対応したコードを生成することも提案されている．MDA の利点を以下に示す．

- PIM が再利用可能
  - ひとつの PIM から，プラットフォームごとの PSM に変換可能
- マッピングルールが再利用可能
  - 同じプラットフォームの PSM への変換は，同じマッピングルールを利用可能

4　MDA を用いた組み込みソフトウェアの開発

E-AOSAS に基づく組み込みソフトウェアの開発に MDA を用いる．E-AOSAS では，組み込みソフトウェアは並行に動作する状態遷移機械の集合と規定しているので，本研究では並行処理に着目する．

4.1　PIM の作成

E-AOSAS に基づく組み込みソフトウェアのアーキテクチャを構築し，構築したアーキテクチャをもとに PIM を作成する．アーキテクチャには複数の側面があり，側面ごとに着目する観点が異なる．アーキテクチャを整理するために 4+1 View Model[4] を用いる．PIM や PSM として，MDA では UML[6] の図が多く用いられているので，本研究でも UML を用いる．

E-AOSAS に基づく組み込みソフトウェアのモデルから並行処理に関するコードを自動生成するためには，モデルに並行処理に関する記述をする必要がある．並行処理に関する情報である，並行処理実体の構造や処理の流れは，言語や並行処理ライブラリによって異なる．PIM を作成するさいに，並行処理に関する記述をプラットフォームに独立にあらわさなければならない．

並行処理に関する情報はアーキテクチャにおいて Logical View と Process View で整理できる．Logical View では，組み込みソフトウェアの各システムと並行処理ライブラリとの関係を整理する．Process Views では，並行処理に関する処理の流れを整理する．

本研究では，静的構造を UML のクラス図で，処理の流れを

シーケンス図であらわす．並行処理はクラス図中のハードウェアをあらわすクラスとライブラリの関係，シーケンス図中の並行処理に関する処理の流れは，言語や並行処理ライブラリに依存する．並行処理を行うクラスやオブジェクトに <<concurrent>> をつけて，並行処理をプラットフォームに独立にあらわす．<<concurrent>> は UML Profile で定義できる．

E-AOSAS に基づく自動販売機制御ソフトウェアを例に，PIM に含まれるクラス図を**図 1** に，シーケンス図を**図 2** に示す．**図 1** は，自動販売機制御ソフトウェアの商品ボタンシステムのクラス図である．並行処理を行うクラスに <<concurrent>> をつけてあらわす．**図 2** のシーケンス図は，商品購入時の処理の流れをあらわす．並行処理を行うオブジェクトに <<concurrent>> をつけてあらわす．

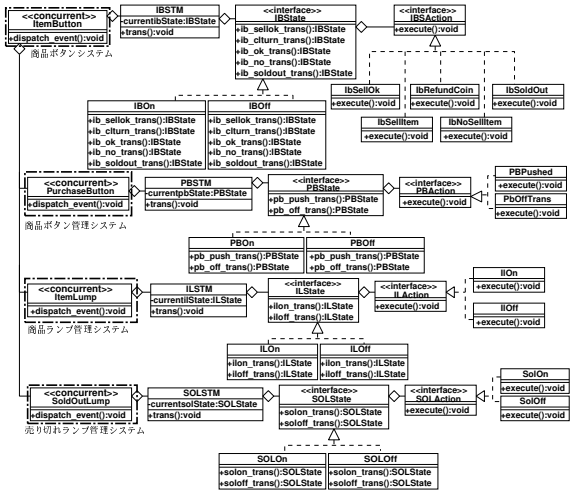

図 1　商品ボタンシステムの PIM(クラス図)

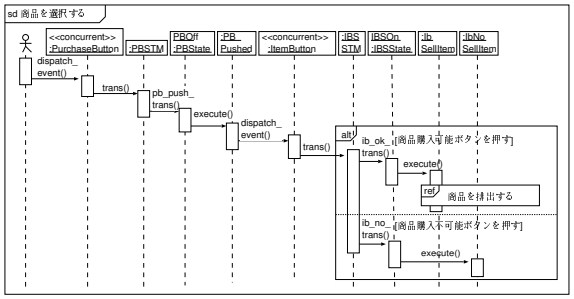

図 2　商品購入時の PIM(シーケンス図)

4.2　PIM から PSM への変換

本研究で作成した PIM から PSM へ変換するさいのマッピングルールを考える．PIM から PSM へ変換するさい，使用する言語や並行処理ライブラリごとの並行処理に関する情報を図に付加する必要がある．付加する情報を以下に示す．

- 並行処理を行うクラスと並行処理ライブラリとの関係
- 並行処理に関する処理の流れ

例として，言語に一般的なオブジェクト指向言語である Java，並行処理ライブラリに Thread を用いた場合を以下に示す．

- 並行処理を行うクラスと並行処理ライブラリとの関係
  - 並行処理を行うクラスを java.lang.Thread のサブクラスにする
- 並行処理に関する処理の流れ
  1. start() でスレッドを作成する
  2. run() 内の wait() でスレッドを待機状態にする
  3. 状態遷移機械にイベントを通知し，notify() でスレッドを活性状態にする

本研究で作成した PIM に含まれるクラス図とシーケンス図を，言語 Java および並行処理ライブラリ Thread に依存した PSM へ変換するマッピングルールを考える．クラス図の変換を**図 3** に，シーケンス図の変換を**図 4** に示す．

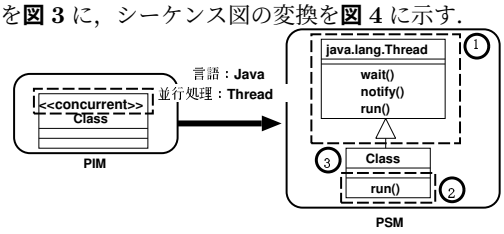

図 3　PIM から PSM への変換 (クラス図)

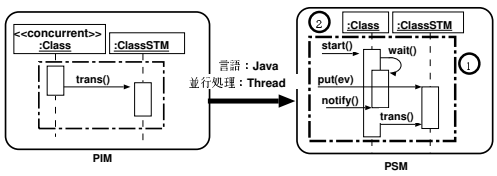

図 4　PIM から PSM への変換 (シーケンス図)

クラス図では，Java の Thread と <<concurrent>> のついたクラスとの関係を追加する．クラス図の変換手順を以下に示す．

1. <<concurrent>> のついているクラスを，java.lang.Thread のサブクラスにする
2. <<concurrent>> のついているクラスに，run() メソッドを追加する
3. <<concurrent>> を削除する

シーケンス図では，Java の Thread を用いて実現した並行処理の流れを追加する．シーケンス図の変換手順を以下に示す．

1. Java の並行処理ライブラリ Thread を用いた場合の流れに変更する
2. <<concurrent>> を削除する

自動販売機性制御ソフトウェアの PIM に，本研究で考えたマッピングルールを適用し，言語 Java および並行処理ライブラリ Thread に依存した PSM へ変換する．クラス図の変換の様子を**図 5** に，シーケンス図の変換の様子を**図 6** に示す．

図 5, **図 6** において，<<concurrent>> のついている ItemButton，PurchaseButton，ItemLump，SoldOutLump が変換される．

5　実現

PIM に含まれるクラス図から，言語 Java および並行処理ライブラリ Thread に依存した PSM への変換を実現した．実現するにあたり，Eclipse[1] を用いた．Eclipse は，オープンソースの統合ソフトウェア開発環境で，プラグインとして新しい機能を追加できる．本研究では，PIM から PSM へ

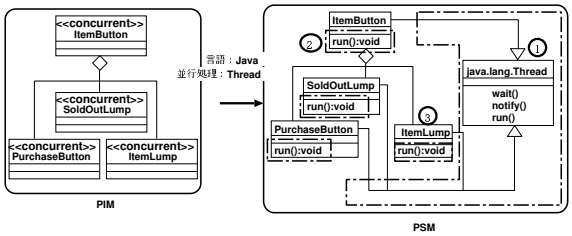

図 5　PIM から PSM への変換 (自動販売機制御ソフトウェアの商品ボタンシステムのクラス図)

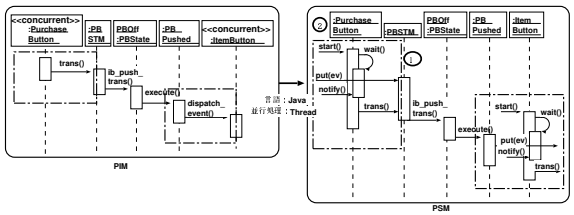

図 6　PIM から PSM への変換 (商品購入時のシーケンス図)

の変換を Eclipse のプラグインとして実現した．Eclipse を用いた PIM から PSM への変換，PSM からのコード生成の流れを**図 7** に示す．Eclipse 上で UML を図示するために，EclipseUML[2] というプラグインを用いる．EclipseUML に PIM に含まれるクラス図を入力すると XML[3] 形式で保存される．保存された XML を本研究で実現したプラグインに入力すると，入力された XML を解析し，PSM へ変換する．PSM を EclipseUML に入力するとコードが生成される．

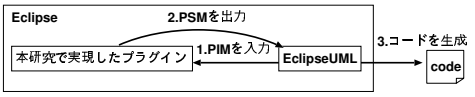

図 7　Eclipse を用いた変換，生成の流れ

Eclipse を用いた変換の例として，PIM において <<concurrent>> のついているクラスの PSM への変換，PSM からのコード生成の様子を**図 8** に示す．XML 形式の記述を変更することで，EclipseUML 上で図示したモデルを変更することができる．PSM からのコード生成は，EclipseUML を用いて行う．EclipseUML は，入力された UML の情報をもとに，Java のコードを生成する．本研究では，プラットフォームの言語を Java としているので，EclipseUML を用いてコードの生成を行う．

6　考察

E-AOSAS に基づく MDA による開発においての，実開発への応用可能性について考察する．応用可能性として，以下について考察する．

- プラットフォームの変更に対する考察
- PIM から PSM への変換手順に関する考察

6.1　プラットフォームの変更に対する考察

E-AOSAS に基づく組み込みソフトウェアの開発に，MDA を用いることで，プラットフォームの変更に対応できることを考察する．例として，プラットフォームを言語 C++ および並行処理ライブラリ pthread とした場合を考える．

プラットフォームが異なると，PIM から PSM へ変換するさ

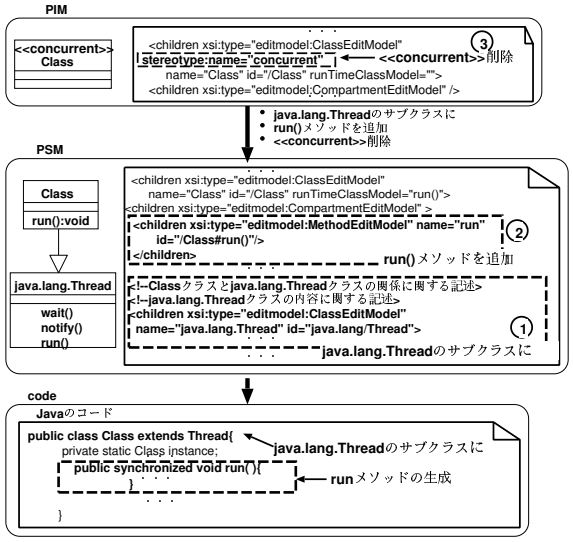

図 8　Eclipse を用いた変換例 (クラス)

いに用いるマッピングルールが異なる．プラットフォームを言語 C++ および並行処理ライブラリ pthread としたさいの，PIM から PSM への変換を**図 9**，**図 10** に示す．Java には Thread を実現するライブラリである java.lang.Thread が存在するが，C++ には存在しない．pthread を用いて Thread クラスを生成することで，言語 Java，並行処理ライブラリ Thread の場合と同様の手順で変換が行える．

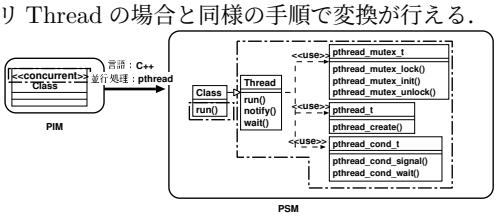

図 9　PIM から C++ および pthread に依存した PSM へ変換 (クラス図)

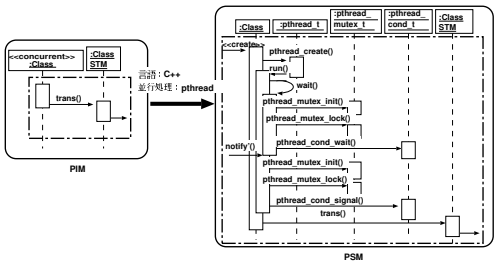

図 10　PIM から C++ および pthread に依存した PSM へ変換 (シーケンス図)

クラス図を PIM から C++ と pthread に依存した PSM へ変換するマッピングルールを以下に示す．

1. Thread クラスを生成する
2. <<concurrent>> のついているクラスを，Thread のサブクラスにする
3. <<concurrent>> のついているクラスに，run() メソッドを追加する
4. Thread クラスを pthread ライブラリと use の関係にする

5. <<concurrent>> を削除する

シーケンス図を PIM から C++ および pthread に依存した PSM へ変換するマッピングルールを以下に示す.

1. pthread を用いた場合の流れに変更する
   (a) pthread_create() でスレッドを生成する
   (b) pthread_mutex_init() で mutex を初期化する
   (c) pthread_mutex_lock() でロックする
   (d) pthread_cond_signal() でスレッドを再開する
2. <<concurrent>> を削除する

E-AOSAS に基づく自動販売機制御ソフトウェアの，プラットフォームを言語 C++ および並行処理ライブラリ pthread としたさいの PIM から PSM への変換を**図 11** に示す．**図 11** は**図 1** の PIM を PSM へ変換したものである．

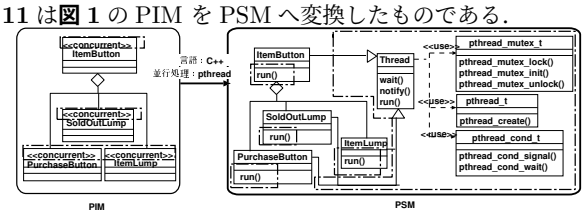

図 11　商品ボタンシステムを C++ と phread に依存した PSM へ変換

自動販売機制御ソフトウェアにおいて，クラス図，シーケンス図ともに，PIM に C++ および pthread 用のマッピングルールを適用することで PSM へ変換できることを確認した．E-AOSAS に基づく組み込みソフトウェアにおいて，プラットフォームごとのマッピングルールを作成し，適用することで，プラットフォームの変更に対応できると考える．

6.2 PIM から PSM への変換手順に関する考察

本研究では PIM から PSM への変換を一度で行った．プラットフォームを実行モデルと言語としたので，PSM への変換時にふたつのプラットフォームの情報を考慮する必要があった．複数のプラットフォームの情報を持つマッピングルールは複雑になるので，作成は容易ではない.

マッピングルールの作成を容易にする方法として，プラットフォームごとに分割して作成する方法を考える．プラットフォームごとにマッピングルールを作成し，PIM をプラットフォームごとに段階的に変換することで，PSM に変換できる．プラットフォームごとにマッピングルールを作成する場合，ひとつのマッピングルールでは，ひとつのプラットフォームに関する情報についてだけ記述する．複数のプラットフォームについてのマッピングルールを作成するより，ひとつのプラットフォームについてのマッピングルールを作成する方が，容易であると考える.

本研究においては，言語に関する変換と，実行モデルに関する変換を分割して作成できる．例として，言語 Java および並行処理ライブラリ Thread に依存するクラス図の変換を，分割したマッピングルールを以下に示す.

- Thread に関する変換
  - <<concurrent>> のついているクラスを，Thread のサブクラスにする
  - <<concurrent>> のついているクラスに，run() メソッドを追加する
  - <<concurrent>> を削除する
- Java に関する変換
  - Java に関する情報 (例：Thread を並行処理ライブラリ java.lang.Thread に変更) を追加

言語と実行モデルに関する変換を段階的に行う方が，一度の変換で行うよりマッピングルールの作成が容易になると考える．各マッピングルールはひとつのプラットフォームの情報だけを持つので，マッピングルールの作成が容易になると考える.

7 おわりに

本研究では，E-AOSAS に基づく組み込みソフトウェアの開発に，MDA を用いるという開発方法の，実開発への応用可能性を考察した．組み込みソフトウェアの開発に，MDA を用いることで，プラットフォームの変更に対応できることを確認した.

本研究では，並行処理の生成について考えた．今後の課題として，以下の処理に関する記述を生成するさいに MDA を適用することがあげられる.

- コンフィギュレーションコントロール
- 実時間処理
- 耐故障性

本研究室の他の研究において，E-AOASA では上記の処理を考慮する必要があると考えている．並行処理同様に MDA を適用することで，生成可能であると考える.

本研究では，クラス図の変換を考えたが，シーケンス図，ステートマシン図なども用い，より多くのコードを生成することも今後の課題とする.

謝辞

本研究を進めるにあたり，二年間御指導いただいた野呂昌満教授，張漢明助教授，有益なアドバイスをしていただいた大学院生の石川智子さん，石見知也さん，加藤隆広さん，小久保佳将さん，八木晴信さん，坂野将秀さん，久松康倫さん，本多克典さん，水野耕太さんに深く感謝いたします.

参考文献

[1] Eclipse Project, http://www.eclipse.org/eclipse/, Jan. 2006.
[2] Eclipse - Omondo - The Live UML Company, http://www.omondo.com/, Jan. 2006.
[3] Extensible Markup Language (XML), http://www.w3.org/XML/, Jan. 2006.
[4] P. B. Kruchten, "Architectural Blueprints - The "4+1" View Model of Software Architecture, " *Paper published in IEEE Software*, vol. 12, no. 6, pp. 42-50, 1995.
[5] MDA, http://www.omg.org/mda/, Jan. 2006.
[6] Object Management Group - UML, http://www.uml.org/, Jan. 2006.