モデル駆動型アーキテクチャを用いたアスペクト指向ソフトウェアアーキテクチャからのコード生成に関する研究
− プラットフォームを C++ 言語にして −

2004MT027 堀田 大記　　2004MT109 東海 由敬
指導教員　沢田 篤史

1 はじめに

近年，組込みシステムは，大規模化，複雑化が進んでおり，系統的な開発をおこなう必要がある．ソフトウェア開発において再利用性を考慮し，系統的な開発を支援する手法としてプロダクトラインソフトウェアエンジニアリング (以下，PLSE)[2] がある．PLSE では，一般にアーキテクチャをコア資産として再利用している．ソフトウェアアーキテクチャには規則性がある．この性質を利用してコードを自動生成することで，生産性を向上させることができる．一方，組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ので，プラットフォームに依存しないソフトウェアの開発をおこなう必要がある．プラットフォームに依存せず，モデルを中心にしてソフトウェア開発をおこなう方法としてモデル駆動型アーキテクチャ (以下，MDA) [3] がある．MDA では，プラットフォーム非依存モデル (以下，PIM) をプラットフォーム依存モデル (以下，PSM) に変換し，さらにプラットフォームコードを生成する．

本研究室では，組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++ は，統一的な記述をおこなっているので，ソフトウェア開発に E-AoSAS++ を適用することにより，再利用性の高いソフトウェアの開発が可能となる．E-AoSAS++ のための開発支援環境は，PLSE に基づき構成されている．

現在，E-AoSAS++ を適用したコードを作成するには，開発者が手作業でコードを記述しなければならない．また，プラットフォームが異なる場合は，プラットフォームに応じてコードを記述する必要がある．

本研究の目的は，E-AoSAS++ の開発環境整備のためにコードの自動生成ツールを作成することである．

我々は，E-AoSAS++ を適用したコードの自動生成ツールの作成にあたり，複数のプラットフォームに適応させるために MDA を用い，PSM を C++ として PIM との対応づけをしおこなった．そして，PIM から PSM への変換規則を考察し，自動生成ツールを作成した．なお，PIM については本研究室において，既に設計されている．

研究手順を次に示す．

- E-AoSAS++ を適用したプラットフォームコードの作成
- 自動生成部分を考察
- 開発支援ツールの作成

2 関連研究

2.1 PLSE

PLSE とは，ソフトウェア開発において再利用性を考慮し，系統的な開発を支援することで開発期間の短縮化をする手法である．PLSE には，次の 3 つの活動がある．

- ドメインエンジニアリング
  ソフトウェア群の分析をおこない，コア資産を開発する．
- アプリケーションエンジニアリング
  コア資産をもとに，ソフトウェア製品を開発する．
- 管理
  コア資産と開発プロセスの管理をおこなう．

2.2 MDA

MDA は，モデルを中心にしてソフトウェア開発をおこなう方法である．MDA では，開発の各工程におけるモデルを規定することで，各モデルの再利用性を高めている．実現したいプラットフォームに対する変換規則を PIM に適用することで，PSM を作成する．

3 E-AoSAS++

E-AoSAS++ は，並行状態遷移機械 (以下，CSTM) の集合として規定されている．CSTM の構造を図 1 に示す．E-AoSAS++ には，並行処理，状態遷移，例外処理，実時間処理，耐故障性などの関心事 (コンサーン) がある．これらのコンサーンは，グローバルコンサーンとローカルコンサーンに分類される．グローバルコンサーンは，システム全体を横断するコンサーンであり，並行処理，状態遷移に対応する．ローカルコンサーンは，システムの一部のコンポーネントを横断するコンポーネントであり，例外処理，実時間処理，耐故障性に対応する．

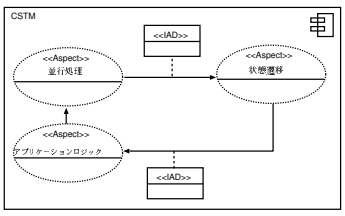

図 1　CSTM

CSTM は，次のアスペクトとアスペクト間記述によって構成される．

- 並行処理アスペクト

状態遷移機械を並行に動作させる処理
- 状態遷移アスペクト
  状態遷移機械の状態の遷移に関する処理
- アプリケーションロジックアスペクト
  状態遷移機械ごとの処理
- アスペクト間記述 (IAD)
  アスペクト間の関連を記述

また, E-AoSAS++ には, CSTM の構成を管理するコンフィギュレーションコントロール (以下，CC) が規定されている．ConfigurationCompositeCSTM は，CCPolicyCSTM と ConfigurationCSTM により構成されている．CCPolicyCSTM は各 ConfigurationCSTM の active 状態，sleep 状態を管理する．AggregationCompositeCSTM は, 複数の CSTM を協調動作させる AggregationPolicyCSTM と複数の CSTM によって構成されている．

4 MDA に基づくコード生成

4.1 E-AoSAS++ のための開発支援環境

E-AoSAS++ のための開発支援環境は，PLSE に基づき構成されている．図 2 に，PLSE のアプリケーションエンジニアリングに関する部分を示す．
状態遷移図とシーケンス図をもとにして中間表現が出力され，中間表現を Code Generator に入力することで，プログラムコードを生成する．我々は，Code Generator のうち，PIM から PSM(C++ 言語) の生成をおこなう Generator 部分を担当する．なお，PIM については本研究室において既に設計されている．

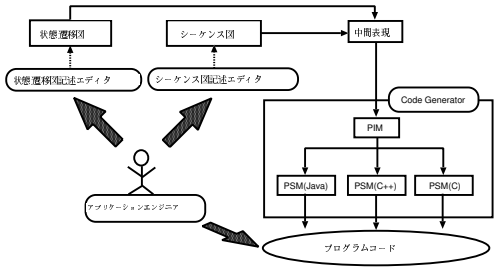

図 2　E-AoSAS++ のための開発支援環境

4.2 PIM

PIM の構成を図 3 に示す．PIM を生成するさい，状態遷移図，シーケンス図と静的構造図を入力とする．PIM は E-AoSAS++ に基づいたアーキテクチャに現れる CSTM の階層構造と各状態遷移機械のアスペクトから構成されている．ConcurrencyAspect，StateTransitionAspect，ApplicationLogicAspect ノードを構成するノード群には，PSM で必要な情報が保持されている．PSM 生成にあたり，必要な PIM の各ノードの情報を表 1 に示す．

4.3 PSM

この節では，C++ の PSM について説明する．各状態遷移機械を実現するコードにおいて，共通の処理をまとめたクラスをライブラリクラスとする．また，ライブラリクラスは，C++ の標準ライブラリと pthread ライブラリの宣言をおこなう．サブクラスは，ライブラリクラスをスーパークラスとし，スーパークラスのメソッドを呼び出すことで，C++ の標準ライブラリと pthread ライブラリを使用する．生成するクラスは，ライブラリクラスのサブクラスとすることで，自動生成箇所が削減される．

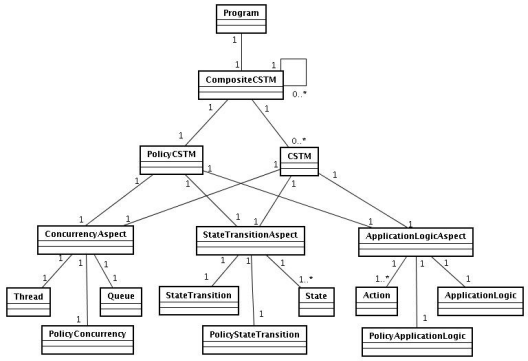

図 3　PIM

表 1　PIM の各ノード情報

	ノード名	ノードの情報
並行処理アスペクト	PolicyConcurrency	状態遷移機械名，初期状態
	Queue	状態遷移機械名
	Thread	状態遷移機械名
IAD	PolicyStateTransition	状態遷移機械名
状態遷移アスペクト	StateTransition	状態遷移機械名，状態名，イベント名，初期の状態名
	State	状態遷移機械名，自分の状態名，イベント名，他の状態名
IAD	PolicyAction	状態遷移機械名，状態名，イベント名，アクション名
アプリケーションロジックアスペクト	Action	状態遷移機械名，自身のアクション名，メソッド名，他の状態遷移機械名
	ApplicationLogic	状態遷移機械名

並行処理アスペクト

並行処理アスペクトは，Concurrency クラス，Queue クラス，Thread クラスから構成される．並行処理アスペクトを C++ で実現するさいのクラス図を図 4 に示す．Concurrency クラスは，他の CSTM からのイベント追加とスレッド管理をおこなうインタフェースである．また，flag はイベント受理，不受理の切り替えをおこなう．Queue クラスは，C++ の標準機能である vector クラスを使用する．イベントを保持するデータ構造としてキューを実現し，追加，排出をする．Thread クラスは，pthread クラスを使用することで，並行実行を実現する．

状態遷移アスペクト

状態遷移アスペクトは，状態管理をおこなう StateTransition クラス，状態を表すクラスから構成される．状態遷移アスペクトを C++ で実現するさいのクラス図を図 5 に示す．状態遷移アスペクトでは，STATE パターン [1] を使用し，状態の変更をおこなう．StateTransition クラスの trans メソッドでは，イベントごとに状態遷移機械の切替えをおこなう．状態を表すクラスの event1，event2 メソッドは，アスペクト間記述のジョインポイントであり，これらのメソッド

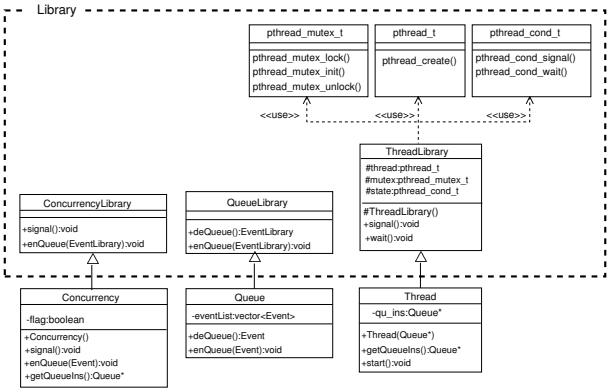

図 4　並行処理アスペクト

が呼び出されるとアプリケーションロジックアスペクトが処理される．

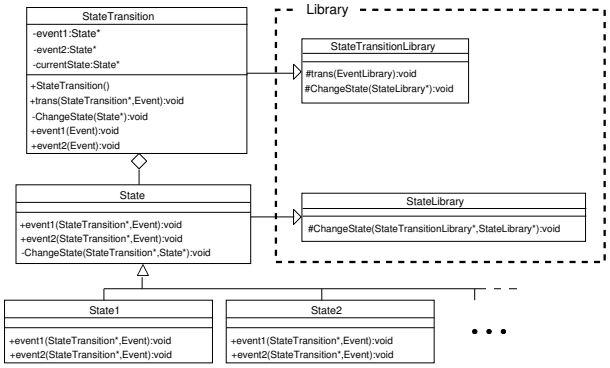

図 5　状態遷移アスペクト

アプリケーションロジックアスペクト

アプリケーションロジックアスペクトは，アクションクラス，ApplicationLogic クラスから構成される．アプリケーションロジックアスペクトを C++ で実現するさいのクラス図を図 6 に示す．アクションクラスは，コマンドパターン [1] を使用してイベントに対応した処理をおこなう．複数の異なる操作について，オブジェクトを用意し，切替えることで操作の切替えを実現する．ApplicationLogic クラスは，CSTM が保持するデータへの操作をおこなう．

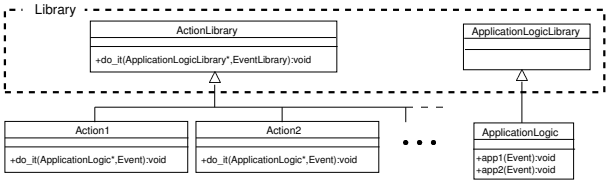

図 6　アプリケーションロジックアスペクト

イベント

EventLibrary クラスは，各 CSTM の共通のイベントである active 状態，sleep 状態を切替えるイベントを属性する．Event クラスは EventLibrary をスーパークラスとし，各 CSTM で取り扱うイベントを属性する．Event クラスを C++ で実現するさいのクラス図を図 7 に示す．

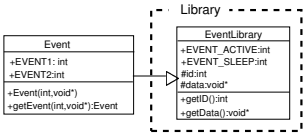

図 7　Event

アスペクト間記述

アスペクト間記述には，並行処理アスペクトから状態遷移アスペクトへのアスペクト間記述，状態遷移アスペクトからアプリケーションロジックアスペクトへのアスペクト間記述の二種類がある．並行処理アスペクトから状態遷移アスペクトへのアスペクト間記述を図 8 に示す．並行処理アスペクトから状態遷移アスペクトへのアスペクト間記述は，Queue クラスのイベントを排出するメソッドをジョインポイントとし，イベントを並行処理アスペクトから状態遷移アスペクトへ渡す．また，状態遷移アスペクトからアプリケーションロジックアスペクトへのアスペクト間記述は，状態を切替えるメソッドをジョインポイントとし，イベントに対応する処理を実行するクラスを呼び出す．

```
aspect IADofConcurrencyToState {
  advice execution( "% Queue::deQueue()" ) : after(){
    Event ev = *(tjp->result());
    Queue *q = tjp->target();

    Manager *m;
    StateTransition *next = m->getIns()->getStateTransition();
    next->trans(next,ev);
  }
};
```

図 8　並行処理アスペクトから状態遷移アスペクトへのアスペクト間記述

4.4　モデル変換論理の設計

モデル変換論理

この節では，PIM と PSM のクラスの対応を示す．コード生成ツールは，PIM の各ノードの情報を取得し，対応する PSM のクラスのコードを生成する．また，PSM の Event クラスは，StateTransition ノードにおいて各状態遷移機械が取り扱うイベントの情報を保持しているので，他のクラスと同様に Event クラスのコードを生成できる．PIM のノードと PSM のクラスの対応を図 9 に示す．

PIM とコードの対応

この節では，実際のコード生成を示す．状態遷移機械を表すコードの中で状態遷移機械ごとに異なるのは，状態遷移機械名などであり，状態遷移機械名などの情報は，対応する PIM のノードから取得する．PIM のノードの中にはリストを用いて複数の情報を保持しているものがある．複数の情報を保持している．その場合は，リストから複数個の情報を取得する．また，同じ処理をおこなう部分は定型コードとする．PIM から PSM への変換を Concurrency クラスを例として，図 10 に示す．PIM の Concurrency ノードでは，Concurrency ノードに対応して，Concurrency クラスが生成される．Concurrency クラスに対応するプログラムコードは

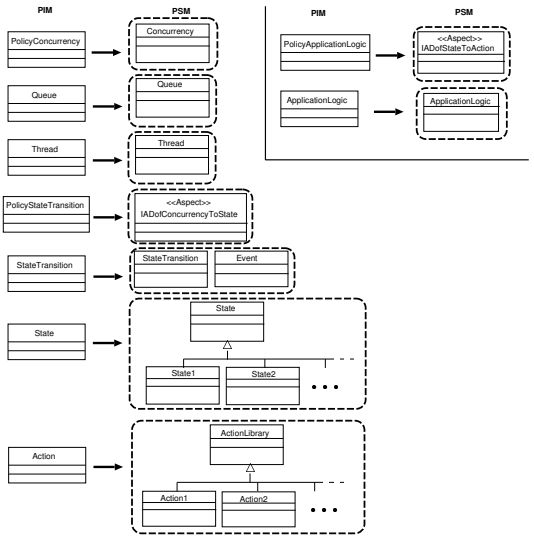

図 9　変換論理

ノードから状態遷移機械名と初期状態を取得することで自動生成可能である．
同様に，並行処理アスペクト，状態遷移アスペクト，並行処理アスペクトから状態遷移アスペクトへのアスペクト間記述と状態遷移アスペクトからアプリケーションロジックアスペクトへのアスペクト間記述に対応するプログラムコード，各ノードから情報を取得することで自動生成可能である．
アプリケーションロジックアスペクトでは，ユーザが一部プログラムコードを記述する必要がある．また，E-AoSAS++ のコードを記述すると，１つのクラスから複数のインスタンスを作成することがあり，複数のインスタンスを区別するために Manager クラスを記述する必要がある．

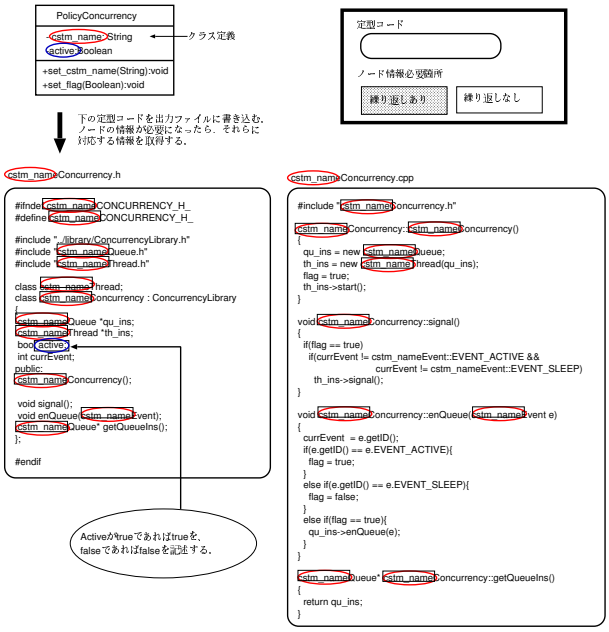

図 10　PolicyConcurrency

5　考察

5.1　プラットフォームの変更に対する考察

本研究で我々が作成したコードは，C++ の標準ライブラリや pthread ライブラリの宣言をライブラリクラスでおこなっている．他言語においても，言語ごとのライブラリクラスを作成することで，C++ と同様にコード生成可能であると考えられる．
例えば，プラットフォームを C# とした場合を考える．並行処理アスペクトの Thread クラスは言語ごとに実現方法が異なる．図 11 に，C# の Therad クラスを示す．
C# で並行処理を実現するにあたり，.NET Framework クラスライブラリ [4] の Therad クラスを使用した．C# で並行処理を実現するさい，スレッド開始は start メソッド，スレッド中断は Suspend メソッド，スレッド再開は Resume メソッドを使用する．
このように，他言語においても言語ごとのライブラリクラスを作成することで，コードの生成が可能である．

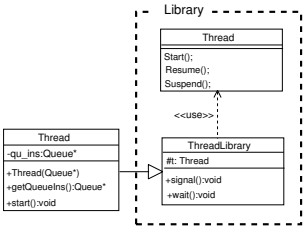

図 11　Thread クラス (C＃)

6　おわりに

本研究では、E-AoSAS++ のモデルから，PSM を C++ として PIM との対応づけをし，PIM から PSM への変換規則を考察することで，自動生成ツールを作成した．
今後の課題は，アクションクラスと Manager クラスのコードを自動生成可能にすることである．Manager クラスは，PIM の PolicyCSTM ノードと CSTM ノードに生成するインスタンス数とアクション名の情報を含ませることができた場合，自動生成可能であると考えている．また，アクションクラスについては，Manager クラスが生成され，他の状態遷移機械のどのインスタンスにイベント通知するかを把握できれば自動生成可能であると考えている．

参考文献

[1] E. Gamma, J. Vissides, R. Helm, and R. Johnson, Design Pattens Elements of Reusable Object-Oriented Software, Addison Wesley Longman, 1995.
[2] L. M. Northrop, SEI's Software Product Line Tenets, IEEE Software, Vol. 19, No. 4, pp. 32-40, 2002.
[3] Object Management Group, MDA, http://www.omg.org/mda, 2007.
[4] Microsoft, .NET Framework, http://msdn.microsoft.com/library/ja/default.asp?url=/library/ja/netstart/html/sdkstart.asp , 2007.