ソフトウェアアーキテクチャからプログラムコードへの自動変換に関する研究

M2006MM025 太田 将吾
指導教員　野呂 昌満

1　はじめに

ソフトウェアアーキテクチャに基づく開発 [4][3] では，プログラムコードの自動生成が開発の鍵である．ソフトウェアアーキテクチャに基づくソフトウェア開発では，プログラムコードに一定の規則性が存在し，定型コードを自動生成することで開発の労力を削減できる．

近年，組込みソフトウェアはハードウェアの高性能化にともない，ソフトウェアの規模の増大や構造の複雑化が進んでいる．ソフトウェアの規模の増大，構造の複雑化という問題を解決するための研究がさまざまな観点から進められている．ソフトウェアのモジュール化技法ではアスペクトクト指向 [7]，ソフトウェア開発プロセスではプロダクトラインソフトウェアエンジニアリング (Product Line Software Engineering：以後，PLSE)[6] が代表的な技法としてあげられる．

我々の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 [5](Embedded-Aspect Oriented Software Architecture Style：以後，E-AoSAS++) を提案している．E-AoSAS++ は組込みソフトウェアのアーキテクチャを，並行に動作する状態遷移機械の集合と規定する．E-AoSAS++ は組込みソフトウェアに散在する関心事をアスペクトとして適切にモジュール化することにより，再利用性，柔軟性の高い組込みソフトウェアアーキテクチャの構築を可能にである．E-AoSAS++ に基づく開発においてもプログラムコードの自動生成が生産性向上の鍵だと考える．

本研究の目的は，PLSE のアプリケーションエンジニアリングにおいて，アーキテクチャからプログラムコードへの自動生成に関する考察をおこなうことである．アーキテクチャスタイルとして E-AoSAS++ を用い，E-AoSAS++ に基づき構築した組込みソフトウェアのアーキテクチャからプログラムコードを自動生成する自動生成系を試作する．試作した自動生成系の有用性を評価することで，ソフトウェアアーキテクチャからプログラムコードへの自動変換に関する考察を行う．

試作するプログラムコード生成系は，MDA[4] の概念を用い複数のプラットフォームに対応したプログラムコードを生成する．プラットフォームを言語とし，AspectJ[2](Java)，AspectC++[1](C++,C) のコードを自動生成するプログラムコード自動生成系を試作する．

2　E-AoSAS++

E-AoSAS++ は組込みソフトウェアのアーキテクチャを構築するための，系統的な記述方法を規定している．

E-AoSAS++ では，組込みソフトウェアを並行に動作する状態遷移機械と複数の状態遷移機械を管理する PolicyCSTM の集合とする．

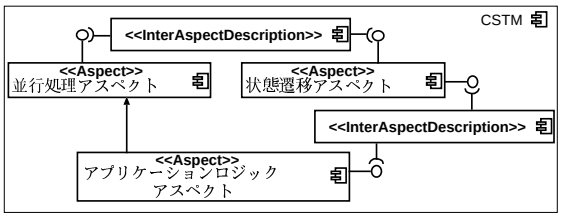

図 1　アスペクトの関連

図 1 は状態遷移機械を構成するアスペクトとアスペクト間の関連を示している．状態遷移機械は並行処理アスペクト，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で構成されている．

3　関連研究

3.1　MDA

MDA はモデルを中心としたソフトウェア開発の枠組であり，プラットフォームに依存しないソフトウェア開発を目的とする．MDA ではプラットフォームに独立モデルである PIM(Platform Independent Model) の設計を行い，プラットフォームに依存したモデルである PSM(Platform Specification Model) への変換を行うことで，複数のプラットフォームに対応したプログラムコードの自動生成が可能となる．MDA の概要を図 2 に示す．

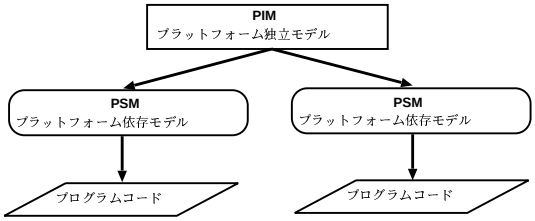

図 2　MDA の概要

4　E-AoSAS++ に基づく開発支援環境

組込みソフトウェアの開発支援環境を考えた場合, PLSE を導入することで規模の増大，構造の複雑化などの問題を解決可能であり，E-AoSAS++ においても同様である．本研究で試作するプログラムコード自動生成系は，PLSE のアプリケーションエンジニアリングのプログラ

ムコード自動生成系である．

試作するプログラムコード生成系は，静的構造図，シーケンス図，状態マシン図を入力とし，各プラットフォームに対応したプログラムコードを生成する．複数のプラットフォームに対応したプログラムコードの自動生成を考えた場合，アーキテクチャから中間形を作成し，中間形を走査することでプログラムコードを生成可能である．試作したプログラムコード生成形では，中間形として各プラットフォームにおける共通言語の抽象構文木を用いる．各プラットフォームコードを分析し，プログラムコードの構文規則を導出することで，共通言語の抽象構文木からプログラムコードの生成が可能である．本研究では，MDA における PIM を抽象構文木，PSM を各言語におけるプログラムコードと定義し，MDA の概念に基づくプログラムコード自動生成系を作成する．

5　プラットフォームコードの設計と実現

5.1　プラットフォームコードの設計

PLSE はアーキテクチャに基づくソフトウェア開発プロセスである．PLSE では，アプリケーションエンジニアリングにおいてアーキテクチャに基づきソフトウェアコンポーネントを組み合わせることにより，ソフトウェアを実現する．PLSE に基づく開発支援環境を考えた場合，コンサーンを適切にモジュール化することで再利用性，柔軟性の高いソフトウェアコンポーネントを実現する必要がある．上記の点を考慮し，本研究ではデザインパターンを用い状態遷移機械を構成する各アスペクトのプラットフォームコードの設計を行うことで，柔軟性，再利用性の高いプラットフォームコードの実現を図る．

本研究で試作するプログラムコード自動生成系は，MDA の概念に基づき，複数のプラットフォームのプログラムコードを自動生成する．複数のプラットフォームのプログラムコードの自動生成を考えた場合，一般性の高い手法を用いることでプラットフォームコードの設計を行う必要性があり，プラットフォームの設計指針とする．

5.1.1　並行処理アスペクト

並行処理アスペクトを図 3 に示すように設計した．

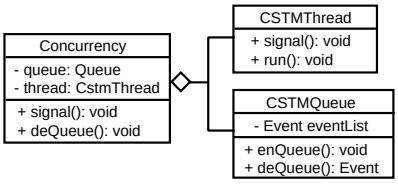

図 3　並行処理アスペクト

並行処理アスペクトは以下の 3 つのコンポーネントから構成される．

- CSTMConcurrency
- CSTMThread
- CSTMQueue

並行処理アスペクトの設計を考えた場合，状態遷移機械が受信するイベントのキューの処理，起動，停止などの同期処理をモジュール化する必要がある．キューの処理，起動，停止などの同期処理をクラスとしてモジュール化することで，各処理の実現方法の変更に対して柔軟に変更することが可能である．CSTMThread はキューの処理，CSTMThread は状態遷移機械の起動，停止などの同期処理をモジュール化したコンポーネントである．Concurrency は並行処理アスペクトのインタフェースである．状態遷移機械に通知されるイベント，及び状態遷移機械の動作を要求するメッセージは Concurrency が受信することにより状態遷移機械は動作する．

同期手法としては Signal-Wait 方式や P 操作と V 操作によるセマフォが代表的な手法であるが，Signal-Wait 方式を採用する．Signal-Wait 方式はオペレーティングシステムなどで一般的に採用される同期方法であり，複数のプラットフォームにおいて同期を実現する方法として妥当であると考える．

5.1.2　状態遷移アスペクト

状態遷移アスペクトを図 4 に示すように設計した．

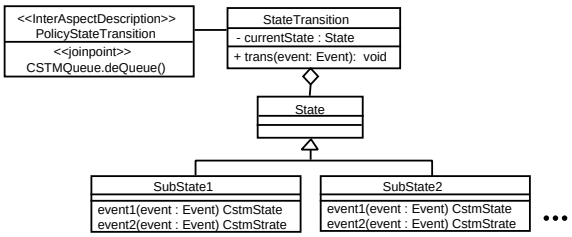

図 4　状態遷移アスペクト

状態遷移アスペクトは，以下の 3 つの構成要素から成り立つ．

- CSTMStateTransition
- CSTMState
- CSTMConcreteState

CSTMStateTransition は CSTM の状態を管理するクラスである．各並行状態遷移機械の状態を State パターンを利用し CSTMState のサブクラスとして実現し，受信したイベントに対応した状態に遷移をする．オブジェクト指向においてオブジェクトの状態を State パターンを用い表現する手法は一般的であり，状態遷移機械の状態を表現する手法として妥当であると考える．

5.1.3　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を図 5 に示すように設計を行った．

アプリケーションロジックアスペクトは，以下の 3 つの構成要素から成り立つ．

- PolicyApplicationLogic
- Action
- ApplicationLogic

アプリケーションロジックアスペクトは状態遷移機械のデータ構造，データ構造への操作，及び他の状態遷

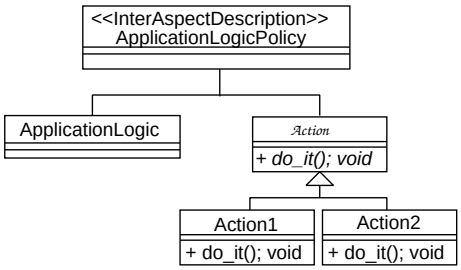

図 5　アプリケーションロジックアスペクトの構成

移機械への振舞を記述するコンポーネントの集合である．状態遷移機械の振舞いの変更を考えた場合，振舞いをモジュール化することで振舞いの変更に柔軟に対応可能であると考え，Command パターンを用いることでモジュール化した．Action コンポーネントのサブクラスとして定義するコンポーネントが，状態遷移機械の振舞をモジュール化したコンポーネントである．状態遷移アスペクトとアプリケーションロジックアスペクト間のアスペクト間記述である ApplicationLogicPolicy が Action コンポーネントにメッセージ送信することでイベントに対応した振舞いを実行する．

5.2　プラットフォームコードの実現

各プラットフォームコードは，共通コードをライブラリとして実現する．ライブラリをスーパークラスとし，サブクラスにおいて各アスペクトを構成するコンポーネントを実現ことで，ライブラリを再利用し，自動生成系はサブクラスのみを生成すればよい．

C 言語のように，オブジェクト指向言語ではない言語においてプラットフォームコードを実現する場合，各言語においてデータ抽象化を用いる．設計したプラットフォームコードの構造に基づき，データ抽象化を用いプラットフォームコードを実現することで，擬似的にオブジェクト指向言語として実現する．C 言語の場合，構造体を用いクラスを定義し，構造体の変数が関数ポインタを保持することで，擬似的にオブジェクト指向実現することが可能である．

6　プログラムコード生成系の設計と実現

MDA では，一般に UML，MOF，XMI などを用いることでモデルの記述，モデル変換を行うが，本研究では PIM に共通言語の抽象構文木を用いた手法を提案・実現する．プラットフォームコードを分析することで，モデル変換論理を定義可能であり，自動生成系の実現が容易であると考える．

プログラムコード生成系は，静的構造図，シーケンス図，状態マシン図を入力とし PIM を生成し，PIM から PSM に変換することでプログラムコードを自動生成する．PIM 生成系，および PSM へのモデル変換は Interpreter パターンを用い実現する．抽象構文木の各ノードが対応するコンポーネントのプログラムコードを生成する．図 6 は抽象構文木の木構造を示している．

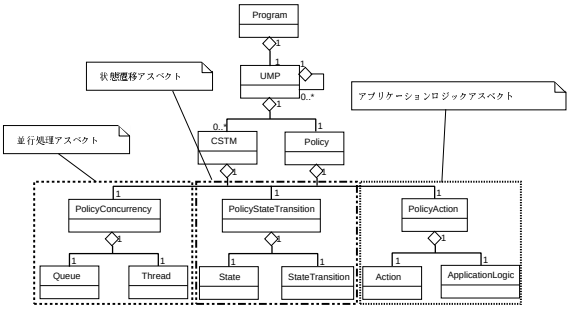

図 6　PIM

図 6 に示す木構造は，E-AoSAS++ を用い実現するソフトウェアの構成要素を示した抽象構文木である．抽象構文木の各ノードはソフトウェアを構成するコンポーネントに対応する．図式情報を取得することにより，抽象構文木を作成する．

6.1　モデル変換論理

プラットフォームコードの設計に基づき実現したプログラムコードには一定の規則性が存在する．実現したプログラムコードを分析することで PIM から PSM へのモデル変換論理を定義する．図 7 は Java プログラムの Concurrency コンポーネントを分析結果を示している．

図 7 に示すように，Concurrency コンポーネントは状

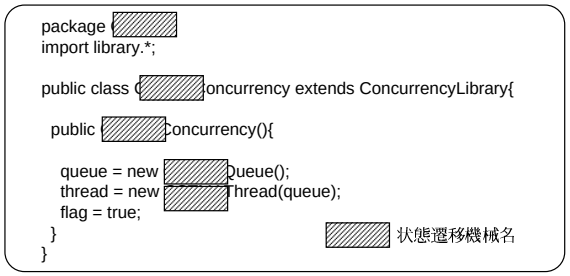

図 7　モデル変換論理

態遷移機械名と定型コードで構成されている．PIM における Concurrency コンポーネントが定型コードを示す構文規則と状態遷移機械名を用いたプログラムコードを生成することでモデル変換をおこなうことが可能である．

7　考察

本研究で試作したプログラムコード自動生成系を以下の 3 つの観点から考察することで，ソフトウェアアーキテクチャからプログラムコードの自動変換方法に関する考察をおこなう．

- プラットフォームコードの妥当性
- 自動生成の可能性
- コードカスタマイズ可能性

7.1　プラットフォームコードの妥当性

試作したプログラムコード自動生成系では，デザインパターンを用い，プラットフォームコードの設計をおこ

なった．デザインパターン使用の利点は，再利用性，柔軟性の向上である．

状態遷移機械の振舞いの変更を考えた場合，Command パターンを使用して状態遷移機械の振舞いをモジュール化した Action コンポーネントを変更することで機能の変更をおこなうことが可能である．プログラムコード記述の変更箇所はアスペクト間記述であり，コンポーネントの組み換えが容易である．状態遷移機械の状態の変更を考えた場合，状態を表現するコンポーネントに修正箇所が散在する．この問題は State パターンの問題点であるが，修正する必要があるプログラムコンポーネントを自動生成することで解決することが可能である．

PLSE のアプリケーションエンジニアリングの観点における，アーキテクチャからプログラムコードへの自動生成を考えた場合，生成したプログラムコンポーネントを再利用することで，ソフトウェアを実現すことが望まれる．本研究で設計，実現したプラットフォームコードは再利用性，柔軟性を保持したプラットフォームコードであると考えられる．

7.2 自動生成の可能性

自動生成の可能性に関する考察をおこなう．試作したプログラムコード自動生成系は図式情報を基に，状態遷移機械を構成するコンポーネントのプログラムコードを自動生成する．表 1 は図式情報から全コードを自動生成可能なプログラムコンポーネント群，及び記述を追加する必要があるコンポーネント群を示している．

構成要素	自動生成
Concurrency	○
CSTMThread	○
CSTMQueue	○
PolicyStateTransition	○
StateTransition	○
State	○
PolicyApplicationLogic	○
ApplicationLogic	△
Action	○

表 1　図からの取得情報

表 1 に示すように，ApplicationLogic 以外のコンポーネントは図式情報から全コードを自動生成が可能である．自動生成可能箇所を増大させた要因として考えられるのは，プラットフォームコードの設計における，コンサーンの適切なモジュール化とパターンの使用である．コンサーンの適切なモジュール化とパターンの使用により，プログラムコードが定型化し，プログラムコードの大部分を自動生成することができたと考えられる．

7.3 コードカスタマイズ可能性

試作したプログラム自動生成系では，図式情報から自動生成したプログラムコードの ApplicationLogic コンポーネントをカスタマイズすることによりソフトウェアを実現する．図 8 は ApplicationLogic コンポーネントの Java プログラム例を示している．

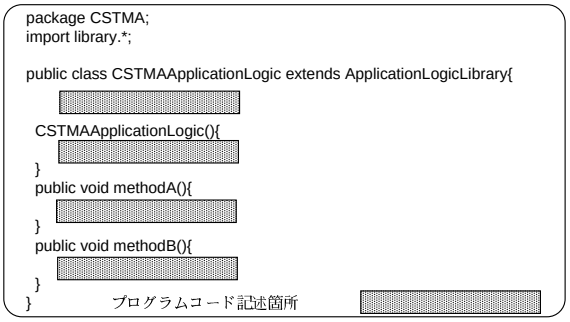

図 8　ApplicatioLogic のプログラム例

図 8 に示すように，ApplicationLogic コンポーネントにおける，変数定義，メソッドの記述を追加することでプログラムコードの実現が可能である．

以上より，自動生成したプログラムコードのユーザ記述箇所を局所化することができた．ユーザ記述箇所の局所化は，ソフトウェアに散在するコンサーンを適切にモジュールした結果であり，プラットフォームコードの妥当性を証明するものでもある．

8 おわりに

本研究では，アーキテクチャスタイルを E-AoSAS++ に特定し，PLSE のアプリケーションエンジニアリングにおけるプログラムコード自動生成系を試作した．ソフトウェアアーキテクチャからプログラムコードの自動変換を考えた場合，プラットフォームコードの設計において，コンサーンを適切にモジュール化することが重要である．

参考文献

[1] AspectC++, http://www.aspectc.org/.

[2] AspectJ, http://eclipse.org/aspectj/.

[3] J.A.D.Pace, and M.R.Campo,”ArchMatE: From Architectual Styles to Object-Oriented Models through Exploratory Tool Support,” *Proc.OOPSLA ’05*,2005.

[4] OMG,”Model Driven Architecture”, http://www.omg.org/mda/,2001.

[5] M.Noro, A.Sawada, Y.Hachisu, M.Banno “E-AoSAS++ and its Software Development Environment”, *Proceedings of the 14th Asia-Pacific Software Engineering Conference(APSEC2007)*, pp. 206-213, Dec. 2007.
Addison-Wesley,p.249-325,1995.

[6] L.M.Northrop,”SEI’s Software Product Line Tenets”,*IEEE Software,Vol.19 No.4*, pp.32-40(2002).

[7] T.Elrad,M.Aksits,G.Kiczales,K.Lieberherr,and H.Osser,”Discussing Aspects of AOP,” *OMMUNICATION of The ACM, Vol.44,No.10*,pp33-38,2001.