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 –

2004MT076 岡本 侑久　　2004MT126 山本 陽司
指導教員　野呂 昌満

1　はじめに

ソフトウェア開発において，ソフトウェアアーキテクチャからプログラムコードを自動生成し，プログラム作成を省力化することが一般的に行われている．プログラムコードを自動生成する方法の一つとして，モデル駆動型アーキテクチャ (MDA)[3] がある．MDA ではプラットフォーム非依存モデル (PIM) を作成し，PIM をプラットフォーム依存モデル (PSM) に変換する．MDA を用いることで様々なプラットフォームのプログラムコードを生成を容易に行うことができる．

われわれの研究室では組込みソフトウェアのため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++)[2] が提案されている．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 (CSTM) の集合と定め，アーキテクチャを記述する．CSTM は並行処理，状態遷移などのコンサーンをアスペクトとしてモジュール化している．E-AoSAS++ に基づいて組込みソフトウェアを設計することで，再利用性の高い組込みソフトウェアを開発することができる．

E-AoSAS++ に基づく開発でもプログラム作成を自動化するのが有効であるが，E-AoSAS++ に基づく開発を支援する環境が未整備であり，プログラム作成の省力化ができていない．

本研究の目的は，E-AoSAS++ に基づくアーキテクチャからプログラムコードへの変換方法を考察し，プログラムの自動生成ツールを実現することである．これにより，E-AoSAS++ に基づく開発を省力化する．組込みソフトウェアは実行環境によって，実現に用いるプログラミング言語が異なるので，生成ツールは MDA を用いて様々なプログラミング言語に対応させた．われわれの研究室ですでに PIM は設計されており，本研究ではプラットフォームを C 言語とした PSM と，PIM から PSM へのモデル変換論理を設計した．

以下，2 章で E-AoSAS++ の概要，3 章で生成ツールの概要，4 章で生成するコードの考察を行い，5 章でまとめをする．岡本は主に PIM から PSM への変換論理を，山本は主に PSM の設計と考察を担当した．

2　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++ では組込みソフトウェアを CSTM と，CSTM の構成を管理する CSTM(PolicyCSTM) の集合としている．CSTM は外部からイベントを受け取り，状態を遷移させながら動作し，他の CSTM へイベントを送信する．

2.1　グローバルコンサーン

E-AoSAS++ は全てのコンポーネントに横断する並行処理や状態遷移などのコンサーンをグローバルコンサーンととらえ，アスペクトとしてモジュール化する．CSTM の構成とアスペクト，IAD の関連を**図 1** に示す．

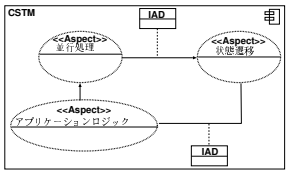

図 1　グローバルコンサーンの構成と関係

CSTM 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る．CSTM を構成する各アスペクトは，アスペクト間記述 (IAD) を介して協調動作する．

2.2　ローカルコンサーン

E-AoSAS++ では，特定のコンポーネントに横断する，例外処理，実時間処理，耐故障性などの非機能特性をローカルコンサーンとして規定する．

ローカルコンサーンを実現するうえで CSTM を管理する CSTM を PolicyCSTM とし，PolicyCSTM と管理される複数の CSTM を 1 つの CompositeCSTM とした．PolicyCSTM が管理する CSTM を active 状態 (イベントを受け付ける状態) と sleep 状態 (イベントを受け付けない状態) に切替えることで，CompositeCSTM の構成を切替える．CompositeCSTM の構成を切替えることで，ローカルコンサーンを実現する．

3　MDA を用いた自動生成ツール

3.1　MDA を用いた自動生成ツールの概要

われわれは E-AoSAS++ に基づく開発において，コード作成を省力化するために MDA を用いて自動生成ツールを設計した．自動生成ツールの概要を**図 2** に示す．

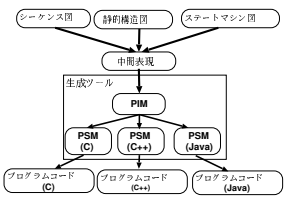

図 2　自動生成ツール

自動生成ツールは，E-AoSAS++ に基づいて CSTM と PolicyCSTM の集合として記述された静的構造図と，各 CSTM に対応するステートマシン図と，ステートマシ

ン図のアクションに対応するシーケンス図を入力とする．これらの図式は，XML による中間表現に変換されてからツールに入力される．ツールは中間表現の情報から PIM を生成し，PIM に言語ごとに定めた変換論理を適用し，選択した言語のプログラムコードである PSM に変換する．PIM は設計済みであり，本研究では C 言語で記述したコードの生成について取り扱う．

自動生成ツールは Java 言語を用いて実現する．PIM は Java のインスタンスで表現し，PSM は出力するコードとして扱う．出力された各 CSTM を構成するアプリケーションロジックアスペクトの一部の関数の中身をツールのユーザが記述することで．ソフトウェアを実現するコードが完成する．

3.2 各アスペクトの設計

われわれはコードを生成するうえで，CSTM を構成する各アスペクトを複数のコンポーネントを用いて設計した．

3.2.1 並行処理アスペクト

並行処理アスペクトは，並行処理アスペクトを管理するコンポーネントである PolicyConcurrency，CSTM が受け取るイベントのキューを表すコンポーネントである Queue，CSTM を並行に実行する機能を表すコンポーネントである Thread で構成される．

3.2.2 状態遷移アスペクト

状態遷移アスペクトはデザインパターンの中のステートパターン [1] を用いて，状態遷移アスペクトを管理するコンポーネントである PolicyStateTransition，CSTM の状態を管理するコンポーネントである StateTransition，CSTM の各状態を表すコンポーネントである複数の State で構成される．

3.2.3 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はコマンドパターン [1] を用いて，アプリケーションロジックアスペクトを管理するコンポーネントである PolicyApplicationLogic，CSTM の状態ごとに受け取ったイベントに対応する動作を表すコンポーネントである複数の Action，CSTM のアプリケーションロジックを表す ApplicationLogic で構成される．

3.3 プラットフォーム非依存モデル

われわれの研究室では E-AoSAS++ に基づくソフトウェアを 3.2 節で示したコンポーネントをノードとした木構造で表し，PIM とした．

抽象構文木の各ノードの関連を**図 3** に示す．リーフノード以外はノード独自の情報を保持しない．全てのリーフノードは自身が構成要素になる CSTM 名を持つ．PolicyConcurrency, State, PolicyApplicationLogic, Action は CSTM 名以外に**表 1** の情報を持つ．

3.4 プラットフォーム依存モデルの設計

PSM は C 言語で記述したプログラムコードで表される．C 言語でオブジェクト指向におけるクラスを記述する方法を定めた．

C 言語でクラスを記述する方法を定めることにより，同じ定義の CSTM を 1 つ 1 つ記述することなく複数生成することが可能となる．

コンポーネントの共通な処理をライブラリとしたクラスで実現し，コンポーネントはライブラリのクラスと，ライブラリのクラスをスーパクラスとしたサブクラスで表す．ライブラリのクラスを用いることでコードの自動生成を簡潔にした．

3.4.1 C 言語におけるクラス

C 言語におけるクラスは構造体のメンバに関数ポインタを保持させ，表す．記述の詳細を以下に示す．**図 4** が C 言語で記述したクラスの例である． ヘッダファイルに

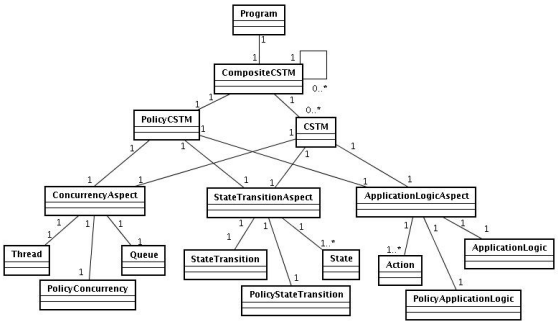

図 3　PIM

表 1　リーフノードが持つ情報

リーフ名	CSTM名以外の情報
PolicyConcurrency	CSTMのメタ状態名
State	CSTMがどのイベントで，どの名前の状態に遷移するか
PolicyApplicationLogic	CSTMがどの状態で，どのイベントを受け取ると，どのアクションをするか
Action	自身のアクション名 どの名前のCSTMに，どのイベントを送るか

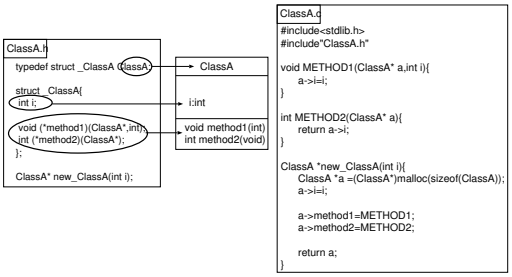

図 4　クラス (C 言語)

は構造体定義と，コンストラクタを表す関数のプロトタイプ宣言を含めた．構造体にはメンバとして関数ポインタを保持させ，クラスが持つメソッドに対応させる．その他のメンバは属性データに対応させる．ソースファイルでは，メンバ関数ポインタに対応する関数と，コンストラクタを表す関数の実装を含める．コンストラクタを表す関数で，構造体のメンバ関数ポインタに，対応する関数のアドレスの入力を行う．

メソッドを表す関数は必ず引数に自身を呼び出した構造体のアドレスをとる．構造体のアドレスからデータにアクセスする．メソッドをプロトタイプ宣言で公開するのではなく関数ポインタで構造体のメンバとして持たせる

ことで大域的なスコープからメソッドが呼び出されることを防ぐ．
継承関係はサブクラスのメンバ関数ポインタにスーパクラスのメンバ関数ポインタのアドレスを代入することで実現する．スーパクラスはソースファイルを記述しないことで，インタフェースクラスとして扱うことができる．C 言語では同じ名前のメンバを持つ構造体ポインタの型を変換することができる．サブクラスを表す構造体のメンバをスーパクラスと同じにし，サブクラスの型をスーパクラスの型に変換することで多相性を実現する．

3.4.2 PSM におけるコンポーネント

PSM において並行処理アスペクトを構成するクラスの関係を**図 5** に示す．

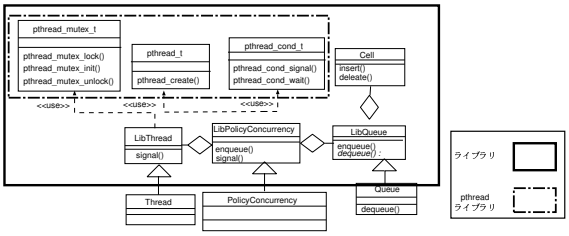

図 5　並行処理アスペクト (PSM)

並行処理アスペクトは LibPolicyConcurrency クラス，PolicyConcurrency クラス，LibQueue クラス，Queue クラス，LibThread クラス，Thread クラス，Cell クラスで構成される．
PolicyConcurrency は，LibConcurrency クラスと，LibConcurrency クラスをスーパクラスとした PolicyConcurrency クラスで表した．
LibPolicyConcurrency クラスの enqueue メソッドは，active，sleep 状態を切替え，Queue にイベントを送る．signal メソッドは，Thread に並行処理させる．
Queue は Cell クラスと，LibQueue クラスと，LibQueue クラスをスーパクラスとした Queue クラスで表した．
Cell クラスではイベントと，次セルの情報をもつことでリスト構造を実現した．LibQueue クラスの enqueue メソッドは，受け取ったイベントをキューに挿入する．Queue クラスの dequeue メソッドは，キューの先頭のイベントを返す．
Thread は LibThread クラスと，LibThread クラスをスーパクラスとした Thread クラスで表した．
オペレーティングシステムで複数のタスク間の同期をとる方法として一般的な，Signal と Wait を用いた並行処理を C 言語で実現するために，C 言語のライブラリである pthread を使用した．LibThread クラスの pthread_create メソッドは新しいスレッドを生成し，pthread_cond_wait メソッドはスレッドを停止する．signal メソッドは pthread_cond_signal メソッドを呼び出し，スレッドを再開する．pthread_mutex_lock メソッドと pthread_mutex_unlock メソッドはスレッド間の同期を取る．
状態遷移アスペクトを構成するクラスの関係を**図 6** に示す．

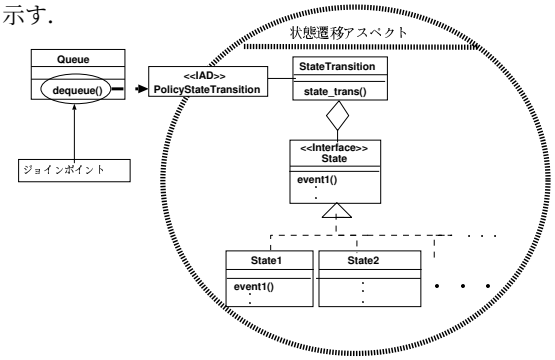

図 6　状態遷移アスペクト (PSM)

状態遷移アスペクトはステートパターンを用いて，PSM で StateTransition クラス，State インタフェースクラスと，State インタフェースクラスを実装した State クラス，PolicySteteTransition で構成される．
PolicyStateTransition は，並行処理アスペクトから状態遷移アスペクトにメッセージを送信する処理を表す IAD である．IAD は Queue クラスの dequeue メソッドをジョインポイントとして，dequeue メソッドの処理が終わった後に，StateTransision クラスにイベントを送る．
StateTransition は共通な処理がなく，ライブラリにできる部分がなかったので StateTransition クラスのみで表した．
StateTransition クラスの tarans メソッドは，受け取ったイベントを判断し，状態を遷移する．
State は State インタフェースクラスを実装した State クラスで表した．各 State クラスの受け取ったイベントに対応するメソッドは遷移先の状態を返す．
アプリケーションロジックアスペクトを構成するクラスの関係を**図 7** に示す．

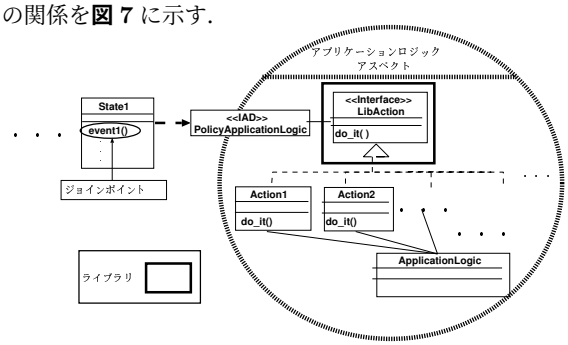

図 7　アプリケーションロジックアスペクト (PSM)

アプリケーションロジックアスペクトはコマンドパターンを用いて，PolicyApplicationLogic クラス，Action インタフェースクラスを実装した Action クラス，ApplicationLogic クラスで構成される．
PolicyApplicationLogic は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にメッセージを送信する処理を表す IAD である．IAD は各 State クラスのイベントに対応するメソッドをジョインポイントとして，イベントに対応するメソッドの処理が終わった後に，各

クラスの do_it メソッドを呼び出すことで，イベントを各 Action クラスに送る．
各 Action はライブラリの LibAction をスーパクラスとした複数の Action クラスで表した．
各 Action クラスの do_it メソッドは，各状態で受け取ったイベントに対応し，ApplicationLogic クラスのメソッドを呼び出し，他の CSTM にイベントを送信する．
ApplicationLogic は共通な処理がなく，ライブラリにできる部分がなかったので ApplicationLogic クラスのみで表した．ApplicationLogic クラスにはどのようなメソッドを持つかは記述されないので，ユーザが記述する．

3.5 モデル変換論理の設計

PIM のノードは同じコンポーネントを表す PSM のクラスに対応している．PIM は，リーフノードから PIM の情報に対応したプログラムコードである PSM に変換される．
PIM から PSM へのモデル変換は PSM のクラスごとに用意した定型コードに，対応する PIM のノード情報を書き込むことで実現する．
State ノードから State インタフェースクラスの生成を例を**図 8** に示す．

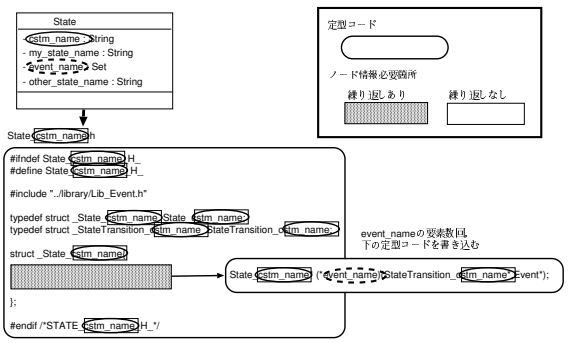

図 8　モデル変換論理 (State インタフェースクラス)

State インタフェースクラスの生成の例では定型コードの構造体の名前など，CSTM の名前が必要な箇所にノードが持つ情報を書き込む．構造体がメンバとして持つ関数ポインタは，ノードがリストとして持つイベント名の数だけイベント名の名前を使って定義する．

4 考察

ここでは，生成ツールを新たな手続き指向言語に対応させる方法と，生成ツールで出力される C 言語で記述されたプログラムコードについて考察する．

4.1 生成ツールを新たな手続き指向言語に対応させる方法

C 言語以外の手続き指向言語でもクラスを記述する方法を定義し，継承，多相性を実現できれば，C 言語と同じように生成ツールで扱うことができると考えた．
C 言語では，構造体のメンバにメソッドを表す関数ポインタを持たせることでクラスを表した．ポインタが扱えない言語の場合はわれわれが C 言語で定めた方法ではクラスを記述することができない．メソッドを表す関数の引数に構造体のアドレスをとると定めれば，プロトタイプ宣言で関数をクラスのメソッドとして公開してもインスタンスのデータを扱うことができ，ポインタが扱えない言語でもクラスとして表すことができる．
定めたクラスの記述方法でコンポーネントの共通な処理をライブラリとしたクラスで実現し，コンポーネントはライブラリのクラスとライブラリのクラスをスーパクラスとしたサブクラスで表し，定型部分を示す．
コードの定型部分と PIM の情報に基づいて生成する部分を考察し，設計したモデル変換論理を生成ツールに反映すれば，新たな手続き指向言語に対応させることができる．

4.2 他プラットフォームでの自動生成

生成ツールから出力される C 言語のコードでは Thread を実現するために pthread ライブラリを使用しているが，pthread が使えない環境で組込みソフトウェアを開発する場合も考えられる．
スレッドを扱う C 言語のライブラリの一つに MSDN ライブラリ [4] がある．MSDN ライブラリを用いた場合の Thread の実現方法について考察する．
MSDN ライブラリを用いた場合の Thread では_beginthreadex メソッドでスレッドを生成する．生成したスレッドは無限ループ内で直ちに WaitForSingleObject メソッドを呼び出して停止する．停止したスレッドは CreateEvent で生成したイベントの状態を SetEvent メソッドでシグナル状態にすることで再開される．EnterCriticalSection メソッドと LeaveCriticalSection メソッドでスレッド間の同期をとる．
PIM をライブラリに依存した PSM に変換することで，環境に適したライブラリを使ったコードが出力できる．
今後の課題としてライブラリなど，言語以外のプラットフォームに対応した生成ツールを考える必要がある．

5 おわりに

本研究では，E-AoSAS++ に基づいたアーキテクチャから C 言語のプログラムコードを生成する方法を考察した．コードを生成するために，MDA を用い，PSM のプラットフォーム言語を C 言語として，PIM から PSM への変換論理を考察し，コード生成ツールを実現した．

参考文献

[1] E. Gamma, J. Vissides, R. Helm, and R. Johnson, Design Patterns Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995.
[2] 坂野 将秀，組込みソフトウェアのためのアスペクト指向アーキテクチャスタイルの提案，南山大学大学院数理情報研究科修士論文要旨集，2006.
[3] Object Management Group，MDA，http://www.omg.org/mda，2007.
[4] Microsoft，MSDN ライブラリ，http://msdn2.microsoft.com/ja-jp/library/default.aspx，2007.