アスペクト指向に基づくソフトウェア開発の支援に関する研究
～ プラットフォームコードの設計 ～

2005MT071 森山 貴文
指導教員　蜂巣 吉成

1　はじめに

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++ 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（以下，CSTM）の集合として規定する．各 CSTM はイベントを受信することで動作し，複数の CSTM が連動することにより組込みソフトウェアの機能を実現する．既に E-AoSAS++ に基づき構築したアーキテクチャから複数の言語のプログラムコードへ自動変換をおこなうコード自動生成系 [1] が試作されている．だが，E-AoSAS++ ではインスタンスを取り扱う方法が定義されていない．よって，E-AoSAS++ に基づくコード生成系のプラットフォームコードでは，複数のインスタンスを生成し処理することができない．
本研究の目的は，E-AoSAS++ におけるインスタンスの取り扱い方法として，CSTM にインスタンス処理アスペクトを加えることを提案し，CSTM を再設計する．そして，Java によるプラットフォームコードの設計・実現をおこない，プラットフォームコードの妥当性ついて考察をおこなう．

2　E-AoSAS++

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 (以下，CSTM) の集合として規定する．複数の CSTM が互いにメッセージを送ることで協調動作し，並行に動作する．E-AoSAS++ では組込みソフトウェアに散在する横断的コンサーンを特定し，アスペクトとしてモジュール化することで，CSTM においておこなう処理を明確に分割する．CSTM は，並行処理，状態遷移，アクションの各コンサーンをアスペクトとしてモジュール化したものにより構成される．
本研究室では，E-AoSAS++ に基づくプログラムコード自動生成系によるソフトウェア開発の労力削減と開発効率の向上を目指している．コード生成系には MDA[2] を用いる．MDA はプラットフォームに依存しないモデルからプラットフォームに依存するモデルへ変換するアーキテクチャである．コード生成系では，UML 図からプログラミング言語へと変換することで，柔軟にプログラミング言語の変更に対応できる．

3　インスタンス処理問題

E-AoSAS++ に基づき作成されたソフトウェアは，CSTM を構成する各アスペクトのクラスのインスタンスを生成し実行することで動作する．しかし，従来の E-AoSAS++ はインスタンスの取り扱い方法について定義しておらず，CSTM を構成する各アスペクトのクラスのインスタンスが複数個生成された際，その複数個のインスタンスの関連を明記していないので，クラス間の関係のみによりイベントやメッセージの送信がおこなわれる．よって，意図した通りに CSTM が動作しない場合がある．例えば，E-AoSAS++ における CSTM でイベントが発生した場合，発生したイベントに応じて他の CSTM へそのイベントが送信され状態遷移を実行しなければならない．しかし，同一の CSTM のインスタンスが複数存在すると，イベント送信先が特定できず状態遷移を実行できないという問題が発生する．

4　インスタンス処理アスペクト

インスタンスの取り扱いに関する問題解決方法として，インスタンスの取り扱いを横断的コンサーンと捉え，アスペクトとしてモジュール化しインスタンス処理アスペクトとして CSTM の構成に加える．インスタンス処理アスペクトにおいて CSTM のインスタンス情報を一元管理すると，プログラムコード上へのインスタンスの情報の散在を防ぐことができる．そして，インスタンスを取り扱う記述をまとめることで，プログラムコードの生成も容易におこなえる．また，アスペクト間記述 (以下，IAD) が，保持されているインスタンスの情報を用いて各アスペクトへとイベントやメッセージを送信することができる．例えば，イベント受信側となる CSTM を発生したイベントとイベント送信側の CSTM と関連付けてインスタンス処理アスペクト内に登録・保持する．ある CSTM においてイベントが発生した場合，その情報から発生したイベントとイベント送信側の CSTM をキーにして，イベント受信側の CSTM を特定しイベントを送信する．
E-AoSAS++ でインスタンス処理アスペクトを定義することにより，E-AoSAS++ に基づき作成される組込みソフトウェアにおいて，インスタンスの取り扱い方を統一する．そして，コード生成系が生成するコードであるプラットフォームコードにおいてインスタンス処理アスペクトを実現することで，プラットフォームコードはより実用的なものとなる．

5　プラットフォームコードの設計と実現

5.1　設計

本研究では，様々なプログラムに適用できる再利用性とプログラムの変更・修正を容易におこなえる柔軟性の高

いプラットフォームコードの設計を主眼とした．
インスタンス処理アスペクトは，インスタンスの情報を InstanceaTable(インスタンス表) において一元管理し，その情報を基に各アスペクトを構成するクラスのインスタンスにイベントやメッセージを送信する．イベントやメッセージの送信には，アスペクト間記述を用いることにより，他アスペクト内にインスタンスの取り扱いに関する記述をしなくても実現可能である．
InstanceaTable に登録・保持すべき情報は，CSTM を構成するアスペクト同士の関連とイベントを通知し合う CSTM 同士の関連の 2 つである．イベント発生時の処理の流れを**図 1** に示す．InstanceTable は，CSTM を構

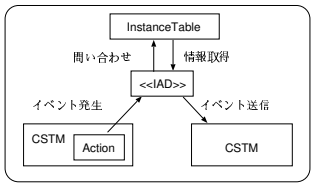

図 1　イベント発生時の処理

成するアスペクト同士の関連を表す CStmConstruction クラス，イベントを通知し合う CSTM 同士の関連を表す CStmRelation クラスをリスト構造によって登録・保持する．それら CSTM の情報を基に IAD が各アスペクトにイベントやメッセージを送信する．インスタンス処理アスペクトは InstanceTable と次の 3 つの IAD により構成される．

- ConcurrencyToState
  並行処理アスペクトから状態遷移アスペクトにイベントを送信する．
- ExecuteAction
  状態遷移に合わせて Action を実行する．
- ActionToConcurrency
  CSTM から CSTM へとイベントを送信する．

各 IAD は，実行すべきクラスのインスタンスを状態やイベントから特定し，決まったタイミングでイベントやメッセージの送信をおこなう．

5.2　実現

プラットフォームコードをアスペクト指向で実現することにより，CSTM における処理を分割してモジュール化でき，プラットフォームとなる言語の変更に対応してソフトウェア開発をおこなうことが可能である．InstanceTable は，生成したインスタンスの情報保持をおこなうコンポーネントで，記述を変更することなく再利用することができ，プログラムコードの作成における労力を削減することが可能である．**図 2** に ActionToConcurrency のプログラムコード例を示す．
ActionToConcurrency をコード生成系により生成する場合，CSTM 名を静的構造図から，アクション名を状態遷移図から取得し，定型コードと合わせることで実現可

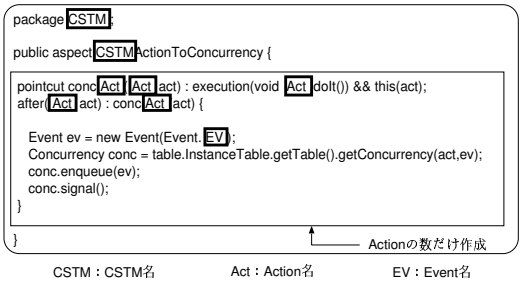

図 2　プログラムコード例 (ActionToConcurrency)

能である．イベント送信先となる CSTM のインスタンスの情報取得には，InstanceTable の利用することにより記述を変更することなく実現できる．Concurrency-ToState，ExecuteAction も同様に図式から情報を取得し，定型コードと合わせることで生成ができる．インスタンスの情報取得も同様に記述を変えることなく実現できる．また，新たに図式にオブジェクト図を用いることにより，CSTM のインスタンス生成に関する情報を取得することができる．

6　考察

InstanceTable にインスタンスに関する記述をまとめることで，コード生成系において InstanceTable の生成のみでインスタンスの処理・管理を実現することが可能であり，コード生成を容易にできる．
各アスペクトのプログラムコードにインスタンスの処理を直接記述してインスタンスの処理・管理をした場合，各アスペクトにインスタンスの情報を散在させてしまうことになり，コード生成が困難になる．本研究のインスタンス処理アスペクトを用いた場合，InstanceTable は記述を変更することなく再利用可能であり，コード生成が容易で，プラットフォームコードとして妥当であると言える．

7　おわりに

本研究では，プラットフォームとなる言語を Java(AspectJ) と特定してプラットフォームコードの設計・実現をおこなったが，MDA に基づくコード生成系においては複数のプラットフォームに対応することが求められる．よって，Java 以外の言語をプラットフォームとして同様に動作するプラットフォームコードの設計・実現をおこなう必要がある．

参考文献

[1] 太田将吾,“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,” 南山大学大学院数理情報研究科 2007 年度修士論文要旨集, Mar. 2008.
[2] Object Management Group, *Model Driven Architecture* , Jan. 2009; www.omg.org/mda/.