組込みソフトウェア開発環境に関する研究
– プラットフォームコードの再設計 –

2007MI065 伊木 友浩　　2008MI188 大西 恵太
指導教員　野呂 昌満

1　はじめに

近年，組込みシステムは複雑化し，その実現や実行環境に多様性が求められる一方，生産性の向上が課題となっている．本研究室では組込みシステム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 (以下，E-AoSAS++) を提案しており，ソフトウェア開発を支援する環境を提案している [2]．E-AoSAS++ では，組込みシステムを並行状態遷移機械 (以下，CSTM) の集合と規定している．E-AoSAS++ の開発支援環境では，並行に動作する状態遷移機械の集合を実現するために必要な実行時核を定義している．実行時核の設計では，必要十分な機能を考察した結果，並行処理，状態遷移とインスタンス処理の三つを実現する必要があることを確認している．E-AoSAS++ は，アーキテクチャに基づいた開発体系をとっており，モデル駆動型アーキテクチャ (以下，MDA) の概念に基づいたコード自動生成ツールを試作し生産性の向上を図っている．MDA は，MDA プラットフォームに独立なモデル (以下，PIM)，MDA プラットフォームに依存したモデル (以下，PSM) を定義し，プログラムコードを自動生成する技術である [1]．

PIM の実現においては，プログラミング言語を MDA プラットフォームと考えており，そのプログラミング言語のうち Java についてだけ実現されている．よって，MDA プラットフォームの変更に柔軟に対応できないという問題がある．

本研究の目的は，PIM を MDA プラットフォーム独立に再設計し，MDA プラットフォームの変更に柔軟に対応できるようにすることである．E-AoSAS++ はアスペクト指向で実現されており，アスペクトの変更が他のアスペクトに影響しない [3]．よって，MDA プラットフォームの変更が生じた際に，コンポーネントの入れ替えのみで対応可能な PIM を再設計する．
再設計の手順を以下に示す．

1. MDA プラットフォームの整理
2. 実行時核の標準化
3. アーキテクチャとコンポーネントの定義
4. アーキテクチャのホットスポットの定義

以上の手順に従い，PIM の再設計を行なった．その結果，依存箇所を個別にコンポーネント開発を行なうことが可能となった．よってコンポーネントの入れ替えのみで，MDA プラットフォームの変更に，柔軟に対応可能となった．本論文では，並行処理アスペクトについて再設計を行なった過程を示す．

2　E-AoSAS++

2.1　E-AoSAS++ の概念

E-AoSAS++ は，本研究室で提案している組込みシステム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である [2]．E-AoSAS++ は，アスペクト指向を用いて実現されており，非機能特性の選択によって，様々な実行時核を構築できることから，ポータビリティを確保している．既存の実行時核に影響を与えることなく，新たな非機能特性の追加に，柔軟に対応することが可能である．E-AoSAS++ では，並行処理，状態遷移，アクション，インスタンス処理といった関心事をアスペクトとしてモジュール化し，アスペクト間をアスペクト間通信を用いて疎結合させている．E-AoSAS++ は組込みソフトウェアを CSTM の集合と規定している．CSTM の実現モデルを図 **1** に示す．CSTM は，イベントを受け取るとイベントキューへイベントを格納し，スレッドに実行権を割り当てイベント処理を開始する．イベントキューを走査し，受理可能イベントを取り出し，イベントに応じて状態を遷移させアクションを実行し非同期に他の CSTM へイベントを送信することで協調動作する．

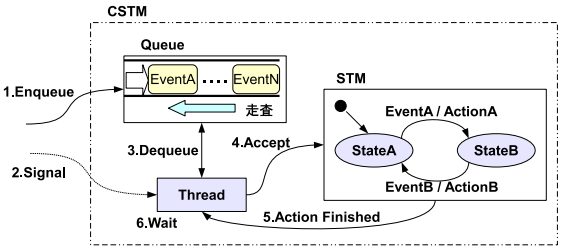

図 1　CSTM の実現モデル

2.2　コード自動生成ツール

E-AoSAS++ に基づく開発支援環境の一つに，MDA の概念を用いたコード自動生成ツールがある．コード自動生成ツールの概要を図 **2** に示す．E-AoSAS++ のアーキテクチャ記述を入力とし，PIM，PSM の二段階のモデル変換を行なうことで様々なプラットフォームのソースコードを自動生成することが可能である．E-AoSAS++ では，E-AoSAS++ の実行時核の抽象モデルを PIM と規定し，MDA プラットフォーム毎の実行時核のモデルを PSM と規定している．

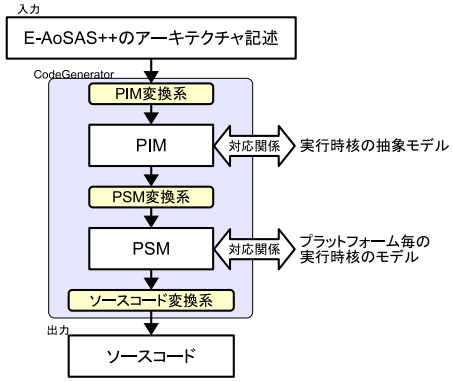

図 2　コード自動生成ツール

3　再設計した実行時核の抽象モデル

再設計前の PIM を図 **3** に示す．PIM はアスペクト指

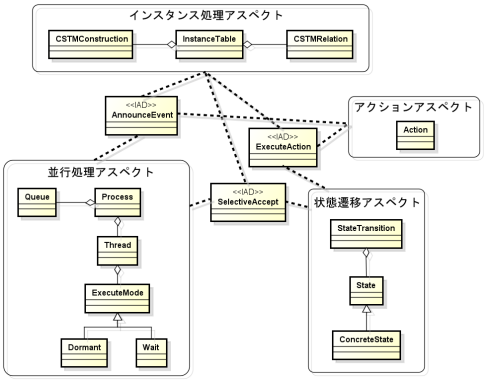

図 3　再設計前の PIM

向で実現されていることから，各アスペクトについて個別に分析，再設計ができる．分析した結果，並行処理アスペクトについて再設計の必要があり，再設計を行なった．再設計後の PIM を図 **4** に示す．並行処理アスペク

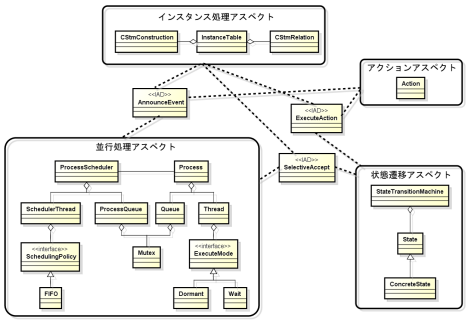

図 4　再設計後の PIM

ト以外の他のアスペクトには影響を及ぼさず再設計することが可能であり，状態遷移アスペクト，アクションアスペクト，インスタンス処理アスペクトは再利用可能である．

4　実行時核の再設計

4.1　再設計の課題

実行時核を MDA プラットフォーム独立に再設計するには，プラットフォーム毎の差異を考慮する必要がある．一般に MDA におけるプラットフォームとしてプログラミング言語，ソフトウェアプラットフォーム，ハードウェアが挙げられる．プログラミング言語が異なればシンタックスやセマンティクスが異なる．ソフトウェアプラットフォームが異なれば使用するライブラリなどが異なる．ハードウェアにおいても同様である．それぞれについて分析し考慮する必要がある．

4.2　MDA プラットフォームの整理

PIM を MDA プラットフォーム独立に再設計を行なう際に，考慮する MDA プラットフォームが整理されていないので，整理する必要がある．

考慮するプラットフォームとしてプログラミング言語，ソフトウェアプラットフォーム，ハードウェアがあるが，E-AoSAS++ はソフトウェアの開発支援を想定していることから，ハードウェアは考慮する必要がない．よって，本研究で対象とする MDA プラットフォームは，プログラミング言語とソフトウェアプラットフォームである．この二つの次元の組合せを考慮する必要がある．

4.3　対象とする実現方法

MDA プラットフォームであるプログラミング言語とソフトウェアプラットフォームは，複数の組合せがあるので，整理する必要がある．本研究では，組込みシステム開発で広く用いられているプログラミング言語，ソフトウェアプラットフォームを対象とする．よって，プログラミング言語として C 言語，C++，Java を，ソフトウェアプラットフォームとして Linux，Windows，JavaVM を対象とした．これらの組合せを整理した結果，五つの組合せが考えられた．並行処理アスペクトにおける実現方法の組合せを図 **5** に示す．

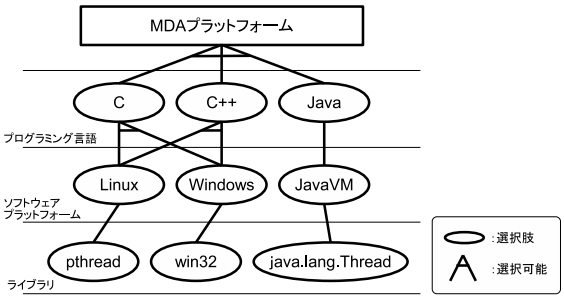

図 5　並行処理アスペクトにおける実現方法の組合せ

4.4 実行時核の並行処理アスペクトの標準化

既存の実行時核は，標準化されておらず，MDA プラットフォームに変更が生じた際に一から考える必要がある．よって，MDA プラットフォーム毎で実現する際の構造が異なる可能性が考えられ，MDA プラットフォームの変更に柔軟に対応できない．実行時核の並行処理アスペクトを標準化することで，MDA プラットフォーム間で共通の構造を定義することができ，生産性を向上させることができる．並行処理アスペクトでは，プロセス間同期処理が MDA プラットフォームに依存した機能であるので，標準化を行なう．

同期処理の原始的な実現方法として，Signal-Wait，PV 操作が挙げられる．原始的な同期処理方法を選択することで，幅広く対応できると考える．Signal-Wait はプロセスがお互いにシグナルを送信しあい，同期を取る方法である．PV 操作はプロセスが共有セマフォを持ち，このセマフォ資源に対して P 操作，V 操作を行い，同期をとる方法である．この二つの同期処理実現方法について，比較し決定する．E-AoSAS++ の並行処理アスペクトでは，並行に動作するプロセス間で同期を取る必要がある．Signal-Wait はプロセスがお互いにシグナルを送信しあい同期を取る方法でプロセスを並行オブジェクトととらえると適しており，同期処理実現方法の事実上の標準でもある．よって，Signal-Wait で標準化を行なった．

4.5 アーキテクチャとコンポーネントの定義

実行時核の並行処理アスペクトを標準化したことで，MDA プラットフォーム間で共通の構造を定義でき，アーキテクチャを定義することが可能となった．アーキテクチャとコンポーネントを定義した並行処理アスペクトを図 **6** に示す．実行時核において，再利用可能なアーキテクチャを定義し，各 MDA プラットフォームでこのアーキテクチャに従い，機能を実現することで，生産性の向上が見込める．機能の実現に必要なライブラリのインタフェースが MDA プラットフォーム毎で異なるので，統一する必要がある．よって，コンポーネントに共通なインタフェースを定義する．これにより，ライブラリのインタフェースの違いを吸収することができる．

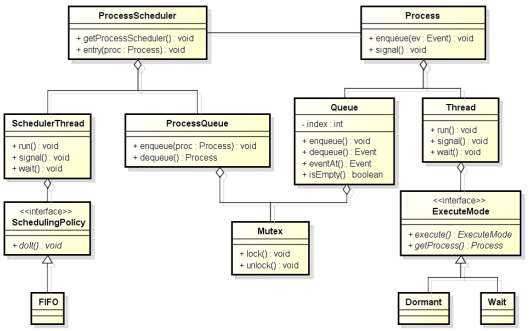

図 6　コンポーネントを定義したアーキテクチャ

さらに，プログラミング言語のセマンティクスの違いも考慮する必要がある．セマンティクスの違いを吸収ために，インタフェースを定義し分析を行なうことが有用であると考えた．

Thread オブジェクトを例に挙げる．並行処理アスペクトのプラットフォームコードを標準化したので，Thread オブジェクトには Signal-Wait で実現することから signal()，wait() のインタフェースを定義した．本研究では，定義したインタフェースに想定される仕様を与えた．signal() は，E-AoSAS++ のプラットフォームコードで待機スレッドの一つを起動させ，処理を再開させることが求められていることから，シグナルを送信する仕様とする．wait() は，処理が終わったプロセスを待機させる仕様とする．

つまり，すべての MDA プラットフォームを対象とするのではなく，仕様を定義し，仕様を満たす場合のみ再利用できるアーキテクチャを定義した．

4.6 アーキテクチャのホットスポットの定義

前節で定義したアーキテクチャにおいて，ソフトウェアプラットフォームに依存した箇所のみをコンポーネント開発し，MDA プラットフォームの変更にコンポーネントの入れ替えのみで対応可能にする．そのために，再利用可能な共通部分であるフローズンスポット，独自開発が必要な依存箇所であるホットスポットを明示的に示す必要がある．アーキテクチャにおけるフローズンスポットは，プログラミング言語のみによって異なるコンポーネントである．ホットスポットはソフトウェアプラットフォームが関係するコンポーネントである．ホットスポットを定義した並行処理アスペクトを図 **6** に示す．4.3 節で示した通り，並行処理アスペクトでは，同期処理が必要であり，ソフトウェアプラットフォームが提供するライブラリを用いて実現する．このことから同期処理が必要なコンポーネントがホットスポットとなる．よって，Thread オブジェクト，Mutex オブジェクト，SchedulerThread オブジェクトをホットスポットと定義でき，残りはフローズンスポットと定義できる．

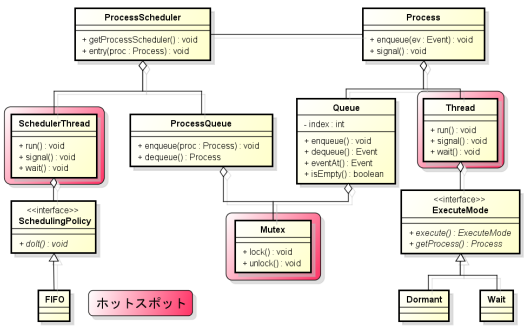

図 7　ホットスポットを定義したアーキテクチャ

5 MDA プラットフォーム独立に再設計したことに関する考察

既存の実行時核を MDA プラットフォーム独立に再設計したことに関して考察する．

並行処理アスペクトのプラットフォームコードの標準化に関して考察する．同期処理の実現方法を Signal-Wait と定義したことで，再設計前の並行処理アスペクトの多くの部分を再利用することができた．加えて，Signal-Wait は事実上の標準であることから幅広い MDA プラットフォームに対応可能なことが考えられる．本研究で対象とした MDA プラットフォームは，並行性をサポートしており Signal-Wait の実装が可能であることから妥当であったと考える．

コンポーネントの定義に関して考察する．コンポーネントを定義することで，すべての条件で成り立つのではなく条件を満たす場合のみと限定した．Thread オブジェクトを例に考察する．Thread オブジェクトでは，Signal-Wait での動作を想定し，インタフェースを定めた．signal() ではシグナルを送信する仕様と定めた．本研究で対象としたソフトウェアプラットフォームの Linux，Windows，JavaVM で適用可能か考察する．Linux では，並行性の実現に POSIX Thread ライブラリを用いた．POSIX Thread ライブラリでは，pthread_cond_signal を用いることで定義した仕様を満たすことが可能である．Windows では，Win32 ライブラリを用いた．Win32 ライブラリでは，SetEvent を用いることで仕様を満たすことができた．最後に JavaVM では，notify を用いることで仕様を満たすことができる．以上から対象すべてで仕様を満たすことができたことから，これらの MDA プラットフォームに対してライブラリのインタフェースの違いを吸収することが可能と考える．しかし，MDA プラットフォーム独立にするには，プログラミング言語毎で異なるセマンティクスの違いを吸収する必要がある．実現するために，本研究ではインタフェースを定義のみ行なった．セマンティクスの違いを吸収可能な設計とするために考察する必要があり，今後の課題である．

コンポーネントの入れ替えのみで MDA プラットフォームの変更に柔軟に対応可能かに関して考察する．各 MDA プラットフォーム用のコンポーネントを設計するには，一から実現方法などを考慮する必要がある．あらかじめインタフェースの仕様を定義したことで，仕様に従うことで容易にコンポーネントを開発することが可能であった．Thread オブジェクトを例に挙げる．各 MDA プラットフォーム用の Thread コンポーネントを開発する際，最低限の部分のみ開発するだけでよく，容易にコンポーネントを開発することができた．Thread オブジェクトの変換の例を図 **8** に示す．各 MDA プラットフォーム用のコンポーネントを開発する際，再設計を行なったことで半自動的に動作に必要な MDA プラットフォーム依存の変数等を定義することができた．また，

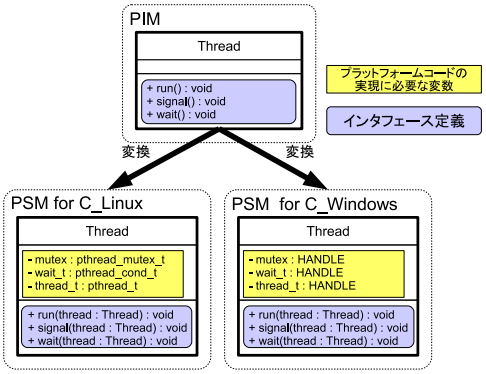

図 8　Thread オブジェクトの変換イメージ

プログラミング言語，ライブラリと二段階で考察することで，ライブラリ依存部分をコンポーネントとして開発することができ，MDA プラットフォームの変更に柔軟に対応可能であった．さらに，定義したインタフェースは，各 PSM 間でトレーサビリティを確保することができ，保守性の向上も見込めた．本研究で対象としたパターンにおいても同様の考察が行なえた．

本研究では，E-AoSAS++ に基づき再設計を行なったので，再設計前と再設計後で，振舞いは同一である．そして，簡単な例を用いてコンポーネントの入れ替えのみで既存の実行時核へ影響を与えずに MDA プラットフォームの変更に対応可能であった．よって，MDA プラットフォームの変更の際に柔軟に対応可能なことから，再設計は有用であったと考える．

6 おわりに

本研究では，E-AoSAS++ の実行時核を再設計し，MDA プラットフォームの変更に柔軟に対応可能な設計とした．簡単な例を用いて，対象とした五つのパターンについて，コンポーネントの入れ替えのみで MDA プラットフォームの変更に対応できることを確認した．今後の課題として，引き続き MDA プラットフォーム独立の考察，本研究で対象としなかった MDA プラットフォームについての考察，事例を用いて再設計した実行時核が有用か確認することが挙げられる．

参考文献

[1] Object Management Group, “ *Model Driven Architecture*, ” http://www.omg.org/mda, 2007.

[2] 長大介，加藤大地，蜂巣吉成，沢田篤史，野呂昌満，“ E-AoSAS++ に基づく開発支援環境 コード生成ツールの提案，” 情報処理学会報告，vol.2009, no.13，pp.121-128，2009.

[3] 野呂昌満，“ アスペクト指向プログラミング概観，” Seamail，vol.13，no.11.