# 組込みソフトウェア開発におけるプラットフォームコードのマルチプラットフォーム化に関する研究

M2010MM036 佐藤俊成
指導教員：野呂昌満

## 1 はじめに

組込みソフトウェアは，プロダクトラインを構成する典型的な応用領域のである [2]．リアクティブシステムである組込みソフトウェアは，一般に並行に動作する状態遷移機械の集合としてモデル化されている [5]．我々は，開発経験から耐故障性や実時間性などの非機能特性を実現する必要があることを確認している．我々の研究室では，組込みシステム開発支援のために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，これに基づく開発支援環境を考察してきた [3]．E-AoSAS++では，組込みシステムを並行に動作する状態遷移機械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を実現する必要があることを確認している．それ以外の非機能特性の実現については，実行時核上で定義される状態遷移機械の組合せによって実現できることを確認している．我々は，これまでに Java 実行環境を始めとする開発支援環境を実現してきた [6]．これらの実現において，実行時核の移植性を考慮したにも関わらず考察が不十分であり，マルチプラットフォーム化への対応を実現するには至らなかった．

本研究の目的は，実行時核と耐故障処理と実時間処理を実現するコンポーネント群を標準化し，マルチプラットフォーム化への対応を実現することである．このために，実行時核と耐故障処理と実時間処理を実現するコンポーネント群のプラットフォームを MDA[4] における変動要因として扱い整理する．それに基づいて，横断的関心事を識別し，アスペクトとして定義するアスペクト指向アーキテクチャを再定義する．耐故障処理と実時間処理を実現するコンポーネント群も同様に整理を行なう．

実行時核のアーキテクチャと耐故障処理と実時間処理を実現するコンポーネント群をプラットフォームの選択に独立であるように定義することで，マルチプラットフォーム化への対応が実現可能であることを確認した．

## 2 E-AoSAS++と PLSE に基づく開発支援環境

E-AoSAS++は，組込みシステムを対象とした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．E-AoSAS++は，並行に動作する状態遷移機械を構成要素とし，システムを並行状態遷移機械の集合として規定する．並行状態遷移機械間は，キューを介した非同期のイベント通信によって，協調動作を行なう．

### 2.1 PLSE に基づく開発支援環境

E-AoSAS++では，PLSE に基づく開発支援環境の一つとして自動生成ツールを提案している．自動生成するための技術として MDA を自動生成ツールに適用し，プログラム作成の労力を削減している．自動生成ツールは，MDA の概念に基づき入力から出力までをモデル変換によって行なう．自動生成ツールへの入力はアーキテクチャ記述であり，出力は対象とするプラットフォームのソースコードである．図 1 に提案する自動生成ツールの概要を示す．

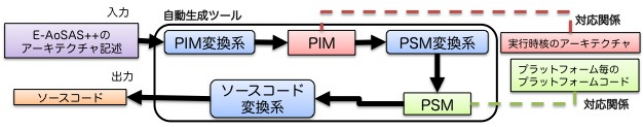


図 1 自動生成ツールの概要

### 2.2 並行状態遷移機械の実現モデル

E-AoSAS++の実行単位はコンポーネントであり，並行状態遷移機械として振舞う．全ての並行状態遷移機械は，並行処理モニタ（スケジューラ）によって，スケジューリングされ同時に並行動作する．並行状態遷移機械にはそれぞれ一つのスレッドを割り当て，汎用性が高く原始的である Signal / Wait 方式によって同期処理を行なう．並行状態遷移機械はイベントを受理しアクションの実行が終了するまで，次のイベントの受理を行わない．図 2 に Signal / Wait 方式による並行状態遷移機械の実現モデルを示す．

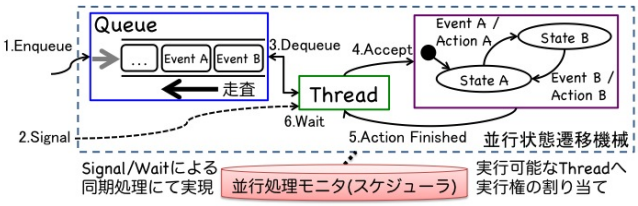


図 2 Signal / Wait による並行状態遷移機械の実現モデル

### 2.3 実行時核のアーキテクチャ

実行時核では，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機能である並行処理，状態遷移，インスタンス処理を定義している．並行処理の実現は，一般的にプラットフォーム毎に異なる．実行時核のアーキテクチャ設計では，プラットフォーム独立の観点から並行処理と状態遷移機械を分離し，移植性を実現している．図 3 に実行時核のアーキテクチャを示す．

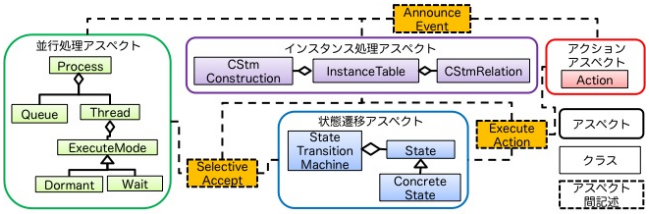


図 3 実行時核のアーキテクチャ

### 2.4 状態遷移機械の組合せによる非機能特性の実現

E-AoSAS++では，耐故障処理や実時間処理などの非機能特性を実行時核上で定義する状態遷移機械の組合せによって実現する．基底コンポーネントと基底コンポーネントの集合を管理するメタコンポーネントによって，非機能特性を実現する．図 4 に実時間処理の実現例を示す．RealTimePolicy が TimerTrigger と Timer からのイベントを基に構成を切替えることで，実時間性を実現している．Timer では，プラットフォームに依存するタイマの API を利用したコンポーネント群を用いて，時間を計測することでタイムアウトの通知を行なっている．同様に耐故障処理についても，コンポーネント群を用いて実現することが可能である．

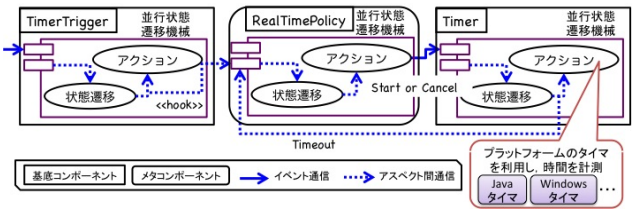


図 4 実時間処理の実現例

## 3 E-AoSAS++が対象とするプラットフォームの定義

### 3.1 プラットフォームの分類化

E-AoSAS++では，アプリケーションの開発支援のために MDA に基づくソースコードの自動生成を実現しており，プラットフォームを MDA における変動要因として扱う．MDA における変動要因とは，ハードウェアやプログラミング言語，オペレーティングシステム，利用可能なライブラリ群などである．これらを整理し，E-AoSAS++が対象とするプラットフォームは以下の 2 種類の組合せと定義する．

- プログラミング言語
- ソフトウェアプラットフォーム（オペレーティングシステムや仮想マシン）

従来の E-AoSAS++におけるプラットフォームは，プログラミング言語を対象としており，ソフトウェアプラットフォームが提供するライブラリの違いまで特定することを考慮していなかった．プラットフォームを 2 種類の組合せと定義したことは，プログラミング言語と同期処理や例外処理，時間計測処理に必要な機能を実現するライブラリや言語機能の特定を目的としている．MDA の変動要因の一つであるハードウェアは，E-AoSAS++においてハードウェア環境を含む支援を考慮していないので，除外する．

### 3.2 プラットフォームの整理

開発支援環境で実現してきたプラットフォームである C や C++，Java などのプログラミング言語と，POSIX，Windows，JVM，KVM などのソフトウェアプラットフォームを事例に整理を行なう．2 種類の選択するプラットフォームの関連は，実現不可能な組合せが存在することや，一対一に対応しないことから，従属であると考える．よって，2 層の階層構造を用いてプラットフォームを定義した．図 5 に定義したプラットフォームを示す．

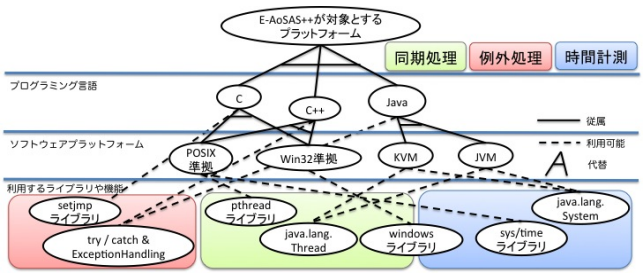


図 5 プラットフォームの階層構造を用いた整理

## 4 マルチプラットフォーム化への対応の課題と解決手段

### 4.1 プログラミング言語独立の実現

プログラミング言語独立とは，プログラミング言語の構文の違いに対処することである．言語ごとに，実現する要素との対応関係を明確にすることで対処が可能である．実行時核は，並行に動作するオブジェクトを実現に用いる．対象とした言語とオブジェクト実現の対応関係の一部を表 1 に示す．手続き指向のプログラミング言語である C 言語は，手続き指向でソースコードを実現する方が自然であるが，複数のプラットフォームにおいて統一的に扱うことを目的としているので，疑似オブジェクト指向コードで実現を行なう．

表 1 構文の対応関係の一部

| 実現する要素 ＼ 言語 | Java | C++ | C |
|---|---|---|---|
| クラス | class | class | struct |
| インタフェース | interface,implements | virtual 関数と多相型 | ヘッダファイルの include |
| 多相性 | Sub extends Base | class Sub : public Base | struct Sub{Base b; … } |

### 4.2 ライブラリや言語機能独立の実現

ライブラリや言語機能独立とは，ライブラリや言語機能が提供する実現方法の違いに対処することである．コンポーネントに対して，共通なインタフェースを定義することが対処の第一歩である．プログラミング言語の意味や実現方法の違いについては，定義したインタフェースとの関係を整理することで，対処が可能であると考えている．インタフェースと意味を整理することで，表 2

に示す Thread の Signal / Wait 方式による同期処理の実現方法の違いを吸収することが可能となる．

表 2 Signal / Wait の実現方法の違い

| ライブラリ ＼ Thread への操作 | signal | wait |
|---|---|---|
| pthread Library | pthread_cond_signal | pthread_cond_wait |
| windows Library | setEvent | WaitForSingleObject |
| java.lang.Thread Library | notify | wait |

## 5 実行時核のアーキテクチャの再定義

現行のアーキテクチャと定義したプラットフォームから，以下の課題を特定した．

- Signal，Wait 操作を実現するコンポーネントの明確化
- 共有資源を管理するためロック機能の細分化
- 並行処理モニタ（スケジューラ）の追加

再定義したアーキテクチャでは，同期処理を Signal / Wait 方式で標準化し，スケジューリングポリシの追加とロック機能の細分化を行なった．同期処理の選択では，並行に動作するオブジェクトとの親和性や実現可能性の観点から，Signal / Wait 方式を採用した．プラットフォーム毎に異なるスレッドのスケジューリングポリシに対処するために，スケジューラのモデル化を行なった．キューのロジックとロック機能を分離することで，プラットフォーム毎に実現を変更する必要があるコンポーネントを細分化している．このように再定義したアーキテクチャでは，ライブラリや言語機能に依存するコンポーネントを局所化し，変動箇所を明確にした．図 6 に再定義したアーキテクチャを示す．その他のアスペクトは，プラットフォームに依存した機能を利用して実現を行なわないので，再定義の必要はない．

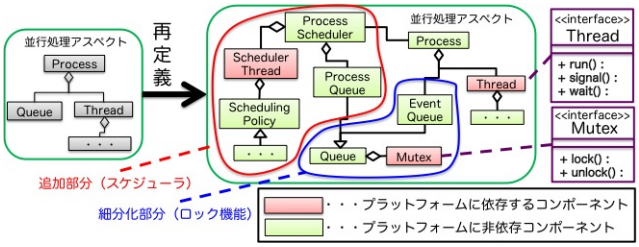


図 6 並行処理アスペクトとインタフェースの一部

## 6 非機能特性を実現するコンポーネント群の再定義

現行の耐故障処理や実時間処理を実現するコンポーネント群の再定義の課題を以下に示す．

- 標準化のために，目的によって異なる処理の変更に対処可能
- ライブラリや言語機能を利用するコンポーネントの明確化

基底コンポーネントは，非機能特性を実現するためにコンポーネント群を利用する．非機能特性を実現するコンポーネント群では，故障やタイムアウト時にメタコンポーネントへイベントを通知する．通知の一実現として，ライブラリや言語機能に依存する例外処理機能を用いる．

### 6.1 耐故障処理を実現するコンポーネント群の再定義

耐故障処理では，実時間制約やハードウェア環境を考慮し，N バージョンプログラミングやリカバリブロック，N セルフチェッキングプログラミングの 3 種類の処理方法を選択する [1]．E-AoSAS++における耐故障処理の対象は，ハードウェアの一時的な誤動作による誤ったイベントの受信である．処理方法を決定するポリシである Decider コンポーネントの実現を変更することで，処理の変更に対処可能である．図 7 に再定義した耐故障処理を実現するコンポーネント群を示す．

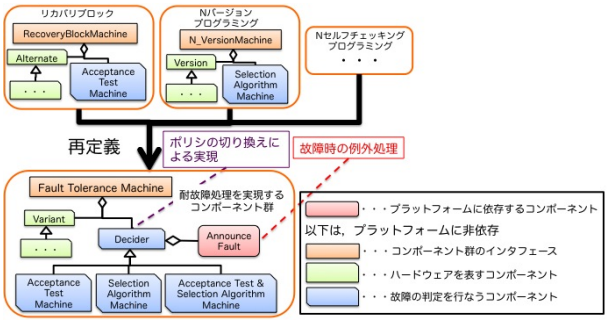


図 7 耐故障処理を実現するコンポーネント群の再定義

### 6.2 実時間処理を実現するコンポーネント群の再定義

実時間処理では，タイムアウト発生時にシステムが与える損害の度合いを考慮し，ハードリアルタイム処理とソフトリアルタイム処理の 2 種類の処理方法を選択する．再定義前のコンポーネント群は，ソフトリアルタイムを実現している．ハードリアルタイムのタイムアウト発生時の処理内容を他のライブラリや言語機能を利用しないシステムを停止させるなどの処理に限定することで，タイムアウト処理の実現を変更することによって対処が可能である．図 8 に再定義した実時間処理を実現するコンポーネント群を示す．

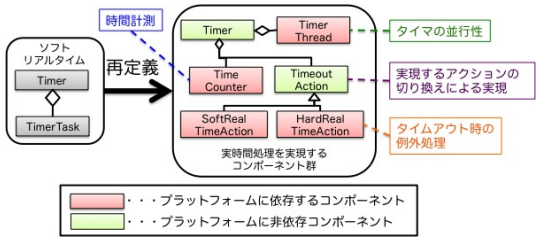


図 8 実時間処理を実現するコンポーネント群の再定義

## 7 考察

### 7.1 標準化に関する考察

実行時核の並行処理アスペクトの標準化が妥当であるのか考察を行なう．並行処理アスペクトは，同期処理を Signal / Wait 方式で標準化している．同期処理の代替

案として，セマフォの PV 操作による実現が考えられる．PV 操作は，セマフォに対する操作であり，特に並行処理アルゴリズムにおいて共有資源の管理を行なうことに適している．実行時核は，並行に動作するオブジェクトの集合として実現される．並行に動作するオブジェクトをプロセスと捉えれば，その同期処理には Signal / Wait 方式が適している．また，整理した標準的なプラットフォームを用いて，標準化した必要な機能は全て実現可能であることを確認している．よって，標準化は妥当であると考える．

### 7.2 プラットフォームの定義に関する考察

対象とするプラットフォームをプログラミング言語とソフトウェアプラットフォームの 2 種類の組合せで定義したことが妥当であるのか考察を行なう．整備してきた開発支援環境では，組込みシステムにおいて標準的なプログラミング言語とソフトウェアプラットフォームを用いた．プログラミング言語として C#を，ソフトウェアプラットフォームとして Web ベースのアプリケーションを取り入れた環境である.NET を用いた際も，矛盾なく定義が可能である．図 9 に，C#と.NET を追加した際の同期処理の実現に対するプラットフォームを示す．本研究で

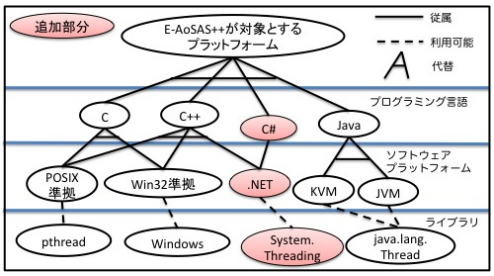


図 9 C#と.NET を取り入れたプラットフォームの一部

定義した 2 層の階層構造では，.NET のみの追加を考えた場合に対応することが可能である．再定義前のプラットフォームをプログラミング言語の選択とした対応では，ライブラリの違いまでを表現することができず，マルチプラットフォーム化への対応の実現には至らない．二つの異なるプラットフォームの追加に対して，矛盾なく定義することが可能である．その他の処理についても同様に定義可能であると考えられるので，妥当であるといえる．

### 7.3 マルチプラットフォーム化への対応に関する考察

実行時核のアーキテクチャを再定義することによって，マルチプラットフォーム化への対応の実現が可能であるか考察を行なう．図 10 に，再定義した実行時核のアーキテクチャを用いたマルチプラットフォーム化への対応の実現を示す．再定義後のアーキテクチャでは，プラットフォームをプログラミング言語とソフトウェアプラットフォームの組合せとし，インタフェースに従い実装することで，マルチプラットフォーム化への対応を実現している．実行時核では同期処理を標準化しプラットフォーム毎に共通なアーキテクチャの再定義を行なったが，代替案としてプラットフォーム毎に適応する実現のパターン

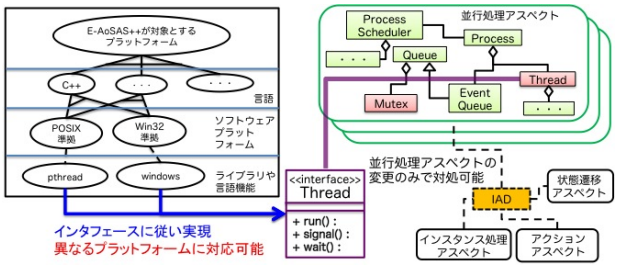


図 10 実行時核のマルチプラットフォーム化の実現

を定義し提供する方法が考えられる．後者の方法の問題点は，パターン数のアーキテクチャを定義する．パターンの定義と実現を行なう必要があり，前者と比較するとプラットフォームの変動の際にかかるコストが高くなると考えられる．再定義したアーキテクチャを利用することによって，マルチプラットフォーム化の対応の実現が可能であることを確認した．非機能特性を実現するコンポーネント群も同様に位置づけることで実現が可能である．

## 8 おわりに

本研究では，実行時核のアーキテクチャと耐故障処理と実時間処理を実現するコンポーネント群を再定義し，マルチプラットフォーム化への対応について考察を行なった．今後の課題として，定義したコンポーネントのインタフェースとプログラミング言語の意味の違いの関係について考察をすることがあげられる．

## 参考文献


[1] J. Laprie, J. Arlat, C. Beounes, and K. kanoun, “Definition and Analysis of Hardware and Software-Fault-Tolerant Architectures,” *IEEE Computer Society,* vol. 23, pp. 39-51, 1990.

[2] K. Pohl, G. Bockle, and F. Linden, *Software Product Line Engineering Foundations, Principles, and Techniques,* Springer-Verlag, 2005.

[3] M. Noro, A. Sawada, Y. Hachisu, and M. Banno, “E-AoSAS++ and its Software Development Environment,” *Proceedings of the 14th Asia-Pacific Software Engineering Conference(APSEC 2007),* pp. 206-213, 2007.

[4] Object Management Group(OMG), “Model Driven Architecture,” http://www.omg.org/mda/, 2011.

[5] P. A. Hsiung, and C. Y. Lin, “VERTAF: An Application Framework for the Design and Verification of Embedded Real-Time Software,” *Proc. IEEE Transaction on Software Engineering,* vol. 30, no. 10, pp. 656-674, 2004.

[6] 伊藤英樹，“自動販売機制御ソフトウェアの再開発 －核資産の定義とコード生成ツールの設計と試作－，” 南山大学大学院数理情報研究科 2010 年度 修士論文 (OJL 成果報告書)，2011.