2009－SE－163 (15)
2009／3／18

# E-AoSAS++に基づく開発支援環境
## — コード生成ツールの提案 —

長 大介[*1]，加藤 大地[*1]，蜂巣 吉成[*2]，沢田 篤史[*2]，野呂 昌満[*2]

[*1] 南山大学大学院数理情報研究科
[*2] 南山大学数理情報学部
〒 489–0863 愛知県瀬戸市せいれい町 27
{m08mm027,m07mm011,hachisu,sawada,yoshie}@nanzan-u.ac.jp

ソフトウェアアーキテクチャは，ソフトウェアの構造を多様な視点から抽象的に記述したものであり，ソフトウェア開発の様々な場面で利用される．我々はソフトウェアアーキテクチャ設計をすべての開発行為の基礎と位置付け，これまでにアスペクト指向に基づくアーキテクチャスタイル E-AoSAS++を提案してきた．

本研究では，E-AoSAS++にしたがって記述されたソフトウェアアーキテクチャからモデル駆動アーキテクチャの考え方に基づいてプログラムコードを生成するツールの設計と実装について説明する．

# Software Development Environment based on E-AoSAS++
## — Design of the Code Generation Tool —

Daisuke OSA[*1], Daichi KATO[*1]
Yoshinari HACHISU[*2], Atsushi SAWADA[*2], Masami NORO[*2]

[*1] Graduate School of Mathematical Sciences and Information Engineering, Nanzan University
[*2] Faculty of Mathematical Sciences and Information Engineering, Nanzan University
27 Seirei-cho, Seto-shi, Aichi, 489-0863 Japan
{m08mm027,m07mm011,hachisu,sawada,yoshie}@nanzan-u.ac.jp

Software architecture which describes the structure of software from multiple viewpoints is one of the most important artifacts throughout development process. We have proposed the aspect-oriented software architecture style E-AoSAS++ (Aspect-oriented Software Architecture Style for Embedded systems) for a basis of systematic and integrated support environment for software development.

In this article, we explain the design and implementation of a MDA-based code generation tool for the software of which architecture is described using E-AoSAS++.

## 1 はじめに

ソフトウェアアーキテクチャは，アプリケーションドメインに関する知識をソフトウェアの基本構造として表現したものであり，分析・設計，検証，評価，保守，再利用など，あらゆるソフトウェア開発作業の基礎をなす要素として利用される．ソフトウェア開発者がアーキテクチャを利用する立場は多様であるが，アプリケーション開発者にとってのアーキテクチャは，設計・実装のための骨格を記述したものであり，アーキテクチャに個々のアプリケーションに依存する部品を付け加えることで開発が行われる．

アーキテクチャ記述は，ソフトウェア構造やそれに対する制約条件を，静的側面，動的側面，開発プロセスの側面など，さまざまな視点から抽象化して表現したものである [1]．我々は，複数視点からのソフトウェア構造とそれらの間の関係を効果的に表現することができる技術として，アスペクト指向に着目し，E-AoSAS++ (Aspect-oriented Software Architecture Style for Embedded systems) と呼ぶアスペクト指向ソフトウェアアーキテクチャスタイルを提案し，それに基づく開発環境を整備している [2]．E-AoSAS++では，アスペクトを用いてモジュール化が行なわれ，モジュール間の包含関係によって静的な分割構造を，モジュール間通信によって使用や参照などの依存関係を表現する．

本稿で我々は，E-AoSAS++にしたがって文書化されたソフトウェアアーキテクチャ記述にアプリケーション依存の論理を加えた設計記述からプログラムコードを自動生成するツールを提案する．ツールの設計にあたっては，モデル駆動アーキテクチャ（MDA）[3] の考え方にしたがうモデル変換を利用する．すなわち，UML を拡張した E-AoSAS++のモデル（PIM）から，特定のプログラミング言語に依存しない実行コードのモデルを通じ，Java や C++などプログラミング言語（PSM）への変換を行う．

モデル変換論理の設計にあたっては，アーキテクチャの要素である並行状態遷移機械がアスペクトとして振舞い，互いに通信を行う際の論理を決定する必要がある．このような E-AoSAS++の操作的意味をプログラムコードとして実現するために，モジュール間に非同期でやりとりされるイベントと，それに伴うモジュールの状態遷移を実現する論理を特定のプログラミング言語に依存しない共通のプラットフォームモデルとして組み込んでいる点が提案ツールの特徴である．これにより，複数のプログラミング言語への対応とともに，E-AoSAS++の操作的意味の拡張や，非機能特性を考慮したチューニングにも柔軟に対応可能なツールとすることを狙っている．

本稿では，以下，2 章で E-AoSAS++とそれに基づく開発プロセス，E-AoSAS++におけるアーキテクチャ記述について説明した上で，3 章で E-AoSAS++モデルからコード生成を行うための PIM の設計について説明する．次いで，4 章で提案方式を MDA の概念に基づいて議論した後，5 章で本稿のまとめをする．

## 2 E-AoSAS++の概要

E-AoSAS++は，我々が提案するソフトウェアアーキテクチャのスタイルであり，特に組込みシステムのソフトウェアアーキテクチャを記述する目的で設計されている．

### 2.1 アスペクト指向アーキテクチャ

組込みソフトウェアにおいては，実時間処理に対する要求，耐故障性，並行性，ハードウェア制約など，さまざまに絡み合う関心事が存在しており，対象とするアプリケーションドメインの概念構造をそのままアーキテクチャ設計に反映させると，結果として類似した役割を持つモジュール（実装レベルではソースコード断片）が複数の要素にまたがって含まれるという横断的関心事の問題が発生する．この問題を解決するために，E-AoSAS++ではアーキテクチャ設計にアスペクト指向の考え方を導入している．

E-AoSAS++におけるアーキテクチャの基本構成要素であるモジュールは，並行状態遷移機械（CSTM）であり，複数 CSTM の階層的な包含関係によってソフトウェアの静的構造を，CSTM 間の通信によって使用や参照などの依存関係を表現する．個々の CSTM の振舞いは状態遷移論理により表現されており，複数の CSTM にまたがる振舞いは CSTM 間のメッセージ通信とその系列によって表現される．

E-AoSAS++では，これら CSTM に共通する関心事として，状態遷移および並行処理のための論理を，それぞれアスペクトとして記述する．これら共通の論理をアスペクトとし，CSTM 独自のアプリケーション論理から独立させ，アスペクト間記述（IAD: Inter Aspect Description）を介して疎結合させることで，柔

軟で再利用性の高いアーキテクチャ設計を可能とする．

## 2.2 開発プロセスと支援環境

E-AoSAS++を用いてアーキテクチャを設計し，そのアーキテクチャに基づいてプロダクトを開発するプロセスの概要を図 1 に示す．E-AoSAS++のドメイン工学プロセスでは，概念アーキテクチャおよび実装アーキテクチャと呼ぶ二種類のアーキテクチャを設計する．

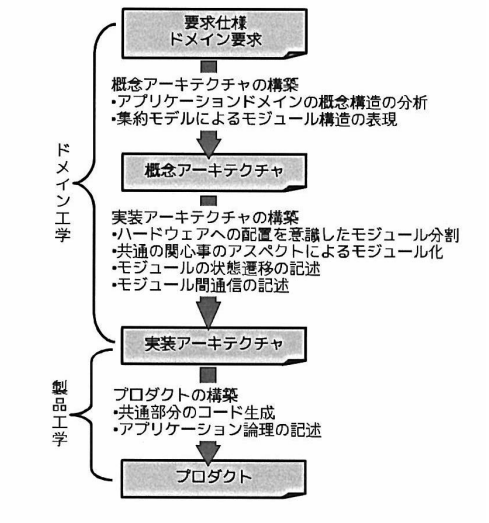


図 1: E-AoSAS++を用いた開発

概念アーキテクチャは，アプリケーションの抽象構造を表現するものであり，対象ドメインの概念構造を分析した結果を集約モデルとして表現する．このプロセスで抽出された個々の構成要素が，後述する実装モデルにおけるモジュール（CSTM）の候補となるが，この段階ではアスペクトへの分割は意識しない．

実装アーキテクチャは，共通する機能のアスペクトへの分割，利用可能なハードウェア構成などを想定して作成するアスペクト指向モデルであり，アスペクトとして抽出される CSTM の間の静的構造および依存関係，動的な振舞い，物理的な配置について記述する．実装アーキテクチャの設計においては，概念アーキテクチャにおいて分割されたモジュールを，アプリケーションが実行される時に利用されるハードウェアの物理的な構成や，複数のモジュールに横断する関心事などに基づいて CSTM に分割し，システムの静的構造を決定する．さらに，各 CSTM の振舞いと CSTM 間のやりとりについて記述することで，全体としてドメイン要求と整合したモデルを構成する．

製品工学プロセスでは，実装アーキテクチャにおける各モジュールの内部論理をプログラムコードに変換する．状態遷移や並行処理，リアルタイム処理など汎用のモジュールをプラットフォームモデルとして整備することで，MDA の考え方に基づくコード生成が可能となる．このような開発形態により，開発者は，対象アプリケーションに特有の処理論理を実装することに専念できる．

我々は，これら概念アーキテクチャと実装アーキテクチャからなるソフトウェアアーキテクチャ記述をソフトウェア開発の基礎と位置付け，図 2 に示すような，エディタ，実行前検査ツールなどを含む支援環境の整備を試みている．本稿で提案するコード生成ツールもこの支援環境の構成要素である．

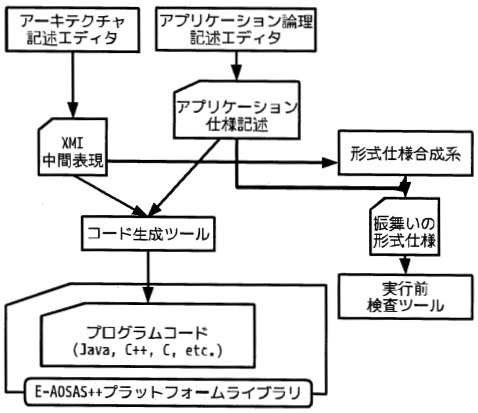


図 2: E-AoSAS++に基づく開発支援環境

## 2.3 アーキテクチャ記述方式

前節で述べたとおり，E-AoSAS++における記述対象となるアーキテクチャには，概念アー

キテクチャと実装アーキテクチャがあり，そのうち実装アーキテクチャは，システム静的構造モデル，モジュール振舞いモデル，モジュール間通信モデル，モジュール配置モデルの四種類のモデルを用いて表現される [4]．図 3 に E-AoSAS++アーキテクチャ記述と UML 図式との対応関係を示す．

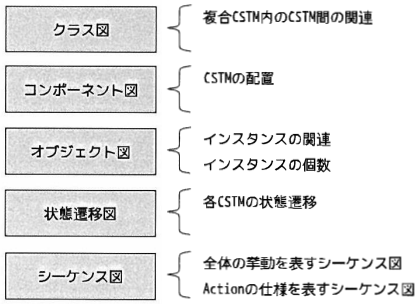


図 3: UML を用いた E-AoSAS++の文書化

E-AoSAS++における概念アーキテクチャは，図 4 に示すようなクラス図を用いて表現する．概念アーキテクチャにはアスペクト指向の概念が導入されていないので，図式の意味的拡張は行わず，主に集約関係を用いて CSTM 候補となる概念要素間の階層的包含関係を表現する．

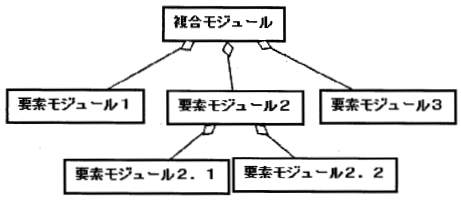


図 4: 概念アーキテクチャの表現

実装アーキテクチャのうちシステム静的構造モデルの表現には，図 5 に示すコンポーネント図を用いてモジュールの間の複合構成関係と依存関係を表現する．E-AoSAS++におけるモジュールはアスペクトであり，アスペクトの複合構成をコンポーネントの階層関係で表現し，横断的な依存関係を提供インタフェースと利用インタフェースの共有関係で表現する．コンポーネントを複合アスペクトの意味で用いていることを明示するために，新たに<<aggregation>>ステレオタイプが導入されている．また，アスペクトに対応する CSTM の詳細構造はクラス図を用いて表現される．

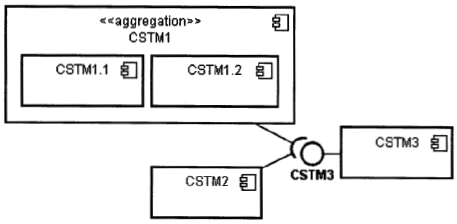


図 5: 静的構造モデルの表現

実装アーキテクチャのモジュール振舞いモデルでは，図 6 に示すとおり，状態機械図とシーケンス図を用い，イベント受け取りに伴う状態遷移とアクション起動の仕様を記述する．

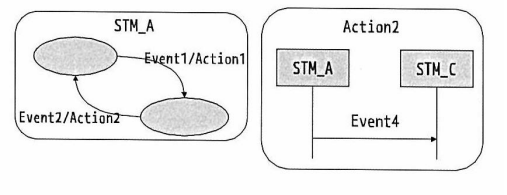


図 6: モジュール振舞いモデルの表現

実装アーキテクチャのモジュール間通信モデルは，図 7 のとおりシーケンス図を用い，CSTM 間の一連のやりとりを示す．ここで，これらのやりとりがアスペクト間記述を参照することで行われるのか，固定的に相手を指定したメッセージ通信によって行なわれるのかを示すステレオタイプ（<<IAD>>および<<MP>>）が導入されている．

ここまで記述した概念アーキテクチャ，実装アーキテクチャにしたがって構築されたアプリケーションが動作する時には，モデルに示された型定義にしたがって実体化されたモジュールのインスタンスがハードウェアや OS プロセスなどの計算主体に割り当てられる．個々のインスタンスがどのような計算主体に割り当てられ，他のモジュールとどのように関連するのかは，オブジェクト図を用いて記述される．

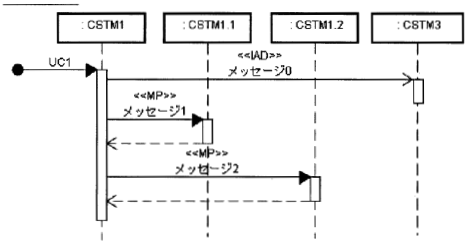


図 7: モジュール間通信モデルの表現

## 3 コード生成ツールの設計

本章では，E-AoSAS++にしたがって記述された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するツールの設計について述べる．コードの自動生成には，MDA の考え方に基づき，図 8 に示す二段階のモデル変換を利用する．すなわち，一段階目のモデル変換では，

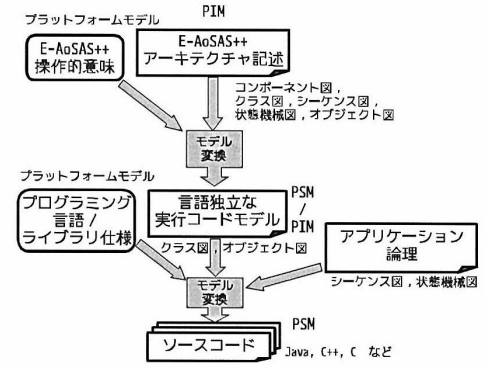


図 8: 二段階のモデル変換

前章で説明した UML を拡張した E-AoSAS++のアーキテクチャ記述（PIM）から，特定のプログラミング言語に依存しない実行コードのモデル（PSM）を生成する．このモデル変換においてプラットフォームモデルにあたるのが，E-AoSAS++の操作的意味であり，アスペクトで表現されたモジュールの振舞いとモジュール間の通信方式を定めるものである．PSM としての実行コードモデルは，クラス図とオブジェクト図として表現される．

二段階目のモデル変換では，前述の実行コードモデルを PIM とし，アプリケーションに依存した論理として作成された記述を加えることで，Java や C++などプログラミング言語（PSM）への変換を行う．この変換におけるプラットフォームモデルは，変換先のプログラミング言語仕様に，状態遷移やイベントの送受，並行処理などを実現するために利用できるライブラリの仕様を加えたものであり，それらのライブラリの利用を前提としたソースコードが生成される．

これら二つの変換のうち，二段階目の変換は，プログラミング言語とそこで利用可能なライブラリの仕様が決定されれば，単純な変換ルールによって実現可能であるのに対し，一段階目の変換については記述された PIM から，次の要素をいかに実行コードモデルとして実現するのかが重要となる．

- 並行状態遷移機械からなるモジュールの振舞い
- モジュールのインスタンス処理

本章では以下，アスペクトとしてのモジュール間に非同期でやりとりされるイベントとそれに伴うモジュールの状態遷移を実現する論理，型定義より生成されるモジュールのインスタンスを管理する論理を反映した実行コードモデルの設計を中心に説明する．このような設計方針を採用することで，複数のプログラミング言語への対応とともに，E-AoSAS++の操作的意味の拡張や，非機能特性を考慮したチューニングにも柔軟に対応可能なツールとすることを狙っている．

### 3.1 モジュール振舞いモデル

図 9 に，E-AoSAS++におけるモジュールとアスペクトとの関係を示す．一つの CSTM モジュールを構成するアスペクトは，並行処理，状態遷移，アプリケーション論理の三つであり，それらの間はアスペクト間記述（IAD）を介して結合している．並行処理や状態遷移に関する処理，IAD を介した通信に関する処理については，モジュールが実現するアプリケーション論

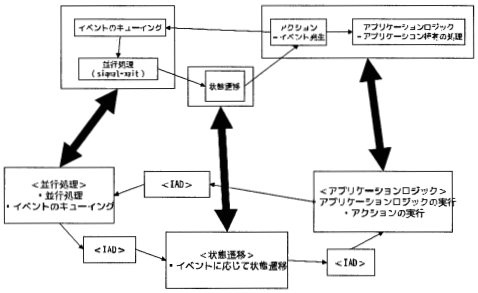


図 9: アスペクトを用いたモジュールの表現

理からは独立であり，アーキテクチャを記述することでその内容を導出することができる．

図 10 には，実装アーキテクチャのモジュール振舞いモデルと実行コードモデルとの対応付けを示す．一つのモジュールとしての CSTM の

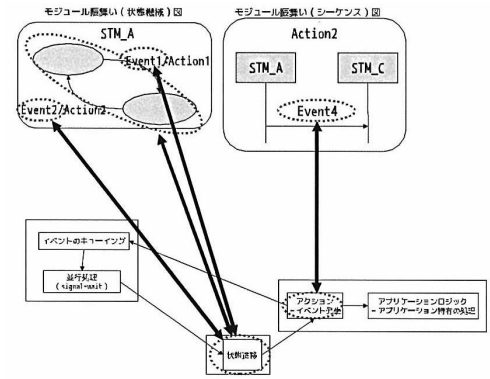


図 10: 状態遷移論理

振舞いは，アーキテクチャ記述上では状態機械図で表現され，これに対応する状態と受信イベントの定義は状態遷移アスペクトとして実現する．また，状態遷移に伴って他のモジュールへ送信されるイベントの仕様は，アーキテクチャ記述ではシーケンス図によって表現され，アプリケーション論理に対応するアスペクトの一部として実現する．状態遷移に伴って実行されるアクションの詳細については，アーキテクチャとしては記述せず，実行コードモデルにはスタブとして反映される．

一方，アスペクト間のやりとりに関しては，実装アーキテクチャのモジュール通信モデルとしてシーケンス図を用いて記述される．並行して動作する CSTM 間のイベントのやりとりについては，並行処理アスペクトが担当し，異なる CSTM にまたがるイベントは，図 11 に示すように IAD により中継する．すなわち，ある

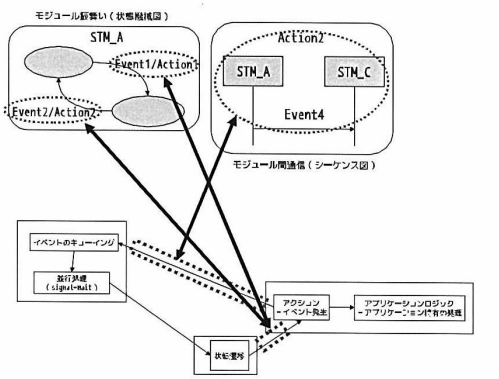


図 11: IAD を用いたイベントの中継

CSTM でアクションの実行に伴い発生するイベントの通知先は，アプリケーション論理アスペクトと並行処理アスペクトとの間の IAD が管理し，通知先 CSTM におけるアクションの実行は状態遷移アスペクトとアプリケーション論理アスペクト間の IAD がそれぞれ管理する．

これらの方針を反映して設計した実行コードモデルの雛型を図 12 に示す．図には一つの

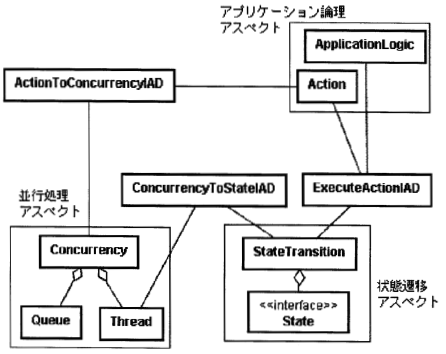


図 12: CSTM の実行コードモデル

CSTM に対応するクラス図の構成のみが示されているが，実際にはアーキテクチャ記述の要

素である各 CSTM に対応する状態遷移アスペクトが生成され，さらに各 CSTM に定義されるアクションに対してアプリケーション論理アスペクトが生成されることになる．アスペクト間を仲介するそれぞれの IAD には，例えばある状態遷移に対してどのアプリケーション論理が対応するのかなど，アスペクトの対応関係を表形式で管理する設計としている．

### 3.2 インスタンス処理モデル

E-AoSAS++の実装アーキテクチャ記述のうち，静的構造モデルとモジュール振舞いモデル，モジュール間通信モデルは，CSTM の型の構造や振舞いを示したものである．実際にアプリケーションが実行される際には，この型に従ったモジュールのインスタンスが生成され，プロセスやスレッド，ハードウェアなどの計算主体に割り当てられることになる．

一つのモジュールの型から複数のインスタンスが生成される場合には，それらは区別して扱われなければならないので，実行コードモデルにおいても，CSTM におけるアクションの実行主体やイベントの通知先について，複数のインスタンスの存在を扱うことが必要となる．

この要求に対応するために，図 13 に示すように，実装アーキテクチャ記述の中のモジュール配置図の記述に基づいて CSTM インスタンスとそれらの間のイベントのやりとりを管理するための，インスタンス処理アスペクトを導入する．インスタンス処理アスペクトでは，ある

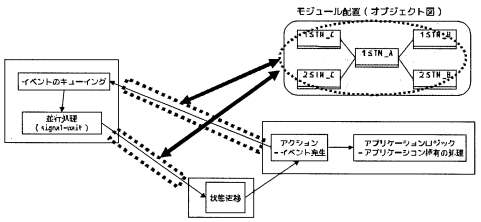


図 13: モジュール配置図とインスタンス処理

CSTM において発生したイベントの通知先インスタンスを決定する際に用いることができるよう，モジュール配置図に示された CSTM インスタンスを表形式で保持する設計とした．このインスタンス処理アスペクトを導入した結果，CSTM の実行コードモデルは図 14 のようになる．InstanceTable クラスで示されるイン

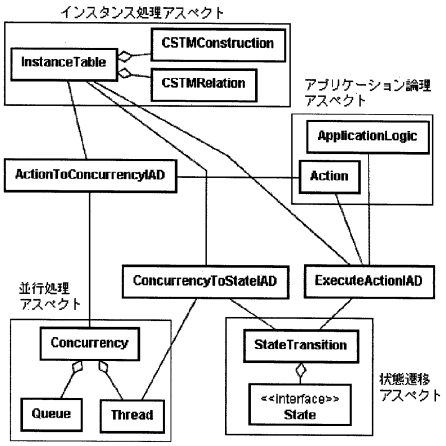


図 14: インスタンス処理を加味した実行コードモデル

スタンス表を構成する CSTMConstruction クラスと CSTMRelation クラスは，それぞれモジュール配置図に示されたインスタンスとそれらの間の関係に対応する要素である．

### 3.3 ツールの実装

ここまで説明した方針にしたがい，UML を用いて記述した E-AoSAS++のアーキテクチャモデルから実行コードモデルを生成するツールを試作した．本ツールでは，UML エディタとして Enterprise Architect[1] を用いることを前提とし，これが出力する XMI 記述からクラス，オブジェクト，状態，ライフライン，メッセージなどに対応する識別子を抽出し，各 CSTM に対し図 13 に相当するクラスモデルを生成する．

実行コードモデル生成ツールとともに，実行コードモデルから Java のソースコードを生成するツールの試作も行い，単純な例題を用い，正しく動作するソースコードが生成されることを確認している．

[1] http://www.sparxsystems.jp/ea.html

## 4 考察

本ツールでは，前章の図8に示した通り，二段階のモデル変換を通じてアーキテクチャモデルからプログラムコードを生成する．

第一段目のモデル変換では，アスペクトとして表現されるモジュール間の通信とモジュールの振舞いに関する操作的意味をプラットフォームと位置付けることで，E-AoSAS++の意味の変更に対するツールの柔軟性を確保することを狙っている．これにより，実行性能など非機能要求に応じたコード生成方針の動的な変更に対応することができる．例えば，前章に示した実行コードモデルでは，モジュール間の通信は全てIADによる表の間接参照を介するものとして生成されるが，インスタンスが1個のみのモジュールに対しては直接参照による通信を採用し，実行効率を上げるなどのチューニングも可能である．

第二段目のモデル変換では，プログラミング言語の仕様およびライブラリの仕様をプラットフォームととらえることで，プログラミング言語やOS，ライブラリに対する柔軟性を確保できる．特にE-AoSAS++が主な支援対象とする組込みソフトウェアでは，実行プラットフォームやプログラミング言語が多様であることから，この構成は妥当であると考えている．

ただし，我々が試作したツールにおいては，いずれの段階のモデル変換についてもその論理がプログラムに組み込まれて実装されている点は問題であり，モデル間のマッピングを形式的に記述する方式を導入し，それを解釈実行することでモデル変換を行うよう改良する必要がある．

## 5 おわりに

本稿では，アスペクト指向に基づくソフトウェアアーキテクチャスタイルE-AoSAS++に基づくアーキテクチャ記述から，MDA概念に基づいてソースコードを生成するツールについて，アーキテクチャ記述とソースコードを繋げるための中間形式である実行コードモデルの設計を中心に説明した．本ツールの有効性や妥当性については，実アプリケーションに近い例題を記述して確認することが有効であり，現在，MDDロボットチャレンジでの飛行船自律制御課題[5]を例に作業を行っている．

4章でも考察した通り，試作した生成ツールでは，モデル変換論理が固定的に記述されている点が問題で，形式的記述を用いることでこれを柔軟に変更できるように改良することも今後の重要な課題であると考えている．

**謝辞** 本研究の一部は，平成20年度科学研究費補助金（基盤研究(C)（一般）19500028）および平成20年度南山大学パッヘ奨励金I-A-2の助成による．

## 参考文献

[1] ISO/IEC 42010 IEEE Std 1471-2000, *Systems and Software Engineering – Recommended Practice for Architectural Desciption of Software-intensive Systems*, 2007.

[2] Noro, M., Sawada, A. Hachisu, Y. and Banno, M.: "E-AoSAS++ and its Software Development Environment," *Proceedings of the Asia-Pacific Software Engineering Conference (APSEC2007)*, pp. 206–213, IEEE Computer Society, 2007.

[3] Mellor, S. J., Scott, K., Uhl, A. and Weise, D., *MDA Distilled — Principles of Model-Driven Architecture,* Addison-Wesley, 2004.

[4] 加藤大地，蜂巣吉成，沢田篤史，野呂昌満："アスペクト指向に基づくソフトウェアアーキテクチャの文書化方式，" 電子情報通信学会技術研究報告，知能ソフトウェア工学研究会(KBSE)，2009．(to appear)

[5] MDDロボットチャレンジ編集委員会編，MDDロボットチャレンジ2005：産学連携によるモデルベース組込み開発の実践，情報処理学会，2006.