組込みソフトウェア開発環境に関する研究
– コード生成ツールのアーキテクチャの再設計 –

2006MI142 尾関 孝道　　2008MI141 水谷誠孝
指導教員　野呂 昌満

1　はじめに

組込みソフトウェア開発において，アーキテクチャからプログラムコードを自動生成し，開発労力を削減することが試みられている．アーキテクチャからプログラムコードを自動生成する手法として，モデル駆動型アーキテクチャ (以下，MDA) が提案されている [3]．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++) を提案している [4]．小池らの論文 [5] では，E-AoSAS++ が提案する開発支援環境のひとつとして，MDA に基づくコード生成ツールが提案されている．

提案されているコード生成ツールの設計は，Java のみを考慮している．MDA プラットフォームが変更された際は，Java に依存した箇所を修正しなければならないので，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，MDA プラットフォームの変更に伴うコード生成ツールの変更箇所を局所化できるようなモデル変換系のアーキテクチャの提案である. そのようなアーキテクチャを提案することで，MDA プラットフォームの変更に柔軟に対応可能にする．

本研究では，GoF のデザインパターン [1] をモデル変換系に適用して，コード生成ツールのアーキテクチャを再設計した．適用するデザインパターンは，パターンカタログに基づき，コード生成ツールに求められる品質特性について評価を行うことで決定した．デザインパターンを適用することで変更箇所を局所化し，アーキテクチャの進化を図った．モデル変換論理の処理を部品化して組み合わせることで，モデル変換論理の処理内容の構築を目指す．このように再設計を行なうことで，MDA プラットフォームの変更に伴う影響を小さくし，コード生成ツールの開発労力を削減する．本研究の手順を以下に示す.

1. 提案されているコード生成ツールの問題点の整理
2. コード生成ツールに求められる品質特性の特定
3. コード生成ツールのアーキテクチャ設計
   - 適用するデザインパターンの決定
     (a) 品質特性を基にしたデザインパターンの評価
     (b) 関連するデザインパターンとの比較
   - デザインパターンの適用
4. モデル変換論理の処理の部品化

本研究で行なったアーキテクチャの再設計の妥当性は，再設計の代替案と比較することで確認した．

2　E-AoSAS++

E-AoSAS++ では，システムの振舞いを並行に動作する状態遷移機械 (以下，CSTM) の集合で表す．CSTM は状態遷移アスペクト，並行処理アスペクト，アクションアスペクトで構成される．各 CSTM が，イベントキューを介したイベントのやりとりで協調動作する事でシステムとしての機能を実現する．CSTM の並行処理や状態遷移機械としての振舞い，CSTM 間の通信方法などをプラットフォームコードとして設計されている．

2.1　プラットフォームコードモデル

プラットフォームコードモデルは，E-AoSAS++ の振舞いをプラットフォームコードで実現するための枠組みである．E-AoSAS++ は，ソフトウェアに散在する関心事をアスペクトとしてモジュール化する．

2.2　E-AoSAS++ に基づくコード生成ツール

コード生成ツールは，MDA の概念に基づき，2 段階のモデル変換を行っている．2 段階のモデル変換を行なうことにより，複数の MDA 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できる．このことにより，コード生成ツールは開発工程の短縮化を実現している．2 段階のモデル変換では，E-AoSAS++ のアーキテクチャ記述を入力として, プラットフォーム独立モデル (以下, PIM) に変換する．次に PIM からプラットフォーム特化モデル (以下，PSM) へと変換し，PSM からソースコードを出力する．E-AoSAS++ では，PIM はプラットフォームコードの抽象モデル，PSM は MDA プラットフォームごとに実現されたプラットフォームコードと対欧関係にある．提案されているコード生成ツールにおける PIM と PSM はそれぞれ対応関係を持っている．本研究において対象とする MDA プラットフォームは C，C++，Java である．

3　コード生成ツールの再設計

本章では，MDA プラットフォームの変更に伴う変更箇所を局所化できるようなコード生成ツールのモデル変換系のアーキテクチャを提案する．

3.1　問題分析

既存のコード生成ツールのアーキテクチャは Composite パターンと Interpreter パターンを適用している．図 1 に既存のアーキテクチャにおいて MDA プラットフォームが変更された際の様子を示す．

MDA プラットフォームが変更されるとモデル変換論理の処理を変更する必要がある．この際，モデル変換論理の処理が各構文要素に散在しているので変更に労力

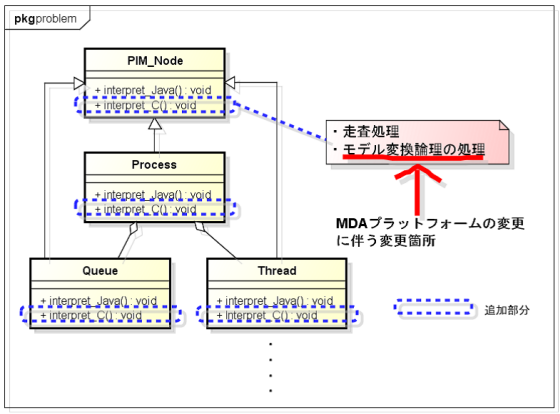

図 1　提案されているコード生成ツールにおける MDA プラットフォームの変更

がかかり，他の構文要素に影響を与える．この点から，MDA プラットフォームの変更に柔軟に対応できない．本研究では，コード生成ツールのモデル変換論理の処理を対象として GoF のデザインパターンを適用することで問題点の解決を図る．

3.2　コード生成ツールに対する要求

既存のコード生成ツールの問題点から，求められる要求を整理し，要求に対応する品質特性および品質副特性を特定した．本研究において，コード生成ツールに求められる要求を以下に示す．

- コード生成ツールが対象とする MDA プラットフォームが変更，追加された場合，モデル変換論理の処理を容易に変更，追加できるようにする
- コード生成ツールのモデル変換の処理が変更，追加された際，走査の処理や他のモデル変換論理の処理，データ構造に影響を与えない

これらの要求から，我々は ISO9126[2] を基に，コード生成ツールに求められる品質特性を特定した．要求から，コード生成ツールのモデル変換論理の処理に保守性が求められると考えた．求められる品質特性を以下に示す．

- モデル変換論理の処理に対する変更性
- モデル変換論理の処理に対する解析性
- モデル変換論理の処理の変更による安定性

3.3　コード生成ツールのアーキテクチャ設計

GoF のデザインパターンはオブジェクト指向で広く使われており，品質特性を向上させることができる．本研究では，GoF のデザインパターンをモデル変換論理の処理に適用することでアーキテクチャを再設計した．

3.3.1　デザインパターンの評価

デザインパターンのパターンカタログ [1] に基づき，生成系を除いた 18 種類のデザインパターンの評価を行なった．この評価は，コード生成ツールのモデル変換論理の処理にデザインパターン適用した際の品質特性の向上が求められる品質特性に適合するかという観点から行なった．評価の結果を表 1 に示す．求められる品質特性に適合するものには’+’ で，影響がないものには’=’ で表現し，適用することで求められる品質特性に悪影響を及ぼすものには’ ×’ で表現した．また，モデル変換論理の処理に適用できないと考えたデザインパターンについては’/’ で表現した．

表 1　モデル変換論理の処理に関するデザインパターンの評価

デザインパターン	変更性	解析性	安定性
Adapter パターン	×	×	×
Bridge パターン	+	+	×
Chain of Resposibility パターン	+	×	×
Command パターン	+	+	+
Composite パターン	/	/	/
Decorator パターン	+	×	+
Facade パターン	/	/	/
Flyweight パターン	/	/	/
Interpreter パターン	/	/	/
Iterator パターン	/	/	/
Mediator パターン	×	×	+
Memento パターン	=	=	=
Observer パターン	+	=	+
Proxy パターン	+	×	+
State パターン	+	+	=
Strategy パターン	+	+	=
Template Method パターン	+	+	=
Visitor パターン	+	+	+

3.3.2　関連するデザインパターンの比較

表 1 の評価の結果から選択肢を抽出出来るが，選択肢が多く，適用するデザインパターンを円滑に決定できない．関連するデザインパターンとの比較評価を行なうことで選択肢を絞り．適用するデザインパターンの決定を円滑にする．コード生成ツールのモデル変換論理の処理にデザインパターンを適用した際の品質特性の向上が，どちらのデザインパターンで優れているかという観点から比較評価を行なった．表 2 から表 6 に，比較評価を行ったデザインパターンと，比較評価を行った結果を示す．品質特性に優れると考えたものには’優’ で，劣ると考えたものには’劣’ で表現した．また，変わらないと考えたものには’=’ で表現した．

表 2　Proxy パターンと Decoretor パターンの比較

デザインパターン	変更性	解析性	安定性
Proxy パターン	劣	=	=
Decoretor パターン	優	=	=

表 3　Decoretor パターンと Strategy パターンの比較

デザインパターン	変更性	解析性	安定性
Decoretor パターン	＝	劣	優
Strategy パターン	＝	優	劣

表 4　Strategy パターンと Template Method パターンの比較

デザインパターン	変更性	解析性	安定性
Strategy パターン	＝	優	優
Template Method パターン	＝	劣	劣

表 5　State パターンと Strategy パターンの比較

デザインパターン	変更性	解析性	安定性
State パターン	劣	＝	＝
Strategy パターン	優	＝	＝

表 6　Strategy パターンと Visitor パターンの比較

デザインパターン	変更性	解析性	安定性
Strategy パターン	＝	劣	劣
Visitor パターン	＝	優	優

3.3.3　デザインパターンの決定

表 1 から，評価の結果，適用できないと考えたデザインパターンと適用することで求められる品質特性に悪影響を及ぼすパターンを選択肢から除外する．表 2 から表 6 から，比較評価して劣る選択肢を除外する．以下に最終的な選択肢として抽出したデザインパターンを示す．

- Command パターン
- Observer パターン
- Visitor パターン

選択肢を複数適合した際の，Command パターンと Visitor パターンの複合も選択肢として抽出した．抽出した選択肢について比較を行なうことで，コード生成ツールのモデル変換論理の構築に適用するデザインパターンを決定した．表 7 に Command パターンを採用したと仮定し，他の抽出したデザインパターンの選択肢との比較を示す．求められる品質特性が Command パターンを適用した際と比較して向上するものには'+' で，低下するものには'-' で，変わらないものには'=' で表現した．

表 7 より，変更性と解析性に優れており，安定性の向上に適合するので，Visitor パターンと Command パターンの複合がコード生成ツールには適していると判断した．

3.3.4　モデル変換論理の構築

求められる品質特性に基づいた評価から，われわれは Visitor パターンと Command パターンの複合をモデル変換系に適用した．適用後のアーキテクチャを並行処理アスペクトを一例に図 2 に示す．

各構文要素に対するモデル変換論理の処理を，InterpretVisitor インタフェースのサブクラスとして実現できるように設計した．モデル変換論理の処理を，構文要素が Visitor を訪問するメソッドで実現しており，構文要素とその走査処理からモデル変換論理の処理を分離して Visitor に集約している．具象 Visitor に集約されたモデル変換処理のうち，複数の MDA プラットフォームと構文要素に横断している処理をまとめられるように設計した．

3.4　モデル変換論理の部品化

モデル変換論理の処理を部品化することで，MDA プラットフォームの変更に，より柔軟に対応可能な設計にし，開発労力の削減を図る．MDA プラットフォームにかかわらず共通な処理が横断しており，これを部品とする．部品化した処理を，具象 Visitor の visit メソッド上で組み合わせてモデル変換論理の処理を構築することができるように設計した．

4　考察

4.1　アーキテクチャ設計の妥当性に関する考察

再設計したアーキテクチャが MDA プラットフォームの変更に柔軟に対応可能であるか考察する．提案されているコード生成ツールのアーキテクチャにおいて MDA プラットフォームが変更された際について述べる．3 章で示した通り，処理の変更箇所が各構文要素に散在しているので，コード生成ツールの再実現は，構文要素に関連して労力がかかる．再設計の代替案として，3 章で評価を行なった Observer パターンを挙げる．代替案を適用したアーキテクチャにおいて MDA プラットフォームが変更された際は，変更を行なうことによるデータ構造への影響は少ない．しかし，モデル変換論理の振舞いの解析性が低いので，柔軟に対応可能できないと考える．再設計したアーキテクチャは，3 章で示した通り，モデル変換系に Visitor パターンと Command パターンを適用している．再設計したモデル変換系のアーキテクチャにおいて，MDA プラットフォームが変更された際の，モデル変換論理を変更する様子を図 3 に示す．

モデル変換論理の処理は MDA プラットフォームごとに用意された具象 Visitor に集約しており，訪問する具象 Visitor を変更するだけですべての構文要素のモデル変換論理の処理を変更することができる．訪問する具象 Visitor の変更は構文要素の数やモデル変換論理の処理に関わらず容易に行なうことができる．MDA プラットフォームの変更に伴う箇所は具象 Visitor に局所化しているので，MDA プラットフォームの変更柔軟に対応できると判断し，われわれの行った再設計は妥当であったと考える．また，Command パターンを適用してモデル変換論理の処理において横断している処理を部品化する

表 7　Command パターンとの比較

デザインパターン	変更性	解析性	安定性
Observer パターン	=	-	+
Visitor パターン	=	=	=
Visitor パターン +Command パターン	+	+	=

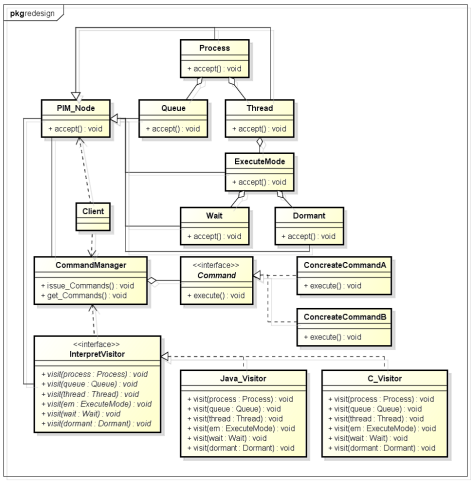

図 2　再設計後の並行処理アスペクトを実現しているアーキテクチャ

ことで，モデル変換論理の処理の再利用性も向上しており，生産性の向上にも貢献できると考える．

4.2　対象とする MDA プラットフォームに関する考察

対象とする MDA プラットフォームについて考察を行う．本研究では，MDA プラットフォームとして C,C++,Java を対象とした．各モデル変換後の構成要素はそれぞれ対応関係を持つ．入力されたアーキテクチャ記述から作成されたモデル変換を行っていくことでプログラムコードが出力される．この際，アーキテクチャ記述とプラットフォームの選択のみでコード生成に必要な情報は十分である．他のプラットフォームを対象とする際でも同様に，入力されたアーキテクチャ記述からの情報をモデル変換を行なっていくことのみで十分にプログラムコードを出力できることが必要であると考える．特定のプラットフォームのプログラムコードを出力する際，付加するべき情報が存在する場合は，プログラムコードの自動生成は困難であると考える．また，対象とするプラットフォームは組込みソフトウェア特有の要求も考慮するべきであり，そのプラットフォームについて実現されたプラットフォームコードが必要であると考える．

5　おわりに

本研究では，MDA プラットフォームの変更に柔軟に対応可能なコード生成ツールのアーキテクチャの再設計

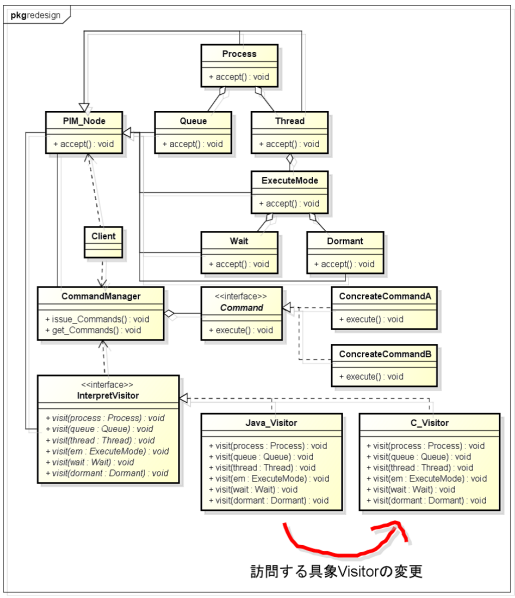

図 3　再設計したアーキテクチャにおける MDA プラットフォームの変更

について考察した．アーキテクチャの再設計には，コード生成ツールに求められる要求から品質特性を特定し，品質特性に適合するようにデザインパターンを適用した．コード生成ツールのアーキテクチャの再設計の妥当性については，再設計したアーキテクチャと再設計の代替案を MDA プラットフォームの変更に柔軟に対応可能かを比較することで確認した．

今後の課題として，再設計したアーキテクチャに基づいたツールの開発と，他の事例を用いた MDA に基づくアーキテクチャへの適応の考察が挙げられる．

参考文献

[1] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns:Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

[2] ISO/IEC, 9126-1:2001: *Software Engineering - Product Quality - Part 1: Quality Model*, 2001.

[3] K. Anneke, W. Jos, and B. Wim, *MDA Explained:The Model Driven Architecture: Practice And Promise*, Addison-Wesley, 2003.

[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，” 知能ソフトウェア工学研究会 (KBSE), vol.108，no.449，pp.55-60，2005.

[5] 長谷川勇雄，小池由和，中村信太，“ アスペクト指向アーキテクチャから Java ソースコードの自動生成に関する研究，” 南山大学　数理情報学部　情報通信学科，2010 年度卒業論文要旨集.