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

2007MI034 長谷川 勇雄　　2007MI108 小池 由和　　2007MI160 中村 信太
指導教員　野呂 昌満　沢田 篤史　蜂巣 吉成

1　はじめに

ハードウェアの高性能化に伴って，組込みソフトウェア開発が大規模化しており，開発工程の短縮化が課題となっている．開発工程を短縮するための方法として，プログラムコードの自動生成が挙げられる．ソフトウェアアーキテクチャに基づく開発では，プログラムコードに一定の規則性が見られる．この性質を利用し定型コードを自動生成することで，実装労力の削減が可能である．プログラムコードを自動生成する手法のひとつ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．

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．E-AoSAS++ では，ソフトウェア開発を支援する環境を整備しており，その一つとして MDA に基づくコード生成ツールがある [2]．先行研究では，Java や C 言語等のプログラミング言語を対象としたコード生成ツールが試作された．試作されたコード生成ツールでは，E-AoSAS++ の振舞いをプログラムコードで実現するための枠組をプラットフォームコードモデルとして設計している．対象とするプラットフォーム毎にプラットフォームコードモデルを実現するライブラリを用意し，図式情報からの情報を用いて実現する．コード生成ツールの特徴は，プラットフォームコードモデルを基に PIM を設計することで，E-AoSAS++ の振舞いを様々なプログラミング言語に対応付けていることである．

組込みシステム開発では，システムやハードウェア毎に様々な制約が考えられる．携帯端末などで用いられる低性能な CPU ではリソース制約，自動車の組込みシステムでは時間制約が考えられる．コード生成ツールが出力したソースコードを，これらの非機能要求を考慮して書き換えることは実装労力の増加に繋がる．

本研究の目的は対象とする組込みシステムに求められる非機能要求に対して，柔軟な対応に可能なコード生成ツールの設計である．

コード生成ツールの開発にプロダクトラインソフトウェアエンジニアリング (以下，PLSE) を適用する．PLSE は，製品系列で共通する部分と変更部分を区別して，核資産として開発・再利用することで，ソフトウェア開発の生産性の向上を実現する開発プロセスである．コード生成ツールを製品系列と考えた場合，共通部分がプラットフォームコードモデルとなり，変更部分が非機能要求を実現する部品・モデル変換論理・定型コードだと考える．本研究では，変更部分である非機能要求の設計にアスペクト指向を用いる．非機能要求をアスペクトとして設計することで，アスペクトコードの織り込みの有無で，非機能要求を考慮したプログラムコードの生成が可能となり，非機能要求に対してコード生成ツールが柔軟に対応できると考える．

非機能要求としてメモリ制約と時間制約を例に挙げて設計を行った．事例検証では，設計したメモリ制約を考慮したアスペクトが，メモリ使用量の削減を行えているのか確認した．設計した結果から，アスペクト指向と PLSE を用いたことについて考察を行う．

本研究の結果を以下に述べる．コード生成ツールがアスペクトコードの織り込むことでメモリ制約と時間制約を考慮したソースコードを出力可能となった．また，PLSE に基づき共通部分と変更部分を整理することで，今後のコード生成ツールへの要求の変化に伴う拡張に対して柔軟に対応可能となった．

2　背景技術

2.1　アスペクト指向

アスペクト指向とは，ソフトウェアに散在する関心事をアスペクトとして適切にモジュール化することで，再利用性，柔軟性の高いソフトウェアの実現を可能とする概念である．アスペクト指向では，複数のオブジェクトに横断する関心事を独立したモジュールとして分離することで，ソフトウェアの開発効率や保守性の向上を可能とする．

2.2　MDA

MDA とは，特定のプラットフォームに依存せず、UML などの標準モデリング技法を使ってシステムの機能をモデル化し、さらにそのモデル情報を基にコードを自動生成する開発スタイルである．MDA に基づくコード自動生成のプロセスでは，二段階のモデル変換を行い，様々なプラットフォームのプログラムコードを出力する．まず，特定のプラットフォームに依存しないモデル (以下，PIM) を定義する．次に，定義した PIM を変換し特定のプラットフォームに依存したモデル (以下，PSM) に変換する．最後に，PSM からプログラムコードを生成する．

3　E-AoSAS++

3.1　概要

E-AoSAS++ は，本研究室で提案され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．組込みシステムは，従来から状態遷移機械の集合として捉えられ設計されている．しかし，それだけでは大規模化・複雑化する組込みシステムの横断的関心事の問題を解決できない．本研究室ではアスペクト指向を用いて横断的関心事の問題を解決する

E-AoSAS++ を提案している．

E-AoSAS++ では，組込み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) の集合で表わす．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 間の通信方法が E-AoSAS++ の動的意味として定義されている．

3.2 E-AoSAS++ に基づくコード生成ツール

E-AoSAS++ に基づく開発支援環境の一つとしてコード生成ツールが提案・試作されている．コード生成ツールは，MDA の概念に基づき，二段階のモデル変換を行いソースコードを生成する．コード生成ツールの概要を図 1 に示す．

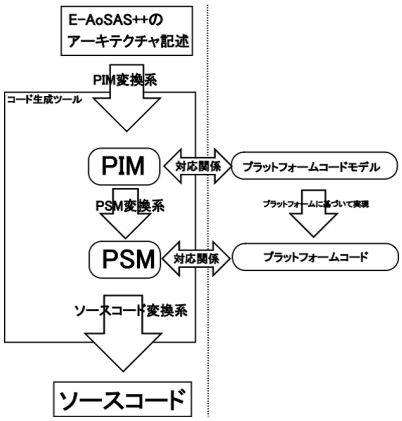

図 1　コード生成ツールの概要

コード生成ツールでは，UML を拡張した E-AoSAS++ のアーキテクチャ記述を入力とし，2 段階のモデル変換を行い様々なプログラミング言語のソースコードを出力する．

3.3 プラットフォームコードモデル

E-AoSAS++ では，プラットフォームコードモデルをアスペクト指向を用いて設計している．図 2 にプラットフォームコードモデルを示す．

CSTM は，並行処理アスペクト・状態遷移アスペクト・アクションアスペクトで構成されており，それらをアスペクト間記述によって繋いでいる．インスタンス処理アスペクトは，インスタンスの情報とそれらの関連を管理するアスペクトである．プラットフォームコードモデルはこれらの関心事をアスペクト指向設計により分離することで，E-AoSAS++ の動的意味の拡張に柔軟に対応できるように設計されている．

4 非機能要求を考慮したコード生成ツールの設計

本研究では，組込みシステムでの非機能要求としてメモリ制約と時間制約を挙げる．メモリ制約と時間制約を考慮したプラットフォームコードモデルの再設計を行う．設計の際には，GoF のデザインパターンを参考にした．プラットフォームコードモデルに適用可能だと考えた理由と動的意味として定義されている E-AoSAS++

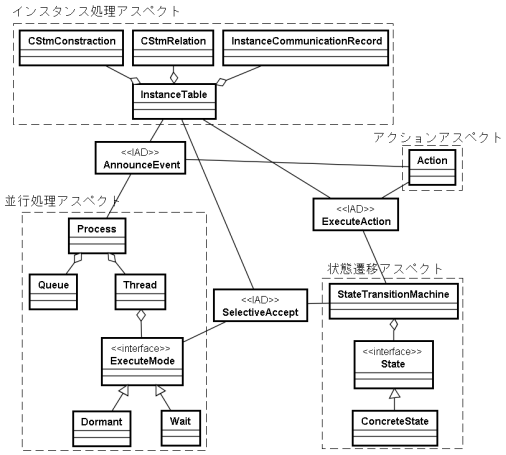

図 2　プラットフォームコードモデル

の振舞いとの整合性が保証できるかを考慮して設計を行った．

4.1 メモリ制約を考慮したプラットフォームコードモデルの再設計

扱えるメモリ使用量に制限がある環境では，実行時のメモリ使用量をどのようにして減らすかが重要となる．Java 仮想マシンの仕様を調べた結果，Thread 数・インスタンス生成数を制限することでメモリ使用量を削減できることがわかった．プラットフォームコードモデルを分析した結果，Thread 数，状態遷移機械の状態のインスタンス生成を制限できるのではないかと考える．

一般的に行われている Thread 数とインスタンス生成を制限する方法であるスレッド・プールとインスタンスの再利用を参考に，本研究で扱う E-AoSAS++ のプラットフォームコードモデルでの適用を考える．スレッド・プールでは，複数のタスクに対して Thread を再利用する．そして，スレッド・プール内の Thread 数を適切に調整し，一定のしきい値を超えた要求は，Thread が処理可能になるまで待たせる．この方法を用いることで，Thread によるメモリ使用量の削減が可能である．インスタンス生成を制限する方法としては，インスタンスの再利用が挙げられる．再利用することで無駄なインスタンス生成を無くし，メモリ使用量をすることができる．Thread 数と状態遷移機械の状態のインスタンス生成の制限を責務とするアスペクト (以下，メモリ管理アスペクト) の設計を行う．

4.1.1 Thread 数の制限

E-AoSAS++ のプラットフォームコードモデルでは，CSTM の並行処理を実現する要素として Thread を各 CSTM に割り当てている．プラットフォームコードモデルにおける Thread は並行処理を実現する方法として Signal-Wait を用いて，BusyWait が起こらないようにしている．イベントを受信した際には，Thread に Signal が送られ動作を開始する．Thread は動作してい

ない CSTM にも割り当てられており，この無駄を無くすことでメモリ使用量の削減が行えるのではないかと考えた．また，動作が開始されていない Thread のインスタンスは再利用できると考える．Thread 数を制限する方針を説明する．スレッド・プールを用いて，Thread の生成数に上限値を設けて管理する．割当て可能な Thread がなく，生成している Thread の数が上限値未満の場合，新たな Thread を生成をする．Thread 数が上限値に達している場合，他の Thread が実行可能状態になるまで待機する．設計指針として，デザインパターンの FlyWeight パターン・Singelton パターンを用いる．Thread のインスタンス生成をメモリ管理アスペクトへの問い合わせに置き換えることで実現する．

プラットフォームコードモデルでは，signal が送られた全ての CSTM に実行権を割り当てられる可能性がある．今回行う Thread 数の制限は，実行可能な Thread を割り当てられる CSTM 数を制限し，実行可能な Thread を割り当てられない CSTM は処理要求順に待機させている．このことから，E-AoSAS++ の振舞いのサブセットであると考える．

4.1.2 状態のインスタンス生成の制限

プラットフォームコードモデルでは状態遷移機械が状態遷移する度に，遷移先の状態のインスタンスを生成している．生成したインスタンスを再利用することで無駄なインスタンス生成を省き，メモリ使用量を削減できると考える．

状態遷移機械の状態のインスタンス生成を制限する方針を説明する．状態のインスタンスを保存するオブジェクトプールを用意する．状態遷移する際にオブジェクトプールに遷移先の状態のインスタンスが保存されているか問い合わせる．保存されていなければ新たにインスタンスを生成し，保存されている場合はそのインスタンスを再利用する．設計指針として，デザインパターンの FlyWeight パターン・Singelton パターンを用いる．状態遷移機械の状態の初期化と状態遷移する際の状態のインスタンス生成の代わりにメモリ管理アスペクトに問い合わせに置き換えることで実現する．

プラットフォームコードモデルでは状態遷移アスペクトを State パターンを用いて実現し，Command パターンを用いて状態遷移に伴うアクションを分離することで状態遷移アスペクトの責務を状態遷移のみにしている．このことから，状態遷移機械の状態のインスタンスが CSTM の複数のインスタンスから共有された場合にも，E-AoSAS++ の振舞いとの整合性が保証できると考える．

4.2 時間制約を考慮したプラットフォームコードモデルの再設計

時間制約のある環境では，問い合わせ回数を少なくすることが重要である．既存のプラットフォームコードを分析した結果，インスタンス処理アスペクトに対する問い合わせ回数を削減することで，時間制約を考慮できるのではないかと考えた．

一般的に行われている問い合わせ回数を削減する方法であるキャッシュを参考に，本研究で扱う E-AoSAS++ のプラットフォームコードモデルでの適用を考える．キャッシュは，使用頻度の高いデータを記憶し，必要となった際にその都度読み出す無駄を省いて処理の高速化を行う．問い合わせ回数の削減を責務とするアスペクト (以下，時間管理アスペクト) の設計を行う．

4.2.1 インスタンス処理アスペクトへの問い合わせ回数の削減

プラットフォームコードモデルでは，インスタンス処理アスペクトがインスタンスの情報とそれらの関連を管理しており，通信する際にはその都度通信先の情報を問い合わせている．通信先の情報をキャッシュしておくことで，インスタンス処理アスペクトへの問い合わせ回数の削減が可能であると考えた．

インスタンス処理アスペクトへの問い合わせ回数の削減の方針を説明する．キャッシュされていないインスタンス処理アスペクトに対する問い合わせ処理の場合，インスタンス処理アスペクトに問い合わせる．問い合わせる際に，通信先の情報をキャッシュする．既にキャッシュされた処理を行う場合は，キャッシュされているデータを参照する．インスタンス処理アスペクトに対する問い合わせの代わりに時間管理アスペクトに対する問い合わせに置き換えることで実現する．

キャッシュを用いたことで重複して行われる処理のみを省略していることから，E-AoSAS++ の振舞いとの整合性は保証できると考える．

図 3 にメモリ管理アスペクト織り込み後のプラットフォームコードモデルを示す．

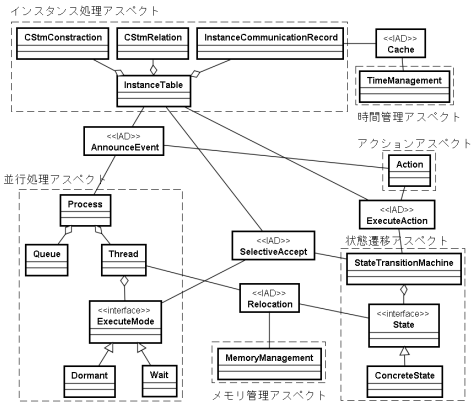

図 3　設計したアスペクトを織り込んだプラットフォームコードモデル

5 事例検証

メモリ制約を考慮して設計したアスペクトを織り込むことで，メモリ使用量が削減できているか事例検証を行う．事例として，Lejos を用いる．

5.1 Lejos

Lejos は，ロボットの挙動を Java で記述できるようにしたプログラミング環境である [1]．マルチスレッド・ガベージコレクションがサポートされており，標準的な Java プログラミングが行える．扱えるメモリ容量が

512KB と小さいことから，Lejos での非機能要求はメモリ制約であると考える．

5.2 メモリ使用量の削減量の測定

メモリ管理アスペクトを用いてメモリ使用量が削減できているのか確認する．Thread 数は，制限することで確実にメモリ使用量の削減が可能であると考える．状態遷移機械の状態のインスタンス制限では，同一の CSTM の型から多くのインスタンスが生成される場合に，有効にメモリ使用量の削減が可能であると考える．以上のことを踏まえて測定を行う．事例には CSTM が三種類の簡単なアーキテクチャを用いる．

Thread の測定は，Thread 生成数の上限値を CSTM インスタンス数が等しい数から順に減らしていき，一つの場合まで測定を行う．測定結果を表 1 に示す．

表 1　測定結果 (Thread)

	織り込み前	織り込み後	
Thread の上限値	3	2	1
メモリ使用量 (Byte)	10488	7188	6976

測定結果から，Thread 数を制限することでメモリ使用量の削減を確認した．

状態のインスタンス生成の制限の測定は，同一の CSTM の型から 25 個のインスタンスを生成する場合と，一つのインスタンスのみ生成される場合を測定する．測定結果を表 2 に示す．

表 2　測定結果 (State)

	織り込み前		織り込み後	
同一型のインスタンス生成数	25	1	25	1
メモリ使用量 (Byte)	57816	10488	57416	10520

測定結果から，同一の CSTM の型から多くのインスタンスが生成される場合に有効であることが確認できた．同一の CSTM の型から一つのインスタンスを生成した際にメモリ使用量が増えているのは，状態遷移機械の状態のインスタンスは型レベルの振舞いを実現するのみであり，インスタンスのメモリ使用量が小さいことが原因であると考える．状態のインスタンスが複数生成されない場合には有効でないことから，コード生成ツールが処理の織り込みを判断する必要があると考える．

6 考察

6.1 アスペクト指向を用いたことに関する考察

非機能要求をアスペクト指向を用いないで設計した場合を考える．アスペクト指向を用いない場合，非機能要求を考慮するためにプラットフォームコードモデルを変更しなければならない．プラットフォームコードモデルが変更されることで，特定の非機能要求のみを考慮した設計となる．コード生成ツールはプラットフォームコードモデルを基に PIM を設計していることから，特定の非機能要求を考慮したソースコードのみ出力可能なコード生成ツールとなる．これでは，コード生成ツールが非機能要求に柔軟に対応できていない．

今回行った設計では，アスペクト指向を用いて非機能要求を実現する部品を設計した．アスペクト指向設計したことで，既存のプラットフォームコードモデルの構成を変更することなく非機能要求に対応可能となった．非機能要求を考慮する必要がある際にはコード生成ツールを用いてアスペクトコードの織り込みを選択することで非機能要求を考慮したソースコードの出力が可能である．以上のことから，非機能要求をアスペクトとして設計したことは妥当であると言える．

6.2 コード生成ツールの PLSE への適用に関する考察

本研究ではコード生成ツールを製品系列と考え，非機能要求であるメモリ制約と時間制約を考慮したソースコードを出力するコード生成ツールの設計を行った．非機能要求を実現する部品を設計し，コード生成ツールのモデル変換論理を変更し，新たな定型コードを作成することでコード生成ツールが非機能要求を考慮したソースコードを出力可能となった．以上の結果から，コード生成ツールのプラットフォームは，既存のコード生成ツールが共通部分・非機能要求を実現する部品・定型コードが変更部分であると考える．コード生成ツールは，アーキテクチャ記述からプログラムコードを生成しており，これはどのデバイスを対象とする場合にも共通である．アスペクト指向を用いたことで，必要な箇所のみ非機能要求を考慮した処理を織り込むことで可能となった．

PLSE に基づきコード生成ツールのプラットフォームの整理を行うことで，共通部分・変更部分が明確になった．本研究は，PLSE の三つのプロセスの内のドメインエンジニアリングであると考える．組込みシステムではデバイス毎に非機能要求が異なるが，PLSE に基づき整理されたプラットフォームを再利用することで設計可能である．PLSE を用いたことで，今後のコード生成ツールへの要求の変更に柔軟に対応可能となった．以上のことから，コード生成ツールに PLSE を適用したことは妥当であると言える．

7 おわりに

本研究では，コード生成ツールを製品系列と考え，非機能要求に対して柔軟に対応可能なコード生成ツールの設計を行った．コード生成ツールに PLSE を適用し，変更部分の設計にアスペクト指向を用いたことについて考察を行った．結果として，メモリ制約と時間制約を考慮したソースコードをコード生成ツールが出力可能となった．また，コード生成ツールの共通部分と変更部分の整理を行うことで，今後新たな非機能要求を考慮する際にコード生成ツールが柔軟に対応可能となった．

今後の課題として，リアルタイムシステムで考えられるメモリ制約と時間制約を同時に実現可能なアスペクトの設計が挙げられる．

参考文献

[1] *Lejos*，*“Java for LEGO MindStorms，”* http://lejos.sourceforge.net

[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，” 情報処理学会研究報告，vol2009，no13，pp.121-128.