非機能特性の特定によるアーキテクチャの構造決定に関する研究

M2011MM015 平崎唯善
指導教員：野呂昌満

1　はじめに

アスペクト指向技術によって，従来のソフトウェア開発技術では難しかった横断的関心事の問題に対処することが可能となった [5]．横断的関心事とは，システムをモジュール分割した際に，複数のモジュールに散在する関心事である．横断的関心事は分散問題やもつれ問題 [5] を引き起こすので，ソフトウェアに対する理解や保守を困難にする要因となっている [1]．横断的関心事の問題に対処するために，アスペクトを明示的に捉えたアーキテクチャを構築する必要がある．アスペクト指向技術を用いて，横断的関心事をアスペクトとしてモジュール化することで，ソフトウェアの複雑さの軽減が見込める．この利点を活かすために，横断的関心事を特定してモジュール化することが求められる．

非機能特性は横断的関心事になりうる [1]．横断的関心事は保守性や再利用性の低下の原因となるので，非機能特性をアスペクトと認識した上でアーキテクチャを構築しなければならない．仕様モデルに記述された製品の特徴や機能をもとにアーキテクチャは構築されるが，アスペクトを考慮したアーキテクチャと仕様モデルに記述される特性との関連は明確にされていない．これにより，仕様モデルをもとにした系統的なアーキテクチャ構築が困難になっている．

本研究の目的は，仕様モデルに記述される特性とアーキテクチャのアスペクトとの関連を明確化し，アスペクト指向アーキテクチャの構築を支援することである．仕様モデルに記述された特性を満たすための構造を，GoF デザインパターン [2] をもとに整理して類型化する．本研究では非機能特性を対象にアーキテクチャのアスペクトとの対応付けをおこなう．関連を明確にすることで仕様モデルとアーキテクチャの一貫性を保つことができ，アスペクト指向アーキテクチャの構築支援に繋がると考える．

本研究では，非機能特性を型付きのフィーチャとして定義し，非機能特性を満たすための構造と対応付けた．アーキテクチャは機能特性と非機能特性に対応するアスペクトの集合およびアスペクト間記述で構成されると仮定する．この仮定をもとに非機能特性とアスペクトの関連を整理した．仕様モデルには Cardinality-Based Feature Model(以下，CBFM)[4] を採用し，型付き要素を表現するためにメタモデルを拡張した．アスペクトは GoF デザインパターンをもとに抽象化し，仕様モデルの型付き要素と対応するものとして位置付けた．型付きフィーチャに対応するアスペクトをアーキテクチャに配置することによって，非機能特性を満たすためのアーキテクチャを構築できる．本研究では組込みシステムを対象に研究を進めた．プリンタシステムを事例に仕様モデルを作成し，提案手法を用いてアーキテクチャを構築した．この結果をもとに，型対応以外の手法と比較して考察する．

2　アーキテクチャ構築における課題

ソフトウェア開発において，アーキテクチャは重要な役割を果たす．アーキテクチャはソフトウェアの基本的な構造を示すが，ソフトウェアの変更性や再利用性と密接に関わっている．この節ではアーキテクチャ構築における問題点と課題について述べ，課題解決に向けたアプローチを提案する．

2.1　アーキテクチャ構築における問題点

保守性や再利用性が高いソフトウェアを開発するために，アーキテクチャ構築の段階で横断的関心事をモジュール化することが求められる．横断的関心事はソフトウェアの保守性や再利用性を低下させる原因となる．横断的関心事をモジュール化することで関心事ごとの変更や修正が可能となり，保守にかかる労力の削減が見込める．横断的関心事のモジュール化のための技術としてアスペクト指向技術がある．アスペクト指向技術は支配的分割に対して横断する関心事をアスペクトとして分離し，モジュール化するための考え方や技術を提供する．アーキテクチャ構築の際にアスペクト指向技術を適用し，横断的関心事を明示的に捉えたアーキテクチャを構築する必要がある．

ソフトウェアの品質を保証するための非機能特性は支配的分割に対して横断しうるので，アスペクトとしてモジュール化する必要があるが，非機能特性に対応するアスペクトの構造は明確にされていない．これにより，仕様モデルをもとにした，非機能特性を保証可能なアーキテクチャの系統的な構築が困難になっている．このためにアーキテクチャの構築には労力がかかり，また，構築したアーキテクチャの再利用性や保守性が低くくなる可能性がある．

2.2　問題解決に向けての課題とアプローチ

2.1 節で提示した問題は，非機能特性と対応するアスペクトを関連付け，仕様モデルから直接的にアーキテクチャを構築可能とすることで解決できる．このための技術的課題として，非機能特性に対応するアスペクトの定義方法と，非機能特性とアスペクトとの対応関係が与えられたときに仕様モデルからアーキテクチャを構築する方法の 2 点が挙げられる．

非機能特性に対応するアスペクトの定義には GoF デザインパターンを利用する．各パターンはオブジェクト指向設計における特定の問題や課題に焦点を絞った解法であり，非機能特性を満たすための構造と関連があると考えた．デザインパターンをもとに抽象化し，パターン化することで，非機能特性に対応するアスペクトを定義する．

仕様モデルをもとにしたアーキテクチャの構築には型対応を用いる．型対応とは，仕様モデルに記述される非

機能特性を型付きのフィーチャとして定義し，非機能特性に対応するアスペクトと関連付けることであるとする．本研究では仕様モデルとしてフィーチャモデルを採用し，非機能特性を型付きのフィーチャ(以下，型付きフィーチャ) として表現する．これにより，仕様モデルからの直接的なアーキテクチャ構築を実現する．

本研究では，アーキテクチャは機能特性と非機能特性に対応するアスペクトおよびアスペクト間記述で構成されると仮定する．機能特性に対応するアスペクトは，システムの主要な機能を実現するためのコンポーネント群である．非機能特性に対応するアスペクトは，障害許容性や効率性といったシステムの品質や満たすべき特性を実現するためのコンポーネント群である．アスペクト間記述はアスペクト間の織込みの規則を記述したコンポーネントである．これらの組み合わせでアーキテクチャは構成される．

3　型付きフィーチャを用いた対応付け

本研究では型付きフィーチャを用いてアーキテクチャを構築する手法を提案する．型付きフィーチャを表現するためには，仕様モデルの表現を拡張する必要がある．拡張した表現を用いてアスペクトとの対応を定義する．これらを用いて，仕様モデルからアーキテクチャを構築する．

3.1　フィーチャモデルのメタモデルの拡張

本研究で対象とする仕様モデルはフィーチャモデルである．フィーチャは製品の機能特性や非機能特性を表現することができるので，仕様モデルとして適していると考える．フィーチャモデルについて調査をおこない，Czarnecki らが提案した CBFM[4] を仕様モデルとして採用した．FORM に基づくフィーチャモデル [3] では層の記述を追加することでフィーチャの役割を明示的に示すことができるが，仕様モデルの段階で実現技術は現れないと考えたことから CBFM を選択した．仕様モデルとアーキテクチャのアスペクトとの型対応を実現するために CBFM のメタモデルを拡張し，型の表現を追加する．拡張した CBFM のメタモデルを図 1 に示す．

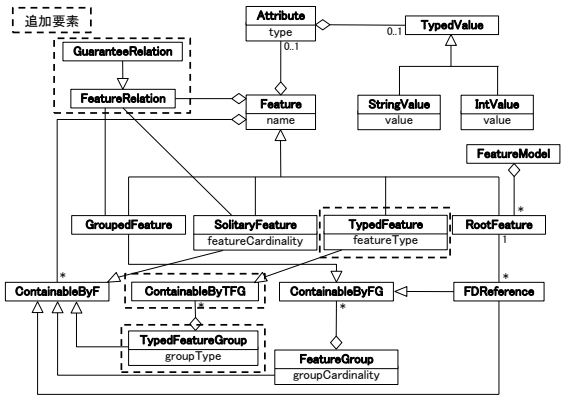

図 1　拡張した CBFM のメタモデル

追加要素のうち，TypedFeature は型付きフィーチャを表現する．TypedFeatureGroup は TypedFeature の集合である．ContainableByTFG は型付きフィーチャと関連を持つフィーチャの種類を表している．FeatureRelation は CBFM では定義されていないフィーチャ間の関係の種類を表現するために追加した．GuaranteeRelation は，どの機能特性や部品を対象として非機能特性を保証するかを表現するために追加した関係である．保証関係はアスペクトの織込み先を明示するために必要となる．これらの表現を用いてフィーチャモデルを作成する．

3.2　型付きフィーチャとアスペクトの関係

非機能特性をアスペクトとして捉え，仕様モデルからアーキテクチャを構築するために型付きフィーチャを定義した．型付きフィーチャとして表現される非機能特性は，非機能特性に対応するアスペクトと対応する．図 2 に型付きフィーチャと非機能特性に対応するアスペクトの関係を示す．

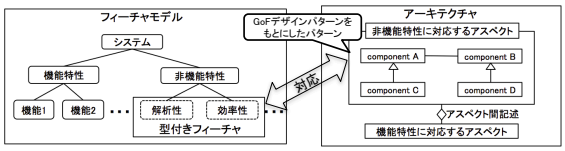

図 2　型付きフィーチャと非機能特性に対応するアスペクトの関係

非機能特性に対応するアスペクトは，非機能特性を満たすための構造を GoF デザインパターンをもとに定義したパターンで表現される．各非機能特性に適するパターンを特定し，その構造をもとに実現技術に依存しない形に抽象化する．このパターンを用いることで，実現技術に依存することなく非機能特性を満たすための構造を決定することができる．型付きフィーチャと他のフィーチャの関連として，どの機能特性を対象として非機能特性を保証するかを明確にするために，保証関係を記述することができる．機能と非機能特性の関連を明示的に表現することで，アスペクトの織込み先を明示的に示すことができる．

4　解析性に対応するアスペクト

対応付けの枠組みにもとづき，前章で示した非機能特性とアスペクトとの対応関係を整理し，類型化する．組込みシステムでは故障原因を特定できることが重要であるので，解析性に対応するアスペクトをとり上げて整理する．他の非機能特性についても同様にアスペクトとの対応付けが可能である．

4.1　解析性に対応するアスペクトの定義

解析性に対応するアスペクトは，Observer パターンをもとに抽象化した．図 3 に解析性に対応するアスペクトを示す．Observer パターンはあるオブジェクトの状態が変化した際に，そのオブジェクトに依存するオブジェクトに変化を通知するためのパターンである．解析性を満たす手法として，監視対象の状態の変化に応じたログ取得や監視モニタへの問題通知といった方法が考えられる．

これらの実装はログ取得や監視の対象となるコンポーネントに依存する．さらに，変化に応じたログ取得などは各コンポーネントに横断する可能性があるので，アスペクトとして分離する必要がある．Observer パターンをもとに抽象化することで，監視対象のコンポーネントとの依存を小さくすることができ，アスペクトとして分離することができると考える．

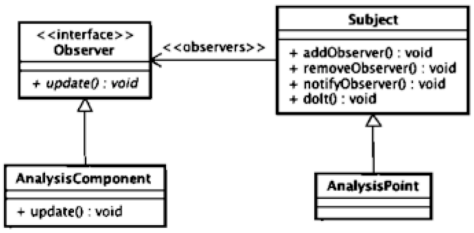

図 3 解析性に対応するアスペクト

AnalysisComponent と AnalysisPoint は抽象化されたコンポーネントである．AnalysisComponent にはデータロガーや監視用モニタがあてはまる．AnalysisComponent は実現手段によって変化する．解析性を満たすための手法は複数考えられるが，AnalysisComponent を変更することで各手法による解析性の実現が可能となる．AnalysisPoint は非機能特性を保証する対象を表現するコンポーネントである．監視やログ取得をおこなうコンポーネントがあてはまる．

4.2 解析性に対応するアスペクトのアーキテクチャ記述

アーキテクチャ記述として，E-AoSAS++(Aspect-Oriented Software Architecture Style for Embedded system)[6] の記述方法に従ったコンポーネント図を用いる．アスペクト指向アーキテクチャのための記述方法が E-AoSAS++では確立されているので，この記法を用いた．E-AoSAS++の記述に従った解析性に対応するアスペクトの記述を図 4 に示す．図 4 は図 3 をもとに記述したコンポーネント図である．この記法を用いることで，アスペクトごとにその構成を明示的に表現することができる．

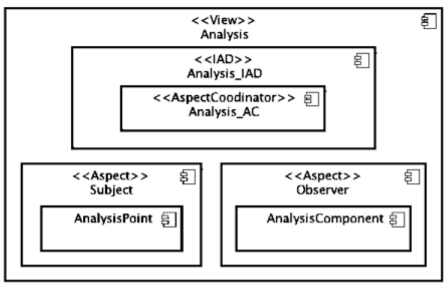

図 4 解析性に対応するアスペクトのアーキテクチャ記述

5 事例検証

本研究では事例としてプリンタシステムを用いる．満たすべき機能特性は，印刷，コピー，印刷の取消し，スキャン，進捗とエラーの表示である．プリンタシステムには故障原因の特定と故障の回避が必要と考え，解析性，追跡性，障害許容性が要求される可能性があると考えた．このプリンタシステムのフィーチャモデルを作成し，アーキテクチャを構築する．

5.1 プリンタシステムのフィーチャモデル

プリンタシステムの仕様をもとに，プリンタシステムのフィーチャモデルを作成した．図 5 に作成したフィーチャモデルを示す．

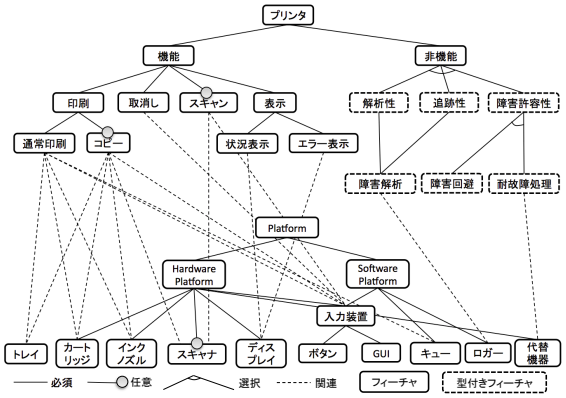

図 5 プリンタシステムのフィーチャモデル

機能特性のうちスキャン機能とコピー機能はスキャナの有無によって変化するので，任意のフィーチャとして定義した．型付きフィーチャは点線の四角で表現している．解析性，追跡性，障害許容性は型付きフィーチャのグループであり，障害解析，障害回避，耐故障処理は型付きフィーチャである．図 6 は非機能特性の保証関係を記述している．このフィーチャモデルをもとに，型対応によって非機能特性に対応するアスペクトを分離したアーキテクチャを構築する．

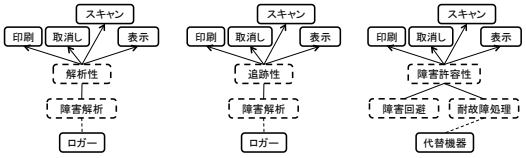

図 6 非機能特性の保証対象の記述

5.2 プリンタシステムのアーキテクチャ記述

前節のフィーチャモデルをもとに，型対応によってアーキテクチャを構築した．構築したアーキテクチャの一部を抜粋したものを図 7 に示す．解析性に対応するアスペクトは，4.2 節で示したアーキテクチャ記述と仕様モデル上のソフトウェアプラットフォームの記述を考慮して作成した．支配的分割はオブジェクト指向にもとづいた分割であり，それに対して解析性が横断していることを示している．Subject には保証する機能に関連するコンポーネントを，Observer には解析性を満たすために用いるコンポーネントを配置している．

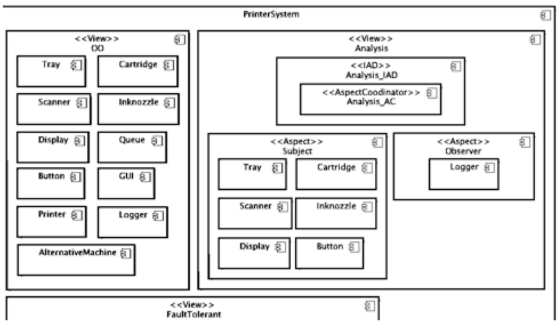

図 7 プリンタシステムのアーキテクチャ記述 (一部抜粋)

6 考察

6.1 型対応を用いた対応付けに関する考察

本研究では，型付きフィーチャを用いてアスペクトとの関連付けをおこなった．型対応以外の方法としてタグを用いた対応付けと比較し，妥当性について議論する．

型対応では，フィーチャモデル上で非機能特性の下層に配置されるフィーチャも同じ型に属することができる．解析性の下層に配置される障害解析は，解析性に対応するアスペクトの構造にロギングのような実現技術を加味することでコンポーネントを構築することができる．解析性を満たす手法には障害解析だけでなく障害監視も考えられる．抽象化した構造との型対応により，実現技術に依存することなくパターン化することができた．一方で，タグ対応が別の手段として挙げられる．タグ対応を用いた対応付けでは，タグの示す内容に特化した関連付けが必要となる．型対応に比べて抽象度が低く，限定的と考える．さらに，非機能特性とその下層に配置されるフィーチャごとにタグを付与しなければならない可能がある．上記より，型対応による対応付けが妥当であった．

6.2 Observer パターンの利用に関する考察

Observer パターンをもとに解析性を満たすためのアスペクトを定義した．State パターンと Command パターンを用いた場合と比較し，Observer パターンの利用の妥当性について議論する．

State パターンは状態が変化した際にオブジェクトの振舞いを切り替えることを可能にするパターンである．解析性を満たすためには，エラーの発生や処理の開始と終了のような状態が遷移する場合に状態に応じたログ取得や監視モニタへの通知をおこなう必要がある．State パターンを用いることで状態に応じたログ取得といった方法を実現することができる．しかしながら，State パターンを用いた場合では，支配的分割にもとづいて分割されたモジュールに解析性に関する振舞いが含まれるので，横断的関心事として分離できていないといえる．Observer パターンを用いた場合では，解析性に関する振舞いは AnalysisComponent 内に収まる．

Command パターンはあるオブジェクトに対する要求自体をオブジェクトとして扱い，カプセル化するためのパターンである．Command パターンを用いることで解析性を満たすための処理ごとにカプセル化することが可能になり，処理の組み合わせが可能になる．しかしながら，Command パターンを用いた場合では，状態遷移が起きるたびに Command を実行する必要がある．支配的分割にもとづいて分割されたモジュールにこの処理を記述しなければならないので，横断的関心事として分離できていないといえる．

前述の比較から Observer パターンが適していると判断した．解析性を満たすためのパターンはこの他にも考えられるので，他のパターンとも比較する必要がある．また，パターンを組み合わせて実現することも考えられる．これらの比較をおこない，Observer パターンの利用が妥当であったことを確認しなければならない．

7 おわりに

本研究では，仕様モデルをもとにした系統的なアーキテクチャ構築を支援するために，非機能特性を型付きフィーチャとして扱い，GoF デザインパターンをもとに一般化したアスペクトと対応付けた．仕様モデルとして CBFM を採用し，型の表現を追加するためにメタモデルの拡張をおこなった．型付きフィーチャに対応するアスペクトを配置することでアーキテクチャを構築した．解析性に対応するアスペクトを示し，事例検証の結果からアプローチの妥当性について議論した．今後の課題として，他の非機能特性に関しても同様に型対応を考え，カタログ化することが挙げられる．

参考文献

[1] A. Rashid, P. Sawyer, A. Moreira and J. Araujo, “Early Aspects: a Model for Aspect-Oriented Requirements Engineering,” *Proceedings of the IEEE Joint International Conference on Requirements Engineering (RE’ 02)*, 2002.

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

[3] K. C. Kang, S. Kim, J. Lee, K. Kim, G. J. Kim and E. Shin,“FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures,” *Annals of Software Engineering*, vol. 5, no. 1, pp. 143-168, 1998.

[4] K. Czarnecki, S. Helsen and U. Eisenecker, “Staged Configuration Using Feature Models,” *Proceedings of Software Product Lines: Third International Conference(SPLC 2004)*, vol. 3154, pp. 266-283, 2004.

[5] L. Rosenhainer, “Identifying Crosscutting Concerns in Requirements Specifications,” *Proceedings of OOPSLA Early Aspects 2004*, 2004.

[6] M. Noro, A. Sawada, Y. Hachisu, M. Banno, “E-AoSAS++ and its Software Development Environment,” *Proceedings of the 14th Asia-Pacific Software Engineering Conference(APSEC2007)*, pp. 206-213, 2007.