ソフトウェア工学 131－8
（2001．6．1）

ソフトウェアアーキテクチャのためのアスペクト指向分析

岸 知二

JAIST/NEC

ソフトウェアアーキテクチャは、ソフトウェアの進化を通じて、あるいは製品系列の展開を通じて安定したものであることが望まれる。そうしたソフトウェアアーキテクチャの設計においては、その上に構築される潜在的なソフトウェアの共通性や差異を捉え、それをソフトウェア構造に反映させることが必要となる。それに際しては、機能面からだけでなく、品質特性面からの共通性や差異を捉えることが重要である。機能が同様であっても、要求される品質特性が異なればソフトウェアアーキテクチャは異なるものになるからである。本稿では、こうしたアーキテクチャ設計のために、品質特性を含めた共通性や差異を分析するためのアスペクト指向分析を提案する。

Aspect-Oriented Analysis for Software Architecture

Tomoji Kishi

JAIST/NEC

It is expected that software architecture remains steady throughout its evolution or development of product-lines. In order to design software architecture to have the characteristics, it is important to capture the commonality and differences between potential software that will be developed on the software architecture, and design the software architecture to accommodate the commonality and differences. We have to capture the commonality and differences not only from functional aspect, but also from aspects relate to quality attributes, such as performance and reliability, because requirements on quality attributes also have strong impact on software architecture. In this paper, we propose aspect-oriented analysis method, in which we can capture commonality and differences between software, considering requirements on quality attributes.

1 はじめに

ソフトウェアアーキテクチャ(以下アーキテクチャ)[1][8]は、ソフトウェアが修正、拡張されていく進化の過程を通じて、その基盤としての役割を果たすことが求められる。あるいは製品系列(プロダクトライン)を開発する際には、製品群の共通の基盤となるアーキテクチャを設定することが望まれる。このためアーキテクチャの設計においては、そのアーキテクチャの上に時間的、空間的に展開されうるソフトウェア群を想定し、共通性と差異とを捉え、共通の基盤となる基本的な構造を見いだすことが重要となる。

こうしたアーキテクチャ設計においては、機能に関してだけではなく、品質特性に関する共通性や差異の理解も必要となる。同じ機能を実現するにしても、求められる性能のレベルが全く異なれば、アーキテクチャは異なったものとなり得る。またソフトウェアが作られる基盤の理解も必要となる。利用するOSやミドルウェアが異なれば、アーキテクチャは異なったものになり得るからである。

我々は、ソフトウェアの進化に耐えるアーキテクチャや、製品系列のアーキテクチャを設計するために、アスペクト指向分析を検討している。この手法は、想定される製品群の要求事項をアスペクトに分離して分析することにより、品質特性に関してどのようなレベルの要求が存在するかを分析することをねらっている。アーキテクチャ設計にアスペクト指向分析を適用することで、想定されるソフトウェア群が持つ共通性や差異を、品質特性を含めて理解することができ、より適切なアーキテクチャ設計を行うことができる。また、基盤の変化に対する考慮についても検討する。なお本手法は[4]での提案をさらに拡張、リファインしたものとなっている。

本稿では、アスペクト指向分析とそれに基づくアーキテクチャ設計について述べる。2章では本研究の目的について述べる。3章ではアーキテクチャやアーキテクチャ設計に関する基本概念、本研究の立場について述べる。4章ではアスペクト指向分析の手法について提案する。5章では例題を用いて、本手法について具体的に説明する。6章では関連する議論を行う。

2 目的

本研究の目的は、進化に耐えられるアーキテクチャの設計方法を検討することである。特に本稿では、そのアーキテクチャ上に潜在的に作られるソフトウェア群の持つ共通性と差異とを踏まえた設計の方向づけを行うための手法について提案する。

ここで進化に耐えられるとは、ソフトウェアの修正や拡張を通じてその基盤としての役割を果たすことができることを意味する。プロダクトラインの開発における共通基盤としてのアーキテクチャも、製品の時間的、空間的な展開に対応できることが求められており、同様の特性が望まれる。

オブジェクト指向手法などでは、機能や情報は明示的にモデル化されるため、その共通性や差異は相対的に捉えやすい。一方、品質特性に関する要求は捉えづらく、分析が困難である。本稿ではこうした品質特性に関する共通性や差異の分析に特に焦点を当てる。

進化の過程において、想定した機能のバリエーションや、品質特性のレベルから逸脱する要求がくると、そのアーキテクチャを基盤とすることは困難となる。従って、設計の方向づけにおいて重要なことは、詳細な共通性や差異の解析ではなく、進化の過程を通じてソフトウェアに対してどのようなカテゴリの要求が想定されるかを判断し、それに基づいたアーキテクチャの設計方針をたてることである。本研究ではこうした方向づけまでを対象とし、具体的な構造の設計方法については言及しない。

3 基本概念

本章では、本研究に関る基本的な概念や、本研究の立場について述べる。

3.1 アーキテクチャとアーキテクチャ設計

アーキテクチャは、ソフトウェアの**基盤**が提供する要素の**構造**である。基盤はソフトウェアが関る様々な局面ごとに存在する。例えばOSやミドルウェアはソフトウェアの実行時の基盤であり、それらが提供するタスクやプロセスの協調関係などは実行構造である。あるいはファイルシステムやプログラム言

語は開発時の基盤であり、プログラム構造やファイル構成などは開発構造である。

こうした基盤上の構造は、その関る局面に対する**要求**を満たすように設計される。すなわち実行構造は性能などの実行時の品質特性に関する要求を満たすように設計されるし、開発構造は拡張性などの開発時の品質特性に関する要求を満たすように設計される[7]。

我々はこのフレームワークに対し、オブジェクト指向での開発モデルを図1に示すようにあてはめて適用することを想定している。**要求モデル**はユースケースで表され、機能のカテゴリとアクタとを示す。**論理モデル**は求められる機能の（実現方法を踏まえない意味的な）モデルであり、**物理モデル**はその実現方法を示すモデルである。物理モデルは想定する基盤に依存している。論理モデル、物理モデルはコラボレーションモデル(specification level)を利用して表現する。**基盤モデル**はクラス図等で示すことができるが、本稿では特段の規定はしない。

要求モデル中のユースケースは、論理モデル中の**機能**(コラボレーション)群に対応する。論理モデル中の機能は物理モデル中の**実現構造**(コラボレーション)によって実現される。物理モデルは基盤中の要素の構造を示し、その要素はステレオタイプで示される。なお**サービス**とはユースケースと対応づけることのできる機能である。

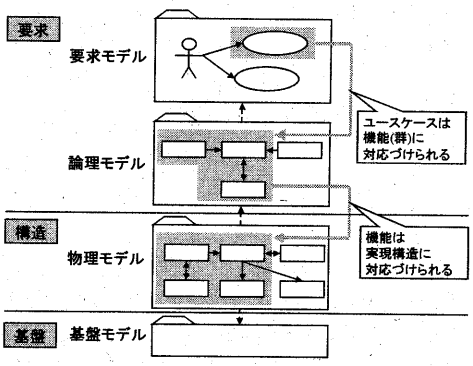

図1：アーキテクチャを捉えるモデル

アーキテクチャ設計とは、機能(論理モデル中のコラボレーション)を実現するための、実現構造(物理モデル中のコラボレーション)を決定する作業である。通常は、実現構造の候補群の中から、要求に対して最も適切な構造を選択することになる。

3.2 品質特性に関する要求

品質特性には様々なものが存在するが、本稿ではサービス(群)と対応づけられる品質特性に限定する。例えば応答性はサービス(群)に対して定義され、信頼性はサービス群の達成レベルとして定義される。

品質特性に対する要求は要求モデルと論理モデルの二つの段階で定義することが有用である。要求モデルの段階では、例えばアクタが人でありインタラクティブに使うならば応答時間は2、3秒以内にしたいとか、このユースケースは非常時に活性化されるので信頼性への要求が高いというように、アクタ、ユースケース、それらのおかれる状況、利用局面などに照らして、品質特性に関る要求の基本的な考え方を定義することができる。一方論理モデルでは、その枠組みを踏まえながら、個別のサービスに対する要求を定義することができる。

要求モデルを活用してどのようなカテゴリの要求が存在するかを捉えることは初期のアーキテクチャ設計において有効であり、またそれを踏まえて個々の要求を定義することによって、品質特性に関する要求の一貫性を保つことができる。

3.3 進化

ソフトウェアが修正、拡張され、あるいは製品展開されることは、要求の変化、もしくは基盤の変化（あるいは他の基盤への展開）と捉えることができる。

要求の変化は、機能要求の変化と、品質要求の変化とに分けられる。これらはモデル上では、要求モデルや論理モデル（あるいはそれに付随して定義される要求記述）に対する変化として表現される。機能要求の変化はユースケースやコラボレーションの変化として捉えられるし、品質要求の変化は、ユースケースやサービスに対して与えられる要求記述の変化として捉えられる。一方基盤の変化は、設計モデルが依存している基盤の変化として捉えられる。

アーキテクチャは、要求を満たすための基盤中の要素による実現構造であるから、進化に対して安定した構造を見いだすためには、想定される要求や基盤の変化パターンを捉え、それらを受け止めることのできる構造を検討しなければならない。

なおこうした要求や基盤の変化には、順次製品をバージョンアップするような時間的な変化と、日本市場向け製品と米国市場向け製品というような空間的な変化とが存在する。時間的な変化は、順序関係という制約が加わるが、その両者の検討は、多くの共通性を含んでおり、本稿ではその両者を視野にいれて議論する。

4 アスペクト指向分析

本章ではアスペクト指向分析の手法について説明する。

4.1 基本的な考え方

我々が品質特性を考える際には、その機能のカテゴリや、品質特性に関る尺度（以下これらを**要因**と呼ぶ）に注目していると考えられる。例えばデータ検索の応答時間を考える際には、経験的に、検索パターン(ランダムかシーケンシャルか等)、検索空間の大きさ、検索するデータサイズ、それが固定長か可変長かなどに注目し、それが達成されるかどうかを考える。一方検索されるデータの意味などは応答時間とは無関係なので考慮しない。

アスペクト指向分析では、品質特性毎にそれを検討する際に経験的に有用と考えられる要因を利用して機能を特性づける。特性づけられた機能を、注目する品質特性に関る要因だけに注目して整理することで、いくつかのカテゴリに分類することができる。そのカテゴリ毎にアーキテクチャに対する影響を分析することで、アーキテクチャ設計に影響を持つ要求を特定しようとするものである。

4.2 分析の手順

分析のステップを以下に示す。想定される複数のソフトウェア群が存在し、それに対する機能面、品質特性面からの要求、またそれぞれのソフトウェア群の基盤に関する情報に基づいて分析を行う。(1)から(4)までのステップは、分析対象となるソフトウェア群中の各ソフトウェアに対して行い、(5)はその結果を総合して行う。

(1)機能の特性づけ：分析対象となるソフトウェアの各機能に対して、要因で特性づけを行う。要因は検討対象とする品質特性に応じて設定される。品質特性は、性能や信頼性といった大項目の品質特性では粗すぎるため、データアクセスの応答時間、通信のスループットなど、より詳細なブレークダウンが必要である。複数の品質特性の分析が必要ならば、それぞれに関る要因を用いて特性づけする。なおひとつの要因が複数の品質特性に関ることもある。なお現実のシステムにおいては機能の数が多いため、要求モデルなどに基づき、類似の機能から代表的なものを選んで対象を設定するなどの操作が必要である。

(2)要求の分離：個々の品質特性毎に、その品質特性に関る要因と要求だけを取り出す。関らない要因は無視する。その操作によって、各要因の値がすべて同じオーダーである列ができた場合には、ひとつの列に縮退させる。各列は、その品質特性に関る要求のひとつのカテゴリを示し、列の数が要求カテゴリの数を示す(図２)。

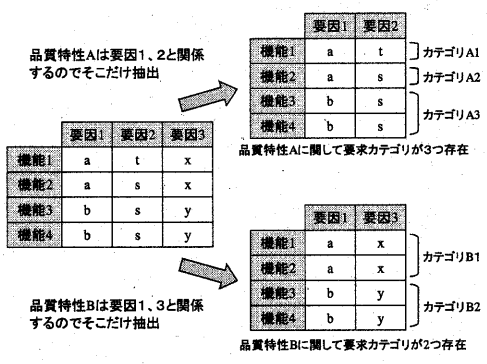

図２：要求の分離と縮退

(3)インパクトの分析：(2)の結果に基づいて、アーキテクチャへ影響を持つと思われる要求カテゴリがあるかどうか検討する。基本的には要求レベルが高く、かつ要因から見てその達成がクリティカルと判断される部分に対して何らかのアーキテクチャ上の工夫が必要という判断を行う。特段の工夫が不要であれば、論理構造を踏まえた実現構造にする。なお、

この分析には利用する基盤やその上で達成できる品質特性のオーダーなどに対する知識が必要とされる。

(4)候補の検討：アーキテクチャへの影響があると特定された部分に対して、どのような実現構造の候補があるか検討する。もしも複数の候補があり得る際には、求められる要件に最も適すると考えられるものを選択する。なお評価基準が複数あり特定の候補の選択が困難な状況では AHP 法を適用するなどの方法も考えられる[5]。また選択された実現方法が、基盤にどれほど強く依存しているかどうかについても検討する。

(5)共通性と差異の理解： 想定されるソフトウェア群に対して(1)から(4)までのステップが終了したら、それらを総合して、1)機能、2)品質特性の要求カテゴリ、3)適用される実現方式、4)基盤それぞれの共通性と差異を理解する。

4.3 設計の方向づけ

分析結果に基づいた設計の方向づけの考え方を示す。

進化に対して安定した構造を実現するために、分析から必要と判断された構造を、図3に示すリファレンスに照らして整理する。

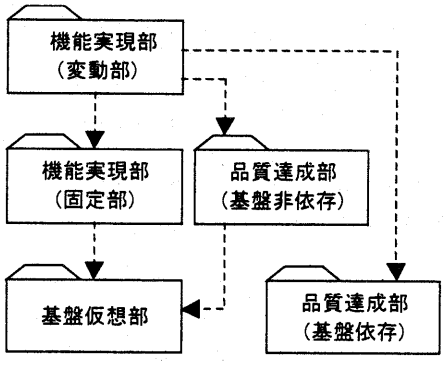

図3：アーキテクチャのリファレンス

このリファレンスは実際のアーキテクチャを設計する際に、どのような主旨で構成すればよいかというメタな意図を示すものである。機能の共通性、差分のみに注目した場合には、機能実現部(変動部)、機能実現部(固定部)、基盤仮想部だけで構成することが可能であるが、品質特性の共通部、差分を考慮すると、その変動を吸収するために品質達成部が必要となる。また基盤の変動を考慮して、品質達成部は基盤依存のものと、基盤非依存のものとを区別することが望ましい。なお機能面からは変動がなくても、その機能にかかわる品質特性の要求が変化する場合には、その機能実現は固定部ではなく変動部に含める必要がある。

なおこのリファレンスはメタな意図を表現したものであるから現実の構造を規定するものでない。このままのレイヤ構造を作ってもよいし、本主旨を保存しながら他のスタイルで実装してもよい。それは実設計の段階で具体的な特性上の得失を踏まえて検討しなければならない。

5 例題

本章では例題を用いて本手法について説明を行う。

5.1 地図情報検索システム

本章では、CD-ROM 中に格納された地図情報を検索するシステムの製品系列開発を取り上げる。図4は地図情報のクラス図である。なお area は map あたり 100、landmark は area あたり 1000 あり、description 属性のみサイズが大きく不定長であり、他の属性はサイズが小さく固定長とする。

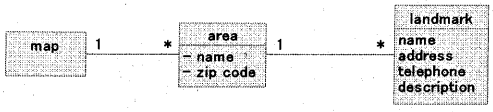

図4：地図情報の構造

ここでは3つの製品 P1、P2、P3 を考え、表1のような機能要求(検索の入力と出力)と応答時間の要求があるとする。P3 は P2 と同様の要求であるが、利用する CD-ROM の物理フォーマットが P1 や P2 とは異なるとする。物理フォーマットの違いは、大量のシーケンシャルアクセスの時間に影響を及ぼすものとする。

表1：機能と応答時間への要求

P1 への要求

	入力	出力	応答時間
S1	tel	name	< 2-3 sec

S2	name	tel	< 2-3 sec
S3	zip	name	< 2-3 sec

P2、P3 への要求

	入力	出力	応答時間
S1	tel	name	< 1 sec
S2	name	tel	< 1 sec
S3	zip	name	< 1 sec
S4	string	name	< 5-6 sec

なお S4 は description の全文検索である。

5.2 アスペクト指向分析

以下、手順に沿ってアスペクト指向分析を行う。

(5) **機能の特性付け**：応答時間を検討するための要因として、検索パターン(R:ランダム、S:シーケンシャル)、データ数（検索空間の大きさ）、データサイズ(S:小、L:大、F:固定長、U:不定長)を用いる。

表２：応答時間に関する特性づけ

P1 の特性づけ

	入力	出力	ﾊﾟﾀｰﾝ	数	ｻｲｽﾞ
S1	tel	name	R	100,000	S/F
S2	name	tel	R	100,000	S/F
S3	zip	name	R	100	S/F

P2、P3 の特性づけ

	入力	出力	ﾊﾟﾀｰﾝ	数	ｻｲｽﾞ
S1	tel	name	R	100,000	S/F
S2	name	tel	R	100,000	S/F
S3	zip	name	R	100	S/F
S4	string	name	S	100,000	L/U

(2)要求の分離：応答時間に関係のないデータの意味(入出力)の列を無視し、同じ項目を縮退させることにより、応答時間に関る要求事項を取り出すことができる。例えば P1 の S1、S2 は応答時間に関しては同様の要因のため、同じカテゴリ(C1a)であると判断される。

表３：応答時間に関する要求の分離

P1 への要求

	ﾊﾟﾀｰﾝ	数	ｻｲｽﾞ	応答時間
C1a	R	100,000	S/F	< 2-3 sec
C1b	R	100	S/F	< 2-3 sec

P2、P3 への要求

	ﾊﾟﾀｰﾝ	数	ｻｲｽﾞ	応答時間
C2a	R	100,000	S/F	< 1 sec
C2b	R	100	S/F	< 1 sec
C2c	S	100,000	L/U	< 5-6 sec

(3)インパクトの分析：分離された要求を分析し、潜在的にインパクトを持ち得ると考えられるカテゴリを検討する。ここでは要因面からの特性と、要求される応答時間のレベルから、利用する基盤の上では、C2a、C2c の達成について検討が必要であると判断されたたものとする。

(4)候補の検討：C2a、C2c について、どのような実現構造でそれが達成できそうか、候補を挙げ、妥当と思われる方式案を検討する。その結果、C2a の実現には起動時にインデクス部分をメモリに読み込む方式で、C2c については、CD-ROM のシーク時間を最少にするように物理フォーマットに依存したサーチ方式[3]を採用することで、要求の達成ができると判断されたものとする。前者のための実現構造を A1、後者のための実現構造を A2 とする。また論理構造に沿った実現構造を A0 とする。なおここで、A0、A1 は基盤への依存性が弱いが、A2 は基盤へ強い依存性をもった方式である。P1、P2 の利用する基盤(I1)に依存したものを A2a、P3 の利用する基盤(I2)に依存したものを A2b とする。

(5)共通性と差異の理解：以上に基づき、P1、P2、P3 間の共通性と差異を理解することができる（図５）。

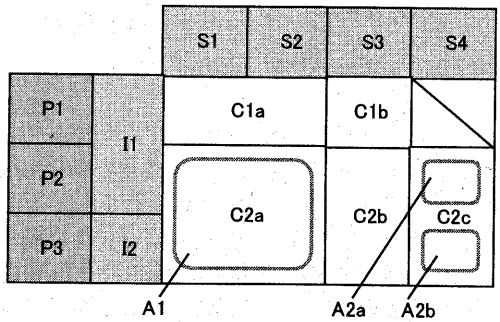

図５：P1、P2、P3 の共通性と差異

P1、P2、P3 は類似した機能要求を持っている、すなわち図４の地図情報を管理しておけば､機能面からはすべて機能を実現できる。しかしながら応答時間を考慮すると、P1 と P2 、P3 は、かなり異なった要求を持っていることがわかる。また P2 と P3 は、要求は同じだが基盤が異なるためアーキテクチャ的に異なる部分を含むこと､P1 は基盤への依存性が弱いことなどが分かる。

5.3 設計の方向性検討

アスペクト指向分析の結果に基づいて設計の方向性を決める。図５の結果をそのまま踏まえて図３のリファレンスとの対応を考えると、表４のように整理することができる。なおここで AS1～AS4 はそれぞれ S1～S4 を実現するための構造を、また H0a、H0b は A0 のための基盤仮想部（それぞれ I1、I2 に対応）を示すものとする。

表４：リファレンスとの対応

機能実現部(変動部)	AS1、AS2、AS3, AS4
機能実現部(固定部)	A0
基盤仮想部	H0a、H0b
品質達成部(基盤非依存)	A1
品質達成部(基盤依存)	A2a、A2b

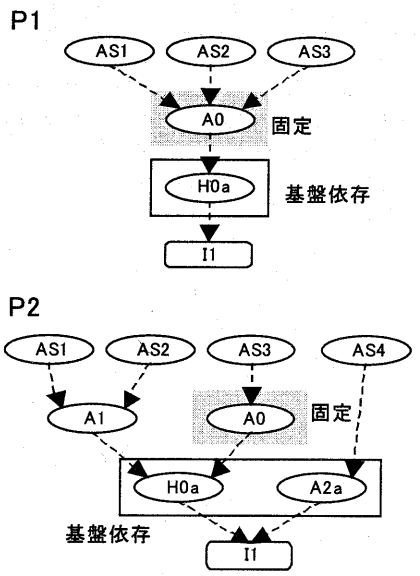

図６：P1 と P2 のアーキテクチャ構成

なお、AS3 を固定部に含めるのがよいかどうかは検討が必要である。これはソフトウェアの進化をどう想定するかに依存する判断であるが、ここでは将来 S3 に対する品質要求が変動する可能性を考慮し、論理構造に対する基本的な処理を提供する A0 のみを固定部に入れ、それに基づく様々なサービス実現は変動部に入れるという判断を行った。これに基づいた P1 と P2 のアーキテクチャの基本的な構成は図６のようになる。P3 は、P2 のアーキテクチャのうち、基盤依存部分が H0b、A2b になる。

実際の設計段階では、ここで示された構造の分離や依存関係を反映するようにアーキテクチャを設計する。

6 議論

本稿では、ソフトウェアの進化や、プロダクトラインの開発を想定し、要求や基盤の変化に対して安定なアーキテクチャを検討するための、アスペクト指向分析について提案した。特に本手法は、品質特性面の要求を分析し、アーキテクチャへの影響という観点から要求をカテゴライズする点に特徴を持つ。

アスペクト指向の考え方は当初プログラミングの分野でアスペクト指向プログラム(AOP)として紹介され[2]、その分野での研究が継続して進められており、また設計への適用も検討されている[6]。AOP におけるアスペクトはコンポーネントをまたがるクロスカットをカプセル化するものであり、設計の立場からは、コラボレーションのカプセル化と捉えることができる。一方本手法におけるアスペクトは、機能（コラボレーション）に対して、表レベルでの投影と縮退を適用することによって得られるが、これに際して品質特性に関る要因で機能を特性づけてから分割を行っている点に特徴がある。これにより、単なる機能面の分割からは見えない側面の要求カテゴリを捉えることに特徴がある。

アーキテクチャは、潜在的にその上に構築されるソフトウェア群を捉えることによって、それに対する要件がくっきりと見えてくる。そのときに大切なことは、そのア

ーキテクチャがどのような要求レベルを受け止めることができるかという、要求カテゴリの認識である。特にその際には、機能面だけでなく、品質特性面への考慮がかかせない。トイプログラムとミッションクリティカルなシステムとは、仮に類似した機能を持っていたとしても、要求される品質特性が異なるため、同じアーキテクチャで実現することはできない。本手法のねらいは、潜在的なソフトウェア群が内包している要求のレベルを発見することである。

本手法はアーキテクチャ設計の初期段階に適用し、設計の方向性を見いだすことを目的としている。この段階の目的は、個別の詳細な検討に入り込む前に、全体の特性を概観し、適切な開発戦略をたてることである。設計の選択肢は数多くあるが、我々はすべての選択肢について詳細な検討をすることはできないし、いったん方向づけがなされると、それに依存した設計上の判断が次々となされていくため、その方向づけを変更することはほとんど不可能になる。アーキテクチャ設計においては、こうした初期の方向づけを適確に行わないと、いきなり詳細に入り込み大きな方向を間違う危険性がある。

本手法では、リファレンスに照らしてアーキテクチャの基本的な構成を検討した。プロダクトラインを構成する各製品について構成を検討することによって、ある製品から次の製品へ展開する際に、アーキテクチャ上でどのような変更があり、それがどの範囲に影響を持つかを判断する材料にすることができると考えられる。プロダクトラインの適切な開発や展開のシナリオや戦略を検討することは重要なテーマであり、本手法がそれにどのように応用できるかは、今後の課題である。

7 おわりに

本手法は、プロダクトラインのアーキテクチャ手法を検討する過程で得られた観測に基づいて整理体系化したものである。有効性の評価等を今後進めていきたい。

謝辞

本研究に関し、貴重な議論とアドバイスを頂いた北陸先端科学技術大学院大学片山卓也教授、青木利晃助手、ならびに研究室の皆様に深謝いたします。

参考文献

[1] Bass, L., et.al.: Software Architecture in Practice, Addison-Wesley, 1998.

[2] Kiczales, G., et.al.: Aspect-Oriented Programming, In proceedings of the European Conference on Object-Oriented Programming (ECOOP), Jun. 1997.

[3] Kishi, T. and Noda, N: Analyzing Hot/Frozen-spots from Performance Aspect, OPLA ECOOP, 1999.

[4] Kishi, T. and Noda, N.: Aspect-Oriented Analysis for Product Line Architecture, SPLC1, 2000.

[5] 岸知二: ソフトウェアアーキテクチャに関する考察 –AHPを活用したアーキテクチャ選択 -, ソフトウェア工学研究会報告, vol.2001, No.31, 2001.

[6] Noda, N. and Kishi, T.: On Aspect-Oriented Design – An Approach to Designing Quality Attributes -, In proceedings of the 6th Asia-Pacific Software Engineering Conference (APSEC '99), Dec. 1999.

[7] Ran, A., "Architectural Structures and Views", *Proceedings of the Third International Software Architecture Workshop (ISAW3)*. 1998.

[8] Shaw, M. and Garlan, D.: Software Architecture, Perspectives on an Emerging Discipline, Prentice Halll, 1996.