Vol.2014-SE-183 No.13
2014/3/19

SOA アプリケーションプラットフォームのプロダクトライン化

江坂 篤侍[1]　野呂 昌満[2]　沢田 篤史[2]

概要：本研究の目的は SOA アプリケーションプラットフォームのプロダクトライン化である．一般にプロダクトラインにおける共通部分，可変部分は適切な仕様モデル (フィーチャモデル等) 上に明示的に表現される．通常，大規模ソフトウェアにおける仕様モデル上のコンポーネントとアーキテクチャ (プロダクトモデル) 上のコンポーネントの関係は複雑である．この複雑性に起因して，モデル間の追跡性が確保出来ず，結果としてソフトウェアの保守性が損なわれる．本研究では，SOA アプリケーションプラットフォームの仕様モデルとアーキテクチャの対応関係を整理した．アーキテクチャは非機能特性ならびに，いくつかの機能特性を関心事として分離する事で，アスペクト指向アーキテクチャとして定義できた．複数のプロダクトを分析することで，プロダクトライン上の共通部分と変動部分を整理した．この結果に基づいて仕様モデル上に共通部分と変動部分を記述した．さらにこの仕様モデルとアーキテクチャ記述を用いて別の複数事例のプロダクトラインを定義した．これにより，仕様モデル上のコンポーネント群と対応するアーキテクチャ上のコンポーネント群は多対多の関係であることが確認出来た．

Construction of Product Line for SOA Application Platform

Esaka Atsushi[1]　Noro Masami[2]　Sawada Atsushi[2]

Abstract: This paper concerns about construction of a product line for SOA application platforms. In practice, relationship between a specification model which shows commonality and variability of the product according to the requirements and an architecture which gives the structure of software is so complicated. The traceability from the specification model to the architecture is not so transparent because of the complexity. We have defined the relationship between the specification and the architecture. The architecture we have constructed is aspect-oriented one in which non functional and several functional characteristics are identified as crosscutting concerns. Through the implementation of several SOA products, we could have constructed the aspect-oriented software architecture. Also, we have identified the commonality and variability of the product line. To apply the specification model and the architecture to a different case, we ended up with that several components in a specification model relate to the several components in an architecture.

1. はじめに

SOA に基づくシステム (以下, SOA システム) の開発においては，メッセージ変換やサービスレジストリなどの SOA アプリケーションを動作させるためのミドルウェア [7][8] の利用が不可欠である．ミドルウェアはそれぞれに想定するアプリケーションのアーキテクチャをもち，開発プロセスや利用出来るツールを限定する．例えば，Microsoft Silverlight[13] を Web アプリケーションフレームワークに採用すると，オペレーティングシステムは Windows に，プログラミング言語は C#に制限され，データベースアクセス時の排他制御にはモニタの利用が強制される．ミドルウェア間のメッセージ通信にも前提条件が存在し，例えば Web サービスフレームワークに WCF[16] を採用すると，Silverlight へのメッセージングはコールバック方式に限定される．

[1] 南山大学大学院数理情報研究科
Graduate School of Mathematical Sciences and Information Engineering, Nanzan University
[2] 南山大学情報理工学部ソフトウェア工学科
Department of Software Engineering, Nanzan University

本研究の目的は，特定の製品や技術に依存しないアプリケーションプラットフォームの開発環境の実現である．そのために SOA アプリケーションを動作させる共通プラットフォーム (以下，アプリケーションプラットフォーム) を定義する．アプリケーションプラットフォームをプロダクトライン化することにより，ミドルウェアの差異を吸収し開発環境を整備する．ミドルウェアが実現すべき機能を標準化し，可変点と不動点を識別し，ミドルウェアのプロダクトラインを定義する．すなわち，可変点と不動点を斟酌し，仕様モデルを記述する．SOA アプリケーションの非機能特性と機能特性を考慮してアスペクト指向ソフトウェアアーキテクチャを定義し，プロダクトラインアーキテクチャとする．仕様モデルとアーキテクチャの追跡性を確保することで，プロダクトラインアーキテクチャからプロダクトアーキテクチャの生成を自動化する．プロダクトラインアーキテクチャの実現であるアプリレーションフレームワークにおいてミドルウェア間の差異を吸収するメカニズムを実現する．

大規模な SOA システムの場合，仕様モデルのコンポーネントとアーキテクチャのコンポーネントの間には複雑な関係がある．本研究ではこの複雑性は横断的関心事に起因するものと考えた．仕様モデル上のコンポーネントは関心事を表現しており，これが支配的分割に散在する．変動点としての関心事とこれを構成する候補を整理し，複数の候補によって規定されるアーキテクチャ上のコンポーネントを整理する．変動点の整理には，Bachmann ら [4] によって示されているプロダクトラインにおけるバリエーションポイントの分類に従う．整理された変動点と候補値に基づき仕様モデルを記述することで，仕様モデルとアーキテクチャの関係を明確にする．

2. SOA に基づくシステムのためのアプリケーションプラットフォーム

SOA システムのためのアプリケーションプラットフォームは，SOA アプリケーションを動作させるための共通プラットフォームである．メッセージ変換やサービスレジストリなどの SOA システムを動作させるための基盤となる機能を提供する．この機能は既存のミドルウェアを組み合わせることで実現される．ミドルウェアの制約により，様々なミドルウェアに関する知識無しには，適切なミドルウェアを選定して，SOA アプリケーションを開発するのに多大な労力が必要となる．

本研究では，プロダクトラインソフトウェア開発 (以下，PLSE)[12] に着目し，特定の製品や技術に依存しないアプリケーションプラットフォームの開発環境を実現する．PLSE は，プロダクトラインにおける共通性と変動性を識別し，再利用資産に基づく開発を行なう．これは本研究の，ミドルウェアの制約から独立し，普遍的な開発環境を実現するという目的に合致する．一般に PLSE では次の要素をコア資産として定義する．

- 共通アーキテクチャとしてのプロダクトラインアーキテクチャ
- 共通性と変動性を分析し，この結果に基づいて記述される仕様モデル
- プロダクトラインアーキテクチャに基づき，共通の枠組みの元にプロダクトを構築可能なフレームワーク

我々が構築するプロダクトラインでは，プロダクトラインアーキテクチャはミドルウェアから独立したプロダクトラインの共通構造を示す．互いに依存し合うアプリケーションプラットフォームに対する要求は，これを考慮したプロダクトアーキテクチャのコンポーネントの特定を困難にする．

仕様モデルは，ミドルウェアの差異が表現される．互いに依存し合う要求を仕様モデル上に記述することは困難である．

フレームワークでは，共通構造とミドルウェアを適用する再利用コンポーネントが定義される．再利用コンポーネントではミドルウェア間の差異を吸収するメカニズムを実現する必要がある．

さらに，コア資産間の追跡性が確保は一般に仕様モデルのコンポーネントとアーキテクチャのコンポーネントは散在，もつれ合いの複雑な関係にあることから困難である [14]．この複雑な関係の例を**図 1** に示す．仕様モデルとアーキテクチャ間の複雑性は横断的関心事に起因するものと考えた．仕様モデルのコンポーネントが表現する関心事の組み合わせによって規定されるアーキテクチャ上のコンポーネントを整理する必要がある．図 1 では，特定の技術や動作環境の組み合わせによってアーキテクチャのコンポーネントが特定されることを示す．

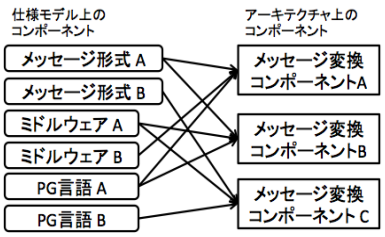

図 1 仕様モデルのコンポーネントとアーキテクチャのコンポーネントの関係

3. アプリケーションプラットフォームのプロダクトライン化の手順と関連技術

アプリケーションプラットフォームのプロダクトライン化のプロセスと，各工程において用いた関連技術を**図 2** に示す．我々は横断的関心事の抽出にあたり，SOA システムにおける標準アーキテクチャと品質に関する国際基

準を参考にする．そこで，我々は Clements らの Views & Beyond[15] と S3 Architecture[1] と ISO9126[6] を用いた．プロダクトラインアーキテクチャは，アスペクト指向技術を適用することにより，抽出された横断的関心事を分離するアスペクト指向プロダクトラインアーキテクチャとして構築した．プロダクトラインにおける共通点，変動点の整理には Bachmann らによって示されているバリエーションポイントの分類の標準に従った．仕様モデルは，共通部分と変動部分の分析結果に基づき，Feature Oriented Reuse Method(以下，FORM)[9] の仕様モデル記述法に従って記述した．また，複数の仕様モデル上のコンポーネントによって規定されるアーキテクチャ上のコンポーネントを整理した．アプリケーションプラットフォームのためのフレームワーク，およびミドルウェアをフレームワークに適合させるための再利用コンポーネント群を定義した．

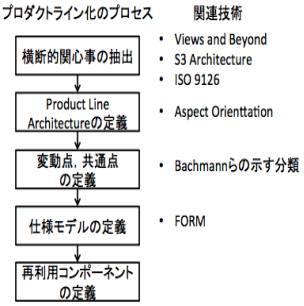

図 2 プロダクトライン化の手順と関連技術

3.1 Views and Beyond

Clements らはアーキテクチャ文書化のための標準的な視点と，視点におけるアーキテクチャスタイルを定義している (Views and Beyond; 以下 V&B)．V&B はアーキテクチャの記述法の標準化を目指したものとして広く受け入れられているとの認識に立ち，本研究はこれに基づいてアーキテクチャ構築のための関心事抽出を行なう．

V&B では SOA システムの構成要素とその構造を記述するための SOA スタイルが提案されている．SOA スタイルを基本構造としたアプリケーションプラットフォームの構造を**図 3** に示す．SOA スタイルは Service Provider, Service Consumer, MessageServiceProvider, Orchestration Server，Registry of Service，MessagingConnector を構成要素としている．アプリケーションプラットフォームはサービス間のメッセージングに関する機能を提供するプラットフォームであることから，MessageServiceProvider，Registry of Service，Messaging Connector を基本構造とする．

3.2 S3 Architecture

S3 Architecture は SOA システムのための標準的な参照

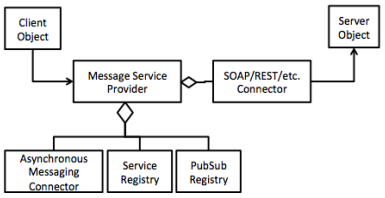

図 3 SOAStyle を基本構造としたアプリケーションプラットフォームの構造

アーキテクチャである．SOA の構築に必要な関心事の分離を行なう指針を提供するために，SOA システムの構成要素とその役割を抽象化し，9 層の階層構造を示している．S3 Architecture で示される階層構造を**図 4** に示す．図中の左側の 5 つの層は SOA アプリケーションの関心事を階層構造により分離している．右側の 4 つの層はシステム全体に横断する関心事を階層構造により分離している．右側の各層で扱われる関心事のは次のとおりである．サービス統合に関する Integration 層，サービス品質に関する Quality of Service 層，システム全体で用いられるデータに関する Information Architecture 層，システムの開発指針などに関する Governance and Policies 層．このうち，サービス間のメッセージングを分離するアプリケーションプラットフォームの関心事は主に Integration 層，Quality of Service 層によって扱われると考え，分析の対象とする．

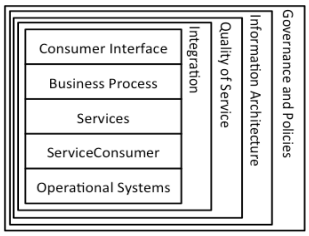

図 4 S3 Architecture の階層構造

3.3 ISO9126

ISO9126 はソフトウェア品質の評価に関する国際基準である．ISO9126 はソフトウェアの持つ様々な特徴を品質の観点から整理したものであり，機能性，信頼性，使用性，効率性，保守性，可搬性の 6 種類の品質特性を定義している．さらに，品質特性をより細かく分類した 27 種類の品質副特性を定義している．

この品質特性を実現する関心事を考察することにより，一般的なソフトウェアに存在する関心事が特定出来ると考えた．一般に非機能特性は横断的関心事である．

3.4 アスペクト指向技術

アスペクト指向技術はシステムを複数の視点でとらえ，これまでのソフトウェア開発技術では解決し辛かった問題に対処することができる [17]．アスペクト指向技術は，支

配的分割によって得られる構造にまたがる大域的な特性 (横断的関心事) をアスペクトとして分離することを可能にする技術である．さらに，分離されたアスペトは互いに依存すること無く変更可能になる．

我々が構築するプロダクトラインのプロダクトラインアーキテクチャはアスペクト指向アーキテクチャとして定義する．一般にプロダクトラインアーキテクチャはアプリケーション群の設計を支配する基本的な構造を表現する [11]．アプリケーションプラットフォームのプロダクトラインアーキテクチャでも，プロダクトラインに共通なアスペクト群とそれらの関係を記述する．一般に非機能要求は横断的関心事となるのでこれをアスペクトとして分離することにより，非機能要求に応じたチューニングを容易にすることを狙っている．アスペクト指向アーキテクチャの構築に必要となる横断的関心事の抽出にあたり，前節までに説明した SOA システムにおける標準アーキテクチャを参考にする．

3.5 ソフトウェアの変動部分の分類の標準

Bachmann らはプロダクトラインにおける変動部分の分類をアーキテクチャを中心に整理している．Bachmann らの示す分類は, Function, Data, Control Flow, Technology, Quality Goal，Environment の 6 種類である．

プロダクトラインにおける変動点と不動点を分析では Bachmann らによって示されている分類を参考にする．また変動点の候補値と他の変動点の候補値の間の依存関係についても整理する．この分類を参考にすることで標準的な視点からプロダクトラインにおける変動点を分析出来る．これらを通じ，プロダクトに求められる特性やその特性を実現するミドルウェアが整理される．Bachmann らの示す分類は，システムで考慮される関心事の分類として考えることができる．すなわち，これらに分類される変動点の候補値はアーキテクチャ上の複数のコンポーネントに影響を与える．逆にいくつかの候補値の組み合わせにより，アーキテクチャ上のコンポーネントが特定される．

3.6 フィーチャモデル

フィーチャモデルは，プロダクトライン開発において代表的に用いられている仕様モデルである．フィーチャモデルはフィーチャ図とフィーチャの補足情報で構成される [10]．フィーチャ図は，プロダクトラインの共通部分，変動部分をフィーチャ群とこれらの関係で表現している．フィーチャ図の構成を**図 5** に示す．フィーチャの補足情報はフィーチャ図で表されるフィーチャの変動性を要求に応じて決定するための情報である．Kang らは FORM において，フィーチャの役割を明示するためにフィーチャを層によって分類する表記法を提案した．

プロダクトラインにおける変動点と不動点を分析の結果

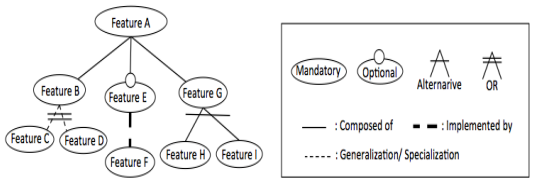

図 5 フィーチャ図の構成

に基づき，コア資産としての仕様モデルを記述する．仕様モデルはプロダクトライン開発において一般的に用いられている FORM の仕様モデル記述法に従って記述する．

4. SOA アプリケーションプラットフォームのプロダクトライン化

図 2 に示した手順に従い，前章で説明した各技術を適用することで，SOA アプリケーションプラットフォームをプロダクトライン化した．

以降よりアプリケーションプラットフォームのプロダクトラインを説明するために，コア資産としてのプロダクトラインアーキテクチャと仕様モデル，およびこれらの追跡性を定義した過程と共に述べる．プロダクトラインアーキテクチャに基づき，フレームワークを実装することでアプリケーションプラットフォームの普遍的な開発環境が実現される．

4.1 横断的関心事の抽出

4.1.1 SOA スタイルから抽出した横断的関心事

V&B における SOA スタイルの構成するコンポーネントの責務から横断的関心事を特定した結果を**図 6** に示す．この図には SOA スタイルのコンポーネントと抽出した関心事の対応が示されている．例えば，MessageService-Provider(MSP) は WebRegistry を利用し位置透過なメッセージングを実現することから，MSP と WebRegistry 間に位置透過 (LocationTransparency) についての関心事が横断していることを示している．

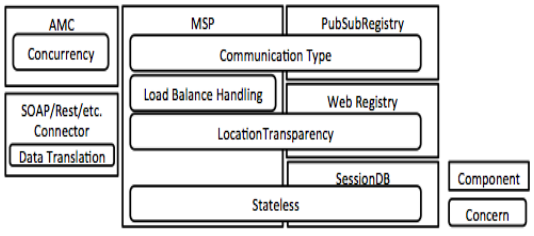

図 6 SOA スタイルの構造と特定した関心事 (一部)

4.1.2 S3 Architecture から抽出した横断的関心事

S3 Architecture の各層の扱う抽象化された関心事から，具体的に SOA システムで考慮される関心事を特定した結果を**図 7** に示す．この図には各層の関心事とそれを具体化した関心事を示している．例えば，Integration 層のサービ

ス統合では特定のメッセージ形式にメッセージを変換することから，Integration Concern の具体的なものの一つとしてデータ変換 (Data Translation) についての関心事が存在することを示している．

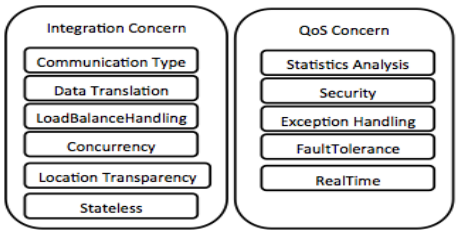

図 7 S3 Architecture の層と特定した関心事 (一部)

4.1.3 ISO9126 から抽出した横断的関心事

ISO126 の示す品質副特性から，この品質副特性を実現するための関心事を特定した結果を**表 1** に示す．この表には品質副特性とこれを実現する関心事を示している．例えば，時間効率性を向上させるためには負荷分散処理や実時間処理を実現，修正することから，時間効率性に対応して負荷分散処理 (LoadBalanceHandling) や実時間処理 (Real Time) についての関心事が存在することを示している．

表 1 ISO9126 の品質特性と特定した関心事 (一部)

Concern	分割 (ISO9126 品質特性.副特性)	関心事
Quality Characteristic Concern	Functionality.Security(セキュリティ)	Security
	Reliability.Maturity(成熟性)	FaultTolerance
		ExceptionHandling
		Persistence
	Reliability.FaultTolerance (耐故障性)	FaultTolerance
		ExceptionHandling
	Reliability.Recoverability (回復性)	FaultTolerance
		Persistence
	Maintainability.Analyzability(解析性)	StatisticsAnalysis
	Maintainability.Testability(試験性)	StatisticsAnalysis
	Efficiency.TimeBehavior (時間効率性)	LoadBalanceHandling
		RealTime
	Efficiency.ResourceUtilization (資源効率性)	LoadBalanceHandling
		Concurrency
		Persistence
		RealTime
	Portability.Replaceability (置換性)	LocationTransparency
		Stateless

4.2 アスペクト指向プロダクトラインアーキテクチャの定義

前節で特定した横断的関心事に基づき，これを分離したアスペクト指向アーキテクチャを記述した．記述したプロダクトラインアーキテクチャを**図 8** に示す．黒色の線では Object 間の Packet の送信，受信関係を示している．このオブジェクト間のメッセージングに関する関心事をアスペクトとして分離していることを赤色の線で示している．オブジェクトが別のオブジェクトにメッセージを送信するさい，これを実行せず Communication Type Aspect にメッセージが送られる．Communication Type Aspect では 1 対 1，1 対多の通信方式によるメッセージングが行なわれる．この Communication Type Aspect とこれに対して横断する Aspect 群 (Concurrency Aspect や Data Translation Aspect など) の協調動作により本来メッセージが送られるはずだったオブジェクトにメッセージが送信される．

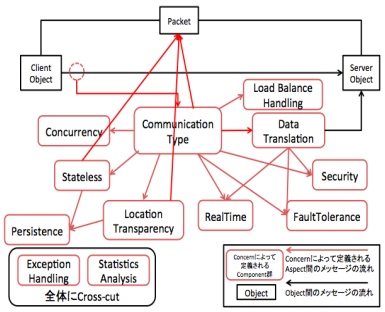

図 8 アプリケーションプラットフォームのプロダクトラインアーキテクチャ

4.3 プロダクトラインにおける変動点と不動点の分析

前節で定義したプロダクトラインアーキテクチャに基づき，複数の SOA システムの構築を行なった．この開発を通じて得られた知見に基づいてプロダクトラインにおける変動点と不動点を分析し，仕様モデルを定義，アーキテクチャ上の変動するコンポーネントを整理した．

Bachmann らの分類を参考にし，過去の開発事例における施策を分析することによりプロダクトラインの変動点と不動点を整理した．この結果を**表 2** に示す．

表 2 アプリケーションプラットフォームのプロダクトラインにおける変動点と候補値 (一部)

分類	変動部分(Variation Point)	候補値(Variant)
Function	送信方法	OneToOne, OneToMany
	メッセージ変換	メッセージ変換無,独自メッセージ形式変換, 標準メッセージ形式変換
Data	Message Format	Message Format 無,SOAP, REST,Binary,JSON,RMI
	Session Data	Session Data 有/無
Control Flow	Scheduling Algorithm	Scheduling Algorithm 無, FIFO,優先度付き
	サービス特定Algorithm	サービス特定Algorithm 無, 重み付き,Round Robin
	障害回復処理	Recovery,Retry,N-Version, Data Recovery
Technology	WS Framework	Apache CXF,Axis2
	Middleware.ESB	ESB無,JBossESB, MuleESB,ServiceMix
	Middleware. Marshalling Library	JIBX,CXF,MessagePack,Jackson, MessageProtocol,JavaRMI
Quality Goal	Interoperability	Data Translation Aspect
	Fault Tolerance	FaultTolerance Aspect,Exception Handling Aspect,Persistence Aspect
	Time Behaviour	LoadBalanceHandling Aspect, RealTime Aspect,ServiceDeployment Aspect
	Replaceability	Location Transparency Aspect
Environment	Programming Language	Java,AspectJ
	Service Allocation	Local,Remote,WS
	Data Allocation	File,DB,OnMemory

ここでは，この変動点と候補値を特定した過程を説明する．過去の開発事例では，いくつかのコンポーネントにより 1 対 1 のメッセージングと，1 対多のメッセージングが実現された．これは機能における変動点として送信方法，候補値として OneToOne，OneToMany と考えた．また，こ

のメッセージングを実現するコンポーネントはSOAP形式のメッセージを構築し送信している．実行効率を向上させるためにSOAP形式のメッセージとしていたものをRESTメッセージに変更した．このことから，Quality Goalの分類に属する変動点として実行効率，Dataの分類に属する変動点としてメッセージ形式，Technologyの分類に属する変動点としてMarshalling Libraryがあり，この候補値の間には依存関係があることが分かる．以上のように，施策に対して各分類に該当する側面があるか考察することにより，変動点について整理を行なった．

4.4 アプリケーションプラットフォームの仕様モデル

これまでに整理した変動点と不動点をFORMの記述法に従って仕様モデルとして表現する．変動点としての関心事とこれを構成する候補値が仕様モデル上のコンポーネントとして記述される．

仕様モデルを記述するために，FORMの層とBachmannらの示す分類の対応関係を整理した．これにより変動点に関連するフィーチャをどの層に記述すべきか明確となった．Bachmannらの示す分類とFORMの層の対応関係を**図 9**に示す．Capability層には機能，非機能フィーチャを配置することから，Functionと Quality Goalの分類がフィーチャとして現れる．Operating Environment層には環境に関するフィーチャを配置することから，動作環境の分類であるTechnologyと開発環境の分類であるEnvironmentがフィーチャとして現れる．Domain Technology層にはドメイン特有の実現技術に関するフィーチャを配置することから，Dataと Control Flowの分類の中でもSOAシステム特有のものがフィーチャとして現れる．Implementation Technique層には一般に用いられる実装技術に関するフィーチャを配置することから，Dataと Control Flowの分類の中でも一般的なものがフィーチャとして現れる．

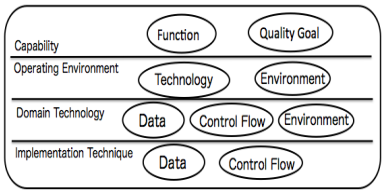

図 9 Bachmannの分類とFORMの層の関係

変動点と不動点の分析結果と分類と層との対応関係に基づき仕様モデルを記述した．仕様モデルの一部を**図 10**に示す．また，要求に応じてフィーチャを選択するための情報として，Czarneckiらの文献で示されているフィーチャの補足情報[10]を整理した．変動フィーチャを選択することにより，プロダクトの仕様が表現される．この仕様モデルとアーキテクチャの追跡性により，要求に応じてプロダクトのアーキテクチャが構築可能となる．

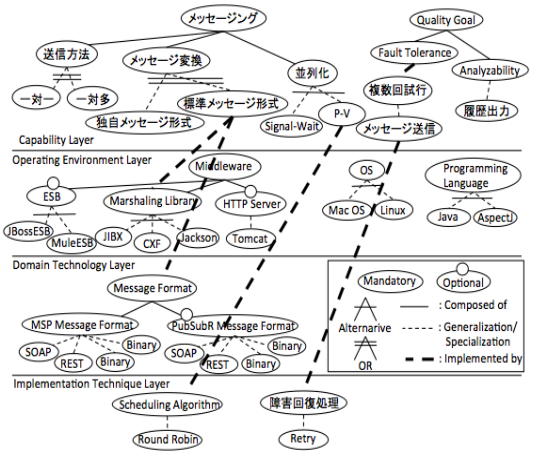

図 10 アプリケーションプラットフォームの仕様モデル (一部)

4.5 候補値の組み合わせに対応して変動するアーキテクチャ上のコンポーネント

過去の開発事例に基づき，いくつかの変動点としての関心事の候補値によって規定されるアーキテクチャ上のコンポーネントを整理した．整理した結果の一部を**表 3**に示す．Message Translation Concernを実現するアスペクトの変動点として，Functionの分類に属するメッセージ変換と，Dataの分類に属するメッセージ形式と，Technologyの分類に属するMarshalling Libraryがある．これらの候補値を選択することにより，アーキテクチャ上のコンポーネントが特定される．機能として，メッセージ変換が存在するかどうかによって，Data Translation Aspectの有無が決定される．次に，5種類の変換するメッセージ形式とメッセージ変換に利用される7種類のMarshalling Libraryを特定した．候補値の間の依存関係に基づく組み合わせ方により現状のプロダクトラインでは8種類のコンポーネントが特定可能である．アーキテクチャ上のコンポーネント全てについて同様に分析し，候補値とコンポーネントの関係を整理した．

表 3 候補値に対応するアーキテクチャ上のコンポーネント例

バリエーションポイント	バリアント \ Component	Original Connector	<< JIBX >> SOAP Connector	<< CXF >> SOAP Connector	<< CXF >> REST Connector
メッセージ変換	メッセージ変換無	-	-	-	-
	独自メッセージ形式変換	○	-	-	-
	標準メッセージ形式変換	○	○	○	○
Message Format	Message Format 無	○	-	-	-
	SOAP	-	○	○	-
	REST	-	-	-	○
	Binary	-	-	-	-
	JSON	-	-	-	-
	RMI	-	-	-	-
Marshaling Library	Marshaling Library 無	○	-	-	-
	JIBX	-	○	-	-
	CXF	-	-	○	○
	Message Pack	-	-	-	-
	Jackson	-	-	-	-
	ProtocolBuffers	-	-	-	-
	Java RMI	-	-	-	-

5. 考察

5.1 プロダクトライン化のアプローチの妥当性

SOA ミドルウェアの標準化は発展途上にあり，各ベンダは独自の仕様から実装している．この差異によりミドルウェアの取り替えによるチューニングは困難である．したがってミドルウェアの差異を吸収し，普遍的なアプリケーションプラットフォームを提供する必要がある．

既存のプロダクトライン開発に関する研究の多くは，ミドルウェアの効果的な利用を対象としていない．機能を豊富に備える汎用的なミドルウェアは，過度のリソース消費やパフォーマンスの低下に繋がることから最適化が必要である．Gokhale らは，これ問題をとし，ミドルウェアの利用も取り入れたプロダクトライン開発を提案している [2]．

本研究では，ミドルウェアに依存しない共通アーキテクチャを構築し，これを中心としたプロダクトライン開発の実現を目指すものである．これにより，要求に応じたミドルウェアの選択と，特定のミドルウェアに依存しない普遍的な開発が可能となる．

コア資産には，アスペクト指向アーキテクチャに基づく追跡性が確保されており，保守や進化に適した構成になっている．ミドルウェアの仕様が今後変更されたとしても，この追跡性によりプロダクトラインの洗練が容易である．

一般にミドルウェアの効果的な利用を対象としたプロダクトライン化が必要とされており，本研究のアプローチはこれを実現するものであることから，妥当と考える．

5.2 アスペクト指向プロダクトラインアーキテクチャの妥当性

プロダクトラインアーキテクチャはプロダクトラインに共通なアーキテクチャであり，これにより前節で述べた本研究の目的を達成する．本研究で示したプロダクトラインアーキテクチャは SOA システムにおけるアーキテクチャに関する標準に基づいて構築したことから，特定の製品や技術に依存しない．一般に SOA システムで考慮される関心事の識別という目的に対して V&B，S3 Architecture，ISO9126 からほぼ共通した関心事が特定したので参考とする標準の種類として，十分であると考える．

本研究と同様のアプローチの関連研究として，Kumara らの提案する ESB のためのアスペクト指向フレームワーク [5] がある．Kumara らは，ESB 製品の共通アーキテクチャを定義し，アスペクトの織り込みにより要求に応じた ESB を構築している．ESB 製品毎にアーキテクチャが異なるものの，共通の機能を提供しておりその実装も同じサードパーティのライブラリを利用している，共通アーキテクチャを構築し ESB に特化したアスペクトを織り込むことにより普遍的な ESB 製品の構築を実現している．本研究と対象は異なるもののアプリケーションプラットフォームについても，アスペクト指向アーキテクチャに基づいてフレームワークを構築することで前節で述べた目的を達成出来ると考える．

5.3 仕様モデルとアーキテクチャの対応関係の妥当性

仕様モデルのコンポーネントとアーキテクチャのコンポーネントは散在，もつれ合いの複雑な関係にある．この複雑性に起因して，モデル間の追跡性の確保が困難である．

本研究では，いくつかの例において，横断的関心事を介して仕様モデルとアスペクト指向アーキテクチャの関係が明確となった．仕様モデル上の機能，非機能フィーチャ，およびこれらに関連するフィーチャを選択することにより，アスペクトとこれを構成するコンポーネントが特定可能であることを確認した．仕様モデルとアーキテクチャの散在，もつれ合いの関係は横断的関心事に起因するものであると考えた．従って，横断的関心事を分離したアスペクト指向アーキテクチャを構築し，変動点としての関心事とこれを構成する候補を仕様モデルとして記述し，これらの関係を整理した．結果として，いくつかの例において，要求に応じて Feature を選択することにより，対応するアスペクトとこの構造が決定されることを確認した．

5.4 アプリケーションプラットフォームの有用性

アプリケーションプラットフォームはアプリケーションデータに依存しない単純なメッセージ通信を必要とする SOA システムに適用可能である．SOA システムのアーキテクチャに関する標準から一般的なメッセージングに関わる関心事を識別し，これをアスペクトとして実現している．従って，一般的なメッセージングに関する関心事を実現している．いくつかの事務システムについてアプリケーションプラットフォームを適用可能であることを確認した．

アプリケーションプラットフォームと同様にメッセージング機能を提供するものとして ESB 製品がある．Kumara らによると一般に広く用いられている ESB 製品群に共通して提供されている機能は，ルーティング，メッセージ変換，メッセージング，セキュリティ，負荷分散，永続化である．アプリケーションプラットフォームは ESB 製品群が共通に提供するメッセージング関する機能を提供する．

アプリケーション特有のメッセージングに関する要求に対応可能となるように洗練する必要がある．例えば，アプリケーション特有のメッセージ形式やルーティングのストラテジなどに柔軟に対応可能とする構造について考察する．また，特定の ESB 製品の提供する独自の機能などを参考に拡張する必要がある．これらについて新たなアスペクトの追加として捉え，構築することによりプロダクトラインの拡張は容易であると考える．

5.5 今後の課題

今後の課題として事例検証を繰り返し，これによって得られる知見に基づいて仕様モデルとアーキテクチャ，およびこれらの関係を洗練することが上げられる．また，新たな変動部分が特定された場合，新たな候補値が特定された場合，本研究の手順に基づき仕様モデルを拡張可能であることを確認する．

さらに，プロダクトアーキテクチャに応じてプロダクトを構築するためのフレームワークを実現することも今後の重要な課題である．本研究により，要求を入力とし，プロダクトの仕様モデルを構築し，これに応じてプロダクトアーキテクチャの構築が可能となった．アーキテクチャに応じてプロダクトを構築するためのフレームワークを構築することにより，要求からプロダクト構築までの系統的な手順による開発が実現可能となる．

また，プロダクトを構築する手順，プロダクトラインを拡張する手順について整理し，定型的な作業を自動化するツールを構築することによりプロダクトライン開発を支援することも必要であると考えている．Mattsson らの記述ルールモデルとモデル変換ルールを用いた手法 [3] を参考にし，プロダクトアーキテクチャを自動構築する実行可能な仕様モデルを構築することも検討していきたい．

6. おわりに

本研究では，ミドルウェアの差異を吸収し，特定の技術や製品から独立した普遍的な SOA システムの開発を実現することを目的とした．この目的を達成するために，SOA システムのメッセージング機能を提供するプラットフォームのプロダクトライン化を行なった．さまざまな標準を組み合わせることにより，コア資産としてのアスペクト指向アーキテクチャと仕様モデル，およびこれらの関係を整理した．これにより，プロダクトに対する要求を入力とし，プロダクトアーキテクチャの生成が自動化された．

今後の課題は，仕様モデルとアーキテクチャ，およびこれらの関係について洗練，アーキテクチャに基づき実装を行なうためのフレームワークを整備することである．また，定型的な作業を自動化するツールを構築することも今後の課題と位置づけている．

謝辞 本研究の一部は，JSPS 科研費 (基盤研究 (C)24500049)，および 2013 年度南山大学パッヘ奨励金 I-A-2 の助成による．また，株式会社キャナリーリサーチの支援のもとに遂行された．ここに謝意を表する．

参考文献

[1] A. Arsanjani, L. J. Zhang, M. Ellis, A. Allam, and K. Channabasavaiah, “S3: A service-oriented reference architecture,” *IT Professional*, vol. 9, no. 3, pp. 10-17, 2007

[2] A. Gokhale, A. Dabholkar, S. Tambe, “Towards a Holistic Approach for Integrating Middleware with Software Product Lines Research,” Proc. *of the 1st Workshop on Modularization, Composition, and Generative Techniques for Product Line Engineering held as part of GPCE08*, 2008.

[3] A. Mattsson, B. Fitzgerald, B. Lundell and B. Lings, “An Approach for Modeling Architectural Design Rules in UML and its Application to Embedded Software,” *ACM Transactions on Software Engineering and Methodology*, vol. 21, no. 2, article 10, 2012.

[4] F. Bachmann, L. Bass, “Managing Variability in Software architectures, ” *ACM SIGSOFT Software Engineering Notes*, vol. 26, no. 3, pp. 126-132, 2001.

[5] I. Lumara, C. Gamage, “Towards Reusing ESB Services in Different ESB Architectures,” *Computer Software and Applications Conference Workshops (COMPSACW), 2010 IEEE 34th Annual*, IEEE, pp. 25-30, 2010.

[6] ISO/IEC, *Software Engineering Product quality - Part 1: Quality model*, 2001.

[7] Jackson (online), 入手先 ⟨`http://jackson.codehaus.org/`⟩(2014.02.18).

[8] jUDDI (online), 入手先 ⟨`https://juddi.apache.org/`⟩(2014.02.18).

[9] 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.

[10] K. Czarnecki, and U. W. Eisenecker, *Generative Programming: Methods, Tools, and Applications*, Addison-Wesley, 2000.

[11] K. Pohl, G. Bockle, and F. Linden, *Software Product Line Engineering Foundations, Principles, and Techniques*, Springer, 2005.

[12] L. M. Northrop, “SEI’s Software Product Line Tenets,” *IEEE Software*, vol. 19, pp. 32-40, 2002.

[13] Microsoft Silverlight (online), 入手先 ⟨`https://www.microsoft.com/ja-jp/silverlight/`⟩(2014.02.18).

[14] P. Sochos, M. Riebisch, and I. Philippow, “The feature-architecture mapping (farm) method for feature-oriented development of software product lines,” Proc. *IEEE International Symposium and Workshop on Engineering of Computer Based Systems*, 13th Annual IEEE International Symposium and Workshop, pp. 308-318, 2006.

[15] P. Clements, F. Bachmann, L. Bass, D. Garkan, J. Ivers, R. Little, R. Nord, and J.Stafford, *Documentiong Software Architectures Views and Beyond Second Edition*, Addison Wesley, 2010.

[16] Windows Communication Foundation (online), 入手先 ⟨`http://www.visualstudio.com/ja-jp/downloads/`⟩(2014.02.18).

[17] 野呂昌満, “アスペクト指向プログラミング概観,” Seamail, vol.13, no11.