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のプロダクトライン化に関する研究

M2011MM007 江坂篤侍
指導教員：野呂昌満

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，生産性，信頼性の高い製品の構築にミドルウェアの利用は必要不可欠である．これらのミドルウェアは位置透過性の保証や，サービスレジストリ，メッセージングなどの機能を実現する．

ミドルウェアの製品毎にアーキテクチャが異なり，定義される API の構文とその実装が同一でないことから，SOA システムの開発において以下の問題が発生する．

- ミドルウェアの選択が開発形態を左右する
- ミドルウェアが重視する非機能特性をアプリケーションは引き継ぐ

ミドルウェアの差異を吸収し，アプリケーションをミドルウェア独立にすることは，ミドルウェアの利用に対してシームレスな開発を実現することから有用である．

本研究は，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を目的とし，プロダクトライン化のための基礎を定義する．アプリケーションプラットフォームは SOA アプリケーションとミドルウェア間の相互作用を仲介する．本研究で提案される仕様モデルとプロダクトラインアーキテクチャにより，プロダクトライン開発のプロセスが定義可能となる．これにより，特定の技術や製品から独立した普遍的な開発支援の提供を可能とする．

プロダクトラインアーキテクチャの定義はアスペクト指向に基づいて行なう．プラットフォームに存在する横断的関心事を抽出し，可変性の分析として，それぞれ関心事の実現に用いられるミドルウェアを分析する．関心事と可変性の分析結果は仕様モデルとして記述し，プロダクトラインアーキテクチャとの追跡性を確保する．このアーキテクチャに基づいてプラットフォームを実現するフレームワークと，そのホットスポットにミドルウェアを適合するための再利用コンポーネント郡を整備する．

考察では，SOA システムの開発における課題を整理するとともに，本研究のアプローチの妥当性について述べる．

2 SOA システムの開発とミドルウェアへの依存性

SOA ミドルウェアは多くのベンダにより配布されているが，これらのアーキテクチャは標準化されていない．したがって，ミドルウェアの API の構文や実装はミドルウェア毎に異なる．

例えば，Axis2 を WebService フレームワークに採用すると，Service で用いるメッセージの形式は SOAP に限定される．さらに，プログラミング言語は java に制約される．その他にも同様にミドルウェア製品毎に，要素技術，言語，OS に制約が存在し，ミドルウェアを用いたシステムではその制約が引き継がれる．要求の変化に別のミドルウェアの乗り換えによる解決は．乗り換え前のミドルウェアの持つ制約により，移植には多大な労力が必要である．

アプリケーションプラットフォームにより，ミドルウェアの差異を吸収し，普遍的なプロセスに基づいた開発を支援するために，次の四つの課題を解決する必要がある．1) 仕様化：SOA システムに求められる非機能要求の整理と仕様化，2) 設計：ミドルウェア郡が提供する機能の共通性と可変性の分析とモデル化，3) 実装：共通の枠組みのもとに可変性を吸収するためのメカニズムの実現，4) 追跡性：仕様化，設計，実装の間の意味的な一貫性の保証．

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

前章で掲げた課題を解決するために，本研究ではプロダクトラインソフトウェア開発に着目し，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を行なう．プロダクトライン開発は，製品系列における共通性，可変性を分析し，定義された共通部品，可変部品を組合せることによる開発のパラダイムである．これは，SOA システムの開発において様々なミドルウェアの差異を吸収し，普遍的な開発環境を整備する目的にも合致する．SOA ミドルウェアはサービスレジストリ，ルーティングなど同類の機能を提供する．同類の機能を提供するミドルウェアの要素技術，言語，OS などを可変性とするプロダクトラインの形成が可能であると考えた．

本研究が目的とするプロダクトラインにおけるコア資産は次の通りである．1)SOA システムに対する関心事とその可変性を表現する仕様モデル，2) アスペクト指向に基づくプロダクトラインアーキテクチャ，3) 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するための再利用コンポーネント郡．さらに，この仕様モデル，アーキテクチャ，フレームワーク間の関係を整理することが，前章の「仕様化」,「設計」,「実装」,「追跡性」の課題に対する解と位置づける．

3.1 横断的関心事の抽出とプロダクトラインアーキテクチャ

プロダクトラインアーキテクチャをアスペクト指向アーキテクチャとして構築する．一般に非機能要求は横断的関心事となる．ミドルウェアの利用はアプリケーションの性能を決定付ける要因となる．プロダクトラインアーキテクチャに対して，アスペクト指向を導入し，横断的関心事を分離する．分離することで，ミドルウェアの乗り換えによる性能のチューニングが可能となる．

本研究では，Views and Beyond[6] で標準化しているアーキテクチャの記述法に従いアプリケーションプラットフォームの基本構造を定義した．さらに，S3 アーキテクチャ[1] を参照することで，横断的関心事を抽出した．

3.1.1 アプリケーションプラットフォームの基本構造

Documenting Software Architectures : Views and Beyond(以下，V&B) で定義される SOA Style に基づき，アプリケーションプラットフォームの構造を定義する．V&B にしたがってアーキテクチャを記述することにより，ステークホルダがソフトウェアを理解することを支援する．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ことから，本研究はこれを受け入れる．

V&B は複数の視点におけるシステムの基本構造とそれらの関係を文書化することでアーキテクチャを記述している．すなわち，特定の視点における記述はアーキテクチャの側面を表現し，これらの集合によってアーキテクチャは表現される．

V&B はアーキテクチャとして文書化すべき視点を次の 3 つに標準化している．1)Module View Type : ソフトウェアの静的構造を表現，2)Component and Connector View Type : ソフトウェアの動的構造を表現，3)Allocation View Type : ソフトウェアと開発及び実行の環境との関係を表現．各ビュータイプにおいては，構成要素の型とそれらの関係の型を定義して意味的な取扱いの基礎を定義している．

各ビュータイプはそれぞれ複数のスタイルとして整理している．これらのスタイルは多くのアーキテクチャから共通するパターンを見つけ出し，定義したアーキテクチャの表記法である．

V&B で定義されている SOA Style は SOA システムのアーキテクチャの書き方の標準である．これに従うことでアプリケーションプラットフォームを特定のドメインに特化しない標準的なものとする．SOA Style は，要素として Message Service Provider，Service Bus，Registry，Messaging Component，Service Consumer，Service Provider とそれらの関係を定義している．Service Consumer，Service Provider 以外の要素とそれらの関係がアプリケーションプラットフォームの基本構造である．

3.1.2 横断的関心事の抽出

本研究では，S3 アーキテクチャを分析することで横断的関心事の抽出を試みる．S3 アーキテクチャは SOA のリファレンスアーキテクチャであり，V&B のスタイルである Layered Style に従い，SOA を整理している．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，階層構造として定義されている．

S3 アーキテクチャは SOA システム全体を 9 層の階層構造で表現する．SOA システムの関心事を層に分けて整理している．S3 アーキテクチャの階層構造を図 1 に示す．S3 アーキテクチャは，SOA システムを構築するために必要とされる関心事を分離する指針となる．左 5 層は SOA システムを機能階層に分割したものである．右 4 層は機能階層対する横断的関心事を分離した層である．9 つの層の概要を述べる．1)Operational System 層 : 運用環境で動作する SOA システムの実現に必要な全てのアプリケーションの集合を扱う，2)Service Component 層 : サービスの機能を実現するコンポーネントを扱う，3)Service 層 : SOA のサービスの定義すべてを扱う，4)Business Process 層 : 複数のサービスを順序付けられたプロセスとして集約する，5)Consumer 層 : 人間，他のアプリケーションにアプリケーションへのエントリーポイントを提供する，6)Integration 層 : 右 4 層間の相互作用を仲介する横断的な層であり，メッセージの仲介，ルーティング，送信を行なう，7)QoS 層 : SOA が非機能要件を満たす事を確実にする手段を提供する横断的な層，8)Information Architecture 層 : システムで用いられるデータを管理する横断的な層，9)Governance and Policies 層 : SOA システム全体におけるシステム提供者の意思決定のための情報が定義される横断的な層．

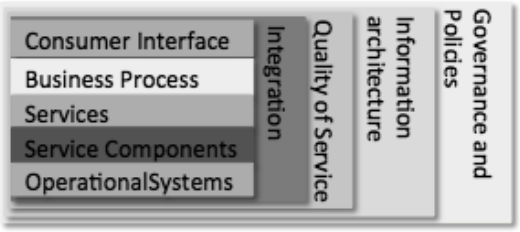

図 1 S3 アーキテクチャの階層構造

アプリケーションプラットフォームの構成要素は Integration 層に配置され，QoS，InformationArchitecture，Governance and Policies 層は上位の層に横断する関心事を分離する．このことから，QoS，InformationArchitecture，Governance and Policies 層に基づき，アプリケーションプラットフォームの横断的関心事を抽出する．横断的関心事を分離した結果として，図 2 に示すようにプロダクトラインアーキテクチャは構築された．

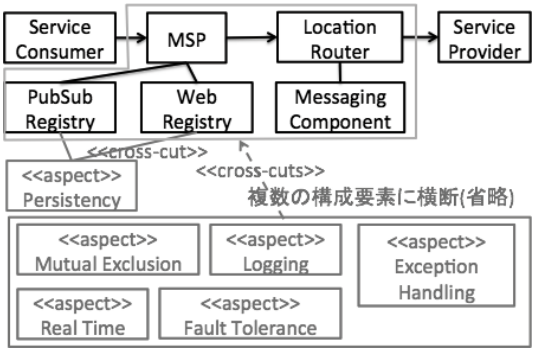

図 2 プロダクトラインアーキテクチャ(一部)

QoS 層は，各層のサービス品質の準拠を監視し，信頼性，可用性，管理性，スケーラビリティ，セキュリティを扱うことから，実行効率，例外処理，耐故障性，ロギング，排他制御，ステートレス，実行効率の関心事を抽出した．S3 アーキテクチャでは物理的配置について表現し

ないが，物理的配置を考慮することにより，Information Architecture，Governance and Policies 層において次の関心事を抽出した．Information Architecture 層は，データ構造，メタデータ，データ交換を扱うことから，永続データとして配置されたデータを扱う永続性の関心事を抽出した．Governance and Policies 層は，システムの要求を実現するポリシを扱う事から，ハードウェアやネットワーク上へのコンポーネントの配置に対する関心事を抽出した．

3.2 プロダクトの変動要因の分析と仕様モデル

構築したアーキテクチャに基づき，SOA システムを構築し，変動要因を分析した．関心事に対する可変性は用いられる要素技術や動作環境であることが分かった．構築された SOA システムの実行結果の比較により得た知見から品質特性との関係を整理した．

実行効率の関心事に対する可変性を例に整理の過程を説明する．構築した SOA システムを実行，比較した結果，次の要素が実行効率に影響を与えることが分かった．1) セッション管理方法，2) メッセージの形式，3) データ永続化の実現方法，4) コンポーネントの実現方法，5) 物理的配置．これらの要因がどのように実行効率と関わるのかを詳細に分析し，表 1 に示すような形で整理した．

表 1 実行効率の関心事と可変性の関係 (一部)

		効率性
		時間効率性
セッション管理	Cookie	CookieはSessionDBへのアクセスが無い分，実行効率が高い．しかし，Cookieを用いることによるメッセージサイズの肥大化に伴う実行効率の低下の可能性がある．SessionDB:メッセージング毎にSessionDBにアクセスし，送信先の状態を取得する必要があるので時間効率は低い
	SessionDB	
メッセージ形式	SOAP	Binaryが最速．それ以外はメッセージの内容の大きさによって，速度は変わる．XML形式のメッセージはシリアライズ / デシリアライズに時間が掛かる．軽量なメッセージであれば，REST形式が速いが，メッセージが大きくなるに従ってJSONの方が速くなる． (ここでのRESTはRESTアーキテクチャに従った，メッセージの形式を指す．例：HTTPリクエストに対し，XML形式のメッセージを返す)
	REST	
	XML	
	JSON	
	Binary	
通信方式	同期	非同期通信:クライアント側は応答を待たずに次の処理を続行可能．しかし，プロバイダ側の処理完了から即時応答があるとは限らない．非同期通信実現のためにメッセージが冗長になる可能性があり，それにより遅くなることも考えられる．同期通信:クライアント側はプロバイダ側が処理を完了するまで待つ．また，プロバイダ側の稼働状況に依存する．プロバイダ側で処理が完了したら即時応答がある．
	非同期	
物理的配置	同一マシン上	物理的な距離を考えても，同一マシン上に配置した方が速い．1つのマシンにアクセスが集中する場合は，別マシン上に配置した方が速い．
	別マシン上	
エンティティ実現方法	DB化	メモリ化>ファイル化>DB化の順でデータへのアクセス速度が速いことから，この順に時間効率性が高い
	メモリ化	
	ファイル化	
コンポーネント実現方法	WS化	Serviceに対するメッセージより，メモリ上のオブジェクトに対するメッセージの方が速い
	メモリ化	

次に，それぞれの要素技術とミドルウェアの依存関係を表 2 に示す通りに整理した．ミドルウェアは言語，OS と依存関係にあることからこれらの関係も整理した．

同様の分析を，前節で抽出した関心事に対して行ない，それぞれ表の形式で整理した結果に基づき，プロダクトラインの仕様モデルを図 3 に示すように定義した．仕様モデルの表記法には，FORM(Feature-Oriented Reuse Method)[5] に基づくフィーチャモデルを用いた．最上段の Capability Layer には非機能要求に対する関心事と，アプリケーションプラットフォームで提供される機能が配置されている．その下には，ミドルウェアとミドルウェアが実装する要素技術が表現されている．フィーチャ間の依存関係は破線として表される．図 3 は，理解性を考慮し，非機能フィーチャに関連する依存関係以外は省略する．これにより，非機能要求によってそれを実現するミドルウェア明確となる．可変フィーチャの選択を入力とし，プロダクトラインアーキテクチャのアスペクトの具体的な要素が決定される．

表 2 ミドルウェアと言語と OS と可変性の関係 (一部)

		言語		OS		変動要因との関係
		Java	C#	Win	Mac	
DB	jUDDI	○	×	○	○	データ永続化の実現
	PostgreSQL	○	○	○	○	データ永続化の実現
	Oracle	○	○	○	○	データ永続化の実現
OR Mapper	Hibernate	○	×	○	○	データ永続化の実現
	TopLink	○	×	○	○	データ永続化の実現
	JDBC	○	×	○	○	データ永続化の実現
Logger	Log4j	○	×	○	○	
WS Framework	Axis2	○	×	○	○	コンポーネントのWS化 対応メッセージ形式：SOAP
	CXF	○	×	○	○	コンポーネントのWS化 対応メッセージ形式：SOAP, JSON, REST, Binary
シリアライズ/デシリアライズライブラリ	JIBX	○	×	○	○	SOAPメッセージ実現
	Jackson	○	×	○	○	JSONメッセージ実現
	MessagePack	○	○	○	○	Binaryメッセージ実現
	GoogleMessageProtocol	○	○	○	○	Binaryメッセージ実現

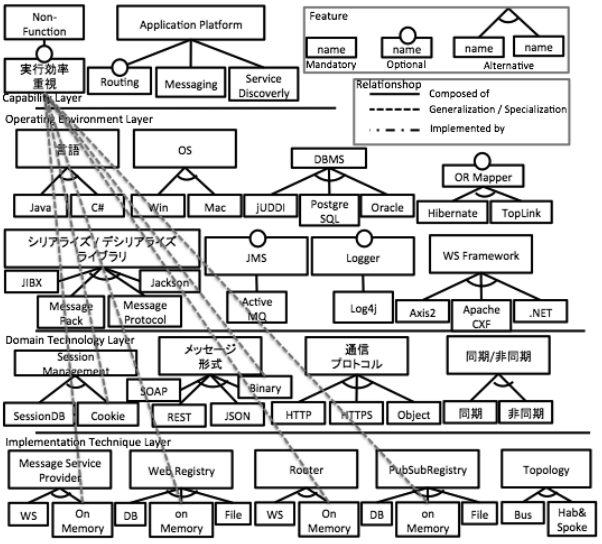

図 3 プロダクトラインの仕様モデル

3.3 フレームワークと再利用コンポーネント

本研究では，ミドルウェアの差異を吸収し，アーキテクチャに適合させるための手段として，アプリケーションプラットフォームのフレームワークを定義する．アーキテクチャ上のアスペクトは可変フィーチャ選択により変動することから，ホットスポットとして定義する．さらに，各々のホットスポットにミドルウェアを適合させるための再利用コンポーネント郡を用意する．図 4 のように，フレームワークのホットスポットは Interface ステレオタイプによって識別する．

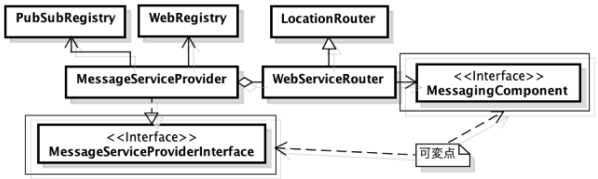

図 4 ホットスポットの設計 (一部)

再利用コンポーネントの設計は GoF デザインパターンの適応し設計する．例えば，アダプタパターンを用いることにより，継承関係によって API の差異を統一し，実装の差異を吸収する．図 5 は，メッセージ形式のホットスポットに対し，Axis2(SOAP)，MessagePack(Binary) を適合させるアダプタの設計である．

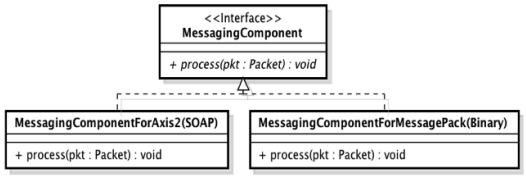

図 5 定義したアダプタの例

4 考察

4.1 アプローチの妥当性

アプリケーション構築において信頼性，開発コスト削減などの利点からミドルウェアの利用は不可欠である．しかし，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．Gokhale らは，これ問題をとし，ミドルウェアの最適化も取り入れたプロダクトライン開発が必要としている [2]．SOA システムの既存研究においても同様の問題がある．

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．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．また，ミドルウェアの仕様の変更に柔軟に対応可能な方法論を提案する必要がある．

次の点から提案するプロダクトラインは，目的に対して妥当であると考える．1) 提案するプロダクトラインは性能要求を満たすミドルウェアの選択を支援する，2) プロダクトライン開発はコア資産の保守や進化も想定した方法論である．本研究で提案した仕様モデルは，プロダクトの変動要因と関心事の関係が明確に表現されている．すなわち，アプリケーションの要求の変更に対してそれを実現するミドルウェアの識別が可能である．本研究で提案したプロダクトラインアーキテクチャは，ミドルウェア間で共通のモデルとして定義されている．アスペクト指向技術アーキテクチャを導入したことにより，ミドルウェア乗り換え，ミドルウェアの仕様変更に対して柔軟に対応可能となった．仕様モデルでは，関心事と変動要因の関係が整理されており，アーキテクチャでは横断的関心事を分離している．これにより，可変フィーチャの選択から，プロダクトアーキテクチャの構築が可能となった．

5 今後の課題

今後の課題は，次の 3 点である．1) 一般性の確認，2) アーキテクチャの洗練，3) 開発プロセスの提案．

事例検証を繰り返すことで提案するプロダクトラインの一般性の確認を行なう．様々な SOA システムを用いて事例検証を繰り返すことで本研究の一般性を確認する．

SOA Style 以外の Style も考慮し，アプリケーションプラットフォームのアーキテクチャの洗練する．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 [3] は，Pipes and Filters Style に従い，ESB のアーキテクチャを構築している．ESB は，Client からのメッセージを段階的に変換し，Provider に通知する特徴から，Pipes and Filters Style が適しているとしている．本研究は，SOA Style を支配的分割としているが，今後，Pipes and Filters Style と SOA Style の混合 Style やその他の Style についても考察を行なうことで，変更に対する柔軟性や，提供する機能の変動要因が増すと考えている．

プロダクトアーキテクチャからコードの自動生成の仕組みを提案する．MDA を導入することでプロダクトの高い生産性の実現が可能である．

6 まとめ

本研究は，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を目的とし，SOA システム構築の支援した．提案した仕様モデル，プロダクトラインアーキテクチャに基づき，開発プロセスを定義することで，プロダクトライン化される．アプリケーションプラットフォームが SOA アプリケーションとミドルウェアの間を仲介することで，アプリケーションをミドルウェア独立にした．これにより，ミドルウェアの仕様に依存しないアプリケーションの構築，要求に対し，適切なミドルウェアの選択の支援行なった．

参考文献

[1] A. Arsanjani, L. J. Zhang, M. Ellis, A. Allam, and K Channabasavaiah, "*S3: Service-Oriented Reference Architecture*", IEEE Computer Society, vol. 9, 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] I. Lumara, C. Gamage," *Towards Reusing ESB Services in Different ESB Architectures*", Computer Software and Applications Conference Workshops (COMPSACW), 2010 IEEE 34th Annual. IEEE, 2010.

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

[5] 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.

[6] P. Clements, F. Bachmann, L. Bass, D. Garkan, J. Ivers, R. Little, R. Nord, and J.Stafford, " *Documentiong Software Architectures Views and Beyond*", Addison Wesley, 2007.