Vol.2013-SE-179 No.25
2013/3/12

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

江坂 篤侍[1]　野呂 昌満[2]　沢田 篤史[2]

概要：SOA ミドルウェアは製品毎に API の構文とその実装が同一ではないことから，特定のミドルウェアを選択することでアプリケーションの開発形態が左右される．本研究では，ミドルウェアの差異を吸収し，シームレスな開発支援を実現することを目的に，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う．コア資産としてのプロダクトラインアーキテクチャは，プラットフォームに対する非機能要求を関心事として抽出し分離することで，アスペクト指向アーキテクチャとして定義する．それぞれのアスペクトに対する可変性はミドルウェア製品の調査に基づいて分析し，その結果をアーキテクチャ文書化のための標準的な視点にしたがって整理する．この結果に基づき，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と，そのホットスポットに適用する再利用コンポーネントをアーキテクチャに対するアダプタとして定義する．本稿では，これらを通じたプロダクトライン化の概要について説明するとともに，提案手法の妥当性，一般性についての考察を行う．

On Product Line of Application Platform for SOA Based Systems

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

Abstract: The development process of an SOA based system is profoundly affected by the SOA middleware used. The reason is that each middleware product has different syntax and semantics of API. This paper presents our approach to constructing a product line for SOA application platform which mediate between SOA based applications and the middleware used. In order to provide an effective support for SOA application development, we defined an aspect-oriented architecture in which we implement non-functional requirements for SOA platforms as aspects. Variability analyses are done by inspecting middleware products and the results are classified according to the views defined in the existing standard for architecture documentation. In order to bridge over the gap between a middleware and another, we designed and implemented application framework. We also defined reusable components which fill out the hot spots in the application framework.

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，信頼性の高い製品を高い生産性で構築するためにミドルウェアの利用は必要不可欠である．これら SOA ミドルウェアは，位置透過性を保証し，サービスレジストリ，メッセージングなどの機能を提供するが，製品毎にアーキテクチャが異なり，定義される API の構文とその振舞いが同一でない．これに起因して，SOA システムの開発において，ミドルウェアが前提とする制約によって開発プロセスや利用できるツールが限定されてしまい，結果としてシステムがその制約に制約されるという問題がある．

本研究の目的は，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のプロダクトライン化である．これにより，ミドルウェアの差異を吸収し，特定の技術や製品からは独立した普遍的な開発支援の提供を可能とする．

プロダクトラインのコア資産を構成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

[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

う．プラットフォームに対する関心事を抽出し，それぞれの関心事に対するミドルウェア製品毎の可変性を分析することで，アスペクト指向アーキテクチャとして定義する．関心事とその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個々の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群を整備する．

本稿では，以下，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のプロダクトライン化に向けた課題を整理するとともに，既存の技術や標準の組み合わせによって定義したプロダクトラインについて説明する．これらアスペクト指向アーキテクチャに基づくプロダクトラインの妥当性と有用性についての考察も行う．

2. SOA システムの開発とミドルウェアへの依存性

SOA システムの開発は，ミドルウェアを利用して行うことが一般的である．ここで SOA システムのためのミドルウェアとは，OSI のレイヤ 3 上でネットワークを介した位置透過性を保証し，サービスの要求者から提供者間のメッセージングや，サービスのレジストリ，オーケストレーションなどの機能を提供する基盤ソフトウェア群である．

SOA システムのミドルウェアは，多くのベンダにより配布されており，多くの SOA に基づくアプリケーション開発に用いられているが，提供する API の構文やその振舞いが製品毎に異なっている．

例えば，Microsoft Silverlight[1] 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[2] 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．

同様の制約は他のミドルウェア製品にも存在し，それらに起因して，SOA システム開発のプロセスや利用可能なツールが制限され，構築されるシステムもミドルウェアの持つ前提条件が強制される．開発初期に決定したミドルウェア群が開発プロセスを支配し，特にシステムの非機能特性を左右する状況は好ましいとは言えない．万一，開発したシステムが顧客の要求に照らして妥当でなかった場合，ミドルウェアの前提に反する修正やチューニングは困難で，別のミドルウェア群への乗り換えで解決できたとしても移植には多大な労力が必要となる．

SOA のためのアプリケーションプラットフォームにおいてこのような問題を引き起こすミドルウェア間の差異を吸収し，普遍的なプロセスに基づいて開発支援を提供するためには，次の四つの課題を解決する必要がある．

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

設計　ミドルウェア群が提供する機能の共通性と可変性の分析とモデル化

実装　共通の枠組みの元に可変性を吸収するためのメカニズムの実現

追跡性　仕様化，設計，実装の間の意味的な一貫性の保証

3. 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

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 [3] 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う．プロダクトライン開発は，製品系列における共通性と可変性の構造的なモデル化と再利用資産化に基づいて包括的な再利用を行うための方法論である．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．

本研究で提案するプロダクトラインのコア資産は次の通りであり，

- 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
- 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
- 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
- 一貫した開発プロセスにしたがって SOA システムの開発を支援する環境

それぞれ，前節の「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 関心事の抽出とプロダクトラインアーキテクチャ

SOA システムを開発する際に開発者は様々な関心事について考慮する．本研究では，SOA システムのための参照アーキテクチャである S3 アーキテクチャ [4] を分析することで関心事の抽出を試みる．S3 アーキテクチャは，既存の SOA システムを構成する要素とその役割を抽象化し，図 1 に示す 9 層の階層構造により表現する．参照アーキテクチャの仕様記述から，Integration 層とその上にある 5 層が SOA システムの構成要素とその統合の視点から見た関心事，すなわちシステムの構造に対応していることが分かる．一方，右側の 3 層 (Quality of Service, Information Architecture, Governance and Policies) は，それぞれがシステムの構造に横断する関心事を表現している．

これらの関心事を明確に分離し，プロダクトラインの中

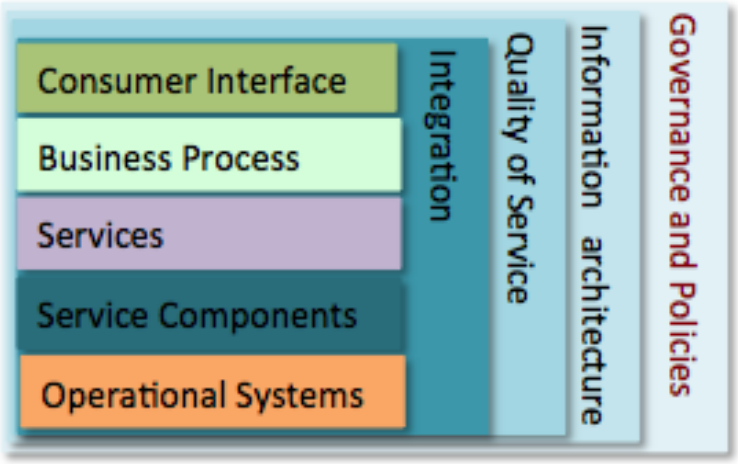

図 1　S3 アーキテクチャの階層構造 [4]

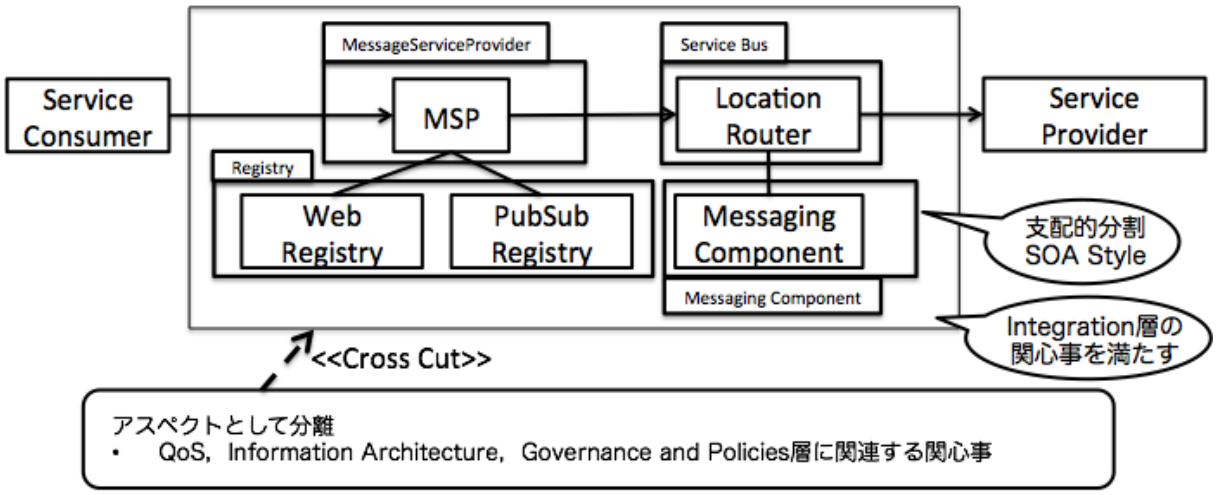

図 2　アスペクト指向プロダクトラインアーキテクチャ

で再利用可能とするために，アスペクト指向によるプロダクトラインアーキテクチャを定義する．アーキテクチャ定義は，Clements らによるアーキテクチャ文書化のためのフレームワーク (Views and Beyond; 以下 V&B)[5] に基づいて行う．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 Style が提供されている．本研究では，Integration 層とその上の 5 層を，このスタイルに基づき，図 2 に示す通り，Message Service Provider (MSP)，Service Bus，Registry，Messaging Component の構成要素からなる基本構造として定義する．3 層に対応する関心事は，それぞれ層の仕様をソフトウェア品質モデルの標準 [7] を参考にしながら分析し，この基本構造に横断するアスペクトとして詳細化する．

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ことから，ハードウェアやネットワーク上へのコンポーネント配置に対する関心事を抽出した．これら関心事は，いずれも SOA システムに対する非機能要求に相当し，基本構造に横断することから，プロダクトラインアーキテクチャにおけるアスペクトとして定義することができる．

3.2　可変性の分析と仕様モデル

我々は，前節で定義したアスペクト指向アーキテクチャに基づき，異なるミドルウェアを用いて，複数の小規模な SOA システムの構築を行った．この開発を通じて得られた知見に基づいて，ミドルウェアの差異に起因する可変性と関心事との関係を整理する．

ここでは実行効率の関心事に対する可変性の分析を例に整理の過程を説明する．開発した SOA システムの比較検討を行った結果，次の要素が実行効率を変動させる要因となることが分かった．

表 1　実行効率の関心事と可変性の関連

			効率性
			時間効率性
アーキテクチャから識別したプロダクトの可変要因	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>JSON>REST>SOAPの順にシリアライズ / デシリアライズが速い
		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	
配備の視点から識別したプロダクトの可変要因	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	

表 2　ミドルウェアと可変性との関係

			言語		OS		エンティティの実現方法	メッセージング
			Java	C#	Win	Mac		
DB	jUDDI		○	×	○	○	○	
	自作	PostgreSQL	○	○	○	○	○	
		Oracle	○	○	○	○	○	
OR Mapper	Hibernate		○	×	○	○	○	
	TopLink		○	×	○	○	○	
シリアライズ/デシリアライズライブラリ	JIBX(SOAP)		○	×	○	○		○
	Jackson(JSON)		○	×	○	○		○
	MessagePack(Binary)		○	○	○	○		○
	GoogleMessageProtocol (Binary)		○	○	○	○		○

- メッセージの形式，
- セッション管理方法
- エンティティの実現方法
- コンポーネントの実現方法

これらの要因に対する実現技術の選択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．

次に，それぞれの実現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，プログラミング言語とオペレーティングシステムとも依存関係にあることからこれらの関係も含めて分析している．

同様の分析を，前節で抽出した関心事に対して行い，それぞれを表の形式で整理する．この結果を統合し，プロダクトラインの仕様モデルとして図 3 に示すように定義する．仕様モデルの表記法には，FORM(Feature-Oriented Reuse Method)[8] の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の構成要素がプロダクトラインアーキテクチャに対応付いて配置される．その下には，ミドルウェアを選択することによる要素技術の依存関係が表現されている．フィーチャ間の依存関係 (破線) により[*1]，非機能要求に対する関心事がミドルウェアの選択にどのような影響を与えるかを明確にすることができる．

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て，アーキテクチャに適合させるための手段として，SOA 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を構築する．前節で説明した仕様モデルにより，SOA システムに対する非機能要求に照らしたミドルウェアの選択基準を開発者に提示することができる．それぞれの非機能要求すなわち関心事は，プロダクトラインアーキテクチャに定義したアスペクトと対応することから，ミドルウェアを選択することでアーキテクチャのどの箇所が影響を受けるかも明確となる．フレームワークの構築では，影響を受けるアスペクトをホットスポットとして定義し，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群を用意する．

フレームワークの設計は，特定のアスペクト指向プログラミング言語への依存性を排除するために，アスペクト指

*1　図の理解性を考慮し，実行効率に関係するもの以外は省略している．

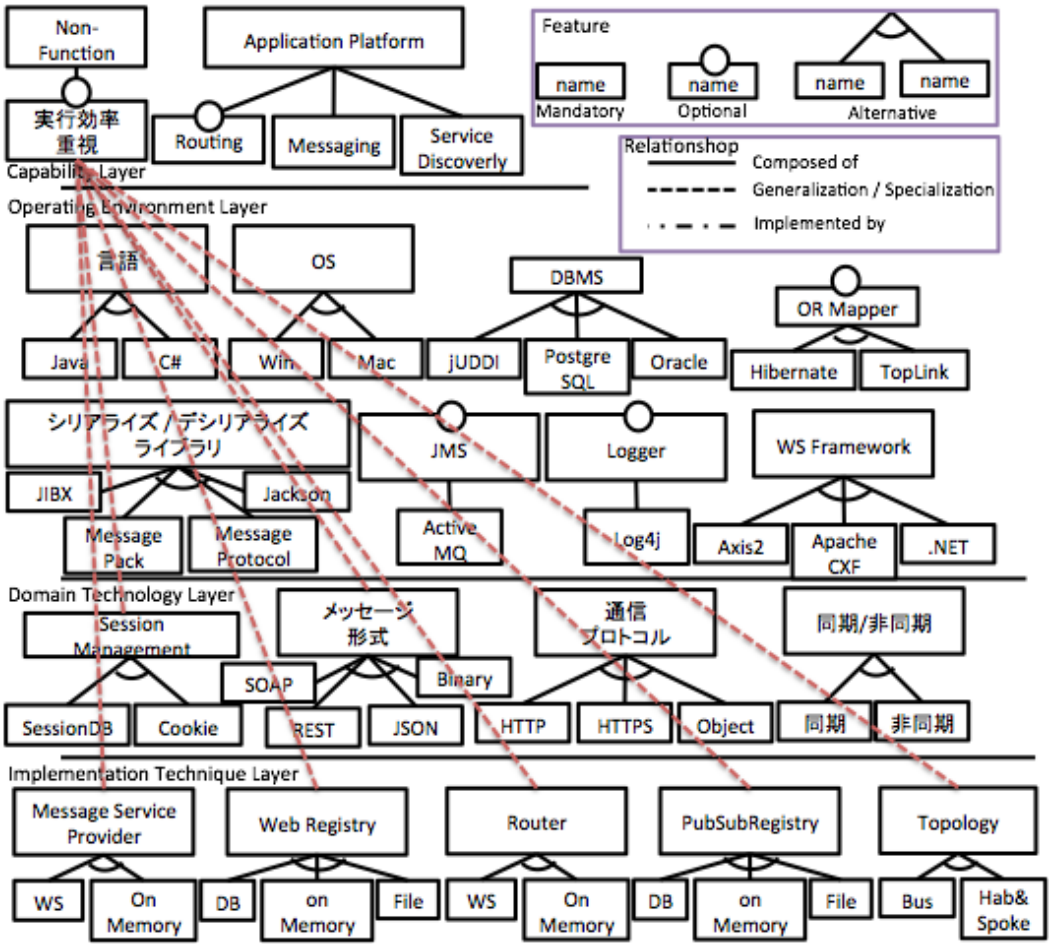

図 3　プロダクトラインの仕様モデル (一部)

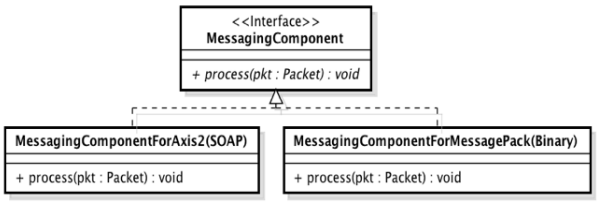

図 5　アダプタコンポーネントの例

向アーキテクチャをオブジェクト指向設計へ対応付けることで行う．GoF デザインパターン [6] を参考に，非機能要求に対するアスペクトをオブジェクト指向により表現する．

ホットスポットは，図 4 に示すように，Interface ステレオタイプにより識別する．再利用コンポーネントの設計は，ホットスポットに対してそれぞれのミドルウェアが採用する要素技術を，継承関係あるいは提供利用関係を用いて適合させることで行う．例えば，図 5 は，メッセージング方式のホットスポットに対して，SOAP(Axis2[9]) と，Binary(MessagePack[10]) を継承関係の利用により適合させるアダプタの設計を示す．

3.4　ツール環境に向けて

前節までに，仕様モデル，アーキテクチャ，フレームワークおよび再利用コンポーネントの設計について説明した．これらのコア資産により，SOA システムに対する関心事 (非機能要求) に対し，特定のミドルウェアに依存することなく開発を行うための普遍的なプラットフォームの提供が可能となる．

我々は，個々の SOA システム開発だけでなく，プロダクトラインの保守や進化をも考慮に入れた開発支援の提供を検討している．非機能要求を考慮したアーキテクチャ設計指針 [11] やアスペクト指向アーキテクチャスタイル [12] とそれに関連する支援ツールなどの成果を利用し，一貫したプロセスに基づく開発支援環境の構築を行っている．これらを洗練し実用化することは今後の課題である．

4.　考察

現在，W3C などを中心に SOA 技術に対する標準化が進められているが，SOA に関連する技術は発展途上であり，その応用領域は今後も拡大すると一般に考えられている．あるミドルウェア製品がその優位性から競争力を持った結果，標準に取り込まれるというデファクト型の標準化のプロセスが予想される．このことは，普遍的な SOA アプリケーションプラットフォームを提供するためには，入手可

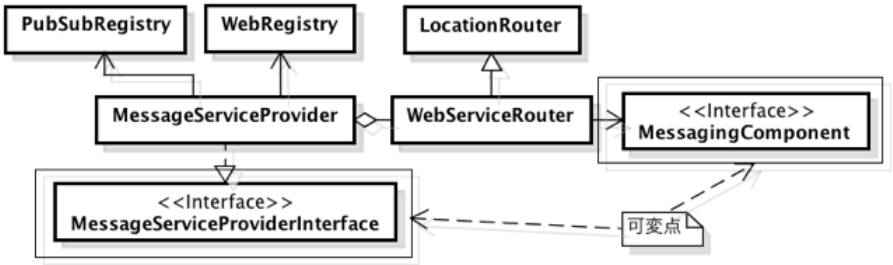

図 4　ホットスポットの設計 (一部)

能な標準に基づいて既存のミドルウェアの差異を吸収するフレームワークを単に設計するだけではなく，それを将来にわたり保守し続ける必要性を示している．

本研究では，ミドルウェア製品の差異を吸収するために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定義するアプローチを採用した．プロダクトラインソフトウェア開発は，コア資産の保守や進化も想定した方法論であることから，目的に対して妥当であると考えている．

我々が定義したコア資産には，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となっている．今後予想されるミドルウェアや要素技術の進化に対しては，個別に対応しなければならないが，この追跡性を利用したモデル変換 [13] などの体系的なアプローチによって保守の効率化が可能である．

5. おわりに

本研究では，SOA システム開発において，ミドルウェアの差異を吸収し，特定の技術や製品からは独立した普遍的な支援の提供を可能とすることを目的に，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った．プロダクトラインのコア資産は，SOA システムに共通の関心事をアスペクトとして抽出したアスペクト指向プロダクトラインアーキテクチャに基づく．ミドルウェアの差異を仕様モデルとして表現し，アーキテクチャとの追跡性を確保した．SOA 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と，そのホットスポットに個々の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群の整備も行った．

本研究の成果は製品レベルの SOA システムの開発に現在適用中である．今後の課題は，ここから得られる知見に基づき，定型的な作業を自動化するツールを充実させることである．プロダクトラインを洗練するとともに実用性の観点から更なる評価を行う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，科学研究費補助金 (基盤研究 (C) 22500037, 24500049)，および 2012 年度南山大学パッヘ奨励金 I-A-2 の助成による．

参考文献

[1] Microsoft Silverlight (online), 入手先 ⟨http://www.microsoft.com/ja-jp/silverlight/default.aspx⟩.
[2] Windows Communication Foundation (online), 入手先 ⟨http://msdn.microsoft.com/ja-jp/vstudio/aa663324⟩.
[3] Northrop, L. M.: "SEI's Software Product Line Tenets", *IEEE Software,* vol. 19, pp. 32–40, 2002.
[4] Arsanjani, A., Zhang, L. J., Ellis, M., Allam, A. and Channabasavaiah, K.: "S3: Service-Oriented Reference Architecture", *IT Pro,* vol. 9, no. 3, pp. 10–17, 2007.
[5] Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., Merson, P., Nord, R. and Stafford, J.: *Documenting Software Architectures: Views and Beyond,* 2nd-ed., Addison Wesley, 2010.
[6] Gamma, E., Helm, R., Johnson, R. and Vlissides, J.: *Design Patterns:Elements of Reusable Object-Oriented Software,* Addison-Wesley, 1995.
[7] ISO/IEC 9126-1, *Software Engineering — Product Quality — Part 1: Quality Model,* 2001.
[8] Kang, K. C., Kim, S., Lee, J., Kim, K., Kim, G. J. and Shin, E.: "FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures", *Annals of Software Engineering,* vol. 5, no. 1, pp. 143–168, 1998.
[9] Apache Axis2 (online), 入手先 ⟨http://axis.apache.org/axis2/java/core/⟩.
[10] MessagePack (online), 入手先 ⟨http://msgpack.org/⟩.
[11] Sawada, A., Noro, M., Chang, H., Hachisu, Y. and Yoshida, A.: "A Design Map for Recording Precise Architecture Decisions", *Proceeding of the 18th Asia-Pacific Software Engineering Conference (APSEC2011),* pp. 298–305, 2011.
[12] Noro, M., Sawada, A., Hachisu, Y. and Banno, M.: "E-AoSAS++ and its Software Development Environment", *Proceedings of the 14th Asia-Pacific Software Engineering Conference (APSEC2007)*, pp. 206-213, 2007.
[13] Object Management Group, Model Driven Architecture (online), 入手先 ⟨http://www.omg.org/mda⟩.