Web サービスプラットフォームに関する研究

2008MI215 関　浩徳　2009SE097 金田幸大　2009SE209 丹羽裕樹　2009SE220 小倉尚起
指導教員：野呂昌満

1　はじめに

多くのエンタープライズ系システムにおいて，ビジネス環境の変化に対応する柔軟性とソフトウェア開発期間の短期化が求められる．ビジネス環境の変化に対応する柔軟性やアプリケーションの生産性を向上させる技術としてサービス指向アーキテクチャ (以下，SOA) が挙げられる．SOA は Web サービスを連携して構築するシステムのアーキテクチャであり，SOA に基づくシステムはビジネスプロセスの変更に対して柔軟である．多くの場合，SOA に基づくシステムの開発者はミドルウェアを用いてシステムを開発する．本研究室では On the Job Learning(以下，OJL) として SOA に基づくシステムである ATM 監視システムを開発した．ATM 監視システムは SOA に基づくシステムの構築ソリューションのひとつである Service-Oriented Reference Architecture(以下，S3)[1] をもとに開発した．Web サービスを連携する機能をアプリケーションプラットフォーム (以下，App.PF) が提供する．システムを開発中，実行効率の向上が求められた．実行効率は，単位時間と単位時間で処理可能なトランザクション数の比で，トランザクションは，ブラウザからのリクエストに対するレスポンスを返す処理単位である．

App.PF において，実行効率を向上させるために改善するべき点が不明確であった．システムの実行効率を向上させるために App.PF の改善が考えられたが，App.PF におけるアスペクトと実行効率の関係が不明確であるので，改善すべき点が不明確であった．

本研究の目的は，ATM 監視システムの App.PF におけるアスペクトと実行効率の関係を明確にし，App.PF をプロダクトライン化する．App.PF におけるアスペクトと実行効率の関係を明確にすることで，実行効率に着目してプロダクトを構築することが可能となる．

本研究では，ふたつのアイデアをもとに App.PF におけるアスペクトと実行効率の関係を明確にし，App.PF のプロダクトライン化を目指す．ひとつ目は，プロダクトラインの可変性を分析するために Views and Beyond[2] の各視点における Style と S3 の各層の関係を整理する．Views and Beyond の記述方法に基づいてアーキテクチャを記述し，S3 アーキテクチャと比較することで可変性を分析する．ふたつ目は，クオリティモデル [3] と App.PF におけるアスペクトの関係を整理する．S3 の各層と Views and Beyond の各視点における Style で実現される非機能特性を整理することで，非機能特性に着目してプロダクトを構築することが可能と考える．

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化

2.1　ATM 監視システムのアーキテクチャ

一般的な S3 のアーキテクチャだけでなく，SOA style に基づきアーキテクチャを記述する．S3 では実行効率に関係するコンポーネントの配置が表現されないので，Deployment style で表現する．

ATM 監視システムは SOA に基づいたシステムであるので，Component & Connector view type の SOA style でアーキテクチャを記述した．SOA style で識別されたコンポーネントの責務から，S3 の各層と対応づけたアーキテクチャを図 1 に示す．

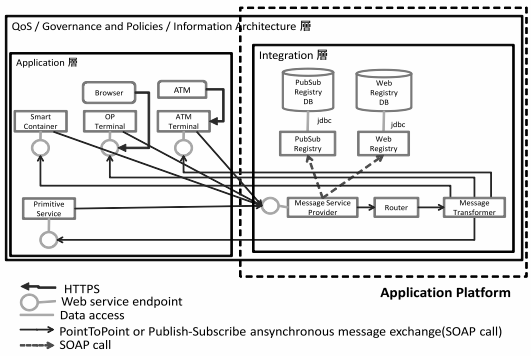

図 1　S3 と SOA style で表現した ATM 監視システムのアーキテクチャ

実行効率に影響を与える要因の一つであるコンポーネントの配置を表現する Deployment style に基づいて ATM 監視システムのアーキテクチャを記述した．記述したアーキテクチャを図 2 に示す．

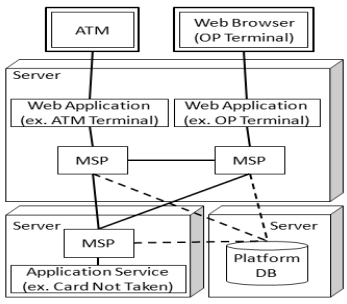

図 2　Deployment style で表現した ATM 監視システムのアーキテクチャ

2.2 プロダクトラインの可変点の導出

プロダクトラインの可変点を特定するために, ATM 監視システムを事例として, S3 の各層と Views and Beyond の各スタイルとの関係を整理する．S3 に基づくアーキテクチャ，SOA style で表現したアーキテクチャ，Deployment style で表現したアーキテクチャを比較することでプロダクトラインの可変点を導出する.

S3 のひとつの層内とスタイルの関係，S3 の複数の層とスタイルの関係を整理する．整理した関係を表 3 と表 4 に示す．表内の○は S3 における層と Views and Beyond におけるスタイルが横断関係にあることを表す.

図 3 Views and Beyond と S3 の関係 (層内)

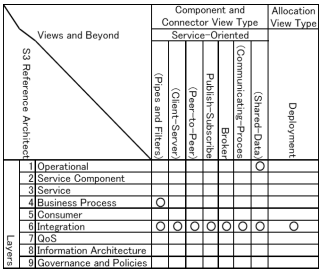

図 4 Views and Beyond と S3 の関係 (層間)

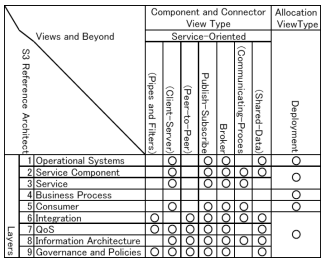

表 3 と表 4 から，アプリケーションと App.PF のインタフェースに可変性が存在することが明確となった．具体的に，ミドルウェアの Application Programming Interface が挙げられる.

2.3 非機能特性とアスペクトの関係の整理

App.PF におけるアスペクトと非機能特性の関係を整理するために，次のふたつの関係を整理する.

- Views and Beyond の各視点における Style とクオリティモデルの関係
- S3 の各層とクオリティモデルの関係

これらの関係を ATM 監視システムの事例をもとに整理し，S3 の各層と Views and Beyond の各スタイルで実現される非機能特性を明確にする.

ATM 監視システムの事例をもとに，Views and Beyond のスタイルとクオリティモデルの関係を整理する．図 1 と図 2 をもとに，各スタイルに基づくコンポーネントが実現する非機能特性を整理することで，Views and Beyond のスタイルとクオリティモデルの関係を整理した (表 1).

S3 の各層とクオリティモデルの関係を整理する．図 1 をもとに，S3 の各層における役割のコンポーネントが実現する非機能特性を整理した．整理した S3 の各層と品質特性の関係を表 2 に示す.

表 1 Views and Beyond とクオリティモデル

Views and Beyond \ Quality Model		機能性				信頼性				使用性				効率性		保守性				移植性			
		合目的性	正確性	相互運用性	セキュリティ	機能性標準適合性	成熟性	障害許容性	回復性	信頼性標準適合性	理解性	習得性	運用性	魅力性	使用性標準適合性	時間効率性	資源効率性	効率性標準適合性	解析性	変更性	安定性	試験性	保守性標準適合性	環境順応性	設置性	共存性	置換性	移植性標準適合性
Modules ViewType	Decomposition	○		○	○		○	○			×	×	×	×		○	○		○	○	○	○						
	Uses	○		○	○		○	○			×	×	×	×		○	○		○	○	○	○						
	Generalization	○		○	○		○	○			×	×	×	×		○	○		○	○	○	○		○			○	
	Layerd	○		○							×	×	×	×		○	○		○	○	○	○		○				
	Aspect	○									×	×	×	×					○	○	○	○		○				
	Data-Model	○		○							×	×	×	×					○	○	○	○						
Connector ViewType SOA-Style	(Pipes-and-Filters)	○			○		○	○	○		×	×	×	×		○	○		○	○	○	○		○				
	(Client-Server)	○									×	×	×	×		○	○			○	○	○						
	(Peer-to-Peer)	○					○	○			×	×	×	×		○	○			○	○	○						
	Publish-Subscribe	○									×	×	×	×						○	○	○						
	Broker	○		○	○		○	○			×	×	×	×		○	○			○	○	○						
	(CommunicatingProcess)	○									×	×	×	×						○	○	○						
	(Shared-Data)	○			○						×	×	×	×					○			○						
Allocation ViewType	Deployment	○	○				○				×	×	×	×		○	○		○	○								
	Install										×	×	×	×														
	Work-Assignment										×	×	×	×														

表 2 S3 とクオリティモデル

S3 Reference Architecture \ Quality Model		機能性				信頼性				使用性				効率性		保守性				移植性			
		合目的性	正確性	相互運用性	セキュリティ	機能性標準適合性	成熟性	障害許容性	回復性	信頼性標準適合性	理解性	習得性	運用性	魅力性	使用性標準適合性	時間効率性	資源効率性	効率性標準適合性	解析性	変更性	安定性	試験性	保守性標準適合性	環境順応性	設置性	共存性	置換性	移植性標準適合性
Layers	1 Operational	○	○	○	×		○	○	○		×	×	×	×		○	○		○	○	○			○	○	○	○	
	2 Service Component	○		○	×		×	×	×		×	×	×	×		○	○		×	○	○						○	
	3 Service	○		○	×		×	×	×		×	×	×	×		○	○		×	○	○						○	
	4 Business Process	○			×		×	×	×		×	×	×	×		○			×	○	○							
	5 Consumer	○			×		×	×	×		○	○	○	○		○	○		×	○	○							
	6 Integration	○		○	×		×	×	×		×	×	×	×		○	○		×	○	○						○	
	7 QoS	○	○		○		○	○	○		×	×	×	×		○			○	○	○	○						
	8 Information Architecture	○		○	×		×	×	×		×	×	×	×		○	○		×	○	○							
	9 Governance and Policies	○	○		○		○	○	○		×	×	×	×		○			×	○	○	○						

2.4 プロダクトラインの可変性の分析

実行効率に着目して，プロダクトラインの可変性を分析する．可変点の導出，非機能特性とアスペクトのふたつの節から，プロダクトラインの可変性を分析した.

可変要因をアーキテクチャ，実現技術，動作環境の 3 つの視点から分類する．アーキテクチャの視点から，コンポーネントの配置，コンポーネントの実現方法，データの管理方法が可変要因となる．コンポーネントの配置は，コンポーネントを同一サーバ上もしくは別サーバ上に配置するかである．コンポーネントの実現方法は，コンポーネントを Web サービス化もしくはメモリ上で扱うかである．データの管理方法は，データをデータベースやファイル，メモリといった管理の仕方である.．実現技術の視点から，メッセージングでは送り先の Web サービスに応じて複数のメッセージ形式を扱う．SOAP，REST，JSON，Binary といったメッセージ形式が挙げられる．動作環境の視点から，Web サービスフレームワークである Axis2 や CXF などのミドルウェアは可変要因となる.

アーキテクチャ，実現技術，動作環境の視点から，それぞれの可変要因が実行効率に関わる．各視点による可変要因とその選択肢を整理し，表 3 に示す．

表 3　可変要因の整理

視点	要因	選択肢
Architecture	コンポーネントの配置	同一サーバ / 別サーバ
	Primitive Serviceで扱うデータの管理方法	DB化 / メモリ化 / ファイル化
	コンポーネントの実現方法	WS化 / メモリ化
実現技術	メッセージ形式	SOAP / REST / JSON / Binary
動作環境	ミドルウェア	WS Framework (Axis2 / CXF)

2.5　プロダクトラインの仕様の定義

プロダクトラインの仕様を表現するフィーチャ図を定義する．フィーチャを階層的に分割して選択要因を表現することで，プロダクトラインの共通性と可変性の識別が可能となる．FORM[4] を参考に階層を分ける．

実行効率に影響を及ぼすプロダクトラインの可変点にはコンポーネントの配置が存在する．コンポーネントの配置を表すフィーチャは FORM の 4 階層では表現されないので Deployment Layer を定義する．Deployment Layer ではコンポーネントの配置についてのフィーチャを表す．実行効率に着目してプロダクトを構築することを可能とするために，実行効率に対する優劣を＋と－で表現する．＋は他の選択肢と比較して実行効率に優れることを示し，－は他の選択肢と比較して実行効率に劣ることを示す．定義したフィーチャ図を図 5 に示す．

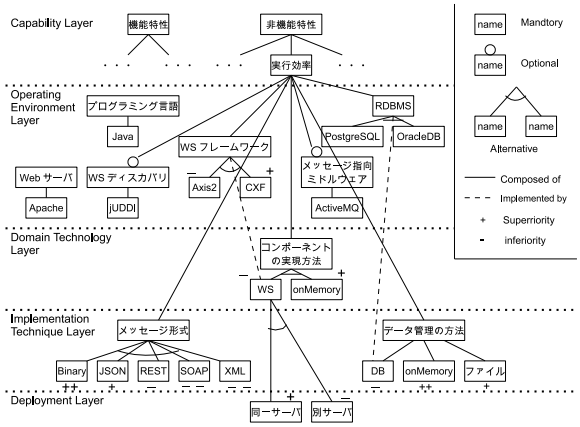

図 5　プロダクトラインの仕様モデル

2.6　プロダクトラインアーキテクチャの定義

プロダクトラインアーキテクチャはプロダクトにおいて共通して存在するアスペクトの集合として定義する．共通して存在するアスペクトとして，Location Transparency, Concurrency, System Configuration, Routing, Message Transform が挙げられる．これらのアスペクトと整理した可変要因から，App.PF のプロダクトラインアーキテクチャを図 6 に示す．

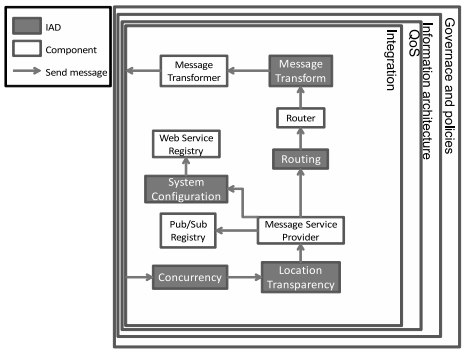

図 6　プロダクトラインアーキテクチャ

可変点として Message Service Provider(以下, MSP) のコンポーネントの実現方法が挙げられる．MSP の実現方法としては WS 化するか，メモリ上で扱うかの選択肢がある．実行効率を考慮した場合，メモリ上に MSP を配置することによって，App.PF のプロダクトアーキテクチャを構築できる．

3　事例検証

ATM 監視システムを事例として，本研究で提案するプロダクトラインから実行効率を重視するプロダクトの構築が可能か検証する．実行効率を重視する前のプロダクトの実行効率は 15tps(transaction par seconds) であり，そのプロダクトの仕様を図 7 に示す．

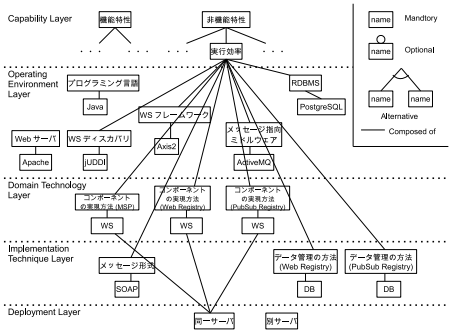

図 7　実行効率を重視する前のプロダクトの仕様

実行効率を向上させるために，コンポーネントの配置，ミドルウェアとメッセージ形式の変更を行ない，実行効率を重視したプロダクトを構築した．WS フレームワークを Axis2 から CXF に変更した．Axis2 のメッセージング形式は SOAP だが，CXF で REST を選択することで実行効率を向上させた．MSP，Web Registry, PubSub Registry を WS 化するとプロトコル変換に時間を要するので，メモリ上で扱うことで実行効率を向上させた．実行効率を重視したプロダクトの仕様を図 8 に示す．

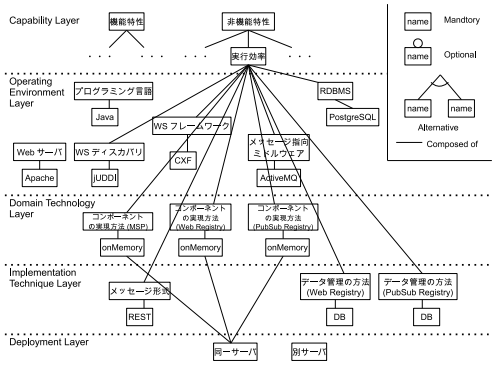

図 8 実行効率を重視したプロダクトの仕様

実行効率を重視する前のプロダクトの実行効率は 15tps で，実行効率を重視したプロダクトの実行効率は 50tps であった．実行効率に優れるプロダクトを構築できたことから，本研究で提案するプロダクトラインでは実行効率に着目したプロダクトの構築が可能であると確認した．

4 考察

4.1 可変性の分析方法の妥当性

プロダクトラインにおける可変点を特定するためのアプローチが妥当であるか考察する．本研究では，SOA Style と Deployment Style に基づき ATM 監視システムのアーキテクチャを記述し，S3 と比較することでプロダクトラインにおける可変点を特定した．

本研究における事例検証では，実行効率が異なるシステムを構築することが可能であったが，ひとつの事例検証のみでは信頼性に欠ける，可変性の分析方法が妥当かどうかを確認するためには，さらに事例検証を重ねて信頼性を向上させる必要があると考える．

4.2 プロダクトライン化の妥当性

App.PF をプロダクトライン化する妥当性について考察する．本研究では，アプリケーションの実行効率に対する要求に柔軟に対応し，アプリケーションの生産性を向上させるために App.PF に PLSE を適用した．PLSE の他に，アプリケーションの生産性を向上させる開発方法論としてモデル駆動型開発 [6] が挙げられる．PLSE とモデル駆動開発を比較することで App.PF のプロダクトライン化する妥当性について考察する．

PLSE は，製品系列において再利用可能な核資産を定義し，核資産を用いてソフトウェアを開発する方法論である．核資産を利用してプロダクトを構築することでアプリケーションの生産性の向上が期待される．核資産は，新たに構築されたシステムや既存のシステムをもとに製品系列における共通性と可変性を定義することで定義される．核資産の組み合わせによって，アプリケーションに対する要求に柔軟に対応するプロダクトの構築が可能である．

モデル駆動型開発はモデルの変換を自動化し，モデルを中心にソフトウェアを開発する方法論である．モデルの変換は CASE ツールで自動化され，CASE ツールはモデル間の関係を規定した変換規則に基づきモデルを変換する．自動化することによってモデル間の一貫性を保ち，保守性や生産性の向上が期待される．モデルとして，プラットフォーム非依存モデルとプラットフォーム依存モデルが挙げられる．実行効率はミドルウェアにより変化するので，実行効率を向上させる際にプラットフォームが変化する場合が存在する．プラットフォームが変化することで変換規則を再定義する必要があり，アプリケーションの実行効率に対する要求に柔軟に対応することができない．

アプリケーションの実行効率に対する要求に柔軟な対応が可能であるので，App.PF のプロダクトライン化は妥当であると考える．モデル駆動型開発はモデルの変換を自動化することで生産性が向上するが，プラットフォームの変化に柔軟な対応ができない．PLSE では核資産を用いて開発することでアプリケーションに対する要求に柔軟に対応し，生産性が向上する．このことから，本研究において，App.PF に PLSE を適用し，プロダクトライン化したことは妥当であると考える．

5 おわりに

本研究では App.PF をプロダクトライン化した．プロダクトラインの仕様と実行効率の関係を整理することで，実行効率に着目してプロダクトの構築が可能となった．

今後の課題として，可変性の分析方法が妥当であるか確認するために事例検証を重ねることが挙げられる．

参考文献

[1] A. Aranjani, L. Zhang, M. Ellis, A. Allam, and K. Channabasavaiah, “S3: Service-Oriented Reference Architecture,” *IEEE Computer Society*, vol. 9, no. 3, pp. 10-17, 2007.

[2] F. Bachmann, L. Bass, P. C. Clements, D. Garlan, J. Ivers, R. Little, P. Merson, R. Nord, and J. A. Stafford, *Documenting Software Architectures Views and Beyond Second Edition*, Addison-Wesley Professional, 2010.

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

[4] K. C. Kang, S. Kim, J. Lee, K. Kim, E. Shin, and M. Huh, “FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures,” *Annals of Software Engineering*, vol. 5, pp. 143-168, 1998.

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

[6] 高田広章，枝廣正人，沢田篤史，清水徹，中島達夫，平山雅之，組込みシステム概論，CQ 出版社，2008.