歯科レセプト処理システムの開発
—ミドルウェアレイヤを軸にしたプロダクトライン化—

M2010MM024 児玉優太
指導教員：野呂昌満

1　はじめに

エンタープライズ系システムに要求される最も重要な非機能特性として，ビジネス環境の変化に対応する柔軟性が挙げられる．ビジネス環境の変化に対して柔軟に変更可能なシステム開発の技術として，SOA(Service-Oriented Architecture)[1] がある．SOA に基づく Web サービス開発のために，各種ミドルウェア (.NET[6]，Axis2[7]) がベンダから提供されている．我々はミドルウェアレイヤのソフトウェアアーキテクチャはアプリケーションのソフトウェアアーキテクチャに影響を与えると考える．Web サービスの設計，実現のさいに，技術者はミドルウェアレイヤに関連する技術や，ミドルウェアが規定するアーキテクチャだけに基づいてシステムを開発しがちな傾向にある．

本研究の目的はミドルウェアに独立なシステムアーキテクチャを定義し，個々のミドルウェアを用いたアプリケーションアーキテクチャを整理することである．このさいミドルウェアレイヤに依存する横断的関心事をアスペクト指向技術を用いて整理し，アプリケーションアーキテクチャを設計する．

PLSE(Product-Line Software Engineering)[5] を適用し，個々のミドルウェアを製品系列における派生プロダクトを決定づけるものと位置づける．ミドルウェアレイヤを軸にしたプロダクトライン化を実現するために，以下のアーキテクチャ設計を行なう．

- ミドルウェアに依存しないシステムアーキテクチャ
- アスペクト指向技術を適用したアプリケーションアーキテクチャ

SOA に求められる特性を考慮し，ミドルウェアに非依存の統一的なシステムアーキテクチャ，すなわちプロダクトラインアーキテクチャを設計する．次に，個々のミドルウェアに依存する関心事をシステムアーキテクチャに横断的する関心事と捉え，アスペクト指向技術を用いる．整理した関心事をもとにプロダクトアーキテクチャとしてアプリケーションアーキテクチャを設計する．事例として.NET を用いて歯科レセプト処理システムのアプリケーションアーキテクチャを整理する．整理したアプリケーションアーキテクチャと Java 関連のミドルウェアを用いて作成したアプリケーションアーキテクチャを比較し，プロダクトライン化の有用性について考察する．

成果として，ミドルウェアに非依存のシステムアーキテクチャの定義と，異なるミドルウェア (.NET，Java) でアプリケーションアーキテクチャが定義可能である事を確認した．

2　歯科レセプト処理システム概要

歯科レセプト処理システムとは，歯科診療所における歯科診療事務を支援するシステムである．社会保険診療報酬支払基金が歯科レセプト提出業務の軽減と迅速化を目的としてレセプト電算処理システム [8] を提案している．我々は OJL(On the Job Learning) の一環として，歯科レセプト作成に関する業務を支援するシステムの開発を行なう．歯科レセプト処理システムが実現する機能は，社会保険診療報酬支払基金が公布する診療報酬請求の仕様に準拠している．図 1 に開発するシステム全体の概要と歯科レセプト処理システムの関係を示す．我々は，電子レセプト提出に関する歯科診療業務 (レセプトチェック，保険点数算定，電子レセプト作成) の開発を行う．

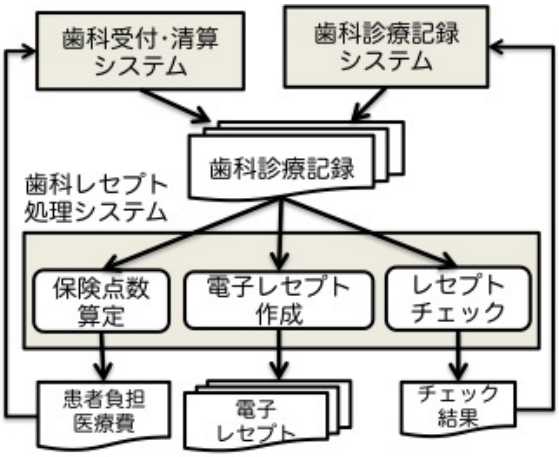

図 1　歯科レセプト処理システムの概要

3　システムアーキテクチャ設計

Web サービスのシステムアーキテクチャは，ビジネス環境の変化に伴うシステムの拡大に対するスケーラビリティや，サービスの物理的な位置に依存しない位置透過性等を実現する必要があると考える．我々はこれらの Web サービスに求められる非機能特性を実現する技術を組み合わせ，統一的なシステムアーキテクチャを設計した．設計したシステムアーキテクチャを図 2 に示す．

位置透過性の実現のためにサービスブローカ [1] を適用し，Web サービスと Web アプリケーションに対するリクエストブローカとレジストリをそれぞれ作成する．リクエストブローカを仲介することで，クライアントやサービスの物理的な位置を意識することなく Web サービスが利用可能になる．Web アプリケーションと Web サービスの独立した動作を実現するために，Concurrent リクエストブローカを作成し，リクエストブローカを並行に稼働

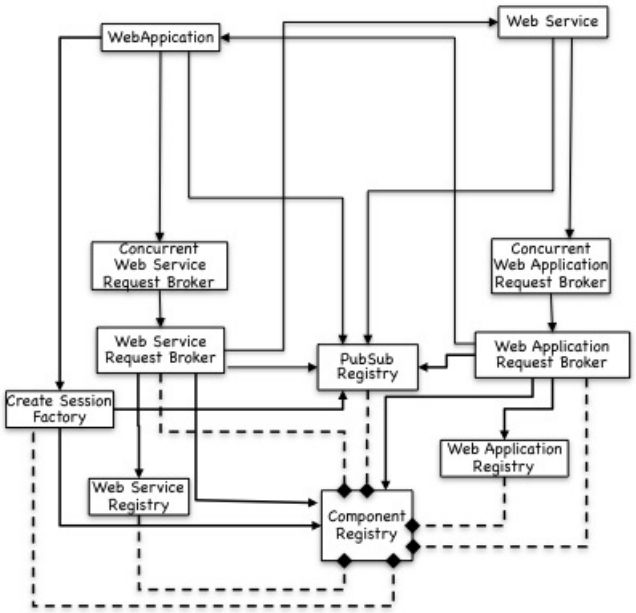

図 2 システムアーキテクチャ

する．Publish/Subscribe[3] を適用し，クライアントとプロバイダの動的な 1 対多の通信を実現する．連携するクライアントとサービスの組を PubSub レジストリに登録し，リクエストブローカが送信先を PubSub レジストリから取得し，送信する．クライアントとプロバイダの動的な 1 対多の通信により，スケーラビリティを実現している．PubSub レジストリに登録するトピックにセッション状態を保持し，セッション状態を PubSub レジストリから取得することでサービスのステートレス化が実現される．加えて，これらレジストリの動的配置が可能なように Component レジストリにそれらの情報をカプセル化し，登録する．

4 アスペクト指向技術を適用したアプリケーションアーキテクチャ

4.1 .NET 関連の実現技術の概要

.NET[6] とは，Windows 系のソフトウェア開発における標準の技術である．我々は，.NET を Web サービス開発の代表的な技術の一つとして取り上げ，ミドルウェアの事例に用いる．実際に，OJL の提携企業から歯科レセプト処理システムの開発に.NET 環境の使用を要求されている．使用する.NET 関連の開発環境を表 1 に示す．

表 1 .NET の開発環境

言語	C#
Web サーバ	IIS (InternetInformationServer)
Web サービス	WCF
Web アプリケーション	Silverlight

Silverlight と WCF の関係を図 3 に示す．WCF を用いて作成する Web サービスはエンドポイント (Web サービスのアドレス，通信プロトコル，サービスのインタフェース) を定義する．エンドポイントを明確化し，Web アプリケーションは Web サービスのプロキシを通じて通信する．

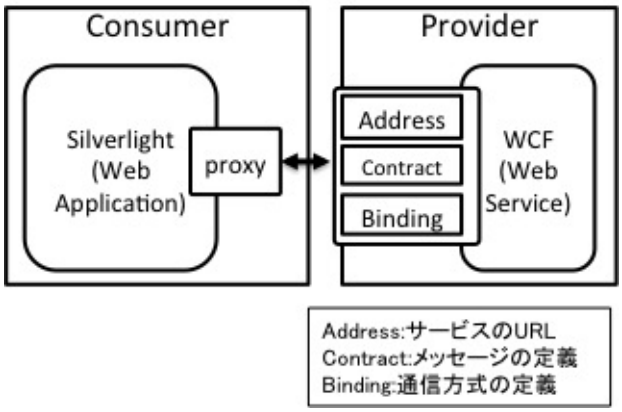

図 3 Silverlight と WCF の関係

4.2 アスペクト指向技術の適用

.NET におけるサービスブローカのレジストリを実現する技術に WCF-Discovery がある．WCF-Discovery は WCF を用いて作成した Web サービスの登録，検索を提供している．しかし，Silverlight における Web システムは図 4 に示すように Silverlight のアプリケーションフレームワークと，WCF のフレームワークが一体となって使用することを前提として提供している．

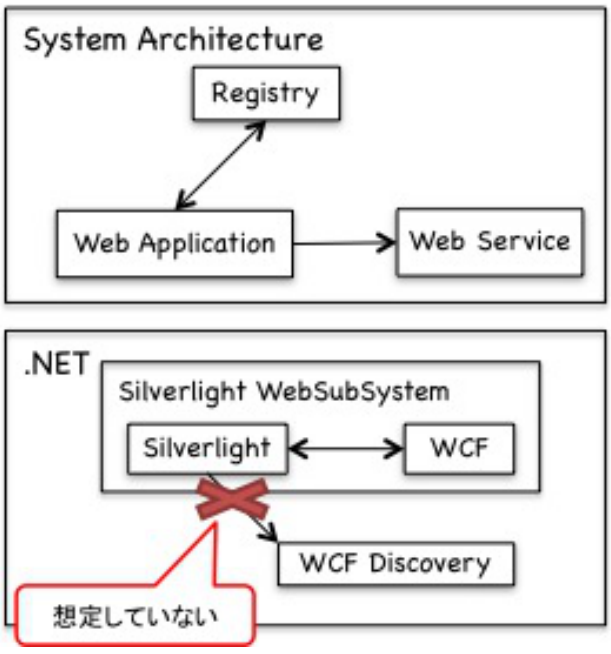

図 4 Silverlight と WCF が前提とする構造

Silverlight と WCF のミドルウェアでは，レジストリを適用した構造は想定していない構造となる．我々は Silverlight と WCF のミドルウェアの構造がシステムアーキテクチャに横断する関心事になるとの認識に立ち，アス

ペクト指向を適用してアプリケーションアーキテクチャを定義する．

ミドルウェアに依存する横断的関心事をアーキテクチャ上で表現するためには，特定の言語に依存しないアスペクトの表現方法が必要であると考える．GoF デザインパターン [2] は，特定の言語に依存しない詳細アーキテクチャ設計の解をパターンとしてまとめたものである．ミドルウェアに依存する関心事を解決するパターンを適用することで，ミドルウェア毎にアプリケーションアーキテクチャが定義可能という認識から，我々は GoF デザインパターンを用いる．

我々は既存のオブジェクトに変更を加えることなく異なるインタフェースを変換するパターンである Adapter パターンを適用する．レジストリのアクセスを行なう WCF を，レジストリへのアダプタと捉える．図 5 に Adapter パターンを適用した Web サービスレジストリの静的構造を示す．リクエストブローカはアダプタに検索を行うことで，WCF-Discovery のレジストリと通信が可能になる．

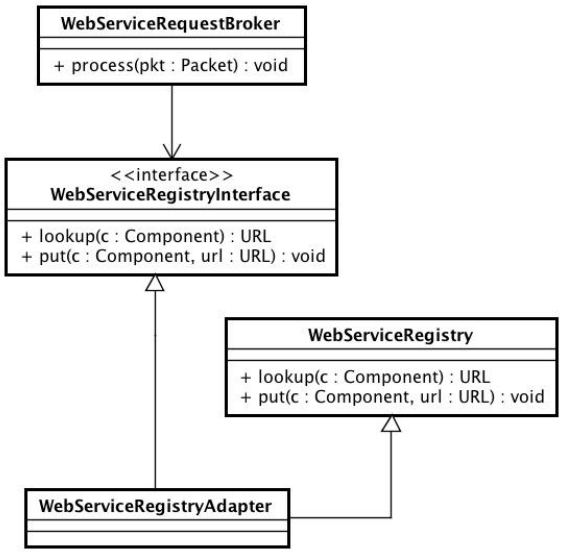

図 5 Adapter パターンの適用

4.3 アスペクト指向適用後のアプリケーションアーキテクチャ

図 6 に Adapter パターンを適用した.NET のミドルウェアを用いたアプリケーションアーキテクチャを示す．黄色の枠で囲まれたコンポーネント郡は，.NET のミドルウェアに依存する関心事をアスペクトとして整理した箇所である．

整理したアプリケーションアーキテクチャに基づき，表 1 の開発環境で歯科レセプト処理システムの Web サービスを作成し，正常に稼働することを確認できた．よって我々が設計したシステムアーキテクチャに基づき，.NET のミドルウェアに依存するアプリケーションアーキテクチャをアスペクト指向を適用して整理することが可能であることを確認した．

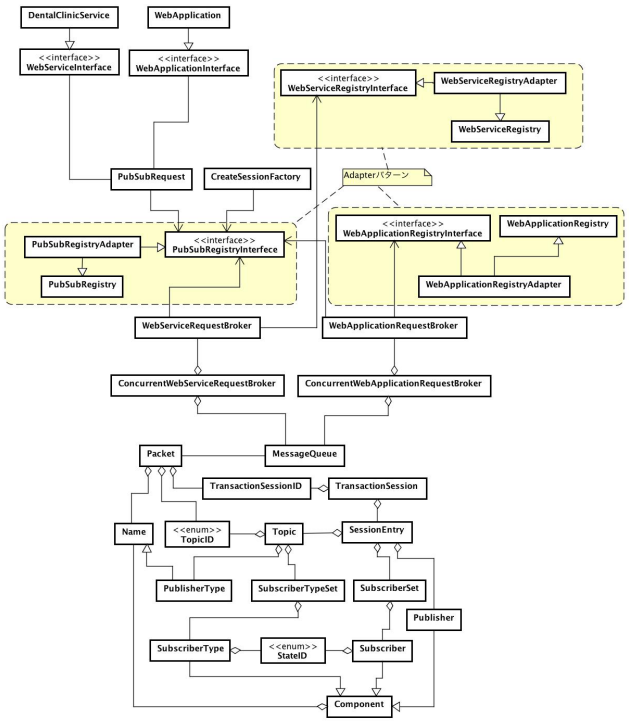

図 6 アプリケーションアーキテクチャ

5 考察

5.1 アプリケーションアーキテクチャ設計に用いたパターンの妥当性

GoF デザインパターンを用いて行った個々のミドルウェアのアプリケーションアーキテクチャ設計について，代替案との比較を通じてその妥当性について考察する．.NET のミドルウェアにおけるアプリケーションアーキテクチャに依存する横断的関心事を GoF デザインパターンを適用して整理した．GoF デザインパターンの他のパターンについて，アプリケーションアーキテクチャの実現可能性の観点で比較，検討する．本研究は Adapter パターンを適用して，構造が異なるレジストリとの通信を可能にした．レジストリへの通信の Adapter を作成することで，アプリケーションアーキテクチャに存在する横断的関心事を適切に表現することが可能になった．.NET における横断的関心事を表現する他のパターンの選択肢として，構造に関するパターンの Proxy パターン，Decorator パターンの適用が考えられる．各パターンを適用した場合の比較を表 2 に示す．

表 2 パターンの比較

デザインパターン	実現可能性
Adapter パターン	○
Proxy パターン	△
Decorator パターン	△

Proxy パターンの場合，レジストリへの通信を代理として行うコンポーネントを Proxy として作成することで，

レジストリへのアクセスを行う．Proxy パターンを適用した場合，Proxy とアクセスするレジストリは同じインタフェースである必要がある．しかし，Silverlight を用いた Web サービスの構造は，レジストリへのアクセスは想定しておらず，インターフェースが同様となるとはいいがたい．異なるオブジェクトを適合することで，レジストリへの通信を実現可能にできる点で，Adapter パターンが実現可能性として適切であると考える．

Decorator パターンの場合，Decorator クラスのサブクラスに異なる責任のオブジェクトを作成することで，構造の動的な変更が可能になる．.NET 関連のミドルウェアに Decorator パターンを適用する場合，Decorator のサブクラスにレジストリへの通信を行なうオブジェクトを作成することで実現が可能であると考える．しかし，レジストリへの通信は常に WCF を仲介することから，動的なオブジェクトの変更はないと考える．よって，パターンを比較，検討した結果，適用したパターンとして Adapter パターンの選択は妥当であったと考える．

5.2 ミドルウェアを軸としたプロダクトライン化

プロダクトラインアーキテクチャに基づいて異なるミドルウェアを用いてプロダクトアーキテクチャが作成可能であるかを異なるミドルウェアで検証し，ミドルウェアを軸としたプロダクトライン化の有用性について考察する．異なるミドルウェアとして，Web サービスのミドルウェアの代表である Java 関連のミドルウェアを対象にアプリケーションアーキテクチャを整理し，.NET のミドルウェアのアプリケーションアーキテクチャと比較する．アプリケーションアーキテクチャ設計に用いる Java の開発環境は Java 関連のミドルウェアで一般に利用されるサーブレットと Axis2 を用いた．

図 7 にプロダクトラインアーキテクチャに基づく.NET 関連と Java 関連のプロダクトアーキテクチャを示す．.NET のミドルウェアにおけるプロダクトアーキテクチャでは，レジストリとの通信に Adapter パターンを適用した．レジストリへの通信が想定されていない構造にアスペクト指向技術を適用したことで，システムアーキテクチャに基づくプロダクトアーキテクチャの作成が可能であった．一方，Java 関連のサーブレットと Axis2 を用いたプロダクトアーキテクチャでは，図 7 に示すように Web アプリケーションであるサーブレットからレジストリへ直接アクセスが可能であった．これら 2 つのミドルウェアを用いて作成したプロダクトアーキテクチャは，プロダクトラインアーキテクチャとして定義したシステムアーキテクチャと異なることなく作成することができたといえる．よって，我々が設計したプロダクトラインアーキテクチャであるシステムアーキテクチャからミドルウェア毎にプロダクトアーキテクチャが作成可能であると考える．

6 おわりに

本研究は PLSE の考えに基づき，ミドルウェアに独立なシステムアーキテクチャを定義し，ミドルウェアに固有な技術を用いたアプリケーションアーキテクチャをアスペクト指向技術を利用して整理した．.NET のミドルウェアを事例としてアプリケーションアーキテクチャを定義した．Java 関連のミドルウェアのアプリケーションアーキテクチャが同様に定義できることを確認し，ミドルウェアを軸とするプロダクトライン化の有用性を確認した．今後の課題として，本研究で整理したアーキテクチャに基づき，ミドルウェアに独立な Web サービス開発体系の整理が挙げられる．

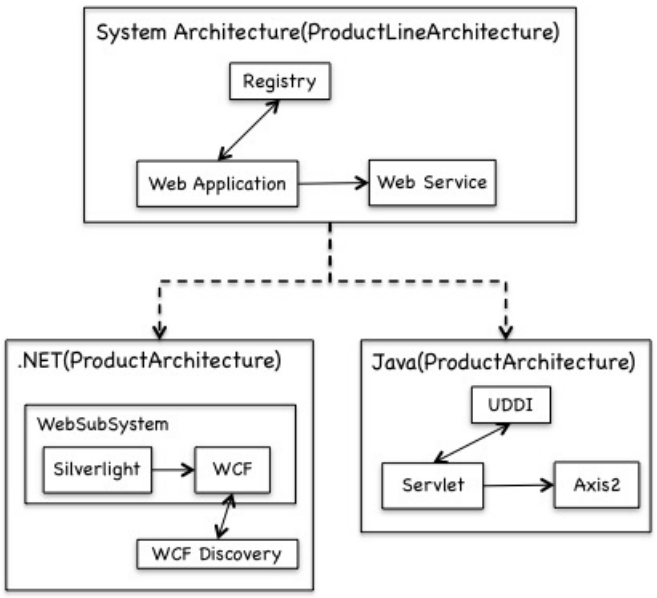

図 7 .NET 関連と Java 関連のプロダクトアーキテクチャ

参考文献

[1] D. Georgakopoulos, and M. Papazoglou, *Service-Oriented Computing,* The MIT Press, 2009.

[2] E. Gamma, J. Vlissides, R. Helm, and R. Johnson, *Design Pattern Elements of Reusable ObjectOriented Software,* Addison-Wesley, 1995.

[3] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern-Oriented Software Architecture,* Wiley, 1996.

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

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

[6] Microsoft, “.NET ,” http://www.microsoft.com/ja-jp/net/, 2012.

[7] The Apache Software Foundation, “Axis2 ,” http://axis.apache.org/axis2/java/core/, 2004.

[8] 社会保険診療報酬支払基金，“レセプト電算処理システム，”http://www.ssk.or.jp/index.html，2010.