マルチプラットフォームに対応した同期処理の実現に関する研究

M2011MM014 服部裕介
指導教員：野呂昌満

1 はじめに

SOA に基づくシステムの開発では，並行プログラミングを扱っており，並行処理を考慮することは重要である．我々の研究室では，Zhang らの論文 [1] で提案されている S3 アーキテクチャに基づき，アプリケーションプラットフォームを設計と試作をおこなっている．試作には，java とそれに関連する技術を用いている．アプリケーションプラットフォームにおいても，並行処理は SOA において重要な関心事のひとつとして位置づけられている．

関連研究 [7] では，SOA に基づくシステム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をおこなっている．アプリケーションプラットフォームのアーキテクチャを分析し，共通箇所，変動個所を明確にすることで仕様モデルを定義している．関連研究において，共通個所，変動個所を明確にする際，S3 だけでは表現できていない関心事が存在することを確認している．これは，開発経験からコンポーネントの配置がアプリケーションの実行効率に影響を与えることを確認したからである．そこで，S3 が定義する関心事と Views and Beyond が定義する関心事が横断するという仮説を立てた．Views and Beyond の SOAStyle は，SOA に基づくシステムのアーキテクチャによく現れる構造を一般化したものであり，SOAStyle を基にアーキテクチャは構築される．すなわち，SOAStyle による支配的分割に対して，S3 の関心事が横断すると考えられる．この仮説を基に，S3 と Views and Beyond の関係を整理した．Views and Beyond はアーキテクチャ記述方法であるが，各視点ごとに Style を定義しており，アーキテクチャは Style を基にして構築される．すなわち，Views and Beyond はアーキテクチャを構築するためのリファレンスアーキテクチャとして考えることができるので，S3 を Views and Beyond を比較できると考えられる．プログラミング言語を選択すると，使用可能な SOA システムを開発するためのミドルウェア (Axis2[6], Silverlight[2]) やライブラリなどが制限される．技術者はプログラミング言語が規定するライブラリを用いて，並行処理を実現するコードを記述する必要がある．しかし，そのコードがデットロックやライブロックなどのシステムの故障の原因となる可能性がある．

本研究の目的は，SOA に基づくシステムのアプリケーションプラットフォームにおける，並行処理のマルチプラットフォーム化の実現である．SOA を構築する際に，各関心事を実現するアーキテクチャを決定する要因となるプラットフォームと，それによって変動する箇所を明確にし，関心事を構成するコンポーネントを再利用可能な部品として定義する．これにより，プラットフォームの変動に対して，コンポーネントの入れ替えによって対処することが可能となる．

本研究では，SOA が実現すべき関心事のひとつである並行処理に焦点を当て，並行処理のアーキテクチャの仕様モデルを定義する．関連研究で明確にされた共通個所，可変個所を基に，並行処理のアーキテクチャの特徴を決定づけるプラットフォームに対する要素を定義する．ATM 監視システムのアプリケーションプラットフォームのアーキテクチャから並行処理，同期処理が必要なコンポーネントの特定をおこなう．プラットフォームの変動による並行処理の実現方法の違いに対処するために，各プラットフォームに共通のインタフェースを定義する．これらによって，並行処理のマルチプラットフォーム化への対処が実現可能と考えた．また，本研究のアプローチの妥当性を確認するために，OJL(On the Job Learning) として開発に取り組んでいる医療情報システムを事例として取り上げた．

2 背景技術

2.1 S3 アーキテクチャ

S3 アーキテクチャは SOA に基づくシステムのための一般的なリファレンスアーキテクチャとして Zhang らの論文で提案されている．S3 アーキテクチャは Consumer, Business Process, Service, Service Components, Operational System の 5 層からなる Application 層と，Integration, Quality of Service, Information Architecture, Governance and Policies の 4 層からなる Application Platform 層で構成されている．Application Platform 層は Application 層すべてに横断している．本研究では，Application Platform 層を扱う．S3 アーキテクチャを図 1 に示す．

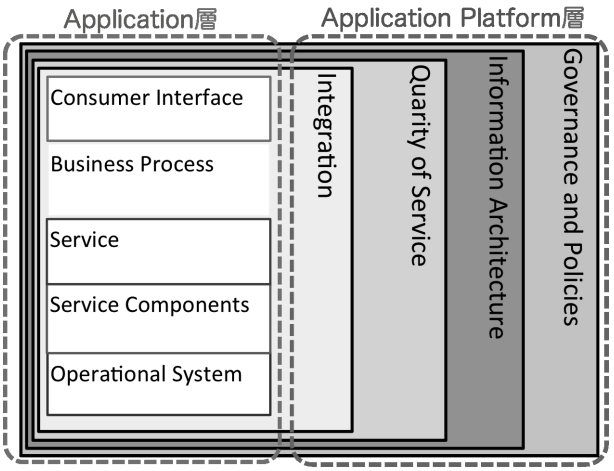

図 1 S3 アーキテクチャ

2.2 Views and Beyond

一般的なアーキテクチャ記述方法として Views and Beyond[5] がある．Views and Beyond は Module View,

Component and Connector View，Allocation View の 3 つの視点からアーキテクチャの記述方法を定義している．また，各視点において，アーキテクチャ記述の際によく現れる構造を一般化し，Style として定義している．アーキテクチャは Style を基に構築されるとの認識から，S3 と Views and Beyond を比較できると考える．

3　並行処理のマルチプラットフォーム化

3.1　マルチプラットフォーム化へのアプローチ

並行処理のマルチプラットフォーム化の実現へのアプローチを図 2 に示す．並行処理をマルチプラットフォーム化するために，並行処理のアーキテクチャを決定する仕様モデルを定義する．関連研究で明確にされたアプリケーションプラットフォームのアーキテクチャを決定づける変動要因を基に，具体的な要素を定義する．次に，各プラットフォームに共通のインタフェースを定義する．これらを基に並行処理のアーキテクチャを構築することでマルチプラットフォーム化を実現する．

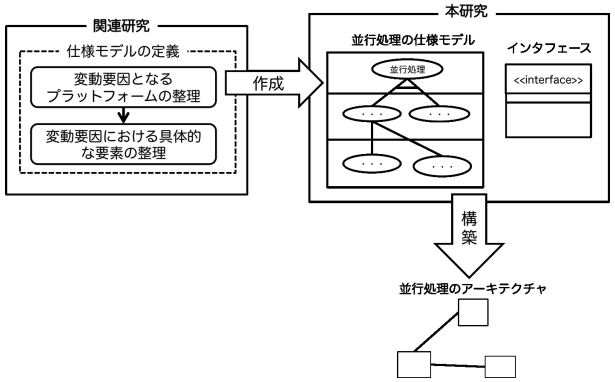

図 2　アプローチの概要

3.2　並行処理のアーキテクチャの仕様モデル

並行処理のアーキテクチャを特徴づける変動要因に対する要素を定義し，仕様モデルを定義する．関連研究で明確にされたアプリケーションプラットフォームのアーキテクチャを決定する変動要因を基に定義をおこなう．関連研究において，変動要因を明確にする際，S3 だけでは表現できていない関心事が存在することを確認した．それは，開発経験からコンポーネントの配置が実行効率に影響することを確認したからである．そこで，S3 が定義する関心事と Views and Beyond が定義する関心事が横断すると仮説を立て， 横断的関心事の整理をおこなう．Views and Beyond の SOAStyle は SOA に基づくシステムのアーキテクチャによく現れる構造を一般化したものであり，この Style を基にアーキテクチャは構築される．つまり，SOA に基づくシステムの支配的分割は SOAStyle であると考えられ，S3 が定義する関心事が横断すると考えられる．また，Views and Beyond はアーキテクチャ記述方法であるが，構造，振る舞い，配置の各視点で定義されている Style を基にアーキテクチャは構築されるのでリファレンスアーキテクチャとして考えられる．したがって，S3 と Views and Beyond を比較できると考えられる．関連研究で S3 と Views and Beyond を比較し，明確にした変動要因を以下に示す．

- ミドルウェア
- メッセージ形式
- プログラミング言語
- ライブラリや言語仕様
- コンポーネントの配置

変動要因に対して，並行処理の実現方法となる要素を定義し，並行処理のアーキテクチャの仕様モデルを定義した．定義した仕様モデルを図 3 に示す．

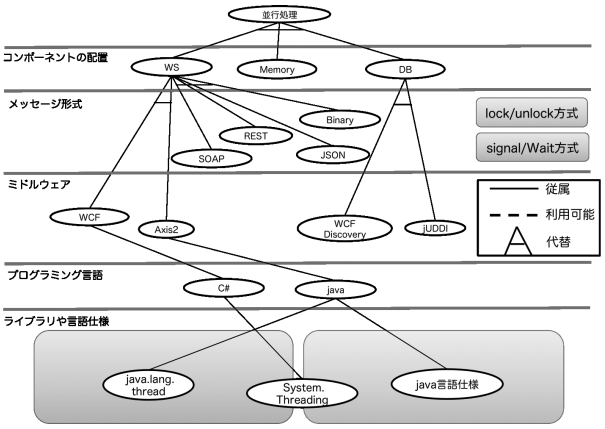

図 3　並行処理のアーキテクチャの仕様モデル

3.3　インタフェースの定義

各プラットフォームに対して共通のインタフェースとなるコンポーネントの定義をおこなう．定義したインタフェースに従って，並行処理を実現するコンポーネントを実装することで，他のコンポーネントに影響を与えずに，ライブラリの実現方法の違いへの対処が可能となる．プラットフォームの変動によって影響を受けるコンポーネントを，ATM 監視システムを事例として特定した．以下に示すコンポーネントは，プラットフォームを並行動作させるための Thread，共有資源に対する排他制御をおこなうための Monitor を用いることで並行処理を実現している．

- Message Service Provider
  - サービス呼び出しの仲介をおこなうコンポーネント
- PubSub Registry
  - PubSub メッセージングのための情報を管理するコンポーネント
- Web Registry
  - WebService の名前と URL を関連づけて管理するコンポーネント
- Component Registry

– プラットフォームの構成要素のなめと URL を管理するコンポーネント

- SessionDB
  - セッション情報を保持する DB

上記の並行処理を実現するコンポーネントにおいて，ライブラリや言語機能に依存する処理は以下の 2 点である．

- プロセスの同期のための Signal 操作と Wait 操作
- 共有資源に対する排他制御のための lock 操作と unlock 操作

1 点目に関しては，Signal/Wait を定義したインタフェースを実現する．2 点目に関しては，lock/unlock を定義したインタフェースを実現する．図 4 に定義したインタフェースを示す．定義したインタフェースに従って，並行処理を実現するコンポーネントを実装する．

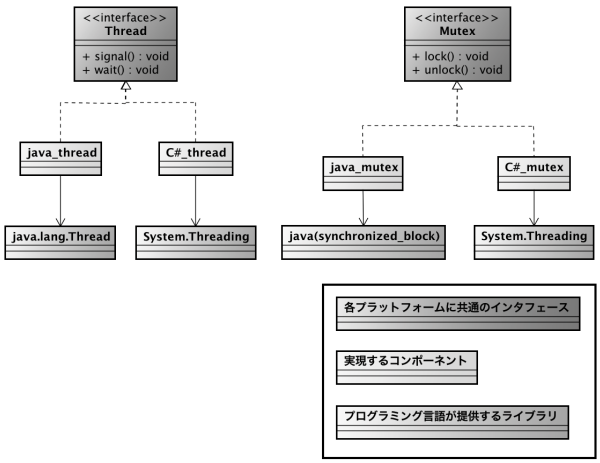

図 4 signal/wait，lock/unlock を定義したインタフェース

定義したインタフェースに対して，各プログラミング言語が提供するライブラリの並行処理の実現方法の整理をおこなう．実現方法を整理することで，各プログラミング言語による違いに対処する．整理した実現方法を表 1，表 2 に示す．

表 1 Signal / Wait の実現方法の違い

ライブラリ ＼ Thread への操作	signal	wait
System.Threading	Monitor.Pulse	Monitor.Wait
java.lang.Thread	notify	wait

表 2 lock / unlock の実現方法の違い

ライブラリ ＼ ロック機能	lock	unlock
System.Threading	Monitor.Enter	Monitor.Exit
java.lang.Thread	synchronized ブロック	synchronized ブロック

4 医療情報システムの並行処理の実現

4.1 医療情報システム概要

医療情報システムとは，歯科診療所の歯科診療請求事務をサポートするシステムである．医療情報システムが実現する機能は患者受付・精算，診療記録入力，レセプトチェック，保険点数算定，電子レセプト作成である．医療情報システムは WA フレームワークに Silverlight，WS フレームワークに WCF(Windows Communication Foundation)[3]，サービスの発見に WCF-Discovery[4]，サービスが稼働するサーバに IIS(Internet Information Services) を用いている．図 5 に医療情報システムの概要を示す．

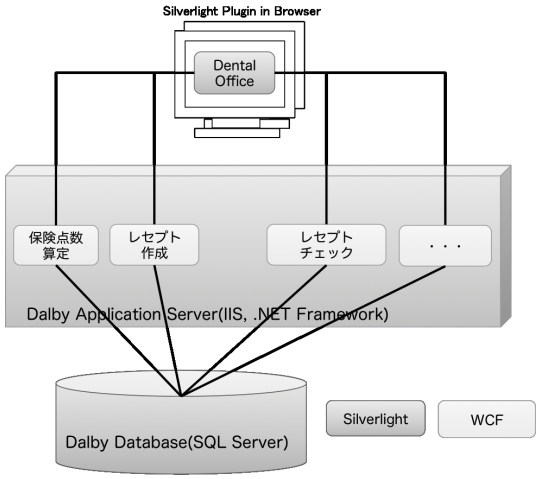

図 5 医療情報システムの概要

4.2 並行処理の実現

医療情報システムにおける並行処理を実現する．医療情報システムの並行処理アーキテクチャを決定するために，3.2 節で定義した仕様モデルからプラットフォームを選択する．プラットフォームの選択をおこない，決定された並行処理アーキテクチャの仕様モデルを図 6 に示す．

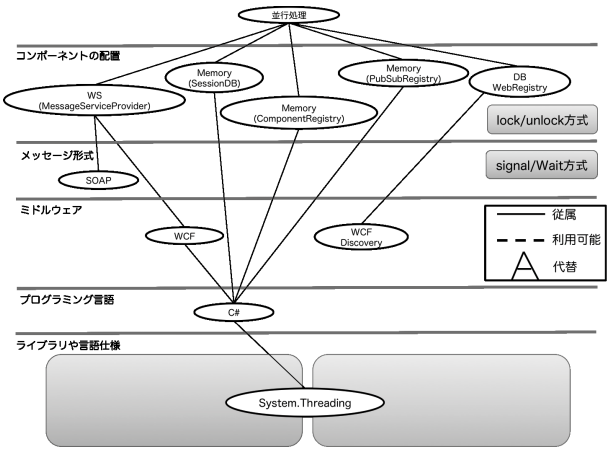

図 6 医療情報システムの並行処理アーキテクチャの仕様モデル

医療情報システムで用いている技術を分析し，プラットフォームを選択した．コンポーネントの配置において，MessageServiceProvider は WebService を選択し．SessionDB，ComponentRegistry，PubSubRegistry は On-Memory，WebRegistry は Database を選択した．またメッセージ形式において SOAP，ミドルウェアにおいて WebService フレームワークには WCF，サービスの発見には WCF-Discovery，プログラミング言語には C#を選択した．

仕様モデルの決定によって，並行処理のアーキテクチャを定義した．その結果を図 7 に示す．3.3 節で定義したインタフェースに従い，プラットフォームの選択によって決定されたライブラリを用いてコンポーネントを実装することで，並行処理が実現可能となる．

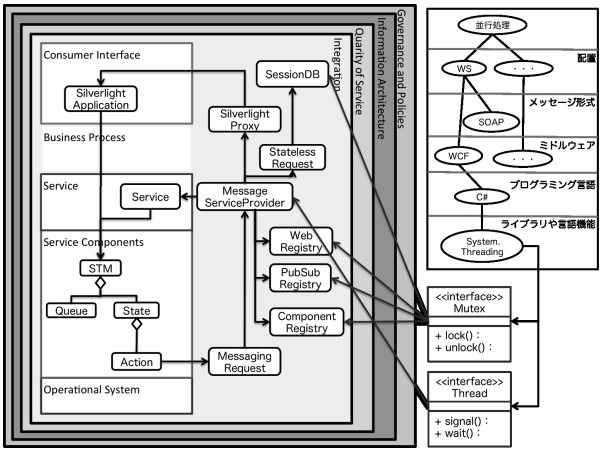

図 7 医療情報システムの並行処理アーキテクチャの実現

5 考察

5.1 マルチプラットフォーム化への対応の実現に関する考察

並行処理のアーキテクチャを特徴づけるプラットフォームを整理し，各プラットフォームに共通のインタフェースを定義したことによって，マルチプラットフォーム化が実現可能か考察する．図 8 に，並行処理のアーキテクチャのマルチプラットフォームへの対応の実現を示す．並行処理のアーキテクチャを特徴づけるプラットフォームを，コンポーネントの配置，メッセージ形式，ミドルウェア，プログラミング言語，ライブラリの 5 層の階層構造で整理した．signal/Wait 方式と lock/unlock 方式を用いてプラットフォームに共通のインタフェースを定義した．プラットフォームの選択によって決定されたライブラリを用いてインタフェースを実現することでマルチプラットフォーム化が実現可能であると考える．

6 おわりに

本研究では，並行処理のアーキテクチャを決定づける変動要因に対して，具体的な要素を整理した．アプリケーションプラットフォームを設計する際に，考慮すべき関心事のひとつである並行処理を実現する部分の仕様モデルの定義をおこなった．アプリケーションプラットフォームにおける並行処理を実現するコンポーネントの特定と，各プラットフォームに共通なインタフェースの定義をおこなった．最後にプラットフォームの変動に対し，コンポーネントの入れ替えによって柔軟に対応できるか考察し，本研究の有用性を確認した．これらによって，マルチプラットフォーム化を実現することで，プラットフォームの変動によるアプリケーションプラットフォームの再設計にかかるコストを削減できたと考える．今後の課題として，並行処理のアーキテクチャからコード自動生成の実現が考えられる．

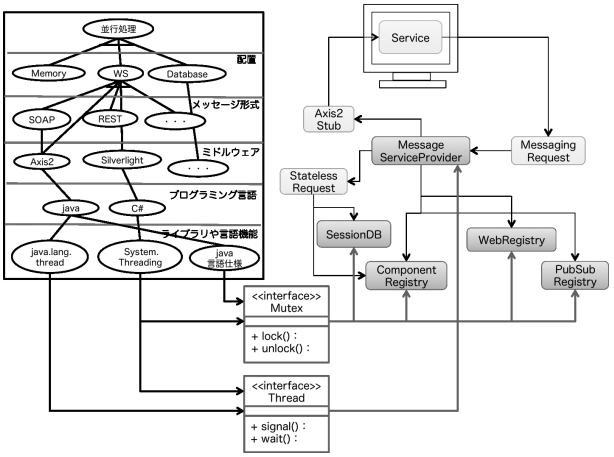

図 8 マルチプラットフォーム化の実現

参考文献

[1] A. Arsanjani, L. Zhang, M. Ellis, A. Allam, and K. Channabasavaiah, “S3: A Service-Oriented Reference Architecture, ” IT Pro, pp.10-17, 2007.

[2] Microsoft, “Silverlight, ” http://www.microsoft.com/ja-jp/silverlight/, 2013.

[3] Microsoft, “WCF, ” http://msdn.microsoft.com/ja-jp/library/, 2013.

[4] Microsoft, “ WCF-Discovery, ” http://msdn.microsoft.com/en-us/library/vstudio/dd456782(v=vs.100), 2013.

[5] P. Clements, F. Bachmann, L. Bass, D.Garlan, J. Ivers, R. Little, P. Merson, R. Nord, and J. Stafford, Documenting Software Architectures Views and Beyond Second Eddition, Assion Wesley, 20.

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

[7] 江坂篤侍, “ 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, ”南山大学大学院 数理情報研究科 数理情報専攻.