レガシーシステムへのSOAの適用に関する研究
– ラッパーの設計方法の提案 –

2007MI085 岩脇 弘晃　　2007MI231 高津 幸治
指導教員　張 漢明

1　はじめに

技術の進歩によってソフトウェアへの要求が多様化し，企業のシステムにも多様性が求められるようになった．企業の持つシステムは柔軟性を失ったレガシーシステムになっていることが多い．企業の持つレガシーシステムは，保守，運用の繰り返しによって企業の業務に最適化されている．企業の持つレガシーシステムの柔軟性を確保することによる再利用が求められている．企業の持つレガシーシステムの内部構造は複雑化していることが多い．内部構造が複雑なシステムの変更には，システムの大規模な変更が必要である．レガシーシステムへのサービス指向アーキテクチャ (SOA) の適用による，内部の実装を隠蔽した再利用が注目されている．レガシーシステムに変更を加えずにラッピングによって SOA に移行するための研究が進められている．既存の研究において，サービス抽出の手法 [4] とサービス化のアーキテクチャ [3] が提案されている．

既存の研究で提案されている手法では，レガシーシステムのサービス化が困難である．既存の研究では，サービス化で考慮すべき問題をシステム内部に変更を加えずに解決する方法が考慮されていない．サービスの柔軟性の確保には，セキュリティ，疎結合，粗粒度，ステートレス化の実現が重要である [3]．既存のサービス抽出手法では，内部に状態の管理を隠蔽して粒度の大きいサービスを抽出する．既存のサービス化のアーキテクチャは，サービスのステートレス化においてインタフェースに適合するようにサービスの内部の変更が必要である．サービスのステートレス化が実現できていない場合，ユーザごとの状態を保持することができず，サービスの要件を満たすことができない．本研究ではサービスのステートレス化に着目する．

本研究の目的は，ラッパーを用いてレガシーシステムの内部に変更を加えずにステートレス化を実現するアーキテクチャを提案することである．本研究はアスペクト指向技術を用いて，サービスの内部に変更を加えずに状態の管理を分離する．サービスのステートレス化が困難な場合，サービスのクライアント側のアプリケーションにユーザの状態を保持する．アスペクトによってユーザの状態を記録し，復元する．

本研究では既存のサービス化のアーキテクチャにアスペクト指向技術を適用し，サービスのステートレス化を実現するためのアーキテクチャを提案した．研究は以下の手順に行なった．柔軟性の確保に必要なサービスの特徴について既存の手法を考察し，識別した問題点を解決する方法を提示した．事例検証によって，アーキテクチャが実現可能であることを確認した．本研究はアスペクトによってサービスから状態の管理を分離し，サービスの利用を効率化するためのアーキテクチャを提案した．

2　背景技術

2.1　レガシーシステム

レガシーシステムは，過去の技術で構築されたシステムである [3]．企業の持つレガシーシステムは長年に渡って保守，運用が繰り返されていることが多い．レガシーシステムは企業の特定の業務に最適化され，企業の重要な情報資産になっている場合が多い．

企業の持つレガシーシステムは保守，運用の繰り返しによって内部構造が複雑化し，変更を加えることが困難になっている．レガシーシステムの拡張には，システムの大規模な変更が必要になる．レガシーシステムの開発者は異動していることが多く，レガシーシステムの詳細を把握している技術者が少なくなっている．以上の理由によって，レガシーシステムの大規模な変更は困難になっている [1]．レガシーシステムは内部構造の複雑化によって，ソフトウェア技術の進歩による要求の多様化に柔軟に対応することが困難になっている．

2.2　SOA

SOA はシステムへの要求の変化に柔軟に対応することができるシステムを開発できるアーキテクチャとして注目されている．企業の持つシステムはシステムへの要求の変化に柔軟に対応できることが求められ，SOA を適用することによる柔軟性の確保が注目されている．

SOA におけるサービスには，以下の 3 つの特徴がある [4]．

- 自己完結
- 粗粒度
- 標準化されたオープンなインタフェース

技術の進歩によってソフトウェアへの要求が多様化し，企業のシステムにも多様性が求められている．多様化した要求に柔軟に対応可能なシステムを開発できるアーキテクチャとして SOA が注目され，企業のシステムに SOA を適用する研究が進められている．

2.3　アスペクト指向技術

アスペクト指向技術は，互いに関連する複数の関心事を分離するための技術である．アスペクト指向技術の目的は，異なるソフトウェアコンポーネントに横断するソフトウェアの関心事をカプセル化することである [2]．アスペクト指向システム開発は，ソフトウェアの関心事

の相互作用を分析，実装することによって実現される．コードの織り込みによって，関心事は他の関心事から明確に分離される．

2.4 レガシーマイグレーション

レガシーマイグレーションとは，レガシーシステムをオープン環境上のシステムへと移行することである．レガシーマイグレーションにはリビルド，リライト，リホスト，ラッピングの 4 つの手法がある．

レガシーシステムを SOA に移行する場合，レガシーシステムに変更を加えずに再利用することが求められている．レガシーシステムは長年の保守，運用によって，企業の業務に最適化された重要な情報資産となっている．変更が困難になっているレガシーシステムに変更を加えずに再利用するための手法が求められている．

ラッピングは 4 つの手法の中で最も迅速で安価にレガシーシステムの SOA への移行を実現する．ラッピングは，パラダイムの違いを吸収するラッパーを実装することによって SOA への移行を実現する．レガシーシステムを最大限再利用する移行の手法としてラッピングが注目されている．

3 関連技術

3.1 既存のサービス化のアーキテクチャ

既存の研究 [3] において，レガシーシステムをサービス化するためのアーキテクチャが提案されている．**図 1** は，既存の研究で提案されているサービス化のアーキテクチャを示す．

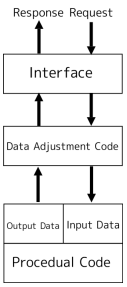

図 1　既存のアーキテクチャ

既存のサービス化のアーキテクチャは，技術の違いによるデータの形式の差を埋めるためにラッパーにふたつのコンポーネントを追加している．“ Input Data”は外部からのデータを入力データに変換する．“ Output Data”は出力データを外部で利用できるデータに変換する．

既存の研究で提案されているレガシーシステムをサービス化する手順を以下で示す．各手順は以下の順序で実行される．

- サービスの抽出
  レガシーシステムからサービスの処理に関連するコードを識別し，抽出する．
- 抽出されたコードのラッピング
  抽出されたサービスのインタフェースを定義する．
- サービスとして利用可能する
  サービスをビジネスプロセスに対応付ける．

3.2 既存のサービス抽出手法

既存の研究 [4] において，レガシーシステムからサービスを抽出する手法が提案されている．この手法では，レガシーシステムのソースコードをデータフロー図 (DFD) に変換することでシステムの分析を行なう．システムの分析では，DFD 上のデータに注目する．

既存のサービス抽出手法は，自己完結かつオープンなインタフェースを持つサービスを抽出可能である．既存のサービス抽出手法は，データの分類によって外部で利用可能なデータを出力するサービスを抽出できる．システムの内部でのみ利用可能なデータのやりとりをサービスの内部に隠蔽することによって，自己完結なサービスを抽出できる．

4 レガシーシステムの柔軟性の向上

既存の研究では実現できていない，レガシーシステムから抽出されたサービスのステートレス化を実現する．サービスが自身の機能に対して単純であることが，レガシーシステムの SOA への移行を成功させる鍵である [3]．サービスは，内部から状態の管理を分離されるべきである．既存のサービス抽出手法によって抽出されたサービスは，システム内部でのみ利用されるデータをサービス内部に隠蔽している．ユーザの状態の情報は，システム内部でのみ利用されるデータとしてサービス内部に隠蔽される．既存のサービス化のアーキテクチャは，サービスのステートレス化においてインタフェースを定義するためにサービスの内部に変更を加える必要がある．本研究では，内部に変更を加えることが困難なレガシーシステムのステートレス化を実現するためのアーキテクチャを提案する．

4.1 ステートレスなサービス化の実現方法

既存のサービス化のアーキテクチャにアスペクト指向を適用し，サービスの状態の管理をアスペクトとして分離する．既存のサービス抽出手法を用いてレガシーシステムから抽出されたサービスは，状態の情報を出力できない場合がある．既存のサービス化のアーキテクチャでは，サービスのステートレス化において，サービスの内部を変更する必要がある．

状態の管理の分離でアプリケーション側でユーザごとの状態を保持して対処する．アプリケーション側で保持する状態をアスペクトで管理する．アスペクトはサービス終了時に状態を取得してアプリケーション側に保存し，サービスの呼び出し時に状態を復元する．サービスから状態の管理をアスペクトとして分離することにより，サービスのステートレス化を実現する．

4.2 ステートレスなサービス化のアーキテクチャ

本研究では，既存のサービス化のアーキテクチャにアスペクト指向を適用してステートレス化を実現するためのアーキテクチャを提案する．**図 2** は本研究で提案するアーキテクチャの概要を示す．

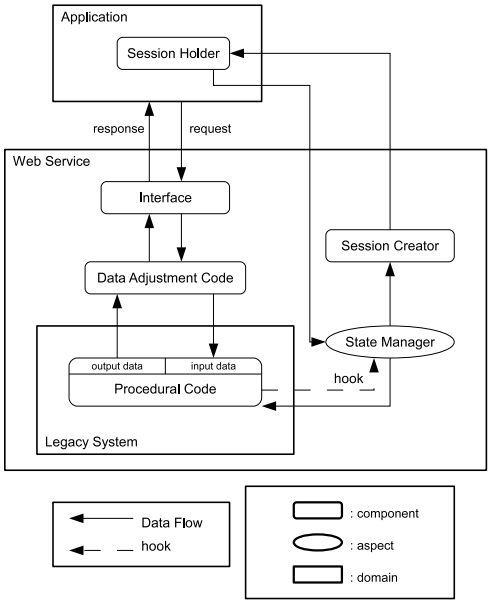

図 2　サービスのステートレス化のアーキテクチャ

図 2 は，サービスから状態の管理を分離できない場合にサービス内部に変更を加えずにステートレス化を実現するためのアーキテクチャを示す．“ State Manager”は状態の管理を示すアスペクトである．アスペクトはサービスの開始時に織り込まれてユーザの状態を復元し，サービスの終了時に織り込まれて状態を記録する．“ Session Creator”は“ State Manager”から状態を受け取り，“ Session Holder”内で受け取った情報を保持する．“ Data Adjustment Code”は“ Procedural Code”の入出力を Interface に適合させるためのグルーコードである．Interface が受信したメッセージ内のデータは，“ Data Adjustment Code”によって取り出されて“ Procedural Code”に渡される．“ Procedural Code”の出力データは，“ Data Adjustment Code”によって送信メッセージに変換されて Interface に返される．

4.3 サービスのステートレス化のプロセス

抽出されたサービスのステートレス化を実現するための手順として“ Analyzing the salvaged code”を追加した．ステートレス化を実現するためには，既存のアーキテクチャにおける手順に加えて抽出されたサービスの分析が必要であると考える．以下の**図 3** は拡張後の手順である．追加した手順の概要は後で述べる．

“ Analyzing the salvaged code”では，レガシーシステムから抽出されたサービスの分析を行なう．レガシーシステムから抽出されたサービスの分析では，レガシーシステムから抽出されたサービスを分析してステートレス化の実現方法を決定する．サービスから状態の管理の分離が困難な場合，本研究で提案するアーキテクチャにしたがってステートレス化する．

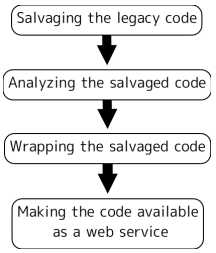

図 3　拡張後のサービス化の手順

5 事例検証

酒在庫システムの発注プロセスを用いた事例検証によって，提案したアーキテクチャが実現可能であることを確認した．事例となるシステムは C 言語を用いて実装した．

5.1 システムの概要

酒在庫システムの発注プロセスは，注文された商品の一覧を出力する．発注プロセスへの入力は商品数と商品名であり，出力は入力された商品の一覧である．

5.2 サービスの抽出

既存のサービス抽出手法を用いてサービスを抽出した．システムのソースコードから DFD を作成してデータを分類した．**図 4** は，データの分類後の DFD を示す．

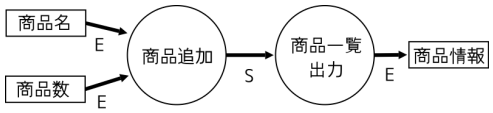

図 4　データの分類後の DFD

データの分類の結果，以下のサービスが抽出された．

- 商品追加サービス
- 商品一覧出力サービス

5.3 サービスの分析

抽出されたサービスを変更せずに，擬似的なステートフルサービスにする．抽出されたサービスは，内部の状態を出力することができない．アプリケーション側で状態を管理するためにアスペクト指向技術を用いる．

5.4 サービスのラッピング

システムをラッピングし，サービス化した．**図 5** は，サービス化されたシステムの構造を示す．アスペクトは“ Add_Item”と“ Print_Item”の実行前にユーザの状態を復元し，“ Add_Item”の実行後にユーザの状態を取得する．

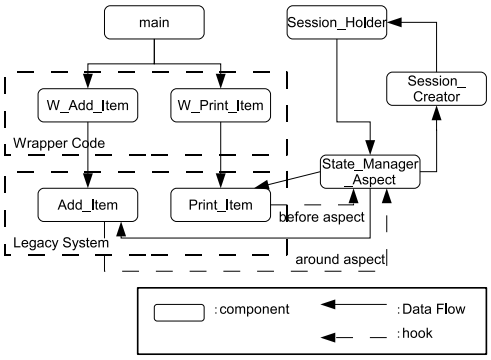

図 5　システムの構造

6　考察

事例検証により，本研究で提案したアーキテクチャが実現可能であることが確認できた．事例検証において，酒在庫管理システムに対してアーキテクチャを適用することができた．本研究で提案したアーキテクチャは，レガシーシステムのサービス化を実現することができると考える．

6.1　アスペクト指向技術の適用の考察

本研究ではサービスから状態の管理を分離するためにアスペクト指向技術を用いた．アスペクト指向技術の利用によって，抽出されたサービスの内部に変更を加えずに状態の管理の分離を実現できた．既存のサービス化のアーキテクチャでは，サービスのステートレス化の実現には内部の変更が必要である．アスペクト指向技術によってサービスの内部に変更を加えずにステートレス化を実現できる．内部に変更を加えることが困難なレガシーシステムに対するアスペクト指向技術の適用は有用であると考える．

6.2　アーキテクチャの考察

既存の研究において，サービスの柔軟性を確保する鍵が関心事の分離であることが挙げられている [3]．本研究では，アスペクト指向技術を適用することによってサービスの内部に変更を加えずに状態の管理を分離した．既存のサービス化のアーキテクチャでは，状態の管理を分離する場合にはサービスの内部を変更することが必要である．本研究のアーキテクチャは内部に変更を加えることが困難なレガシーシステムのステートレスなサービス化のアーキテクチャとして妥当であると考える．

サービスのステートレス化の実現により，サービスは同時により多くのユーザに対する処理を行なうことができるようになる．サービスのステートレス化により，サービス側で状態を保持する必要がなくなる．サービス側のサーバの付加が軽減されるので，同時により多くの要求に対応することができる．サービスがより多くの要求に対応できることにより，レガシーシステムの柔軟性が向上すると考える．本研究で提案したアーキテクチャにより，レガシーシステムの柔軟性を向上できると考える．

6.3　アーキテクチャの欠点の考察

本研究で提案したアーキテクチャでは，サービスの粗粒度を満たすことはできない．本研究で提案するアーキテクチャは，抽出されたサービスの内部を変更せずに SOA への移行を実現することを前提としている．既存のサービス抽出手法によって大きい粒度で抽出されたサービスを分割するためには，サービスの内部に変更を加えることが必要である．本研究は，既存の手法によってサービスの内部に隠蔽された状態の管理を分離することを目的としている．サービスの内部に変更を加えることなくステートレス化を実現しているので，目的を達成できていると考える．

7　おわりに

本研究では，レガシーシステムから抽出されたサービスのステートレス化を実現するためのアーキテクチャを提案した．既存のサービス化のアーキテクチャではサービスから状態を分離する場合に内部に変更を加える必要があるので，サービスのステートレス化が困難である．本研究のアーキテクチャではアスペクト指向技術を用いることによって，サービスに変更を加えずにステートレス化を実現することができた．アスペクトはユーザの状態をアプリケーション側に記録し，サービスの実行時にユーザの状態を復元する．ステートレス化の実現によってサービスは同時により多くの処理を行なうことができる．本研究のアーキテクチャによって，レガシーシステムからより柔軟にサービスを構築できると考える．

今後，大規模なシステムを用いてアーキテクチャの妥当性を確認することが必要である．本研究では，アーキテクチャが実現可能であることを簡単なシステムを用いて確認した．実際に稼働しているレガシーシステムを用いて，本研究のアーキテクチャが妥当であることを実証する必要がある．

参考文献

[1] A. G. Neat, “*Embracing SOA for the legacy world*,” http://www.ibm.com/developerworks/webservices/library/ar-embsoa/, 2006

[2] C. D. Induruwana, “Using an Aspect Oriented Layer in SOA for Enterprise Application Integration,” *ICSOC*, pp. 19-24, 2005.

[3] M. Sneed, “Wrapping Legacy Software for Reuse in a SOA,” *Multikonferenz Wirtschaftsinformatik 2006*, vol. 2, pp. 345-360, 2006.

[4] 井垣宏，木村隆洋，中村匡秀，松本健一，“DFD を利用したプログラム処理間の依存解析によるレガシーソフトウェアからのサービス抽出，” ソフトウェアエンジニアリング最前線 2009 情報処理学会 SE シンポジウム，pp.89-96，2009.