Web サービスのインタフェース変換の動的アスペクト記述

岩田 英一郎† 松本 倫子† 吉田 紀彦†

†埼玉大学大学院理工学研究科

1 はじめに

Web サービスによる異種アプリケーション連携では，アプリケーション固有のインタフェースを共通のインタフェースへ変換する必要がある．しかし，このために既存のアプリケーションのソースコードを手動で変更するのは望ましくない．先行研究 [1] において，異種アプリケーションを透過的に連携させる手法が提案されているが，アプリケーションの再コンパイルが必要であるため，大規模アプリケーションの連携への適用が難しいという問題がある．そこで本研究では，インタフェース変換コンポーネントをアスペクトとして記述し，アプリケーションのロード時・実行時に埋め込みを行うことを提案する．これにより，大規模アプリケーションの連携への適用が容易になる．

2 異種アプリケーションの透過的連携

ソフトウェアシステムは複数の既存アプリケーションを組み合わせて構築されることが多くなってきている．既存アプリケーションは他のアプリケーションと連携させることを考慮せずに設計されていることもある．また，異なるプラットフォームを対象とし，異なるプログラミング言語で開発された異種アプリケーションを連携させたい場合には，アプリケーション間でのインタフェースの変換 (命令やデータの変換) が必要になる．異種アプリケーションの連携を可能とする技術として Web サービスがあるが，それだけでは透過的 (既存のコードを手動で変更せず) に連携させることはできない．

2.1 Web サービス

Web サービスとは，異種アプリケーションを連携させる技術である．サービスを利用するプログラムをリクエスタ，サービスを提供するプログラムをプロバイダと呼ぶ．Web サービスでは，WSDL(Web Services Description Language) でサービスのインタフェースを記述し，リクエスタとプロバイダは SOAP(Simple Object Access Protocol) メッセージを用いて通信する．Web サービスは特定のプラットフォームやプログラミング言語，ミドルウェア技術に依存しないインタフェースを提供するので，インターネット上での異種アプリケーション連携が可能となる．

2.2 透過的な連携

Web サービスによって異種アプリケーションを連携させることが可能になったが，それだけでは透過的に連携させることができない．

異種アプリケーションを透過的に連携させる直感的なアプローチは，アプリケーション固有のインタフェースから Web サービスのインタフェースへ変換を行うブリッジプログラムを用いることである．図 1(a) のようにアプリケーションの外側でインタフェースの変換を行うことで透過性を満たすことができるが，プロセス間 (もしくはコンピュータ間) のリダイレクトによりオーバーヘッドが発生する欠点がある．

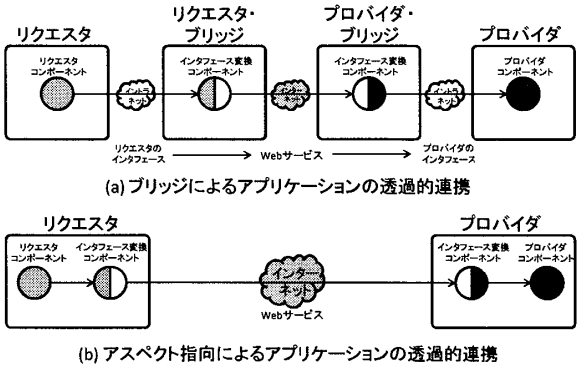

図 1: 異種アプリケーションの透過的連携

2.3 先行研究

上述の問題を避けるために，図 1(b) のように，アプリケーションの内側でインタフェースの変換を行う手法が提案されている [1]．この手法では，アスペクト指向を用いてコンパイル時にインタフェース変換コンポーネントを埋め込むことで透過性を満たす．しかし，アプリケーションを再コンパイルする必要があるため，大規模なアプリケーションを連携させる際に開発効率が悪くなるという問題点がある．

3 動的アスペクト指向

アスペクト指向の分野ではプログラムのコンパイル時にアスペクトの織り込みを行う静的織り込みのアプローチが主に議論されてきたが，近年では，プログラムのロード時・実行時にアスペクトの織り込みを行う動的織り込みが注目されている．静的織り込みは，コンパイラによる高度な最適化が行われ実行速度が速いが，アスペクトを変更するとプログラム全体を再コンパイルする必要がある．一方，動的織り込みはアスペクトを変更する際にプログラム全体を再コンパイルする必要がないのでアプリケーションの開発が効率化されるが，静的織り込みに比べて実行速度が遅いという問題がある．しかし，動的織り込みの性能を改善する研究 [2, 3] が報告されている．

4 提案手法

以上のことを踏まえて本研究では，インタフェース変換コンポーネントをアスペクトとして記述し，アプリケーションのロード時・実行時に動的に織り込む手

Dynamic Aspects for Interface Conversion of Web Services

Eiichiro IWATA†, Noriko MATSUMOTO† and Norihiko YOSHIDA†

†Graduate School of Science and Engineering, Saitama University

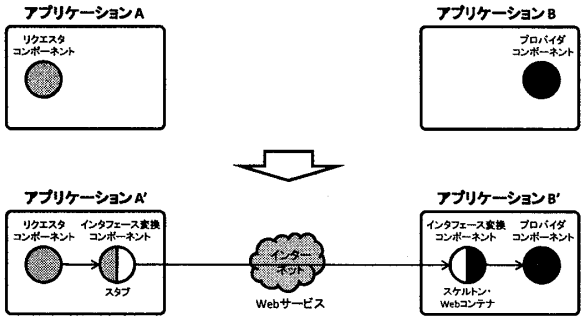

図 2: 提案手法の概要

法を提案する．これにより，アプリケーションの再コンパイルの必要がなく，大規模なアプリケーションの連携に関わる開発の効率化が期待できる．

図 2 に提案手法の概要を示す．アプリケーション A のリクエスタと，アプリケーション B のプロバイダを連携させる．アプリケーション A, B は異種アプリケーションであり，一般に異なるインタフェースを持っている．提案手法では，アプリケーション A, B にインタフェース変換コンポーネントを埋め込むアスペクトを記述し，動的に織り込みを行う．

提案手法の詳細は以下の通りである．

1. WSDL でインタフェースを定義する (プロバイダから自動生成できる)
2. プロバイダ側の作業
   (a) スケルトンを作成する (インタフェース定義から自動生成できる)
   (b) スケルトンをプロバイダを使って実装し，起動する (もしくは，プロバイダを Web コンテナにデプロイする)
3. リクエスタ側の作業
   (a) スタブを作成する (インタフェース定義から自動生成できる)
   (b) リクエスタにスタブを埋め込むアスペクトを記述し，動的に織り込む

ほとんどの作業は Web サービスのツールを利用することで自動で行うことができる．主な作業はリクエスタのアスペクトを記述することである．

5 実験

提案手法の有効性を確認するために，アスペクトを記述し，適用実験を行った．今回はリクエスタ・プロバイダ共に既存の Java アプリケーションを利用し，アスペクトの記述に AspectJ[4] を用いた．また，Web サービスの実行環境として Axis2[5] を用いた．なお，AspectJ では動的織り込みとしてロード時の織り込みのみをサポートしている．

図 3 にリクエスタのアスペクトの記述例を示す．ポイントカットでリクエスタコンポーネント内のジョインポイントを指定し，アドバイスでスタブを呼び出すようにしている．

実際に，リクエスタのアスペクトについてアプリケーション A のロード時に織り込みを行い，アプリケーションの透過的連携ができたことを確認した．また，動的織り込み処理による実行速度の低下が見られた．しかし，

```
import aPackage.Stub;
public aspect WebService {
 private Stub stub;
 pointcut atSomeMethod(ParamType param)
  : call(ReturnType aClass.someMethod(ParamType));
 ReturnType around(ParamType param)
  : atSomeMethod(param) {
  // 必要なパラメータをセットする
  Stub.SomeMethodRequest req = new Stub.SomeMethodRequest();
  req.setParam(param);
  // Web サービスを呼び出す
  stub.SomeMethodResponse res = Stub.someMethod(request);
  // 処理結果を使う
  return proceed(res.get_return());
 }
}
```

図 3: リクエスタのアスペクトの記述例 (AspectJ)

ロード時に 4，5 秒遅れる程度であり，開発効率にほとんど影響がない範囲であると考えられる．なお，実験に用いた計算機環境は CPU が AMD Sempron 3000+ 1.80GHz で，メモリが 1GB，OS は Linux である．また，AspectJ はバージョン 1.5.3 を，JVM として Sun Java SE 6 の HotSpot^{TM} Client VM を用いた．

6 まとめ

異種アプリケーションを透過的に連携させるには，インタフェースの変換コンポーネントを付加させることが必要になるが，本研究では，アプリケーションのロード時におけるインタフェース変換コンポーネントの埋め込みをアスペクトの動的織り込みによって実現する手法を提案した．これにより，既存アプリケーションを手動で変更することなく再利用して，他のアプリケーションと連携できるようになる．また，既存アプリケーションの再利用が向上することにより，既存アプリケーションの価値向上が期待できる．

今後の課題としては，実際に大規模なアプリケーション連携に適用し効果を検証することが挙げられる．

参考文献

[1] S. M. Sadjadi and P. K. McKinley: “Using Transparent Shaping and Web Services to Support Self-Management of Composite Systems”, *In Proc. of the Int’l Conf. on Autonomic Computing(ICAC’05)*, 2005

[2] Andrei Popovici, Gustavo Alonso, Thomas Gross: “Just-In-Time Aspects: Efficient Dynamic Weaving for Java”, *In Proc. of the 2nd International Conference on Aspect-Oriented Software Development(AOSD’03)*, 2003

[3] 佐藤, 千葉: “効率的な Java Dynamic AOP システムを実現する Just-in-Time Weaver”, 情報処理学会論文誌:プログラミング, vol. 44 no. SIG 13, 2003 年 10 月

[4] The Eclipse Foundation, The AspectJ Project, http://www.eclipse.org/aspectj/, 2009

[5] The Apache Foundation, Apache Axis2, http://ws.apache.org/axis2/, 2009