Web アプリケーションの Web サービス変換に関する研究

2002MT014　林　佑亮　　2002MT030　加藤　房良

指導教員　青山　幹雄

1. はじめに

近年, Web サービスの基盤技術(SOAP, WSDL, UDDI)が成熟しつつある. しかし, 利用可能な Web サービスが少ないため普及に至っていない. そこで本研究では既存に存在する豊富な Web アプリケーションに着目した. それを Web サービスとして利用可能にする変換方法を提案する.

2. Web サービスの問題点と解決策

2.1. Web サービスとは

Web サービスは, コンピュータ同士が連携することによってネットワーク上に分散している複数のサービスを統合して扱うことを目的とした技術であり, プログラミング言語, OS, 通信プロトコル等に依存しない特徴がある.

2.2. Web サービスの問題点

現在の Web サービスの問題点として, 我々は Web サービスの実装にかかる開発コストに着目した. 実際に利用・連携したい Web アプリケーションが, その開発コストのために Web サービス化されていない. 両者のインタフェースが異なるため, 既存の Web アプリケーションと同内容のシステムを Web サービスにするには, 一からシステムを再構築しなければならない場合がある.

2.3. 解決策の提案

前述した問題の解決策として, Web アプリケーションの入出力を Web サービスの入出力と対応できるようにマッピングする方法を考えた. Web アプリケーションのインタフェースは, HTTP によって受信される HTML に限定されており, 人がブラウザを利用して扱うことが前提である. これを Web サービスとして扱えるようにするための変換方法を提案する.

3. Web アプリケーション Web サービス変換

3.1. 変換アーキテクチャの提案

Web アプリケーションを Web サービスに変換するアーキテクチャとして, 以下の 2 つの方法が考えられる.

(1) Web アプリケーションに直接変換機能の追加

Web アプリケーションを構築しているサーバ上に, 直接 Web サービス変換プログラムを作成する.

(2) Proxy サーバによるラッピング

サービスリクエスタと Web アプリケーションの間に Proxy サーバを設置する. Proxy サーバに仲介機能(ラッパー)を置いて, 変換を実現する. Proxy サーバと Web アプリケーションを 1 つのサービスプロバイダと見なす.

方法(2)は, Web アプリケーションの HTML レスポンスを利用して変換する方法をとるため, Web アプリケーションの実装言語に依存しない. また, 方法(1)のように直接 Web サーバ内に変換プログラムを作成しなくて済むため, Web アプリケーションを一切変更せずに再利用できる. そこで本研究では, 図 1 に示すように Proxy サーバによるラッピングアーキテクチャを用いる[2].

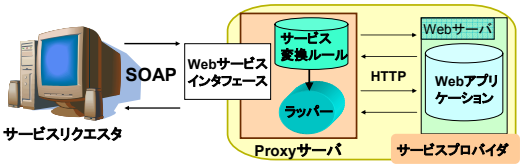

図 1 ： ラッピングアーキテクチャ

3.2. インタフェースの変換方法

ラッピングアーキテクチャを実現するためには以下の 2 つの要素が必要となる.

(1) ラッピングタスク

ラッピングタスクは, Web アプリケーションと Web サービスのインタフェースの違いを克服するため, Web アプリケーションとの通信を可能にするインタフェースを持つ. さらに Web アプリケーションからの HTML レスポンスを Web サービスのレスポンスとして扱える機能が必要となる[3].

(2) Web サービス変換ルール

Web サービス変換ルールでは Web アプリケーションとの通信手順や Web アプリケーションと Web サービスのリクエスト, レスポンスの対応方法を規定する.

3.3. ラッピングタスク

図 1 で示したアーキテクチャの詳細を図 2 に表す.

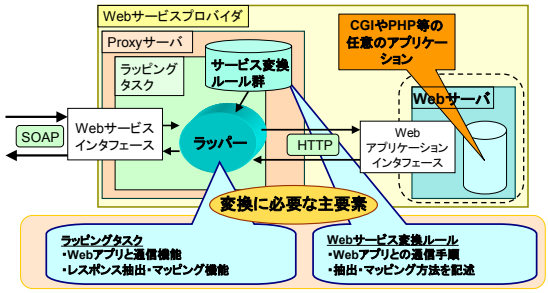

図 2 ： インタフェース変換の詳細

ラッピングタスクとして持つ機能を以下に示す(図 3).

(1) リクエストジェネレータ

サービスリクエスタから受け取ったリクエストメッセージから Web アプリケーションへ送信する要素を抽出する. 次に, Web アプリケーションの入力インタフェースである HTML 文の FORM 要素の子要素のパラメータへ, 対応づけて代入する.

(2) HTTP リクエストの送信

(1)で生成したリクエストを Web アプリケーションへ送信する. 送信パターンは以下の 2 通りである.

1) リクエスト送信・レスポンス受信を一度だけ行う
2) 複数回にわたって送信・受信を繰り返す

(3) HTML Tidy によるレスポンスの整形化

HTML を走査し, 文法的な誤りを検出・修正し, XHTML に整形化する. (2)によって得られた HTML レスポンスを抽出可能な形式にするために HTML 文書を整形式にする. 本研究では, HTMLTidy をプログラム途中で呼び出し, HTML を整形化する.

(4) レスポンスパーサ

(3)によって整形化された XHTML 文書からサービスリクエスタへのレスポンス要素を抽出する.

(5) レスポンスマッピング

(4)で抽出した求められる要素をサービスリクエスタが処理可能な形にマッピングする.

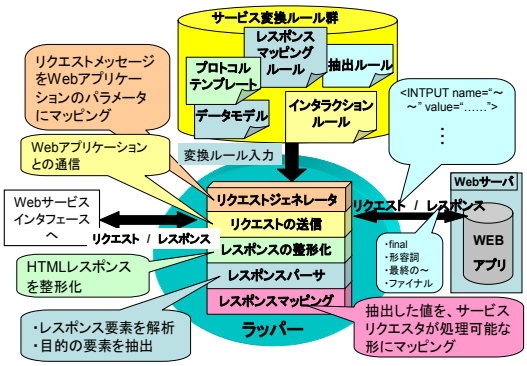

図 3: ラッピングタスクの処理

また, ラッピングタスクによるデータフローを図 4 に示す.

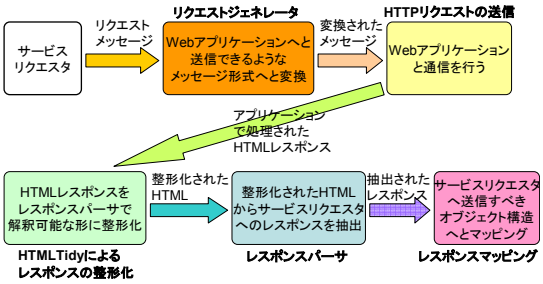

図 4 ： ラッピングタスクによるデータフロー

4. Web サービス変換ルール

4.1. 変換ルールの構成

Web サービス変換のために必要なルールを XML で記述する[3][4][5]. また, 変換ルールの関連を図 5 に示す.

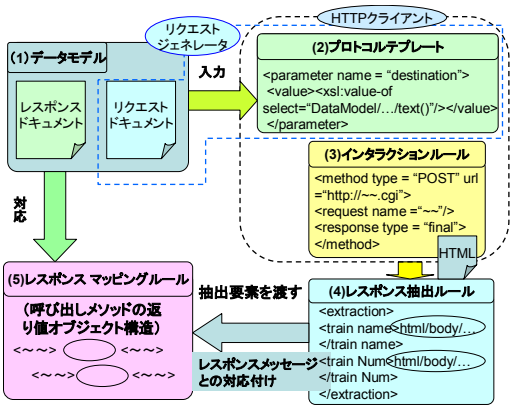

図 5 ：Web サービス変換ルールの関連

変換ルールとして以下のものを考えた.

(1) データモデル

Web アプリケーション上で使用されるデータを We サービスで使用可能なデータ形式に変換時に用いられるモデル

(2)プロトコルテンプレート

Web アプリケーションのリクエスト, レスポンスと Web サービスリクエスタとの通信と対応づけるルール

(3)インタラクションルール

Web アプリケーションとの通信順序を定めたルール

(4)レスポンス抽出ルール

Web アプリケーションからの HTML レスポンスから目的の要素を抽出するためのルール

(5) (2)と同様

4.2. 変換ルールの具体例

変換ルールを英単語検索の Web アプリケーションを例として説明する.

(1) データモデル

Webアプリケーションで使用されるデータをWebサービスで使用可能な形のモデルとして記述する. データモデル例を図 6 に示す.

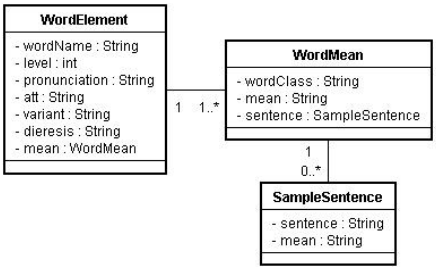

図 6: データモデル例

(2) プロトコルテンプレート

Webアプリケーションへ送信すべきリクエストを規定する. サービスリクエスタのリクエストを Web アプリケーションのリクエストであるパラメータ変数と対応するようにマッピングする. オブジェクトモデルを図 7 に示す.

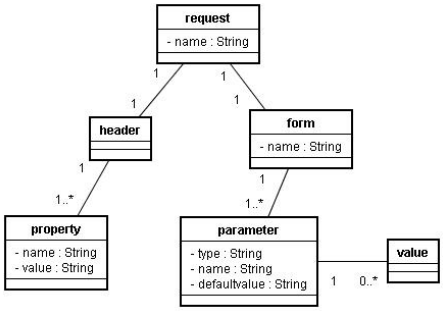

図 7 : プロトコルテンプレート

(3) インタラクションルール

Proxy と Web アプリケーションとの通信を規定する.

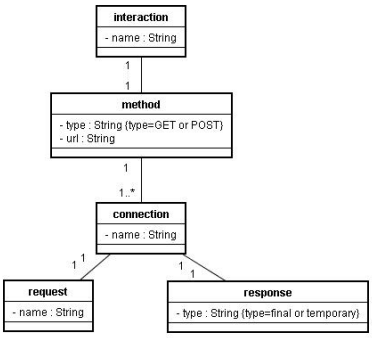

図 8 : インタラクションルール

(4) レスポンス抽出ルール

Web アプリケーションから得た HTML レスポンスから, サービスリクエスタへのレスポンス要素が存在する場所を XPath で記述する. この時の HTML レスポンスは XHTML 文書へと整えられる.

(5) レスポンスマッピングルール

HTML レスポンスから抽出した各要素をサービスリクエスタへのレスポンスとしてマッピングするため, 抽出した要素との対応関係を記述する.

5. 実装方法

ラッパー開発者が既存の Web アプリケーションをラッピングし, Web サービスに変換するための実装方法を以下に示す.

(1) データモデルの作成とサービス定義

ラッパー開発者は, ラッピングするWebアプリケーションが使用するパラメータを抽出する. 次に HTML レスポンスを解析し, 目的のHTMLレスポンスを得るまでの通信の流れを解析する. そして, HTML レスポンスから抽出され得る要素の集合をデータモデルとして定義する.

(2) Web アプリケーションの挙動の解析

次にサービスリクエスタから送られてくるリクエストメッセージと Web アプリケーションへ送信するパラメータとの互換性を確立する. それを基にして, プロトコルテンプレートを作成する. また, 通信の流れをインタラクションルールとして記述する.

(3) HTML レスポンスの解析と抽出ルールの規定

サービスリクエスタへ提供するレスポンス内容を含んでいる HTML レスポンスの中で, 目的の要素がどこに存在しているかを把握する. 把握した内容を基にして, レスポンス抽出ルールを記述する.

(4) 抽出した要素のマッピング

レスポンス抽出ルールから得た要素を, サービスリクエスタが求めている形にマッピングするためにレスポンスマッピングルールを記述する.

(5) Proxy サーバ内にラッパーを構築

前述までに生成したルールを利用してラッパーを作成する. ラッパーはラッピングタスクを基にして作られる. 例として Java を用いたラッパーを作成する方法を示す.

Proxy サーバ内にラッパークラスを作成する. ラッパークラスとは別に, サービスを供給するためのメソッドを含むサービスクラスを公開する. サービスクラスでは, ラッパークラスのラッピングメソッドを利用して, Web アプリケーションから目的のレスポンスを抽出し, 目的のサービスを提供するメソッドを公開する(図 9).

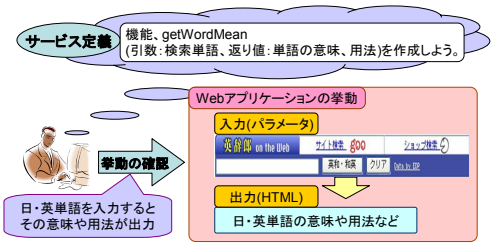

図 9 ： ラッパー開発者の作業

6. プロトタイプによる評価

6.1. Web サービス変換ルールの妥当性検証

本研究で提案した 5 種類の各変換ルールの妥当性や必要性を検証する. 検証内容は以下の 3 つである.

(1)データモデルの効果

適切な変換のためにはサービスリクエスタより送られてきたリクエストデータや送り返すレスポンスデータの意味を知る必要がある. そのため, Web アプリケーションのデータ構造をモデル化した. この結果, 意味定義をしたために, Web アプリケーションのデータを Web サービス上でのオブジェクトとして扱えるようになった.

(2)プロトコルテンプレートとインタラクション分割の効果

一つの Web アプリケーションがタイプの異なる機能を提供する場合, リクエスト送信パターンが異なる場合がある. そこで, 上記ルールを分割することにより, リクエストを独立に送信可能となり, 適時に送信内容を決められる効果がある.

(3)抽出ルールとマッピングルールの分割の効果

抽出ルールとマッピングルールを分割して定義することで, マッピングルールにレスポンスオブジェクトの構造を定義し, 複数の抽出ルールを組み合わせて利用可能となる. これにより抽出ルールを追加する場合, マッピングルールは変更せずに済む.

6.2. 本研究によるコストダウンの検証

本研究で提案した変換方法を実現するラッパーのプロトタイプを Java で実装した. ここでは英単語の辞書 Web アプリケーションを例題とした. ラッパーと変換ルールのコード行数を表 1 に示す.

表 1 ： ラッパーと変換ルールの実装規模

サブシステム	ラッパー		変換ルール
	ラッピングタスク	サービス実現クラス	
コード行数	432 行	442 行	101 行

ラッピングタスクは再利用できる. 異なる Web アプリケーションを変換する際には変換ルールとサービス実現クラスのみを書き換えるだけで良いので, 開発コストの削減が期待できる.

7. 考察と今後の課題

Web アプリケーション作成者でなくともラッピング可能にするために, サービスリクエスタが求めるレスポンスを Web アプリケーションの HTML レスポンスから抽出する方法を提案した. この方法は HTML レスポンスのみに着目すれば良いので, ラッパー開発者は Web アプリケーションの実装を知らなくても良い. よって, 既存の Web アプリケーションを変更せずに再利用可能なため, 開発コストの削減が期待できる.

関連研究として, スクリプト言語を用いてラッピングをする方法がある[1]. これは変換ルール記述がその実装言語に依存する. 本研究では実装言語に依存しない変換ルールを表現するため, XML で記述した.

今後は以下の課題を検討する必要がある.

(1) ラッパー開発者のルール作成時の負担の軽減

ラッパーを開発する場合に, 開発支援環境を充実させ, 変換ルールの半自動生成を考える.

(2) 複数の Web アプリケーションのラッピング

類似のサービスを提供している複数の Web アプリケーションを同一の Proxy サーバ内で統合し, 組み合わせたサービスを実現するラッピングの応用が考えられる.

8. まとめ

本研究では Web アプリケーションを Web サービスとして扱える仕組みとして, 変換アーキテクチャ, ラッピングタスク, サービス変換ルールを提案した. またそれに伴い, 考えられる問題点や今後の課題も提示した. 今後は, 提示した問題点や課題を解決していくことを目指す.

参考文献

[1] 藤原 克哉ほか, フォーム自動記入のための Web サービス変換フレームワークの開発, 情報処理学会研究報告ソフトウェア工学, 2004, pp 61-66.

[2] 高橋 健一ほか, プロキシサーバによる WEB アプリケーションの WEB サービス変換, 情報処理学会ウィンターワークショップ 2005 論文集, 2005, pp.95-96.

[3] TaMeX, http://www.cs.ualberta.ca/~stroulia/TAMEX/.

[4] S. Jabbour and A. Vercoustre, Wrapping Web Pages into XML Documents : A Practical Experience and Comparison of Two Tools, AusWeb'02, 2002.

[5] Y. Jiang and E. Stroulia, Towards Reengineering Web Sites to Web-services Providers, CSMR ' 04, 2004, p.296.