社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2007－DD－59 (1)
2007／1／26

冗長性排除に基づくWSDL文書の可読性と記述性の向上

福盛秀雄　村岡洋一
早稲田大学 理工学研究科

2007年1月26日

概要

WSDL(Web Services Description Language)はWebサービスの仕様記述言語であり、様々な利用形態に対応できるような柔軟性を持った記法を特徴としている。しかし、実際に利用されているWebサービスのWSDLの内容はそのような柔軟性をかならずしも必要とせず、結果として記述の冗長性を生み出している。このような冗長性はWSDLの可読性と記述性の観点から望ましいものではない。本発表ではWebサービスの記述内容の分析に基づき、一定の記述ルールが存在しているかについての検証をまず行い、次にその検証結果に基づいた短縮記法の提案を行う。

Enhancing readability and writability of WSDL documents by removing extra verbosity

Hideo Fukumori　Yoichi muraoka
Graduate School of Science and Engineering, Waseda University

January 26, 2007

abstract

WSDL (Web Services Description Language) is a language used to describe specification of Web services. While it features great flexibility to accomodate wide range of usage style, actual WSDLs in Web Services does not make use of it and usually ends up in extra verbosity. To enhance readability and writability of WSDL, we first analyze actual WSDL files in seveal Web Services. Based on that analysis, we present shorter description format for WSDL.

1　はじめに

Webサービス (Web Services) はインターネット上での分散コンピューティングの方式として広く普及しつつある。

Webサービスは、サービスの仕様を記述する方法として、WSDL(Web Services Description Language) と呼ばれる、XMLを用いた記法を用いている。WSDLはプログラミング言語に依存しない記法であり、サービスの提供側 (サーバ) と利用側 (クライアント) が異なる言語、あるいはオペレーティングシステムであるような、いわゆるヘテロジーニアスな環境にも容易に対応可能であるという利点を持つ。

WSDL は柔軟性を重視した記法であり、通信データの形式、サービスの形態、その配備 (デプロイメント) のそれぞれについて、さまざまな形態に対応することが出来るような工夫がこらされている。

しかし実際の Web サービスの WSDL 記述においては、一定の規則による記述が普及しており、WSDL が用意している柔軟性は活用されていないだけでなく、その記述を無用に冗長とする結果につながっている。

WSDL は Web サービスの通信フォーマットである SOAP とは異なり、人間がサービスの仕様を決定・理解するために読み書きすることが必要な文書であるが、冗長な記述は人間による読み書きの効率を低下させる。これは Web サービスのさらなる普及を妨げる原因の一つとなり得る問題である。

本発表では、現在の Web サービスの WSDL の記述内容の分析をまず行い、その中に見られる規則性を示す。次にその規則性を活用し、情報を失うことなく WSDL の記述量を短縮することが可能であること、およびその方法の一例を提案した後、その効果について論ずる。

ここで述べる WSDL の記述の短縮は、現存の WSDL を短縮記法に変換する、また短縮記法で記述された WSDL を現存の WSDL に変換することの両者を実現することを目標とするものである。

2 Web サービスと WSDL

Web サービス (Web Services) は従来から行われてきた分散コンピューティングを WWW の枠組みを活用して実装しようとするものである。

Web サービスの構成要素としては SOAP[1], WSDL[2], UDDI[3] の三つが挙げられる。SOAP は XML に基づいたデータ送受信形式の標準として広く使われている。一方、UDDI は IBM, Microsoft, NTT による公開レジストリが存在したものの、2006 年にはいずれも稼動を停止しており、普及したとは言い難い状況となっている。

今回題材とする WSDL は Web サービスの提供する機能を XML 形式で記述したものであり、プログラミング言語に依存しない仕様記述方式として、多くの Web サービスで利用されている。

WSDL は

- データ型定義 types
- メッセージ定義 message
- オペレーション定義 operation
- ポート型定義 portType
- バインディング定義 binding
- ポート定義 port
- サービス定義 service

を主な構成要素として持つ。これら構成要素は Web サービスを提供する側 (サーバ) の製作者が XML のタグとして記述し、提供する。

3 WSDL の冗長性

WSDL の記法は様々な利用形態を想定し、柔軟性を最大限確保するように定義されているが、その柔軟性の確保のために記述が冗長となる部分も多い。

一例として Amazon Web Services[4] の E-Commerce Service[5] を取り上げる。同サービスの WSDL 定義にある portType タグの内部には、以下にに示すような形式で operation 定義が列挙されている箇所が見られる。

```
<operation name="Help">
  <input message="tns:HelpRequestMsg" />
  <output message="tns:HelpResponseMsg" />
</operation>
<operation name="ItemSearch">
  <input message="tns:ItemSearchRequestMsg" />
  <output message="tns:ItemSearchResponseMsg" />
</operation>
...
```

同様の operation 要素の記述は WSDL ファイル内に約 20 個存在しており、その定義だけで 77 行にもおよんでいる。

上記の例を見ると、operation 要素の内容は

```
<operation name="{OpName}">
  <input message="tns:{OpName}RequestMsg" />
```

```
  <output message="tns:{OpName}ResponseMsg" />
</operation>
```

という形態が繰り返し現れることが分かる。{Opname} は上記に示す例においては" Help" および"ItemSearch" がそれぞれ入る。

このような規則性がもし一般的に存在するとすれば、これを利用することにより WSDL ファイルの記述を短縮し、サービス提供者が WSDL ファイルを記述する際に、またサービス利用者がそれを理解する際の負担が軽減できるものと期待される。

4 プログラミング言語によるインタフェース記述とその課題

上記で述べた WSDL の記述の冗長性に対する一つの解決策としては、インタフェースを特定のプログラミング言語で記述するというアプローチも存在する。以下、その概略と問題点について説明する。

WSDL はプログラミング言語に依存しない仕様記述方式であるが、サービスを提供するプログラム、あるいはサービスを利用するプログラム自体は特定のプログラミング言語により記述する必要がある。特定のプログラミング言語と WSDL の間を結ぶ方法としては、以下の二つのアプローチが存在する [6]。

トップダウン: WSDL を記述後、これを Web サービスの実装に用いるプログラミング言語で記述されたインタフェースへ変換し、そのインタフェースを雛形として Web サービスの本体となるプログラムを記述する方式

ボトムアップ: Web サービスのインタフェースおよび本体となるプログラムを特定のプログラム言語で記述した後に、Web サービス用のツールキットを用いて WSDL を生成する方式

特にボトムアップ式のアプローチを利用する場合、サービス提供者およびサービス利用者は WSDL の代わりに特定のプログラミング言語によりインタフェースが記述出来るという利点がある。しかしその一方で

- 言語ごとに個別のツールキットを用意する必要がある。また Python などの動的言語の場合にはインタフェース記述による方式が提供されていないケースもある。
- サービス提供側のプログラムとサービス利用側のプログラムが異なるプログラミング言語であっても互換性が保たれるように考慮する必要がある場合において、プログラミング言語で表現されるデータ型と、WSDL にて表現されるデータ型の対応ルールによっては非互換性が発生する場合がある。

という課題が発生する。

本発表では、XML の持つ言語中立性を確保したまま WSDL の記述を以下に簡潔に行うかという観点から検討を行うものとする。

5 既存の Web サービスにおける WSDL の記述内容の分析

WSDL 内のポート型 (portType) 定義内にある operation 要素の記述に関し、既存の Web サービスの WSDL において、ここまでで述べた規則性が一般的に観察できるものであるかを検証するために、表 1 に示すサービスについて確認を行った。

表 1 operation 要素の規則性確認対象となる Web サービス一覧

サービス名	operation 数
Amazon E-Commerce	19
Amazon Mechanical Turk[7]	27
Amazon Simple Queue[8]	11
Google CampaignService[9]	10
Google Search[10]	3
Google TrafficEstimator[11]	3
eBay Web Service[12]	122
Microsft TerraService[13]	16

その結果、いずれの operation 定義についても

```
<operation name="{OpName}">
```

```
  <input message="{Prefix}{OpName}{Suffix1}" />
  <output message="{Prefix}{OpName}{Suffix2}" />
</operation>
```

という形態を基本としていることが明らかとなった。これは WSDL 1.1 仕様の "2.4.2 Request-response Operation" に基づく形式であるが、特に

- operation 要素の name 属性は input および output 要素の message 属性の一部分に一致する
- input および output 要素の message 属性の prefix はともに同一のものが使用される

という規則性が見られる点が特徴的である。

Web サービスによっては、input および output 要素の属性として name 属性が存在する、および operation 要素の中に fault 要素が含まれる場合も存在するが、その場合でも上記の規則性は満たされている。

6 WSDL 短縮記法の検討

上記にて述べた分析結果に基づく、WSDL 短縮記法について検討する。

ます、短縮記法を実現するに当たっては、

- 他のマークアップ言語などを利用することにより XML の記法に起因する煩雑さを取り除き、記述を簡潔にする
- 記述されている内容そのものに存在する冗長性に着目し、これを排除することにより、記述量を削減する

の二つの方向からのアプローチが考えられる。

XML の記法そのものを別のマークアップ言語などにより簡潔化する方式は、Relax NG Compact Syntax[14] や S 式により XML を表現する手法 [15]、または YAML[16] などのマークアップ言語を用いる手法など、従来より多数存在する。

一方、記述されている内容そのものに存在する冗長性を排除することにより記述量を削減する方式としては、先に述べたような WSDL の operation 定義の例に見られるような、一定の記述が繰り返し発生している部分に注目し、その記述のルールを抽出する方式が考えられる。

記述内容に存在する冗長性の排除は、前者の XML の記法の簡潔化によるアプローチではカバーできないことも多い。また、この二つのアプローチは背反するものではなく、両者を併用することも可能である。

今回の短縮記法においては

- 既存の XML のツールキットを用いて実現できること
- 必要に応じ WSDL の従来の記法の中に埋め込むことが出来ること

を重視し、XML 記法の簡潔化は採用せず、記述内容の冗長性の排除により記述量を削減するアプローチを採用した。以下に operation 要素の記述短縮のための記法の一例を示す。

```
<operations namePrefix={NamePrefix}?
            messagePrefix={MessagePrefix}
            inSuffix={InSuffix}
            outSuffix={OutSuffix}>
  {OpName1}
  {OpName2}
  ....
  {OpNameN}
</operations>
```

従来の operation 要素に代わり operations 要素を定義し、その属性として、messagePrefix, inSuffix, outSuffix の三つを用意する。これらに加え、オプションの属性として namePrefix を定義する。この属性は input および output 要素で name 属性が存在する場合に使用される。

operations 要素の本体には、従来の operation 要素の name 属性の値 (上記では {OpName1}...{OpNameN} で示した) を列挙するものとする。

実際の定義において operations 要素は、WSDL の名前空間 (namespace) と異なる名前空間で定義し、適切な前置詞 (prefix) を付けることにより、WSDL 文書内に直接埋め込んだ場合にも短縮記法部分の切

り出しは可能となるように配慮を行う。

なお、上記のルールに従わない記述については従来の記法に基づく operation 要素をそのまま記述することにより情報を失うことなく記述することができる。このようなケースは WSDL の定義が本来意図していた柔軟性の確保に沿った記述であり、それによる記述量の増加は許容されるものと考える。

7 WSDL 短縮記法の効果

本短縮記法を用いた記述について、まず実際の例を挙げて説明する。

Amazon Simple Queue の場合、WSDL 内にある(従来型の)portType 定義は以下のような形態となっている。

```
<portType name="AWSSimpleQueueServicePortType">
  <operation name="CreateQueue">
    <input message="tns:CreateQueueRequestMsg"/>
    <output message="tns:CreateQueueResponseMsg"/>
  </operation>
  <operation name="ListMyQueues">
    <input message="tns:ListMyQueuesRequestMsg"/>
    <output message="tns:ListMyQueuesResponseMsg"/>
  </operation>
  (....)
  (.... 他の operation 定義 (26 行)....)
  (....)
  <operation name="GetQueueEntryCount">
    <input message=
          "tns:GetQueueEntryCountRequestMsg"/>
    <output message=
          "tns:GetQueueEntryCountResponseMsg"/>
  </operation>
  <operation name="Help">
    <input message="tns:HelpRequestMsg"/>
    <output message="tns:HelpResponseMsg"/>
  </operation>
</portType>
```

これを短縮記法で表現したものを以下に挙げる。

```
<portType name="AWSSimpleQueueServicePortType">
  <operations messagePrefix="tns:"
              inSuffix="RequestMsg"
              outSuffix="ResponseMsg">
    CreateQueue ListMyQueues DeleteQueue
    ConfigureQueue Enqueue Read Dequeue
    ReadById ReadLock GetQueueEntryCount Help
  </operations>
</portType>
```

短縮記法では従来の WSDL の記述に比べ、参照する場合においては portType 内のオペレーションの種類を一覧できる点、および記述する際にはオペレーションの名称だけを意識して列挙できる点が特徴となっている。

次に portType 内で定義されている operation 要素について、オリジナルの WSDL による記述と短縮記法にて記述した場合の行数に基づいた比較を行う。

行数の計測は以下の条件に従って行った。

- portType 要素、operation 要素、operations 要素の開始タグ、終了タグはそれぞれ 1 行として取り扱う
- 空行、コメント行は行数に含まない

結果を表 2 に示す。

表 2 portType を対象とした WSDL と短縮記法の行数比較

サービス名	WSDL	短縮記法
Amazon E-Commerce	78	23
Amazon Mechanical Turk	110	30
Amazon Simple Queue	46	15
Google CampaignService	52	17
Google Search	14	7
Google TrafficEstimator	17	8
eBay Web Service	490	126

portType 要素全体の記述行数から見た場合には、短縮記法により記述量は最小でも 50%、最大で 25% 程度にまで削減されることが分かる。

本比較は operation 名を一行に一つずつ列挙した場合についてであるが、本短縮記法においては一行に複数の operation 名を記述しても良い。この場合、オリジナルの WSDL と比較した場合の行数の削減はさらに大きなものとなる。

8 関連研究

WSDL 記述と参照の負担をエディタにより軽減する方式はいくつか試みられている [17][18]。エディ

タによるWSDL参照と記述のサポートは基本的に

- グラフィカルな表現によるWSDLの内容の表示と編集
- ツリービューによるWSDL要素のナビゲーション

を特徴とする。しかしグラフィカルな表現による表示と編集はWSDLの規模が大きくなるにつれ操作が困難になること、またエディタが高機能になるほどその操作習得のコストが大きくなることという課題がある。またツリービューによるWSDL要素のナビゲーションについても、WSDLの規模が大きくなった場合には全体像を把握、および必要な要素を探し出すという作業の負担はそれほど軽減されないという問題が残る。

実装言語から自動的にWSDL等を生成する方式は、先に述べた言語ごとのツールキットを用いる方法の他に、XSLTを拡張したGridXSLと呼ばれる言語により、実装を記述し、これを元にWSDLを自動的に生成する方法[19]がある。同様の方式として、独自言語によりWebサービスのインタフェース他の情報を記述し、必要に応じこれにメタデータを加えることにより、WSDLの一部分を生成する、あるいはWSDLから独自言語への変換を行おうとする試み[20]も行われている。これらについては独自言語を用いて仕様から実装までの領域をカバーしようとしている点、および実際のWebサービスのWSDLに見られる規則性に基づく記述の短縮については言及していないという点が本発表にて説明した方式との相違点となっている。

9 まとめと今後の課題

本発表ではWebサービスの仕様記述言語であるWSDLを題材に、その冗長性にまつわる問題と、実際のサービスにて記述されているWSDLの内容の分析を元に、その記述を短縮する方式について述べた。

今回取り上げたのはWSDL内にあるportType要素に含まれるoperation要素、についてであるが、今回のアプローチはWSDL内のmessage要素、およびbinding要素内に存在するoperation要素など他の構成要素についても適用が可能であるものと考えられる。特にportType要素とbinding要素の共通部分をさらに統合することにより、より徹底した記述の短縮化が出来るものと期待される。

今後の課題としてはさらに、既存のWSDLから短縮記法への変換、あるいは短縮記法からWSDLへの変換のための実装が挙げられる。XSLTを利用する、あるいは汎用のプログラミング言語を用いるかについては現在検討中である。

WSDLの多くの部分を占めるtype要素の実体はXML Schemaである。XML Schemaについても同様の規則性が見られるものと予想されるが、WSDLに限定されないかたちでの短縮記法の方式を含め、さらに検討を進めていく予定である。

参考文献

[1] Martin Gudgin et al., "SOAP Version 1.2 Part 1: Messaging Framework", http://www.w3.org/TR/soap12-part1/, June 2003

[2] Erik Christensen et al., "Web Services Description Language", http://www.w3.org/TR/wsdl, March 2001

[3] Luc Clement et al., "UDDI Version 3.0.2", http://www.oasis-open.org/committees/uddi-spec/doc/spec/v3/uddi-v3.0.2-20041019.htm, October 2004

[4] Amazon Web Services, http://aws.amazon.com/

[5] Amazon E-Commerce Service, http://developer.amazonwebservices.com/connect/kbcategory.jspa?categoryID=5

[6] Open Grid Services Development Framework User's Guide, http://www.globus.org/toolkit/3.0/ogsa/docs/java_programmers_guide.html, March 2003

[7] Amazon Mechanical Turk,

http://aws.amazon.com/mturk

[8] http://www.amazon.com/Simple-Queue-Service-home-page/b/ref=sc_fe_l_2/102-1960346-6942557?ie=UTF8&node=13584001

[9] Google Adsense CampaignService, http://www.google.com/apis/adwords/ developer/adwords_api_services.html

[10] Google SOAP Search API, http://code.google.com/apis/soapsearch/ reference.html

[11] Google AdWords TrafficEstimatorService, http://www.google.com/apis/adwords/ developer/TrafficEstimatorService.html

[12] eBay developers program, http://developer.ebay.com/developercenter/xml/

[13] TerraServer, http://terraserver.microsoft.com/ webservices.aspx

[14] James Clark, "RELAX NG Compact Syntax", http://www.oasis-open.org/committees/relax-ng/compact-20021121.html, November 2002

[15] C. M. Sperberg-McQueen, "S-expressions for XML documents and for XML Schema components", http://www.w3.org/People/cmsmcq/2001/xs-sexp/xscomp.html, July 2001

[16] YAML, http://www.yaml.org/

[17] Introduction to the WSDL Editor WTP 1.5.1, http://wiki.eclipse.org/index.php/ Introduction_to_the_WSDL_Editor

[18] WSDL Editor and Documentation Generator, http://www.altova.com/products/xmlspy/ graphical_wsdl_editor.html

[19] Peter M. Kelly et al., "A Simlified Approach to Web Service Development", Proceedings of the 2006 Australasian workshops on Grid computing and e-research, 2006

[20] Dominic Cooney et al., "A programming language for web service development", Proceedings of the Twenty-eighth Australasian conference on Computer Science, 2005