Globus Toolkit 4における WSRFサービス記述のアノテーションによる補助

中田秀基[†] 竹房あつ子[†] 岸本誠[†,††]
工藤知宏[†] 田中良夫[†] 関口智嗣[†]

グリッド上のサービスを記述するための枠組として，Webサービス規格の一つである Web Services Resource Framework (WSRF) がある．Globus Toolkit 4は，WSRF を用いてサービスを記述するためのツールの一つで，多くのプロジェクトで用いられているが，記述は煩雑で十分に容易であるとは言いがたい．我々は，Globus Toolkit 4 による WSRF 記述を補助するツールを作成した．この補助ツールを用いると，ユーザは通常のオブジェクト記述にアノテーションを適切に付加するだけで，煩雑な Web Services Description Language (WSDL) によるインターフェイス記述をすることなく，WSRF に基づくサービスを記述することができる．さらに，通信対象となるデータのスキーマが独立に規定されている場合にも，対処することができる．簡単なサービスをこの補助ツールを用いて記述した結果，大幅に記述量を削減できることを確認した．

An Annotation-based Helper for WSRF Service Description on Globus Toolkit 4

Hidemoto Nakada ,[†] Atsuko Takefusa ,[†]
Makoto Kishimoto,[†,††] Tomohiro Kudoh ,[†] Yoshio Tanaka [†]
and Satoshi Sekiguchi [†]

Web Services Resource Framework (WSRF) is a Web Service standard to describe services for grid environment. The Globus Toolkit 4 is one of the tools to describe WSRF services. Although it is widely used in several grid projects, to describe WSRF service with the tool is not that easy. We designed and implemented a helper tool to complement the Globus Toolkit 4. It allows programmers to write WSRF services with just adding annotations to ordinary Java programs, without writing annoying interface descriptions in WSDL (Web Services Description Language). We tested the helper tool by writing a simple service with it and confirmed that it drastically reduced lines and numbers of files to achieve the service.

1. はじめに

グリッド上で利用されるサービスには，高度な相互運用性が求められる．このため，サービスの基盤として，インターフェイスを明示的に宣言できる枠組が必要となる．Webサービス をベースとした Web Services Resource Framework (WSRF)[1] は，このような枠組の候補の一つで，WSDL (Web Services Description Language)[2] による明確なインターフェイス定義が可能であり，リソース概念によるサーバ側の状態を分離して管理することができる．

Webサービス規格の多くは，多くのソフトウェアスタックから構成されているため非常に複雑であり，なんらかのツールを使用せずにプログラマが直接プログラミングをすることは難しい．このため，WSRF に対しても複数のツールキットが提案されている．その中でもっとも広く用いられているものが，Globus Toolkit 4[3] である．Globus Toolkit 4 を用いると，比較的容易に WSRF に基づくサービスを構築することができるが，プログラミングモデルが複雑である上，プログラム以外の複数のファイルの記述が求められるため，十分に容易であるとは言いがたい．中でも，WSDL の記述は煩雑で，詳細な知識を必要とするためプログラマにとって大きな障壁となる．

我々はこの点に対処するべく，Globus Toolkit 4 による WSRF サービスの記述を補助するツールを作成した．このツールを用いると，プログラマは Java のソースコードに適切なアノテーションを付加するだけで，WSRF サービスを構成することができる．さらに，通信に用いる XML のフォーマットが外部から与

† 産業技術総合研究所 National Institute of Advanced Industrial Science and Technology (AIST)
†† エス・エフ・シー S.F.C Co., Ltd.

えられている場合にも，これを容易に取り込むことができる．

本稿の構成を以下に示す．2で既存のWebサービス記述補助技術について述べる．3で，WSRFとGlobus Toolkit 4を用いたWSRFサービスの記述について述べる．4に提案補助ツールの設計と実装を示し，5で提案補助ツールを用いたサービスの実装を実例を挙げて示す．6はまとめと今後の課題である．

2. Webサービス記述補助技術

一般に，Webサービスを記述する際には，サービス本体の記述と，WSDLによるサービスインターフェイスの定義の2つを行わなければならない．サービス本体にもインターフェイスの定義がある程度存在するため，この双方を完全に独立に行うことは無駄である．

このため，既存のWebサービス記述の枠組の多くでは，以下の2つの方法のいずれかによって記述が支援されている．

- **WSDL先行型**
  WSDLをプログラマに記述させ，そこからサービス本体のスケルトンを生成する．
- **サービス先行型**
  サービス本体をプログラマが記述し，そこからWSDLを生成する．

Web Servicesの実装として広く用いられているApache Axis[4]には，この双方が実装されている．

Axisでは，WSDL先行型の支援機能として，WSDL2Javaと呼ばれるツールが提供されている．これを用いるとWSDLファイルから，Javaプログラムのスケルトンを生成することができる．プログラマは，このスケルトンを実装する形で，サービスを記述する．この方法の問題点は，非常に煩雑なWSDLファイルをサービスプログラマが直接記述しなければならない点である．

Axisのサービス先行型の機能として，単純なWebサービスをJavaプログラムから自動的に作成する方法が用意されている．特定の書き方でJavaプログラムを作成し，ソースファイルを特定のディレクトリに配置しておくだけで，Web Serviceを配備することができる．Axisは，サービスに対するWSDLの要求に対して，動的にJavaのソースプログラムをコンパイルしてclassファイルを作成し，classファイルからWSDLを生成して返却する．サービス呼出し時には，クライアントとサービスの間でやりとりされるデータ構造を自動的に変換し，サービスプログラマが定義したサービスに渡す．したがって，サービスプログラマはサービスが実際にやりとりするデータの形にとらわれることなくプログラムを記述することができる．

この方法には2つの問題点がある．1つは，Javaのインターフェイス定義とWSDLのインターフェイス定義のギャップである．基本的に，WSDLによるインターフェイス定義は，Javaのそれよりもリッチな枠組である．Javaのインターフェイスから WSDLを生成することは不可能ではないが，もともと存在しない情報をいくつか決め打ちで補うことになる．このため，生成されるWSDLをJavaのソースプログラムから完全に制御することができない．

もう1つの問題は，通信の対象となるデータのスキーマを外部から与えられないことである．グリッド関連では，多くのデータスキーマが規格化されている．これらのデータスキーマはXMLスキーマ(xsd)で定義されているため，容易にWSDLに取り込むことができる．しかし，Java言語からWSDLを生成する方針で行う場合，データスキーマをJavaのクラス定義としてプログラマが記述しなおす必要があるが，これは非常に煩雑である上，一度クラス定義となったものを再度XMLスキーマに変換した場合，オリジナルのXMLスキーマと同一になる保証はない．

3. WSRFとGlobus Toolkit 4

3.1 WSRFの概要

WSRF (Web Services Resource Framework)は，OASIS[5]で標準化が行われたWebサービス規格の一つである．多くのWebサービス規格と同様に，WSDL (Web Services Description Language)[2]による明確なインターフェイス定義が可能である．

WSRFは，さまざまなグリッドプロジェクトで標準的な記述基盤として用いられている．OGF(Open Grid Forum)[6]で作成されているグリッドアーキテクチャOGSA(Open Grid Service Architecture)[7]でも，WSRFをターゲットとしたプロファイルが策定されている．

Webサービスで複雑なサービスを記述する上で問題になる点の一つがサーバ側「状態」の管理である．サーバ側に状態を持つことは不可欠であるにもかかわらず，Webサービスではサービスそのものに，状態を持たせることを嫌う．WSRFでは，複雑なサービスを記述する上で不可欠なサーバ側の「状態」を，**リソース**という概念で明確化し，サービス本体から分離している．そして，サービスとリソースをWS-Addressing[8]規格で定められた記法でひとまとまりとして扱うことで，実質的に「状態を持つサービス」を実現する．

例として，インクリメントが可能なカウンタサービスを考えてみよう(図1)．カウンタには現在の値を示す内部状態が必要になるが，これをリソースとして表現し，サービスの外部に置く．サービスそのものには内部状態を持たない．リソースにはIDが割り当てられ，クライアント側はサービスとこのIDをペアとして保持する．カウンタサービスに対するインクリメント操作の呼出の際には，クライアントはリソースID

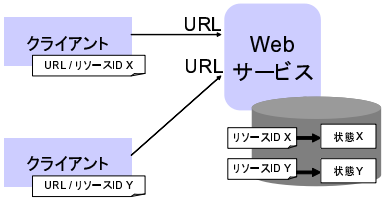

図 1 WSRF におけるリソース管理

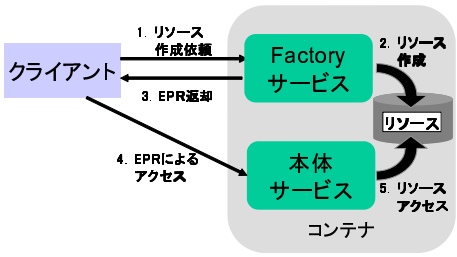

図 2 Factory パターン

をサービスに提示する．サービスはリソース ID を用いてリソースを参照し，インクリメント操作を行う．複数のクライアントがそれぞれ異なるカウンタ値をもつサービスを利用する場合には，サービスではなくリソースを複数作成する．サービス本体は複数のクライアントが共有し，リソースはそれぞれに割り当てられたものを利用する．

リソースの作成には，通常 **Factory サービス**と呼ばれる本体のサービスとは別のサービスを用いる．これを Factory パターンと呼ぶ．クライアントは，まず Factory サービスにアクセスしリソースの作成を依頼する (図 2:1)．Factory サービスは，リソースを作成し (図 2:2)，その ID と本体サービスの URL のペアを WS-Addressing に従った XML 文書として作成し，これをクライアントに返却する (図 2:3)．この XML を EPR (End Point Reference) と呼ぶ．クライアントはこの EPR を用いて本体サービスにアクセスする (図 2:4)．本体サービスは，EPR の中に納められたリソース ID を用いて，リソースを取得 (図 2:5)，処理を行う．

3.2 Globus Toolkit 4

Globus Toolkit 4 は，WSRF のコア機能を提供するモジュールとツール群，WSRS コアを利用して構成されたグリッド向けサービス群, および，少数の非 WS コンポーネントから構成される．WSRS によるグリッド向けサービス群には，ジョブ起動 (GRAM4), 情報サービス (MDS4)，ファイル転送管理 (RFT) などが含まれる．さらに Globus Toolkit 4 は，プログラマが定義する WSRF サービスのプラットフォームとしての機能と，プログラマによる WSRF サービスの作成環境としての機能を持つ．WSRF コア部分は，Java 言語を用いて Apache Axis[4] を利用して実装されている．WSRF サービス群は，サービスコンテナを

表 1 Globus Toolkit 4 による WSRF サービス記述に必要なファイルと行数

	説明	言語	行数
1	Factory サービス	Java	51
2	本体サービス	Java	50
3	リソース	Java	73
4	リソースホーム	Java	20
5	QName	Java	16
	小計		210
6	Factory サービスの WSDL	WSDL	71
7	本体サービスの WSDL	WSDL	119
	小計		190
8	サービスのデプロイファイル	WSDD XML	27
9	デプロイ時の JNDI 設定ファイル	JNDI XML	37
	小計		64
	計		464

```
public class Math {
  private int internal;
  public int add(int val){
    internal += val;
    return internal;
  }
  public int sub(int val) {
    internal -= val;
    return internal;
  }
}
```

図 3 サンプルプログラムのロジック

共有する．すなわち，全てのサービスが，単一の Java VM 上でスレッドとして稼働する．

文献[9] にしたがって，Globus Toolkit 4 による WSRF サービスの記述方法を見てみよう．Globus Toolkit 4 は WSDL 先行型の記述補助機能を備えている．Factory パターンに従った一般的な WSRF サービスの記述には下記が必要となる．

(1) WSDL によるインターフェイスの記述
(2) Java によるソースファイル群
(3) デプロイ用設定ファイル群

文献[9] で示されている Factory パターンを用いたサンプルサービスの場合のファイルと行数を表 1 に示す．このサンプルサービスは，整数値の状態を持ち，その値に対する加算と減算を定義しただけのものであり，通常の Java 言語で記述した場合には，図 3 に示すように，10 行強で記述できる程度のロジックしか持たない．にもかかわらず，9 つのファイルにまたがって 464 行もの記述が必要であった．インターフェイスを定義しているだけの WSDL に，合わせて 190 行もの記述が必要なことに注意してほしい．

このように Globus Toolkit 4 を用いて WSRF サービスを記述するには，プログラム以外に大量の記述をおこなわなければならない．特に煩雑な WSDL ファイルの記述はプログラマにとって大きな負担となる．

もう一つの問題点は，サービスとリソースが分離していることでロジックが煩雑になっているとである．

```
// サービス
public class MathService {
  private MathResouce getResouce(){
    return (MathResouce) ResouceContext.
        GetResouceContext().getResouce();
  }

  public AddResponse add(int val){
    MathResouce mr = getResource();
    mr.setValue(mr.getValue() + val);
    return new AddResponse(mr.getValue());
  }
  ...
}

// リソース
public class MathResource {
  private int value;
  public int getValue(){
    return value;
  }
  public void setValue(int value){
    this.value = value;
  }
}
```

図 4 Globus Toolkit 4 におけるサービスとリソースの疑似コード

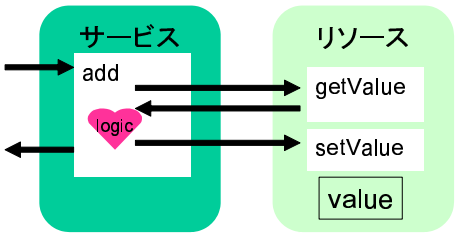

図 5 Globus Toolkit 4 におけるサービスとリソース

図 4 に サービスとリソースの疑似コードを示す．リソースにはデータとそのアクセサメソッドのみが定義されており，サービス側には，リソースのアクセサを利用して内部状態を更新するロジックが書かれている．図 5 に示すように，ロジックとデータが分離されているのである．この実装は，WSRF のコンセプトを直接的に反映したものではあるが，プログラマにとっては無用に煩雑であることは間違いない．

4. 補助ツールの設計と実装

4.1 設計の指針

前節で見た Globus Toolkit 4 の問題点に基づき，以下にしめす補助ツール設計の指針を得た．

- プログラマが記述しなければならないファイルの数，行数を削減する．特に煩雑な WSDL をプログラマに書かせることは避ける．
- サービスとリソースが分離しているのは煩雑であるので，マージする．
- アクセサメソッドのように単純なメソッドは自動生成する．
- データ構造が XML スキーマで定義されている場合，これを利用可能にする．

4.2 設　　計

上述の設計指針に基づき，補助ツールを設計した．基本方針としては，Java 1.5 で導入された，アノテーション[10] を利用することで，一つのソースファイルに多くの情報を付加し，その情報を用いて，他のファイル群を生成することとした．

表 1 に示したように，Globus Toolkit 4 でサービスを記述するためには，9 つのファイルが必要となる．このうち，デプロイ関連のファイル (8,9) は定型的であるため，いくつかの情報が揃えば比較的容易に生成することができる．Factory 関連の WSDL(6)，Java コード (1)，リソースホーム (4) も定型的である．QName ファイル (5) はプログラマの便宜上別ファイルになっているだけなので省略することができる．残るファイルは，本体サービス (2)，リソース (3)，本体の WSDL(7) の 3 つである．

まず，サービスとリソースについて考える．Globus Toolkit 4 ではプログラマがサービスとリソースの双方を記述しなければならない．これを避けるため，サービスとリソースの役割分担を変更し，リソース側に機能を集中させることにした (図 7)．すなわち，従来サービス側に存在したオペレーションのロジックはリソースに実装され，サービスはリソースに定義されたオペレーション (メソッド) をただ呼び出す．サービスは一種のプロキシオブジェクトとなるため，容易に自動生成が可能になる．プログラマはリソースのみを実装すればよい．

また，リソースのアクセサは，プロパティとなるメンバ変数がわかれば容易に生成できる．プロパティとするべきメンバ変数を明示するために，アノテーション`PropertyAN` を導入し，自動生成を行う．

図 6 下部に，本ツールを用いた場合のリソースの定義を示す．図 4 下部のリソースと比較すると，アクセサメソッドがなく，かわりに `add`, `sub` の両オペレーションが実装されていることがわかる．アクセサメソッドは，自動生成が可能なため，省略されている．図 6 上部に自動生成されるサービスの疑似コードを示す．リソースに対して`add` メソッドを呼び出すだけの，アプリケーションロジックを持たない定型的なコードになっているため，自動生成が可能なのである．

次に WSDL ファイルについて考える．WSDL ファイルはサービスのインターフェイスを定義したもので，リソースクラスに定義されたメソッド群から生成することができる．しかし，リソースクラスには，サービスのインターフェイスとして公開する必要のないメソッドが含まれている可能性がある．このため，アノテーション `OperationAN` を用意し，サービスのオペレーションとして公開するべきメソッドをプログラマに明示させることにした．

以上のように，リソースの定義ファイルにオペレーションをまとめて記述し，さらにアノテーションを適

```
// サービス（自動生成）
public class MathService {
  ...
  public AddResponse add(int val){
    MathResouce mr = getResource();
    return new AddResponse(mr.add(val));
  }
  ...
}

// リソース（ユーザが定義）
public class MathResource {
  @PropertyAN
  private int internal;

  @OperationAN
  public int add(int val){
    internal += val;
    return internal;
  }
  :
}
```

図 6 補助ツールを利用したリソースの記述

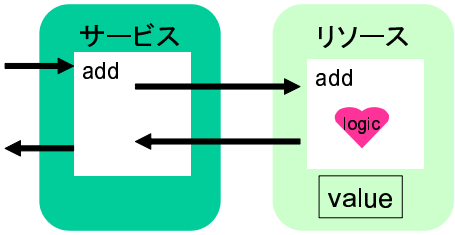

図 7 本ツールを用いた場合のサービスとリソース

切に付加することによって，他の 8 つのファイルを自動生成または省略することができる．

データ構造が XML スキーマで定義されている場合の対処としては，XML スキーマから Java へのバインディングを生成する機構と，XML スキーマを自動生成される WSDL に取り込む機構を提供する．

4.3 実　　装

提案ツールは以下の 4 つのモジュールから構成される．

(1) リソースから他のファイル群を生成するツール
(2) XML スキーマからバインディングを生成するツール
(3) リソースにアクセサメソッドを動的に付加するリソースホーム
(4) ビルドプロセスを制御する ant 用 build ファイル

(1) は，リソースのクラスファイルを解析し，クラスのインターフェイス情報とアノテーションとして付加された情報を用いて他のファイルを生成する．アノテーションの処理には，バイトコードレベルでのクラスファイルの操作を実現するライブラリである，Javassist[11),12)] を用いた．

(2) は，対象となる XML スキーマをダミーの WSDL ファイルでラップし，そに対して Globus Toolkit 4 の WSDL からスケルトンファイルをつくり出す機能を適用することで実現している．ダミーの

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://gtrc.aist.go.jp/test2"
  targetNamespace="http://gtrc.aist.go.jp/test2">
  <xsd:complexType name="ComplexType">
    <xsd:sequence>
      <xsd:element name="imaginal" type="xsd:float"/>
      <xsd:element name="real" type="xsd:float" />
    </xsd:sequence>
  </xsd:complexType>
  <xsd:element name="Complex" type="ComplexType" />
</xsd:schema>
```

図 8 複素数の XML スキーマ

WSDL ファイルを生成する部分は，Python のスクリプトとして実現されている．

(3) は，リソースのクラスが Globus Toolkit 4 のコンテナにロードされるタイミングで，バイトコードを動的に改変し，アクセサメソッドを付加する．実装には Javassist を用いた．

5. 補助ツールによる WSRF サービスの実装例

5.1 サービスの記述

本補助ツールを用いて WSRF サービスを実装する例を示す．対象となるサービスは，図 3 のものとほとんど同じだが，通信対象となるデータ型が int ではなく複素数を表す複合型で，しかも XML スキーマが外部から与えられているものとする．複素数の XML スキーマを図 8 に示す．

このスキーマを利用してプログラムを書くには，まずこのデータ型の Java バインディングを生成する必要がある．4.3 の (2) に示したツールを用いてバインディングを生成する．

次にこのバインディングを用いて，サービスのプログラムを記述する．この際にプロパティとして公開するインスタンス変数，およびオペレーションとして公開するインスタンスメソッドに，それぞれ，`PropertyAN`, `OperationAN` アノテーションを付加する．図 9 に記述したサービスを示す．

基本的にプログラマが記述するファイルはこれだけである．後は，本ツールが提供する build ファイルを用いて，ビルドを行うと，必要なファイルがすべて生成され，Globus Toolkit 4 コンテナにデプロイするためのパッケージが生成される．

5.2 議　　論

ここで示したサービスは，機能においては 3.2 で示したものとほぼ同じであり，複雑さという点では，整数ではなく構造データである複素数を処理しているため，より複雑であるといえる．にもかかわらず，わずか 21 行の Java コードの記述によって実装が達成できている．3.2 のサンプルでは，464 行であったことを考えると大幅な簡易化が達成できているということがいえる．

```
public class ComplexResource {
  @PropertyAN
  private ComplexType complex =
          new ComplexType(0.0f, 0.0f);

  @OperationAN
  public ComplexType add(ComplexType addValue) {
    complex = new ComplexType(
     complex.getImaginal() + addValue.getImaginal(),
     complex.getReal()     + addValue.getReal());
    return complex;
  }

  @OperationAN
  public ComplexType sub(ComplexType subValue) {
    complex = new ComplexType(
     complex.getImaginal() - subValue.getImaginal(),
     complex.getReal()     - subValue.getReal());
    return complex;
  }
}
```

図 9 アノテーション付きサービスの例

6. おわりに

Globus Toolkit 4 による WSRF サービスの記述を補助するツールを設計，実装した．このツールは，サービス先行型の欠点をアノテーションによって補い，さらに WSDL 先行型の機能を一部取り込んでいる．サービスプログラマはアノテーション付きの Java のソースコードを用意するだけで，WSDL を記述することなく WSRF のサービスを作成することができる．さらに，サービスのインターフェイスで用いられるデータ型の XML スキーマが与えられている場合にはそれをそのまま利用して WSRF サービスを記述することができる．

本ツールは現在プロトタイプであり，いくつかの問題点がある．

- 現在実装されているアノテーションは，2 種類であり，Java 言語のインターフェイスと WSDL のインターフェイスのギャップを埋めるために十分であるとは言いがたい．今後，さまざまな例でテストを繰り返し，必要に応じてアノテーションを追加していく．
- Globus Toolkit 4 の特徴の一つに Operation Provider がある．これは，オブジェクト指向言語の世界でいうところの mixin[13] のような機能を提供するモジュールで，さまざまな機能を任意のサービスに追加することができる．例えば，リソースのライフタイムサイクルを管理する機能は，いくつかの Operation Provider として実装されている．本ツールでは現在 Operation Provider を追加することができないため，この機構の恩恵を受けることができない．サービスクラス本体にアノテーションを付加することにより，Operation Provider を指定できるようにすることを検討中である．
- 本ツールでは，サービス内部で生じた例外に対する処理が十分ではない．WebServices では Fault のタイプを定義し，エラーに対応した Fault を返すことができるが，本ツールの現状の実装では，すべて RemoteException という型のエラーとしてクライアントに返されてしまう．これに対しては，サービス内で生じる例外と Fault の対応を定義するアノテーションを追加して，対処したい．
- Java 言語には Web サービス向けのアノテーションの規格が存在する[14]．我々のツールは状態を持つ WSRF を指向しているためそのまま利用することはできないが，応用を検討したい．

謝　辞

本研究の一部は，文部科学省科学技術振興調整費「グリッド技術による光パス網提供方式の開発」による．

参考文献

1) Web Services Resource Framework. http://www.oasis-open.org/committees/wsrf
2) Web Services Description Language (WSDL). http://www.w3.org/TR/wsdl.
3) Foster, I.: Globus Toolkit Version 4: Software for Service-Oriented Systems, *IFIP International Conference on Network and Parallel Computing, Springer-Verlag LNCS 3779*, pp. 2–13 (2005).
4) Web Services, Axis. http://ws.apache.org/axis/.
5) OASIS. http://www.oasis-open.org/.
6) Open Grid Forum. http://www.ogf.org.
7) Foster, I., Kesselman, C., Nick, J. M. and Tuecke, S.: *Grid Computing: Making the Global Infrastructure a Reality*, John Wiley & Sons Ltd, chapter The Physiology of the Grid (2003).
8) Web Services Addressing. http://www.w3.org/Submission/ws-addressing/.
9) The Globus Toolkit 4 Programmer's Tutorial, http://gdp.globus.org/gt4-tutorial/.
10) A Metadata Facility for the Java^{TM} Programming Language, Java Community Process JSR-000175.
11) Javassist. http://www.csg.is.titech.ac.jp/chiba/javassist/.
12) Chiba, S.: Load-time Structural Reflection in Java, *ECOOP 2000 – Object-Oriented Programming, LNCS 1850*, Springer Verlag, pp. 313–336 (2000).
13) Bracha, G. and Cook, W.: Mixin-based inheritance, *Proc. of OOPSLA/ECOOP 1990*, pp. 303–311 (1990).
14) Web Services Metadata for the Java^{TM} Platform, Java Community Process JSR-000181.