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

1.産業技術総合研究所、2.エス・エフ・シー

中田秀基[1]、竹房 あつ子[1]，岸本誠[1,2]

工藤 知宏[1]、田中 良夫[1]、関口 智嗣[1]

グリッドとWSRF

- グリッド
  - 複数の管理主体にまたがるVO(仮想組織)による資源の共有と有効利用
  - 明示的，宣言的なインターフェイス記述が必要
- Web Services Resource Framework (WSRF)
  - Web Services をベースに，サーバ側の状態を明示的に取り扱う枠組み
  - WSDL (WS Description Language) による明示的なインターフェイス記述

WSRFサービス記述の問題

- Web Services – 多階層
  - 何重にも積み重ねられたソフトウェアスタック
  - ツールの補助なしにスクラッチから記述するのはほぼ不可能
- Globus Toolkit 4
  - WSRFサービスを記述するための代表的なフレームワーク
  - ○ 抽象度の高い記述が可能
  - × 数多くの非本質的なファイルの記述が必要

研究の目的

- WSRF サービスの記述を容易にするツールの提供
  - Globus Toolkit 4 を補助するツールを作成
    - ユーザはアノテーション付きの Java のソースを記述するだけ
    - 他のファイルはJavaソースから自動生成
  - 転送データがXMLスキーマが既定の場合にも対応

発表の概要

- Web Services と WSRF の概要
- Globus Toolkit 4 の概要
- 提案ツールの設計と実装
- 議論
- まとめと今後の課題

（狭義の）Web Services

- SOAP/HTTP をベースにした通信
  - XML による拡張性の高い形式
  - いくつもの独立した規格による膨大なプロトコルスタック
- WSDL (Web Services Description Language)
  - サービスのインターフェイスを記述
  - 転送データのスキーマも定義

- ○ 明確にインターフェイスが定義できるため
  相互運用性を確保しやすい
- × プロトコルスタック による複雑さ
  - 低速
  - 記述にはツールのサポートが必須

Web Services の記述補助 (1)

- Web Services の実装
  - 実装言語(Javaなど)によるサービス本体
  - WSDLによるインターフェイス定義
    - 非常に煩雑
    - 内容の一部は，サービス本体の記述と重複
      - 整合性の確保
- 転送データのマーシャリング・アンマーシャリング
  - 実装言語のオブジェクトと転送データ形式（XML表現）との間の変換
  - 実装言語の型とXMLスキーマの対応は自明ではない

Web Services の記述補助 – 記述補助ツール

- WSDL先行型： WSDL → 記述言語
  - WSDLから記述言語によるスタブを作成，
    - ex. wsdl2java
  - 転送データのオブジェクトもWSDLから生成
  - ○ サービスのインターフェイスであるWSDLを完全に制御できる
  - × 煩雑なWSDLをまず書かなければならない

- サービス先行型： 記述言語 → WSDL
  - 記述言語からWSDLを構成： ex. java2wsdl
  - ○ WSDL を書かないですむ
  - × オブジェクトを特殊な形で書く必要がある
    - Java Beans
  - × WSDLの内容が完全に制御できない
    - WSDLと言語の間のセマンティクスギャップ

WSRF (Web Services Resource Framework)

- Web Service でサーバ側の状態を統一的に記述するためのフレームワーク
  - 「リソース」という概念を導入
  - サービスそのものは状態を持たない
  - サービスとリソースを一組として扱うことで実質的に「状態をもつサービス」を実現
  - c.f. Open Grid Service Infrastructure
    - サービスそのものに状態を付加

- Open Grid Forum で提案され，OASISで標準化

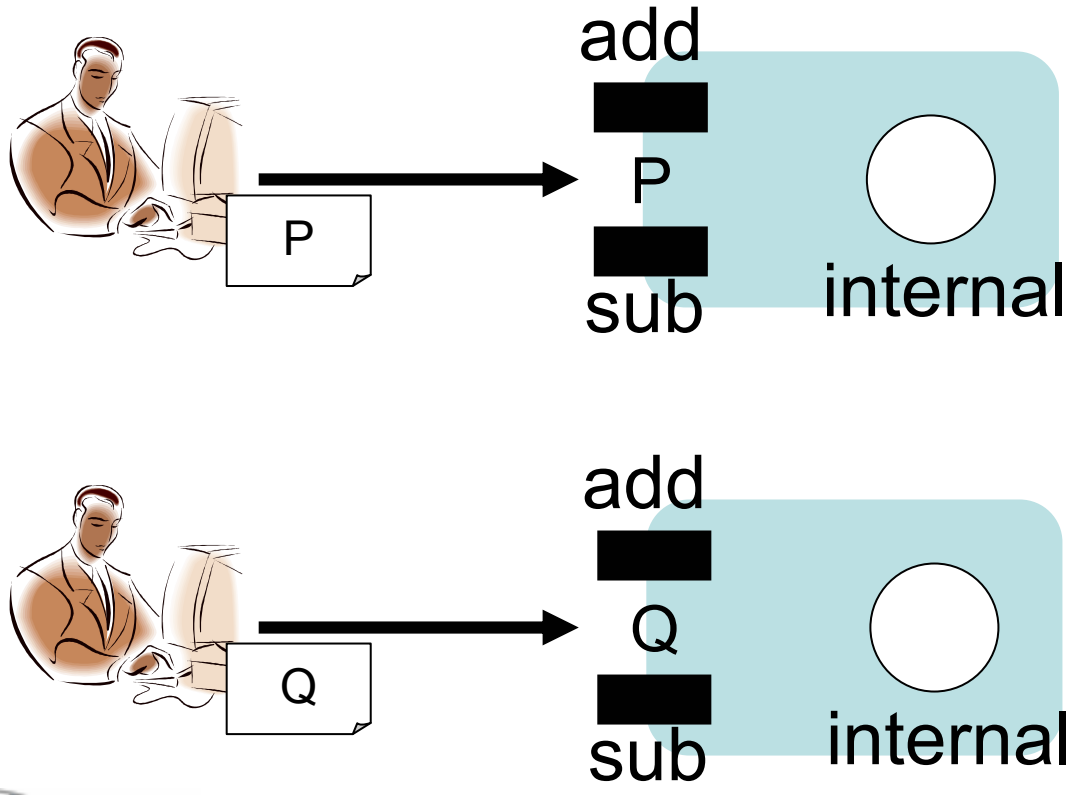
サンプルサービス

- 簡単なカウンタサービス
  - 内部状態としてint値を持つ
  - その値に対して加算と減算が可能

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

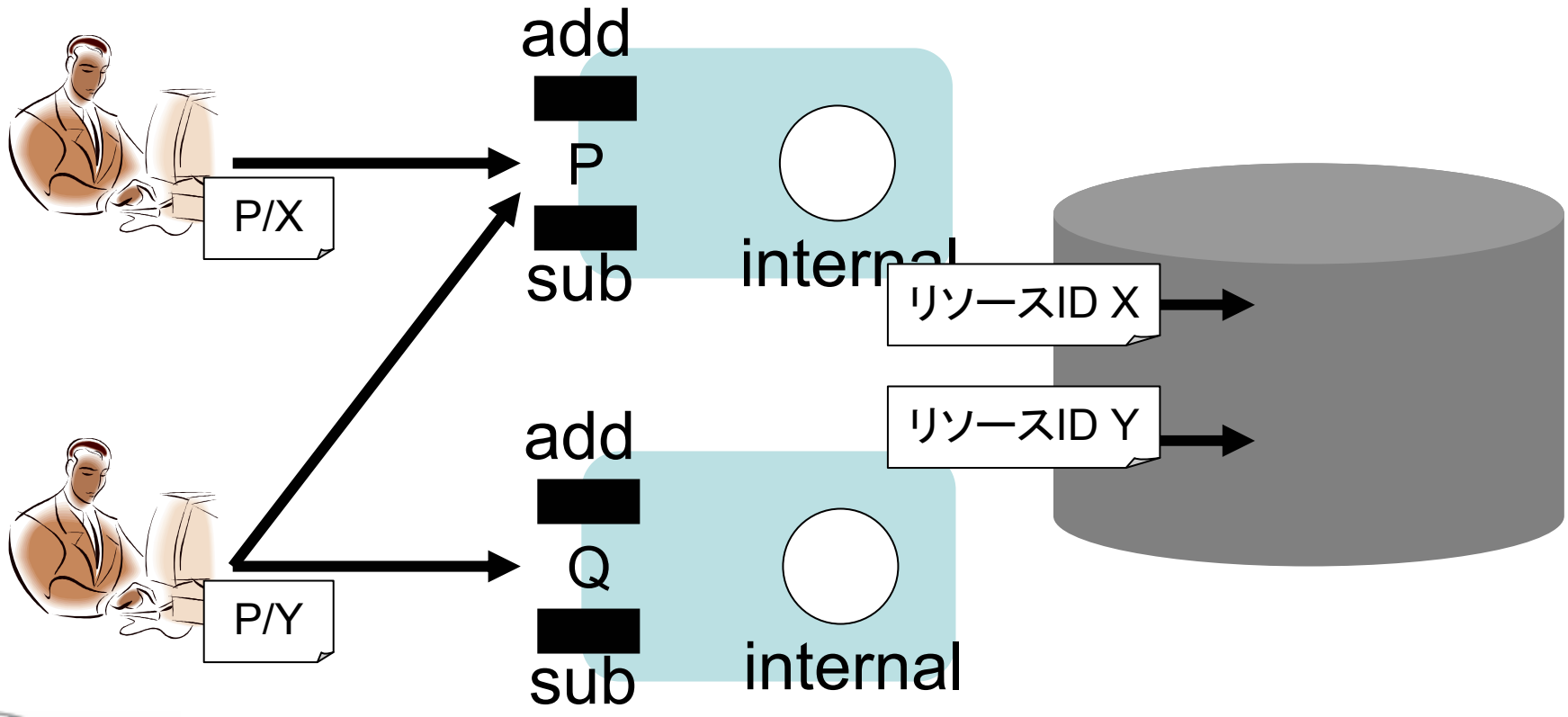
WSRF – サービスとリソースの分離

- サービス = オペレーション
- リソース = データ



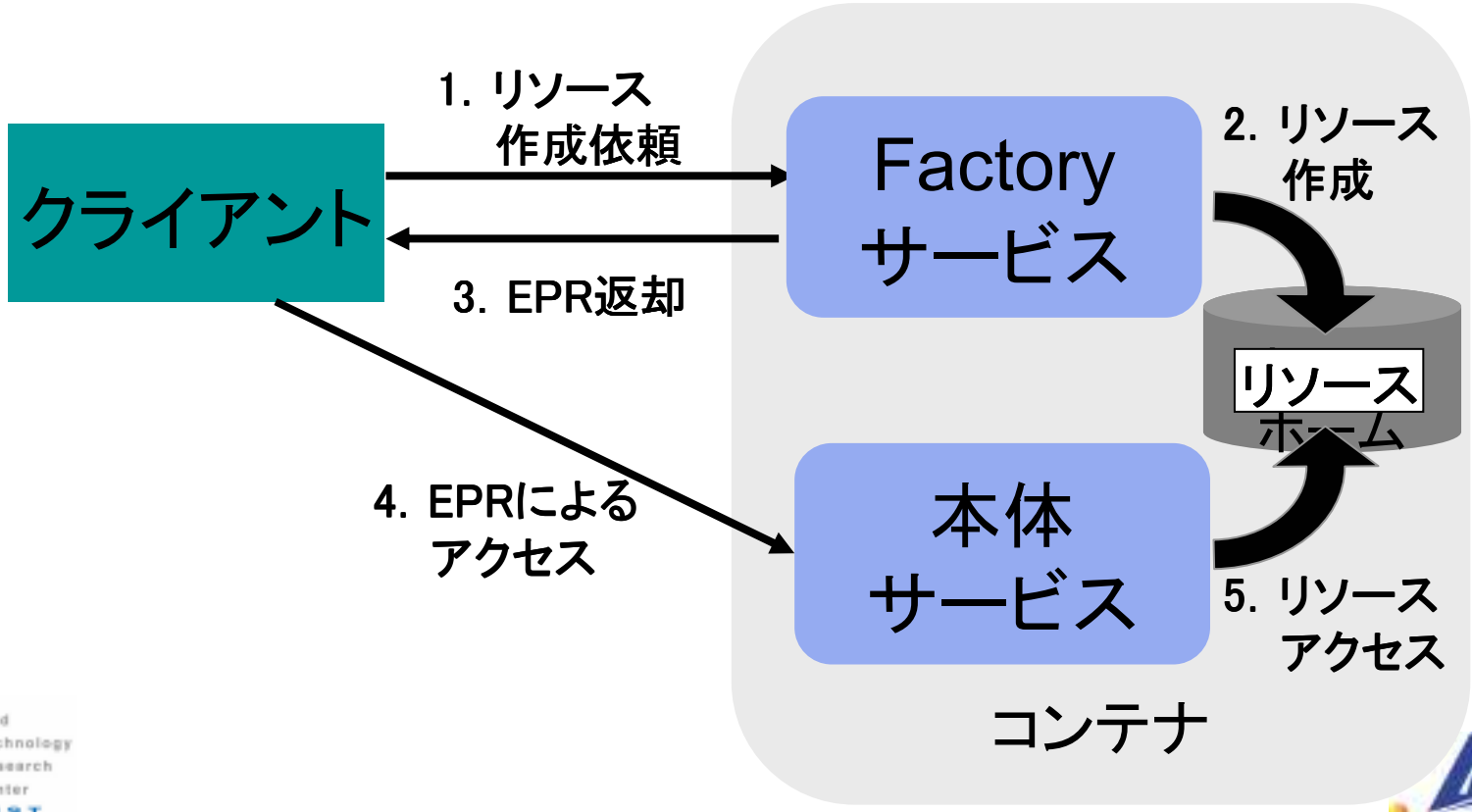
WSRF – サービスとリソースの分離

- サービス = オペレーション
- リソース = データ



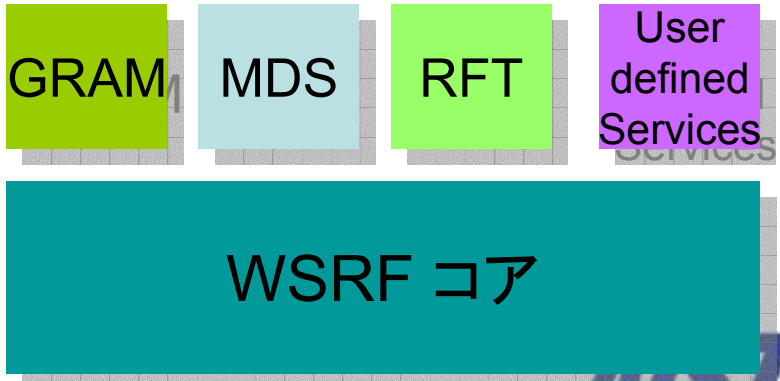
Factory サービスパターン

- Factory サービス
  - リソースを背後のリソースホーム内に作成



Globus Toolkit 4

- 広く用いられているグリッドのツールキット
  - ver. 1/2 – 独自プロトコル
  - ver. 3 – OGSI ベース
  - ver. 4 – WSRF ベース
- WSRF 環境
  - Java によるサービスコンテナ
  - 開発用のツール
- WSRFで実装されたサービス
  - ジョブ起動GRAM
  - 情報サービス
  - ファイル転送サービス

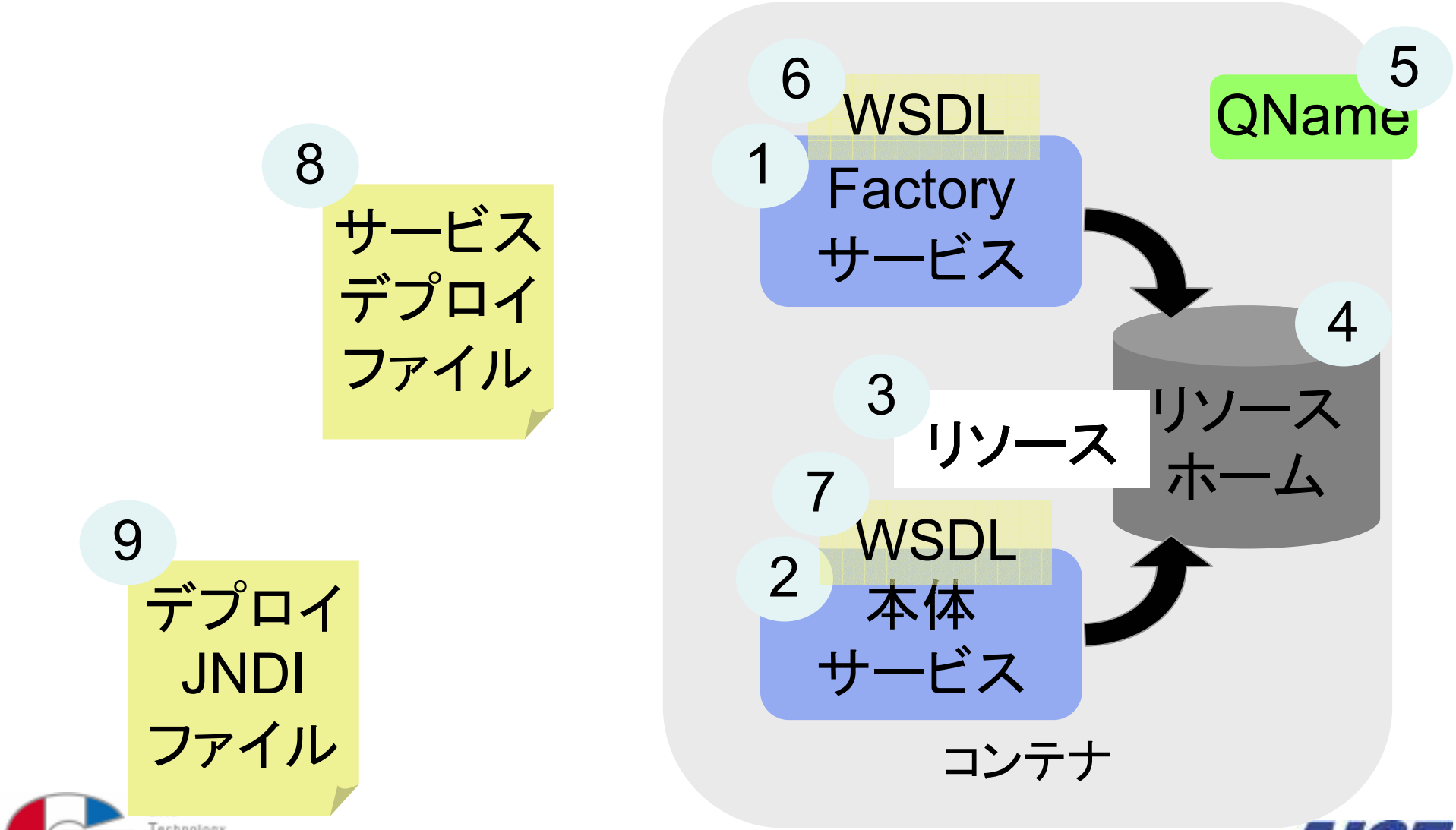


GT4 によるWSRFサービスの記述

- WSDL先行型の記述補助
  - WSDL を書くと転送データのJavaオブジェクトを生成
  - さまざまな基本サービスを提供
    - Operation Provider

- GT4 Tutorial [http://gdp.globus.org/gt4-tutorial/]

GT4によるWSRFサービス記述に必要なファイル



Technology Research Center AIST

AIST

GT4によるWSRFサービス記述に必要なファイル

説明	記述言語	行数
Factoryサービス	Java	51
本体サービス	Java	50
リソース	Java	73
リソースホーム	Java	20
QName	Java	16
小計		210
Factoryサービスインターフェイス	WSDL	71
本体サービスインターフェイス	WSDL	119
小計		190
サービスのデプロイファイル	WSDD XML	27
デプロイ JNDI 設定ファイル	JNDI XML	37
小計		64
計		464

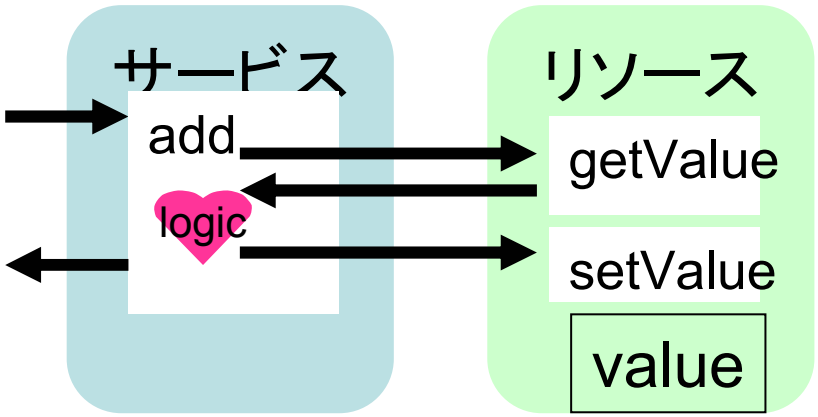
GT4によるWSRFサービス記述の問題点

- サービスとリソースの双方を書かなければならない
  - オブジェクト指向的には単一のオブジェクトなのに独立した2つのオブジェクトとして実装
- リソースにはアクセスメソッドが必要
  - Bean のsetter/getter
- WSDLのインターフェイス記述は煩雑
  - 非常に複雑で専門知識が必要
- その他の記述ファイルが多すぎる
  - 定型的なファイルも多いが手作業ではミスがでる
    - エラーから記述ミスを発見するのも大変

GT4を用いた場合のプログラムの概要

```
// サービス
class MathService {
  private MathResource getResource(){
    return (MathResource) ResourceContext.
        getResourceContext().getResouce();
  }

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

    return new AddResponse(tmp);
  }

  ...

}
```

```
// リソース
class MathResource {
  private int value;

  public int getValue(){
    return value;
  }

  public void
   setValue(int value){
    this.value = value;
```



WSDLの例

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="MathService"

        targetNamespace="http://www.globus.org/namespaces/examples/core/MathService_i
        nstance_rp"
    xmlns="http://schemas.xmlsoap.org/wsdl/"

        xmlns:tns="http://www.globus.org/namespaces/examples/core/MathService_instanc
        e_rp"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsrp="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-
        1.2-draft-01.xsd"
    xmlns:wsrpw="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-
        1.2-draft-01.wsdl"
    xmlns:wsdlpp="http://www.globus.org/namespaces/2004/10/WSDLPreprocessor"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<wsdl:import
    namespace=
    "http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-
        01.wsdl"
    location="../../wsrf/properties/WS-ResourceProperties.wsdl" />

<types>
<xsd:schema
        targetNamespace="http://www.globus.org/namespaces/examples/core/MathService_i
        nstance_rp"

        xmlns:tns="http://www.globus.org/namespaces/examples/core/MathService_instanc
        e_rp"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">


        <!-- Requests and responses -->

        <xsd:element name="add" type="xsd:int"/>
        <xsd:element name="addResponse">
                    <xsd:complexType/>
        </xsd:element>

        <xsd:element name="subtract" type="xsd:int"/>
        <xsd:element name="subtractResponse">
                   <xsd:complexType/>
        </xsd:element>

        <!-- Resource properties -->

        <xsd:element name="Value" type="xsd:int"/>
        <xsd:element name="LastOp" type="xsd:string"/>

        <xsd:element name="MathResourceProperties">
        <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element ref="tns:Value" minOccurs="1" maxOccurs="1"/>
                        <xsd:element ref="tns:LastOp" minOccurs="1" maxOccurs="1"/>
                    </xsd:sequence>
        </xsd:complexType>
        </xsd:element>

</xsd:schema>
</types>
```

```
<message name="AddInputMessage">
       <part name="parameters" element="tns:add"/>
</message>
<message name="AddOutputMessage">
       <part name="parameters" element="tns:addResponse"/>
</message>

<message name="SubtractInputMessage">
       <part name="parameters" element="tns:subtract"/>
</message>
<message name="SubtractOutputMessage">
       <part name="parameters" element="tns:subtractResponse"/>
</message>

<portType name="MathPortType"
    wsdlpp:extends="wsrpw:GetResourceProperty
                    wsrpw:GetMultipleResourceProperties
                    wsrpw:SetResourceProperties
                    wsrpw:QueryResourceProperties"
    wsrp:ResourceProperties="tns:MathResourceProperties">

        <operation name="add">
                    <input message="tns:AddInputMessage"/>
                    <output message="tns:AddOutputMessage"/>
        </operation>

        <operation name="subtract">
                    <input message="tns:SubtractInputMessage"/>
                    <output message="tns:SubtractOutputMessage"/>
        </operation>

</portType>

</definitions>
```

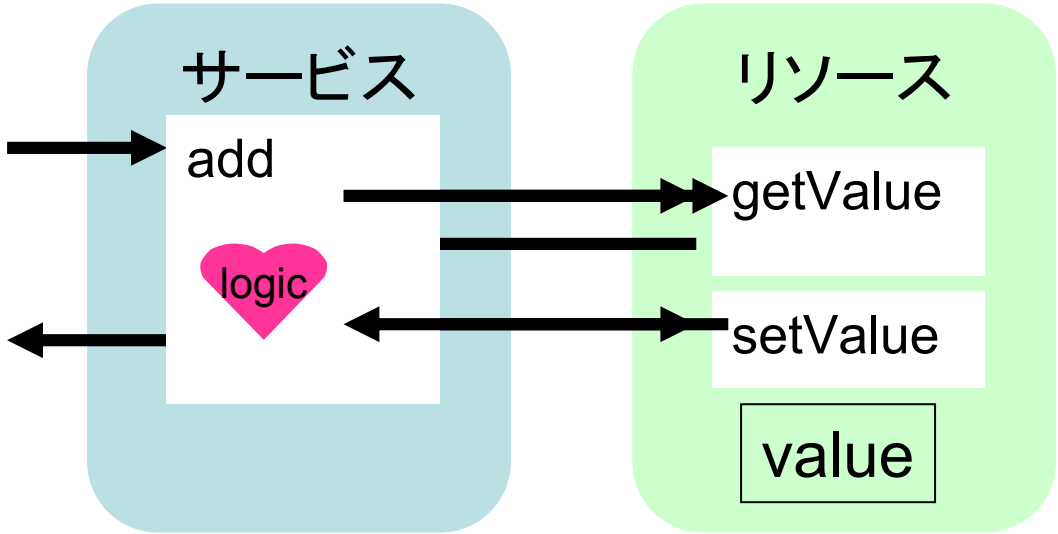
補助ツールの設計指針

- サービスとリソースの双方を書かなければならない
  - リソースのみをユーザに記述させる
  - リソースにロジックを移動
  - サービスはりソースに対するラッパとして自動生成
- リソースにはアクセスメソッドが必要
  - ロード時に動的に付加
- WSDLのインターフェイス記述は煩雑
  - ユーザの定義したコードから自動生成
  - 欠落したセマンティクスはアノテーションで補う
- その他の記述ファイルが多すぎる
  - すべて自動生成

サービスとリソースの融合

- サービスはただのラッパに
- リソースにデータとオペレーションを記述

- 単純になったサービスは自動生成可能



補助ツールを用いた記述

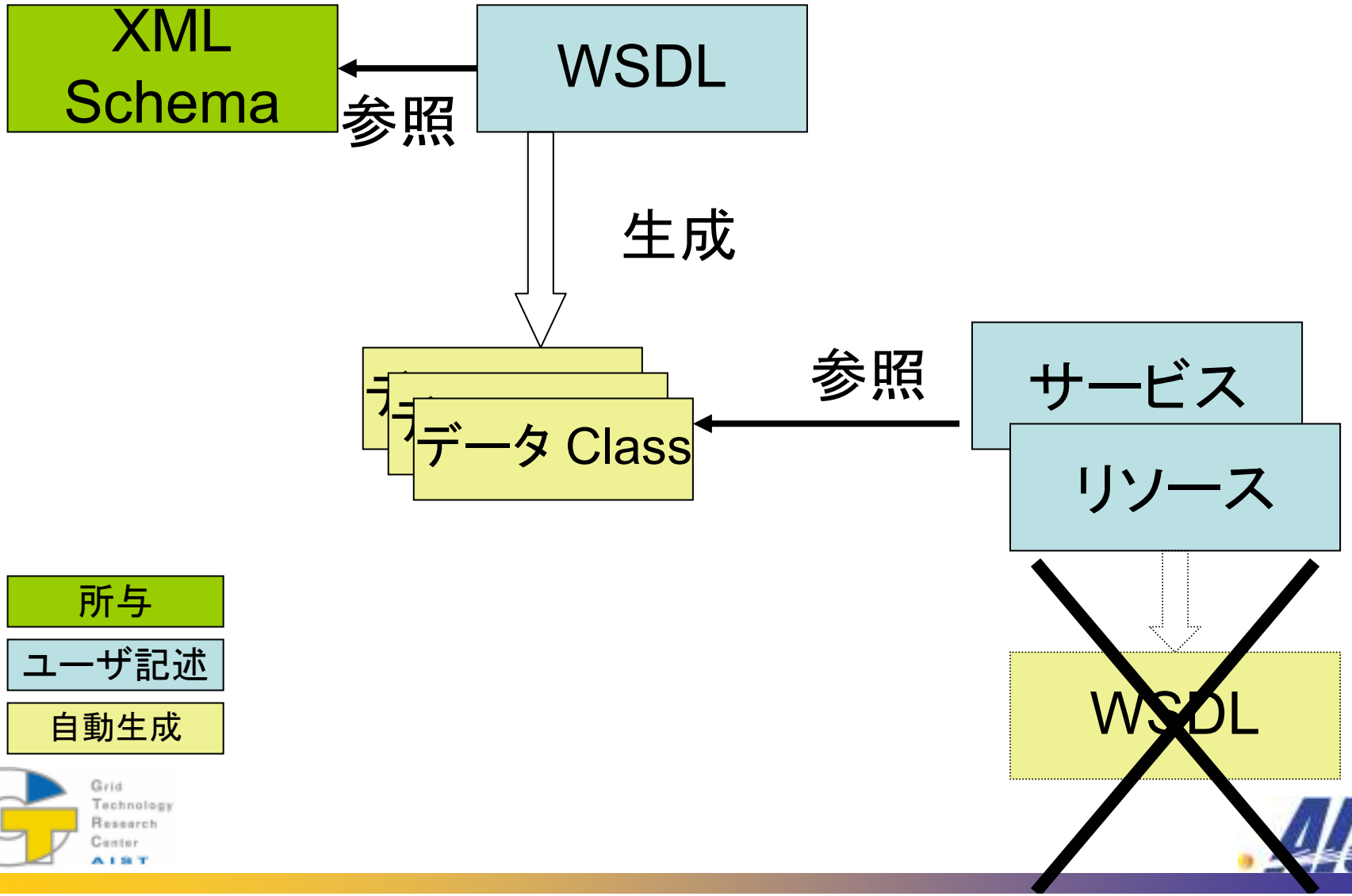
```
// サービス (自動生成)
public class MathService {
   ...
   public AddResponse
   add(int val){
     MathResouce mr =
   getResource();
     return new
   AddResponse(mr.add(val)
   );
   }
   ...
}
```

```
// 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;
   }
   :
}
```

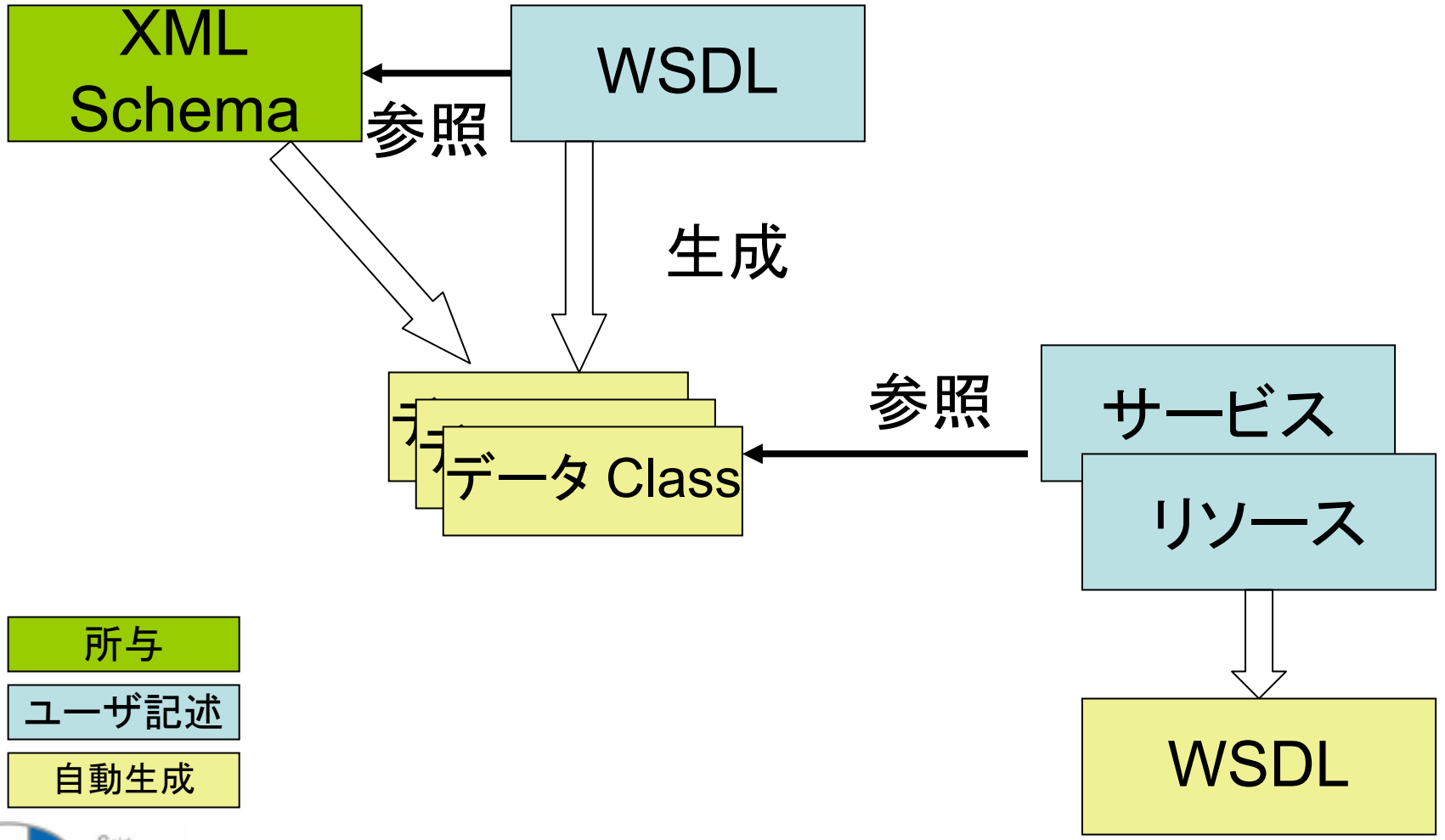
補助ツールの設計

- WSDL の記述は困難
  - Javaのソースから自動生成
  - 問題： Javaのソースだけでは情報が足りない
- Javaのアノテーションを利用
  - オペレーションとして公開するメソッド
  - リソースとして公開するインスタンス変数

GT4での XML schemaの利用



補助ツールでの XML schemaの利用



補助ツールの実装

- Javassist [Chiba ‘00]を利用
  - バイトコードレベルでJavaプログラムを操作
  - アノテーションの解析
  - リソースへのsetter / getterメソッドのロード時追加

提案ツールによるサービスの記述例

- 加算・減算の可能なカウンタ
  - 整数でなく，複素数を対象
  - 複素数のデータ構造は，XMLスキーマによって与えられる.
- ツールの使用法
  - 複素数のXMLスキーマから，Javaクラスを生成
  - 生成されたJavaクラスを活用してリソースを記述
  - リソースからサービス，WSDLなどを生成

複素数のXMLスキーマ

```
<?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>
```

```
public class ComplexType
  implements java.io.Serializable {
    private float imaginal;
    private float real;

    public ComplexType(float imaginal,
                       float real) {
            this.imaginal = imaginal;
            this.real = real;
    }
    public float getImaginal() {
        return imaginal;
    }
    ...
}
```

提案ツールによるサービスの記述

```
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;
  }
}
```

議論

- カウンタサービス
  - 9ファイル 472行 → 1ファイル 20行程度に
  - 大幅な省力化
  - 非専門家でも記述が十分に可能

- XMLスキーマが事前に定義されている場合
  - 容易にJavaクラスを生成
  - WSDLを書かずに利用可能

結論

- Globus Toolkit 4によるWSRFの記述をアノテーションによって補助
  - Java ソースコードだけでWSRFサービスの記述
    - 大幅な記述量の低減を実現
  - データ構造XMLスキーマで定義されている場合にも対応

今後の課題

- 転送データがすでにJava定義されている場合への対応
  - WSDLのためのXMLスキーマの生成
  - マーシャリング・アンマーシャリング機能を動的に追加
- Operation Provider への対応
  - Operation Provider – GT4の機能プラグイン機構
  - アノテーションを追加することで対応
- 例外状態への対応
  - Web Services ではFaultTypeの定義が可能
  - 現状 – すべて同じデフォルトのFaultType
  - プログラマがアノテーションで定義可能に
- Web Services Metadata for the Java Platform (JSR-000181) への対応
  - WSDL の持つ詳細な表現への対応

謝辞

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