

GlobusWorld2004 と WS-Resource Framework の提案

稚内北星学園大学

丸山不二夫

今年の1月20日から23日にかけてサンフランシスコで行われたGlobusWorld 2004の最大の話
題は、Globus と IBM と HP が共同で、WSRF (WS-Resource Framework) を提案したことです。今回は、
予定を変更して、この話題を取り上げたいと思います。

最初の驚き

Gridの標準化はGGFで行われており、GGFの中心的な担い手がGlobusだと筆者は考えていたの
ですが、20日の発表は筆者には衝撃的でした。なぜなら、GGFの中核部隊と思っていたGlobusと
IBMが、GGFの外部でGrid標準化の提案をするという形式で、しかも内容的には、GGFの進めてい
たGrid標準化の二つの大きな要素であるOGSAとOGSIのうちの、OGSIを他のものに取り替えよ
うという提案をしたのですから。いったい、何が起きているのだろうか？ GGFを中心としたGrid
の標準化はどうなるのだろうかというのが、率直な感想でした。

まずは、FAQが面白い

これだけの大きな動きですので、当然といえば当然ですが、Globusには、きちんと説明しなけれ
ばという意識はあったようです。前述の共同声明だけでなく、新しい仕様のドラフトを含めて基
本的な文書類は比較的良好に用意されていました。(もちろん、その内容がすぐに理解できるかど
うかは別の問題ですが) その中でも、筆者が面白いと思ったのは、I. Fosterらが書いたという、
今回の問題に対するFAQです。以下、FAQのポイントを紹介していきたいと思います。このFAQに
は、Globus側の主張だけでなく、今回の動きにどのような背景があったのか、また彼らがどのよ
うな反応を想定しているのかが、よくわかると思います。今回の動きの全体像を、簡単に押さえ
ようと思ったら、まず、このFAQを読むのがいいと思います。

なぜ新しい仕様は、GGFのOGSIワーキンググループで検討されなかったか？

おそらく多くの人々が最初に疑問に感じたことは、OGSIの代替物である新しいWSRFの仕様が、GGF
のOGSI WGで議論するという形をとらなかったかということだだと思います。

最初に注意して欲しいことは、確かに今回の動きは、形の上では、GGF の OGS1-WG の「外部」で行われたのですが、その中心部分には、GGF の中心部分が含まれているということです。以下に、OGS1-WG のメンバーと OGS1 仕様書の編集者と、今回の WSRF のホワイトペーパーの著者のリストを示します。GGF を立ち上げた、I. Foster や GGF OGS1-WG の議長である S. Tuecke の動きを、GGF の「外部」の動きと呼んでいいのか、すこし釈然としないところもあります。筆者の尊敬するある政治学者は、こうした動きを「多数者分派」と呼ぶのだと教えてくれました。政治の世界では、珍しいことだそうです。

GGF OGS1-WG メンバー

Peter Vanderbilt
Lisa Childers I
Julie Wulf-Knoerzer (Administrator)
Savas Parastatidis
Tim Banks
Steve Tuecke (Chair) (Administrator)
Tom Maguire
Jem Treadwell
Ian Foster
Anbazhagan Mani
Charlie Catlett
David Snelling (Chair) (Administrator)
Guy Rixon
Frank Siebenlist
A Djaoui

OGS1.0 仕様書の Editor (*印は、今回の WSRF のホワイトペーパーの著者)

*S. Tuecke (ANL)
*K. Czajkowski (USC/ISI)
*I. Foster (ANL)
*J. Frey (IBM)
*S. Graham (IBM)
C. Kesselman (USC/ISI)
*T. Maquire (IBM)

T. Sandholm (ANL)
D. Snelling (Fujitsu Labs)
P. Vanderbilt (NASA)

今回のホワイトペーパー"Modeling Stateful Resources with Web Services"の著者
(*印は、OGSI1.0仕様書のEditor)

*Jeffrey Frey (IBM) (Editor)
*Steve Graham (IBM) (Editor)
*Karl Czajkowski (Globus / USC ISI)
Don Ferguson (IBM)
*Ian Foster (Globus / ANL) (Editor)
Frank Leymann (IBM)
Martin Nally (IBM)
Tony Storey (IBM)
*Steve Tuecke (Globus / ANL) (Editor)
William Vambenepe (Hewlett-Packard)
Sanjiva Weerawarana (IBM)

第一の理由：スピード

政治的な話は別にして、FAQは、この項目の冒頭でこう語ります。

「ここで議論しているWSRFの仕様は、ドラフトの提案であって、標準化に先立って、これから標準化団体の厳しいレビューと議論にさらされようとしている」ことを強調します。提案したのはドラフトであって、標準化の手続きはこれからきちんとするのだということですが、まあ、それはいわずもがなの当然のことです。FAQは、なぜ、ドラフトをGGFの外から出したのかについて、二つの理由をあげます。

「第一に、ひとたびWSRFの必要性が認識されるならば、進行中の全ての、OGSIにディPENDした実装や標準化の努力の失速を避けるためには、スピードが決定的な重要性を持っていることは明らかでした。」

OGSIからWSRFへの転換を、大急ぎでやる必要があったのだということです。

第一の理由は、まだ続きます。それは、一般的な議論のようにも思えるのですが、同じく一般的な

冒頭の釈明とは、多少論点がことなっています。Globus が、ある意味では無駄な議論を出来るだけ避けて、転換を急ぎたかったということを示しています。

「仕様のドラフトからはじめるという、仕様に到達するために早く動くこのやり方は、実際は、GGF で標準化された最初の OGSi ドラフトにアプローチした方法と同じものです。GGF や他の標準化の努力が成功してきたのは、一枚の白紙から始めるのではなく、ある程度合理的に肉付けされた最初のドラフトからスタートしているからです。実際、GGF を含めて多くの標準化のグループでは、広くデザインの議論を始めるのに先行して、ドラフトを作るというアプローチは、広く認められ推奨されてきた方法です。」

第二の理由：Web サービスの標準化との関係

FAQ は、GGF の外部で新仕様を作らなければいけなかった、二つ目の短い理由を次のように語ります。

「第二に、WSRF の仕様の開発は、専門的な知識、特に Web サービスの標準についての専門的な知識を要求します。OGSi の WG には、そうした代表がありませんでした。」

こうした議論は、OGSi のワーキンググループには、当然、Web サービスの専門家が含まれていると思っていた筆者には、少し意外なものでした。もっとも当事者がそういうのですから、それは本当のことなのでしょう。

OGSi が Web サービスの上に構築されているとは、誰もが知っていることです。OGSi の WG のメンバーが Web サービスを知らないわけはありません。この二番目の短い理由のポイントは、Web サービス一般についての知識ではなく、「Web サービスの標準についての専門的な知識」が Grid 標準化には必要なのだという主張にあります。二番目の理由は、短いものですが、今回の転換を特徴づけるのが、Grid の標準と Web サービスの標準という二つの標準のあいだの関係をどう捉えるのかという問題であることを端的に表現しています。

以下に、2003 年 3/13 に発表された "Web Services Addressing (WS-Addressing)" の著者のリストを掲げます。+印と*印は、先の WSRF のホワイトペーパーに名前が見える人をあらわしています。（*印の J. Frey は、OGSi と WSRF と WS-Addressing の三つに顔を出しています。彼は Web サービスの標準化の専門家ではないのでしょうか？）これを見ると、WSRF で Web サービスコミュニティを代表すべく「増強」されたと思われるメンバーの大部分が、WS-Addressing の著者で占められていることがわかります。このことは、今回の WSRF のデザインに WS-Addressing が大きな影響を与えていることを示しています。それ以外の新メンバーにも興味があって調べてみたの

ですが、Martin Nally は ServiceData 関連の論文を出しています。(SmallTalker のようでもあります) HP の William Vambenepe は、HP の Web サービスの顔のような人で、WSDL, WSDM の仕様策定にも顔を出し、最近では、WS-I の SoapBinding のドキュメントにも名を連ねていますので、「Web サービスの標準化の専門家」だと思います。

Adam Bosworth, BEA
Don Box, Microsoft (Editor)
Erik Christensen, Microsoft
Francisco Curbera, IBM (Editor)
+Donald Ferguson, IBM
*Jeffrey Frey, IBM
Chris Kaler, Microsoft
David Langworthy, Microsoft
+Frank Leymann, IBM
Steve Lucco, Microsoft
Steve Millet, Microsoft
Nirmal Mukhi, IBM
Mark Nottingham, BEA
David Orchard, BEA
John Shewchuk, Microsoft
+Tony Storey, IBM
+Sanjiva Weerawarana, IBM

OGSI に対する三つの批判

Grid の標準化には、「Web サービスの標準化についての専門的な知識」が必要とする FAQ は、続いて「WSRF は、Web サービスのコミュニティからの OGSI に対する批判に向けられているのか？」という問いを立ててそれに答えます。FAQ のこの部分は、今回の動きの背景を知る上では、重要な情報を含んでいると思います。

「WSRF の定義は、第一義的には、Web サービスのアーキテクチャのいくつかの発展、特に WS-Addressing、を統合したいという熱望によって動機付けられてきたのですが、そのデザインは、Web サービスのコミュニティからの OGSI の三つの批判に向けられています。」

ここでは、今回の動きが、Gridの標準化にとどまらず、Webサービスの新しい標準化を強く意識したものであることが語られています。そのためには、Webサービス陣営からの、Grid標準化に対する批判、特にOGSIに対する批判に答える必要があるということです。以下、その「批判」とそれに対する「回答」を見ていきましょう。

第一の批判：「仕様をがわかりにくい」

OGSIに対する第一の批判は次のようなものです。

「ひとつの仕様にあまりに沢山の材料を詰め込みすぎてる。多く人はOGSIのすべてではなく、その一部を使いたいと思っているだろう。一方、OGSI1.0の大部分は、オプション的なものだ。ある人は、一部の利用が全部の利用を義務付けているように感じている。」

こうした批判に、FAQは次のように答えます。

「回答：WSRFは、OGSI1.0の機能を、組み合わせ可能な一群の仕様に分割しました。」

確かに、OGSIの仕様書がわかりにくいというのは、批判としてはあたっているかも知れません。ただ、もともとソフトウェアをその仕様書から理解しようというのは、筆者はあまりいいアプローチとは考えておりません。実際に使ってみて理解が深まることもあれば、ソースを読んでなるほどと思うこともあります。とてもすばらしいソフトなのに、ドキュメントがお粗末というのは、よくある話です。もっとも、仕様がわかりやすいのにこしたことはありません。ここでは、WSRFが、OGSI1.0の機能を引き継いでいることが明言されていることに留意してください。WSRFとOGSIの関係については、また後でも触れますが、新しいWSRFの仕様書とOGSIの基本的な機能との関係を次に示します。

OGSIの基本機能

WSRFドキュメント

Grid Service Reference (GSR)	(WS-Addressing Endpoint Reference)
Grid Service Handle (GSH)	(WS-Addressing Endpoint Reference)
HandleResolver ポートタイプ	WS-RenewableReferences
ServiceData 定義とアクセス	WS-ResourceProperties
GridService 生存時間管理	WS-ResourceLifecycle
Notification ポートタイプ	WS-Notification
Factory ポートタイプ	(ホワイトペーパー)
ServiceGroup ポートタイプ	WS-ServiceGroup
基本 Fault 型	WS-BaseFaults

第二の批判：「Web サービスの一般的なツールが使えない」

Web サービス陣営からの、OGSI に対する第二の批判を見てみましょう。

「OGSI では、存在する Web サービスのツール類が、うまく動かない。OGSI1.0 は、かなり積極的に XML Schema を使っている。例えば、xsd:any、アトリビュート、Document 志向の WSDL を本質的に多用している。こうした特徴が、例えば、JAX - RPC との間で問題を引き起こしている。」

前段のツールがうまく動かないという「批判」は、OGSI に対してでなく、現在市場に存在する開発ツールの「批判」に向けられてもいいような気がします。後段の Schema の多用、他の Web サービスプログラムとの互換性の問題は、確かに、いくつか微妙な問題を含んでいます。ツールが使えるかどうかで、アーキテクチャを議論するのは、すこし筋違いですが、Grid が Web サービスとの互換性を失っていくのを懸念するのは正しい判断だと思います。これに対する FAQ の回答は、次のようなものです。

「回答： WSRF では、XML Schema の利用は、幾分、トーンダウンしています。」

残念ながら、仕様（の一部）が公開されただけで、実装も、また開発手順の概要も筆者には見えませんので、どのようになるのかは、このトーンダウン発言ではわかりません。ツールとの相性は、XML Schema の利用の程度の問題だけではないのです。例えば、GWSDL を WSRF では、使うのか使わないのか（多分、使わないと思います）Document-Literal だけでなく、RPC-Literal も使うのか（多分、それはあるような気がします）、もっと進んで（後退して）、SOAP-Encoding も認めるのかも気になります。それこそ、「Web サービスの標準化」の中核である WS-I の動きを見れば、それはないと思うのですが、現実には WS-I 互換のコードを吐くツールは、まだ少数です。WS-I に参加しているベンダの多数派が、WS-I の外部から、ツールが使えないことを理由に、別仕様を提案することがないことを祈ります。

JAX-RPC との関係

JAX-RPC の名前が出てきていましたが、OGSI が JAX-RPC との関係でトラブルを引き起こしているとは筆者は考えていません。OGSI のコードを見て感心したことのひとつは、OGSI の実装者たちが、すでに設計時から組み込まれていた、WSDL や JAX-RPC の拡張機能を実によく知っていて、それを見事に使いこなしているということでした。JAX-RPC は、もともとは、Sun の J2EE に Web サービスを取り込むために構想されたものです。そうした起源にもかかわらず、JAX-RPC は、Java と XML の型の対応付けや、Service インターフェースの「発見」といった興味深い成果を挙げながら、今では、Web サービスと Grid プログラミングのもっとも有力な開発手段に成長しています。JAX-RPC と Web サービスとの間で問題があるとすれば、筆者が思いつくのは、例えば、JAX-

RPCの古巣ともいふべき J2EE での Web サービスの実装が、あらかじめコンポーネントを作ってデプロイしておくという J2EE の運用上のスタティックな性格から、static Stub の利用が主流になっていて、Dynamic Proxy や、ServiceLocator の利用といった、多様な JAX-RPC のプログラミング・モデルが利用されていないことです。J2EE1.4 のブループリント Adventure Builder では、ServiceLocator のデザインパターンを、EJB や JMS では積極的に使っているのに、Web サービスでは、ServiceLocator を使おうとしていません。こうしたことは、OGSI が進みすぎていてという問題としてではなく、J2EE の JAX-RPC 利用の遅れとして理解されるべきだと筆者は考えています。

Schema ANY の多用

OGSI での XML Scheme の多用については、筆者も感じるどころがいくつかありました。OGSI では XML の ANY 要素は、Java の ExtensibilityType にマッピングされます。ただこれを、Java と XML の型変換の一般的なメカニズムに取り込むのか、例外的な処理と見なすかは、考えなければならぬいろいろな面倒な問題があるように感じています。個人的には、ANY 要素で ANY 属性であるという XML の Schema 定義が意味しているのは、「XML なら何でもいい」ということであって、特定の Java の型に対応するものではないと思っています。あえて言えば、Generic な Object に対応させるのがよさそうですが、ANY 要素 ANY 属性の Schema は、そのような含意を持たないのかもしれませんが。畢竟、Java と XML とは、二つの異なる型システムです。相互の型変換の妥当性は、両者の交わる中心部ではトリビアルなほど明らかですが、それぞれの周縁部に行くにしたがって明証性は失われます。原理的に言えば、トリビアルと思えるものでも、例えば、Java の String と XML の String の型の同一性を語るのは、私達の「文字列」に対する直感に訴えることなしには難しいことがわかります。ANY 要素が、型変換の中で大きな役割を演ずるようなシステムは、すこし疑問です。

ANY 要素の Java でのパートナー ExtensibilityType も、Grid プログラミングの世界、特に ServiceData の周辺では、大きな役割を果たしています。しかし、そのコードは非常にわかりにくいものです。次の例は、"CounterState" という名前を持つサーバ上の ServiceData の値をクライアントから読むプログラムの一部です。findServiceData というのが、ServiceData の値にアクセスする基本的なメソッドなのですが、最後のキャスト以外はほとんど意味不明のコードに思えます。ただ、こうした異様なプログラミングが、もっとも基本的な OGSI の idiom の一部になっているのは不思議なことです。

```
-----  
ExtensibilityType extensibility =  
    gridService.findServiceData( QueryHelper.getNamesQuery("CounterState") );
```

```
ServiceDataValuesType serviceData =
    AnyHelper.getAsServiceDataValues(extensibility);
CounterStateType counterState =
    (CounterStateType)AnyHelper.getAsSingleObject(serviceData,
        CounterStateType.class );
```

わかりやすいのは、多分、次のようなコードだと思います。(あるいは、RMI/IIOP 風に narrow を使ってもいいのですが。型変換の実装は、確かに面倒です。)

```
CounterStateType counterState =
    (CounterStateType) gridService.findServiceData("CounterState" );
```

findServiceData のプロトタイプ

ついでに不満を述べれば、OGSI で ServiceData にアクセスする findServiceData は、次のようなプロトタイプを持っています。ある Grid サービスの ServiceData の値にアクセスするのが、このメソッドの仕事ですので、戻り値が「どのようなタイプかは、呼んでみないとわからない」を意味する ExtensibilityType であるのは納得できます。ただ引数も ExtensibilityType が必要とされるのはなぜでしょう? 先のサンプルを見るとわかりますが、ServiceData の名前が、いったん QueryHelper クラス内のメソッド getNamesQuery に渡されていることがわかります。この getNamesQuery が ExtensibilityType を返すのですが、やりたいことは ServiceName の名前(せいぜい String 型か QName 型だと思います)から、その値を知りたいということです。引数の ExtensibilityType は、あまり意味が無いように思えます。このあたりも、Schema の ANY 型の使いすぎのように筆者には感じられました。

```
public ExtensibilityType findServiceData( ExtensibilityType queryExpression)
    throws RemoteException,
        FaultType,
        TargetInvalidFaultType,
        ExtensibilityNotSupportedFaultType,
        ExtensibilityTypeFaultType;
```

WSRFでの「状態」リソースプロパティの扱い

ちなみに、WSRFでServiceDataに対応するWS-ResourcePropertiesの仕様を見ると、このあたりはすこし改善されるようです。まず、WSDLでのWS-ResourcePropertiesの定義ですが、OGSIではServiceDataがportType要素のサブ要素としてServiceData要素を定義していましたが、次のようにportTypeの、ResourceProperties属性で、リソースプロパティを束ねたリソースプロパティドキュメントを関連付けるという形になっています。

```
-----  
<wsdl:definitions ... xmlns:tns="http://example.com/diskDrive" ...>  
  ...  
  <wsdl:types>  
    <xsd:schema targetNamespace="http://example.com/diskDrive" ... >  
  
      <!-- リソースプロパティ要素の宣言 -->  
      <xsd:element name="NumberOfBlocks" type="xsd:integer"/>  
      <xsd:element name="BlockSize" type="xsd:integer" />  
      <xsd:element name="Manufacturer" type="xsd:string" />  
  
      <!-- リソースプロパティドキュメントの宣言 -->  
      <xsd:element name="GenericDiskDriveProperties">  
        <xsd:complexType>  
          <xsd:sequence>  
            <xsd:element ref="tns:NumberOfBlocks"/>  
            <xsd:element ref="tns:BlockSize" />  
            <xsd:element ref="tns:Manufacturer" />  
            <xsd:any minOccurs="0" maxOccurs="unbounded" />  
          </xsd:sequence>  
        </xsd:complexType>  
      </xsd:element>  
  
      ...  
    </xsd:schema>  
  </wsdl:types>  
  
  ...  
  <!-- リソースプロパティドキュメントをポートタイプと関連つける -->  
  <wsdl:portType name="GenericDiskDrive"
```

```
    wsrp:ResourceProperties="tns:GenericDiskDriveProperties" >
      <operation name="start" .../>
      <operation name="stop" .../>
      ...
    </wsdl:portType>
    ...
  </wsdl:definitions>
```

このサンプルのリソースプロパティドキュメント `GenericDiskDriveProperties` が次のような値をもっていたとしましょう。

```
<GenericDiskDriveProperties xmlns:tns="http://example.com/diskDrive" >
  <tns:NumberOfBlocks>22</tns:NumberOfBlocks>
  <tns:BlockSize>1024</tns:BlockSize>
  <tns:Manufacturer>DrivesRUs</tns:Manufacturer>
</GenericDiskDriveProperties>
```

このとき、WSRF では、次のようなリクエストをポートに送ると、次のようなレスポンスが返って、ドキュメント内のプロパティの値を取り出すことが出来ます。OGSI の `findServiceData` では、`QueryHelper` で作られるリクエストメッセージの姿をすぐには想像できませんでした。WSRF の `GetResourceProperty` のメッセージは、すっきりしたものです。わかりにくいものよりは、それで用事がすむのなら、わかりやすいもののほうがずっといいのです。

GetResourceProperty のリクエスト

```
<wsrp:GetResourcePropertyRequest xmlns:tns="http://example.com/diskdrive">
  tns:NumberOfBlocks
</wsrp:GetResourcePropertyRequest>
```

GetResourceProperty のレスポンス

```
<wsrp:GetResourcePropertyResponse xmlns:ns1="http://example.com/diskdrive" ...>
  <ns1:NumberOfBlocks>22</ns1:NumberOfBlocks>
```

```
</wsrp:GetResourcePropertyResponse>
```

GetMultipleResourceProperties を使えば、複数のプロパティにアクセスすることも出来ます。

```
<wsrp:GetMultipleResourcePropertiesRequest xmlns:tns="http://example.com/diskdrive"
...>
  tns:NumberOfBlocks
  tns:BlockSize
</wsrp:GetMultipleResourcePropertiesRequest>
```

```
<wsrp:GetMultipleResourcePropertiesResponse xmlns:ns1="http://example.com/diskdrive"
...>
  <ns1:NumberOfBlocks>22</ns1:NumberOfBlocks>
  <ns1:BlockSize>1024</ns1:BlockSize>
</wsrp:GetMultipleResourcePropertiesResponse>
```

プロパティへの書き込みも、非常にわかりやすくなりました。次のリクエストは、NumberOfBlock を 143 に値を設定し、Manufacturer を削除し、新たに someElement を挿入しています。

```
<wsrp:SetResourcePropertiesRequest xmlns:tns="http://example.com/diskdrive">
  <wsrp:Update resourceProperty="tns:NumberOfBlocks">
    <tns:NumberOfBlocks>143</tns:NumberOfBlocks>
  </wsrp:Update>
  <wsrp>Delete resourceProperty="tns:Manufacturer" />
  <wsrp:Insert>
    <tns:someElement>42</tns:someElement>
  </wsrp:Insert>
</wsrp:SetResourcePropertiesRequest>
```

このとき、リソースプロパティドキュメントは、次のような値を持つことになります。

```
-----  
<GenericDiskDriveProperties xmlns:tns="http://example.com/diskDrive" >  
  <tns:NumberOfBlocks>143</tns:NumberOfBlocks>  
  <tns:BlockSize>1024</tns:BlockSize>  
  <tns:someElement>42</tns:someElement>  
</GenericDiskDriveProperties>  
-----
```

本当は、この例の場合にはレスポンスの内容が、Header 部分を含めて重要なのですが、それは WS-Addressing を紹介してから説明したいと思います。

第三の批判：「OGSI は、あまりにオブジェクト志向が強すぎた」

もとの議論に戻りましょう。OGSI で、Schema の ANY の多用に少し辟易としたのは事実ですが、仕様がわかりにくいツールが使えないのというのは、筆者には、OGSI を捨てる、大きな理由になるとは思えませんでした。ただ、FAQ が語る次の第三番目の「批判」は、今回の転換の本質を突いているように思われました。（この FAQ の作者は、大事なことを最後に言う癖があるようです。）

「OGSI は、あまりにオブジェクト志向が強すぎる。OGSI 1.0 は、状態を持つリソースを Web サービスとしてモデル化している。そこでは、リソースの状態はカプセルに入れられ、サービスのアイデンティティとライフサイクルは、リソースの状態と結び付けられている。」

確かに、Grid で GridCounter サービスを作って、Counter の値を ServiceData に入れて、それを読み書きしている限りでは、Counter の状態とは GridCounter サービスの状態に他なりません。「状態を持つサービス」として、OGSI の ServiceData を使った Grid サービスをイメージした時、筆者が意識していたのは、おそらく、こうした Counter モデルだったような気がします。ところが、「状態」としてモデルされるものが、Grid カウンターの値ではなく、例えば、丸山の銀行口座の残高だとすれば、その「状態」は、直ちに丸山口座 Grid サービスの残高 ServiceData の値の「状態」と、同一視することは出来なくなります。両者は、ある条件の下では、同じ型の同じ値を持ってきちんと対応しているのですが、実体的には別のものです。両者を、インスタンスと強弁することは出来るかもしれませんが、それは、異なるインスタンスで、別の名前 (Identifier)、別の生存時間をもつでしょう。

stateless なサーバーと状態を持つサービス

第三の、ServiceData 批判は、まだ続きます。

「このアプローチは、Web サービスの純粹主義者の間に心配の種を植え付けた。彼らは、Web サービスは、状態を持たないし、インスタンスももっていないと議論している。それに加えて、Web サービスの実装の中には、ダイナミックなサービスの生成や破壊に、うまく適応できないものも存在する。」

Web サービスに純粹主義者がいるのは知りませんでしたが、オブジェクト志向派にも原理主義者は存在します。純粹主義も原理主義も、ほどほどにしないとかえって世の中が悪くなります。ここでの論点は、むしろ、「ステートレスなサーバは頑丈である」という、経験的で現実的な観点からの批判と思うと納得できることもあります。筆者も、OGSI の Grid サーバでは、どのように ServiceData の persistency が保障されているのか知りたいと思って、コードの中を捜し歩いたことがあります。最初に、PersistentService という範疇を見つけたのですが、それは OGSI 的には、永続性を持つ Service というよりは、Factory を経由しないで生成されるサービスという意味を持たされていました。また、それらのサービスの状態保存は、persistent とされるサービスインスタンスのプロパティが、次のようなメソッドの呼び出しで、デプロイ・ディスクリプタに、セーブされる形で行われていました。それは、最低限のチェックポイント機能は備えているとあっていいものの、「状態を持つサービス」を扱うと言うには貧弱なものだと思います。

```
org.globus.ogsa.impl.core.service.ServicePropertiesImpl#flush()
```

```
-----  
public synchronized void flush() throws ServicePropertiesException {  
    if (!ServicePropertiesHelper.isPersistentLifecycle(this)) {  
        return;  
    }  
    ServiceDeployment deployment =  
        (ServiceDeployment) getProperty(  
            ServiceProperties.SERVICE_DEPLOYMENT  
        );  
    if (deployment != null) {  
        try {  
            String serviceName =  
                (String) getProperty(ServiceProperties.SERVICE_PATH);  
            deployment.saveServiceAsync(serviceName);  
        } catch (Exception e) {  
            throw new ServicePropertiesException(e);  
        }  
    }  
}
```

```
}  
}
```

WS-Addressing と implied resource pattern

こうした批判に対する、FAQの答えを見てみましょう。

「回答: WSRFは、基礎にあるOGSIのアーキテクチャを再分節化して、サービスとそのサービスによって作用を受ける状態をもつ実体との間に、明確な区別を置きます。WSRFは、こうした実体をリソースと呼びます。また、WS-Addressingの約束に基づいてリソースに作用するサービスは、「含意されたリソースパターン(implied resource pattern)」を示すと言います。」

ここが、このFAQ全体の、あるいはWSRFの理論的な核心とっていい部分です。前段のサービスと状態を持つ実体の区別は、重要ですが、わかりやすい話だと思います。問題は、後段の、リソースパターン云々の話です。これを読んだだけでは、何が何なのかわかりません。実は、今回発表されたドキュメントの中に、仕様ではないホワイトペーパーがひとつ含まれているのですが、このドキュメント"Modeling Stateful Resources with Web Services"が、この問題を丁寧に扱っています。このドキュメントはWSRFの基礎にある考え方を示すものとして、仕様以上に重要です。(筆者は、馬鹿正直に仕様から読み始めて、わけがわからず、よけいな回り道をしてしまいました。) リソースパターンとは、サービスと状態をもつ実体(リソース)とを関係付けるために、WebサービスのWS-Addressing技術を利用する上での、いくつかの約束事の集まりだと思ってください。

残念ながら、状態をどのようにモデル化するかといった基本的な問題も、WSRFの中核技術であるWS-Addressingやその上でのImplied リソース・パターンの説明をしている余裕が無くなってしまいました。

Web サービス、Grid、J2EE

これらの大きなかたまりは次回にまわすとして、最後に、すこし、Web サービス、Grid、J2EEの関係について話をしたいと思います。今回、Web サービスとGridは、めでたく合体を宣言して、OGSI的なアプローチを捨てることで、状態を持つサービスをどのように実装するのかといった問題に、それなりの答えを出したかに見えます。ただ、まったく異なる道をとって、おそらくは経験的で現実的な選択の積み重ねで、J2EEがだしていた、stateless、statefull、entityという三つのEJBを使い分けるというアイデアは、状態を持つサービスを現実的に考える上では、かえって

大きな説得力を持っているようにも思えます。(JD0 や SD0 の話は、その意味では、面白いのかもしれない。) 単純な話、OGSI のサンプルがほとんど Counter 一色だったのを、そのときは不思議と思わなかったのですが、OGSI で、ショッピングカートのサンプルを作ろうと思えば、これはうまくいかないだろうと皆が思ったような気がします。ショッピングカートなら、J2EE の statefull ビーンでの実装がスマートです。(サーブレットでもクッキーでもできますが) GlobusWorld 以降、現実的な目で周りを見直し始めているのですが、Web サービス、Grid、J2EE の周辺には、まだまだ面白い話題がありそうです。