SOAの中核技術としてのBPEL入門（1）

BPELはどのようにサービスを結合するか？

丸山不二夫 （稚内北星学園大学）

BPELとは何か？

WS-BPEL（Web Services Business Process Execution Language）[1] は，ビジネスプロセスの統合を目指して，複数の Web サービスを結合するための Web サービス標準です．**図 -1** に位置付けを示します．当初は，BPEL4WS（Business Process Execution Language for Web Services）として IBM, Microsoft, BEA が仕様を策定していたのですが，その後，OASIS で，WS-BPEL（以下 BPEL と略）として標準化にかけられています．ここには，IBM, Microsoft, BEA だけでなく Oracle, Sun, SAP といった主要ベンダがすべて参加しています．Web サービスの統合に関しては，BPEL が中心的な技術であるという大きな合意が産業界に生まれていると考えてよいと思います．その意味で，BPEL は，ネットワーク上のサービスの結合を目的とする SOA の中核をなす標準技術といえると筆者は考えています．現在，OASIS では，WS-BPEL の Version2.0 の Public Review Draft が提案されています．

http://docs.oasis-open.org/wsbpel/2.0/wsbpel-specification-draft.pdf

小論の構成

この連載では，2 回に分けて BPEL の紹介をしていきたいと思います．

第 1 回目の今回は，BPEL はどのようにサービスを結合するのかを，BPEL の PartnerLink という概念を中心に解説したいと思います．BPEL では，同期型・非同期型という 2 つのスタイルのサービスの呼び出しが利用されることが，理解の大事なポイントになります．

第 2 回目の次回は，BPEL での変数の宣言とそのアサイン，BPEL の基本的な制御構造など，XML を用いたプログラム言語としての BPEL の特徴を解説します．

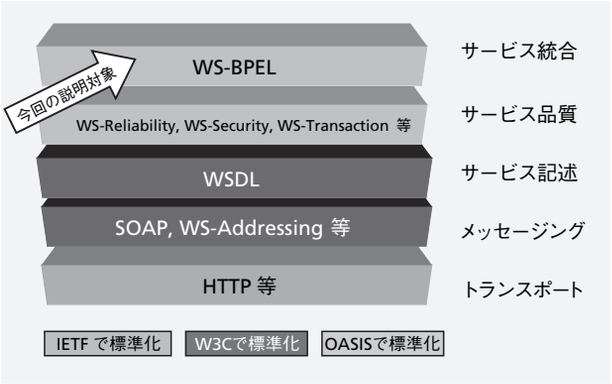

図 -1　WS-BPEL の位置付け

BPELはどのようにサービスを結合するか

BPEL の利用者には，BPEL は，1 つの Web サービスの提供者のように見えるだけです．**図 -2** を見てください．ここでは，クライアント・サービスが BPEL の提供するサービスを呼び出しているのですが，BPEL は，あたかも 1 つの WSDL で定義された，1 つの Web サービスであるかのように，クライアントに対しては振る舞います．

ところが，BPEL は，その 1 つのサービスをクライアント・サービスに提供するために，背後で，複数のサービスを参照しそれを結合しています．**図 -3** を見てください．この図は，BPEL がクライアントに提供しているサービスは，サービス A とサービス B とサービス C の 3 つのサービスの結合されたものだという基本的な関係を表現しています．

ただ，それで終わりではありません．BPEL から呼び出されているサービス A，サービス B，サービス C は，一見すると 1 つのサービスのように見えるのですが，それ自身，BPEL で結合された複数のサービスから構成されたものかもしれません．**図 -4** を見てください．この例の場合には，サービス B は単純なサービスでしたが，サービス A とサービス C は，BPEL で結合されたサービスです．サービス A は，BPEL がサービス A1，サービ

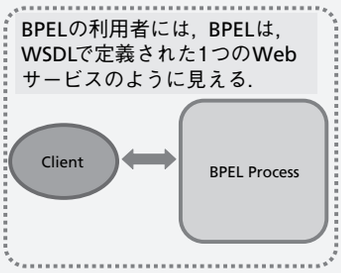

図 -2　利用者から見た BPEL サービス

ス A2，サービス A3 を参照して構成されたサービスですし，サービス C は，サービス C1 とサービス C2 を参照して構成されたものであることが分かります．

サービスの結合は再帰的な性質を持っています．すなわち，複数のサービスが参照されて構成されたサービス（図 -3）も，1 つのサービスにほかなりません（図 -2）．こうした再帰的な性質を持つサービスの連鎖がずっと続くことができます（図 -4）．

サービスの結合関係の基本的な表現

BPEL がどのようにサービスを結合しているかを見るには，図 -2 は単純すぎます．また，図 -4 は，少し複雑です．ただ，よく見れば，図 -4 の中にも，図 -3 のタイプの表現が，繰り返し現れていることが分かります．実は，サービスの結合にとっては，図 -3 が表現している関係が基本的なものなのです．

あらためて，図 -3 を見てください．この図の左側は，BPEL がクライアント・サービスに提供するサービスを表現しています．この図の右側は，BPEL がクライアント・サービスに提供するサービスを構成するために参照している，複数のサービスを表現しています．

注意してほしいのは，この図は，BPEL が，3 つのサービス A, B, C から 1 つのサービスを構成しているという基本的な関係は表現していますが，それぞれのサービスがいつ参照され，どのように呼び出されるのかについての具体的な情報は持っていないということです．今回は，図 -3 で表現されるような，サービス間の結合の抽象的な関係を対象にします．サービスが具体的にどのように結合されるかについては，次回に説明することにします．

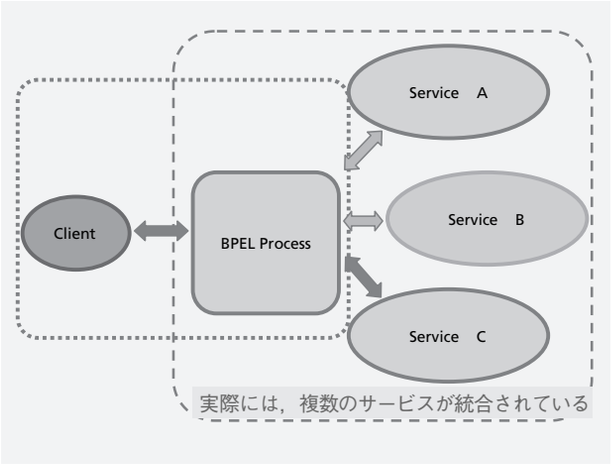

図 -3　BPEL によるサービスの統合

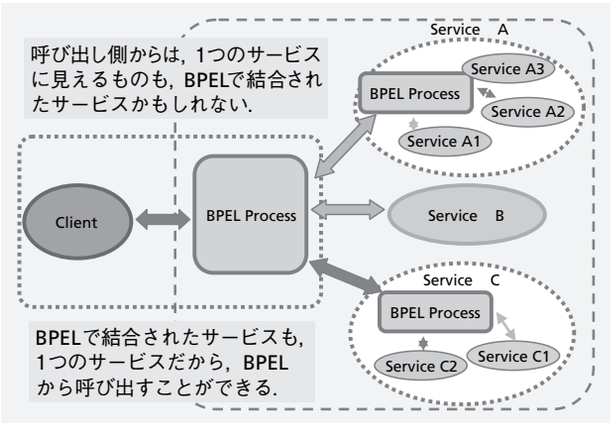

図 -4　再帰的なサービスの統合

PartnerLink とは何か？

図 -4 では，クライアント・サービスから BPEL の呼び出しと，BPEL から参照されるサービスの呼び出しに対応して，双方向の矢印が，BPEL を中心として描かれています．BPEL と関係を持つサービスを，BPEL の Partner といいます．もちろん，Partner であるという関係は相互的なものなので，サービスのほうから見れば，BPEL はそのサービスの Partner ということになります．

図 -3 の双方向の矢印で示されるような，BPEL と Partner の組を，PartnerLink と呼びます．一般的には，PartnerLink は Partner 同士の組なのですが，BPEL で登場する PartnerLink はどちらかの Partner が BPEL を含んでいるということです．PartnerLink には，2 人の Partner が登場するのですが，その 2 人を区別したい時には，Role（役割）で区別します．ParnerLink では，BPEL 中心に考えて，BPEL 側の Role を「myrole」，BPEL

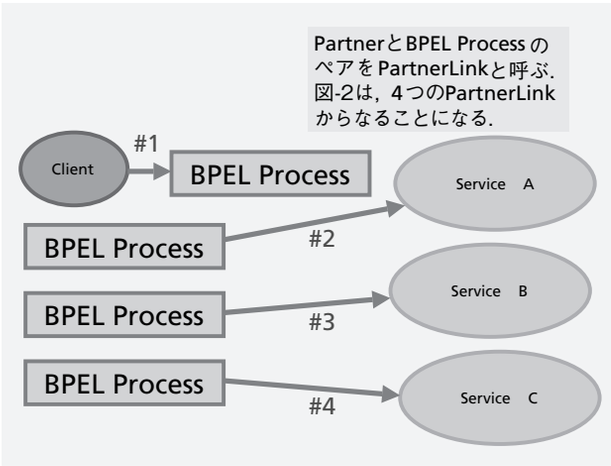

図 -5　PartnerLink

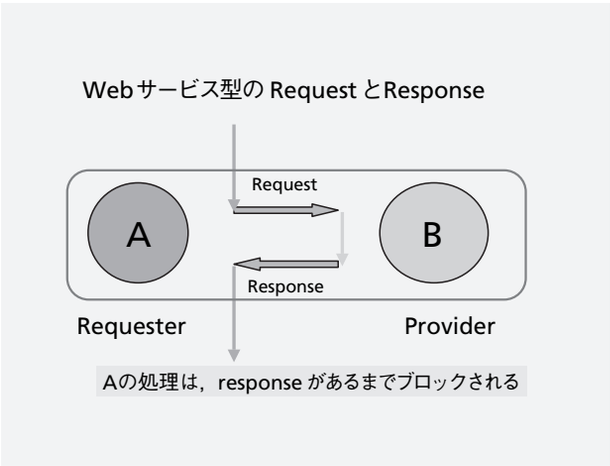

図 -6　同期型のメッセージ交換

に相対するサービス側の Role を「partnerrole」と呼んでいます．この PartnerLink での Role という考え方は，本稿の後半で PartnerLink を形式的に記述する際に，大事な役割を果たします．

図 -5 は，図 -3 を構成する 4 つの PartnerLink を取り出したものです．図 -3 では，メッセージの行きと帰りのやりとりを意識して，PartnerLink は双方向の矢印で表現されていましたが，図 -5 では，サービスの呼び出しの方向を表す単方向の矢印で表現されています．どちらでも，PartnerLink が，BPEL と Partner の組であることには変わりはありません．BPEL の Partner となるサービスには，クライアント・サービスのように BPEL を呼び出すサービスと，BPEL によって参照され呼び出されるサービスの 2 つのタイプがあることが分かります．

先に，図 -3 は，BPEL が結合するサービスの間の基本的な関係を表すと言いました．図 -3 から BPEL とサービスの間の関係を，PartnerLink の形で取り出したものが図 -5 です．ですので，図 -3 と図 -5 は，サービスの結合に関しては同じ情報を持っています．見かけは少し違うように見えますが，図 -3 と図 -5 が同じ関係を表現しているのを見るのは，やさしいと思います．たとえば，先とは逆に，図 -5 のような PartnerLink の組が与えられたとき，それに対応する図 -3 のような図式を構成するのは，難しいことではありません．

こうして，次のことが分かると思います．

「BPEL が結合するサービスの間の基本的な関係は，PartnerLink の組で定義される」

BPEL プログラミングの目的は，さまざまな具体的なサービスを結合することです．BPEL プログラミングの詳細については次回に説明しますが，BPEL プログラムには，それが結合する具体的なサービスの関係についての定義が，すなわち PartnerLink の定義が必ず含まれます．PartnerLink の定義は，BPEL プログラムの最も重要な構成要素の 1 つなのです．

同期型の呼び出しと，非同期の呼び出し

先に，BPEL を呼び出すサービスと BPEL から呼び出されるサービスの 2 つがあるという話をしましたが，サービスの呼び出し方に注目して，BPEL でのサービス間の関係を，もう少し見ていきたいと思います．ここでは，「メッセージ交換パターン」や同期型／非同期型という概念が重要です．

BPEL に限らず SOA では，一般的にネットワーク上でのサービスの結合は，ネットワーク上でのメッセージの交換によって行われます．メッセージの交換には，いくつかの基本的なパターンがあって，それは，メッセージ交換パターン（Message Exchange Pattern），頭文字をとって，MEP と呼ばれることがあります．MEP については W3C が Draft を公開しています [2)]．基本的な MEP の 1 つは，Request-Response MEP と呼ばれるものです．**図 -6** を見てください．Request-Response MEP は，これまでの普通の Web サービスに見られるように，クライアントからのリクエスト・メッセージに対してサーバがレスポンス・メッセージを返すというパターンです．このとき，クライアント側のプロセスは，サーバからの応答を待ちブロックします．クライアントが動作を再開するのはサーバからの応答が返ってからです．リクエストとレスポンスは，同期をとって一対のものになっていて，切り離すことはできません．こうしたメッセージパ

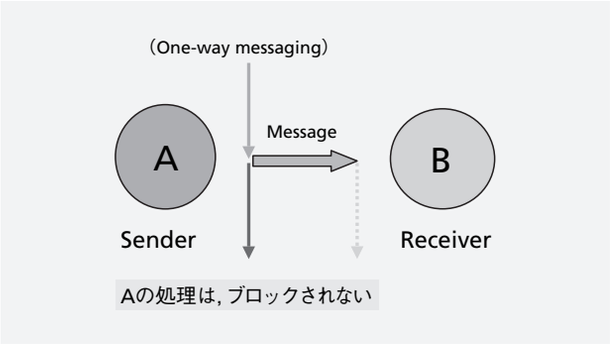

図 -7　非同期型のメッセージ交換

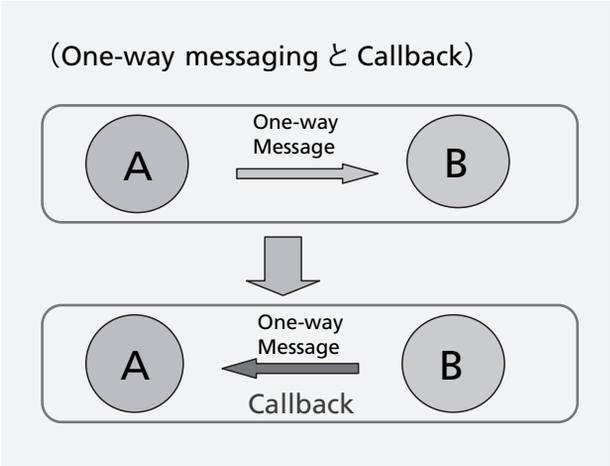

図 -8　非同期型のメッセージ交換

ターンを,「同期型」と呼ぶこともあります.

もう 1 つ基本的な MEP があります.**図 -7** を見てください. One-way MEP と呼ばれるものです. その名前のように, 一方向にメッセージが送られるだけで, レスポンスが返ることはありません. ですから, 送り手がブロックして同期をとることもありません. 連載 4 回目（2006 年 12 月号）で紹介した WS-Notification や, JMS や MQ といったメッセージング系の技術は, このタイプの MEP に従っています. 注目すべきことは, WS-Addressing 等の新しい技術を背景に, Web サービスでもこうした One-way MEP を取り扱おうという流れが生まれていることだと思います.

BPEL で重要な MEP は, 2 種類あります. 1 つは, 先に見た同期型の Request-Response MEP で, もう 1 つは, 非同期型の Callback MEP と呼ばれるものです. Callback MEP は, 2 つの One-way MEP を, 非同期に組み合わせたものです（**図 -8**）.「非同期」にというのは, 最初にメッセージを送り出した側が, Callback メッセージが返るまで同期をとってブロックして待つことをしないという意味です.

非同期呼び出しと WS-Addressing

概念的には, Request-Response MEP も, 2 つの One-way MEP の組合せと考えることができます. Request-Response MEP と Callback MEP との違いは, 同期をとってブロックするのか, 非同期に仕事を続けるのかの違いのように考えることもできます. しかし, 両者の間には, 現実の実装上は大きな違いがあります. 何よりも, ほとんどの Web サービスでの Request-Response MEP は, メッセージの行きと帰りとで, 同一のコネクションを使います. Callback MEP では, そうしたことはありません.

BPEL での非同期の Callback MEP の導入は, 次のような実践的な背景を持っていました. 現実のビジネスプロセスには, 処理に非常に時間がかかる処理があり得ます. たとえば, 社長の決裁を得るには長く待たされることがあるかもしれません. 何日もの間, 同期型でプロセスをブロックさせてコネクションを維持するのは, リソースの無駄使いになります. こうした時, リソースを解放して, 非同期でコールバックを待つというアプローチは有効です. ただし, Web サービスのサーバは状態を持たず, かつ, これまでの SOAP メッセージ・ヘッダには, 返り先の情報は含まれておりません. いったん同期型のコネクションを切ってしまえば, One-way メッセージを受け取った側が, コールバックを返すことはできませんでした.

こうした問題を解決したのが, 連載 2 回目（2006 年 10 月号）で紹介した Web サービスに対する WS-Addressing の導入です. BPEL は, 非同期 Callback MEP の実現のために, WS-Addressing を利用しています. Grid 技術の基礎としての WS-RF が最初の応用例だった WS-Addressing が, ビジネスプロセスの統合を目的とした BPEL の中で重要な応用を見つけ出しているのは, 興味深いことです.

これまでのまとめ — PartnerLink の分類

これまでの説明をベースに, BPEL と BPEL が結合するサービスの間の関係である PartnerLink を, あらためて分類・整理してみましょう.

ParnerLink を分類する, 2 つの軸があります. 1 つには, 今見てきたように, サービスの呼び出しが同期型か非同期型かによって, PartnerLink を分類することができま

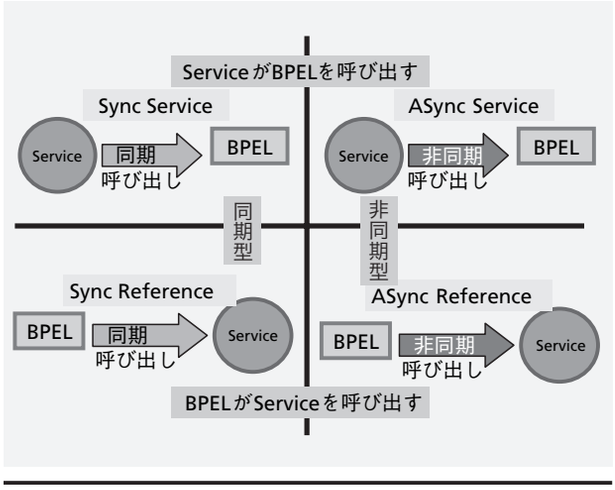

図 -9　PartnerLink の 4 つの種類

す．もう 1 つは，サービスが BPEL を呼ぶのか，あるいは，BPEL がサービスを呼ぶのかによって，PartnerLink を分類することが可能です．

前者のサービスが BPEL を呼ぶという関係は，典型的には，BPEL を利用しようとするクライアントの BPEL に対する関係です．図 -3 でいえば，BPEL の左側に位置していたサービスと BPEL の関係です．この PartnerLink は，クライアントにとっては，BPEL が提供するサービスを表しています．

後者の BPEL がサービスを呼ぶという関係は，BPEL と BPEL が参照するサービスとの関係です．図 -3 で言えば，BPEL の右側に位置していたサービス群と BPEL との関係です．この PartnerLink たちは，BPEL が参照・結合するサービスを表しています．

図 -9 は，同期／非同期，サービス／参照という 2 つの軸で分類された，都合 4 つの，PartnerLink の種類を表しています．ここでは，4 つの種類の PartnerLink に，仮に，Sync Service, ASync Service, Sync Reference, ASync Reference という名前を付けています．

PartnerLink と PartnerLinkType

先に，すべての BPEL プログラムには，必ず PartnerLink が含まれるという話をしましたが，ここでは，PartnerLink の型にあたる PartnerLinkType という概念について説明したいと思います．

Amazon や Google のような，よく知られたネット上のサービスが BPEL の Partner となる場合を考えてみましょう．たとえば，BPEL が Google の検索サービスを同期的なスタイルで呼び出すという関係を表す PartnerLink を考えてみましょう．Google の検索サービスにアクセスする複数の BPEL プログラムが存在し得るのですが，複数の BPEL プログラムごとに，基本的には同じタイプの PartnerLink が，別々の PartnerLink として含まれているのが分かると思います．こうした複数の PartnerLink に共通する同一の型を PartnerLinkType と呼びます．

この例では，PartnerLink の定義は，複数の具体的な BPEL プログラムに含まれているのですが，それでは，それらの共通性を表す PartnerLinkType は，どこで定義されるのでしょう？　PartnerLinkType は，複数の BPEL プログラムに共通のサービスを提供している，サービスの側で定義されるのが自然です．実は，BPEL では，そのために，BPEL と関係を持つサービスの WSDL の定義を拡張しています．こうして，PartnerLinkType は，サービス側の拡張された WSDL で定義されることになります．

WSDLでの同期型・非同期型のサービスの定義

PartnerLinkType の WSDL の定義を見る前に，同期型・非同期型のサービスの WSDL での定義の特徴を振り返っておきましょう．WSDL の portType の定義のスタイルを見ると，そのサービスが同期型であるか非同期型であるかを判断することができます．

◉ 同期型のサービスの定義

同期型，すなわち，request-response MEP の portType でのサービス記述を見ておきましょう．次のリストを見てください．Request 部分が operation 要素中の input 要素に，Response 部分が output 要素に対応しています．このように同期型のサービスの portType の特徴は，operation 要素に，input 要素と output 要素の 2 つの要素が存在することです．

```
<portType name="SyncHello">
   <operation name="process">
       <input  message=
            "tns:SyncHelloRequestMessage"/>
       <output message=
           "tns:SyncHelloResponseMessage"/>
   </operation>
</portType>
```

同じ WSDL 内の，この portType に対応した PartnerLinkType の定義は次のようになります．

```
<plnk:partnerLinkType name="SyncHello">
   <plnk:role name="SyncHelloProvider">
       <plnk:portType name="tns:SyncHello"/>
   </plnk:role>
</plnk:partnerLinkType>
```

この PartnerLinkType 定義は，PartnerLinkType に名前を与えるとともに，partnerLink の Role に名前を与え，同時にこの Role にサービスを提供する portType をその名前で対応付けていることを確認してください．partnerLinkType 内の portType 名である "tns:SyncHello" は，この WSDL の portType 名 "SyncHello" にほかなりません．

1 つの疑問は，PartnerLink は 2 人の Partner ＝ Role の関係を表現するはずなのに，この例では 1 つの Role しか定義されていないのはどうしてかということです．これは，2 つの Role のうちの 1 つが省略されたものです．この 1 つの Role が表現する 1 つのサービスが，任意の Role，任意のサービスに対して，特に条件なしに開かれていると解釈することができます．同期型でサービスを提供する場合には，レスポンスはリクエストがあったところに必ず返りますので，サービスの提供側の Role さえ明確に指定できれば，もう一方の Role の指定の省略が可能だということです．

こうして，同期型のサービスの PartnerLinkType の定義の特徴は，1 つの Role しか含まれていないということになります．

非同期型のサービスの定義

今度は，非同期型のサービスの portType でのサービス記述の特徴を見てみましょう．次のリストを見てください．まず，いずれの portType も，operation 要素内に，input 要素しかないことに注目してください．これは，One-way MEP の特徴です．また，portType が 2 つあることにも注意が必要です．これは，2 つの One-way メッセージのペアから構成される BPEL の非同期型のメッセージングの特徴を表現しています．メッセージ属性の名前が ASyncHelloWorldRequestMessage と ASyncHelloWorldResponseMessage であることは，この 2 つの portType が一対のものであることを反映しています．

こうしたメッセージの名前の対応は，構造的な関連を少しも含意するものではありませんが，構造を反映した適切な名前を付けることは，プログラムを読みやすくする上では重要です．ここの例では，2 つの portType に対する ASyncHelloWorld と ASyncHelloWorldCallback という portType の名付けの意味にも注意が必要です．一方の ASyncHelloWorld は，クライアントが HelloWorld サービスを呼び出す BPEL 側の portType ですが，もう一方の ASyncHelloWorldCallback は，BPEL 側からの Callback を受け取るクライアント側の portType です．

ここでは，物理的には異なるマシン上のサービスが，論理的には同一の WSDL 上で記述されていることに注意が必要です．第 4 回のコラムにも書きましたが，One-way メッセージの向きは，サービスを提供する側から見て決まります．ASyncHelloWorld の initiate オペレーションの input は，このサービスを提供する BPEL 側から見ての input です．一方，ASyncHelloWorldCallback の onResult オペレーションは，BPEL から呼び出されますので，サービスを提供するのはクライアント側ということになります．ですので，ここの input は，クライアント側から見ての input ということになります．

このように，1 つの input 要素しか持たない operation 要素からなる 2 つの portType のペアが存在することが，非同期型の BPEL サービスの portType の特徴になります．

```
<portType name="ASyncHelloWorld">
    <operation name="initiate">
        <input message=
         "tns:ASyncHelloWorldRequestMessage"/>
    </operation>
</portType>

<portType name="ASyncHelloWorldCallback">
    <operation name="onResult">
       <input message=
        "tns:ASyncHelloWorldResponseMessage"/>
    </operation>
</portType>
```

同じ WSDL 内の，この portType のペアに対応した partnerLinkType の定義は，次のようになります．先に見たように，この定義が，PartnerLinkType と partnerLink の Role に名前を与え，Role と portType の対応付けを与えるものだということは分かると思います．

```
<plnk:partnerLinkType name="ASyncHelloWorld">
    <plnk:role name="ASyncHelloWorldProvider">
        <plnk:portType name="tns:ASyncHelloWorld"/>
    </plnk:role>
    <plnk:role name="ASyncHelloWorldRequester">
        <plnk:portType name=
         "tns:ASyncHelloWorldCallback"/>
    </plnk:role>
</plnk:partnerLinkType>
```

こうして，非同期型のサービスの PartnerLinkType の定義の特徴は，2 つの One-way メッセージングのペアを反映して，2 つの Role が含まれているということになります．

BPEL での PartnerLink の定義

先に WSDL での，同期型と非同期型のサービスの PartnerLinkType の定義を見てきましたので，ここでは，先の WSDL でのサービス定義それぞれに対応する BPEL での PartnerLink の定義について見てみようと思います．

BPEL の定義内では，partnerLink は，次の例のように，partnerLinks 要素の内部にまとめられています．次の 2 つの例を見てください．最初の例が同期型の，次の例が非同期型の PartnerLink の定義の対応例になります．PartnerLink の名前がどちらも client であることからも分かると思いますが，両方の例ともに，BPEL がサービスを提供する例です．

Sync Service 型

```
  <partnerLinks>
     <partnerLink name="client"
        partnerLinkType="tns:SyncHelloWorld"
        myRole="SyncHelloWorldProvider"/>
  </partnerLinks>
```

ASync Service 型

```
  <partnerLinks>
     <partnerLink name="client"
        partnerLinkType="tns:ASyncHelloWorld"
        myRole="ASyncHelloWorldProvider"
        partnerRole="ASyncHelloWorldRequester" />
  </partnerLinks>
```

partnerLink は，name 属性で指定される名前を持ちます．また，この partnerLink の属する型は，partnerLinkType 属性で示されます．

PartnerLink で重要なことは，PartnerLinkType が 1 つ（同期型）あるいは 2 つ（非同期型）の Role を持つことに対応して，myRole あるいは，partnerRole という，1 つあるいは 2 つの属性を持つことです．先に見たように，これらの partnerLink はいずれも client という名前を持っていますが，前者の client は，1 つの role，myRole を持ち，後者の client は，myRole と partnerRole という 2 つの role を持っています．

PartnerLinkType のところで説明したように，role は，それぞれが提供するサービス，すなわち，呼び出す portType に結び付けられていることに注意してください．もっとも，partnerLink では，partnerLinkType と Role の名前だけが残っていて，portType の情報は，直接は，消えています．もちろん，partnerLinkType の名前とその Role の名前を，WSDL で調べれば，必要な portType の情報はすぐに得ることができます．

今見てきた例は，2 つとも BPEL がサービスを提供する例でしたので，今度は，BPEL から参照されて BPEL にサービスを提供するタイプの PartnerLink も見ておきましょう．紙幅の都合で，partnerLinks 要素を省略し，対応する PartnerLinkType，portType の定義も省いてあります．

Sync Reference 型

```
<partnerLink name="creditRatingService"
       partnerLinkType="services:CreditRatingService"
       partnerRole="CreditRatingServiceProvider"/>
```

ASync Reference 型

```
<partnerLink name="UnitedLoanService"
          partnerLinkType="services:LoanService"
          myRole="LoanServiceRequester"
          partnerRole="LoanServiceProvider"/>
```

myRole と partnerRole の区別

それでは，partnerLink の定義の中に現れる，2 つの role, myRole と partnerRole には，どのような違いがあるのでしょうか？　これも，大事なポイントになります．

Role が 2 つあるケースは，非同期型になるのですが，BPEL で記述される側のサービスが myRole，BPEL

の外側のサービスが partnerRole だと思ってもらえばいいと思います．ASync Reference 型の例でしたら，myRole が LoanServiceRequester で，partnerRole が LoanServiceProvider になっているのは，BPEL がリクエスタになって，外部のサービス LoanServiceProvider を参照し呼び出していることを表しています．先の ASync Service の例でも，やはり，myRole が BPEL 側であることを確認してください．

Role が 1 つの場合は，同期型のサービスです．Role が myRole だったら，そのサービスは，BPEL が提供する同期型のサービスで，Role が partnerRole だったら，そのサービスは，BPEL の外側の同期型のサービスだということになります．

BPEL 自身をサービスとして呼び出す

BPEL は，XML を利用したプログラム言語です．BPEL では，プログラムの命令を Action と呼びます．その詳しい説明は次回に行いますが，BPEL のサービスの呼び出しには，receive，invoke，reply といった複数の Action がかかわります．ここでは，BPEL 自身が，外部のサービスを参照することなしに，クライアントにサービスを提供する単純な場合を例に，PartnerLink の定義によって，サービスの呼び出しのスタイルが変わることを見ておきたいと思います．

非同期型のサービス呼び出しのシーケンス

最初の例は，単純な非同期型の HelloWorld です．ASync Service 型の PartnerLink のサンプルで示した例と，それに対応する，WSDL での PartnerLinkType，portType の定義を，そのまま借用して，BPEL での非同期型のサービス呼び出しの手法を説明していきたいと思います．

典型的な非同期型のシーケンスは次のような形をとります．まず，receive で client PartnerLink からメッセージを受け取ります．この時，HelloWorld portType の initiate オペレーションが利用されています．

Receive には，もう 1 つの重要な意味があります．Receive だけが，BPEL の中でビジネスプロセスのインスタンスを生成させることができます．この例での createInstance="yes" というのがその働きをしています．

HelloWorld が結果を出したら，今度は client の callback が呼ばれることになります．そのためには invoke を使って HelloWorldCallback portType の onResult オペレーションを呼び出します．

```
------------------------------------------------------------------------
        ......
        <receive name=
         "receiveInput" partnerLink="client"
                 portType="tns:HelloWorld"
                 operation="initiate"
                 variable="input"
                 createInstance="yes"/>
        ......
        <invoke name="replyOutput"
                partnerLink="client"
                portType="tns:HelloWorldCallback"
                operation="onResult"
                inputVariable="output"/>
        ......
------------------------------------------------------------------------
```

同期型のサービス呼び出しシーケンス

今度は，先の Sync Service 型の PartnerLink の定義を使って，単純な同期型のサービス呼び出しの例を見ておきましょう．

client PartnerLink からのリクエストを receive が受け，インスタンスを生成するのは，非同期型と一緒です．ただし，それからが少し違います．invoke ではなく，reply が呼ばれています．ここでは，receive も reply も，SyncHelloWorld という portType の process という同一の operation が呼ばれていることに注意してください．このように同期型の場合には，portType の同一の operation に対する Request/Response に対応して，receive/reply が呼ばれることになります．

```
------------------------------------------------------------------------
        ......
       <receive name="receiveInput"
                partnerLink="client"
                portType="tns:SyncHelloWorld"
                operation="process"
                variable="input"
                createInstance="yes"/>
        ......
        <reply  name="replyOutput"
                partnerLink="client"
                portType="tns:SyncHelloWorld"
                operation="process"
                variable="output"/>
        ......
------------------------------------------------------------------------
```

非同期型，同期型のそれぞれについて，BPEL 自身がサービスを提供する単純な場合での使い方を見てきまし

た．非同期型が，<receive…./> …..<invoke …. />，同期型が，<receive…./> …..<reply …. /> を基本形にすることは分かったと思います．

BPEL が外部の非同期サービスを参照する場合

今度は，BPEL が外部のサービスを参照する場合の呼び出し方を見てみましょう．もしも，外部参照する 1 つのサービスの呼び出しが非同期型の場合，関係するアクションを取り出すと次のようなシーケンスになります．

```
1. <receive name="receiveInput" partnerLink="client" ../>
2. <invoke name="invokeAsyncService" partnerLink=
   "AsyncBPELService" ../>
3. <receive name="receive_invokeAsyncService" partnerLink=
   "AsyncBPELService" ../>
4. <invoke name="replyOutput" partnerLink="client" ../>
```

partnerLink を見ると，1 と 4，2 と 3 がペアになっていることが分かります．1 と 4 のペアは，先に見た非同期の単純な例と同じ使い方です．2 と 3 のペアは，BPEL 外の非同期サービスを invoke で呼び出したあと，その結果を受け取るのに receive が使われているわけです．非同期といいますが，2 で呼び出されたサービスは，このプロセスとは非同期に結果を返しますので，3 の receive は，ブロックすることに注意してください．

myRole, partnerRole に注意してメッセージの流れの向きを見ると分かりますが，いずれの場合でも，receive が外部からフローへのメッセージを受け取るのに対して，invoke はフロー側から外部にメッセージを送ることになります．

BPEL が外部の同期サービスを参照する場合

今度は，BPEL が外部参照するサービスが同期型の場合のシーケンスです．

```
1. <receive name="receiveInput" partnerLink="client" ../>
2. <invoke name="invoke" partnerLink=
   "IncrementService" ../>
3. <invoke name="replyOutput" partnerLink="client" ../>
```

こうすると，1 と 3 が，非同期の client という partnerLink でペアを構成していることが分かります．2 は，partnerLink の定義を見れば，同期型で外部（partnerRole）に位置するサービス Increment ServiceProvider との同期型でのメッセージのやりとりであることが分かります．

次回は，BPEL での変数の宣言，代入，制御構造など，BPEL の基本的な言語仕様を紹介します．

参考文献

1）OASIS WSBPEL TC : http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel

2）W3C “Web Services Message Exchange Patterns” : http://www.w3.org/2002/ws/cg/2/07/meps.html

（平成 18 年 12 月 26 日受付）

丸山不二夫（正会員）
maruyama@wakhok.ac.jp

東大教育学部卒業．一橋大学大学院社会学研究科博士課程修了．「最北端・最先端」をモットーに，稚内で新しいスタイルとコンテンツの情報教育を展開．「新しい時代の新しい大学」を目指して，社会人 IT 技術者をターゲットとしたサテライト校を秋葉原に設置．アジアでの IT 教育も熱心に展開している．現在，稚内北星学園大学学長．