8.　ビジネスプロセスソリューションへの適用による評価

8.1.　はじめに

今や、インターネットが安価で、大容量・高速なデータ通信インフラとして利用されることが当たり前となり、個人向けの情報提供の場としてのみならず、企業同士の情報の交換や、B2B と呼ばれる企業間取引も盛んに行われるようになって来ている。これは、従来の EDI(Electronic Data Interchange：電子データ交換)をより動的に、よりオープンな環境で実現可能とするものであり、e マーケットプレイスと呼ばれるインターネットを介して公開された市場を形成している。新日本製鉄を代表とする鉄鋼業界においても、資材調達や製品注文のプロセスにおいて、特に海外マーケットを対象として、こうした B2B のアプローチが取り入れられ、ここにおいて、より効率的な取り組みを行うことが同業他社との競争力強化につながると考えられる。一方、Web サービスと呼ばれる枠組みが提唱されるようになり、SOAP(Simple Object Access Protocol)という XML(eXtensible Markup Language)をベースとしたプロトコルにより、Web 上でコンポーネント化された機能を提供し、コンピュータ同士が連携できるフレームワークが提供されつつある。これにより、システムソリューションを提供するアプローチにおいても、固有プラットフォーム上に機能を密に作り込み提供する従来の形態から、必要とされる個別機能をネットワーク上から動的な疎結合により組み合わせ、提供するという形態に転換することにより、より低コストで、高機能・高信頼性を持ったシステムを提案・提供できる可能性が高まった。

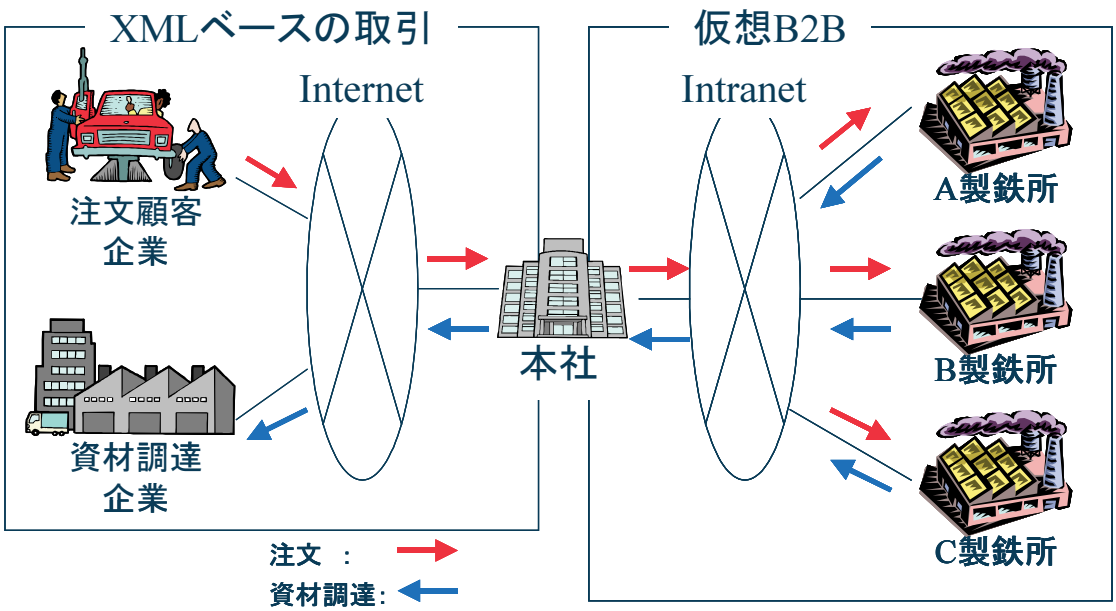

図 8.1.1 B2B 社内統合 Web サービスモデル

こうした背景を踏まえて、本テーマにおいては、本プロジェクトにおいて開発された、Web サービスを実現するためのミドルウェアである OpenSOAP を、実際に B2B 向け Web サービスの枠組みに合わせてアプリケーションレベルで適用し、その有用性および問題点を評価した。

8.2. OpenSOAP 適用と評価

評価のためのモデルとして、インターネット上および、イントラネット上に分散する鉄鋼ＥＣサービスを、ビジネス競争力強化を目的とした、システム統合利用を想定する(図 8.1.1)。

しかしながら、本テーマ内では、その基礎的機能の実現性、実用性の検証および、統合利用の可能性を検証することに重きを置く為、出来る限り簡易なモデルを構築し、これらをプロトタイプとして作成、評価することにより、ビジネスレベルでの実用に向けての各種の検討項目を評価した。

以下に、評価のために実際に作成したプロトタイプに関して述べる。

8.2.1. Calc サービスプロトタイプ

基本 SOAP メッセージ交換の実現を検証する目的で作成した、Web サービスプロトタイプである。このプロトタイプでは、四則演算の機能を提供する Web サービスプログラムと、それを利用するクライアントプログラムを OpenSOAP ミドルウェアを使用して開発した。

以下に、このサービスにおける OpenSOAP サービス設定ファイル(SSML)を示す。

なお、SSML に関しては本研究のサブテーマである「SOAP メッセージ管理技術の実装」の章を参照されたし。

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE SSML SYSTEM "ssml.dtd">
<SSML name='Calc' xmlns="x-schema:ssmlSchema.xml">
        <service name='Calc'
         nsuri='http://services.opensoap.jp/samples/Calc/'>
            <connection name='CalcSocket'>
                <Socket hostname='localhost' port='8766'/>
                <asynchronizedTTL >8000</asynchronizedTTL>
                <synchronizedTTL count="second" >20</synchronizedTTL>
                <MaxProccessNumber>5</MaxProccessNumber>
```

```
            </connection>
            <operation type ='CalcSocket'>Add</operation>
            <operation type ='CalcSocket'>Subtract</operation>
            <operation type ='CalcSocket'>Multiply</operation>
            <operation type ='CalcSocket'>Divide</operation>
            <fault signature='1' />
            <MaxProccessNumber>15</MaxProccessNumber>
        </service>
</SSML>
```

8.2.2.　CalcAsync サービスプロトタイプ

非同期メッセージ処理機能の実現を検証する目的で作成した、Web サービスプロトタイプである。このプロトタイプでは、Calc サービスプログラムと同様の四則演算の機能を提供する Web サービスプログラムと、その Web サービスに対して、機能の要求・結果の取得を非同期の手順で行うクライアントプログラムを OpenSOAP ミドルウェアを使用して開発した。

このサービスにおける SSML は、Calc サービスプロトタイプと同じものとなる。つまり、Web サービスを提供する側は、この非同期メッセージ処理を意識した作りにする必要は無く、OpenSOAP サーバの内部機能として実現、提供されるものである。

8.2.3.　Shopping サービスプロトタイプ

Web サービスにおける一連のビジネスフローとしての動作を検証する目的で作成した、Web サービスプロトタイプである。商品一覧取得、在庫確認、発注処理およびその結果確認といったショッピングをモデルとした Web サービスプログラムと、そのクライアントプログラムを OpenSOAP ミドルウェアを使用して開発した。

以下に、このサービスにおける SSML を示す。

```
<?xml version='1.0' encoding='UTF-8' ?>
<SSML xmlns="x-schema:ssmlSchema.xml">
        <service name='Shopping'
          nsuri='http://services.opensoap.jp/samples/Shopping/'>
            <connection name='ShoppingSocket'>
                <Socket hostname='localhost' port='9877'/>
                <asynchronizedTTL >8000</asynchronizedTTL>
```

```
            <synchronizedTTL count="second" >20</synchronizedTTL>
            <MaxProccessNumber>5</MaxProccessNumber>
        </connection>
        <operation type ='ShoppingSocket'>GetProductCount</operation>
        <operation type ='ShoppingSocket'>GetProductSpec</operation>
        <operation type ='ShoppingSocket'>GetStockQty</operation>
        <operation type ='ShoppingSocket'>PlaceOrder</operation>
        <fault signature='1' />
        <MaxProccessNumber>15</MaxProccessNumber>
    </service>
</SSML>
```

8.2.4. ShoppingSec サービスプロトタイプ

認証や暗号化といった、ビジネスプロセスでは必須となるセキュリティ機能の実現を検証する目的で作成した Web サービスプロトタイプである。Shopping サービスと同様のモデルであるが、発注処理およびその結果確認において、セキュリティ機能を利用するものである。

以下に、このサービスにおける SSML を示す。

```
<?xml version='1.0' encoding='UTF-8' ?>
<SSML xmlns="x-schema:ssmlSchema.xml">
    <service name='ShoppingSec'
      nsuri='http://services.opensoap.jp/samples/ShoppingSec/'>
        <connection name='ShoppingSecSocket'>
            <Socket hostname='localhost' port='9878'/>
            <asynchronizedTTL >8000</asynchronizedTTL>
            <synchronizedTTL count="second" >20</synchronizedTTL>
            <MaxProccessNumber>5</MaxProccessNumber>
        </connection>
        <operation type ='ShoppingSecSocket'>GetProductCount</operation>
        <operation type ='ShoppingSecSocket'>GetProductSpec</operation>
        <operation type ='ShoppingSecSocket'>GetStockQty</operation>
        <operation type ='ShoppingSecSocket'>PlaceOrder</operation>
        <fault signature='1' />
```

```
            <MaxProccessNumber>15</MaxProccessNumber>
        </service>
    </SSML>
```

8.2.5.　Transaction サービスプロトタイプ

トランザクション機能の実現を検証する目的で作成した、Web サービスプロトタイプである。このプロトタイプでは、クライアントが、複数のリクエストメッセージをまとめてサービスに対して送信し、それらを個別に対応するサービスに対して処理をさせ、それぞれのサービスの処理結果に応じてコミットもしくは、ロールバック命令を再度それぞれのサービスに送ることによって、トランザクション機能を実現可能としている。

以下に、このサービスにおける SSML を示す。

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE SSML SYSTEM "ssml.dtd">
<SSML xmlns="x-schema:ssmlSchema.xml">
        <service name='Transaction' nsurl='http://services.opensoap.jp/Transaction/'>
        <connection name='TransactionSocket'>
                <Socket hostname='localhost' port='9879'/>
                <asynchronizedTTL >8000</asynchronizedTTL>
                <synchronizedTTL count="second" >20</synchronizedTTL>
                <MaxProccessNumber>5</MaxProccessNumber>
            </connection>
            <operation type ='TransactionSocket'>TransactionControl</operation>
            <fault signature='1' />
            <MaxProccessNumber>15</MaxProccessNumber>
        </service>
</SSML>
```

8.2.6.　検証・評価

先に挙げた検証用プロトタイプにより、以下に述べる項目に関して検証・評価を行った。

① 基本ＳＯＡＰメッセージ交換の実現

すべてのサービスプロトタイプにおいて、目的とする機能を享受できること

を確認した。これにより、OpenSOAP ミドルウェアの適用で、Web サービスとしての基本的な機能が実現されることが検証された。

② ＳＯＡＰメッセージ転送機能の実現

OpenSOAP サーバのサーバ間転送機能により、複数のサーバを経由しての Web サービスの提供・利用が可能であることを確認した。これにより、サービスを提供する側は、そのサービスの実態が存在するサーバをインターネット上に公開することなくユーザにサービスを提供できることになる。このことは、ビジネスプロセスにおける重要な内部データの保護に関して、サービス提供者が取らなくてはならない対策コストを大きく削減することが可能になることを意味している。これにより、Web サービスシステムの構築に際して、OpenSOAP ミドルウェアを適用することによって、多大なメリットが得られることを検証した。

③ 非同期ＳＯＡＰメッセージ交換機能の実現

Web サービスを利用するクライアントが、リクエストを送信した後、そのままセッションを保持し、最終的なレスポンスを得るまで処理を待機するという同期処理によるサービス呼び出しではなく、必要なリクエストを送信した結果としてのメッセージ ID を一旦レスポンスとして受け取り、必要に応じた別処理の後、保持してあるメッセージ ID を持って最終的なレスポンスを任意のタイミングで取得できる非同期処理としての動作を確認した。

この機能により、提供されるサービスの処理結果が得られるまでに多大な時間がかかるような、非常に複雑なビジネスサイドの機能の利用を可能とし、また、複数の手続きからなる一連のビジネスフローの実現に対しても、柔軟な設計を可能とする。このことから、単純なデモ的 Web サービスの実現ではなく、実際の複雑なビジネスプロセスに対応した Web サービスの構築が、OpenSOAP ミドルウェアの適用により実現できることを検証した。

④ トランザクション機能の実現

Transaction サービスプロトタイプにおいて、クライアントが、複数のメッセージを一括してサービスに対して送信し、それら各々のメッセージが正しく目的のサービスによってそれぞれ処理され、また、その結果により複数サービスの処理結果の反映・差し戻し(コミット・ロールバック)が正しく行われることを確認した。

Web サービスの目指す形のひとつとして、ひとつのサービスを入り口として、そこから関連する複数の Web サービスが動的に連携し、それぞれ担当の機能を処理し、全体として大きなサービスを提供するというワンストップサービスがあるが、これを実現しようとする場合には、このトランザクション機能

は必須の機能であるといえる。その意味で、OpenSOAP ミドルウェアでその枠組みを提供していることは、大きく評価できる。しかしながら、サービス提供側が、この枠組みに従い高い信頼性を保証しながらトランザクション機能に対応するためにはかなりの開発コストが上乗せされることも事実であり、かつサービスを提供する複数のベンダがすべて本枠組みに従うことが必要であるため、現状のまますぐにビジネスレベルに適用できるとは言い難い。
ただ、先にも述べた通り、トランザクション機能実現の枠組みが提供されたことは検証され、評価できたため、今後の展開に期待できる。

⑤ C 言語によるサービス開発効率
本テーマにおいては、各サービスプロトタイプをすべて C 言語により開発することが可能であった。これは、OpenSOAP ミドルウェアが C 言語による API を提供しているためであり、従来からの開発経験やノウハウが豊富な C 言語による開発は非常に効率的であったと評価できる。特に、サービス提供部分を C 言語で実装できることは、その裏に存在する既存のレガシシステムとの連携を容易にし、低コストで有用なサービスを構築できることを示した。

⑥ C 言語によるサービス利用クライアント開発効率
サービスの開発と同様、経験のある C 言語によるクライアント開発はやはり非常に効率的であったと評価できる。ただし、クライアントに関してはレガシシステムとの連携も無視は出来ないが、他の Web アプリケーションなど、ネットワークリソースの効率的利用も考慮に入れた場合、Java による開発要求も大きいと思われる。このことから、OpenSOAP ミドルウェアの Java API の充実を期待したい。

⑦ データセキュリティの確保
OpenSOAP ミドルウェアの認証・暗号化機能を利用し、容易に認証情報の追加および、必要となる情報のみの部分的暗号化の実現を確認した。これにより特別なツール等を事前に用意しなくても、OpenSOAP ミドルウェアが提供する API の利用だけで、ビジネスプロセスにおいて必須であるセキュリティの確保が検証できた。ただし、こうしたセキュリティに関するアプローチは、明らかにすべての Web サービス実現の手段において必要であることから、何らかの標準化の動きが予想されるため、これらとのインターオペラビリティの確保に注意する必要があると思われる。

⑧ パフォーマンス
本テーマにおいては、あくまで、基礎的機能の実現および現実的なビジネスプロセスへの適用の可能性を検証することを目的としているため、厳密なパフォーマンスのチェックは行っていない。しかし、SOAP という XML ベース

のプロトコルを扱う上で、多大な処理コストを占有するであろう XML パースの機能に libxml2 というツールを採用している点や、内部機能を C あるいは C++言語により実装していることから、現状十分なパフォーマンスを得ていると評価している。ただし、実際のビジネスプロセスにおいては、より複雑な処理が行われるであろうことや、非常に多数のサービスへのアクセスが予想されることから、実際のビジネス適用事例を通して、パフォーマンスの問題点は洗い出して行きたい。

⑨ スケーラビリティ

以下の２点の要因から、OpenSOAP ミドルウェアにおけるスケーラビリティに関しては問題無いと評価できる。

- Transaction サービスに代表されるように、OpenSOAP サーバのアーキテクチャは柔軟な拡張性を持っているため、負荷分散機能などの取り込みが可能。
- OpenSOAP サーバのメッセージ転送機能により、サービスを提供するサーバのリソースを分散することが可能。

なお、評価に使用したこれらサービスプロトタイプは、OpenSOAP ミドルウェアのサンプルプログラムとして、配布パッケージ中に同梱してある。

8.3. Web サービス運用面からの評価

本研究の成果物である OpenSOAP ミドルウェアにより、ビジネスプロセスソリューションとしての Web サービスの構築は十分可能であり、その有用性は検証できたが、そうして構築された Web サービスを、実際のビジネスプロセスの中で運用していく場合には多くの課題が残されていることが、本テーマを通して認識された。それらの問題点に関して言及する。

8.3.1. Web サービスの利用上の問題点

Web サービスのメリットとして、ネットワーク上に分散された形で、ユーザのプラットフォームに依存せずに、求める機能が動的に利用できることが挙げられるが、ここにおいて以下のような問題点が挙げられる。

I. Web サービスがどこに存在しているか事前に知っていなければ利用できない。

II. Web サービスを利用する上で、どのような情報を提供する必要があるか、つまりサービスのインターフェース、仕様が事前に知られている必要がある。

III. 提供される Web サービスが、信頼できるサービスであるかどうか。

Web サービスが構築されて、提供されたとしても、ユーザがそのサービスの存在を知り、提供場所を認識できなければ、実際にはサービスを利用することは出来ない。これは、従来のシステム開発における、必要機能をシステム構築時に静的にかつ密な結合を持って組み合わせ実現するというアプローチに対して、Web サービスの優位性である、必要な機能を実際に利用するタイミングで動的にかつ疎な結合を持って組み合わせて利用するというメリットを享受する上において、重要な課題となる。

つまり、事前に Web サービスを提供する側と、それを利用する側がお互いに情報を共有し、その情報を元に Web サービスを利用するというソリューション形態においては、問題にならないのであるが、不特定多数の利用ユーザに対して Web サービスを提供しようとした場合、ユーザが動的に Web サービスの存在場所やその仕様を把握できる仕組みがなければならない。

そこで、その仕組みを提供するものとして、UDDI(Universal Description、Discovery and Integration)が挙げられる。

ここで、簡単にだけ UDDI に関して述べるが、これは Yahoo!や Google のような Web ページを検索するためのディレクトリサービスに対して、Web サービスを検索するためのディレクトリサービスと理解して良い。

Web サービスを提供するものは、UDDI レジストリに対して情報を登録・公開

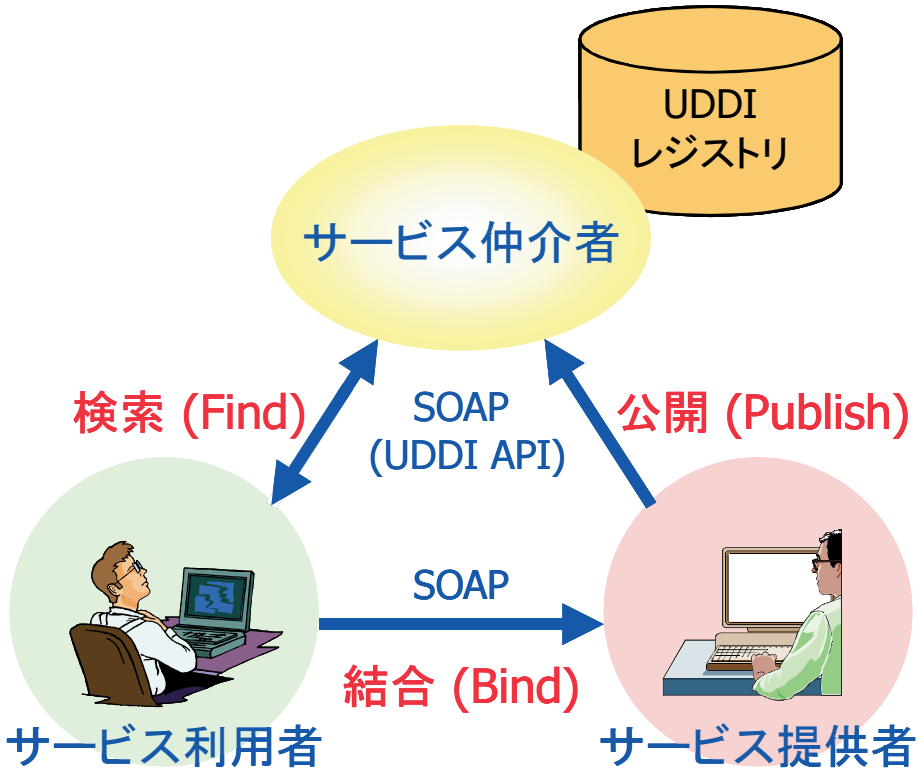

図 8.3.1 UDDI

し、Web サービスを利用したいものは、UDDI レジストリから期待する Web サービスの情報を検索して、その情報を元に Web サービスと結合し機能を利用する(図 8.3.1)。ここで得られる情報とは、大まかに言えば Web サービスが存在する場所と、Web サービスの仕様書に当たる WSDL（Web Services Description Language）に関する情報である。この仕組みを利用することにより、利用者側はまさしく機能を利用するそのタイミングで、希望する最新の機能を提供する Web サービスを選択し利用することが可能となる。

しかし、UDDI によって、Web サービスの存在場所や、その仕様に関する情報が得られたとしても、依然としてその提供されるサービスが信頼できるものかどうかという判断が出来ないという問題が残る。これは非常に重要な問題ではあるが、その解法は明確になっていない。企業や国などの信頼性に対して、格付けを行う機関があるが、今後は Web サービスに関しても、同様な格付けを行い、信頼性の指針となる情報を管理するアプローチが必要となるであろう。

8.3.2. Web サービス開発プロセス

本項では、現状での一般的な Web サービスの開発プロセスに関して言及する。

前項で、UDDI により動的に Web サービスの仕様に関する情報を得ることが可能であることを述べたが、ユーザサイドから見れば、そこで得た情報を元に、そのサービスを利用するためのクライアントプログラムを設計・実装するため(図 8.3.2)、結局静的な情報と同じ扱いとなってしまう。

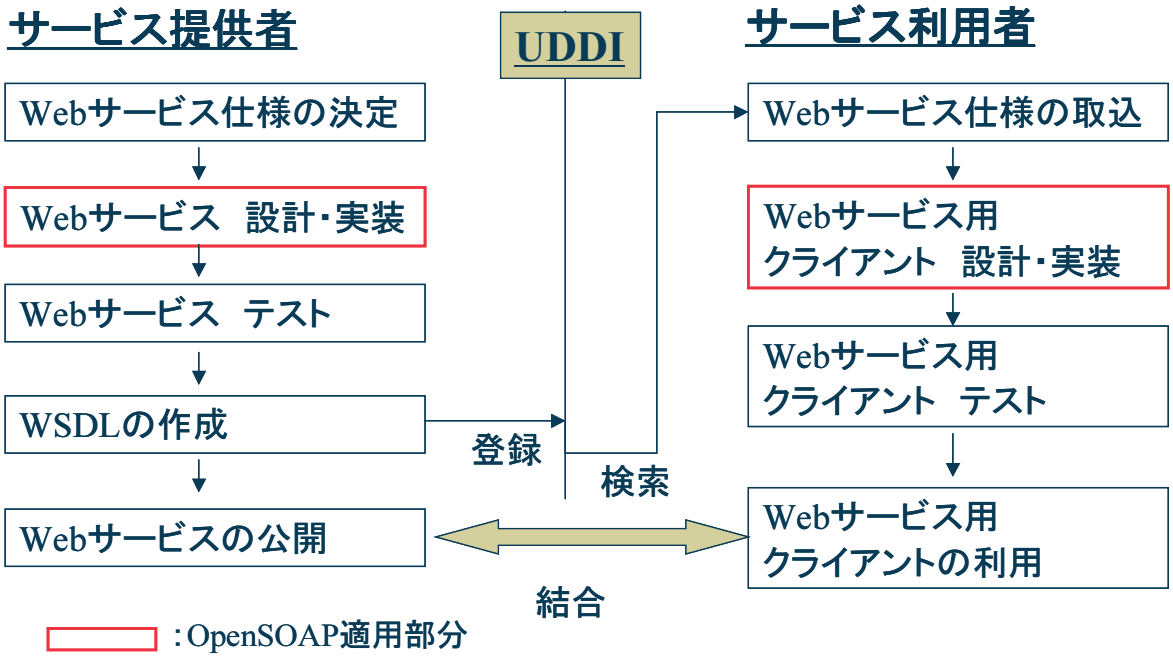

図 8.3.2 Web サービス開発プロセス

つまり、サービス提供側がサービスを公開し、動的にサービスを結合できる状況に至っても、そこからクライアントプログラムの作成が行われるため、開発プロセスにおいてかなりの時間的差異が生じてしまう。

そこで、企業内業務統合システムのような開発プロセスにおいては、その開発工程において、Web サービスの仕様書に当たる WSDL の作成が完了した時点で、サービス提供側およびサービス利用側双方ともに、WSDL から速やかにプログラム開発を完了できるフレームワークを提供する必要がある。それは、WSDL エディタであったり、WSDL インタプリタ、コードジェネレータであったりするのであろうが、こうした環境が提供されることによって、標準的な Web サービス開発プロセスフレームワークが実現される(図 8.3.3)。

さらには、ひとつの Web サービスの仕様である WSDL をサポートするだけに留まらず、複数の Web サービスの振る舞いを定義する WSFL（Web Services Flow Language）をも動的に扱える環境が実現されたならば、Web サービスは、爆発的に普及すると思われる。

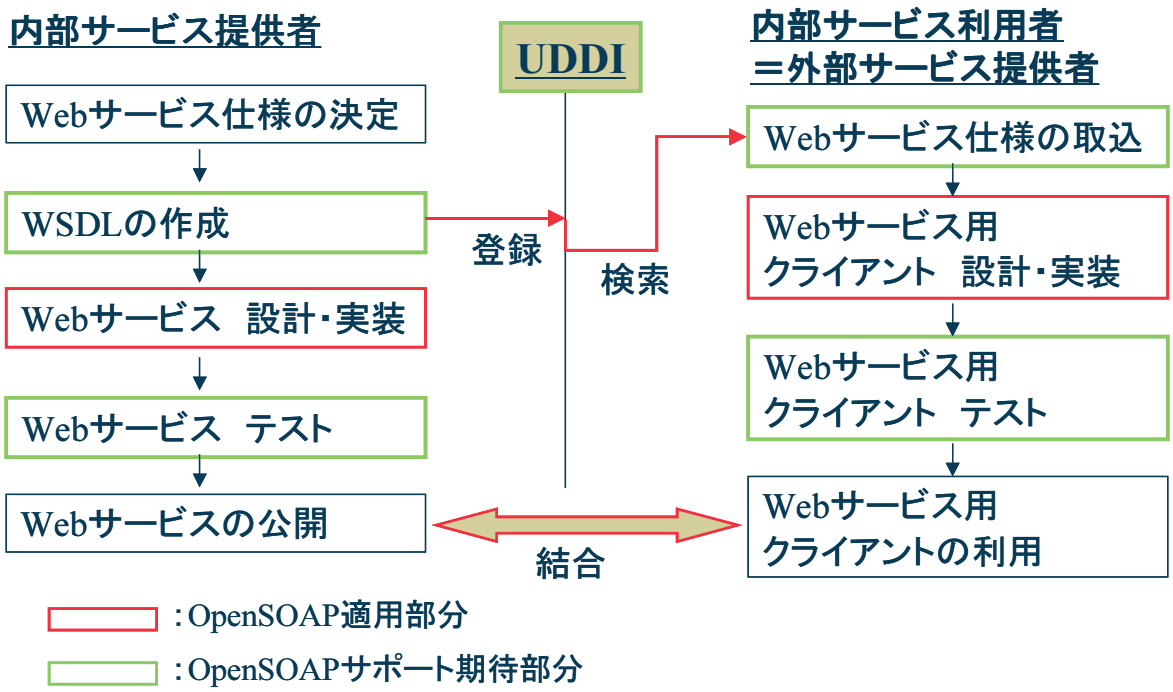

図 8.3.3 業務統合 Web サービスにおける開発プロセス

8.4. おわりに

本研究の成果である OpenSOAP ミドルウェアは、よりブラッシュアップが必要な課題もあるものの、複雑なビジネスプロセスソリューションを提供する Web サービスの構築に対して、十分適用可能であり、非常に有用であることが確認できた。

ただし、Web サービスの実際の利用・運用形態に関する部分における有用な機能は提供していない。もちろんこれらは、当初からの目的範囲外のものであるので当然のことなのではあるが、これまで論じてきたように Web サービスの本当の意味での運用を考える上で、決してはずしては考えられないものである以上、今後それらの機能についても、OpenSOAP コンソーシアムにおいて検討し、対応していくことが必要であることを提言する。