SOAに求められる信頼性を実現するWS-RX, WS-TX

山本展之 （(株)日立製作所）
西山英作 （(株)日立製作所）

本連載のテーマの 1 つである SOA（サービス指向アーキテクチャ）はシステムを複数のサービスの組合せとして構築する手法で，サービスの実現手段として Web サービスが有力な候補として期待されています．一方で，SOA はエンタープライズシステム向けに考えられたものであり，基本的な Web サービスである SOAP 通信だけでは実際のシステムに適用するには信頼性が十分ではありません．エンタープライズシステムで従来から用いられている高信頼メッセージングとトランザクションの信頼性も必要です．そこで，これらの信頼性を実現する既存の技術を Web サービス技術に取り込み，標準仕様として OASIS で策定されたのが WS-RX と WS-TX です（**図 -1**）．

WS-RX は，2 つのサービス間で確実にメッセージを送信するための転送プロトコルで，WS-RX TC（Web Services Reliable Exchange Technical Committee）にて策定され，2007 年 6 月に OASIS 標準として採択されました．なお，WS-RX というのは OASIS の技術委員会（TC）の名前で，厳密には仕様を表す名前ではありませんが，分かりやすさのため以下でも WS-RX と表記します．技術委員会の名前と仕様の名前が異なる理由についてはコラムを参照してください．

WS-TX は，複数のサービスに跨った更新処理の結果の整合性をとるためのトランザクションプロトコルで，2 フェーズコミットとロングトランザクションの 2 種類を規定しています．WS-TX TC（Web Services Transaction Technical Committee）で策定され，2007 年 4 月に OASIS 標準として採択されました．

以下では，WS-RX，WS-TX の順に解説していきます．

WS-RX

2 つのサービス間でメッセージを転送するとき，1 回のみの送信ではネットワーク上でのメッセージ消失や転送先サービスダウンなどの障害が起きてメッセージが到着しない場合があり，確実とはいえません．そのため，従来から用いられている高信頼メッセージングでは一般

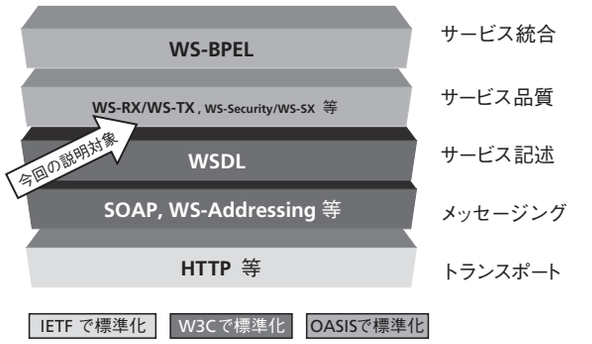

図 -1　WS-RX/WS-TX の位置づけ

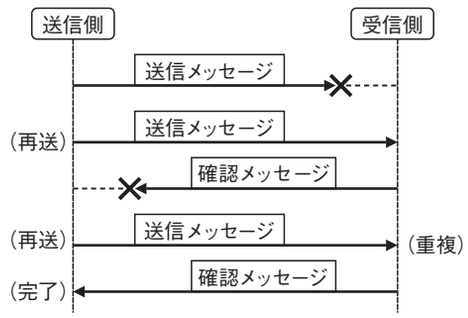

図 -2　WS-RX の AtLeastOnce 配送保証

的に，確実に転送するための方法として受信確認（Ack）が得られるまで再送を繰り返す考え方のプロトコルが用いられています．身近な例では TCP が同様の考え方のプロトコルです．Web サービスでは SOAP と親和性を持ったプロトコルが必要で，その標準として策定されたのが WS-RX です．WS-RX の利用場面としては，銀行間の送金や取引の発注書送信など確実なメッセージ転送が要求される業務処理が想定され，従来から高信頼メッセージングが必要とされる場所での Web サービスの利用が容易になると期待されます．

WS-RX は，ネットワーク上での障害として想定されるメッセージの消失，重複，順序不正に対処する 4 種類の配送保証を提供します．1 種類でないのは，後で説明するように信頼性を得るには，CPU，記憶域，通信量などのリソースコストが必要であり，利用要件に応じて信頼性を選べるように考慮されているためです．

AtLeastOnce（最低 1 回）配送保証は，送信メッセージを送達保証するために，確認メッセージが返ってくるまで送信メッセージを再送します（**図 -2**）．ただし，送信メッセージが消失した場合と確認メッセージが消失し

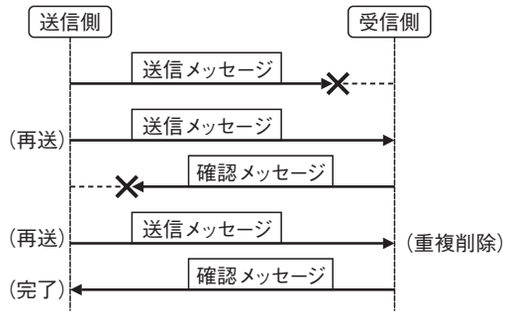

図 -3　WS-RX の AtMostOnce 配送保証

た場合を区別できないため，確認メッセージが消失した場合は受信側では重複が生じます．また，送信側は再送が必要になるのに備えて，成功するまではメッセージを何らかの手段で記憶しておく必要があります．

AtMostOnce（最高 1 回）配送保証では，受信側で重複削除を行います(**図 -3**)．AtMostOnce の名前のとおり，送信側は再送することは必須ではありません．受信側は，重複削除を行うため，メッセージが重複かどうか判断できるよう，メッセージを記憶しておく必要があります．

ExactlyOnce（正確に 1 回）配送保証は，AtLeastOnce と AtMostOnce の組合せに相当し，正確に 1 回だけ送信されます．送信側は成功するまで再送する必要があり，受信側は重複削除が必要です．

InOrder（順序保証）配送保証では，ネットワーク上でのメッセージ追い越しなどによる順序不正に対して順序を保証するため，メッセージに通番を付与し受信側で順序を復元します．受信側では，通番の順にメッセージが到着せず順序不正が発生した場合にそのメッセージを記憶して，追い越されたメッセージが後から到着するまで待ってから順序を整列させる必要があります．

以上のように，これら 4 種類の配送保証は信頼性とリソースコストの特性が異なっており，利用要件に合わせて使い分けることができます．また，これら 4 種類の配送保証が従来から高信頼メッセージングで一般的に用いられている概念と同じであることから，WS-RX は高信頼メッセージングを必要とするエンタープライズシステムを含むビジネスユースで Web サービスを信頼して利用できるようにするものといえます．

WS-RX は 2 つのサービス間で確実にメッセージを転送するためのものでしたが，以下では，エンタープライズシステムで重要なもう 1 つのトランザクションについて解説していきます．

WS-TX

従来から，トランザクションは複数のデータベースなどのリソースに対する更新処理の結果を同期させるために用いられてきました．代表的なものが厳密性を考えた 2 フェーズコミット制御ですが，欠点もあるため，ロングトランザクション制御も用いられています．2 フェーズコミットはアルゴリズムが厳密で，障害からの回復はシステムが自動的に行うためアプリケーションプログラムがトランザクションを意識する必要がなく便利です．しかしその一方，後で触れるようにデータベースへのプリペア中はロックを保持する必要があるため，プリペア中に 1 つのデータベースが停止すると残りのデータベースのロックが保持されたままとなり，結果的にシステム全体の処理が進まなくなる可能性を持っています．そのため，2 フェーズコミットは密結合なシステムで主に使われます．一方，Web サービスの対象の 1 つである企業間取引などでは 2 フェーズコミットのように 1 カ所の停止が全体に影響を及ぼす密結合な制御は不向きで，より疎結合な制御が必要です．ロングトランザクションは従来からこのような場所で用いられてきました．2 フェーズコミットの裏返しとなりますが，ロングトランザクションではアプリケーションプログラムも回復を意識する必要があり，プログラミングはやや面倒になります．WS-TX（Web Services Transaction）はこれら両方のトランザクションを Web サービス上で実現するための標準として策定されました．

WS-TX は，次の 3 つの仕様の総称です．

- WS-AtomicTransaction（Web Services Atomic Transaction）
- WS-BusinessActivity（Web Services Business Activity）
- WS-Coordination（Web Services Coordination）

WS-AtomicTransaction は，ごく短時間で完了するトランザクション処理における 2 フェーズコミットを Web サービス上で実現するためのプロトコルです．一方，WS-BusinessActivity は比較的長時間あるいは長期間に及ぶ連鎖的なトランザクション，たとえばワークフロー処理などにおける補償トランザクションを実現するプロトコルです．これらのプロトコルに基づいてサービス間で交換されるメッセージの形式は WS-Coordination で規定されています．以下，順に解説していきます．

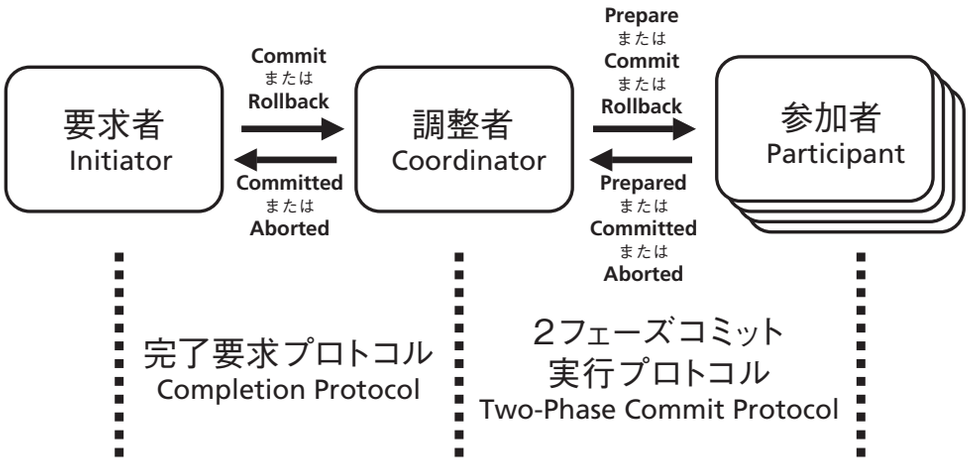

図 -4
WS-AtomicTransaction のモデル

◉ WS-AtomicTransaction

WS-AtomicTransaction では，**図 -4** に示すモデルを採用しています．このモデルでは，以下の 3 種類のエンティティが存在します．

• 要求者（Initiator）
2 フェーズコミットの実行を要求する．通常はアプリケーション．

• 調整者（Coordinator）
要求に基づき，2 フェーズコミット処理を制御する．いわゆるトランザクションモニタ．

• 参加者（Participant）
ある単一のリソースを制御する．たとえばデータベース管理システム．

このモデルに基づき，WS-AtomicTransaction では 2 種類のプロトコルを規定しています．

• 完了要求プロトコル（Completion Protocol）
要求者が調整者に 2 フェーズコミットの実行を要求し，その結果を受け取るためのもの．

• 2 フェーズコミット実行プロトコル（Two-Phase Commit Protocol）
調整者が各参加者との間で 2 フェーズコミットを実行するためのもの．

トランザクションが最終段階に達してコミットが必要になると，要求者が完了要求プロトコルに基づいて調整者に「コミット要求（Commit）」「ロールバック要求（Rollback）」のいずれかのメッセージを送ります．調整者は，要求者からのメッセージ通りの処理を 2 フェーズコミットのロジックに基づいて試みた後，その処理全体の結果として「コミット完了（Committed）」（すべてのリソースがコミットされた）または「中止（Aborted）」（すべてのリソースがトランザクション開始前の状態に戻された）のいずれかを応答メッセージとして要求者に返します．実際の 2 フェーズコミット処理の実行は，次に述べる 2 フェーズコミット実行プロトコルで規定されています．

2 フェーズコミット実行プロトコルでは，伝統的な 2 フェーズコミットのアルゴリズムに従い，次のようなシーケンスが定義されています（**図 -5**）．

1. 調整者は要求者から 2 フェーズコミット実行の要求を受けた後，各参加者に「コミット準備要求（Prepare）」メッセージを送る．
2. 各参加者は自分が制御するリソースに対して，コミットをいつでも実行でき，かつロールバックも行える状態にすることを試みる．それに成功した場合は「準備完了（Prepared）」を，失敗した場合には「中止（Aborted）」を調整者に返す．
3. 調整者はすべての参加者から「準備完了（Prepared）」が返ってきた場合には，各参加者に「コミット要求（Commit）」メッセージを送る．これを受け取った各参加者はコミット処理を実行した上でこのトランザクションのためのリソースを解放し，「コミット完了（Committed）」を調整者に応答する（図 -5 の左側）．一方，もしいずれかの参加者から「中止（Aborted）」の応答があった場合には，全参加者に対して「ロールバック要求（Rollback）」メッセージを送り各参加者はロールバック処理を実行した上でこのトランザクションのためのリソースを解放し，「中止（Aborted）」を調整者に応答する（図 -5 の右側）．

上記のシーケンスは，何十年も前に確立され，トランザクション処理の教科書にも必ず登場しますので皆さんには大変なじみ深いものだと思います．WS-AtomicTransaction の 2 フェーズコミット実行プロトコ

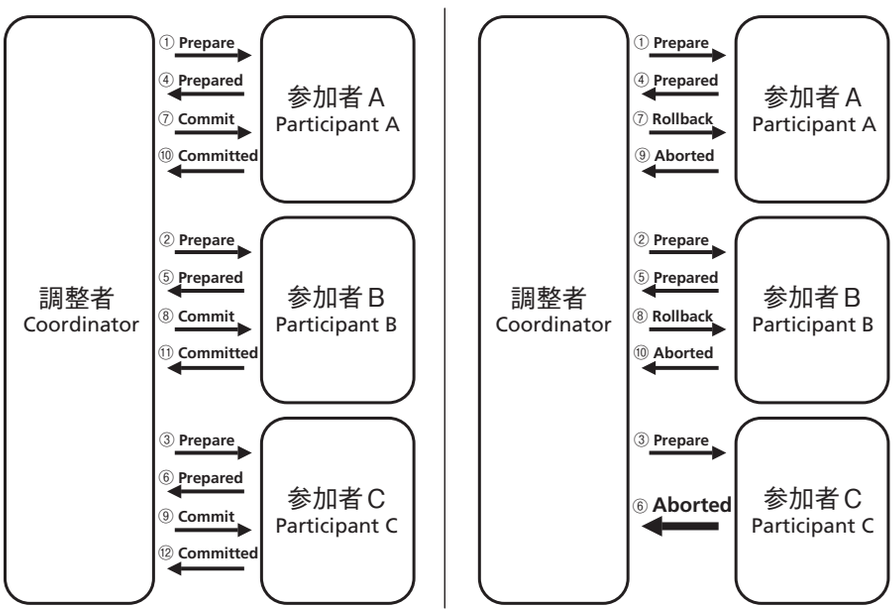

図 -5
WS-AtomicTransaction の 2 フェーズコミットプロトコル

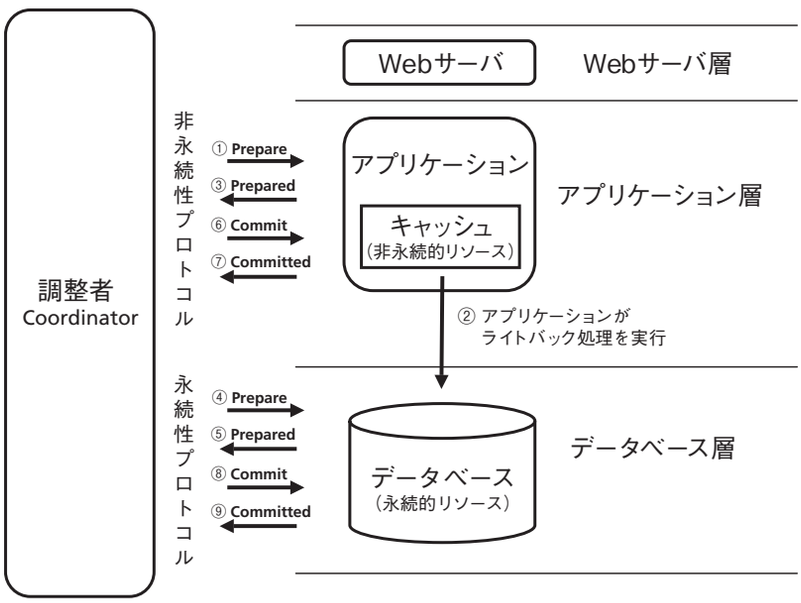

図 -6
WS-AtomicTransaction の永続性プロトコルと非永続性プロトコル

ルでは，これに加えて次に述べるような工夫がこらされています．

〈キャッシュ同期への対応〉

伝統的な 2 フェーズコミットの考え方では，トランザクションに関係するリソースはいずれも永続的であることを必須の条件にしています．つまり，それぞれのリソースはデータベースなどの安定した記憶装置を備えており，コミットとロールバックのどちらでも直ちに実行可能な状態を作り出すための情報を確実に（少なくともトランザクションが完了するまでは）蓄積できることが前提となっています．しかし，この考え方が確立したのはずいぶん昔のことであり，現在一般的な多階層 Web アプリケーションシステムにはなじまない部分があります．たとえば，システムを Web サーバ層・アプリケーション層・データベース層の 3 つに分割するいわゆる 3 階層システムでは，データベース層が最終的に管理しているデータの非永続的な複製，つまりキャッシュをアプリケーション層に保持させることによりシステム全体の処理性能を向上させる手法をとることがよくあります（**図 -6**）．このようなケースでは，キャッシュとデータベースを適切に同期させる必要があります．特に，そのキャッシュを使用しているアプリケーション以外の何者かがデータベース本体を更新しようとする際は，その更新処理よりも前にダーティキャッシュをデータベースに反映させておかなければ矛盾が発生してしまいます．伝統的な 2 フェーズコミットの概念では，各リソースはそれぞれ独立した存在であって順序性を持たないことが

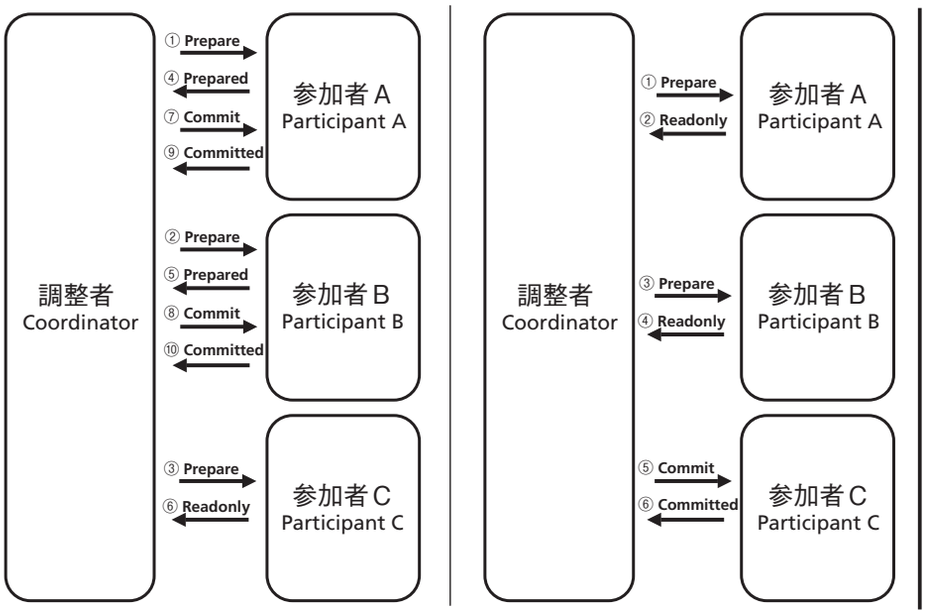

図 -7
WS-AtomicTransaction の読み取り専用最適化

前提となっているため，キャッシュに対する準備要求やコミット要求がデータベース本体に対するものに先立って完了することは保証されません．この問題を解決するために，WS-AtomicTransaction の 2 フェーズコミット実行プロトコルには，永続性プロトコルと非永続性プロトコルの 2 種類が用意されています．各参加者は，自分が管理するリソースの性質に応じて，2 種類の 2 フェーズコミット実行プロトコルのどちらを利用するか選択し，それをトランザクションの実行前に調整者にあらかじめ通知しておきます（この手順も WS-Coordination で規定されています）．調整者は，2 フェーズコミット実行の要求を受けると，まず非永続性プロトコルを利用することを通知した参加者（たとえばキャッシュを管理するアプリケーション）すべてに対してコミット準備要求メッセージを送り，いずれも準備完了メッセージを返してきたことを確認した上で，次に永続性プロトコルを利用する参加者（たとえばデータベース管理システム）へ準備要求メッセージを送ります．これによってキャッシュとデータ本体の適切な同期を保証しています．

〈読み取り専用最適化〉

いわゆる ACID（Atomicity：原子性，Consistency：一貫性，Isolation：独立性，Durability：永続性）を完全に満たす 2 フェーズコミットのアルゴリズムは強固な信頼性を持ちますが，その一方でシステム全体のパフォーマンスを大きく低下させかねないことはよく知られています．これは，いったん 2 フェーズコミット処理が開始されると，それが対象とする全リソースが処理全体の完了までロックされ，その間は他のトランザクションからのアクセスが一切拒否されることに起因します．WS-AtomicTransaction の 2 フェーズコミット実行プロトコルでは，この問題を軽減するための技法の 1 つである「読み取り専用最適化（read-only optimization）」をサポートしています．ある参加者がコミット準備要求メッセージを受け取った際，実際にはデータの更新が発生しないことがあり得ます．このような場合，続いて調整者から送られる要求がコミットであってもロールバックであっても，その参加者はそれに対して何も処理する必要はありませんし，他の参加者すべてがコミット準備要求への応答を返して調整者からコミット要求またはロールバック要求が送られるのを待たなければならない理由もありません．したがって，直ちに 2 フェーズコミットから離脱し，リソースのロックを解除して他の処理に移ればシステム全体のパフォーマンスを向上させることができます．WS-AtomicTransaction では，そのような判断を参加者が下した場合，その参加者はコミット準備要求への応答として「読み取り専用（Read-Only）」を返すことによって，トランザクションからの離脱を調整者に伝えることができます．調整者はこの応答を返した参加者に対してはコミット要求またはロールバック要求を送る必要はありません（**図 -7** の左側）．また，最後の 1 つの参加者にコミット準備要求を送るまでの間に，他のすべての参加者から読み取り専用の応答が返ってきた場合は，最後の参加者に対してはコミット準備要求を送らず直ちにコミット要求を送ることにより処理を簡略化することができます（図 -7 の右側）．

いずれの技法も WS-AtomicTransaction で初めて登場したというものではなく，これらをサポートしているトランザクションモニタ実装は数多くあります．WS-

AtomicTransaction はそれらの技法が実システムで必要とされている実情をきちんと考慮した上で策定された十分実用的なものであるといえます．

◉ WS-BusinessActivity

単一のアトミックなトランザクションに対する 2 フェーズコミットは全リソースに対する更新が確定するか，もしくは全リソースがトランザクション開始前の状態に戻されるかの二者択一の世界ですが，ビジネスプロセスのように，いくつものアトミックトランザクションの連鎖として定義されるトランザクションでは，その一部の実行だけを確定させたり取り消したりする必要があります．しかも，確定や取り消しは一度だけではなく，何回も発生することがあり得ます．WS-BusinessActivity はこのような複雑なトランザクションの制御について規定しています．WS-BusinessActivity のモデルでは，WS-AtomicTransaction と同じく調整者と参加者が存在します（ただし要求者は存在しません）．調整者は参加者に対して，以下の要求メッセージを送ることができます．

- Complete：処理を完了させる
- Close：完了済みの処理を終了させる
- Compensate：完了済みの処理に対する補償トランザクションを実行する
- Exit：完了前の処理を終了させる

WS-AtomicTransaction とは異なり，調整者は各参加者にそれぞれ異なる要求メッセージを送ってもかまいません．また，参加者は調整者からの完了要求を待つのではなく，処理が完了したことを自ら判断して調整者に伝達することもできます（ただし，各参加者はあらかじめどちらかを選んで宣言しておかなければなりません）．

◉ WS-Coordination

トランザクションの開始から完了までの間，WS-AtomicTransaction と WS-BusinessActivity が規定するプロトコルに基づいて調整者と参加者（あるいは要求者）との間で交わされるメッセージの形式を定めているのが WS-Coordination です．交換されるメッセージが，どのトランザクションに関するものなのかは，メッセージ中の CoordinationContext 要素で指定します．WS-Coordination は，これに加えて，トランザクションの開始前に参加者が調整者と交わしておかなければならない以下のようなメッセージを規定しています．

- Activation service：トランザクションの識別子（CoordinationContext 要素）を登録する
- Registration service：ある CoordinationContext が示すトランザクションに対して自分自身を参加者として登録する

◉ WS-TX の課題

以上の説明からお分かりになるとおり，WS-TX はトランザクションやビジネスプロセスそれ自体を定義するものではありません．それらは何らかの手段によってあらかじめ定義されている必要があります．つまり，識別子である CoordinationContext 要素がどのトランザクションあるいはビジネスプロセスを指すのか，そして必要な処理は具体的にはどのようなものなのか等を，調整者や参加者がすでに知っていることを前提にした上で実行時の調整プロトコルを定めているのが WS-TX なのです．ビジネスプロセスを定義するための記述言語としては，本連載の第 6 回〜第 8 回で紹介された Web Services Business Process Execution Language（WS-BPEL）などが存在しますが，WS-TX とそれらの記述言語との関係はまだ明確に定義されていません．Web サービス上で分散トランザクション処理やビジネスプロセス処理を実行するシステムの開発や運用を容易なものとするためには，ビジネスプロセス記述言語，特に現在最も有望視されている WS-BPEL との関係を何らかの形で早期に明確にすることが望まれます．

結び

WS-RX と WS-TX の登場によって，信頼性が重視されるエンタープライズシステムを SOA で実現するための道具立てはほぼ揃いました．前述のとおり，残された課題もありますが，遠からず解消され SOA の普及に弾みをつけるものとなるでしょう．

（平成 19 年 7 月 9 日受付）

山本展之（正会員）　nobuyuki.yamamoto.vw@hitachi.com

1990 年（株）日立製作所入社．現在，SOA 関連技術の研究開発に従事するとともに，OASIS において Web サービス関連の標準化活動に参加．

西山英作　eisaku.nishiyama.dd@hitachi.com

1988 年（株）日立製作所入社．現在，仮想化技術の研究開発に従事するとともに，OASIS，W3C，WS-I などにおいて Web サービス関連の標準化活動に参加．

高信頼メッセージング標準仕様の名称と経緯

本文で触れたように，今回とりあげた 2 つの仕様のうち WS-RX は，標準化団体 OASIS の技術委員会（TC）の名称です．その TC が策定し今回解説した仕様の名称は Web Services Reliable Messaging（WS-ReliableMessaging）といいます．WS-ReliableMessaging は WS-RM と略されることが多いのですが，本稿ではこれを用いず，TC 名の略称である WS-RX を仕様の呼称としました．これには理由があります．

実は OASIS には，Web サービス上での高信頼メッセージング標準仕様の策定を目的とした TC が本年 4 月までもう 1 つ存在しており，その TC の名称が Web Services Reliable Messaging TC（WSRM TC）だったのです．WSRM TC が策定した高信頼メッセージング標準は Web Services Reliability（WS-Reliability）という名称であり，WS-R と呼ばれることもあります．この両者の関係をまとめると**表 -1** のとおりになります．

ご覧のとおり，WS-RM または WSRM と表記した場合や口頭で「ダブリュ・エス・アール・エム」と呼んだ場合，それがどちらを指しているか曖昧になります．そこで，本稿では混同を避けるため，「WS-RX TC が策定した標準仕様 WS-RM」をあえて「WS-RX」と呼称しています．実際，Web サービスの高信頼メッセージングに携わる人々の間では仕様としての WS-RM を WS-RX と呼ぶことは一般的であり，筆者らの独断ではないことはご承知おきください．

ところで, なぜこのように紛らわしい名称を持つ 2 つの仕様, 2 つの TC ができてしまったのか疑問に思う方がいらっしゃると思います．そこで，高信頼メッセージングの歴史をひもときながら，ことの経緯についてご説明します．

高信頼メッセージングの歴史は長く，1960 年代に現在のメインフレームの原型となるアーキテクチャが完成するのとほぼ同時に実用化されました．その後長らく高信頼メッセージング機能は有償の商用製品だけが提供しているという時期が続きます．当時，高信頼メッセージングを必要とするのはミッションクリティカルな企業内システム，あるいは密な関係を持つ企業連合の共同システムに限られており，そのようなシステムは大手ベンダやインテグレータが一括して（または強力に主導して）構築されるのが当然でした．そのため，高信頼メッセージングのプロトコルが特定ベンダのプロプライエタリなものしか存在しなくても，また，そのプロトコルの詳細が公開されていなかったり有償であったりしても，大きな問題とはなりませんでした．

一方，21 世紀を迎えるころから登場した Web サービスでは多数のシステムが企業や団体の垣根を越えて連携することを目指しているため，「誰もが同じものを，安価に，安心して，簡単に使える仕様」が求められていました．XML，SOAP，UDDI，WSDL などの Web サービスの基本技術はいずれも標準が確立され，しかもロイヤリティを支払うことなく無償で利用できることが明言されています．

当初は参照系での利用が主であった Web サービスも徐々に企業などの実システムに適用されるようになり，ミッションクリティカルな基幹システムには不可欠な高信頼メッセージングを Web サービス上で実現するための仕様の標準化が望まれるようになりました．

これに応えるために開発されたのが WS-Reliability 仕様です．WS-Reliability は，ビジネスグリッドコンピューティング研究開発事業の一環として開発されました．ビジネスグリッドコンピューティング研究開発事業は 2003 年春から 2006 年春まで経済産業省，独立行政法人情報処理推進機構（IPA），独立行政法人産業技術総合研究所，および株式会社日立製作所，富士通株式会社，日本電気株式会社が共同で推進した日本の国家プロジェクトです．

このプロジェクトの目的は，先進のグリッド技術を適用したビジネス向けミドルウェアを開発して日本の産業の国際競争力を向上させるとともに，ビジネスシステムにグリッド技術を適用した場合のシステムインタフェース（これには通信

標準仕様を策定する OASIS TC の名称	その TC が策定する標準仕様の名称
正式な名称：Web Services Reliable Messaging TC 正式な略称：WSRM TC	正式な名称：Web Services Reliability 正式な略称：WS-Reliability 通称：WS-R
正式な名称：Web Services Reliable Exchange TC 正式な略称：WS-RX TC	正式な名称：Web Services Reliable Messaging 正式な略称：WS-ReliableMessaging 通称：WS-RM

表 -1　WSRM TC と WS-RX TC

プロトコルが含まれます）を国際標準として提案することにより，グリッド技術のビジネスへの適用という大きなテーマについてグローバルに貢献することにありました．対象がビジネスシステムですから，信頼性とくにグリッドコンピューティングの基本技術である Web サービスにおけるメッセージングの高信頼化技術を確立することがまず求められました．

そこで同プロジェクトに参画していた上記企業グループは，高信頼メッセージング仕様 WS-Reliability を開発し，これを米オラクル社や米サン・マイクロシステムズ社などの海外有力企業と共同で 2003 年 2 月に標準案として OASIS へ提案しました．これを受理した OASIS によって設置された WSRM TC が検討を進めた結果，2004 年 11 月に WS-Reliability は OASIS 標準として成立しました．

WS-Reliability が画期的だったのは，OASIS に提案する以前から，完全にロイヤリティフリーであることを宣言していた点です．当時，Web サービスで利用する仕様のうち，前述の XML や SOAP などの基本的なものはロイヤリティフリーであると宣言されていましたが WS- で始まる名称を持つ一連の応用的な仕様（以下 WS-*）はたいへん曖昧な状態にありました．一部の仕様は RAND（Reasonable And Non-Discriminatory の略．妥当なロイヤリティの支払いを求めるが，誰に対しても差別的な取り扱いをしないことを保証する）条件で利用を許諾することが宣言されていたものの，無償であるとも有償であるとも明言されていないものが大半を占めていたのです．Web サービスおよび，それを基礎技術として用いるグリッド技術が広く普及し実ビジネスシステムに適用されるためには，開発者や利用者が不安を抱くことがないようロイヤリティの条件が明確になっていることが不可欠です．そして，その条件として最適なのはロイヤリティフリーすなわち無償である，と WS-Reliability の開発に参画した企業グループは考えました．そこで，最初に発表した時点からロイヤリティフリーを宣言するという，当時としてはたいへん革新的な姿勢で WS-Reliability を世に送り出したのです．

WS-Reliability が画期的だった点はもう 1 つあります．それは上記企業グループは仕様のドラフト版の執筆を完了したと同時に OASIS というオープンな標準化団体に提案し，仕様の完成をそこに委ねたことです．WS-* の中には，有志企業グループによって仕様が公開された後，世の中に浸透してデファクト・スタンダードの地位が固まってから標準化団体に提案されることがよくあります．このようなケースでは，標準化団体での標準化作業において，仕様をより良いものにするための修正を加えることがもはや困難である，という問題が発生してしまいます．しかし WS-Reliability では，いわゆる「叩き台」としての仕様は上記の企業グループが作成しましたが，それが完了すると同時に OASIS に提案することによって，オープンかつ自由闊達な議論を通じた標準化を可能にしました．また，標準化活動と並行して，その時々での OASIS の最新ドラフト仕様に基づいて実施した普及活動においても，相互接続実験は招待制ではなく誰でも参加できる形で実施したほか，実装のソースコードを公開した際はオープンソースライセンスに基づいて無償かつ改変・再配布をも認めるなど，オープンな姿勢を貫きました．

現在では WS-* の多くが標準化団体への提案前からライセンス条件を明記するようになり，また標準化団体へ提案されるタイミングも数年前よりもかなり早くなっていますが，それには WS-Reliability が非常に大きな影響を与えたことを知っていただきたいと思います．

さて，もう 1 つの高信頼メッセージング仕様である WS-RX こと WS-RM ですが，これを開発した企業グループによって 2005 年 4 月に OASIS へ提案されました．WS-Reliability が OASIS 標準として成立した翌年のことです．すでに WS-Reliability が OASIS の公式標準仕様として存在するにもかかわらず，同じ分野で別の標準を策定することの是非について論議が巻き起こりましたが，最終的には OASIS によって提案が受理され，2005 年 6 月に WS-RX TC が発足しました．WS-Reliability を OASIS に提案した企業グループ全社も WS-RX TC の発足と同時にこれに参加し，オープンな場でさまざまな立場から自由に議論するという OASIS の理念に基づいて検討が続けられた結果，本稿を執筆中の本年 6 月に OASIS 標準として正式に成立しました．これに先立つ本年 4 月，Web サービス仕様の標準化のあり方に大きなインパクトを与えた WSRM TC はその歴史的な役割を終え，TC としての活動を公式に終了しています．

筆者らは WSRM TC および WS-RX TC の両方に発足当初から参加し，4 年半にわたって両仕様の提案から成立までの過程をつぶさに見てきました．Web サービスのような「繋がらなければ意味がない」技術において何よりも重要なのは「標準」の存在であることは言うまでもありませんが，ある標準が健全に成立し健全に普及するためには，標準化プロセスがオープンな場でなされるべきであり，そして利用条件が明確であるべきであるということを繰り返し強調させていただきたいと思います．