ホワイトペーパー

SOA アプリケーションの構築と SOA プラットフォームの機能

- Fiorano Software の SOA コンセプト -

Fiorano
Enabling change at the speed of thought

Fiorano Software, Inc.

日本オフィス

東京都千代田区外神田 3-13-2

03-6777-7530

▶ はじめに

企業におけるシステム インテグレーションやミドルウェアに関わる問題を軽減するために、通信、コネクティビティ、データ変換、ポータビリティ、セキュリティなどの分野で多くの標準規格が定められてきました。標準規格が充実してきたことが主な起因となって SOA (サービス指向アーキテクチャ) というソフトウェア パラダイムが登場し、大きな注目を集めています。

『SOA とは、

1. ビジネス上の業務や機能をサービスという単位でまとめ、
2. 個々のサービスはそのインタフェースを他から利用できる形で定義し、
3. サービスを組み合わせることで、アプリケーションを構築

することである』、と定義できます。こうして構築された SOA アプリケーションでは、サービス同士がデータを交換しながらアプリケーション全体のロジックを実行していきます。各サービス間はゆるやかに結合 (疎結合) されたもので、あるサービスの内部実装が変更されても他のサービスに影響を与えません。SOA のこのような考え方は従来からありましたが、これをサポートする標準規格が多く定義されてきたことで、より現実性を持って再登場してきたといえます。標準規格に則ることで、特定ベンダーによる『囲い込み』を避けられ、汎用機、複数の異なるパッケージ製品や DBMS 製品、異なる OS の上で稼動するアプリケーション プログラムなどが混在する企業が現実に抱えている IT 環境に対応可能となるからです。旧 EAI 製品が失敗した主な理由もこのあたりにありました。標準規格が充実し、それに則ることが、SOA の成功の鍵となっています。

しかしながら、SOA 自体はアーキテクチャとして標準規格化されたものではなく、アプリケーション インテグレーションの方法についてその大枠の考え方を示しているにすぎません 。前述の定義の言葉を借りれば、「サービスとしてまとめる方法」、「サービスの組み合わせ方法」などの標準化はなされておらず、また、SOA に基づくアプリケーションの実行環境 (プラットフォーム) の標準化もなされていません。アーキテクチャとしての SOA が将来標準規格化される可能性もありますが、少なくとも現状では SOA 関連製品は各ソフトウェア ベンダーの考え方に基づいて開発されており、アーキテクチャ規格のみならず SOA アプリケーションのモデリング方法や具体的な実装方においてもベンダーの考え方は異なっています。

このホワイト ペーパーでは、SOA のこのような問題点に焦点をあて、SOA アプリケーションの構築のあり方や SOA 製品の持つべき機能などについて、Fiorano Software の考え方を紹介いたします。

1. SOA によるインテグレーション

1.1 アプリケーションのインテグレーションとは ?

アプリケーション インテグレーションとは、『個別に独立したものとして設計 / 開発されたアプリケーションを複数組み合わせ、それらを協調して稼動させることでビジネス プロセスを構築していく作業である』、と定義できます。インテグレーションは一般的に難しいものであります。それは、各アプリケーションがデータ、プロセス、オブジェクト モデルなどの面で異なる設計コンセプトを用いて開発されているためです。加えて、それぞれのアプリケーションの稼動環境であるオペレーティング システム、ミドルウェア、データベースなどのアーキテクチャが異なっている場合がほとんどで、これら複数の異なるアーキテクチャ間で共通して稼動するソリューションを開発しなければならい点もアプリケーションのインテグレーションを難しいものとしています。

ガートナー社 (Gartner) によれば、企業規模の大小に関わらず企業アプリケーションの設計、開発、メンテナンスの費用の 35% はアプリケーション インテグレーションに費やされています。現代の企業にとってインテグレーションの必要性は明白なものとなっており、その関心事はインテグレーションを行うか否かではなく、どのようにインテグレーションするかにその重点が移ってきています。最も効果的なインテグレーション戦略やデザイン パターン、必要となるソフトウェア ツールとはどのようなものでしょう?

この問いに対する業界の一致した答えが、SOA です。しかしながら、SOA 自体が標準規格として定められていないこともあって、SOA によるインテグレーションの具体的な方法はベンダー毎に異なったものとなっており、その製品もまた異なるものとなっています。

この章ではアプリケーション インテグレーションとはどのようなものであり、インテグレーションがなぜ必要であるのか考えてみます。それによって、SOA に基づくインテグレーションとはどのようなものであるべきかを議論します。以降の章では、より効果的なインテグレーションの実現方法とそれを支援する SOA 関連製品について議論することとします。

1.2 アプリケーション インテグレーションが必要な理由

インテグレーションは難しくコストが嵩むものであるにもかかわらず、(良い結果が得られたかどうかは別にして) 多くの企業がインテグレーションを実施してきました。インテグレーションを効果的に実現できれば、成果をすぐには得られなくとも長期的には費用の削減となることを、IT 部門の管理者は理解していたからです。しかしながら、最終的な決定権限を持っている役員といった上位管理者には数年先に成果があらわれる費用削減よりも単年度の利益に重きをおく傾向があり、インテグレーションにあまり予算を配分しないできました。その結果、インテグレーションを実施するにしても、緊急を要するごく一部を対象とした部分的でアドホックなインテグレーションとなってしまいがちでした。また一方、費用効果が高く技術的にも優れたインテグレーション技術や方法論、それを具現化したインテグレーション製品やプラットフォーム製品がほとんど存在しなかったこともその一因でした。

しかし、今日では、多くの上位管理者もインテグレーションの重要性を認識しています。現在のビジネス環境をとりまく問題を解

決するためには、他に選択の余地がないからです。今日では、インテグレーションの要求は IT 部門内から起こるのではなく、むしろ IT 部門の外からもたらされます。その動機には、企業のビジネス戦略、公的機関による勧告や法的な規制、B2B の実施を依頼してくる取引先企業など、次に挙げるように多種多様な要因があります。

- ビジネス環境の変化に迅速に対応 (アジャイルな企業システム)
- ビジネの可視化 (SOX 法の施行)
- 単独のパッケージ ソリューション (ERP、CRM など) の限界
- 旧来の EAI の限界
- IT コストの削減 (開発、メンテナンス)、ROI の改善

1.3 アプリケーション インテグレーションのパターン

アプリケーションのインテグレーションは、図 2 に示すように 3 つのパターンに大別できます。

- データ統合 (データの整合性のためのインテグレーション)
- ビジネス プロセスを構築するための複数ステップからなるインテグレーション
- コンポジット アプリケーション

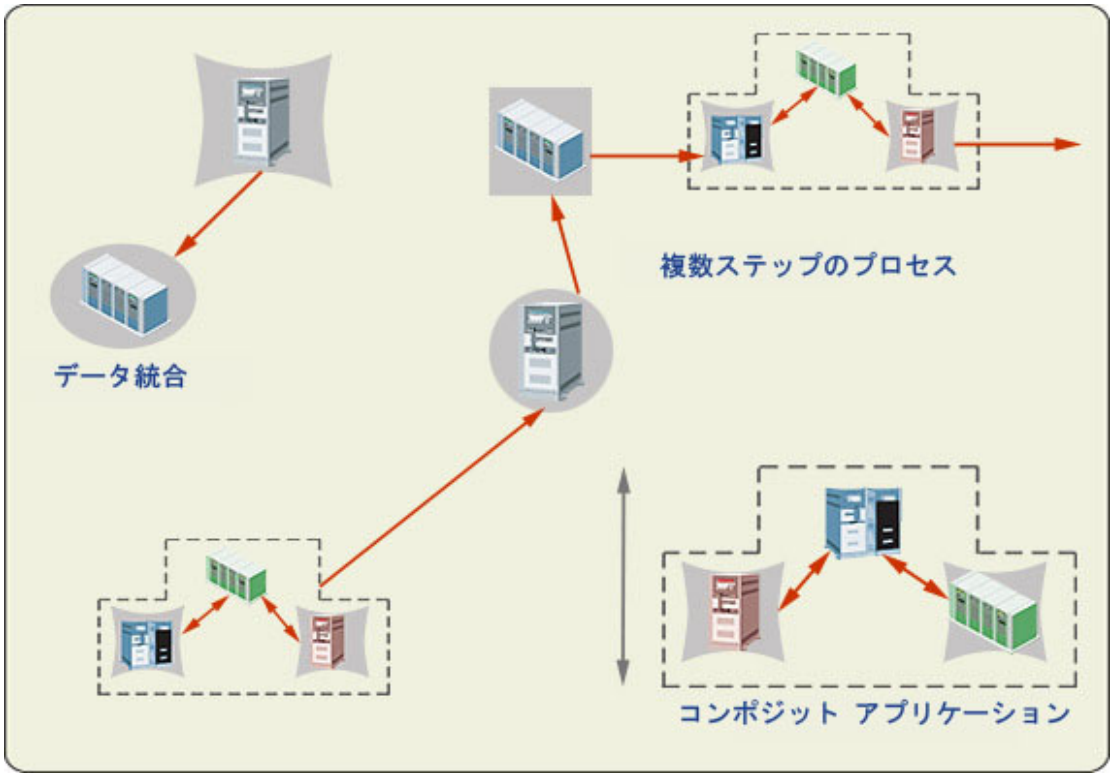


図 1 – アプリケーション インテグレーションのパターン

1.3.1 1対 n のインテグレーション パターン (データ統合など)

1 対 1 または 1 対 n のインテグレーションが、最もプリミティブなものです。このタイプの典型的な具体例は、アプリケーション間でデータの整合性を維持する目的で実施するデータ統合のインテグレーションです。データ統合インテグレーションでは、別々の部門や異なる拠点で稼動しているアプリケーションの間でデータの整合性を維持したり同期をとるために実施されます 。対象が同じデータを複数のデータベースで保持しているようなケース (例えば、顧客住所を複数のデータベースでそれぞれ保持している場合) では、データの整合性を維持することが常に問題となります。このようなケースでは、あるアプリケーションがデータを変更すると、他のアプリケーション側のデータも同じように変更する必要が生じます。これらのアプリケーションの関係は、1 対 1 または 1 対 n の関係にあり、従来のメッセージング ミドルウェア (MOM) でも対応することができます。つまり、データの変更、新規追加、削除などの事象 (イベント) が発生した時に、それらのイベント (例えば、"変更イベント") を同じデータを保持しているアプリケーションにイベント通知メッセージとして配信することで実現します。1対1の場合には、メッセージ キューを、1 対 n の場合にはトピック (パブリッシュ – サブスクライブ) を使用します。このような、ある事象 (イベント) の発生を通知することでアプリケーション間の連携をとる方式を、イベント ドリブン方式と呼びます。

MOM などのメッセージングにおいては、ネットワークや通信機能の障害が発生してもメッセージの送信側と受信側のアプリケーションはそれぞれ独立して処理を続けられる **(注 1)** という意味において、センダー (送信側) とレシーバー (受信側) は論理的に独立しており、それぞれのアプリケーションの処理は非同期に進められます。この独立性と非同期性は、そのままイベント ドリブン方式の特徴にもなっています。

業務内容によっては、アプリケーション間でのデータ変更が厳密に同期していることを求められることがあります。このような場合には、もっとも一般的な方法としてアプリケーション間の同期を取るためにリクエスト – リプライ方式を採用します。この方式では、センダーが送った "メッセージ (データ変更のリクエスト)" を正しく受け取り、リクエストの処理が終了したことを知らせる "メッセージ (リプライ)" がレシーバーから返ってくるまで、センダー側では次の処理に移行しません。これによって、アプリケーション間の同期がとれることとなります。さらに、両アプリケーション間のデータ変更をトランザクション処理として捉え、ACID 特性に基づくトランザクション管理を実装することで、より厳密な管理を実施することも考えられます。

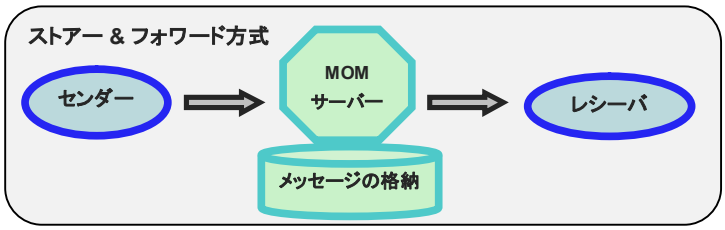
リクエスト リプライ方式をサポートするミドルウェアには、旧 EAI 製品、アプリケーション サーバー製品、CORBA など多くのものがあります。また、RPC (リモート プロシージャ コール) や RMI (リモート メソッド インボケーション) などのプログラミング技術も、リクエスト – リプライ方式を実現します。なお、MOM などのほとんどのメッセージング ミドルウェアでは、メッセージを受け取った側からのリプライを必要とするメッセージング方式をサポートしており、アプリケーション間の同期をとることも可能となっています。

データの整合性や同期のためのデータ統合インテグレーションの特徴をまとめると、次のようになります。

- 独立したアプリケーション間は 1 対 1 または 1 対 n の関係で、イベント ドリブン方式を採ります。業務内容によっては、リクエスト – リプライ方式とします。
- イベント ドリブン方式の場合、一方向 (一方通行) の通信
- リクエスト – リプライ方式の場合、リクエストとそれに対するリプライという双方向の通信

- イベントドリブン方式 ： 何らかの事象が発生した時に、その発生を知らせるイベント通知をメッセージ (データが含まれることもある) として他のアプリケーションに送り、自身は次の処理に移行する。相手側のアプリケーションとは独立しており、相手側の制約をうけない疎結合
- リクエスト－リプライ方式： 相手側からのリプライが返ってくるまで、リクエストを送った側は待機し、次の処理に移行しない。相手側のアプリケーションの状態に依存しているという意味で、イベントドリブン方式よりは密な結合 (RPC や RMI などのプログラミング レベルでの結合と比べれば、結合度は低い)

注 1) MOM などに代表されるメッセジング ミドルウェアは、メッセージのストア & フォワード方式を採用しています。センダーの送ったメッセージはレシーバーが受け取り終わるまでサーバー上に格納されているため、センダー側はレシーバー側のメッセージ受け取り状況に関係なく次の処理 (次のメッセージ送信) に移行できます。イベントドリブン方式の非同期のインテグレーションの実装に、このメカニズムは非常によくマッチします。実際、イベントドリブンをサポートしている ESB 製品の多くは、このメカニズムをその基盤に採用しています。Fiorano ESB も、このメッセージング メカニズムを拡張して採用しています。



1.3.2 ビジネス プロセス (複数ステップからなるインテグレーション パターン)

ビジネス プロセスとはステップの連なりによって構成されるもので、個々のステップはシステム (アプリケーション プログラムやソフトウェア システム) もしくは人間によって実行されます。このようなビジネス プロセスの例として、注文書の処理プロセス、保険請求の処理プロセス、財務処理プロセスなど、多くの業務プロセスがその対象として考えられます。このような業務プロセスには、取引先企業のアプリケーションによって実施される業務や地理的に離れた拠点で処理される業務も、ビジネス プロセスを構成するステップとして含ませることができます。

ビジネス プロセスにも、データ統合の 1対 n のインテグレーション パターンと同様に、イベントドリブン方式とリクエスト－リプライ方式が考えられます。

イベントドリブン方式 (非同期)

イベントドリブン方式では、各アプリケーションはイベント通知を介して連携します。

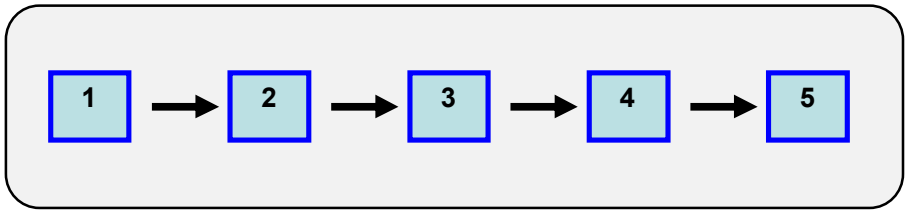


図 2 – イベントドリブン方式

イベントドリブン方式では、例えば 1 のアプリケーションが 2 のアプリケーションにイベントの通知を行うと、2 の処理状態に関わらず 1 のアプリーケーションは次の処理に移行します。この非同期性によって、各アプリケーションはそれぞれ独立して並行的に処理を進めることができ、マルチスレッド処理とすることが可能となります。

各アプリケーションは独立して稼動できるため、人間による処理などの相当に長い時間のかかるものをステップとして組み込むことができます。このように、非同期のビジネス プロセスは、1) 各ステップごとの処理時間に大きな差がある場合、2) 並行処理によって効率をあげることが可能な場合、3) イベントの発生時期が特定できない場合、4) ビジネス プロセスの全体または個々のステップの処理完了までの時間が長く、完了を待機していることが現実的でない場合、などに有効です。また、パブリッシュ – サブスクライブによる複数の相手先にデータを配布することが可能なため、複数の製造ラインを複数台の制御マシンで管理しているようなケースで製造ラインへの指令を一斉に通知するなどのステップをビジネス プロセスに組み込めます。なお、非同期式のビジネス プロセスでトランザクション制御する場合には、ロングライフトランザクションで必要となる補償トランザクションの機能を作成する必要がでてきます。

リクエスト – リプライ方式 (同期)

リクエスト リプライ方式では、各アプリケーションへのリクエスト送信をコントロールするソフトウェア プログラムがまず必要となります。図中で A で示しているものがそれにあたります。つまり、この A ソフトウェア (SOA の世界では、サービス コンシューマーと呼んだりします) が、ビジネス プロセスのフローをコントロールします。

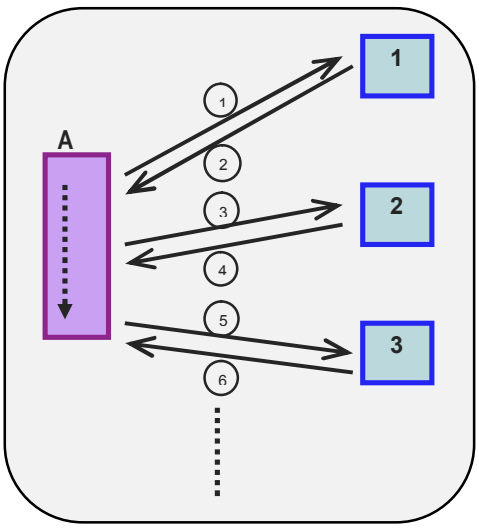


図 3 – リクエスト – リプライ方式

A は、ビジネス プロセスとして定義された順にアプリケーション (1、2、3、....) にリクエストを送っていきます。次のアプリケーションにクエストを送るのは、現在リクエストを処理中のアプリケーションからリプライが返ってきた後になります。リプライが返ってくるまでは、待機状態となります。つまり、分散プラミングで利用している RPC (リモート プロシージャ コール) や RMI (リモート メソッド インボケーション) と非常によく似た形態となります。

各アプリケーションは分散させて配置することが可能ですが、ビジネス プロセスのフローをコントロールしている A は中央集中的な配置となり、後述する “ハブ & スポーク“ の形態となります。また、イベント ドリブン方式とは異なり、マルチスレッド化による並列処理を行えません。

同期式のビジネス プロセスは、プロセス全体の自動化 (ソフトウェア プログラのみによる自動実行) が可能な場合に向いており、人間の処理が介在する場合には難しいものとなってしまいます。また、プロセス全体が完了するまでの時間が長い場合にも、多くのソフトウェア資源が待機させられるため、不向きなものとなります。同期式のビジネス プロセス内の個々のアプリケーション呼び出しは、1対1 のコミュニケーションであり、同一データを同時に多数のアプリケーションに配布するような 1対多のコミュニケーションはできません。ACID トランザクション制御を実施する場合には、同期式が好ましいものとなります。ロング ライフ トランザクションの場合には、保証トランザクションの機能を盛り込む必要がありますが、同期式はもともと長期間のトランザクションには不向きなものです。

1.3.3 コンポジット アプリケーション

コンポジット アプリケーションとは、複数のコンポーネントが組み合わさったソフトウェアを指します。各コンポーネントは、通常、それぞれ異なった実行環境 (ハードウェアや OS など) で稼動しています。コンポジット アプリケーションの例として複数のデータベース (もしくはデータベースを保持している業務アプリケーション) をインボークするサーブレットが挙げられます。この例では、次の順で処理が進みます。

1. まず、サーブレットが呼び出し元からのリクエストを受け取ります。
2. 次に、サーブレットはリクエストに応じてデータベースを保持している複数の業務アプリケーションにデータを送信します。
3. すべての業務アプリケーションからの実行結果が返ってくるまで、サーブレットは待機します。
4. すべての実行結果が揃った時点で呼び出し元に結果をリターンします。トランザクション管理 (コミット処理またはロールバック処理) が必要な場合には、この時点で実行され、呼び出し元のクラインアントで実行する必要はありません。

呼び出し元のクライアントからみると、1つのサーブレットをリクエスト コールするだけで、複数の業務アプリケーションでの実行結果を得ることができます。各業務アプリケーションが異なる実行環境や呼び出しインタフェースによるものであっても、クライアントは複数の業務アプリケーションを一つのアプリケーションとして見做して呼び出すことができます。この例では、各業務アプリケーションとサーブレットが組み合わさって一つのコンポジット アプリケーションを構成し、サーブレットがコンポジット アプリ

ケーションの呼び出しインタフェースとしての機能も果たします。

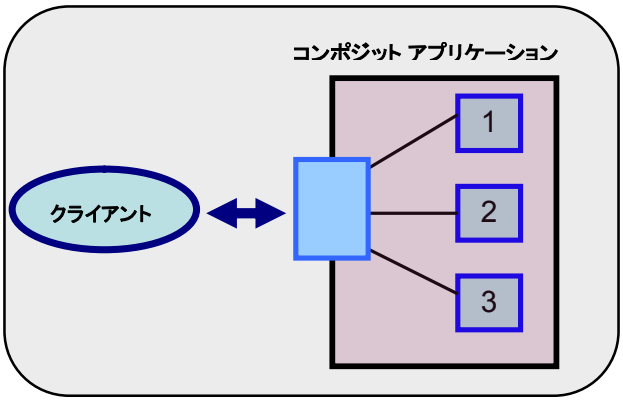


図 4 – コンポジット アプリケーション

コンポジット アプリケーションの呼び出し方法には、様々なコミュニケーション ミドルウェア技術が使用できます。代表的なものには、COM (Component Object Model)、CORBA (Common Object Request Broker Architecture)、RMI 、MOM などがあります。また、ミドルウェアではなく TCP/IP ソケット や SOAP などのプロトコルをそのまま実装することもできます。

コンポジット アプリケーションで最も重要な点は、ビジネス プロセスにおいて一つのステップとして利用できる点にあります。これは、オブジェクト指向プログラミングの技術分野で広く知られている**カプセル化**と**モジュラー プログラミング**の利点をアプリケーション インテグレーションに応用したものです。これらの利点には、以下に挙げるようなものがあります。

- 開発の柔軟性
- 再利用の促進
- ビジネスプロセスの変更の容易性
- コンフィグレーション管理とバージョン管理

1.4 この章のまとめ (SOA に対する共通認識と問題点)

『SOA とは、

1. ビジネス上の業務や機能をサービスという単位でまとめ、
4. 個々のサービスはそのインタフェースを他から利用できる形で定義し、
5. サービスを組み合わせることで、アプリケーションを構築

することである』、ということに異論をとなえるベンダーはないものと思います。この定義を、前節までの説明で用いてきた用語で置き換えてみると、『SOA とは、

ビジネス上の業務や機能 (**既存アプリケーション** や**新規のアプリケーション**、その一部の機能、場合によっては**人間**

による処理) をサービスという単位でまとめ、
個々のサービスが、**同期式 (リクエスト－リプライ)** または**非同期式 (イベントドリブン)**、もしくはその混在したかた形で連携できるようにそのインタフェースを定め、
サービスを連鎖させることで、**ビジネス プロセス**を構築

することである』、となります。(データ統合などの 1 対 n のインテグレーションはビジネス プロセスの一形態であり、コンポジットアプリケーションはサービスという単位にまとめる際の一つの方法である、とみることができます)。

ここで示した SOA の定義を理解しやくするために、「受注処理」のビジネス プロセスを簡単な例として示します。

受注処理のシナリオ

Web から註文書を受付け、データベースに注文書を保存します。注文数に見合う在庫があるか確認します。次に、クレジット会社によるクレジット カードの審査が必要なため、クレジット会社の審査アプリケーションを呼び出して審査を依頼します。審査後、受注の確認書が送付されます。

SOA に基づくアプリケーション (ビジネス プロセス) の構築

1. 「注文書の受け取り」、「在庫照会」、「クレジットの審査」、「受注確認の送付」を、サービスとします。これらのサービスは、実際には、それぞれ別個のアプリケーション プログラムによって実行されます。特に、「クレジットの審査」というサービスは、クレジット会社にあるアプリケーション プログラムが実行します。
2. これらのサービスを組み合わせて、「受注処理」という一連の流れを処理するビジネス プロセスを図 5 のように構築します。(イベント ドリブン式に図式していますが、同期式のリクエスト リプライ方式でも大きな違いはありません。)
3. サービスは、受注処理のアプリケーションだけではなく、「倉庫管理アプリケーション」などの他のアプリケーションで再利用することができます。

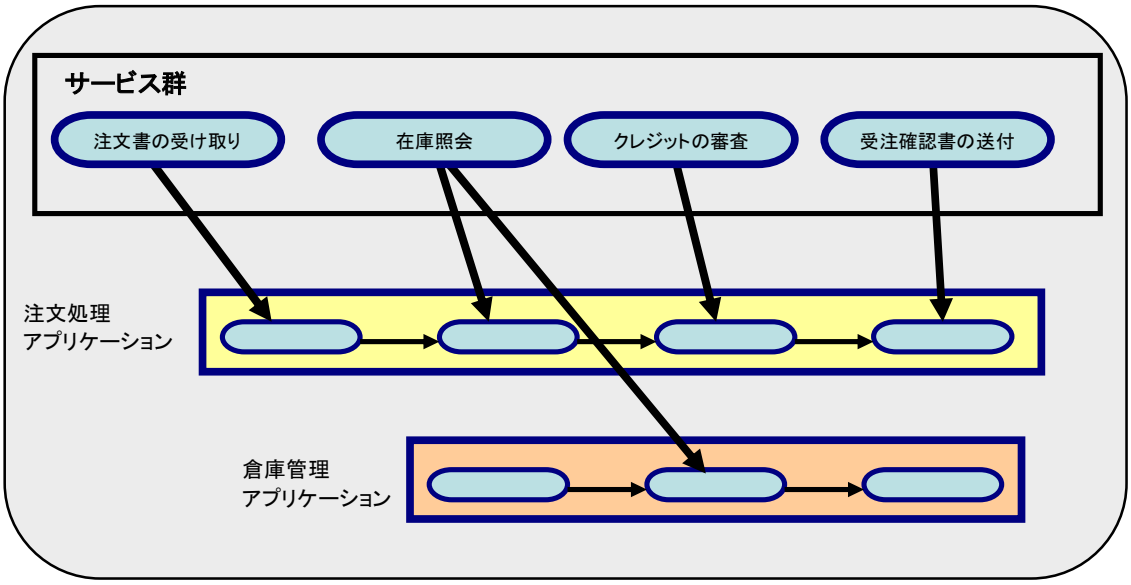


図 5 – SOA アプリケーション (ビジネス プロセス) におけるサービス

SOA の現状の問題点

SOA の現状の問題は、「サービスという単位にまとめる方法」や「サービスの連鎖方式とそのインタフェースの定義方法」、「SOA アプリケーションの稼動環境 (プラットフォーム)」など、具体的な実装方法や方式に対する考え方がベンダー毎に異なっている点です。現状のままですと、A 社の製品上に構築した SOA アプリケーションが、B 社の SOA プラットフォームでは動かすことができないとか、A 社の製品に基づいてサービス化したものが、B 社の SOA アプリケーションで利用できないなどの事態に陥ってしまう可能性があります (注 2)。

以降の章では、これらの問題点と解決方法について議論し、Fiorano Software の現時点での考え方を提示いたします。次の第2章では、SOA アプリケーション (ビジネス プロセス) のプラットフォームである ESB (エンタープライズ サービス バス) をとりあげます。

(注 2) ここにきて、Sun Microsystems が中心になって進めている JBI (Java Business Integration) の標準規格、IBM など数社のベンダーが協同して進めている SCA (Service Component Architecture) など、この問題を解決するための動きがいくつかでてきています。

2 SOA プラットフォームとしての ESB

業界 (ソフトウェア ベンダーやシステム インテグレータなどの間) では、**『インテグレーションは SOA (サービス指向アーキテクチャ) に基づいて実施し、そのベースとなるインフラストラクチャに ESB (エンタープライズ サービス バス) を採用するのが効果的な方法である』**という意見で一致したかのように、各ベンダーから ESB をうたった製品がリリースされています。

「ESB」とはガートナー (Gartner) 社が最初に定義した言葉ですが、機能要件などその内容については多くのベンダーから様々な解釈が生まれています。Fiorano Software では、ESB がサポートすべき基本機能について次のように定義し、それを具現化した製品をリリースしています。

- 広範囲な分散環境の全体にまたがるバス
- 標準規格に準拠したコネクティビティ (接続性)
- データ サービス (XML のサポート、データ変換、データの付加など)
- ビジネス プロセスのサポート機能 (コンテンツ ベース ルーティング、プロセス フロー制御)

Fiorano Software のこの定義は、SOA に基づいて構築されたアプリケーションであるビジネス プロセスとその構成要素である個々のサービスを、効果的にサポートするためのものです。ここでサポートとは、ビジネス プロセスのモデリングおよびその構築から、構築されたビジネス プロセスのデプロイメント、運用、管理まで、ライフサイクル全体をカバーする広範囲なものです。そのため、上記 4つの基本機能に加えて、セキュリティ管理、フォールトレーランス機能、監視ツールなどをはじめとする多くの付加機能を備えたものとなっています。

この章では、上記の基本機能について重点的にとりあげ、ビジネス プロセス (SOA アプリケーション) のプラットフォームとしての ESB が備えるべき機能について説明します。

2.1 ハブ & スポークとバス

ESB の最も重要な特徴は、そのバス形式にあります。まず始めに、インテグレーションの進化をそのトポロジーの面でみてみましょう。インテグレーション用のミドルウェアが整備される前の時代には、アプリケーション同士を結ぶ際には、利用可能な技術をその都度選択していました。その結果、図 6 に示すような “スパゲティ” 状態をまねくこととなってしまいました。

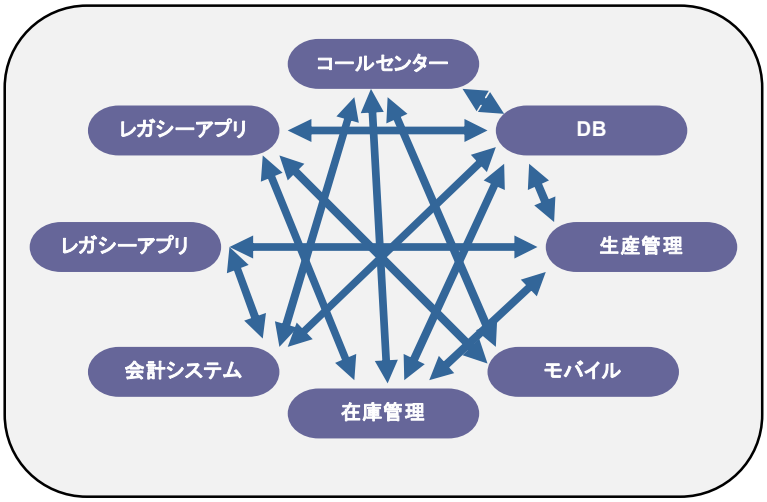


図 6 – アドホックなインテグレーション

その後、旧 EAI 製品に代表されるミドルウェアの登場によって、この “スパゲティ” 状態は改善されました。このミドルウェアは、図 7 に示すように中央のハブに複数のスポークが集中する「ハブ & スポーク」のトポロジーを採用していました。こうすることで、アプリケーション間の通信経路の数は飛躍的に減少させることができました。しかしながら、すべての通信 (メッセージやデータ) が中央のハブに集中することになり、ハブが全体の効率を左右するボトルネックとなってしまいました。最悪の場合、ハブの障害によってすべてのアプリケーション間通信が停止してしまいます。この一極集中による様々な問題に対処するためには、多大な運用管理コストを必要とします。

ハブ＆スポークによるミドルウェアには、旧 EAI 製品、MOM 製品、アプリケーション サーバー (以降、AP サーバーと記します) 製品など多くのものがあります。

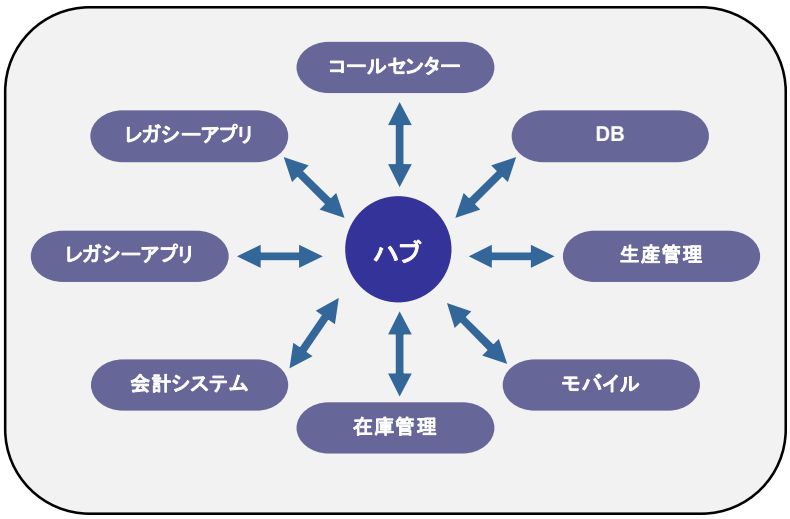


図 7 – ハブ & スポークによるインテグレーション

ハブ＆スポークにおける一極集中の問題は、バス形式を採用することで解消できます。バス形式では、図 8 で示すように、データやコントロールの流れが一箇所に集中するポイントがありません。メッセージやデータは、地理的にも、また、組織 (企業、企業内の部門、取引業者など) 的にも分散されて稼動しているアプリケーションすべてにまたがって “敷設” されている経路上を流れていきます。この場合の “敷設” とはもちろん比ゆ的なものであり、ハードウェア バスとは異なり目に見える形で一本のバスが存在するわけではありません。ESB におけるバスはあくまで論理的なものであり、その具体的な実現方法は各ベンダーによって異なっています。重要な点は、1 本のバスが関係する拠点のすべてに (論理的ではありますが) またがっていることです。バスが 1 つのハードウェア上あるいは 1 つのサーバー ソフトウェア上でしか稼動できない局所的なものであると、それは “ハブ & スポーク” のハブと同じこととなり、依然として一極集中の問題を解決できないままとなってしまいます。

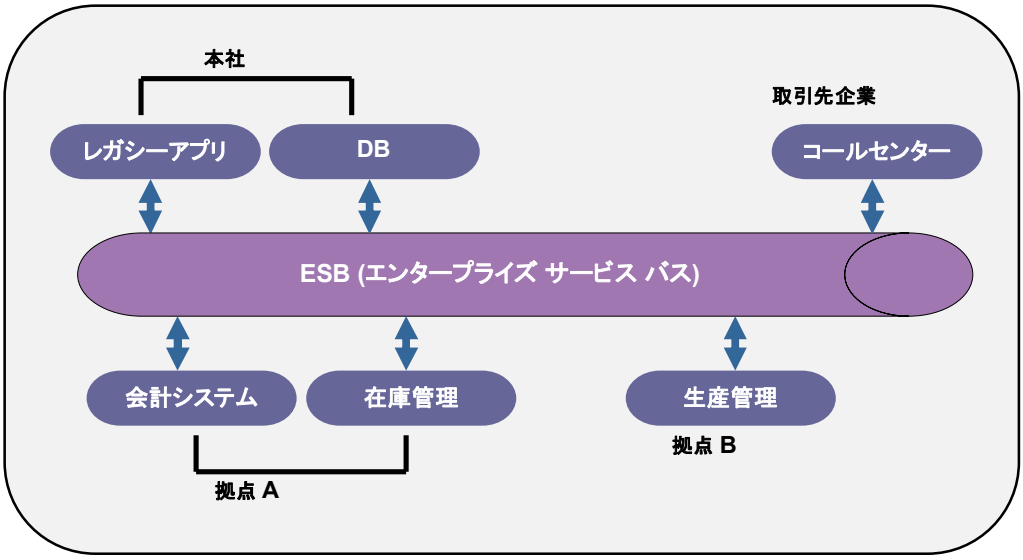


図 8 – バスによるインテグレーション

ESB 製品によるトポロジーの違い

現在市場に出ている ESB 製品は、その実装方法から次のように大別できます。

- 旧 EAI 製品の上に実装したもの
- AP サーバー製品の上に実装したもの
- MOM 製品の上に実装したもの
- 既存製品をベースとしないもの

ここで、前節で述べたように、旧 EAI 製品、AP サーバー製品、MOM 製品といったミドルウェアは、”ハブ & スポーク” に基づくものであることを思い出してください。ESB をこれらの製品の上に実装するということは、ミドルウェアのサーバーと同様に ESB もまたハブとして機能することを意味します。図 9 に示した例では、拠点 A に置かれたサーバーがハブとなっており、各拠点のアプリケーションからのデータがここに集中しています。ESB は、各アプリケーションとの接続をサポートしているにすぎず、バス形式とすることの意味の大半を失っています。このような製品においてデータの一極集中を避けるためには、各拠点にミドルウェア サーバーと ESB をそれぞれ配置し、これらの ESB 間を結ぶなどの処置が必要となってしまいます。

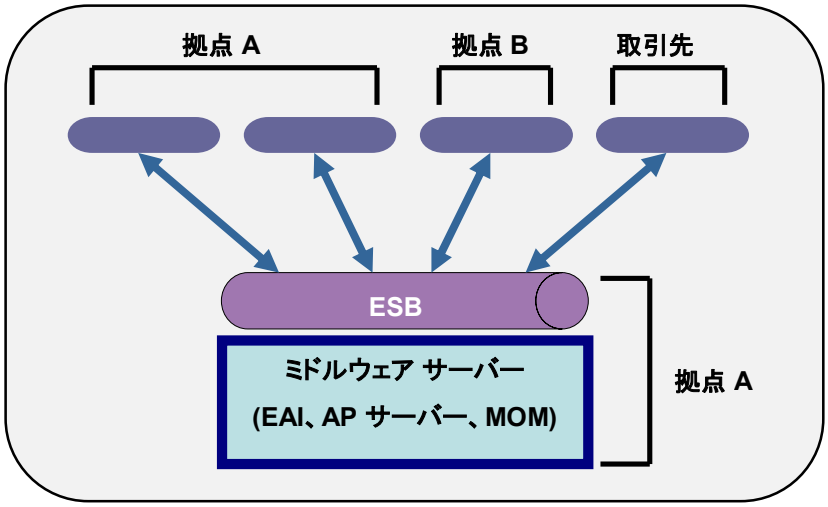


図 9 - 既存ミドルウェア上に実装した ESB

Fiorano Software の ESB 製品 Fiorano ESB では、この問題を解消するために優れた方法を採用し、「分散して点在する各拠点にまたがった 1 本のバス」を実現しています。これによって、一極手中を避け、バス形式の本来のメリットを得ることができます。

2.2 標準規格に準拠したコネクティビティ

ESB の次の役割は、アプリケーションへコネクティビティを提供することです。大切な点は、一つの標準規格に限定しないことです。『SOA のサービスは、Web サービスとして呼び出す』などと限定しているベンダーもありますが、すべてのアプリケーションが Web サービス (SOAP) に対応しているわけではありません。既に稼動している既存のアプリケーションをそのまま活用するためには、ESB が多種のプロトコルをサポートできるようになっていることが不可欠です。

ESB 側で異なる種類のコネクティビティを提供することで、アプリケーション間のプロトコルの違いを ESB で吸収することができます。すなわち、連携するアプリケーション間のプロトコル変換機能として ESB が働くこととになります。アプリケーションは自身と ESB 間のプロトコルのみを制御すればよく、相手側アプリケーションのプロトコルを気にしなくてもよくなります。これにより、相手側アプリケーションの変更や別アプリケーションとの置き換えによる通信プロトコルの影響はなくなります。

ESB がサポートするプロトコルはできるだけ標準規格としたほうが、運用管理コストを抑えられます。次の表に、主なアプリケーションの種類と該当する標準規格をまとめました。Fiorano ESB は、この表にある標準規格をサポートするだけでなく、アプリケーション独自のプロトコルによる接続が必要な場合にはそれを組み込むこともできるようになっています。

アプリケーションの種類	標準規格
パッケージ アプリケーション(ERP、SCM など)	JCA

DBMS	JDBC
汎用機上のアプリケーション	JCA
Web サービス	SOAP / HTTP
.NET アプリケーション	SOAP / HTTP
Web アプリケーション	HTTP
AP サーバー上のアプリケーション	J2EE (EJB など)
MOM に基づくアプリケーション	JMS
EDI に基づくアプリケーション	業界ごとの EDI 標準
メール (人間との連携)	SMTP、POP3 など

2.3 データ サービス

前節で、アプリケーション間のプロトコル変換について述べましたが、ESB では同様にアプリケーション間のデータの違いを吸収できなければなりません。すべてのアプリケーションが XML をサポートしていれば、データ形式の問題はほとんどクリアできるのですが、これは理想的にすぎます。そこで、現実的な解として、ビジネス プロセスを流れていくデータ形式の基本を XML とし、XML をサポートしていないアプリケーションとデータの授受を行う際には XML と他のデータ形式との間の変換を行うようにします。また、XML をサポートしているアプリケーション間であっても、その構造や意味が異なっている場合があります。例えば、日付形式の違い (西暦と和暦など)、数値の単位の違い (グラムとキログラムなど)、ID 構造の違い (商品番号の違いなど) など多くのケースが考えられます。このような場合にも、XML 間のデータ変換が必要になります。さらに、ビジネス プロセスの次のステップにデータを渡す際に、新たなデータを付加したり、2つのアプリケーションから得たデータを合成する必要が生じる場合もあります。これらの機能は、すべて ESB 上で実現すべき機能です。

ほとんどの ESB 製品は、これらのデータ変換をサポートしており、変換内容を定義するためのツールとしてグラフィカルなデータマッピング ツールを提供しています。

2.4 プロセス フロー制御

ESB 基本機能の最後は、ビジネス プロセスのプロセス フロー制御です。ガートナー社が ESB を提唱した際にも、ESB にプロセス フローの制御機能を持たせることの用不要を課題としていました。一部のベンダーでは、『サービスは Web サービスとして実装し、Web サービス間のフロー制御は BPEL によって定義し、実行する』ものとし、ESB がフロー制御の機能を持つ必要はないとしています。Web サービスも BPEL も、リクエスト – リプライによる同期式のインテグレーションに対応したものです。BPEL によるフロー制御を図示すると、図 9 のようになります。これは、『図 3 リクエスト – リプライ方式』と非常によく似ており、中央集中の “ハブ & スポーク” の形態となっていることが理解できます。たしかに、この BPEL によるフロー制御では、ESB にフロー制御機能を持たせる必要はありません。むしろ、BPEL の中央集中的なフロー制御には、バス形式はあまりなじまないといったほうが

よいかもしれません。仮に、BPEL フローを制御しているソフト (下図で BPEL 実行ソフトと記したもの) が SOAP プロトコルをハンドリングできるものであれば、コネクティビティの提供といった ESB の他の機能も必要なくなり、ESB 自体が不要なものとなります。

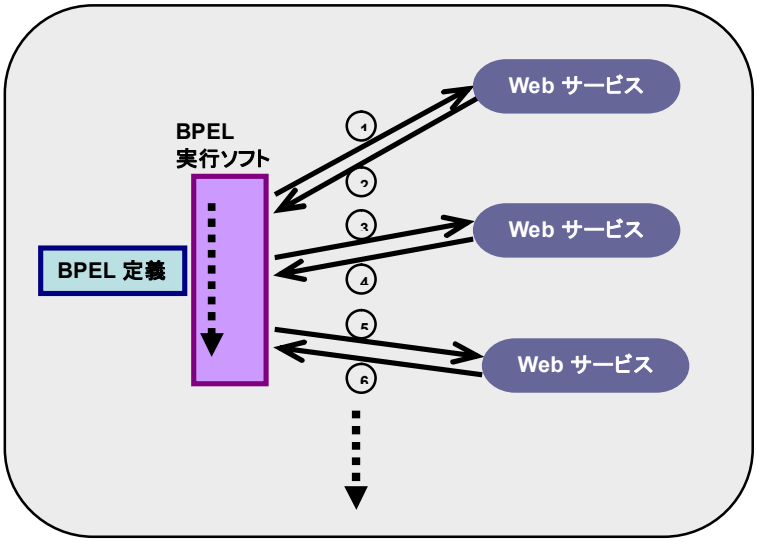


図 10 – Web サービスと BPEL によるフロー制御

Fiorano Software ではこれに反し、ESB というバス形式の最大の特徴である “データ経路” という性質は、データが流れる経路としてだけでなく、フロー コントロールの経路としても非常に有用であると考えています。特にイベント ドリブン方式 (非同期) のフロー制御との相性がよく、Fiorano ESB ではイベント ドリブン方式を最初のバージョンから採用し、以降これをベースに様々な改良を加えてきています。

図 11 は、ESB 上のイベント ドリブン方式のフロー制御の基本的な考え方を示しています。ESB 上に配置された四角は、ビジネス プロセスの各ステップをあらわしています。イベント通知 (およびデータ) が各ステップに順に渡されていきますが、個々のステップは非同期に稼動しています。また、このイベントの流れは、中央の 1 箇所に集中するものではなく、広域に広がった ESB 経路上を流れていきます。外部のサービス (ERP、汎用機、DBMS などの既存アプリケーション) による処理が必要な場合には、JCA や JDBC といった標準プロトコルによって通信します。図中に赤丸で示した箇所は、コンテンツ ベース ルーティング機能によってフローが分岐することを示しています。このコンテンツ ベース ルーティングは、データ (XML 形式) の内容によってフローを分岐させるもので、XPath の技術を応用しています。例えば、金額が 1,000 万以上の注文書には課長の承認が必要、といったようなデータ内容によって処理プロセスが異なる場合に有用なものとなります。

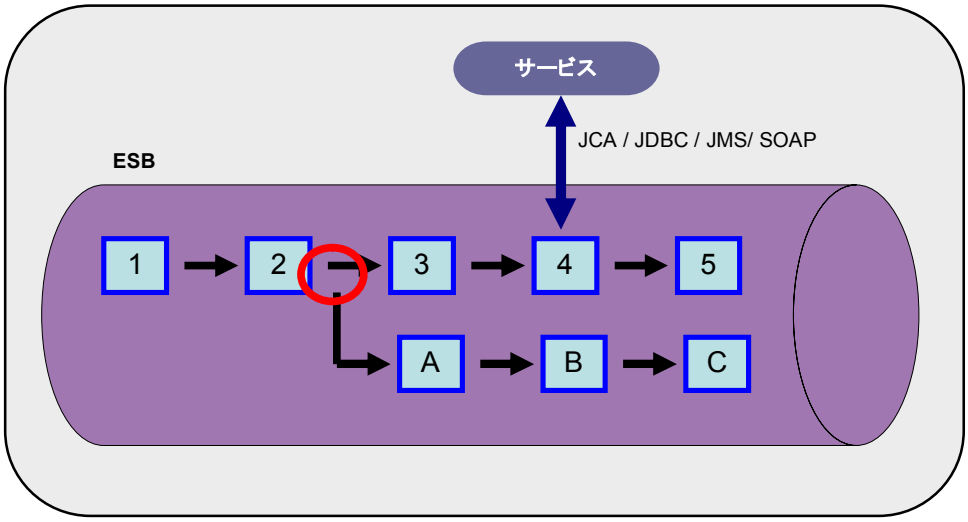


図 11 – ESB 上のフロー制御 (イベントドリブン方式)

ESB 上のフロー制御はイベントドリブン方式 (非同期) が向いておりますが、実際の業務処理をすべて非同期で実施することが最善の方法というわけではありません。また、逆に BPEL のみに基づいてすべての業務処理を同期式に実施することが最善の方法でもありません。より良い SOA アプリケーションとするためには、非同期と同期を組み合わせたビジネス プロセスの構築とそれを稼動させることができるプラットフォームが必要となってきます。(同期式と非同期式の違いについては、『1.3.2 ビジネス プロセス』を参照してください。)

Fiorano Software では、非同期と同期が混在したビジネス プロセスをサポートできる製品をリリースしています。この実現方法について、次章で説明します。

3. SOA 実現のためのフレームワーク（SOAIF）

Fiorano Software では、最適な SOA アプリケーションの構築とその運用管理のためのプラットフォームを製品としてリリースしています。この製品の実装は、SOA 実装フレームワーク (SOAIF) として定義されたデザインに基づいています。このフレームワークの特徴のひとつに、非同期プロセス (イベント ドリブン方式) と同期プロセス (リクエスト リプライ方式) が混在したビジネス プロセスの実現があります。これを実現するためのベースとなったコンセプトが「ビジネス コンポーネント アーキテクチャ (BCA)」です。この章では、最初にビジネス コンポーネント アーキテクチャについて説明し、その後にビジネス プロセスの構築方法、SOA プラットフォームのアーキテクチャについて説明します。

3.1 ビジネス コンポーネント アーキテクチャ (BCA)

ビジネス コンポーネント アーキテクチャ (BCA) もガートナー社が最初に提唱したコンセプトです。Fiorano Software は、このコンセプトを具現化した製品を世界で最初にリリースしました。

BCA のコンセプト

ビジネス コンポーネント アーキテクチャ (BCA: Business Component Architecture) とは、ビジネス コンポーネントをモジュールとしてビジネス アプリケーションの設計、構築を行うモジュラー アーキテクチャのことを指しています。ここで、ビジネス コンポーネントとは、ソフトウェア プログラムもしくはプログラムとデータを組み合わせたものであり、ビジネス上の業務や機能を実装したものです。

上記のビジネス コンポーネントの定義は、SOA におけるサービスの定義とたいへんよく似ています。むしろ、BCA が SOA におけるサービの実装方法の指針を示しているといったほうがよいでしょう。すなわち、サービスの実装方法はモジュラー アーキテクチャの利点を活かしたものとすべきでという考え方です。よく聞かれる “SOA のサービスとは、Web サービスの標準にしたがったものである” という考え方とは、そのベースとなっている視点がまったく異なるものです。

Fiorano Software では、SOA プラットフォームの製品化にあたり、BCA の定義を次のようにしています。

製品化にあたっての Fiorano の定義

1. ESB 上に配置できるものをビジネス コンポーネントとし、ビジネス プロセスの各ステップを実行する構成要素とする。
2. ESB 上に直接配置できないもの (例えば、ERP などのパッケージ アプリケーション、汎用機上のアプリケーション、ユーザー開発の既存アプリケーション、データベースを管理している DBMS など) は、ESB 上のコンポーネントからアクセスするものとする。
3. 各ビジネス コンポーネントは、コンポーネント内のビジネス ロジックとコンポーネントへのアクセス インタフェースを区別し、カプセル化の利点を得られるようにする。カプセル化の利点は、アクセス インタフェースを変更しない限り、コンポーネントの内部実装の変更が連携している他のコンポーネントやビジネス プロセスの全体に影響を及ぼさない点にあります。これによって、ビジネス環境の変化に応じて、ビジネス プロセス内のコンポーネントを別のコンポ

ーネントと置き換えることが容易となり、アジャイルなビジネス プロセスを実現できるようになります。

4. アクセス インタフェースは明確に定義された (Well-defined) ものであり、コンポーネント間の連携は定義されたインタフェースに従ってイベント ドリブン方式 (非同期) および リクエスト – リプライ方式 (同期) の両方 (場合によっては片方だけ) をサポートできるものとする。
5. 1つのビジネス プロセス内でイベント ドリブン方式とリクエスト – リプライ方式のプロセス フローを混在して使用することを可能とする。
6. モジュラーの利点を活かすよう、作成したビジネス プロセス (イベント ドリブンおよびリクエスト リプライ) をより上位のビジネス プロセス内で 1 つのコンポーネントとして再利用できるものとする (コンポジット コンポーネント)。

理解しやくするために、まずイベント ドリブン方式のビジネス プロセスを説明します。イベント ドリブン方式におけるビジネス コンポーネントを図示すると、図 14 のようになります。

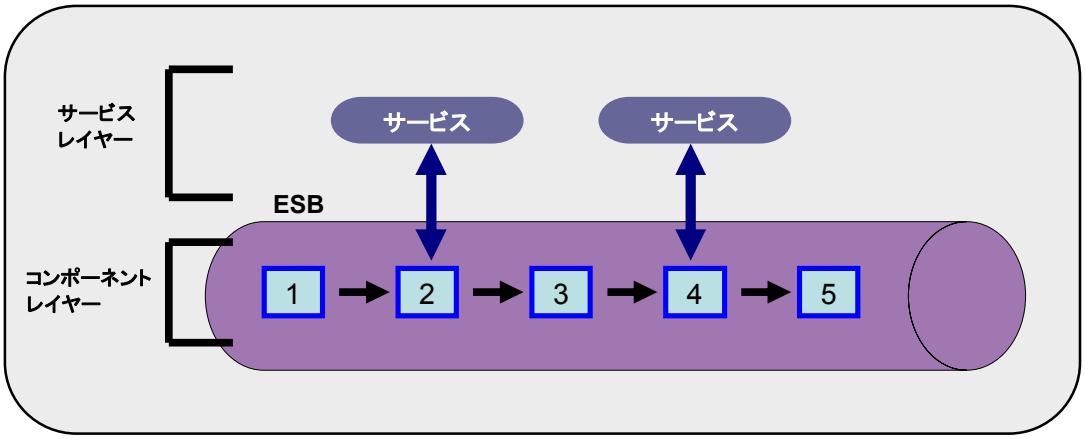


図 12 – ビジネス コンポーネントと外部アプリケーション

図中、四角で表されているものが個々のビジネス コンポーネントです。ビジネス コンポーネントを連鎖させることによって、ビジネス プロセスを構成します。この連鎖は、メッセージングによるイベント通知で実現しています (同期式のビジネス プロセスは、BPEL によって定義します。これについては、次節を参照してください。)。また図では、コンポーネントから呼び出される外部のアプリケーションをサービスと記し、サービス レイヤーという名前を与えていますが、このレイヤーのものだけが SOA におけるサービスに該当するわけではありません。ビジネス コンポーネントもビジネス ロジックを実装したものであり、ビジネス プロセスのステップを構成するものです。違いは、ESB 上に直接配置できるか否かによっています。ただ、粒度という観点からみると、ESB 上に配置されるビジネス コンポーネントの方が、外部のサービスよりも粒度が細かくなる傾向はあります。

ビジネス コンポーネントのタイプと形式

ビジネス コンポーネントには、次の図に示すように、外部のアプリケーションを呼び出すためのアダプター ポートを持ったものと、そうでないものとがあります。また、どちらのコンポーネントもインプット (In)、アウトプット (Out)、エラー用 (Err) のポートを持っています。アウトプット ポートは、複数持つことが可能です。

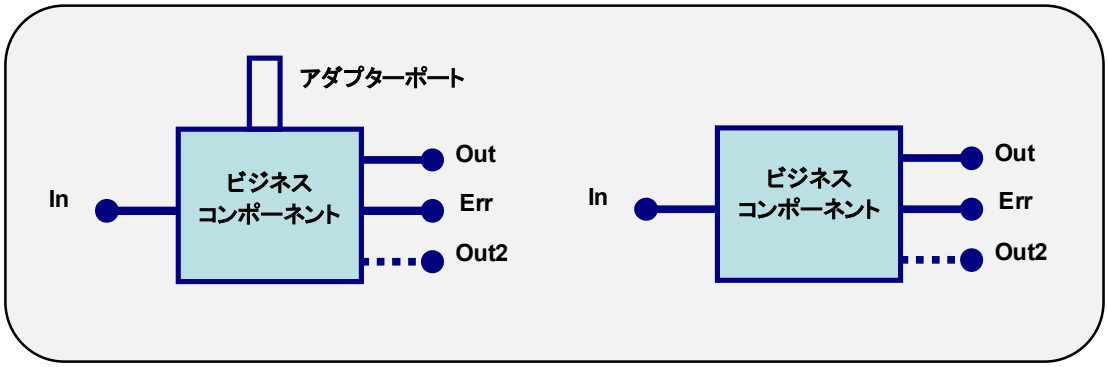


図 13 ビジネス コンポーネントのポート インタフェース

ビジネス コンポーネント間の連鎖は、ビジネス コンポーネントのアウトプット ポートからの出力 (イベントおよびデータ) を次のビジネス コンポーネンのインプット ポートへと結びつけることで定義していきます。アウトプット ポートが複数あるため、出力されるイベントやデータによって次につながるビジネス コンポーネントを異なるものとすることが可能となっています。また、エラー用のポートによってエラー発生時の処理を通常のプロセスとは異なる別のフロー経路で処理することを可能とします。次のセクションで説明する Fiorano Software が提供しているプリビルト ビジネス コンポーネントは、できるだけ多くのビジネス プロセスで利用できるよう、汎用的なものとして作成されています。このため、インタフェースとしての入出力データを定義していません。ビジネス プロセスを設計、構築していく段階で、ビジネス コンポーネント間のデータ定義を行って使用することになります。具体的には、ビジュアルなプロセス構築ツールの上で、各ポートで入出力されるデータ構造の定義、マッピング ツールによるアウトプット ポートと次のコンポーネントのインプット ポートの間のデータ マッピングの定義を行います。ただし、Fiorano SAO プラットフォームでは、プリビルト ビジネス コンポーネントの多くはデータ構造の定義が非常に簡単に行えるよう、ツールに様々な機能を持たせています。例えば、DB アダプタでは、対象となるデータベースから自動的にテーブルの一覧と各テーブルのスキーマをツールが自動的に取得し、ビジネス プロセスの構築者に提示します。構築者は、表示されたテーブル、データ項目などをマウスでクリックするだけで DB アダプターが扱うデータ構造を定義することができます。

プリビルト ビジネス コンポーネント

Fiorano Software のプラットフォームには、80 種ほどのプリビルトされたビジネス コンポーネントが組み込まれて出荷されます。Fiorano Software がこれまでに手がけてきたインテグレーション プロジェクトの経験値から、これらのプリビルト コンポーネントでビジネス プロセスで必要とされる主だった機能の 75% 以上をカバーできることがわかっています。Fiorano の SOA 製品のお客様は、このプリビルト コンポーネントの利用によってプログラミングのコストを飛躍的に削減することに成功しています。

もちろん、独自のビジネス コンポーネントをユーザーが作成することもできますし、第3者のソフトウェア ベンダーやシステム インテグレータから提供することもできます。製品にはこのためのコンポーネント作成ウィザードが付属しています。

現行バージョン に組み込まれているビジネス コンポーネントには、次のようなものがあります。

外部アプリケーションとのインタフェースを持つもの

このタイプのコンポーネントは、JCA、JDBC、SOAP などの標準プロトコルに基づいて外部アプリケーションにアクセスします。

- SAP R/3 アダプター
- DBMS（Oracle、DB2、SQL Server など JDBC ドライバーを持つ DBMS）アダプター
- Web サービス コンシューマー (Web サービスの呼び出し)
- 他社の MOM 製品サーバーとの間のメッセージ送受信
- Web アプリとの間の HTTP による送受信 (GET / POST)
- メール サーバーとの間のブリッジ (SMTP、POP3)
- FTP サーバーとの間のブリッジ (get / put)

外部アプリケーションのインタフェースを持たないもの

外部アプリケーションによってビジネス ロジックを実行する必要がない場合があります。特にデータ暗号化や圧縮などの処理は、わざわざ外部アプリケーションにその処理をまかせる必要はありません。Fiorano Software では、データ サービスといったユーティリティ的な機能をこのタイプのプリビルト コンポーネントとして提供しています。

- データ変換 (XML 間、XML とプレーン テキストの間、XML と EDI フォーマットの間)
- データの圧縮、データの暗号化
- ファイルの Read / Write
- スクリプト言語用コンポーネント (ユーザー独自のロジックを Perl、B-Shell、JavaScript などのスクリプト言語を用いて実装できます)

3.2 BPEL フローにおけるビジネス コンポーネントの利用

Fiorano の SOA プラットフォームでは、同期式のビジネス プロセスは BPEL によって定義、実行することとしています。このために、構築用ツールとして BPEL エディタを提供し、実行用に BPEL エンジンを用意しています。この BPEL エンジンは、ESB の上で稼動します。BPEL フローのリクエスト – リプライ方式という集中的なフロー制御という性格を反映し、この集中制御の拠点を ESB 上の任意の拠点から選択できるようにしています 。複数の BPEL フローをそれぞれ異なる拠点で制御するような場合にも対処できるよう、拠点毎に別個の BPEL エンジンを配置できるようにもしています (この場合、1本の ESB 上に複数の BPEL エンジンが配置されることになります)。

Fiorano BPEL の特筆すべき利点のひとつとして、BPEL フローのアクティビティ (activity) としてビジネス コンポーネントを指定できる事があげられます。他社の BPEL 製品では、Web サービスをインボークすることしかできないものが多いのですが、Fiorano BPEL では JDBC による DB へのアクセス、ERP などのパッケージ 製品などの呼び出し (JCA)、ローカル ファイルの Read / Write、

データの暗号化など、前述のプリビルト ビジネス コンポーネントを利用することができ、同期式ビジネス プロセス構築の柔軟性や利便性が格段に向上します。

BPEL フローにおけるビジネス コンポーネントの使用例を、図 14 に示します。この例では、次の 3 つのコンポーネントを使用しています。

- "Invoke" : これは、プリビルト コンポーネントではなく、BPEL の規格として定義されている Invoke アクティビティを用いている例です。Fiorano BPEL では、Web サービス以外にも、ERP などのパッケージ アプリケーションやメインフレーム上のアプリケーションをインボークできます。
- "DBProc" : JDBC によるデータベースへのアクセスをおこなうビジネス コンポーネント
- "SMTP" : 電子メールの送信をおこなうビジネス コンポーネント

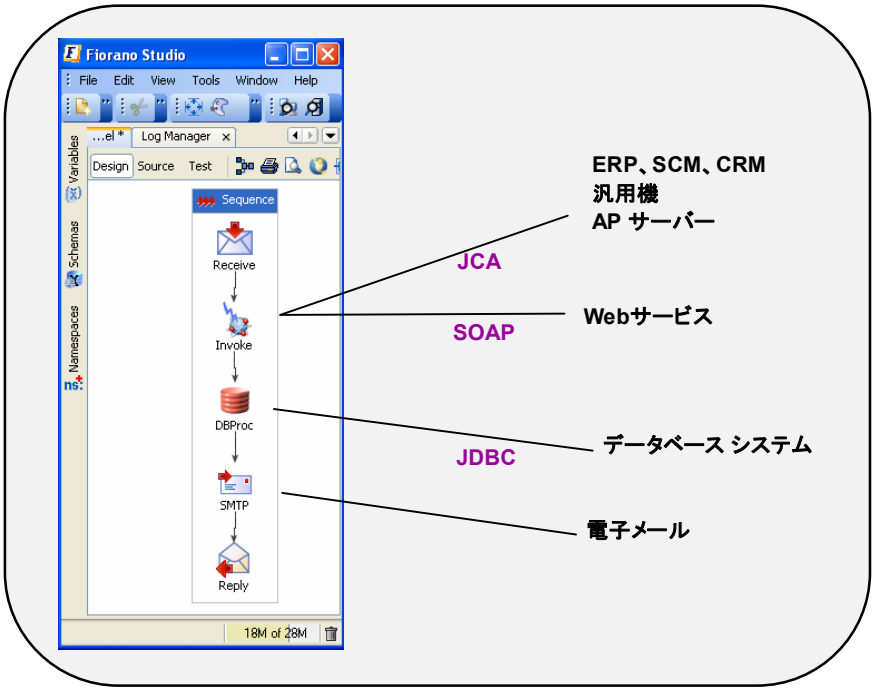


図 14 BPEL とビジネス コンポーネント

Fiorano の SOA プラットフォームでは、BPEL によって構築したフローを単独のビジネス プロセスとして実行させることも可能ですが、次のセクションで説明するようにイベント ドリブン方式のビジネス プロセス内に組み込んで、同期式と非同期式が混在したビジネス プロセスを作成することができます。

3.3 コンポジット コンポーネント

コンポジット コンポーネントは、1章で説明したコンポジット アプリケーションとたいへんよく似たもので、モジュラー アーキテクチャの利点を活かしたものです。コンポジット コンポーネントは、複数のビジネス コンポーネントによって構成されたビジネス プロセスをより上位のビジネス プロセスや他のビジネス プロセスの中で利用するために、一つのコンポーネントとしてまとめたものです。

図 15 にコンポジット コンポーネントの例を示します。この例では、ESB 上のビジネス プロセスの3番目のステップとして置かれたビジネス コンポーネントが、コンポジット コンポーネントとなっています。各ステップの実行順序は、

1 → 2 → A → B → C → 4 → 5

となります。

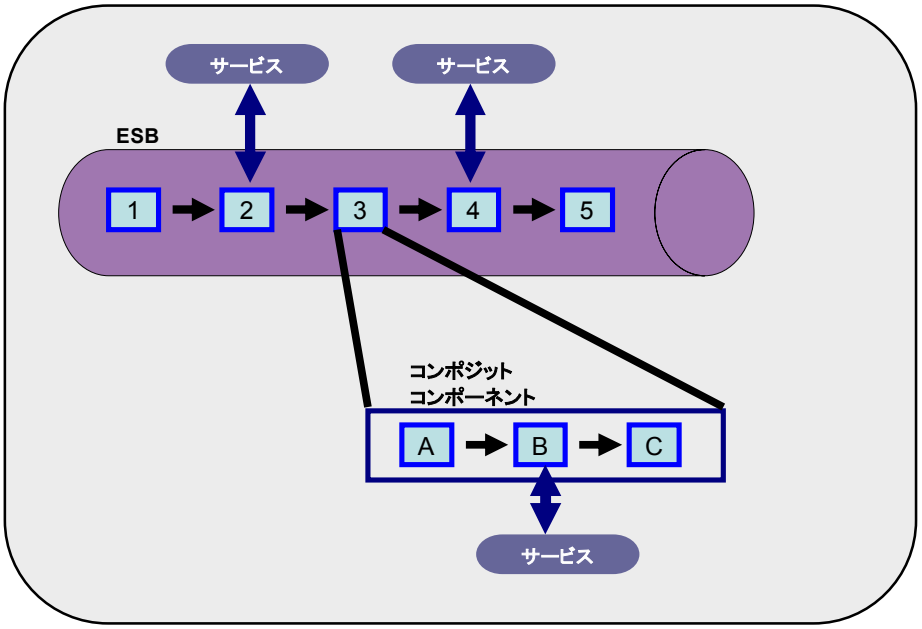


図 15 – コンポジット コンポーネント

図 15 ではイベント ドリブンのコンポジット コンポーネントとしていますが、コンポジット コンポーネントの内部を BPEL のフローとすることも可能です。

3.4 サンプル プロセス

1章で示した『受注処理』のシナリオを、ビジネス コンポーネントを用いて構築してみます。1章のシナリオを少し変更し、次のようにします。

受注処理のシナリオ

Web から註文書を受け取ります。在庫の確認を行った後に、クレジット会社によるクレジット カードの審査を受けます。この審査には、クレジット会社が提供している Web サービスを利用します。審査後、CRM アプリケーションから受注の確認書を送付します。

ビジネス プロセス

ビジネス プロセスは、図 16 のようになります。各ビジネス コンポーネントの説明は、次の通りです。

1. Web サイトから注文書を受け取ります。これには、HTTP による Web アプリとの送受信用コンポーネントを使用します。

2. 注文された個数の商品が在庫されているか否か、確認します。これは、在庫管理アプリケーションを呼び出すのではなく、在庫管理 DB に直接アクセスするものとします。このアクセスに、アダプター形式のコンポーネント、DBMS (JDBC) アダプターを使用します。

3. Web サービス コンシューマ コンポーネントを用いて、クレジット審査会社のアプリケーション (Web サービス) を呼び出します。

4. クレジットカードの審査に合格すると、CRM アプリケーションを呼び出して顧客管理情報を更新し、CRM から確認書を送付します。CRM アプリケーションの呼び出しは JCA を用います。

5. クレジットの審査に不合格の場合、注文が受けられない旨の通知を顧客に通知します。これには、SMTP コンポーネントを利用して電子メールで通知するものとします。
   なお、合否によるルート変更は、ESB のコンテンツ ベース ルーティングの機能を使用します。

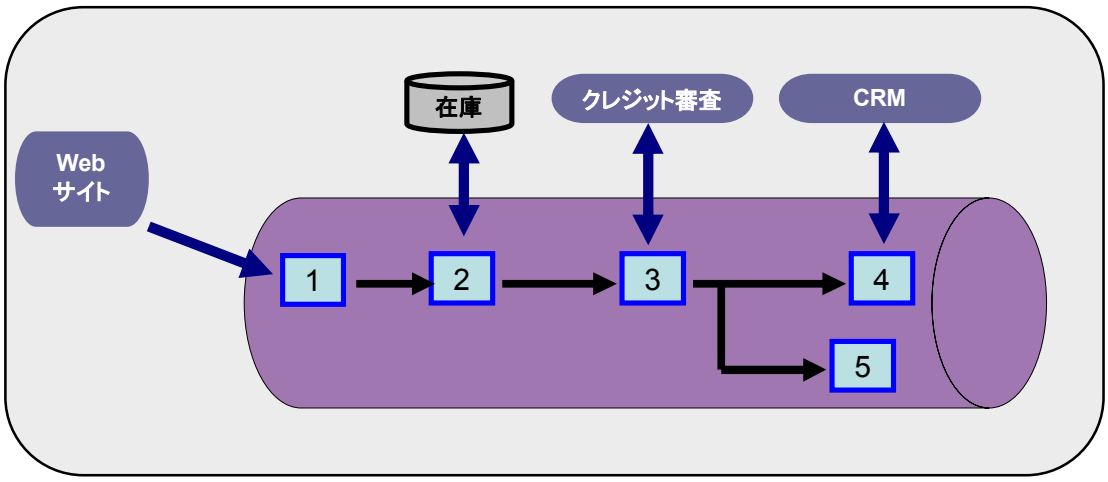


図 16 – 受注処理プロセス (すべてを非同期とした場合)

このビジネス プロセスは、ほとんどのステップにプリビルト ビジネス コンポーネントを利用することができます。新たに作成する必要があるコンポーネントは、CRM 呼び出し用のコンポーネントだけです。

トランザクション管理

さて、ここでトランザクション管理の問題を考えてみましょう。上のビジネス プロセスでは、在庫数の更新について考慮していませんでした。実際には、在庫データベース内に保持されている在庫数を更新する必要があります。更新をどの時点で行うかについては様々な考え方があります。最も安全な方法は、ビジネス プロセスの最後の段階で注文が確定しますので、この時点で在庫数の更新を行うことです。しかし、これではある一人のお客様の注文書の処理が完了していない状態では、別の人からの注文者を受け付けることができなくなります。そこで、2 番目のステップで在庫確認を行った際に同時にデータベース内の在庫数も更新することとします。こうすることで、Web サイトから頻繁に発生する注文書を滞りなく受け付けることができます。しかし、クレジット審査の結果が不合格となった場合には、注文を確定できず、更新してある在庫数も元に戻す処理が必要となります。まさに、トランザクション管理におけるロールバック処理が必要となります。この処理をビジネス プロセスとして実現するために、2番目と3番目のステップを BPEL フローによる同期式とし、トランザクション管理を行うように変更します (図 17 を参照してください)。

Fiorano BPEL は、トランザクション管理をサポートする機能を備えています。まず、BPEL フローを作成する際に、トランザクション管理を行う範囲を指定します。この例では、 2 と 3 の処理をトランザクション管理の範囲とします。次に、3 のクレジット審査の処理結果による判定基準を定義します。つまり、審査に合格したことを示すリターン値の場合にはコミット処理、不合格を示すリターン値の場合にはロールバック処理と定義します。こうしておくことで、プロセスの稼動時には自動的にコミット処理もしくはロールバック処理が実行されるようになります。Fiorano のプリビルト コンポーネントである DB アダプターは、コミットおよびロールバックに DBMS の機能を利用するように作成してあります。このため、コミットやロールバックの処理を新たにプログラミングする必要はありません。

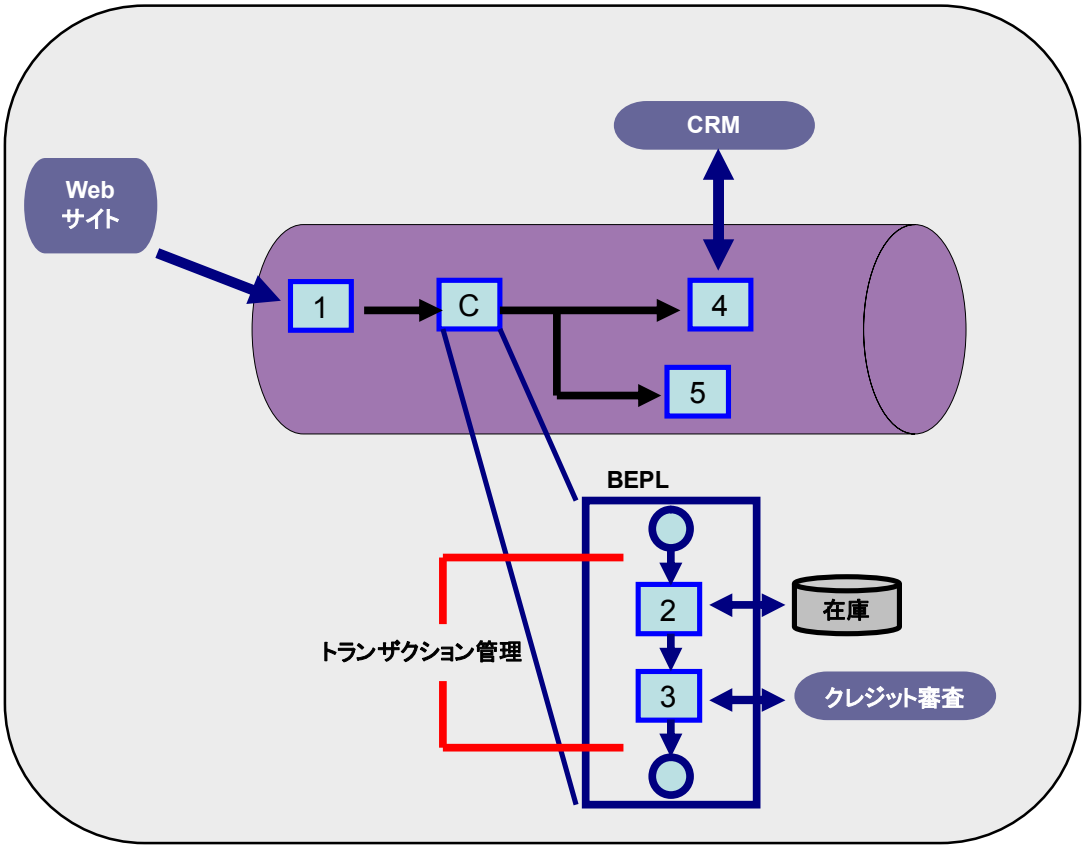


図 17 – プロセスの一部を BPEL フローとし、トランザクション管理を実施

3.5 SOAIF (SOA 実装フレームワーク)

Fiorano Software の SOA プラットフォームは、SOA アプリケーションの設計、構築から運用管理までライフサイクル全体をサポートするもので、ビジネス コンポーネント アーキテクチャの利点を最大限に活かすためのフレームワーク (SOAIF) に基づいて実装されています。

SOAIF は、次に示す 3 つの主要なコンポーネントで構成されています。

- ビジネス コンポーネント パレット
- ツール群 (イベント オーケストレータ、BPEL エディタ)
- SOA プラットフォーム (ESB + BPEL エンジン)

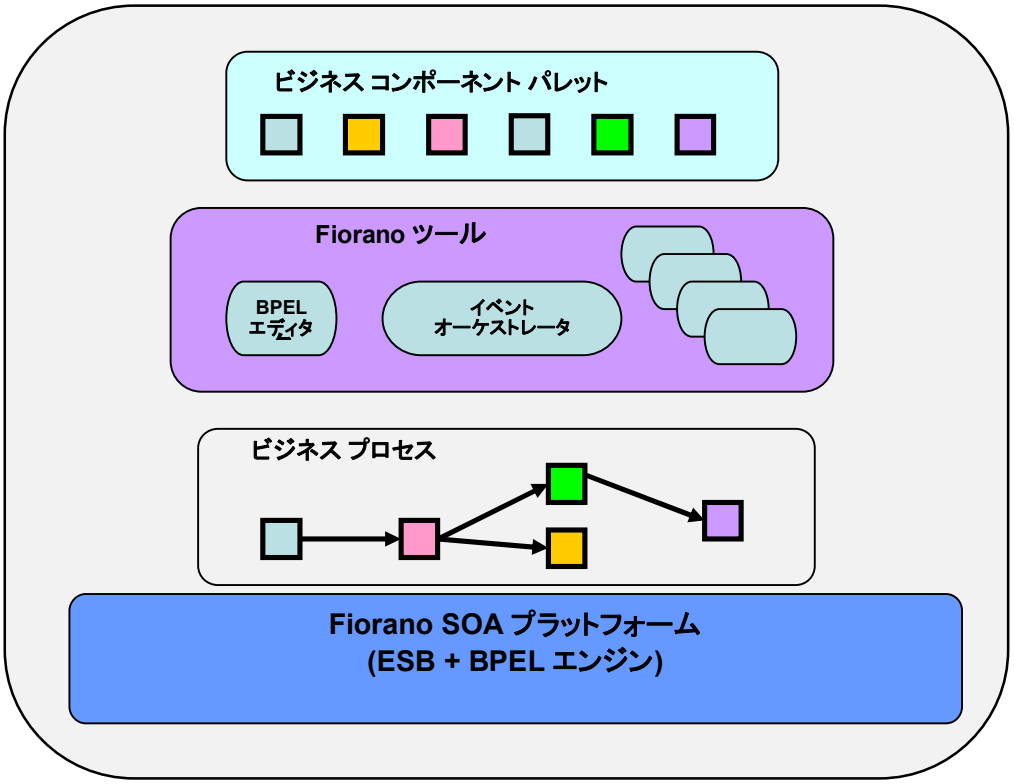


図 18 - SOAIF の主要コンポーネント

図 18 は、SOAIF の主要コンポーネント間およびビジネス プロセスとの間の関係を表したものです。

ビジネス コンポーネント パレット

ビジネス コンポーネント パレットは、ビジネス コンポーネントを格納したリポジトリとして機能し、パレットにあるビジネス コンポーネントはツールによって選択され、ビジネス プロセスに組み込まれていきます。

ツール群

ビジネス プロセスを構築するためのツールおよび実行時に必要となるロギングやセキュリティ管理などのための管理ツールからなっています。Fiorano SOA プラットフォームは、BPEL プロセスを構築するための “BPEL エディタ” とイベント ドリブン プロセスを構築するための “イベント オーケストレータ” の 2 種のビジュアル ツールを備えています。

SOA プラットフォーム

ビジネス プロセスを稼動させるためのプラットフォームで、ESB と BPEL エンジンからなっています。

► まとめ

本ホワイトペーパーで論じてきたポイントは、次の 5 つの結論にまとめることができます。

- SOA アプリケーションとは、ビジネス プロセスを効果的に実現するものである
- ビジネス プロセスは、非同期のプロセスおよび同期プロセスの両方を同時にサポートできるものであること
- SOA アプリケーションは、ビジネス コンポーネント アーキテクチャに基づいて構築する
- SOA アプリケーションの実行プラットフォームである ESB は、地理的にも組織的にも分散されたサービスを１本のバスで結ぶものであること
- SOA をサポートするプラットフォームには、SAO アプリケーションの開発、構築から実行、運用までライフサイクル全体をサポートする機能が備わっていること

このようなポイントは、SOA 製品の選択基準としても十分に通用するものであります。

► Fiorano Software について

Fiorano Software は、カリフォルニアに本社を置く、エンタープライズ インテグレーション ミドルウェアの業界をリードしている企業で、メッセージング インフラストラクチャ技術において数多くのお客様から高い信頼をよせられています。Fiorano のソリューションは、インターオペラビリティ、パフォーマンス、スケーラビリティ、ROI などの面で新たなパラダイムをもたらしています。アメリカン エクスプレス、AT&T ワイヤレス、ボーイング、BP (旧ブリティッシュ ペトロリアム)、エリクソン、FedEx、ロッキード マーチン、モーガン スタンレイ、モトローラ、POSCO、シュルンベルガなどの世界的なリーダー企業で Fiorano の技術が採用され、企業のバックボーン システムとして稼動しております。

Fiorano Software に関する詳細な情報は、弊社のホームページ (www.fiorano.com/jp) をご参照くださるか、info_jp@fiorano.com 宛てに電子メールでお問い合わせください。