特定非営利法人 ものづくり APS 推進機構

APS サミット 2007

システム開発者向け実践セミナー

2007/7/25

ものづくりＡＰＳ推進機構

はじめに

今回の APSOM 実践セミナーでは、APSOM / PSLX の提唱する分散アーキテクチャを実装する一環として、複数のシステム間で PSLX XML メッセージスキーマを利用して、相互連携する I/F 部分を中心として、実装の演習を行います。

具体的には、XML データバインディングにより PSLX XML メッセージを、プログラム中から容易に扱う方法について学び、またシステム連携技術としてメッセージキューによる通信の扱い方を学びます。

ついで、XML データバインディングとメッセージキューを利用して、PSLX の仕様の一部を実装したクライアントおよびサーバを作成し、相互に運用可能なことを確認します。さいごに、ESB を利用して、他システムと連携する方法について学習します。

実装言語は、Java を用います。一部、PSLX クライアントの作成に C# も利用します。

事前準備

- 以下のスペックを満たすノート PC をご持参ください。
  - Windows XP / Vista
  - Pentium 4 2GHz 以上 / 搭載メモリ 512MB (できれば 1GB) 以上
  - 2GB 程度の HDD の空き容量
- 可能であれば、以下のソフトウェアをインストールしておいてください(当日インストールも可能です)。
  - JDK 5.0 Update 12
  - Eclipse 3.2.2
  - Visual C# Express Edition 2005 (SQL Server Express Edition 2005 を含む)

Copyright © 2007 by APSOM
All rights reserved.

第0章　環境準備

今回の実習で利用する環境を整えます。今回は、すべてフリーで入手でき、かつ商用利用に制限のないオープンソース製品のみを利用していますので、作成した成果物については、とくに制限なく業務などにも利用することができます。

開発環境として

- J2SE SDK 1.5.0 Update 12
- Eclipse 3.2.2
- Visual C# Express Edition (.Net Framework v2.0)
- SQL Server Express Edition

実装支援のツール・ライブラリ・ミドルウェアとして

- JAXB　(Sun Web Services Developer Pack 2.0)
- Apache Active MQ 4.1.1
- Mule ESB 1.4.1

を利用します。

JDK 5.0 がインストールされていない方は、配布物の dist フォルダ内にある、jdk-1_5_0_12-windows-i586-p.exe を実行して、インストールしてください。
環境変数 JAVA_HOME をインストール先ディレクトリに指定してください。
(デフォルトは、C:¥Program Files¥Java¥j2sdk1.5.0_12)

Visual C# Express Edition (SQL Server Express Edition) がインストールされていない方は、別途配布する CD-ROM からインストールしてください。

その他必要なファイルは、配布したアーカイブにすべて含まれていますので、適当なフォルダに展開してください。以下は、C:¥apsom-tutorial¥ 以下に展開したものとして、説明しますので、可能であれば C:¥ 直下に展開してください。C:¥ 以外の場所に展開された場合は、適宜読み替えてください。

Copyright © 2007 by APSOM
All rights reserved.

展開したフォルダには、以下のものが含まれます。

¥docs	このテキストなどの参考資料
¥workspace	チュートリアルで使用するソースファイルなど
¥apache-activemq-4.1.1¥	Message Queue 実装(Apache Active MQ)
¥jwsdp-2.0	Sun Java Web Developer Pack (ライブラリ)
¥mule-1.4.1	ESB 実装 (Mule ESB)
¥eclipse	Eclipse 統合開発環境(Eclipse 3.2.2 + 言語パック)
¥dist	配布用アーカイブ類

Copyright © 2007 by APSOM
All rights reserved.

第1章　Lesson1 JAXB を利用したデータバインディング

PSLX のメッセージ仕様は、XML で定義されており、XML スキーマが提供されています。XML のデータをそのままプログラム中で扱うのは、困難ですし、またバグの原因ともなります。

ここでは、JAXB (The Java Architecture for XML Binding)を利用して、PSLX のメッセージを扱うクラスを自動生成し、XML から Java オブジェクトの生成、Java オブジェクトからの XML メッセージの作成が行えることを確認します。

JAXB は、xjc と呼ばれるスキーマコンパイラを利用して、XML Schema から Java のソースファイルを生成します。生成されたソースファイルを利用することで、XML のドキュメントを、 Java のオブジェクトとして扱うことができるようになります。

まず、xjc を利用して、Java のソースファイルを生成してみましょう。
ここでは、Java のビルドツールである ant を Eclipse から利用して、スキーマのコンパイルを行ってみます。

C:¥apsom-tutorial¥eclipse¥eclipse.exe
を起動してください。ワークスペースの選択画面が表示されたら、
C:¥apsom-tutorial¥workspace
を選択してください。

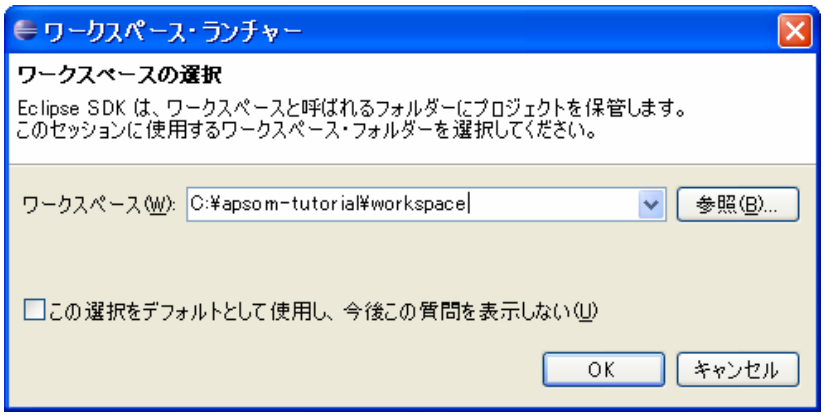

図 1.1 ワークスペースの選択画面

Copyright © 2007 by APSOM
All rights reserved.

起動すると、下のような画面が表示されます。

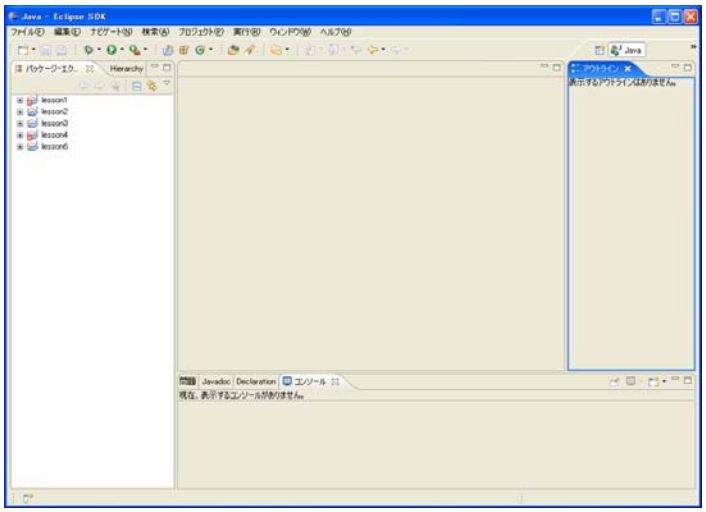

図 1.2 Eclipse の起動画面

起動したら、左側パネルのパッケージエクスプローラから、lesson1 を選択して、開いて下さい。

lesson1/schema に、PSLX version2 で定義されている XML メッセージのスキーマが保存されています。

次に、build.xml をダブルクリックして、内容を表示してください。ここでは、xjc を利用して、schema ディレクトリにある pps-message_top-1.0.xsd ファイルに定義されたスキーマから、Java のソースファイルを生成し、gensrc ディレクトリ以下に保存するという設定が書かれています。

右側のアウトラインから、schema-compile 右クリックし、実行-> Ant ビルドと選択して、スキーマコンパイルを実施してください。

生成されたソースコードは、gensrc 以下に保存されていますので、schema フォルダ内の XML スキーマを見ながら、内容を確認してみてください。

Copyright © 2007 by APSOM
All rights reserved.

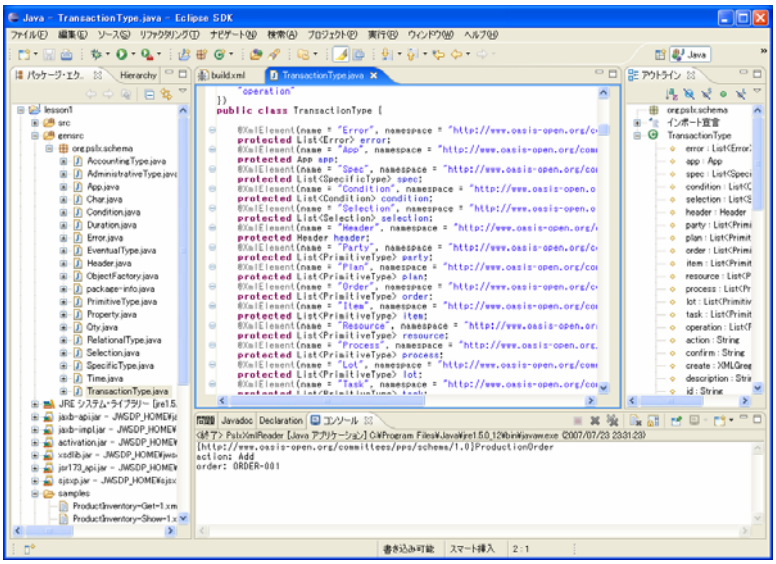

図 1.3 生成されたソースコード

ここまでで、PSLX XML メッセージを扱う Java のクラスは作成できました。それでは、早速、生成されたクラスを利用して PSLX メッセージを読み込んで見ましょう。

XML のメッセージから Java のオブジェクトを作り出す処理を、アンマーシャルと呼びます。実際に動かしてみましょう。src フォルダ内の PslxXmlReader.java ファイルを開いて下さい。右クリックして、実行->Java アプリケーションを選択すると、開いているソースコードをコンパイルして実行できます。

アンマーシャルにより生成された Java オブジェクトを操作してみて、元の XML ファイルに設定されたプロパティやアトリビュートにアクセスできることを確認してみてください。特に複数のプロパティが設定できる場合、どのようなオブジェクトが生成されるかを確認してください。

課題 1-1

PslxXmlReader.java を変更して、サンプルの XML に含まれる Qty 要素の属性を表示できるようにしてください。

今度は、逆に Java のオブジェクトから、XML ドキュメントを作成してみましょう。この処理は、マーシャルと呼びます。PslxXmlWriter.java を開いて実行してください。Eclipse のコンソールに、生成された XML ドキュメントが表示されます。

Copyright © 2007 by APSOM
All rights reserved.

オブジェクトから XML を生成する場合、オブジェクトは ObjectFactory 経由で作成する必要があることに注意してください。

生成するコードと、実際に生成された XML を比較してみてください。

課題 1-2

PslxXmlWriter.java を変更して、サンプルに含まれるファイルと同一の XML ファイルを作成してみてください。

marshal の結果をファイルに書き込むには、以下のようにします。

```
FileOutputStream fos = new FileOutputStream("ファイル名");
marhsaller.marshall (work, fos);
```

課題 1-3

提供された PSLX メッセージファイルを読み込み、Unmarshall した後、生成されたオブジェクトに変更を加えるプログラム PslxXmlEditor.java を作成してください。変更を加えた後のオブジェクトを marshall して、XML ファイルとして保存しましょう。

保存した XML ファイルを開いて、加えた変更が反映されているか確認してください。

Copyright © 2007 by APSOM
All rights reserved.

第2章　Lesson2 ActiveMQ を利用した非同期メッセージ通信

PSLX のメッセージスキーマは、分散したデータベースを持つ多数のシステムが、お互いに非同期にメッセージをやりとりしながら、協調して動作する環境を想定して作成されたものです。

ここでは、非同期メッセージキューのオープンソース実装である Active MQ を利用して、非同期システム間通信を行えることを確認します。

メッセージキューは、複数のアプリケーションの間で非同期のメッセージをやりとりするための仕組みです。送信側と受信側が、1:1 の場合に利用するキューと、1:多の場合に利用するトピックが利用できます。ここでは、キューを利用して、1:1 の通信を行ってみましょう。

メッセーキューは、受信側と送信側がメッセージの仲介を行うブローカを仲介してメッセージのやりとりを行います。送信側は、メッセージをブローカに送り届けた時点で、次の処理に移れるので、受信側が仮に動作していなくても次の処理にうつることができます。

まずは、ActiveMQ のサーバ(ブローカと呼びます)を起動しましょう。

C:¥apsom-tutorial¥apache-activemq-4.1.1
以下に ActiveMQ が含まれています。

コマンドプロンプトを開いて、C:¥apsom-tutorial¥apache-activemq-4.1.1¥bin へ移動し、activemq.bat を起動してください。以下のように表示されれば、正常に起動されています。

Copyright © 2007 by APSOM
All rights reserved.

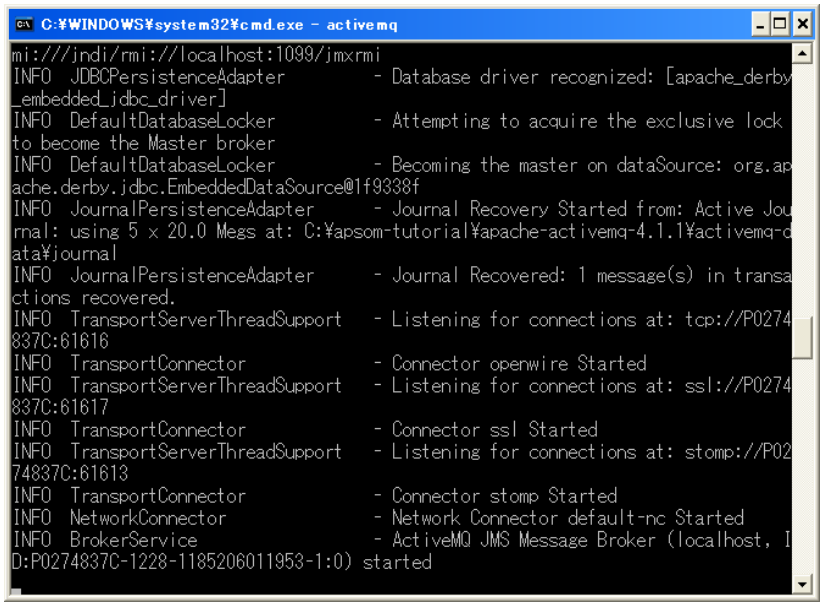

図 2.1 ActiveMQ ブローカの起動画面

ActiveMQ が起動されたら、メッセージキューブローカ上のキューを指定してメッセージを送ったり、受信したりしてみてみましょう。

パッケージエクスプローラから、lesson2 を開いて下さい。

メッセージの送信側が、MessengeSender.java で、受信側が MessageReceiver.java です。
使用するメッセージブローカおよび使用するキューは、jndi.properties に設定されています。

MessageSender.java は、テキストのメッセージを作成し、キューへ送信します。
MessageReceiver.java は、キューからテキストメッセージを受信し、標準出力へテキストを出力します。

MessageSender.java と MessageReceiver.java をそれぞれ起動して、起動の順序にかかわらず正しくメッセージが送られることを確認してください。

これで、非同期メッセージを利用して、システム間でメッセージをやりとりすることができるようになりました。

Copyright © 2007 by APSOM
All rights reserved.

課題 2-1

ここまでは、ブローカ、送信側、受信側も同一のマシン上で実行されていました。jndi.properties を編集して、隣の方の IP アドレスを指定し、マシンをまたいでも、正常にメッセージのやりとりができることを確認してください。

Copyright © 2007 by APSOM
All rights reserved.

第3章　Lesson3 PSLX クライアントの作成

JAXB による XML バインディング、ActiveMQ による非同期メッセージが可能となりましたので、ここで PSLX のメッセージを送信するクライアントを作成して見ましょう。

とりあえず、ファイルから XML メッセージを読み込み、そのままメッセージキューを利用して送信してみます。

受信側は、lesson2 で利用した受信クライアントを利用して、送信したメッセージがどのように送られるか確認してみましょう。

PslxMessageSender.java を開いて下さい。Lesson 2 の MessageSender.java とほぼ同一ですが、プログラム内でテキストメッセージを作成する代わりに、ファイルから XML を読み込みテキストメッセージとしてメッセージキューに送信しています。

PslxMessageSernder.java を実行して、メッセージをキューに送信してみてください。送信したら、Lesson2 の MessageReceiver.java を起動して、メッセージを受信してみてください。

課題 3-1

今回の PslxMessageSender.java では、テキストファイルの内容をそのままキューに送信しています。そのため、不正な XML メッセージでも、そのままキューに送信してしまいます。このような状況を避けるため、テキストファイルを JAXB を利用して読み込み正常な XML ファイルであることを確認してから、送信するようにしてみましょう。

PslxMessageSender.java の createTextMessage を改造して、JAXB を利用してファイルを読み込み(marshal)、unmarshal した結果を、キューに送信するようにしてください。JAXB の読み込み方法については、lesson 1 の PslxMessageReader.java、PslxMessageWriter.java を参考にしてください。

課題 3-2

PslxMessageReceiver は、現時点ではなにも実装されていません。
Lesson 2 の MessageReceiver を参考にして、キューから受信するプログラムを作成してください。Lesson 1 の PslxMessageReader.java を参考にして、受信したメッセージを JAXB で解釈して、内容をコンソールに出力するようにしてください。

Copyright © 2007 by APSOM
All rights reserved.

第4章　Lesson4 PSLX サーバの作成

PSLX のメッセージを送信するクライアントが作成できましたので、今度は送られたメッセージを解釈して、実際に業務ロジックを起動するサーバを作成して見ましょう。

今回は、PSLX のメッセージのうち、オーダ登録および在庫照会を、受け付けるサーバを作成します。メッセージを受け付けたら、登録状況や在庫情報を、別のメッセージキューに PSLX メッセージを書き込むこととします。

これまでの Receiver と違い、メッセージを受信したらすぐに終了するわけにはいきませんので、メッセージの処理方法がこれまでとは変わっています。具体的には、receiver にメッセージリスナを設定して、メッセージを受信したら、onMessage() メソッドが実行されるようにしておきます。

第 3 章で作成した、メッセージ受信プログラムを利用して、メッセージキュー(OrderQueue)から XML 文字列を取り出します。取り出した文字列を unmarshall して Java のオブジェクトを生成しましょう。

受信したメッセージを Java のオブジェクトにすることができましたので、今度は、そのオブジェクトに対してビジネスロジックを実行(データベースへ保存)します。今回は、時間の関係で、ビジネスロジックや永続化のコードは省略していますので、固定のメッセージを作成して、別のメッセージキューへ送信しましょう。

Lesson 4 には、ProductionOrderServer.java と ProductInventoryServer.java が含まれています。それぞれ生産オーダ登録サーバと、在庫照会サーバとなります。
それぞれサーバを起動してから、Lesson 3 で作成した PslxMessageSender と PslxMessageReceiver を利用して、メッセージの送信と受信を行ってみましょう。

それぞれ、利用するキューの名前が、以下のように変更になっていますので、適宜ソースコードを修正してください。

製品オーダ登録キュー	ProductionOrderQueue
製品オーダ登録ステータスキュー	ProductionOrderStatusQueue
在庫照会キュー	ProductInventoryQueue
在庫照会ステータスキュー	ProductInventoryStatusQueue

Copyright © 2007 by APSOM
All rights reserved.

第5章　Lesson5 Visual C# 2005 によるクライアントの実装

Java と比較して、Visual C# を利用すれば、比較的簡単にリッチな UI を持つクライアントを作成できます。ここでは、Visual C# 2005 Express Edition を利用して、オーダを送信し、オーダのステータスを確認できる GUI アプリケーションを作成してみましょう。

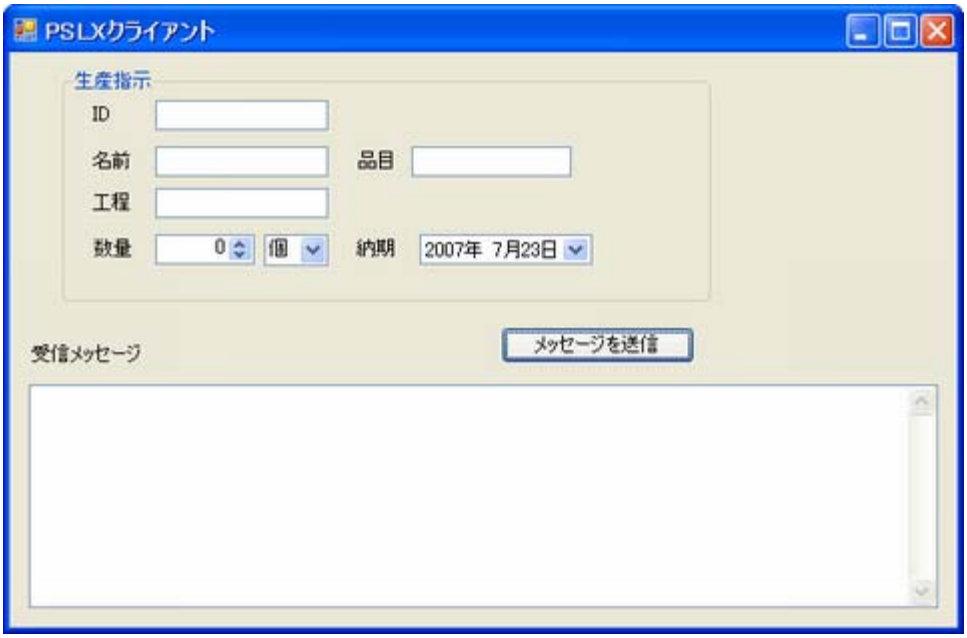

図 5.1 Visual C# 2005 による PSLX クライアント

5.1.　Visual C# 2005 のプログラムの場所

Lesson5 のプログラムは、以下の場所に用意されています。

C:¥apsom-tutorial¥workspace¥lesson5¥ProductionOrder¥ProductionOrder.sln

5.2.　PSLX スキーマから C#クラスを生成する

午前の Lesson1 では、JAXB を利用して PSLX のメッセージを扱う Java のクラスを自動生成しました。Visual C#(.NET Framework)にも同様の仕組みが用意されています。Visual C#に付属している「XML スキーマ定義ツール(xsd.exe)」を使うことで XML スキーマから C#のクラスを自動生成することができます。

今回の Lesson5 のプログラムには、あらかじめ PSLX XML スキーマから C#のクラスへ変換したクラスを用意しました。(pps-message_top-1_0.cs)　このクラスを用いて C#で PSLX メッセージを作成します。

Copyright © 2007 by APSOM
All rights reserved.

5.3. 画面の作成

では、Visual C# 2005 Express Edition を使って、GUI アプリケーションを作ります。Lesson5 のプログラムを開いた状態で、右上の「ソリューションエクスプローラ」にある「ProductionOrder.cs」をダブルクリックして開きます。しばらくすると、中央にウィンドウの設計画面が表示されます。

左側には、「ツールボックス」があり、この中からボタンやテキストなどの部品を選択し、ウィンドウに貼り付けることで、GUI を作成することができます。

図 5.2 ソリューション エクスプローラ

課題 5.1

Lesson5 のウィンドウには、名前、品目、工程の入力欄が用意されていません。図 5.1 のように名前(textName)、品目(textItem)、工程(textProcess)の各 TextBox を貼り付けてみましょう。

部品の名前を変更するには、右下の「プロパティ」にある「(name)」の項目を入力します。

Copyright © 2007 by APSOM
All rights reserved.

5.4. PSLX メッセージの作成

次に、ウィンドウに入力された値をもとに、PSLX メッセージを作成するプログラムを作成しましょう。中央のウィンドウにある「メッセージを送信」ボタンをダブルクリックしてプログラムを開きます。

プログラムを開くと、以下のようなプログラムが表示されます。すでに大半のプログラムは作成されています。

```
private void buttonSend_Click(object sender, EventArgs e)
{
    //ProductionOrder要素
    TransactionType productionOrder = new TransactionType();
    productionOrder.ItemsElementName = new ItemsChoiceType[] { ItemsChoiceType.
    productionOrder.action = "Add";
    productionOrder.sender = "client";

    //Order要素
    PrimitiveType order = new PrimitiveType();
    productionOrder.Items = new PrimitiveType[] { order };

    order.id = textID.Text;

    /* ここでname item processをそれぞれ設定しましょう */
```

図 5.3 メッセージを作成するプログラム(1)

Java の PSLX メッセージの作成するプログラムとは異なりますが、Java も C#も同じ PSLX XML スキーマから生成されたクラスを使用しているため、要素ごとにオブジェクトを作り、値を代入するということには変わりありません。

メッセージのもととなる<ProductionOrder>要素を表すオブジェクト(TranscationType)を生成した後、子要素の<Order>要素を表すオブジェクト(PrimitiveType)を生成して、オブジェクトへ値を設定してきます。

Lesson5 のプログラムでは、すでに<Order>要素の id 属性にテキストボックス(textID)に入力した文字列を代入するように作ってあります。

Copyright © 2007 by APSOM
All rights reserved.

課題 5.2

さきほど貼り付けたテキストボックス(名前(textName)、品目(textItem)、工程(textProcess))に入力した内容を、order オブジェクトのそれぞれの属性(name、item、process)へ代入するプログラムを作成してみましょう。

つづいて、納期を表す<Event>要素と、数量を表す<Qty>要素を追加します。

<Event>要素と<Qty>要素は、さきほどの<Order>要素の中に入る子要素です。<Order>要素に子要素を持たせるには、order オブジェクトの Event プロパティまたは Qty プロパティへ配列してオブジェクトを代入して指定します。

```
//Event要素
Time time = new Time();
time.value = dateDue.Value;
time.valueSpecified = true;

EventualType due = new EventualType();
due.type = "pps:due";
due.Time = new Time[] { time };
order.Event = new EventualType[] { due };


//Qty要素
/* ここでQty要素を追加しましょう */
```

図 5.4 メッセージを作成するプログラム(2)

課題 5.3

Event 要素は、すでにプログラムが作られています。同じように Qty へ数量と単位を代入するプログラムを作ってみましょう。

プログラム例

```
Qty qty = new Qty();
qty.value = numQty.Value;
qty.unit = comboUnit.Text;
qty.valueSpecified = true;

order.Qty = new Qty[] { qty };
```

これで、オーダメッセージが完成しました。

Copyright © 2007 by APSOM
All rights reserved.

5.5. オブジェクトから XML メッセージへの変換

完成したオーダメッセージは C#のオブジェクトであり、ActiveMQ へメッセージを送るためには、オブジェクトを XML へバインディングする必要があります。Lesson1 の marshall と同様の作業が必要です。

.NET Framework では、オブジェクトと XML との変換に XmlSerializer クラスを利用します。このクラスには、オブジェクトから XML を生成する「Serialize(シリアライズ)」メソッドと、XML からオブジェクトを生成する「Deserialize(デシリアライズ)」メソッドが用意されています。

```
//メッセージのシリアライズ
XmlRootAttribute xRoot = new XmlRootAttribute();
xRoot.ElementName = "ProductionOrder";
xRoot.Namespace = "http://www.oasis-open.org/committees/pps/schema/1.0";
xRoot.IsNullable = true;

XmlSerializer serialiser = new XmlSerializer(typeof(TransactionType), xRoot);
MemoryStream ms = new MemoryStream();
StreamWriter writer = new StreamWriter(ms, Encoding.UTF8);
serialiser.Serialize(writer, productionOrder);
writer.Close();

//メッセージの送信
Send(Encoding.UTF8.GetString(ms.ToArray()).Substring(1));
```

図 5.5 メッセージを送信するプログラム

シリアライズする際に、XmlRootAttribute で XML のトランザクション要素名(ProductionOrder)と PSLX の名前空間を指定しておきます。

その後、XmlSerializer クラスを生成して、Serialize メソッドで productionOrder オブジェクトを指定します。

シリアライズされた XML を UTF-8 の文字列として格納するために、MemoryStream および StreamWriter を使います。ファイルとして書き出す場合は、StreamWriter を生成する際にファイル名を指定します。

5.6. ActiveMQ への送信

C#から ActiveMQ の送受信をおこなうには、NMS を利用します。Lesson5 のプログラムではキューを ActiveMQ へ送るための Send メソッドがすでに作られています。ActiveMQ へ送る手順は、Lesson2 と同じです。

Copyright © 2007 by APSOM
All rights reserved.

5.7. クライアントの動作テスト

では、サーバと通信させてみましょう。

まず、ActiveMQ のサーバ(ブローカ)が起動していることを確認して、
Eclipse で Lesson4 の ProductionOrderServer を実行してください。

次に、Visual C# 2005 Express Editionのメニューから「デバッグ(D)」→「デバッグ開始(S)」(F5)でプログラムを実行します。

生産指示の中の各入力欄に値を入力して「メッセージを送信」ボタンをクリックします。

ボタンがクリックされると、Lesson5 で作成したプログラムから ActiveMQ の ProductionOrder キューをメッセージが送られます。Lesson4 で作成したプログラムがこのメッセージを受け取り、データベースへ書き込み、ProductionOrderStatus キューへ完了メッセージが格納します。その後、Lesson5 で作成したプログラムが、完了メッセージを受け取り、「受信メッセージ」へ XML を表示します。

図 5.6 PSLX クライアントの実行例

Copyright © 2007 by APSOM
All rights reserved.

5.8. 応用編　在庫照会クライアントの実装

ここからは応用編として、在庫照会クライアントを実装します。先ほどのプログラムでは、メッセージを送信し、結果を XML メッセージで受け取るだけでしたが、XmlSerializer クラスを使って、XML メッセージをデシリアライズしてオブジェクトを生成し、リストへ結果を表示させることも可能です。PSLX メッセージの通信の応用例として在庫照会プログラムを作ってみましょう。

図 5.7 PSLX クライアントの実行例

5.9. Visual C# 2005 のプログラムの場所

在庫照会クライアントのプログラムは、以下の場所に用意されています。

C:¥apsom-tutorial¥workspace¥lesson5¥ProductInventory¥ProductInventory.sln

Copyright © 2007 by APSOM
All rights reserved.

5.10. XML からオブジェクトを生成する

ソリューションエクスプローラから「ProductionInventory.cs」を選び、上側の「コードの表示」アイコンをクリックすることでプログラムを開くことができます。

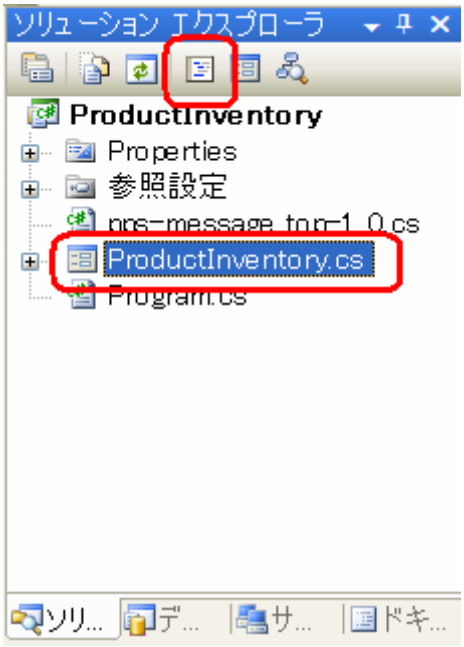

図 5.8 ソリューションエクスプローラからプログラムを開く

buttonSend_Click メソッドでは、在庫問い合わせをするための PSLX メッセージを作成して、ActiveMQ のキューへ送ります。

Receive メソッドは、ActiveMQ から回答メッセージを受け取ったときに実行されます。ここで受け取った回答メッセージは、XML の文字列です。XmlSerializer クラスを使って、デシリアライズして XML からオブジェクトを生成します。

```
//メッセージのデシリアライズ
XmlRootAttribute xRoot = new XmlRootAttribute();
xRoot.ElementName = "ProductInventory";
xRoot.Namespace = "http://www.oasis-open.org/committees/pps/schema/1.0";
xRoot.IsNullable = true;

TransactionType productInventory = new TransactionType();

XmlSerializer serialiser = new XmlSerializer(typeof(TransactionType), xRoot);
StringReader reader = new StringReader(xml);
productInventory = (TransactionType)serialiser.Deserialize(reader);
reader.Close();
```

図 5.9 XML からオブジェクトを生成するプログラム

Copyright © 2007 by APSOM
All rights reserved.

シリアライズの際と同様に、受け取ったメッセージの要素名と名前空間を指定し、XmlSerializer クラスを生成します。デシリアライズするときは、Deserialize メソッドを使います。受け取った XML の文字列を操作するために StringReader クラスを使います。XML ファイルからデシリアライズする場合は StreamReader クラスを使います。

デシリアライズされたオブジェクトは、productInventory オブジェクトに格納されます。このオブジェクトには、XML に記述された在庫情報が格納されていますので、順番にリストへ出力します。

```
//リストへ追加
//Item要素
foreach (PrimitiveType item in productInventory.Items)
{
    //Capacity要素
    SpecificType capacity = item.Capacity[0];

    //Qty要素
    Qty qty = capacity.Qty[0];

    //リストへ追加
    AddToList(item.id + "¥t" + qty.value + qty.unit);
}
```

図 5.10 オブジェクトからリストへ在庫情報を追加するプログラム

今回の例では、送られてくるメッセージの形式がわかっているため、Capacity 要素の配列や Qty 要素の配列の 0 番目から値を取得していますが、本来はサーバからのエラーメッセージなどによって Capacity 要素や Qty 要素が存在しない場合もあることから、要素があることを確認した上で、取得操作を行うべきです。

5.11. クライアントの動作テスト

では、サーバと通信させてみましょう。ActiveMQ のサーバ(ブローカ)が起動していることを確認して、Eclipse で Lesson4 の ProductionInventoryServer を実行してください。
その後、Visual C# 2005 Express Editionのメニューから「デバッグ(D)」→「デバッグ開始(S)」(F5)でプログラムを実行します。

在庫問い合わせを行う ID を入力欄に入力して「問い合わせ」ボタンをクリックします。サーバから、在庫問い合わせの回答を受け取り、下側のリストへ表示します。

Copyright © 2007 by APSOM
All rights reserved.

第6章　Lesson6 システムとの連携　– 既存システムとの連携(ESB の利用)

ここまでは、ActiveMQ を利用して、いろいろな実装言語やマシンから同一のサーバが利用できることを確認してきました。ただ、生産情報システムは、スクラッチから作成できるわけではなく、通常は多くの既存システムが存在して、順次置き換えるという手順になる場合が多くあります。

ここでは、Mule ESB というエンタープライズサービスバスの実装を利用して、既存システムを想定したインタフェースから PSLX サーバを呼び出す手順を検討してみましょう。

まずは、オーダをファイルで受け取り、ステータスもファイルで返す例を作ってみましょう。入出力ともに PSLX XML メッセージであることを想定すれば、実装は行う必要はなく、Mule ESB の設定ファイルのみの作成で連携させることができます。

それでは、やってみましょう。

ActiveMQ が起動していることを確認してから、Mule ESB を起動しましょう。
Mule ESB を利用してファイルの中身をメッセーキューへ送信する構成は、mule-config-filetojms.xml に保存されています。これは、特定のフォルダにおかれた xml ファイルを読み込み、指定されたメッセージキューに送信することと、指定されたメッセージキューに送信されたメッセージをファイルに保存するようになっています。

構成ファイルを開いた状態から、右クリックし、実行 -> 構成および実行　を選択します。左のパネルから MuleServer を選択してください。

Copyright © 2007 by APSOM
All rights reserved.

以下のような画面が表示されます。

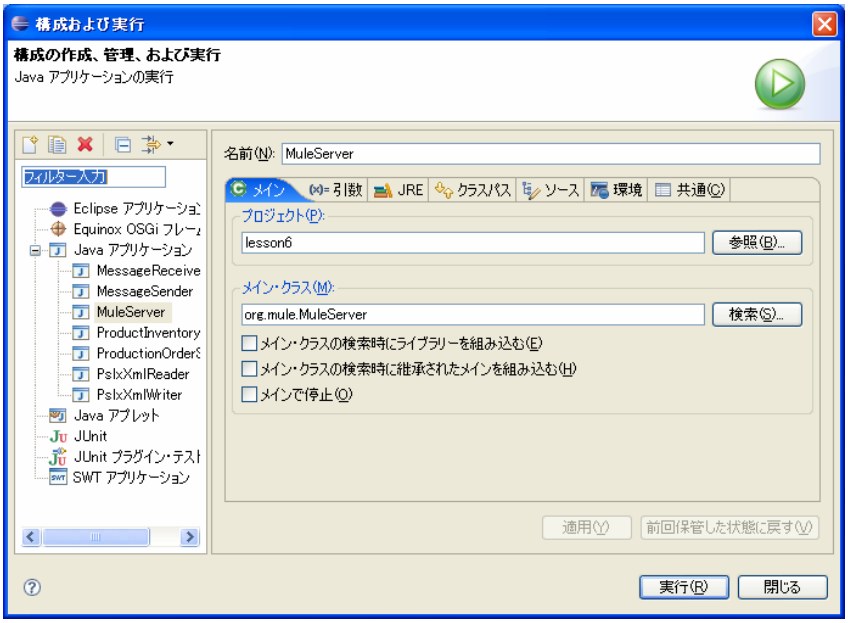

図 6.1 Mule サーバの起動

メインの隣の引数タブを開き、引数に、
-config mule-config-filetojms.xml
が指定されていることを確認します。

起動したら、lesson6/infile フォルダに、PSLX サンプルメッセージをコピーしてみましょう。まもなくファイルを読みこみ(読み込まれたファイルは削除されます)、内容をキューに送信します。キューの送信した結果、ステータスキューから結果を受信したら、内容を標準出力へ出力します。

このように、設定ファイルだけでファイルの受け渡しが可能となりました。ESB と付属するトランスポートを利用する利点は、メッセージスキーマさえ決定していれば、実際のメッセージをやりとりする実装は、ほぼ設定のみで吸収できることです。

今度は、ファイルの読み書きであったものを、ftp を利用したリモートファイルサーバとのやりとりに変更してみましょう。今回も設定ファイルのみで、実行可能です。

さきほどの、引数を
-config mule-config-ftptojms.xml
に変更して、MuleServer を起動してください。

Copyright © 2007 by APSOM
All rights reserved.

指定された ftp サーバのフォルダを監視し、ファイルがあれば、さきほどと同様な動作を行います。

今度は、ステータスキューから受信したデータをメールで送信してみましょう。

さきほどと同様に、引数を
-config mule-config-jmstosmtp.xml
に変更して MuleServer を起動すればステータスキューから受信したメッセージをメールで指定されたあて先に送信します。

Mule ESB を利用すると、今回利用したファイル、ftp、メールなどのほかにも、さまざまなシステム連携方式に対応することができます。また、今回は利用していませんが、連携時に様々なフォーマット変換も実施することができますので、CSV や固定長などの既存の連携フォーマットおよび方式と、今回作成した PSLX XML / JMS のサーバを柔軟に連携させることが可能となります。

Mule-1.4.1 フォルダの examples の中に、いくつかのサンプルが含まれていますので、お使いの環境にあわせて、いろいろと実験してみてください。

Copyright © 2007 by APSOM
All rights reserved.

第7章　さいごに

PSLX メッセージスキーマを利用して、I/F を構築する実例を見てきましたがいかがでしたか？実際に利用するには、エラー処理や例外処理のコード、障害時に復旧させるための考慮などが必要ですが、思ったより簡単に PSLX スキーマを利用した I/F を作成できることがおわかりいただけたのではないかと思います。

Copyright © 2007 by APSOM
All rights reserved.