文書番号：MOE10-業仕様-4003

平成１６年度

流通サプライチェーン全体最適化促進事業

＜実証実験＞

「業務ＡＰアダプタ開発書」

平成 17 年 3 月

日本電気株式会社

改版履歴

日付	版数	改版内容
2005 年 3 月 31 日	初版	新規

実証実験統括責任者
日本電気株式会社　：曽根田　雄一

検　印

目 次

１．はじめに

本書は、流通 SCM 事業にて実施する、ビジネスモジュール※1 を用いた企業間取引の実証実験の業務アプリケーションアダプタ※2 の作成方法について記述するものである。

※１ ビジネスモジュールとは、国内流通業界の企業間取引を効率化することを目的として、本事業において設計・開発された企業間通信システムである。本書では、ビジネスモジュールの詳細については説明しない。

※２ 業務アプリケーションアダプタとは、ビジネスモジュール上で動作するコンポーネントで、既存業務システムとインタフェースをする役割を持つものである。以下、業務ＡＰアダプタと略記する。

１.１ 本書の対象者

本書は業務 AP アダプタの設計及び実装者を対象とする。本書の読者はビジネスモジュールに関する詳細な理解がなくてもよい。

ただし、Java 言語や XML の基礎知識、Excel シートについての基本的な使い方の知識は有するものとする。また、JEDICOS-XML についての知識も有するものとする。

1.2 用語の説明

本書で用いる用語で特殊なものについて下記に説明する。

表 1.1 用語の説明

用語	説明
JEDICOS-XML	2002 年度より流通システム開発センターによって策定が開始されている、XML を利用した流通業界の EDI メッセージ仕様。
業務ＡＰ	流通業界の EDI メッセージを扱う業務アプリケーション。既存のホスト(汎用機)、ERP パッケージ、中継サーバなどを指す。
ビジネスモジュール	実証実験で用いる企業間通信システム。
メッセージ	ビジネスモジュール間で送受信する情報のこと。
業務ＡＰアダプタ	ビジネスモジュールの中の１つのモジュール。業務ＡＰとのインタフェースをする役割を持つ。
送信アダプタ	業務ＡＰアダプタのうち、業務ＡＰからの業務データをビジネスモジュールに渡す処理を行う。
受信アダプタ	業務ＡＰアダプタのうち、ビジネスモジュールから業務データを業務ＡＰに渡す処理を行う。
個別実装部分	業務ＡＰアダプタのうちカスタマイズが必要な部分を各企業にあわせてカスタマイズする部分
フレームワーク	業務ＡＰアダプタのうち共通に利用できる抽象クラスや自動生成ツールのこと。
業務データ	業務ＡＰが扱う業務データ。取引情報や決済情報などデータ交換の対象となるデータのこと。
業務オブジェクト	ビジネスモジュールが扱う JEDICOS-XML 相当のオブジェクト（Java オブジェクト）

２．業務ＡＰアダプタの概要

２.１ ビジネスモジュール

本実験で用いるビジネスモジュールのシステム構成を以下に示す。

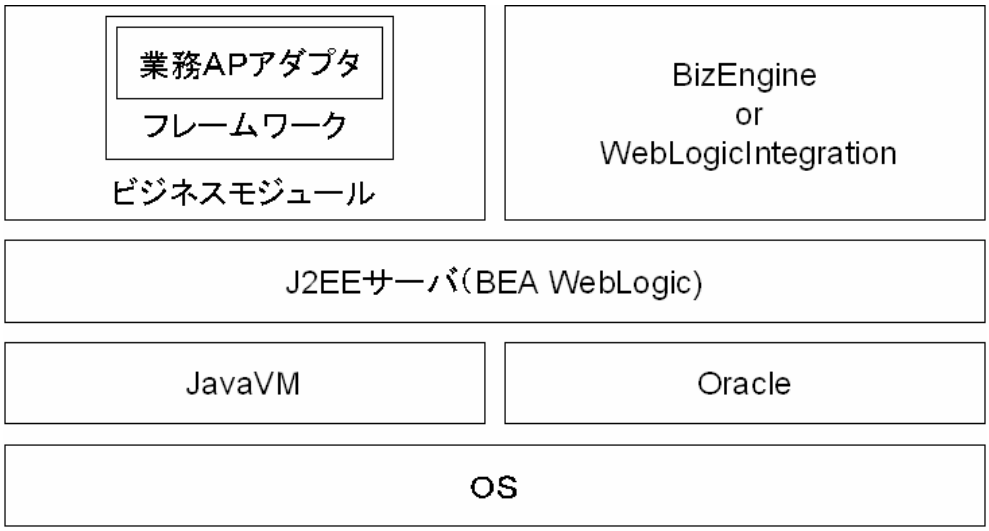


図 ２.１ ビジネスモジュールのシステム構成

本書では分かり易くするために業務ＡＰアダプタとビジネスモジュールを分けて記述しているが、業務ＡＰアダプタはビジネスモジュールの中のモジュールの１つである。また、業務ＡＰアダプタは各企業でカスタマイズすべき個別実装部分とフレームワーク部分に分けられる。

２.２ 業務ＡＰアダプタ

業務ＡＰアダプタでは、大きく分けて２つの処理を行う。

処理１：業務ＡＰからのデータをビジネスモジュールに渡す。

処理２：ビジネスモジュールからデータを業務ＡＰに渡す。

処理１の処理を**送信アダプタ**、処理２の処理を**受信アダプタ**と呼ぶ。それぞれの処理の概要を図２．２、図２．３に示す。

ここでは業務アプリから HULFT を用いてファイルを転送する例を示しているが、これは業務アプリからのデータを渡す方法の一例であり、各企業でデータを渡す方法（図２．２の①）および、ビジネスモジュールに通知する方法（図２．２の②)、ビジネスモジュールから通知を受け取る方法（図２．３の⑪）については個別設計する必要がある。

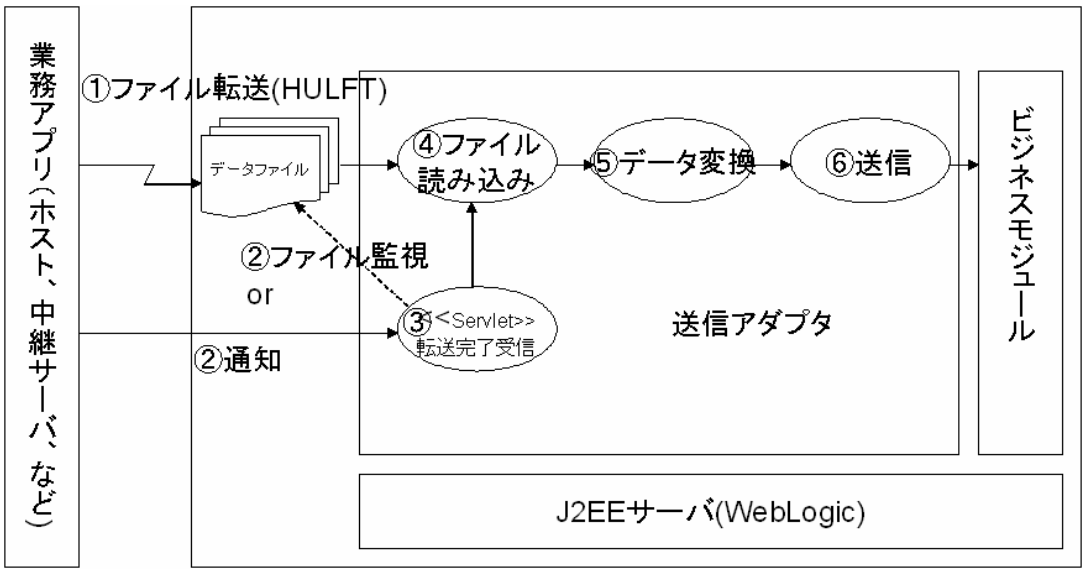


図 2.2　送信アダプタの処理

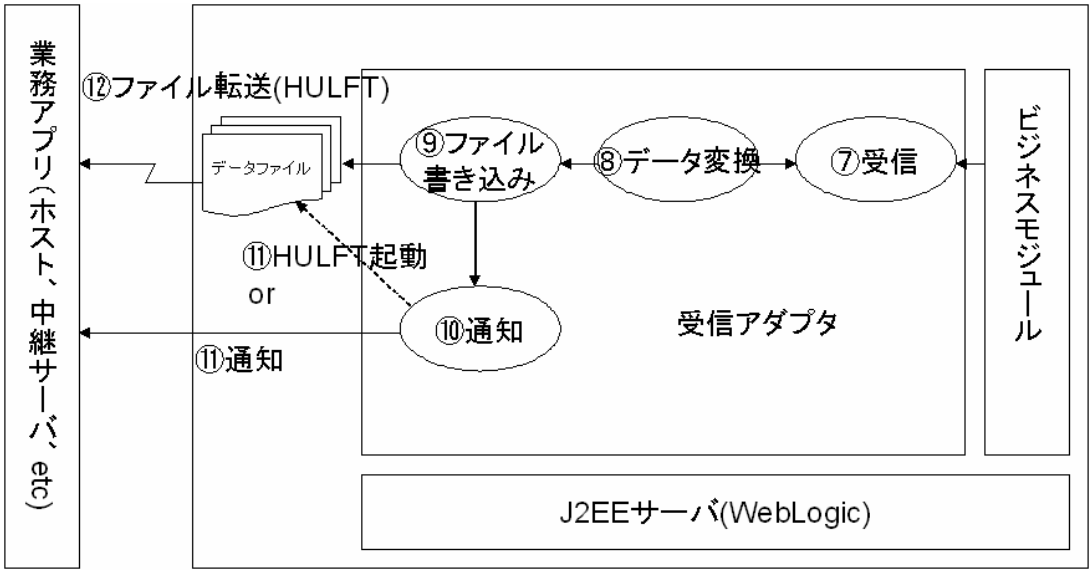


図 2.3　受信アダプタの処理

図２．２と図２．３を見てわかるように、処理１、処理２はそれぞれの処理の向きは逆であるがほぼ対称的な処理を行う。それぞれの処理に対して業務 AP アダプタでは以下の機能を提供する。

機能 1. ファイルの I/O 機能（図２．２の④と図２．３の⑨に相当）

CSV または固定長形式のファイルの読み書きを行なう。

機能 2. データの変換機能（図２．２の⑤と図２．３の⑧に相当）

業務データから業務オブジェクトへ変換する。または業務オブジェクトから業務データに変換する。

機能 3. ディスパッチ機能（図２．２の⑥と図２．３の⑦に相当）

業務ＡＰアダプタからビジネスモジュールの呼び出し、ビジネスモジュールから業務ＡＰアダプタの呼び出しを行なう。

機能 4. 通知機能（図２．２の③と図２．３の⑩に相当）

なんらかのトリガーを受けて送信アダプタの起動を行なう。トリガーとなるのはファイル転送バッチによる呼び出しやディレクトリの監視による起動などが考えられる。

逆に、送信アダプタから業務ＡＰへの通知を行なう。業務ＡＰへの通知にはエラー通知も含まれる。

通知の方式は各企業で個別に設計する必要がある。

（機能 5）．ファイル転送機能及び連携機能（図２．２の①、②、図２．３⑪、⑫に相当）

業務ＡＰからデータを取得し、業務ＡＰアダプタと連携する。

この部分も各企業での方式にしたがって個別に設計する。この部分は業務ＡＰアダプタ内で行なう処理ではない。

これらの機能は個々の企業でカスタマイズが必要な部分と共通に利用できる部分に分かれる。カスタマイズが必要な部分を**個別実装部分**、共通に利用できる部分を**フレームワーク**と呼ぶ。

個別実装部分を容易に作成できるようにフレームワークとして機能１～４の個別実装部分（クラスファイル）を自動生成するツール、共通の処理を行う抽象クラスを提供する。

業務ＡＰアダプタの実装者は機能５および上記で作成された個別実装部分を各企業の方式にあわせて作成しなければならない。個別実装部分の作成に関しては、リファレンス実装を提供するので参照にしていただきたい。

３．作成手順

この章では、業務ＡＰアダプタを、フレームワークを用いて作成する手順を説明する。業務ＡＰアダプタの詳細なＡＰＩについては JavaDoc ドキュメント(HTML)を参照していただきたい。

3.1 全体の流れ

業務ＡＰアダプタの作成の流れを図３．１に示す。ここでは、2.2 で説明した業務ＡＰアダプタの機能１～３をフレームワークで作成する手順を示している。この作業と平行して、機能４～５を実装するための方式の検討やその設計などを行なう必要がある。

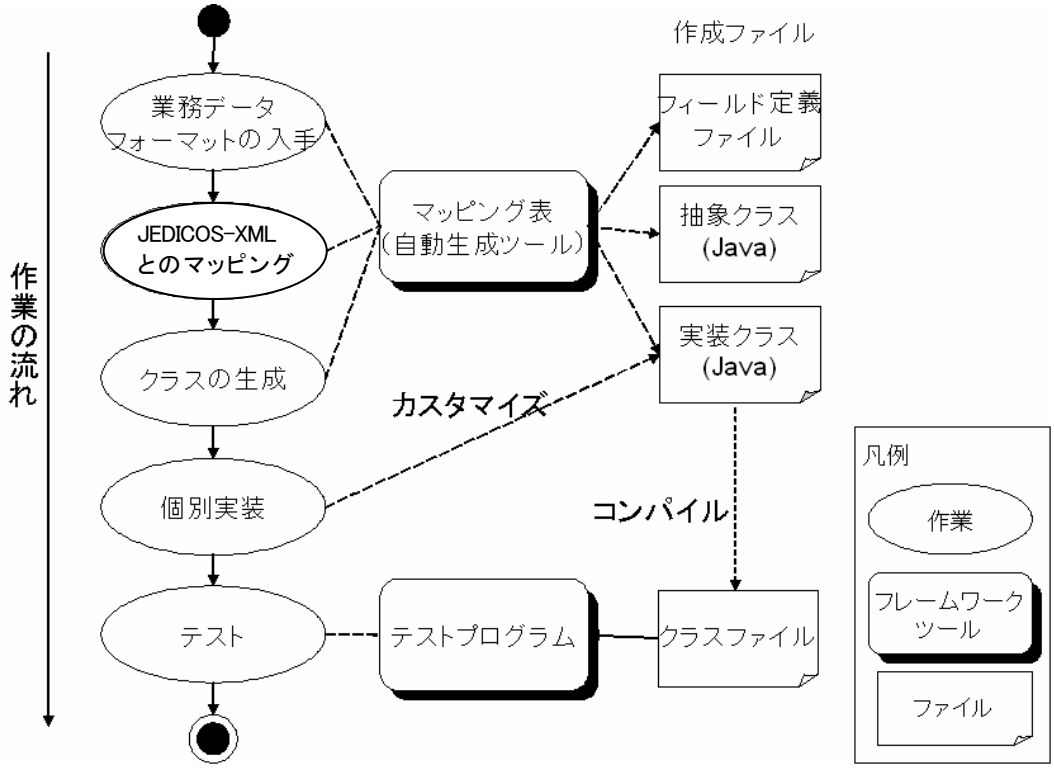


図 3.1　業務ＡＰアダプタ作成の流れと作成ファイル

3.2 業務データのフィールド定義フォーマットの入手

ビジネスモジュールでは表３．２のメッセージを企業間で交換する。ここでは交換するメッセージに対応する業務データの（ファイル中の）フィールド定義フォーマットを入手する。表３．２には参考のため、業務データの一般的な呼称をあげた。

業務ＡＰフレームワークでは、ファイルの形式としてはカンマ区切りのＣＳＶ形式および固定長形式に対応している。

表 3.2 ビジネスモジュールで交換するメッセージの種類と JEDICOS-XML 仕様

#	メッセージの種類	作り手	受け手	対応する JEDICOS-XML 仕様	（参考）業務データの一般的な呼称
1	発注	発注者	受注者	発注	発注
2	受注回答	受注者	発注者	発注	受注情報、欠品情報
3	出荷指示	(業務のフローにより異なる)	出荷拠点	入荷予定梱包	出荷指示、発注情報
4	出荷確定	出荷拠点	(業務のフローにより異なる)	入荷予定梱包	ＡＳＮ（出荷側）
5	入荷予定	(業務のフローにより異なる)	入荷拠点	入荷予定梱包	ＡＳＮ（入荷側）、発注情報
6	入荷確定	入荷拠点	発注者	検品受領	受領、仕入
7	仕入	発注者	受注者	検品受領	仕入、買掛、返品、訂正など
8	売上	受注者	発注者	検品受領	売上、売掛、訂正など ※POS 売上とは異なる
9	支払案内	発注者	受注者	支払案内	入金予定、支払予定
10	請求	受注者	発注者	請求	請求
11	自由記述	任意	任意	商品マスタ	商品マスタ

入手した業務データフォーマットは次項でのマッピング表に転記する。マッピング表への記入の仕方については次項以降で説明する。

3.3 マッピング表の作成

マッピング表の作成について以下の作業の順に説明する。

①マッピング表の起動

②業務データフォーマットの記入

③マッピングの定義

④設定値の変更

⑤ファイルの自動生成

3.3.1 マッピング表の起動

マッピング表(BmMapping.xls、配布の ZIP ファイルの etc フォルダに含まれる)は Microsoft Excel2002 で作成されており、ファイルの自動生成を行なうために VBA の機能を使用している。起動時に以下のダイアログが表示されるが、[マクロを有効にする(E)]を選択して起動する。

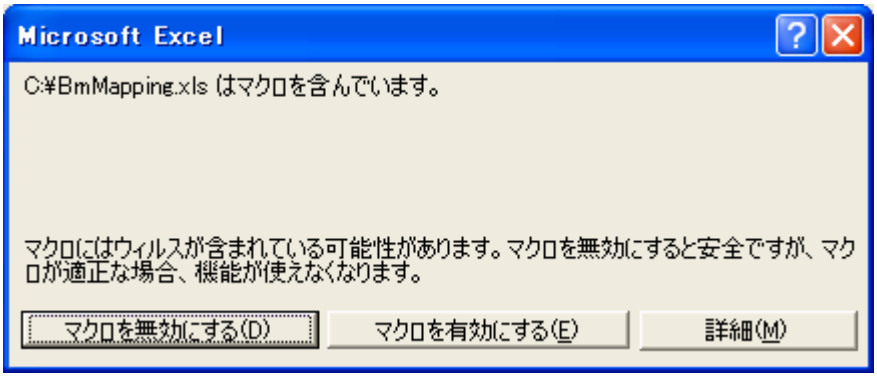


図 3.3　マクロの有効化

上記のダイアログが表示されない場合にはメニューの「マクロ」－「セキュリティ」を選択してセキュリティレベルを低にして再起動すると、ダイアログが表示される。(使用後はセキュリティレベルを元に戻すこと。)

マッピング表（ブック）には以下の６つのシートがある

表 3.4 マッピング表に含まれるシート一覧

#	シート名	使い方	説明
1	記入上の注意点	参考にするのみ	記入上の詳細な注意点が記述されている。本手順書と内容に矛盾がある場合は、こちらを優先することとする。
2	Order	コピーして使用	マッピングシート。業務データのフィールド定義と JEDICOS-XML とのマッピングを行なう。業務データの種類毎に作成する。
3	Order 定数	コピーして使用	定数シート。業務データに含まれない定数を JEDICOS-XML にマッピングしたい場合に使用する。業務データの種類毎に作成する。
4	設定	このシートに記入	設定シート。固定長形式のバイト数。固定長形式の場合は必須。CSV 形式の場合は無視される。
5	種別	このシートに記入	種別シート。小数点以下の桁数。固定長形式の場合のみ有効。CSV 形式の場合は無視される。
6	XPATH 一覧（各メッセージ毎）	参考にするのみ	JEDICOS-XML の各メッセージの XPATH を一覧で参照することができる。マッピング定義を行う際に参考にする。

3.3.2 業務データフォーマットの記入

まず、「Order」と記述されているシートをアクティブシートにする。このシートは別紙１として添付しているので参照のこと。

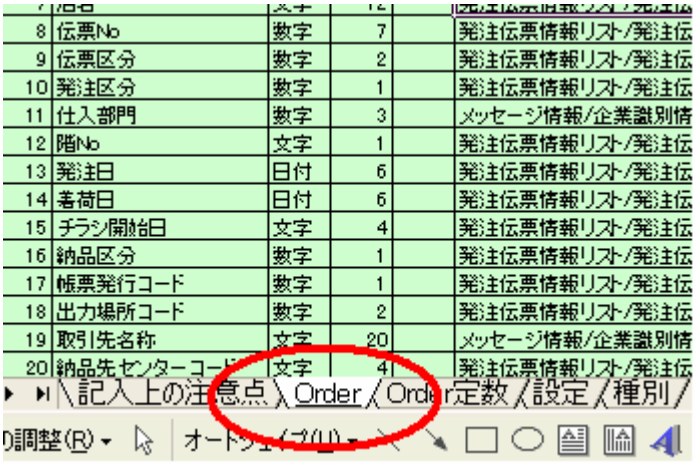


図 3.5 マッピングシート（コピーして使う）

このシートは表２での項番#1 の「発注」のフィールド定義の例である。

新規のフィールドの定義を行う場合には、このシートをコピーして使用する。(シートのコピーはシート名を右クリックして「移動またはコピー」を選択する。) シート名は、後述の自動生成されるフィールド定義ファイルのファイル名およびクラス名の接頭辞となるので英文字で[企業名]+[メッセージ名]などの適切な名前とする。

前項で入手した業務データフィールド定義フォーマットを、フィールド毎表に記入する。記入すべき項目は以下の通りである。

表 3.6 マッピング表の項目

#	表の項目	例	必須か否か	説明
1	NO	1、2、3	必須	項目の順番（項目の出現順に書く必要がある。）
2	項目名	店コード	必須	項目の名称。(「項目名と JEDICOS-XML との対応」に XPath が記入されていた場合に JEDICOS-XML に格納される。)
3	種別	文字、数字、日付、予備	必須	項目のタイプ。文字、数字、日付、予備のいずれかを記入する。
4	桁数	2、3	固定長の場	固定長形式のバイト数。固定長形式の場

			合は必須	合は必須。CSV 形式の場合は無視される。
5	小数点	1、2	オプション	小数点以下の桁数。固定長形式の場合のみ有効。CSV 形式の場合は無視される。
6	JEDICOS-XML との対応	図7を参考	必須	値を格納する JEDICOS-XML の場所を XPath で指定する。※3格納しない場合は”N/A”を記入する。
7	項目名と JEDICOS-XML との対応	図7を参考	オプション	JEDICOS-XML の場所を XPath で指定する。※3格納しない場合は空白もしくは”N/A”を記入する。

※3　指定方法については 3.3.3 マッピングの定義にて詳細に説明する。

種別に関しては以下のように定義している。

表 3.7 種別一覧

#	種別	JEDICOS-XML の型	揃え	パディング	説明
1	文字	String,	左揃え	半角スペース	文字を表す。
2	数字	Integer, Float, Decimal	右揃え	ゼロ(0)	数字を表す。
3	日付	Date	左揃え	半角スペース	日付を表す。 日付の形式は 6 桁の場合は”YYMMDD”形式、8 桁の場合は”YYYYMMDD”形式。
4	予備	マッピングしない	―	半角スペース	ビジネスモジュールの処理の対象外。マッピングの対象としない。

なお､種別と JEDICOS-XML の主な型の変換ルールについては別紙 2 に示すようになる。

3.3.3 マッピングの定義

「マッピングシート」に業務データを JEDICOS-XML のどこに保管するのかという、マッピングを定義する。

簡単な例を挙げて説明する。項目名「売価」が「500」という値を持つデータをマッピングしたいケースを考える。「500」を JEDICOS-XML のどのノードの値として保存するかを指定する。簡単のため、JEDICOS-XML のスキーマが図３．８のようであったとする。

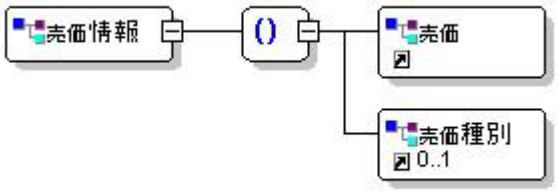


図 3.8　簡単な例

このとき「500」を<売価>ノードの値にマッピングしたいとき、つまり、

<売価情報>

　　<売価>500 </売価>

</売価情報>

のようにマッピングするにはマッピング表の「JEDICOS-XML との対応」に「売価情報/売価」のように記入すればよい。

このような位置の指定の方法は XML のタグを表現する標準である XPath をベース※4※5※6※7としている。XPath では、ルートのノードから階層的にノードとノードをスラッシュ(/)で結合した文字列で指定できる。JEDICOS-XML スキーマは付録の JEDICOS-XML スキーマ図を参考にしていただきたい。

※４ 繰り返し項目については、項目名の後、角カッコでくくって数字を記述する。例えば、 "伝票コメントリスト/伝票コメント[1]/…" のような記述となる。角カッコの中は 1 から始まる数字で連続していなければならない。

※５ 繰り返し項目のうち、１行読み込む毎に番号が増える項目については $denpyou というような＄+英文字でこれを表現することができる。例えば、発注伝票情報リスト/発注伝票情報[$denpyou]/…"のような表現となる。$denpyou は初期値１で、一行読み込む毎に 2,3,4 と１ずつ増える変数という意味である。梱包に対する$konpou、伝票に対する$denpyou、明細に対する$meisai を定義している。これらは以下の ToBpcMapper クラスの下記の API によって増やす、または、リセットするタイミングを決定する。

・isNewKonpou()

梱包明細情報を新規作成するとき true,既存に追加するとき false

・isNewDenpyou()

伝票情報を新規作成するとき true,既存に追加するとき false

・isNewMeisai()

伝票明細情報を新規作成するとき true,既存に追加するとき false

これらは下記のルールにしたがって変数の増加、リセットを行なう。

（上から順に評価して true になればそこで終わりである。）

1.isNewDocument()が true のとき

$konpou=1,$denpyou=1,$meisai=1

2.isNewKonpou()が true のとき

$konpou++,$denpyou=1,$meisai=1

3.isNewDenpyou()が true のとき

$denpyou++,$meisai=1

4.isNewMeisai()が true のとき

$meisai++

（$konpou=1 は$konpou の値を 1 にする。$konpou++は$konpou の値を 1 増加するという意味である。）

※６ XML の属性に項目をマッピングする場合は、”@”を用いた書式を用いる。例えば、”…/税率/@税区分” という項目名は、<税率 税区分="...">にマッピングされる。ただし、今回のマッピングツールでは、アットマークを省略した書式(XPath としては間違い)でも、解釈できるようになっている。

※７ XML の属性を持つノードにマッピングする場合は、”/”の後に content と記述しなければならない。※６の例で”.税率”に値を設定する場合には、マッピングには”…/税率/content”と記述する。

この Excel シートに値を入力する際に[Alt]+[↓]（Alt キーと下矢印キーを同時に押す）もしくは右クリックして[リストから選択]を選ぶことにより下記のように一覧から選択することができるようにしている。（これは Excel のオートコンプリート機能を利用している。オートコンプリート機能では、既に入力している文字列を含む文字列の候補の一覧がリスト表示されるのでその中から選択することができる。表示されるリストは JEDICOS-XML として正しいことを保障しているわけではなく可能性のある候補がすべて表示され

る。JEDICOS-XML として正しいかどうかは X-PATH 一覧シートか JEDICOS-XML の XML スキーマ定義を確認する必要がある）

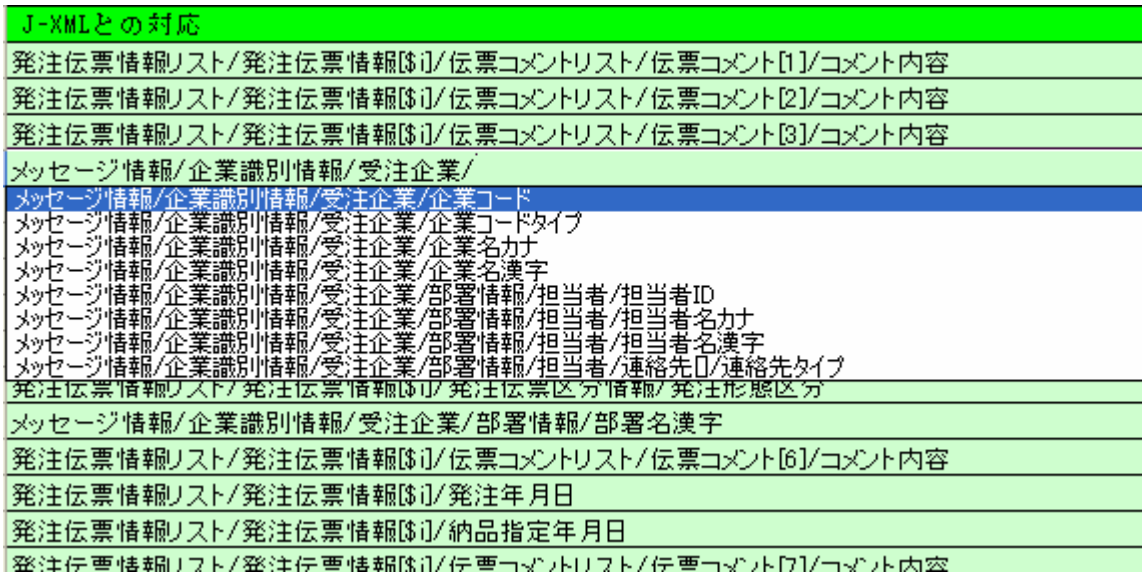


図 3.9　JEDICOS-XML との対応の入力（オートコンプリート機能による入力補助）

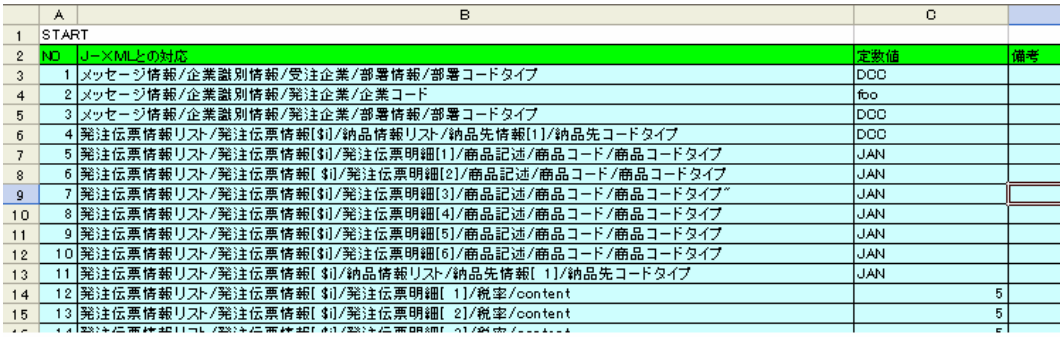
3.3.4 定数シートの記入

「定数シート」に業務データに含まれない定数の JEDICOS-XML マッピングを定義する。例えば、発注企業コードは業務データには含まれていないが JEDICOS-XML では必須項目となっている。このような固定の値を設定したい場合にこのシートを使用する。定数シート名は”マッピングシート名”＋”定数”とする。例えば、「Order 定数」というシート名で定義する。マッピングの記入の仕方は「マッピングシート」の JEDICOS-XML とのマッピングの記入の仕方と同様である。

表 3.10 定数シートの項目

#	設定項目	説明
1	JEDICOS-XML との対応	定数値を格納する JEDICOS-XML の場所を XPath で指定する。
2	定数値	定数値を記入する。
3	備考	システムとしては利用しない。

図３．１１に定数シートの例を示す。



	A	B	C	
1	START			
2	NO	J-XMLとの対応	定数値	備考
3	1	メッセージ情報/企業識別情報/受注企業/部署情報/部署コードタイプ	DOO	
4	2	メッセージ情報/企業識別情報/発注企業/企業コード	foo	
5	3	メッセージ情報/企業識別情報/発注企業/部署情報/部署コードタイプ	DOO	
6	4	発注伝票情報リスト/発注伝票情報[$i]/納品情報リスト/納品先情報[1]/納品先コードタイプ	DOO	
7	5	発注伝票情報リスト/発注伝票情報[$i]/発注伝票明細[1]/商品記述/商品コード/商品コードタイプ	JAN	
8	6	発注伝票情報リスト/発注伝票情報[$i]/発注伝票明細[2]/商品記述/商品コード/商品コードタイプ	JAN	
9	7	発注伝票情報リスト/発注伝票情報[$i]/発注伝票明細[3]/商品記述/商品コード/商品コードタイプ"	JAN	
10	8	発注伝票情報リスト/発注伝票情報[$i]/発注伝票明細[4]/商品記述/商品コード/商品コードタイプ	JAN	
11	9	発注伝票情報リスト/発注伝票情報[$i]/発注伝票明細[5]/商品記述/商品コード/商品コードタイプ	JAN	
12	10	発注伝票情報リスト/発注伝票情報[$i]/発注伝票明細[6]/商品記述/商品コード/商品コードタイプ	JAN	
13	11	発注伝票情報リスト/発注伝票情報[$i]/納品情報リスト/納品先情報[1]/納品先コードタイプ	JAN	
14	12	発注伝票情報リスト/発注伝票情報[$i]/発注伝票明細[1]/税率/content	5	
15	13	発注伝票情報リスト/発注伝票情報[$i]/発注伝票明細[2]/税率/content	5	

図 3.11 定数シート

3.3.5 設定シートの記入

「設定」シートには自動生成するファイルに関するプロパティを設定する。これらのプロパティは自動生成する際に参照される。設定できる値は英数字のみとする。

表 3.12 設定シートの項目

#	設定項目	例	説明
1	パッケージ名	jp.co.foo.bm.adapter	自動生成するクラスのパッケージ名
2	業務データの形式	FIX	固定長データ(=FIX),CSV データ(=CSV)
3	Javaソースファイルの出力先	src	相対ディレクトリ名(デフォルトはこの Excel ファイルと同じディレクトリ。ディレクトリの区切りは¥)
4	フィールド定義ファイルの出力先	csv	相対ディレクトリ名(デフォルトはこの Excel ファイルと同じディレクトリ。ディレクトリの区切りは¥)
5	Javaソースファイルの出力方針	DIR	クラスファイルをパッケージ毎にディレクトリ下に作成する(=DIR)か、フラットなディレクトリに作成するか(=FLAT)

3.3.6 ファイルの自動生成

マッピング表が完成したら、最後にファイルを自動生成する。自動生成されるファイルには①フィールド定義ファイルと②Java ソースファイルの２種類がある。それぞれマッピング表の上部にあるボタン（図３．１３）を押すことで Excel シートのフォルダ下に「設定」シートで指定したディレクトリにファイルが作成される。

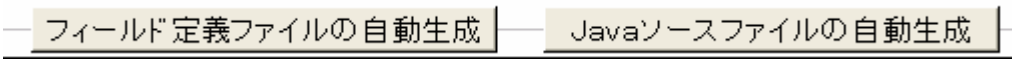


図 3.１３ 自動生成ボタン

①フィールド定義ファイル

フィールド定義ファイルは、マッピングに関する情報を保持するファイルである。自動生成されたソースファイルからから参照されるので変更してはならない。フィールド定義ファイルはシート名+”.csv”の名前で指定の場所に生成される。

②Java ソースファイル

送信アダプタ、受信アダプタのクラスのソースコードが自動生成される。クラス名はシート名を接頭辞として利用する。例えば、シート名が「Foo」という名前であった場合、図３．１４のようなファイルが指定の場所に自動生成される。（シート名は「企業名」＋「メッセージ名」などにしていただきたい）

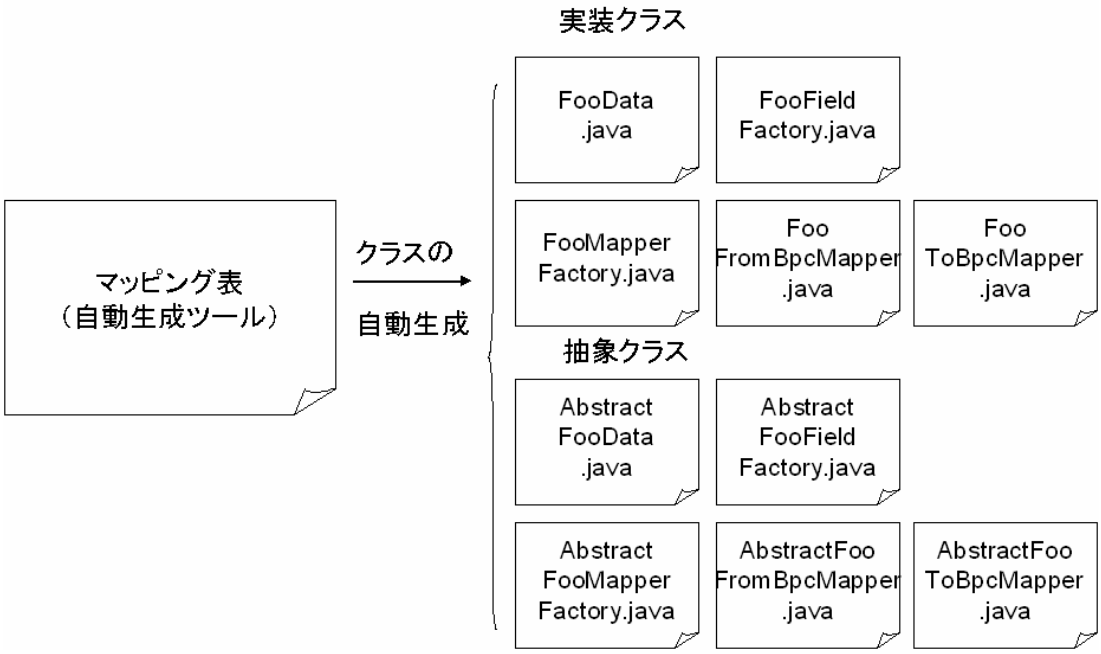


図 3.１４ １つのシートで自動生成されるクラス(シート名が Foo の場合)

自動生成される Java プログラムには実装クラスと抽象クラスがある。抽象クラスには”Abstract”という接頭辞がついている。個別のカスタマイズを行なうには実装クラスにロジックを追加する。実装クラスは同じフォルダに同じ名前のクラスがあれば自動生成ツールで新たにファイルを生成しない。抽象

クラスは自動生成時に必ず上書きされる。各クラスの説明を表３．１５に示す。

表 3.15 自動生成されるクラス（シート名が Foo の場合）

#	自動生成されるクラス名	説明
1	FooData	業務データの１行のデータを表すクラスである。FooFromBpcMapper、FooToBpcMapper が業務データをフィールド毎に読み書きする際に使用する。
2	FooFieldFactory	自動生成ツールで作成されたフィールド定義ファイルを読み業務データのフィールド情報を保持するクラスである。
3	FooMapperFacotry	FooToBpcMapper および FooFromBpcMapper クラスのインスタンスを作成するクラスである。
4	FooFromBpcMapper	業務オブジェクトから業務データへ変換を行いファイルへ書き出すクラスである。
5	FooToBpcMapper	ファイルから業務データを読み込み、業務オブジェクトへ変換を行なうクラスである。
6	AbstractFooData	FooData クラスの抽象クラスである。
7	AbstractFooFieldFactory	FooFieldFactory クラスの抽象クラスである。
8	AbstractFooMapperFacotry	FooMapperFacotry クラスの抽象クラスである。
9	AbstractFooFromBpcMapper	FooFromBpcMapper クラスの抽象クラスである。
10	AbstractFooToBpcMapper	FooToBpcMapper ラスの抽象クラスである。

3.4 クラス図、シーケンス図

3.4.1 送信アダプタのクラス図

送信アダプタは、業務ＡＰから呼び出されてビジネスモジュールに通知するまでの処理を行う。送信アダプタクラスは図３．１６に示すようにAbstractReceive 抽象クラスを継承して実装する。図３．１６のjp.go.meti.dscm.mapper.sample はマッピングツールで自動生成されるクラスの一例である。送信アダプタクラスは LoadData クラスを使って業務データを読み込み、それを BM に送信する。送信アダプタクラスはリファレンス実装を参考に実装していただきたい。送信サーブレットから EJB を経由してこのクラスを呼び出す。

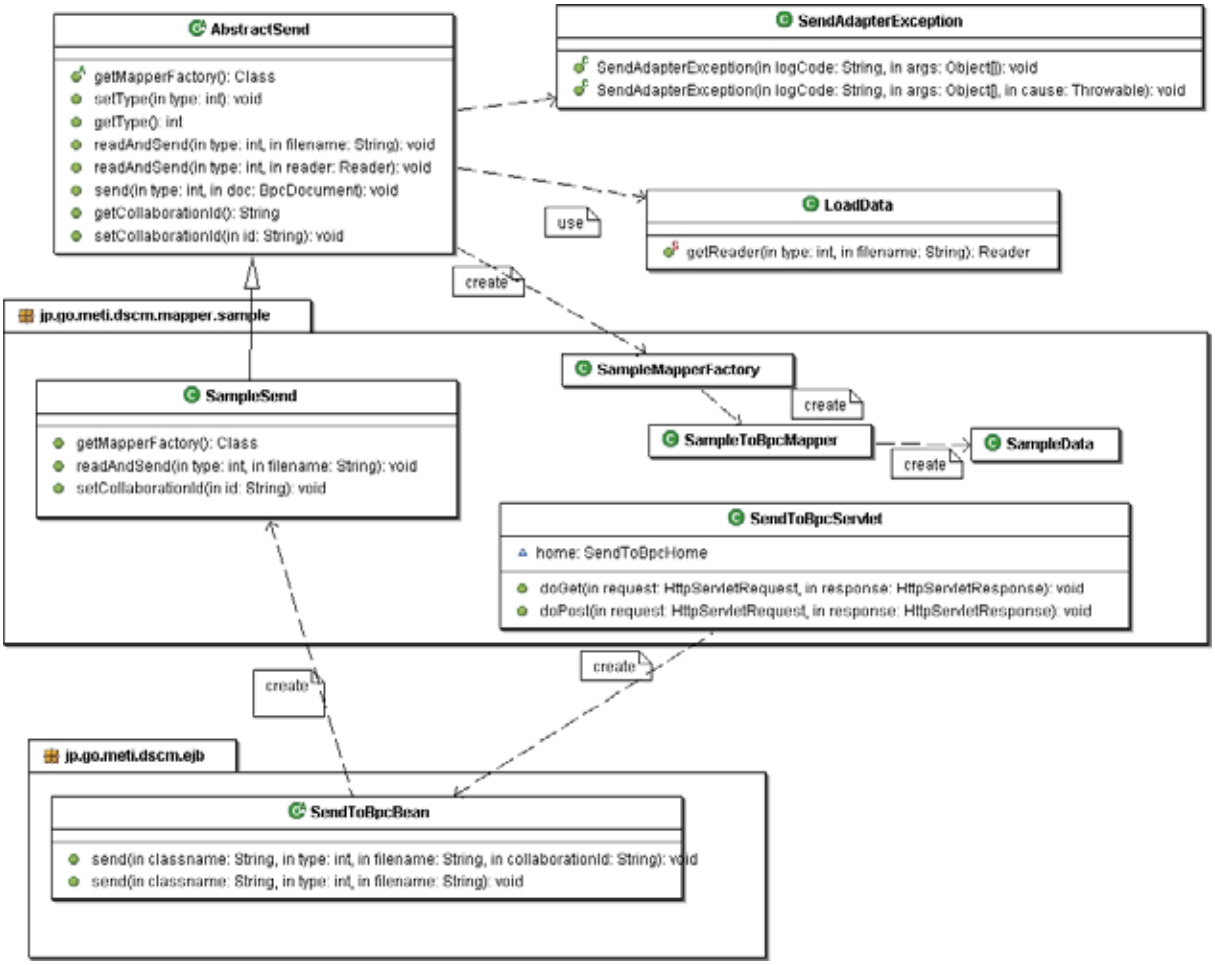


図 3.16 送信アダプタのクラス図(発注)

3.4.2 送信アダプタのシーケンス図

送信アダプタのシーケンス図を以下に示す。大まかな流れを把握するため、詳細なクラスは省略している。この中でマッピングツールから自動生成され

るのはSampleToBpcMapperクラス(ただし､シート名によって名前は変わる)である。SampleSend は AbstractSend クラスのサブクラスとして作成しなければならない。SendToBpcServlet から SendToBpcBean の send メソッドを呼び出せば下記のような流れで処理が行われる。send メソッドは１つのトランザクションとして実行され、途中で例外が発生した場合にはすべての処理がロールバックされる。

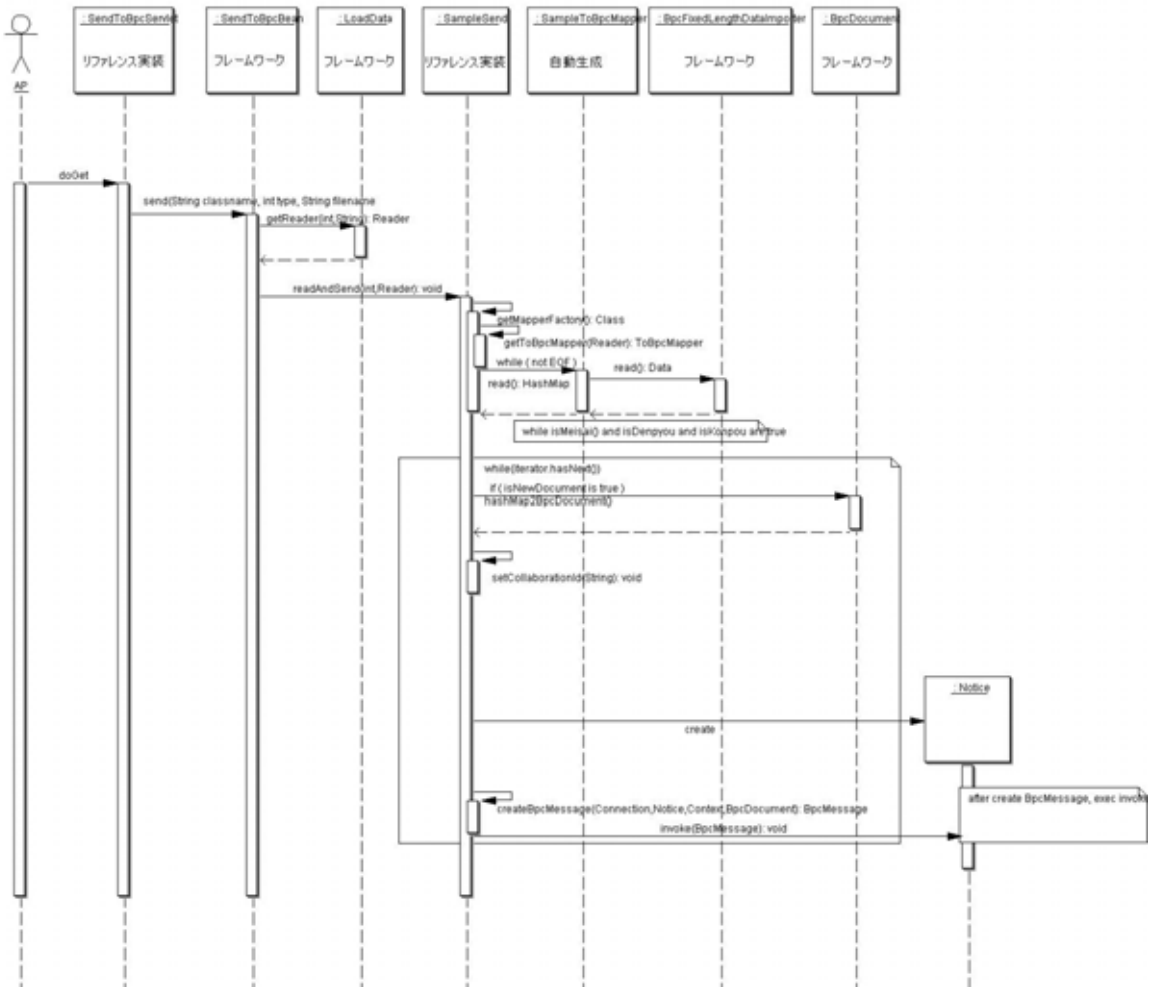


図 3.17 送信アダプタのシーケンス図

3.4.3 送信サーブレットについて

送信アダプタを起動する送信サーブレットはサンプルプログラムの SendToBpcServlet サーブレットを参考にして作成する｡送信サーブレットの HTTP の引数は表３．１８のようになる。

実証実験ではコラボレーション ID は次のルールで順に決定する。

①Servlet から指定されたコラボレーション ID を使う

Servlet の引数でコラボレーション ID を与えた場合、これを使用する。(自由記述のメッセージや②～③のルールで決定できないような場合にはコラボレーション ID を Servlet の引数で必ず与えなければならない)

②業務データから取得する

①でコラボレーション ID を与えられなかった場合、メッセージの内容からコラボレーションを決定する。

・ 請求、支払案内のメッセージの場合

“partyA_”＋”メッセージ情報/企業識別情報/支払企業/企業コード”をキーとする PARTY を表す文字列+” _PAYMENT

・ 発注メッセージの場合

“partyA _”＋”メッセージ情報/企業識別情報/受注企業/企業コード”をキーとする PARTY を表す文字列+”_ORDER”

・ ASN、その他のメッセージの場合

③過去の発注情報を検索して、送信しようとしているメッセージと、伝票番号、明細番号、発注年月日が一致するものと同じコラボレーション ID を使用する。③でなければ②の発注メッセージの場合と同じロジックによりコラボレーションを決定する。

・ なお、企業コードのキーと値は CollaborationInfo というクラスに登録する。

表 3.18 SendToBpcServlet サーブレットの引数

引数名	説明	必須
Type	プロセスを表す文字列を指定する。 Order,OrderResponse,ShippingInstruction, ShippingNotice,ArrivalNotice, ReceiptAdvice, Invoice, RemittanceAdvice,FreeFormat など。	○
Filename	ファイル名（拡張子含む)。ファイル名にパス情報が含まれている場合、パス上のファイルを読み込む。それ以外は bpc_adapter.properties に設定されているディレクトリから読み込む。	○
collaborationid	コラボレーション ID	×

3.4.4 送信 EJB について

送信サーブレットから呼び出される EJB（SendToBpcBean）は StatelessBean で以下の２つのメソッドを持つ。自由記述のメッセージなどでコラボレーション ID を外部から指定する場合には２つめのメソッドを利用する。EJB はトランザクション管理を行なうために利用している。

public void send(String classname, **int** type, String filename)

public void send(String classname, **int** type, String filename, String collaborationId)

3.4.5 受信アダプタのクラス図

受信アダプタは、ビジネスモジュールから呼び出されて業務ＡＰに通知するまでの処理を行う。受信アダプタクラスもリファレンス実装を提供しているのでそれを参考に実装する。図３．１９のように AbstractReceive 抽象クラスを継承して作成する。なお、受信アダプタクラスはビジネスモジュールがメッセージを受信した際に呼び出すことができるようにあらかじめ、定義ファイル(event.xml)に登録しておかなければならない。定義ファイルの書き方などはビジネスモジュールの説明書を参照のこと。

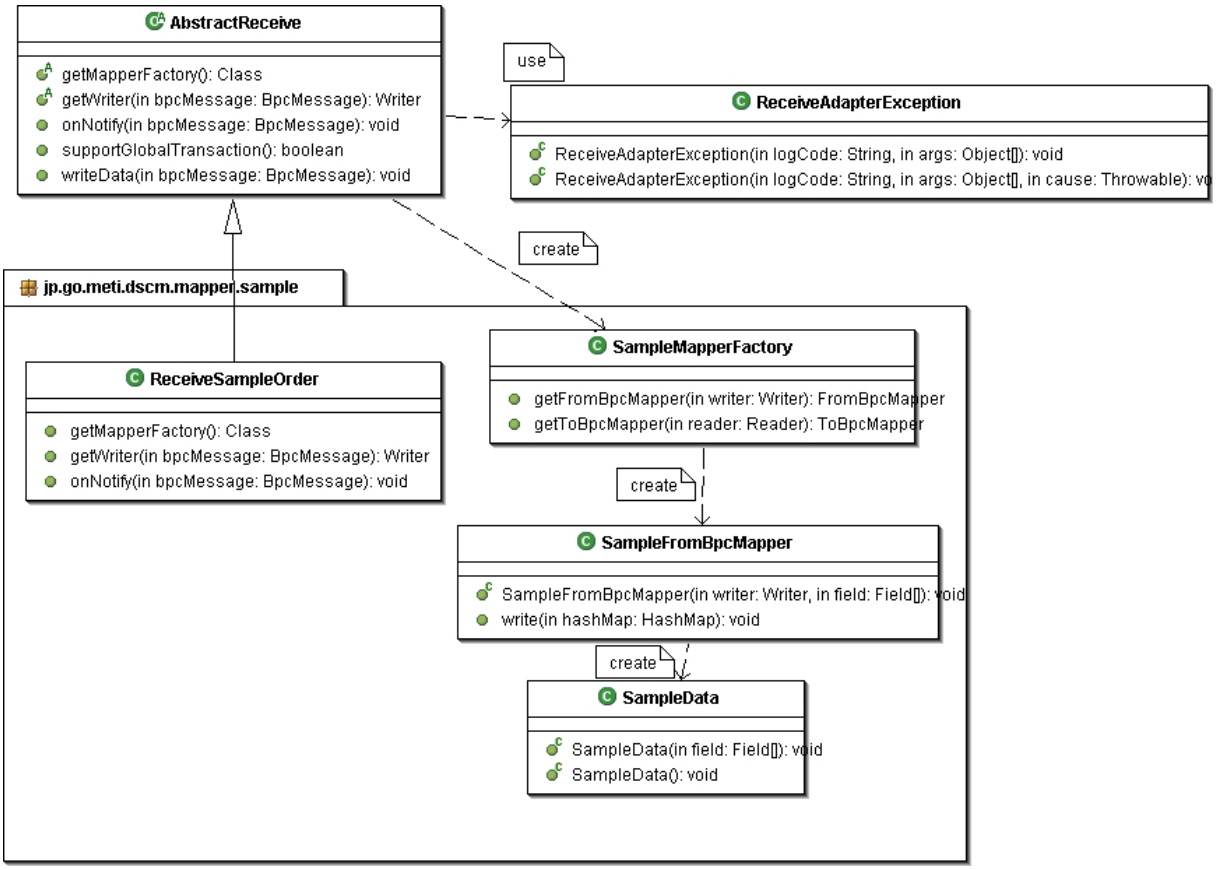


図 3.19 受信アダプタのクラス図(発注)

3.4.6 受信アダプタのシーケンス図

受信アダプタのシーケンス図を以下に示す。大まかな流れを把握するため、詳細なクラスは省略している。NoticeOrderActivity は BM のクラスであり、メッセージを受信すると呼び出される。ReceiveSampleOrder は AbstractReceive のサブクラスとして作成しなければならないクラスである。AbstractReceive の writeData メソッドを呼び出すと以下のような流れで処理が行われる。

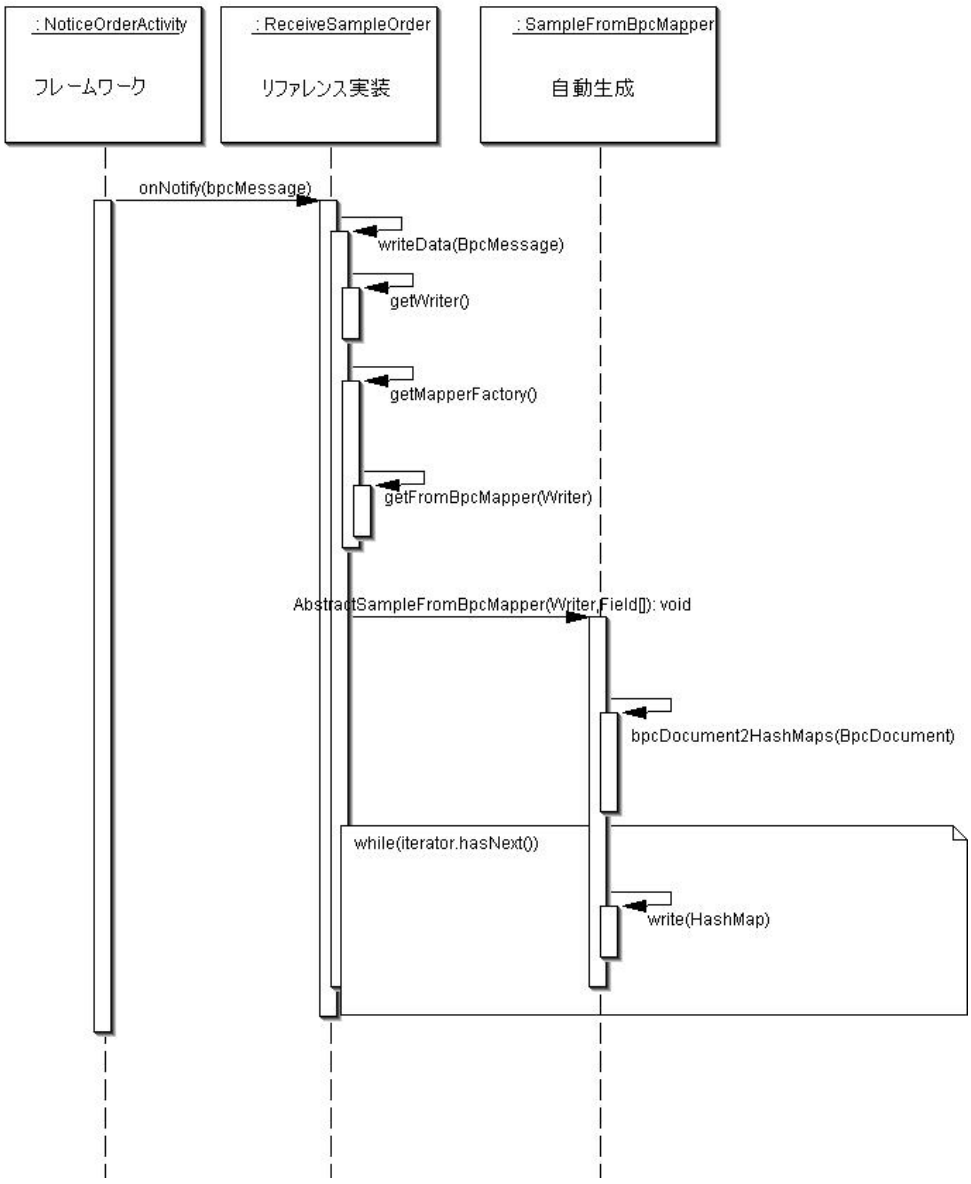


図 3.20 受信アダプタのシーケンス図

3.4.7 mapper パッケージのクラス図

jp.go.meti.dscm.mapper パッケージではマッピング機能を提供する。mapper パッケージの主要なクラス図を示す。これらのクラスはマッピングツールで自動生成されるクラスのスーパークラスである。これらのクラスは直接インスタンス化しないで自動生成されたサブクラスから利用する。

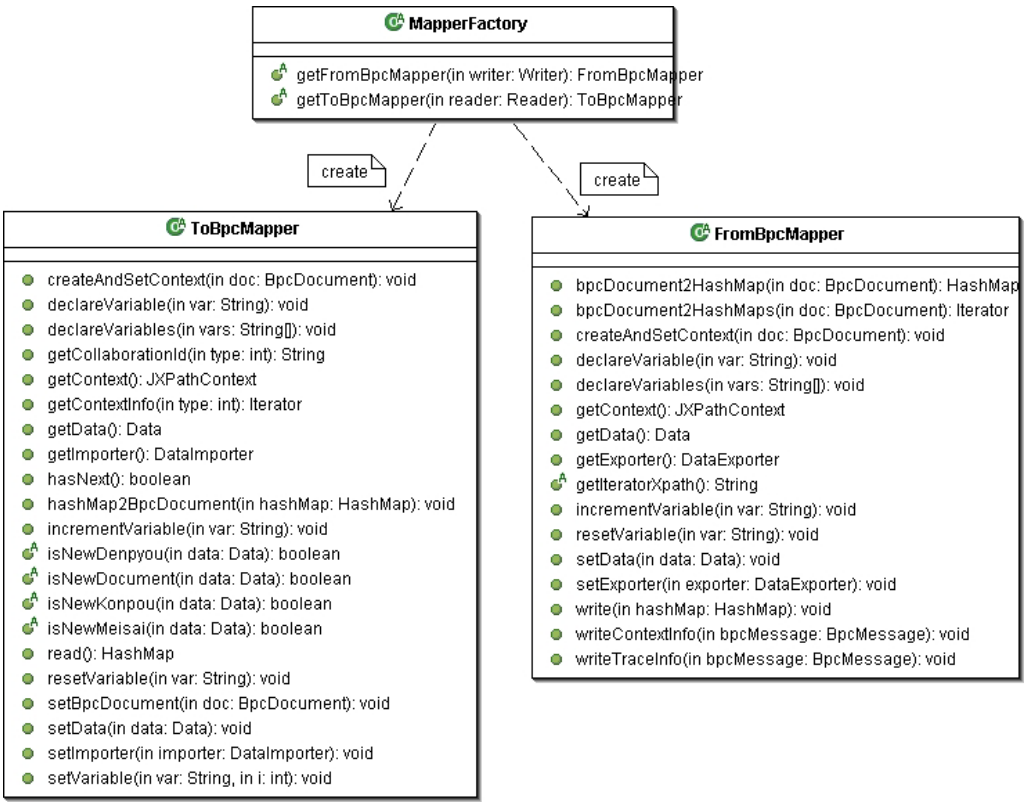


図 3.21 mapper パッケージのクラス図

各クラスの役割は以下のようになる。

MapperFactory クラス：ToBpcMapper オブジェクト、FromBpcMapper オブジェクトを作成する。

ToBpcMapper クラス： HashMap から BpcDocument を作成する。

FromBpcMapper クラス：BpcDocument から HashMap を作成する。

通常、オーバーライドしなければならないメソッド以下の通りである。

ToBpcMapper クラス:isNewKonpou,isNewDenpyou,isNewMeisai,
isNewDocument

FromBpcMapper クラス: setData

3.4.8 field パッケージのクラス図

jp.go.meti.dscm.mapper.field パッケージではフィールド定義に関する機能を提供する。field パッケージの主要なクラス図を示す。業務 AP アダプタが直接利用する可能性のあるクラスは Data クラスのみである。その他のクラスはフレームワークが使用する。

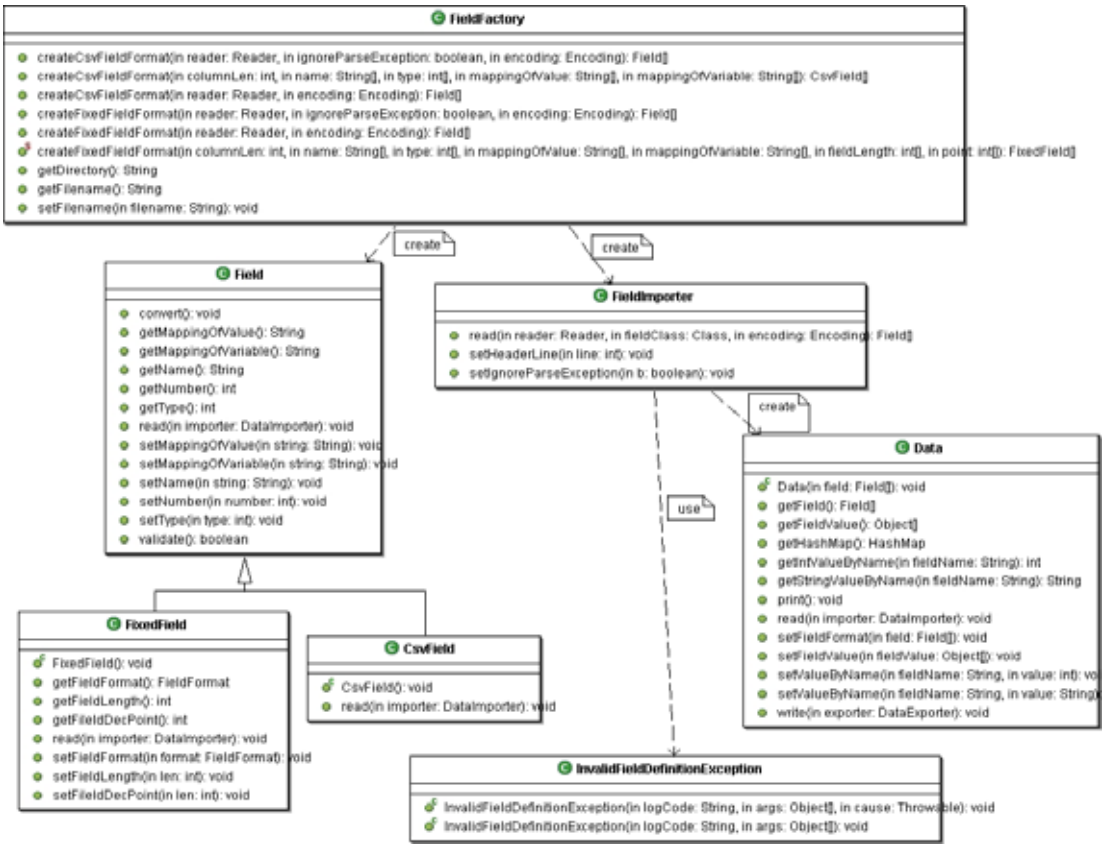


図 3.22 field パッケージのクラス図

各クラスの役割は以下のようになる。

FieldFactory クラス：Field オブジェクト（の配列）を作成する。

Field クラス： フィールド定義を表す。１カラムと対応する。

CsvField クラス：CSV のフィールド定義を表す。Field のサブクラス。

FixedField クラス：固定長のフィールド定義を表す。Field のサブクラス。

FieldImporter クラス：ファイルからフィールド定義を読み出す。

Data クラス：フィールド定義や業務データを読み書きする際のデータを表す。１つのインスタンスがデータファイルの１行を表す。

InvalidFieldDefinitionException:フィールド定義が誤りの場合の例外。

３.４.９ io パッケージクラス図

jp.go.meti.dscm.mapper.io パッケージでは CSV/固定長ファイルの I/O 機能を提供する。通常、これらのクラスはフレームワークが使用するのみで、業務 AP アダプタは直接利用することはない。io パッケージの主要なクラス図を示す。

各クラスの役割は以下のようになる。

DataExportable インタフェース：Data（書き込みデータを表す）クラスのインタフェース。

DataExporter インタフェース： Data を書き込むインタフェース。

TextDataExporter クラス：テキストフォーマットの Data を書き込む。

BpcFiexedLengthDataExporter クラス：固定長の Data を書き込む。

CSVDataExporter クラス：CSV の Data を書き込む。

DataImportable インタフェース：Data（読み込みデータを表す）クラスのインタフェース。

DataImporter インタフェース： Data を読み込むインタフェース。

TextDataImporter クラス：テキストフォーマットの Data を読み込む。

BpcFiexedLengthDataImporter クラス：固定長の Data を読み込む。

CSVDataImporter クラス：CSV の Data を読み込む。

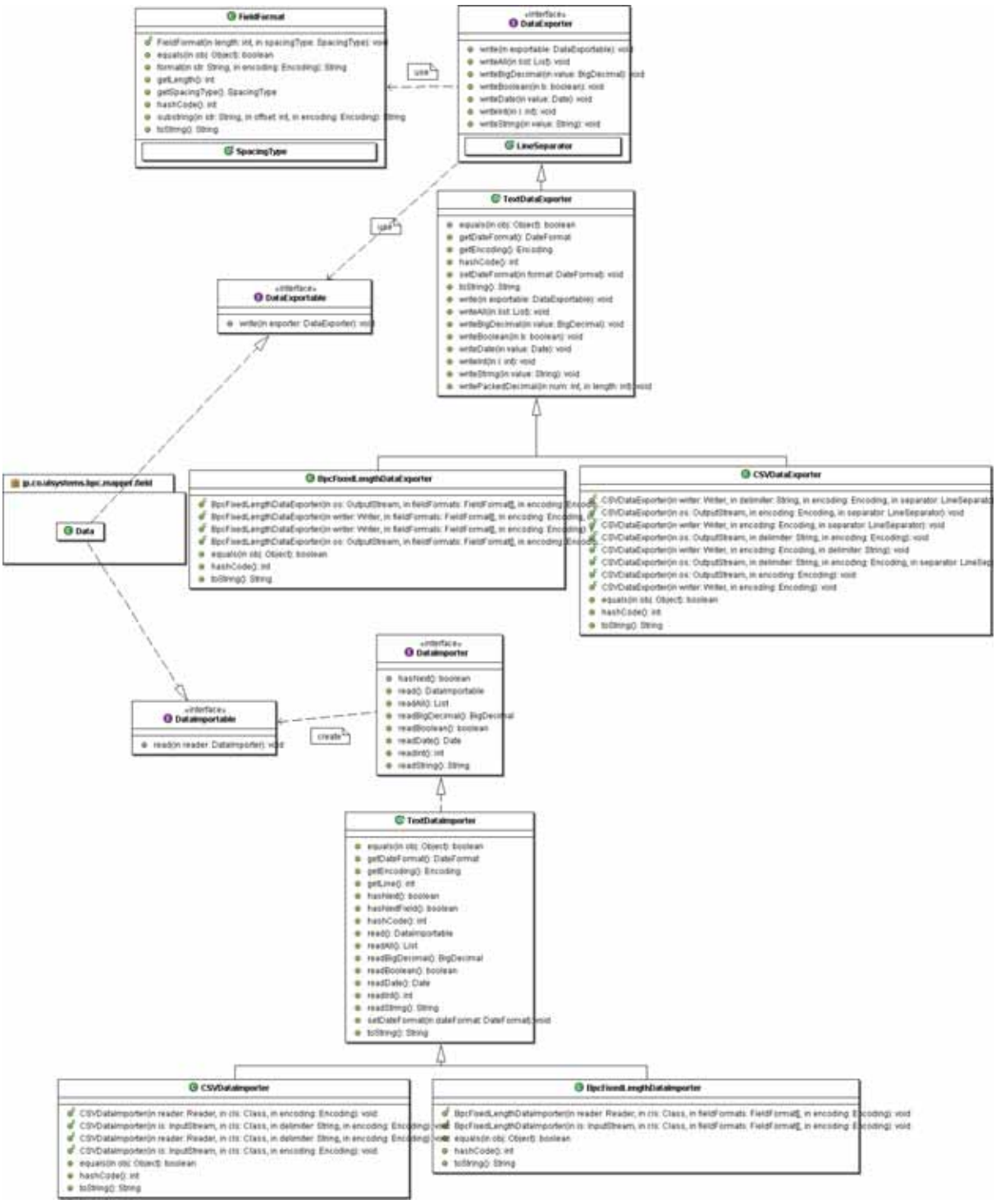


図 3.2.3 io パッケージのクラス図

3.5 個別カスタマイズ実装

3.5.1 自動生成クラスのカスタマイズ

表３の自動生成されたクラスのうち、必要に応じて実装クラスにロジックを追加しなければならない。いくつか例をあげて説明する。

3.5.1.1 ケース１

読み込んだ業務データに含まれていない値を JEDICOS-XML の項目にマッピングして値を設定したい場合がある。FooToBpcMapper の read メソッドは、ファイルから読み込んだ FooData オブジェクトから HashMap を作成するメソッドである。これをオーバーライドしてマッピングを追加する。これを実装したプログラムを示す。なお、マッピングツールの定数シートに記述しても同様の結果となる。（この場合は自動生成された AbstractToBpcMapper に下記のようなコードが埋め込まれる）

```
/** フィールド定義にしたがって1行のデータを読み込み、
  * HashMapとして返します。読み込んだデータがヘッダ情報の
  * 場合には次の１行も読み込み、２行で１つのHashMapを作ります。
  * HashMapのキーはJEDICOS-XMLとのマッピングを表すXPathのStringです。
  * HashMapの値はJEDICOS-XMLとのマッピングに対応する値です。
  */
public HashMap read() throws InvalidFieldException, IOException,ParseException{
        //ハッシュマップの取得
        HashMap hashMap = super.read();
        //マッピングの追加
        hashMap.put("メッセージ情報/企業識別情報/受注企業/部署情報/部署コード", "1");
        hashMap.put("メッセージ情報/企業識別情報/受注企業/部署情報/部署コードタイプ
", "DCC");
        return hashMap;
}
```

このケースではマッピングの値として固定値を設定しているが、この応用として業務データの値を条件にマッピングを追加することもできる。

3.5.1.2 ケース2

業務データの取引先コードによってメッセージを分ける場合を考える。

FooToBpcMapper クラスは送信サーブレットから起動された、ビジネスモジュールから呼び出される。FooToBpcMapper クラスは、既存の業務オブジェクトに追加するか、新規に業務オブジェクトを作成するかを判断するために isNewDocument()というメソッドを実装する。(業務オブジェクトはメッセージを作成する単位と同じである。)

読み込んだ業務データの取引先コードが同じ場合には既存の業務オブジェクトに追加したいので、isNewDocument()は false を返すように実装する。また、取引先コードが異なった場合には isNewDocument()は true を返すように実装する。プログラムは以下のようになる。

```
/** 新規のDocumentとするかどうかを返します。
  * 前行と現在の行でBpcDocumentを分けなければならない場合
  * （例えば、複数の送信先の伝票が含まれているような場合）、
  * にtrueを設定します。
  */
public boolean isNewDocument(Data data){
        //業務データの取引先コードの値を取得する
        String currentCode = data.getStringValueByName("取引先コード");
        if (lastCode.equals(currentCode)){//前の行と同じであった場合
                return false;
        } else {//取引先コードが直前に読み込んだものと異なった場合
                lastCode = currentCode;
                return true;
        }
}

//直前に読み込んだ取引先コード
private String lastCode = "";
```

3.5.1.3 ケース3

ビジネスモジュールから取ったメッセージに明細が複数含まれる場合に明細毎に複数行に分けて出力したい場合がある。

AbstractFooFromBpcMapper クラスのデフォルトの動作では明細が複数含まれていても１行に出力される。これを明細毎に複数行に分けて出力するには FooFromBpcMapper クラスの write メソッドにロジックを追加する。

次に示す例は、この write メソッドをオーバーライドして明細行毎に業務データを書き込むようにロジックを追加した例である。

```
/**
 * ハッシュマップで表された１行のデータをフィールドのフォーマット
 * にしたがって書き込みます。
 */
public void write(HashMap hashMap) throws IOException{
        //Dataの取得
        Data data = super.getData();
        //Dataに設定する値の作成
        Object [] value = new Object[data.getField().length];
        //ハッシュマップからvalue値を設定する
        for (int i = 0; i < data.getField().length; i++) {
                String mapping = data.getField()[i].getMappingOfValue();
                if(hashMap.containsKey(mapping)) {
                        value[i] = hashMap.get(mapping);
                } else {
                        value[i] = "";
                }
        }
        //ここまではsuperクラスで行なわれている処理と同じ
        int line = 1;
        String str = "発注伝票情報リスト/発注伝票情報[$denpyou]/発注伝票明細[";
        //ハッシュマップに複数の明細が含まれていたら明細毎に１行書き出す。
        while (hashMap.containsKey(str + line + "]/商品記述/商品コード")){
                value[55] = hashMap.get(str + line + "]/発注伝票行番号");
                value[58] = hashMap.get(str + line + "]/発注数量情報/発注数量");
                data.setFieldValue(value);
                super.getExporter().write(data)
                line++;
        }
}
```

3.5.1.4 ケース４

受信アダプタや送信アダプタで**ログを出力したい場合**、BM のログ出力機能を利用することができる。BM のログ出力は Apache Commons Logging を利用しており、BM 上で動作する場合には BM のプロパティファイルで指定したファイルに決められたフォーマットでログを出力する。後述のテストツールから使用する場合には J2SE のロギング機能を用いて標準出力に出力される。

BM では jp.go.meti.dscm.log.BPCLog クラスを利用してログ出力を行なう。BPCLog クラスの logMessage メソッドは引数にログコードと埋め込みオブジェクトの配列を取る。ログコードとは、LogCodeConst クラスで定義されている定数文字列で大括弧{}で囲まれた文字列に埋め込みオブジェクトの配列を toString()で変換した文字列を順に埋め込み、ログ出力文字列とする。個別実装のログ出力には LogCodeConst クラスの汎用目的のログコードを利用することができる。ログ出力のフォーマットについては BM 内ではプロパティファイルで指定された場所の log4j.xml によって定義されている。下記に FooFromBpcMapper クラスでログ出力する例を示す。

```
public void write(HashMap hashMap) throws IOException{

        //BPCLog(singleton)のインスタンスを取得します。
        BPCLog log = BPCLog.getInstance();
        Object [] message1 = {"受信アダプタがデータを受け付けました." };
        //ログ出力１
        log.message(this.getClass(), LogCodeConst.INFO_GENERAL_PURPOSE, message1);
        super.write(hashMap);
        Object [] message2 = {"受信アダプタがデータを書き込みました." };
        //ログ出力２
        log.message(this.getClass(), LogCodeConst.INFO_GENERAL_PURPOSE, message2);

   }
```

BM 内でのログ出力は以下のようになる。（テストツールでは若干フォーマットが異なる。）

2004/09/22 10:32:00 I [COMMON-0009] 受信アダプタがデータを受け付けました. at jp.go.meti.dscm.adapter.aeon.AeonOrderToBpcMapper
2004/09/22 10:32:00 I [COMMON-0009] 受信アダプタがデータを書き込みました. at jp.go.meti.dscm.adapter.aeon.AeonOrderToBpcMapper

なお、汎用目的のログコードには以下のものがある。

表 3.2４ 汎用目的のログコード

ログコードの定数文字列	説明
DEBUG_GENERAL_PURPOSE	汎用デバッグメッセージをログに出力する。デバッグメッセージとはシステムが想定どおり処理を行なっているか確認する為のログを示す。想定済みのエラーが発生した場合やロジックの計算結果などが対象。運用者の対処は不要であり安定稼動後は、出力しない。
INFO_GENERAL_PURPOSE	汎用情報メッセージをログに出力する。設定ファイルを読み込んだときの設定情報、システム開始／停止時の状態出力など処理のトレースや確認の為のログを示す。
WARN_GENERAL_PURPOSE	汎用警告メッセージをログに出力する。警告メッセージは想定済みのエラーが発生した場合に出力される。基本的に運用者による対処は不要なメッセージである。
ERROR_GENERAL_PURPOSE	汎用エラーメッセージをログに出力する。エラーメッセージはシステムが運用を継続できる範囲でのエラーが発生した場合に出力される。運用者による対処が必要な場合がある。 業務的な対処が必要である。
FATAL_GENERAL_PURPOSE	汎用重大エラーメッセージをログに出力する。重大エラーメッセージはシステムがリカバリ不可能な事象が発生した場合に出力される（DB サーバ停止、外部システムの停止、想定外のエラーが発生した場合等） 。このログが出力された場合は、運用者による対処が必要である。

3.5.1.5 ケース５

受信アダプタや送信アダプタで**例外を発生させて処理をロールバックさせたい場合**、決まった Exception を投げなければならない。送信アダプタでは SendAdapterException を、受信アダプタでは ReceiveAdapterException を作成してスローする。BM では例外クラスを作成する際に、引数にケース４で説明したログコードを設定しなければならない。クラスでログ出力する例を示す。

```
/**
 * 発注予定通知を受信した場合のコールバックメソッド
 *
 * @param bpcMessage BpcMessage
 */
public void onNotify(BpcMessage bpcMessage) throws ReceiveAdapterException{

        super.onNotify(bpcMessage);
        //ここに通知を受けたときの処理を記述します。
        File dir = new File("c:¥¥bpc");
        try {
            //バッチ処理を起動する
            Process ps = Runtime.getRuntime().exec("copy.bat", null, dir);
            ps.waitFor();
        } catch (Exception e) {
            String msgs [] = {"Command can't execute."};
            //エラーの送出
            ReceiveAdapterException re =
                  new ReceiveAdapterException(LogCodeConst. ERROR_GENERAL_PURPOSE,
                        msgs, e);
            //ログ出力もしたい場合には下記を記述
            re.logMessage(this.getClass());
            throw re;
        }
}
```

上記のように例外を発生させると BM からの受信処理がロールバックされる。処理をロールバックさせないためには BM に例外を渡さないようにする。(上記の例外以外でもアダプタで Runtime 例外が発生するとロールバックするので注意が必要である。)

3.5.1.6 ケース6

送信アダプタで送信する直前の JEDICOS-XML のメッセージを修正したい場合、FooToBpcMapper の beforeSend メソッドで変更することができる。下記の例では１伝票に６明細固定の業務データを読み込む場合に６明細未満の業務データがあった場合に不要な明細データを削除する処理を示す。

```
/**
 * BpcDocumentに対して送る前に処理をします。
 * @param type メッセージ種別
 */
public void beforeSend(JXPathContext context) {
        //不要なノードを削除する処理を記述します。
        Iterator iterator =
            context.iteratePointers("発注伝票情報リスト/発注伝票情報/発注伝票明細");
        while(iterator.hasNext()) {
            Pointer pointer = (Pointer)iterator.next();
            String asPath = pointer.asPath();
            try {
                BigDecimal orderQuantity =
                    (BigDecimal)context.getValue(asPath + "/発注数量情報/発注数量");
                if (orderQuantity == null) {
                    context.removePath(asPath);
                }
            } catch (JXPathException e) {//発注数量情報も存在しない場合
                //何もしない
            }
        }
}
```

上記のプログラムではApache Commons JXPath ライブラリの機能を利用して JEDICOS-XML のノードを削除している。

3.5.1.7 ケース7

送信アダプタで読み込んだ HashMap を修正したい場合、FooToBpcMapper の read メソッドで変更することができる。下記の例では文字として読み込んだデータを加工して JEDICOS-XML の Date で定義されているマッピング先に追加する例と読み込み処理で解析できなかった場合にデフォルト値を設定する例を示す。

```
/**
  * フィールド定義にしたがって1行のデータを読み込み、
  * HashMapとして返します。読み込んだデータがヘッダ情報の
  * 場合には次の１行も読み込み、２行で１つのHashMapを作ります。
  * HashMapのキーはJEDICOS-XMLとのマッピングを表すXPathのStringです。
  * HashMapの値はJEDICOS-XMLとのマッピングに対応する値です。
  *
  */
public HashMap read() throws InvalidFieldException, IOException,ParseException{
      HashMap hashMap = super.read();
      //日付を表す「文字」を基に「日付」型で定義しなおす場合のコードを示します。
      String dateString =
          (String)hashMap.get("発注伝票情報リスト/発注伝票情報[$denpyou]/発注年月日");
      java.util.Date date = new java.util.Date(); // 本来はdateStringを元にdateを作成
      hashMap.put("発注伝票情報リスト/発注伝票情報[$denpyou]/発注年月日", date);

      //読み込めなかった場合の処理
      BigDecimal num = (BigDecimal)hashMap.get("発注伝票情報リスト/発注伝票情報
[$denpyou]/発注伝票明細[6]/発注数量情報/発注単位入数");
      if (num == null) {//もし空白文字の場合には0とする
          hashMap.put("発注伝票情報リスト/発注伝票情報[$denpyou]/発注伝票明細[6]/発注
数量情報/発注単位入数", new BigDecimal(0));
       }
       return hashMap;
}
```

受信アダプタで読み込み時に「数値」「日付」型の場合に HashMap の値には BigDecimal、Date に変換されて格納されるが、その際にパースエラーの場合は値が null で格納されている。その場合の処理を上記で記述している。

3.6 テスト

マッピングに関するテストを行うツールを２つ提供する。これらのツールはバッチファイルから Java プログラムを起動する。起動前に自動生成ツールで生成されたプログラムをコンパイルしておく必要がある。また、環境変数 JAVA_HOME も設定する。動作環境の J2SE のバージョンは 1.4 以上を推奨する。

3.6.1 送信アダプタのテストツール

送信アダプタツールはコマンドラインで送信アダプタ機能をテストする。下記のように引数をつけて実行する。このツールは業務データからフィールド定義にしたがってデータを読み出し、JEDICOS-XML を作成する。

（書式）

% ToBpcMapperTest *mapperFactoryClass input output* [-silent]

[-type=[order|orderresponse|shippinginstruction|shippingnotice
|arrivalnotice|receiptadvice|accountreceivables|accountpayables
|remittanceadvice|invoice|inventoryreport|pricecatalogue|salesdatareport
|salespromotionplan|freeformat]

mapperFactoryCalss には jp.go.meti.dscm.mapper.FooMapperFactory のようにパッケージを含むクラス名を指定する。*input* には入力となる業務データファイルを指定する。*output* は JEDICOS-XML(XML 文字列)である。－type=<メッセージ種別>を指定すると指定のメッセージ種別の JEDICOS-XML を作成する。－type が指定されない場合ルートノードは <BpcDocument>となっている（別紙２参照）。マッピングが正しく JEDICOS-XML に反映されていることを確認する。－silent を指定するとデバッグメッセージを出さない設定となる。

3.6.2 受信アダプタのテストツール

受信アダプタツールはコマンドラインで受信アダプタ機能をテストする。

下記のように引数をつけて実行する。このツールは JEDICOS-XML から業務オブジェクトを作成し、フィールド定義にしたがって業務データを書き出す。

（書式）

% FromBpcMapperTest *mapperFactoryClass input output* [-silent]

mapperFactoryCalss には jp.go.meti.dscm.mapper.FooMapperFactory のようにパッケージを含むクラス名を指定する（.class は含まない）。*input* には入力となる JEDICOS-XML ファイルを指定する。このファイルの JEDICOS-XML は送信アダプタのテストツールで出力されたもの(ルートノードが BpcDocument となっている)とする。 *output* は業務 AP に渡すべき、業務データファイルである。業務データが意図したとおりに作成されることを確認する。－silent を指定するとデバッグメッセージを出さない設定となる。

3.6.3 テストツールのＦＡＱ

(1)送信アダプタのテストツールを実行して下記の ValidationException が表示された場合には JEDICOS-XML のマッピング定義が不足している。この例では、”発注伝票情報リスト/発注伝票情報/発注伝票明細/付帯情報リスト/付帯情報”というノードには”管理組織 ID”が必要だということがエラーメッセージからわかる。これを解決するには、マッピング表のマッピングシートか定数シートに “ 発注伝票情報リスト/発注伝票情報/発注伝票明細/付帯情報リスト/付帯情報/管理組織 ID”をマッピング定義すればよい。

```
ValidationException: The field '_管理組織 ID' (whose xml name is '管理組織 ID') is
 a required field.;
   - location of error: XPATH: BpcDocument/発注伝票情報リスト/発注伝票情報/発注
伝票明細/付帯情報リスト/付帯情報
```

図 3.25 エラーメッセージ（1）

(2)送信アダプタのテストツールを実行して下記の JXPathException が表示された場合には JEDICOS-XML のマッピング定義（XPath）の表記に誤りがあると思われる。この例では、” メッセージ情報/企業識別情報/発注企業/部署情報”というノードは値を持つことができない。適切ではない XPath が指定されたということがエラーメッセージからわかる。これを解決するには、マッピングシートの” メッセージ情報/企業識別情報/発注企業/部署情報”が指定されているところを、”メッセージ情報/企業識別情報/発注企業/部署情報/部署コード”などに修正する。

```
Exception in thread "main" org.apache.commons.jxpath.JXPathException:
Exception trying to create xpath メッセージ情報/企業識別情報/発注企業/部署情報;
Cannot modify property: jp.go.meti.dscm.jxml.発注企業.部署情報;
Cannot convert value of class java.lang.Integer to type class jp.go.meti.dscm.jxml.部
署情報; Cannot convert class java.lang.Integer to class jp.go.meti.dscm.jxml.部署情報
```

図 3.26 エラーメッセージ（2）

４．提供ファイル

４.１ ファイル一覧

業務ＡＰアダプタフレームワークで提供するファイルを以下に示す。（これらはビジネスモジュールソースファイルから生成できる、dscm-bm-1.0-adapter-sdk.zip ファイルに含まれている。）

表 ４.１ 提供ファイル一覧

フォルダ	ファイル名	説明
(ルート)	BmMapping.xls	マッピング表（自動生成ツール）
	readme.txt	提供物に関する説明
bin	ToBpcMapperTest.cmd	送信アダプタのテストツール起動スクリプト
	FromBpcMapperTest.cmd	受信アダプタのテストツール
	OrderToBpcMapperTest.cmd	発注の業務データ→JEDICOS-XML の起動スクリプト
	OrderFromBpcMapperTest.cmd	発注の JEDICOS-XML→業務データの起動スクリプト
	order.dat	発注のサンプル業務データ
	ReceiveToBpcMapperTest.cmd	受領の業務データ→JEDICOS-XML の起動スクリプト
	ReceiveFromBpcMapperTest.cmd	受領の JEDICOS-XML→業務データの起動スクリプト
	receive.bat	受領のサンプル業務データ
	build.cmd	ビルドコマンドファイル。コンパイルには J2SE1.4 以上が必要である。
doc	*.html	JavaDoc API ドキュメント
lib	bpc.jar	ビジネスモジュールのライブラリ
	commons-jxpath-1.1.jar	Jakarta Commons JXPath ライブラリ
	commons-digester-1.5.jar	Jakarta Commons Digester ライブラリ
	commons-logging-1.0.4.jar	Jakarta Commons Logging ライブラリ
	castor-0.9.5.3.jar	ExoLab Group XML mapping ライブラリ
	xercesImpl.jar	Xerces XML ライブラリ
	xml-apis.jar	Apix XML ライブラリ
samplesrc	**/*.java	サンプル実装（参考）

src	**/*.java	マッピング表から自動生成されるファイルおよびサンプル

５．制限事項

業務ＡＰアダプタフレームワークで提供される各種機能を利用して業務ＡＰアダプタを開発する場合に留意すべき制限事項について注記する。

- 業務ＡＰアダプタのフレームワーク、特にファイルの I/O 機能はマルチスレッドアプリケーションでは動作しない。

６．ライセンス

本製品が使用しているソフトウェアのライセンスに関して必要な表記を以下に示す。

"This product includes software developed by the Apache Software Foundation

(http://www.apache.org/)".

"This product includes software developed by The ExoLab

(http://www.exolab.org/)".

"This product includes software developed by James House

(http://http://www.quartzscheduler.org/)".

本製品で利用しているその他のソフトウェアのライセンスに関してはインストール媒体の license¥Readme.txt を参照してください。

７．付録

７.１ （別紙１）業務オブジェクトのスキーマ図

本システムで利用した業務オブジェクトのスキーマを示す。JEDICOS-XML のスキーマをベースとしている。流通システム開発センターによって策定された 2004 年 5 月 28 日版 a をベース（表７．１）にノードの和集合を取った構造になっている。ルートノードは BpcDocument というノードであるが業務ＡＰアダプタ作成者は BpcDocument の下からのノードを XPath でマッピングする。

表 ７.１　業務情報オブジェクトがベースとしたスキーマ

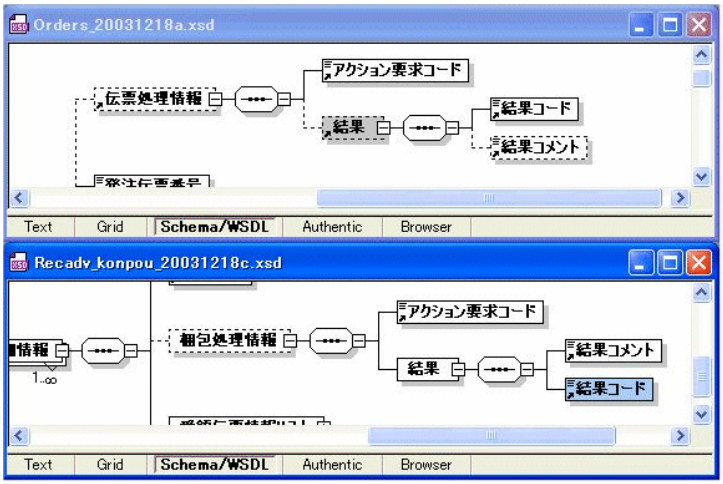
#	JEDICOS-XML スキーマ定義ファイル（XSD）
1	発注（Orders_20031218a.xsd）
2	入荷予定梱包（Desadv_konpou_20040223a.xsd）
3	検品受領（Recadv_konpou_20040223a.xsd）
4	請求（Invoice_20040223a.xsd）
5	支払案内（RemittanceAdvice_20040223a.xsd）

注意事項

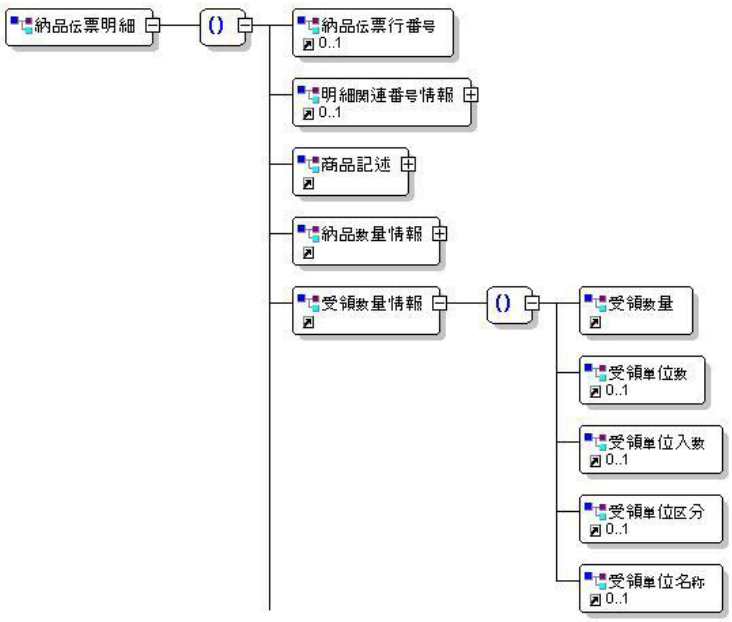
JEDICOS-XML の定義で明らかに誤りであるものをいくつか修正している。

- Desadv_konpou_20040223a.xsd ではタグ名に「振分リスト」と記述されているが、他のメッセージに合わせて「振分けリスト」(「け」を挿入)とする。
- Desadv_konpou_20040223a.xsd での「納品伝票明細/欠品情報/欠品数量」の型を型なしから xsd:integer とする。
- Desadv_konpou_20040223a.xsd での「税率」の型を xsd:string から xsd:integer とする。
- Desadv_konpou_20040223a.xsd での「付帯情報/管理組織 ID」の型を他のメッセージに合わせて xsd:string から xsd:integer とする。
- Desadv_konpou_20040223a.xsd での「振分け先リスト」の子要素「振分け先情報」を maxOccurs="unbounded"とする。
- Recadv_konpou_20040223a.xsd での「税率」の型を xsd:string から xsd:integer とする。
- Recadv_konpou_20040223a.xsd での「振分け先リスト」の子要素「振分け先情報」を maxOccurs="unbounded"とする。

- Recadv_konpou_20040223a.xsd での「振分けリスト」の子要素「振分け情報」を maxOccurs="unbounded"とする。
- Recadv_konpou_20040223a.xsd での「付帯情報/管理組織 ID」の型を他のメッセージに合わせて xsd:string から xsd:integer とする。
- Recadv_konpou_20040223a.xsd での「結果」の構造と型を他のメッセージに合わせる。



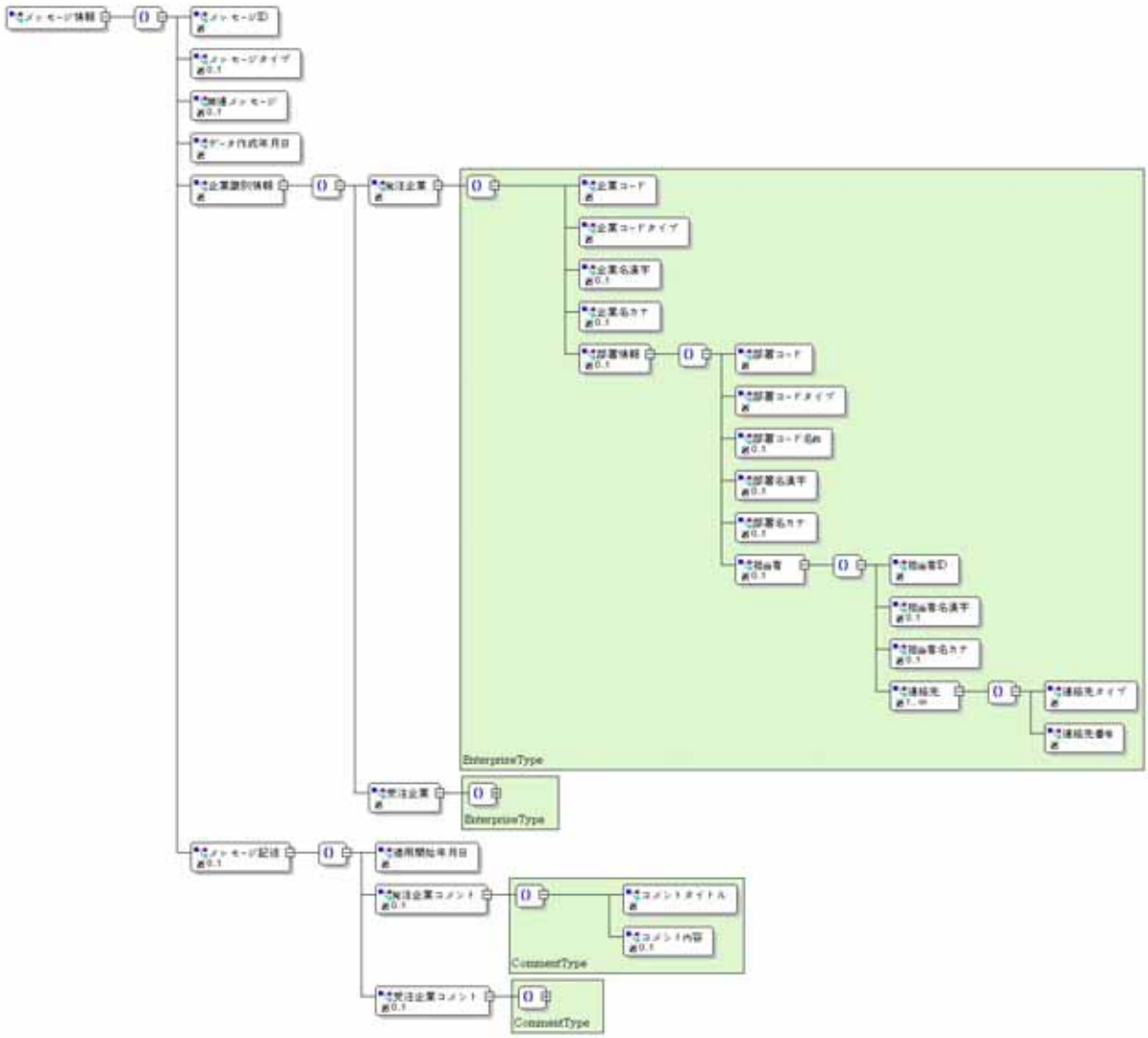
- Orders_20031218a.xsd ではタグ「商品コード」の型が String1to14Type と記述されているが、他のメッセージに合わせて String1to16Type とする。
- RemittanceAdvice_20040223a.xsd の「納品伝票明細」に「受領数量情報」を下記のように追加した。なお、「受領数量」「受領単位数」「受領単位入数」は decimal 定義とした。

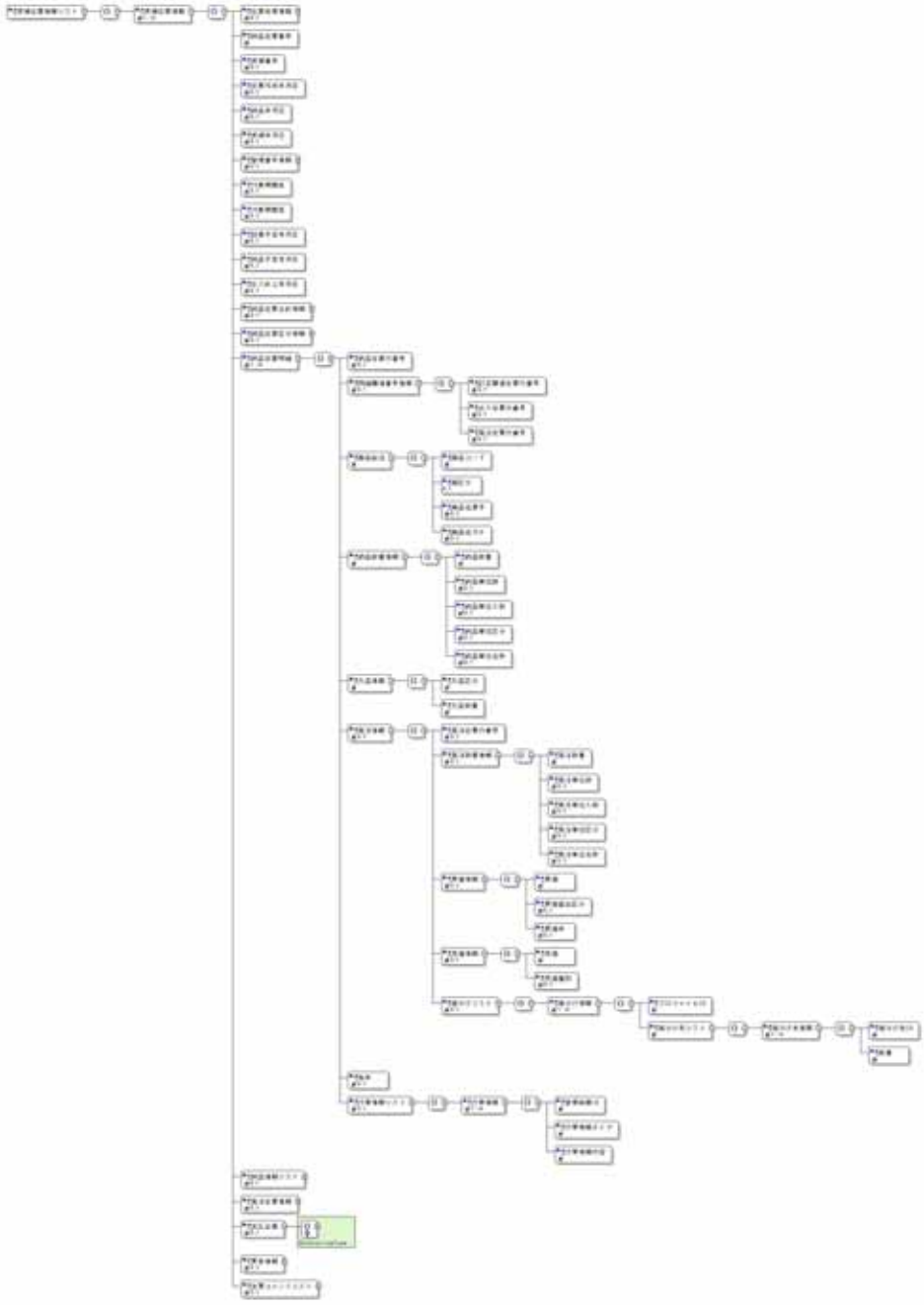


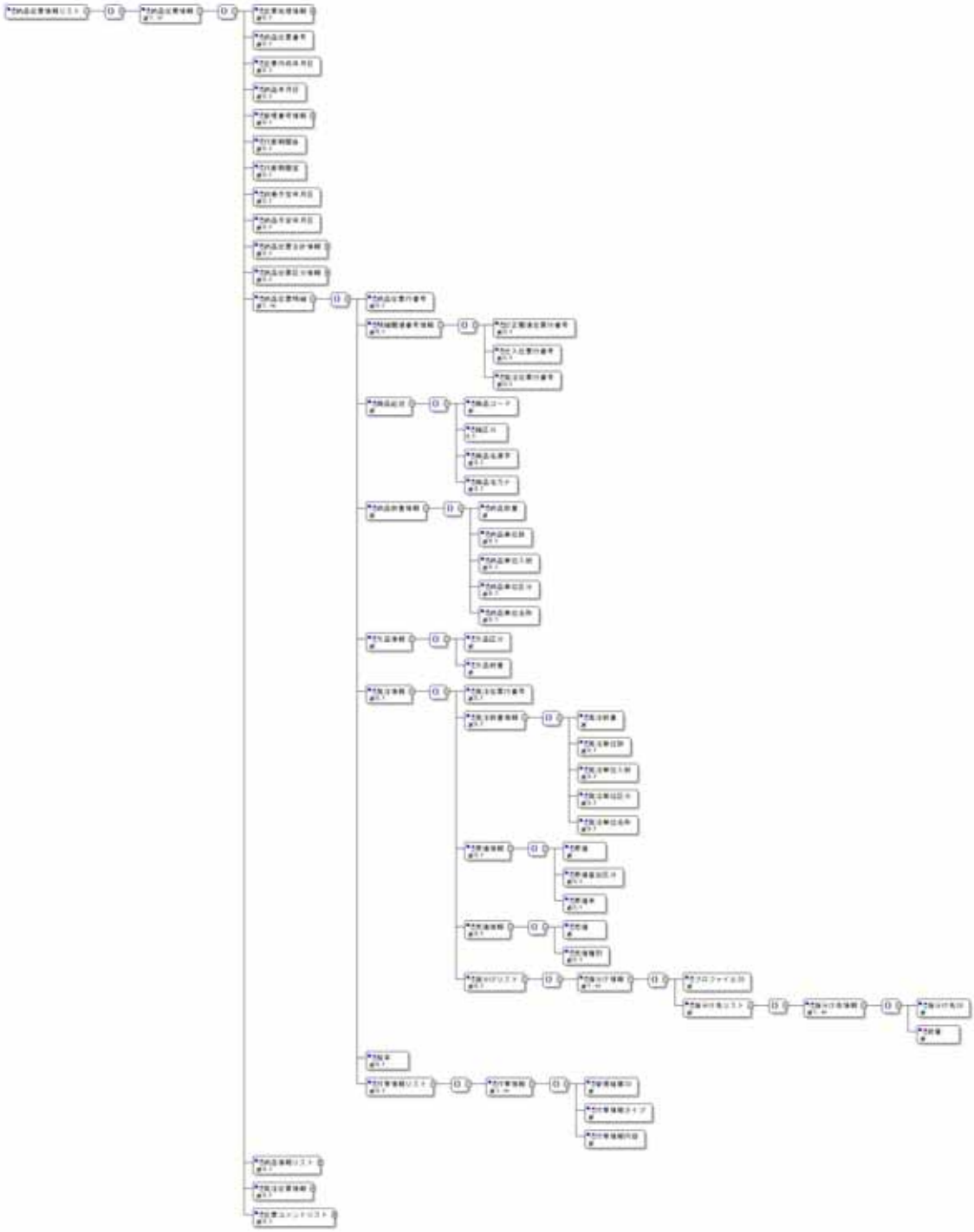
- 支払案内(RemittanceAdvice_20040223a.xsd)および請求(Invoice_20040223a.xsd)の「コメント」「伝票コメント」を複数記述できるように登場回数を maxOccurs="unbounded"**に設定した。**
- 実験企業が 32 文字使用していたため Recadv_konpou_20040223a.xsd の「梱包番号」の型を「String1to20Type」から string に変更した。
- 整数よりも小数のある可能性が高い「原価」、「売価」、「原価金額合計」「売価金額合計」「納品数量」「納品単位数」「納品単位入数」「欠品数量」など数量と価格に関する型を Integer から decimal に変更した。**これにより、マッピングツールで「文字」と指定している場合にこれらの項目にマッピングされていたものが、(いままではパースして数値に変換できればエラーとならなかったものが) エラーとなる可能性がある。**
- 各企業で上記のスキーマに含まれない項目(例えば○○区分コード)があった場合には付帯情報に各企業間で合意した内容で記述する。

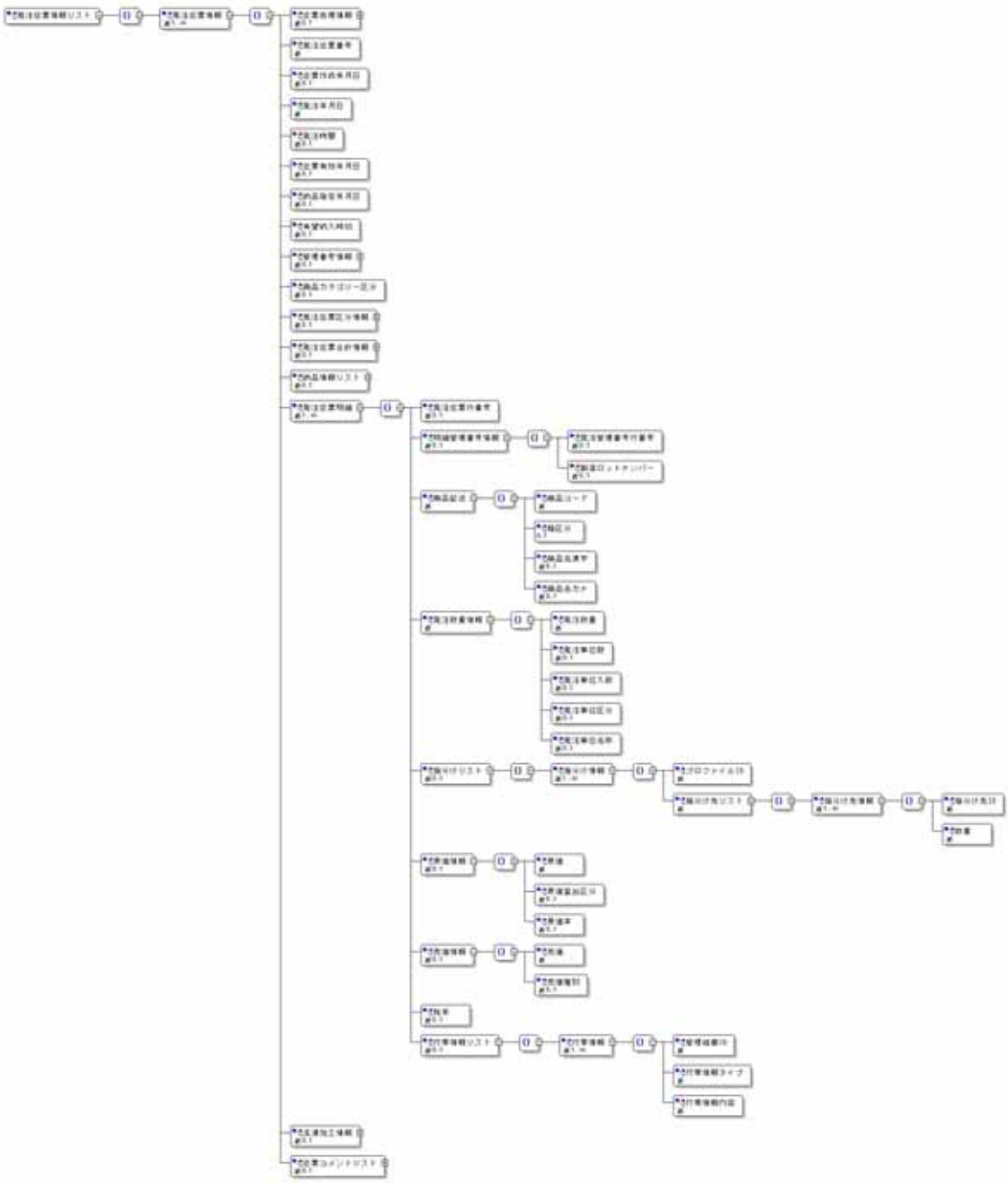
7.2 (別紙2)マッピングルール

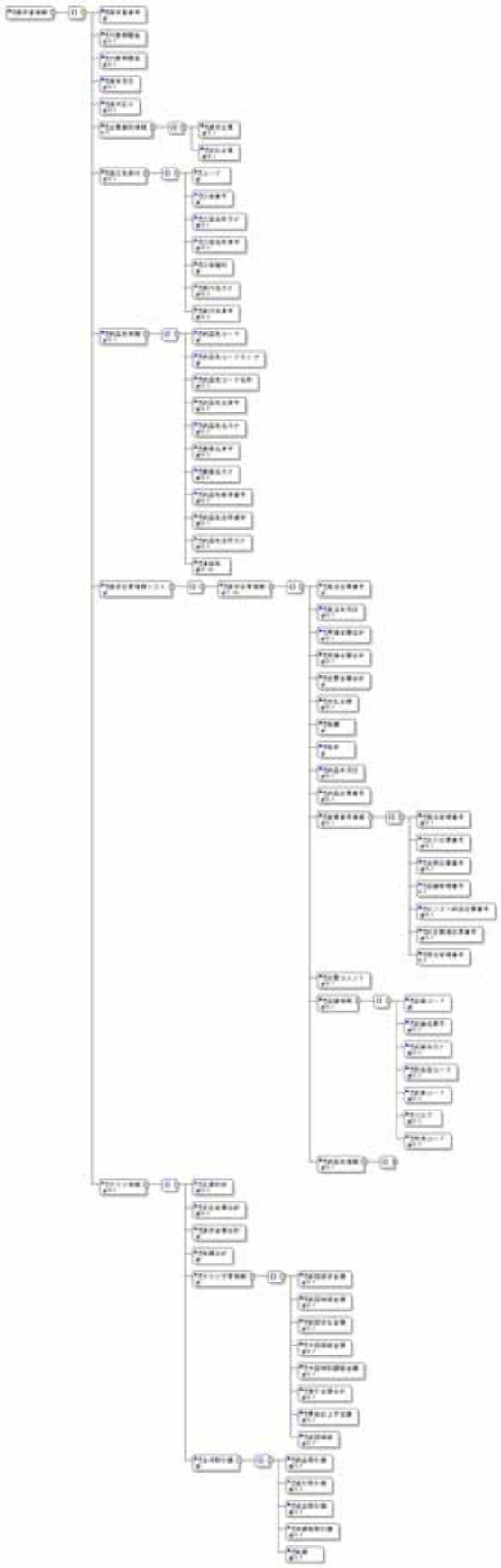

BpcDocument
()
メッセージ情報
梱包情報リスト
0..1
発注伝票情報リスト
0..1
請求書情報
0..1
支払案内情報
0..1
商品リスト
0..1
POS売上情報
0..1
提案タイトル
0..1
提案コメント
0..1
バイヤコメント
0..1
特売記述
0..1
特売発注条件
0..1
特売商品リスト
0..1
トレース情報リスト
0..1

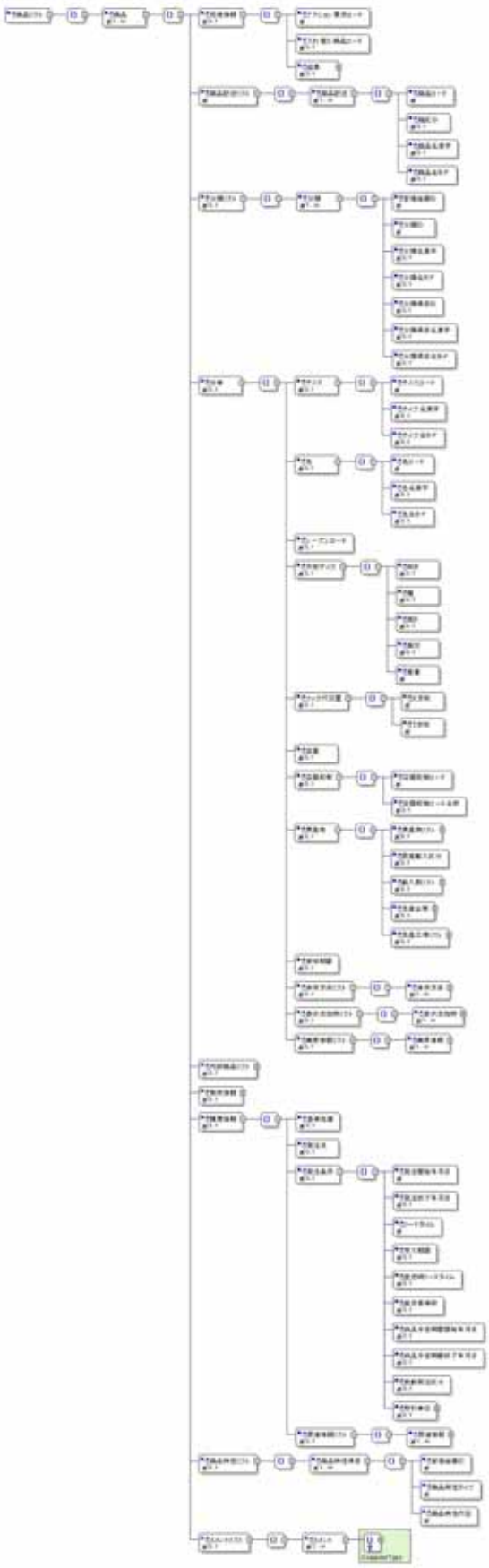

EnterpriseType
EnterpriseType
CommentType
CommentType

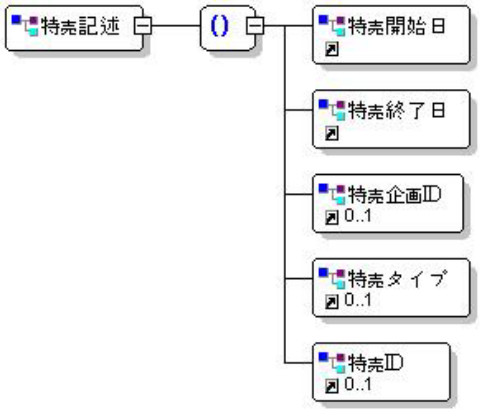


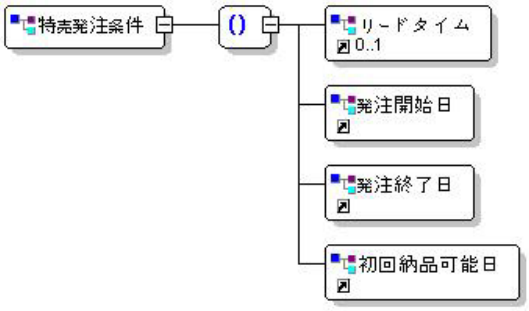


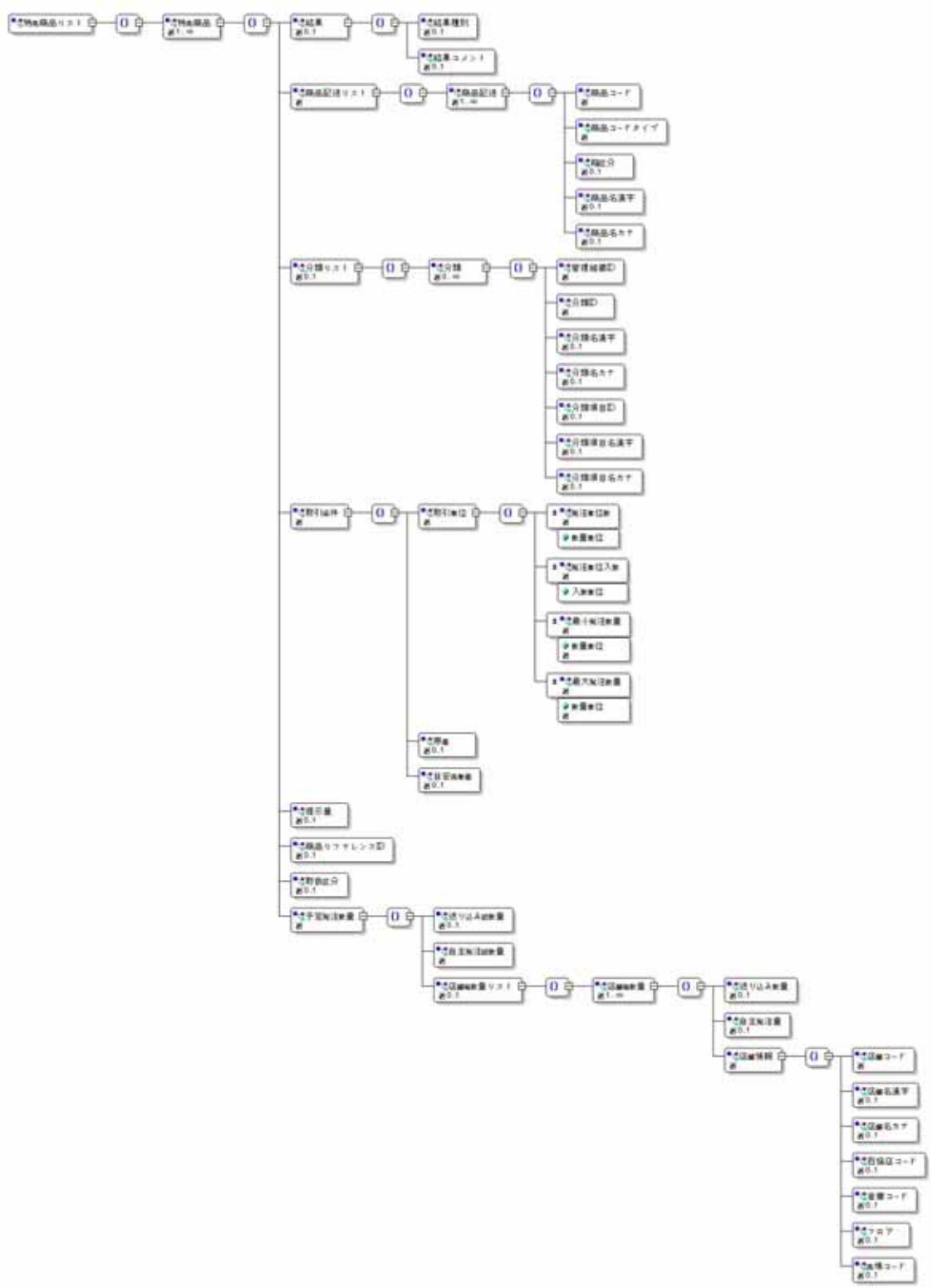







特売記述
()
特売開始日
特売終了日
特売企画ID
0..1
特売タイプ
0..1
特売ID
0..1


特売発注条件
()
リードタイム
0..1
発注開始日
発注終了日
初回納品可能日



業務APアダプタからJ-XMLメッセージへのマッピング

別紙3(1)

#	業務AP（種別）	BM内部表現	J-XML	例（AP(桁数6:小数2)→BM）		備考
1	数値	BigDecimal	Integer	"040930"	409	小数点以下は切り捨て
2			String	"040930"	"409.3"	-
3			Date	"040930"	エラー	常にエラー
4			Decimal	"040930"	409.3	-
5			Float	"040930"	409.2999	精度が悪くなる可能性あり
6	文字	String	Integer	"040930"	40930	Integerに変換できない場合エラー
7			String	"040930"	"040930"	-
8			Date	"040930"	エラー	常にエラー
9			Decimal	"040930"	エラー	常にエラー
10			Float	"040930"	40930	Floatに変換できない場合エラー
11	日付	Date	Integer	"040930"	エラー	常にエラー
12			String	"040930"	"2004-09-30"	
13			Date	"040930"	2004-09-30	"041231"は2005-01-01となる
14			Decimal	"040930"	エラー	常にエラー
15			Float	"040930"	エラー	常にエラー
16	予備	-	Integer	"040930"	-	-
17			String	"040930"	-	-
18			Date	"040930"	-	-
19			Decimal	"040930"	-	-
20			Float	"040930"	-	-

※網掛けのマッピングを推奨