サービス連携における非機能要件設計上の考慮点

2007.5.15

XMLコンソーシアムSOA部会
日本アイ･ビー･エム(株)　根本和郎

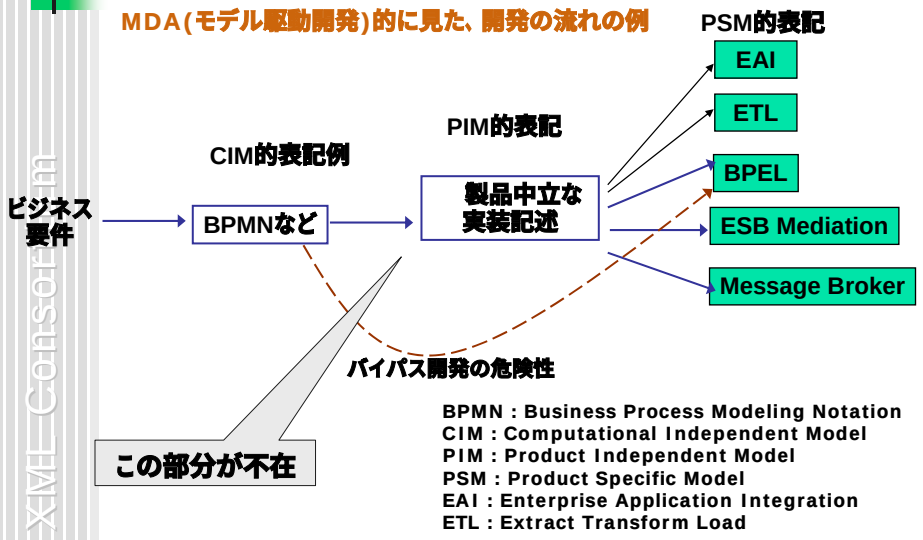
アジェンダ

- 今までのSOAインテグレーション表記法の確認
- 直面している課題は・・・
  - SOAの実装中立な設計手法がない。
- グレゴールグラムを使用することで、実装設計が視覚化できるのではないのか?
  - 試験的に実施し、役立ちそうであることまでは確認した。
- 非機能要件実装上の検討項目の確認

プレゼンテーションの動機と目的

- 動機
  - インテグレーション上の課題点が、網羅的に考察しにくい。
  - 非機能要件を加味した、統一的なインテグレーションモデルが不在。
- 目的
  - インテグレーションモデルを構築するための考慮点の提示。
- 対象者
  - SOAインテグレーション実装設計者

MDA的に見た、開発の流れ



既存のインテグレーションモデルの確認

サービスとメッセージ

- アナロジー：サービス提供体をレール、メッセージを列車に模して考え、レールの結合テクニックを論じる。

サービス・コンポーネント

よく定義されたインターフェイスを持ち、任意に接続可能。あらゆる列車が通過可能。

サービス・メッセージ

定義された車輪を持つ列車同士も連結可能。

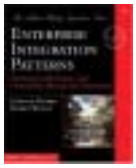
関連仕様: SCAとSDO http://www.osoa.org/

本発表のカバー領域

	コンポーネント (レール)	インテグレーション (レールの連結)
機能	サービス抽出方法論は、多数議論されている。	
非機能	BPEL実装方法などが相当	ここが空白部分であると認識し、ここを議論します

今回のカバー領域

“Enterprise Integration Patterns”

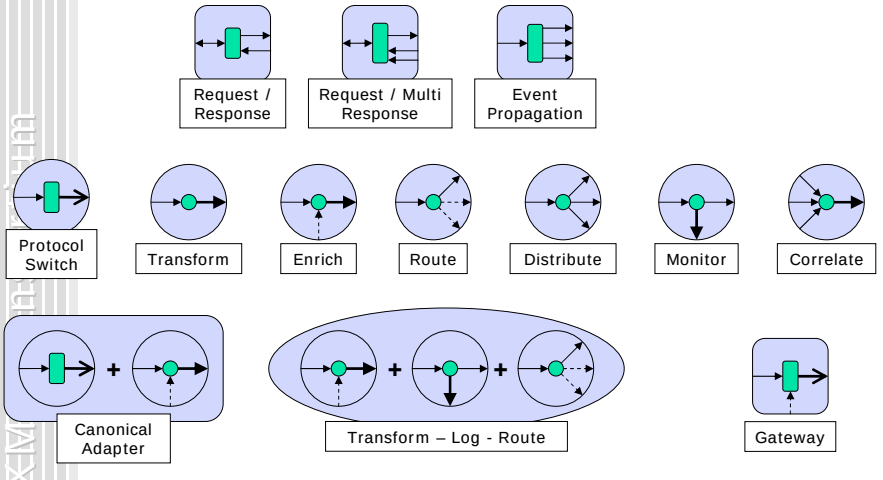


Enterprise Integration Patterns より、「グレゴールグラム」

“Gregorgram” :一昨年より、これに取組中

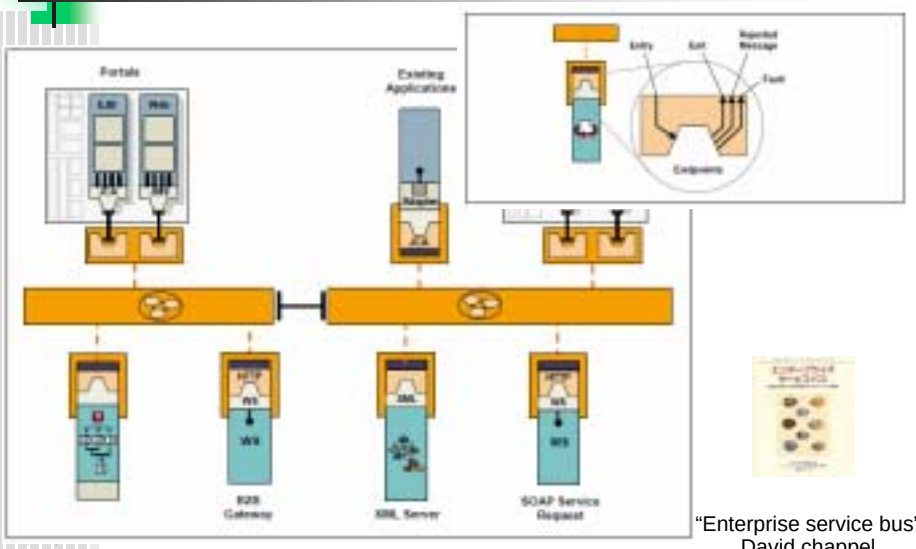
“Mediation Patterns” developerWorksより

http://www-128.ibm.com/developerworks/library/ws-soa-progmodel4/



これはESBの表記のためだけのもの

“Enterprise Service Bus”スタイル



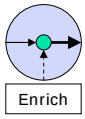
“Enterprise service bus”
David chappel

主にAdapter表記のためのもの

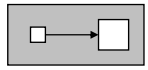
- 表記法の改善活動の一歩として、記載事項の洗い出しを行う。
- SOAの世界での統一表記言語は不在
- いわばUnified Modeling Language for Service Integrationが望まれる

表記の対立している例 “Enricher”

“Mediation Patterns” style

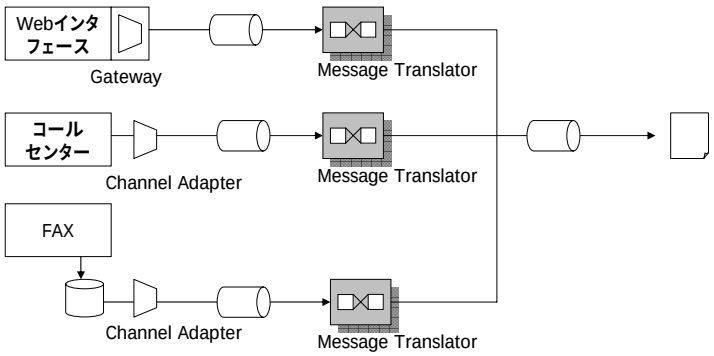


“Enterprise Integration Patterns” style



グレゴールグラムの例

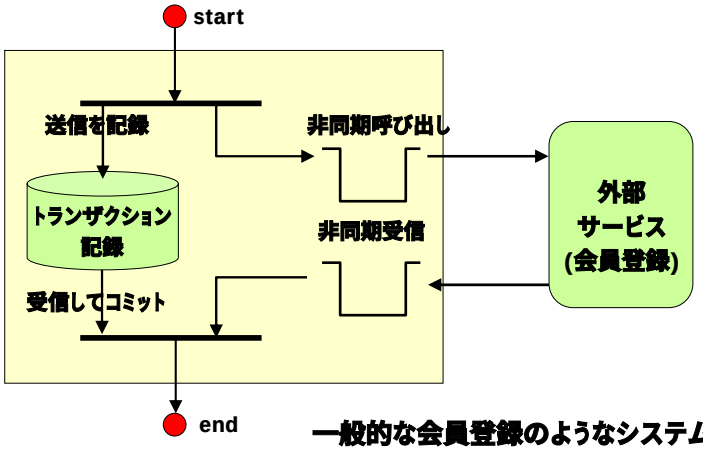
- 3つの異なるチャネルからの注文受付



- 記述すべき非機能要件の確認
- 次項以降、考慮点の一部を提示

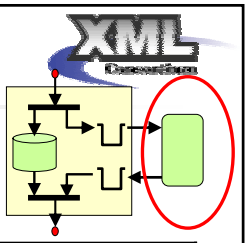
ケーススタディー

- 単純な外部サービスの非同期呼び出しのケースで考察、次項以降述べていく。



一般的な会員登録のようなシステム例

データの種類を特定する



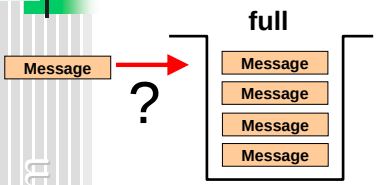
環境の前提条件の確認データの種類 3種。
(CDは容易、Rは普通、Uが困難)

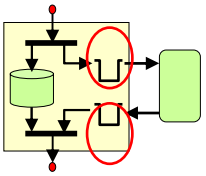
カテゴリー	データのタイプ	CRUD?	例
資源型	マスター	CRU	顧客テーブル。 生産ラインの部品表
	カタログマスター	CR	パンフレットの版 売った車の部品表
管理型	Transaction data	CRUD	ステート管理
イベント型	Activity data	C	トランザクションログ

CRUD = Create Read Update Delete の基本操作タイプ

対立意見 : “CRUDy interface”アンチパターン

キュー飽和ポリシー 4種



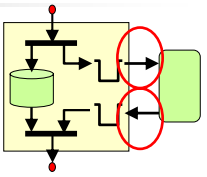


飽和ポリシー	解説
Discard	新着メッセージを消去する
Discard oldest	最高齢未処理メッセージの放棄
Abort	Exceptionを上げ、外部に解決を委託する
Caller-runs	呼び出し側が検知、対応する

参照 “Java並行処理プログラミング
―その「基盤」と「最新API」を究める”

メッセージの再投入可能性4種

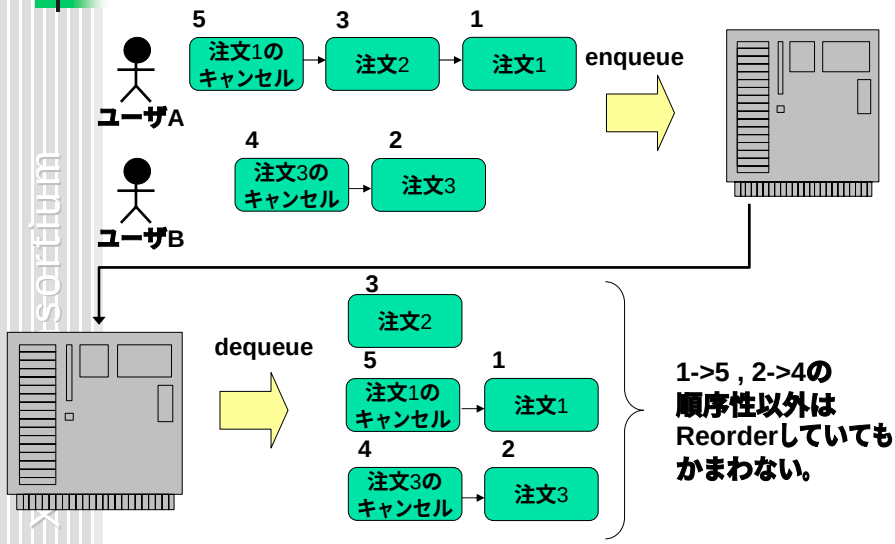
非同期メッセージング時のデータの保全性の確認



ポリシー	解説
ExactlyOnce (旧 Once and only once)	確実な1回のみの送信が許される 例:会計処理のほとんどのケース。 Backward recovery必須。
AtLeastOnce	繰り返し送信しても許される、Set/Reset系の操作。 例:会員の登録、削除など。
AtMostOnce (旧 Best effort)	不達が許されるメッセージ。 例:管理のための負荷情報など。
InOrder	完全順序性の維持、追い抜きなし。

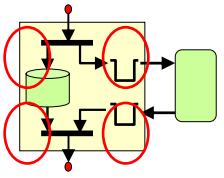
参照 “WS-ReliableMessaging”

InOrderについて補足



完全順序性が必要ないのなら、InOrderの使用は控えるべき

例外操作 4種

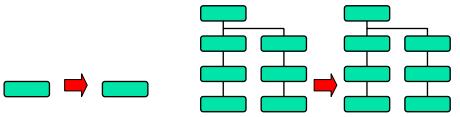
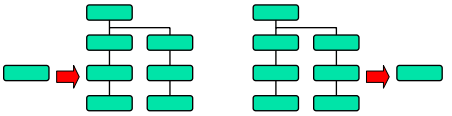


例外操作	解説
Rollback	Short running processで使用 (例:BEGIN TRANSACTION; xxx; COMMIT;)
Compensation	Long running processの複合トランザクションの回復用パスでの動作
Backout	キュー操作の失敗の回復および再試行
Abort	破損メッセージの系外への排出

メッセージ構造の種類

構造	解説
シンプル型	ひとつのメッセージで処理が進む。 例:ひとりの社員の人事異動情報。
階層型	複数のメッセージの処理で、ビジネス上のひとつ意味を持つ処理。 例:部門単位での人事異動情報。

メッセージ連携

	解説
対象型	一対一、または多対多の同一形態。 一回で処理が完了する。 
非対称型	多対一、または一対多 複数回の処理で1トランザクションとなる。 Backward recovery必須。 

非対称な連携操作は、極力避ける

- ここまでの選択肢を複合して考察した場合を次項で示す。

統合されたポリシーの例

CRUD操作毎の再投入性の評価

操作	再投入性	Recovery
Create	AtLeastOnce	Forward
Read	AtLeastOnce ExactlyOnce	Forward Backward
Update	ExactlyOnce	Backward
Delete	AtLeastOnce	Forward

Backward recoveryは、別途実装負担が発生する

統合されたポリシーの例

飽和ポリシーと再投入性の選択肢

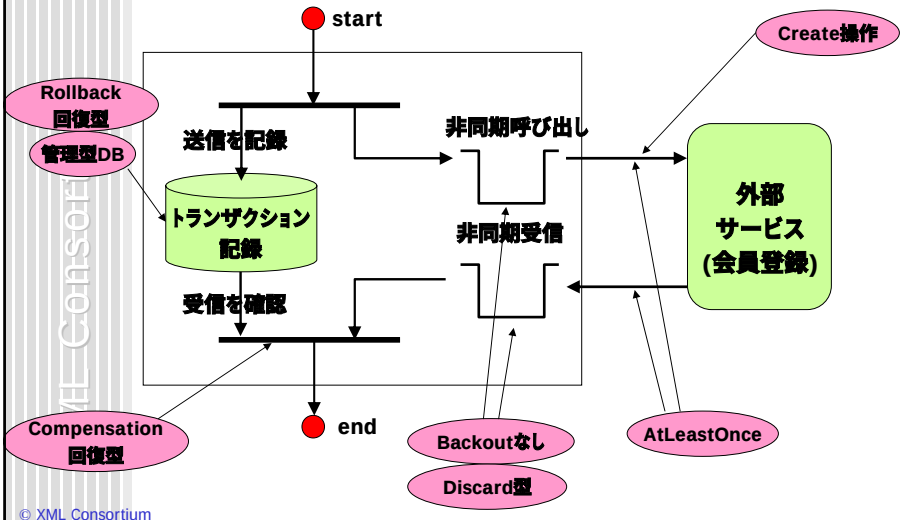
操作	ExactlyOnce	AtLeastOnce	AtMostOnce	InOrder
Discard	×	○	○	×
Discard-oldest	×	○	○	×
Abort	△	○	○	△
Caller-runs	○	△	△	△

Caller-runsは実装コストが高い点が注意点

- ここで、前出のモデルに書き込んでみる。

SOAのモデルに視覚化した例

- 会員登録という外部サービスを起動するモデル



まとめ

- インテグレーション上の非機能要件の洗い出しができた。(ただし部分的)
- 課題を可視化することにより、議論の促進を図る素地ができた。
- インテグレーション実装上の課題の確認。

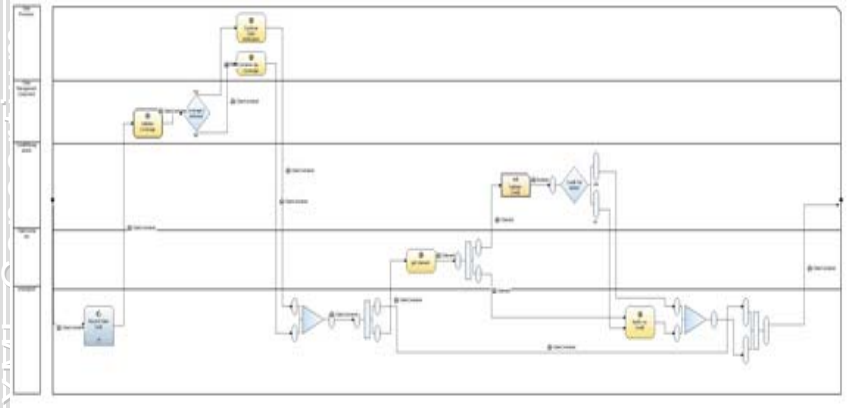
今後について

- さらに継続。
- ベストプラクティスの蓄積。
- 業界標準表記法への要望とりまとめ。
- 参加者募集。

- 以降は、補助資料

BPMN (Business Process Modeling Notation)

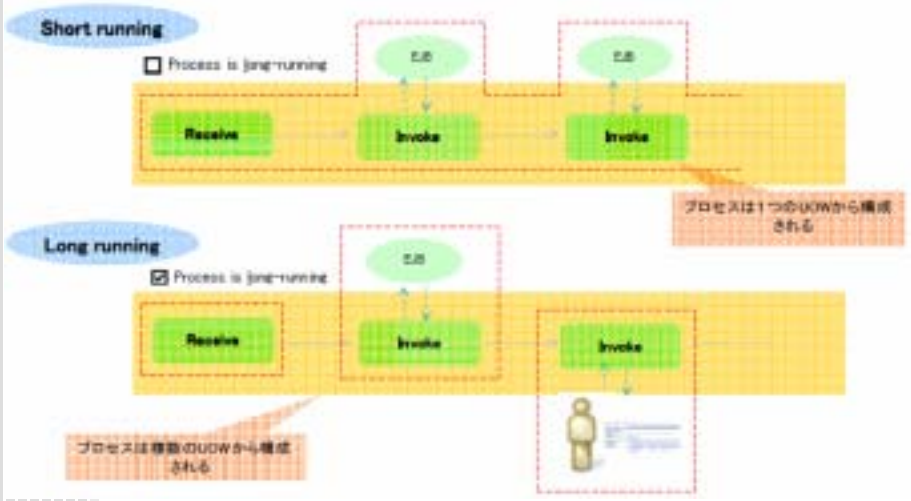
- ビジネス粒度での表記、KPIの評価などを行う
- 人による処理も記述可能、非自動処理もモデル化できる。



※ KPI = Key Performance Indicator

Long running processのサポート

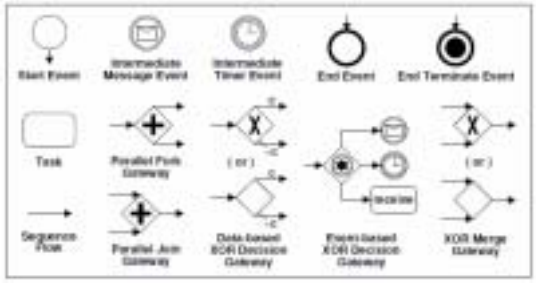
UOW = Unit Of Work



※ Long running transactionはRollbackできない、Compensateするのみ

BPMNからBPELへの変換

BPMN (Business Process Modeling Notation)



これはモデル

これは実装

開発ツール

BPEL (Business Process Execution Language)