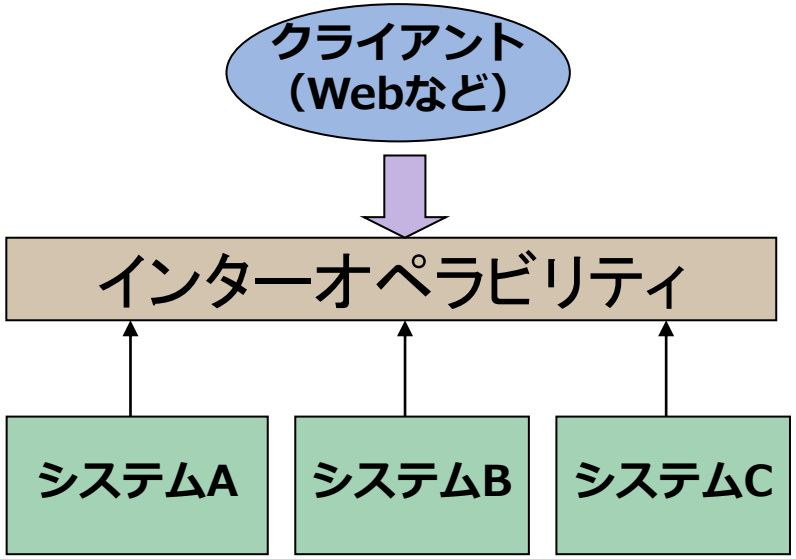
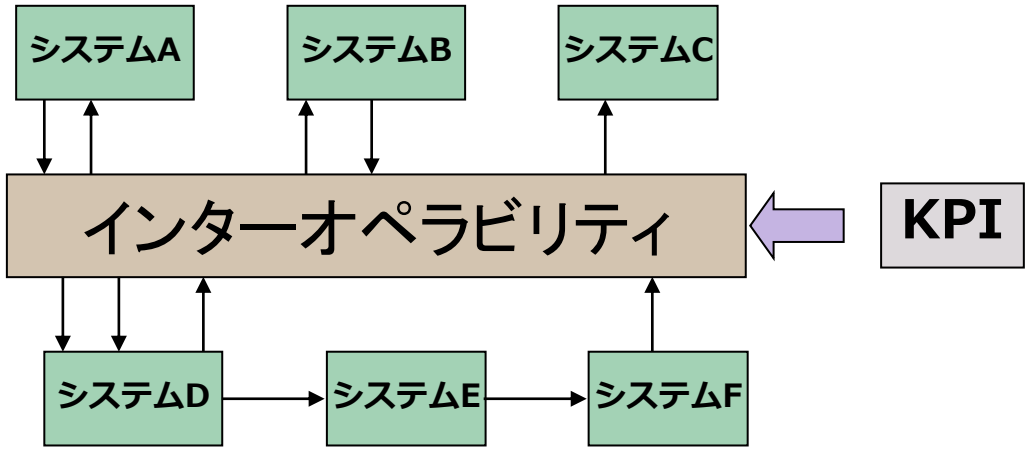
説明内容

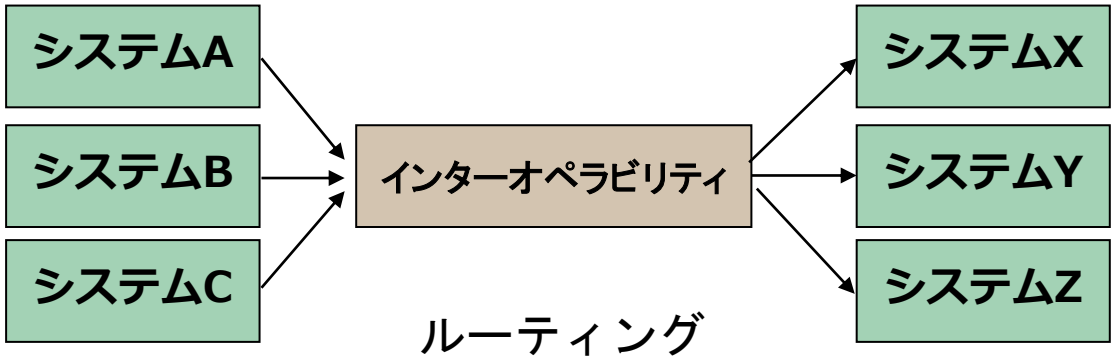
プロダクションのユースケース



複合アプリケーション



システム間のメッセージ交換(ESB)

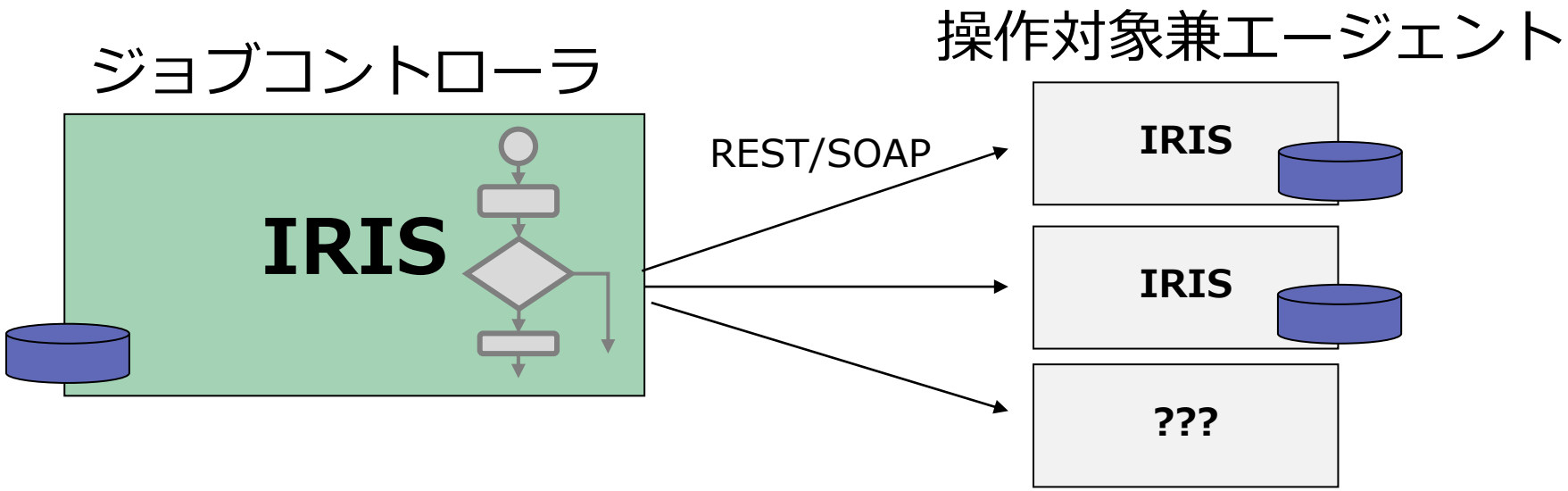


ルーティング

ETL

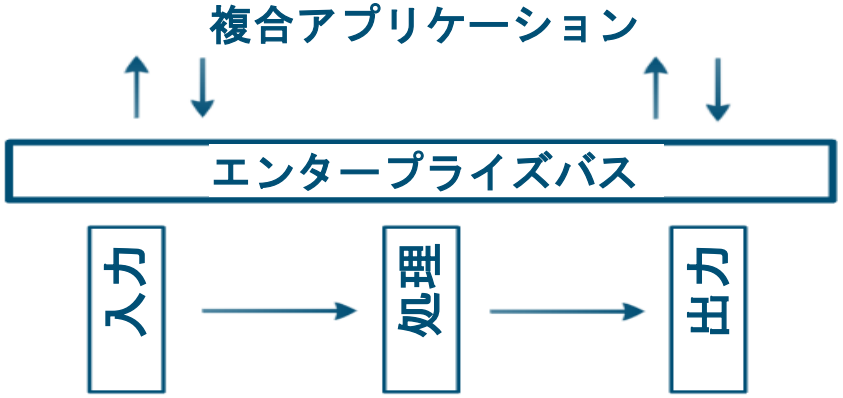
バッチジョブ管理への適用

- 長くて、並列/直列処理、完了の待ち合わせが必要で、終了時間が読みにくい一連の操作の制御を行うためにBPをどのように活用できるかを訴求したものです。
- 「バッチジョブ管理」と題していますが、同じような性質のフロー制御に対して応用が可能です。
- ジョブが実行される側(操作される側)もIRISインスタンスとしていますが、REST/SOAP等の何らかの入出力の仕組みを備えていれば、どのようなソフトウェアでも連携可能です。



なぜビジネスプロセス？

バス型アーキテクチャで多くのシステムの連携を効率的かつスムーズに。



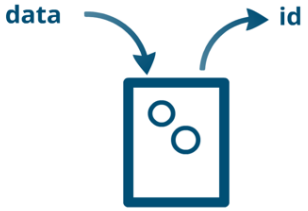
連携されるデータは自動的にメッセージとして永続化される。

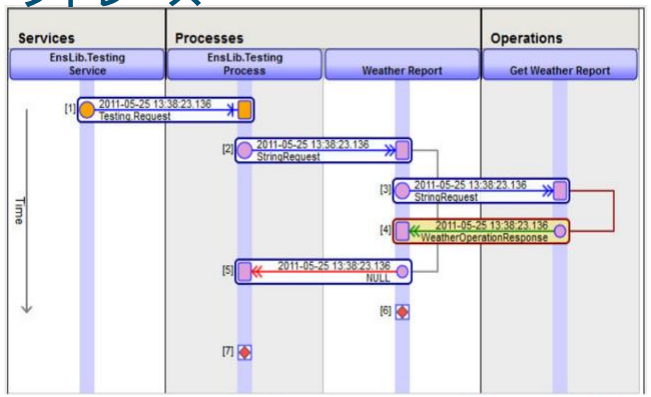
メッセージのトレースによる連携履歴の確認が容易。

障害時にメッセージが失われず、再送などの対応が可能。

オブジェクト化/永続化されたメッセージと

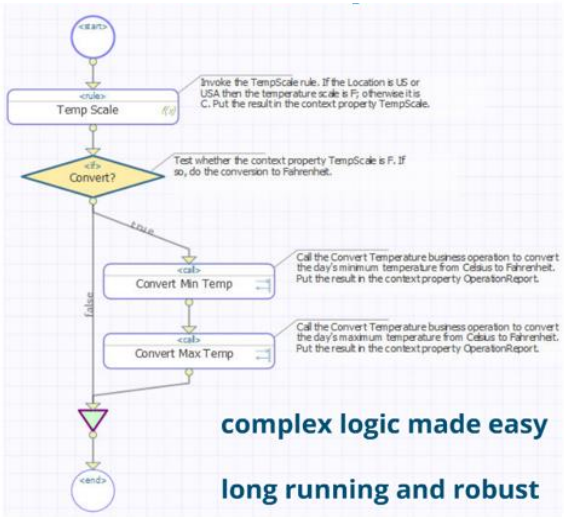
ビジュアルなメッセージトレース





find and resolve problems quickly

ビジュアルなプロセス記述



システム連携のフローをビジュアルに定義可能。

定義したフローは、コンパイルされIRISのコードとして実行される。

人間の介入など長時間実行されるフローもサポート。

豊富なアダプタ

web services
tcp
java
.net
c & c++
hl7
dicom
astm
xml
json
rest

さまざまなデータ形式やプロトコルに対応したアダプタにより、連携の細部のコーディングが不要。

HL7, DICOMなど医療用のアダプタも標準装備。

言葉の定義

- インターオペラビリティ機能

人、プロセス、アプリケーション、およびシステムをつなぐ、つまり、相互運用することを目的としたIRISが提供する「機能の総称」。しばしば、プロダクションと同じ意味で使用。

- プロダクション

インターオペラビリティを実現するにあたって使用する統合フレームワーク、あるいは特定の目的のために定義された、「フレームワーク構成要素をパッケージ化したもの」の名称。実装は、プロダクションの定義から始まる。主にBS, BP, BOというソフトウェア構成要素からなる。

デプロイ、起動・停止、インポート・エクスポートしたりする対象。しばしば、インターオペラビリティと同じ意味で使用。

- BPMN

ビジネスプロセス(業務手順)を図(フローチャートのようなもの)で表現する際の表記法。最近は、地方自治体の業務プロセス・情報システムの標準化に利用されているようで、サーチするとたくさん出てきます。

- BPL

ビジネスプロセスをインターオペラビリティ機能で表現するための言語。2000年ごろに、継続的な業務の改善を目的としたビジネスプロセスの管理(BPM)を実現するために、ざまざまな組織から規格が提案されましたが(経緯は複雑)、BPLは、それらの影響を受けた独自言語です。

- シーケンス

一連のアクティビティの流れ。BPはシーケンスに従って処理を実行します。

- アクティビティ

BPLランタイムにより実行されるなんらかの行動(アクション)。最も重要なアクティビティは、BO経由で外部システムと連携するCallアクティビティ。

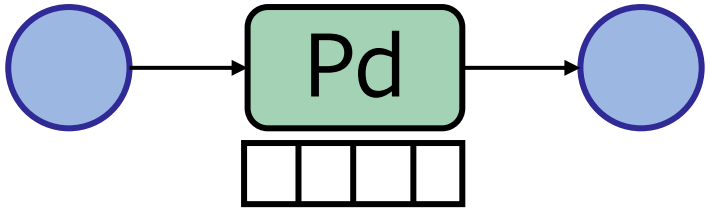
- ワークフロー

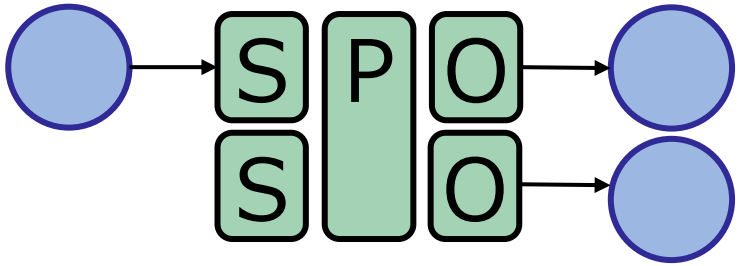
人が関わるBPの構成要素。承認フロー等を実現するために使用。

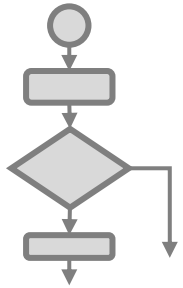
- ジョブ(Job)

BS,BP,BOが動作する際に使用するO/Sプロセス。バッチジョブのジョブのことではありません。

プロダクションのおさらい

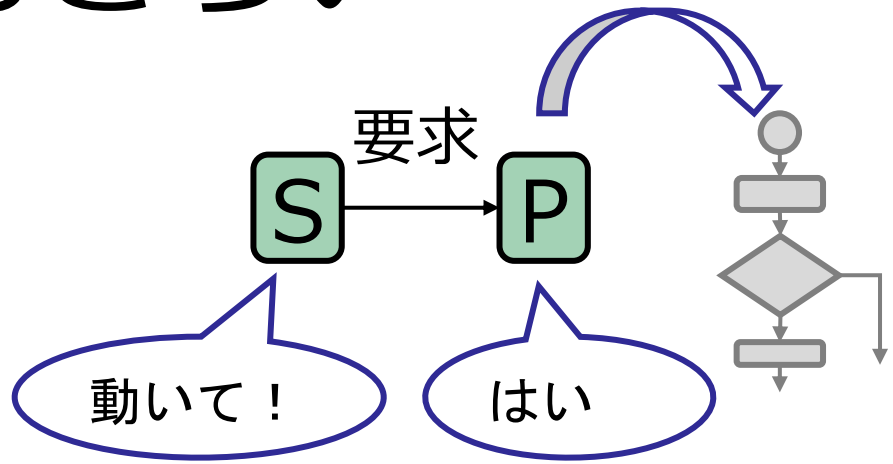


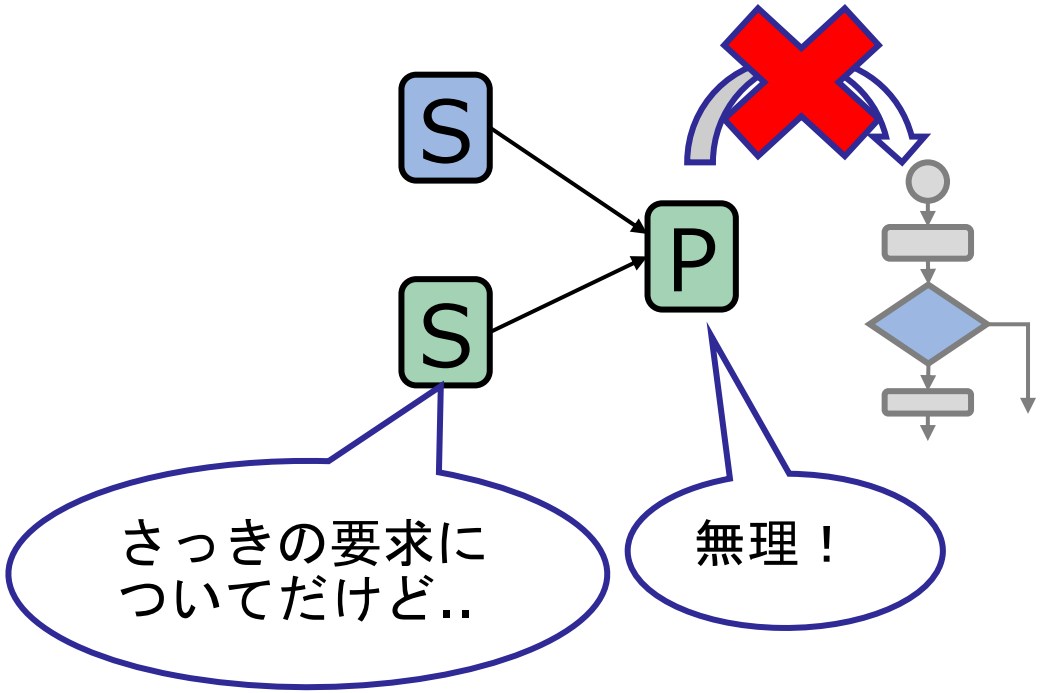


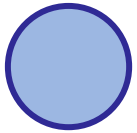


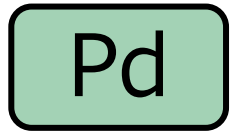
Xdata BPL
{
<process>
}

{
Method OnRequest()
}






外部システム



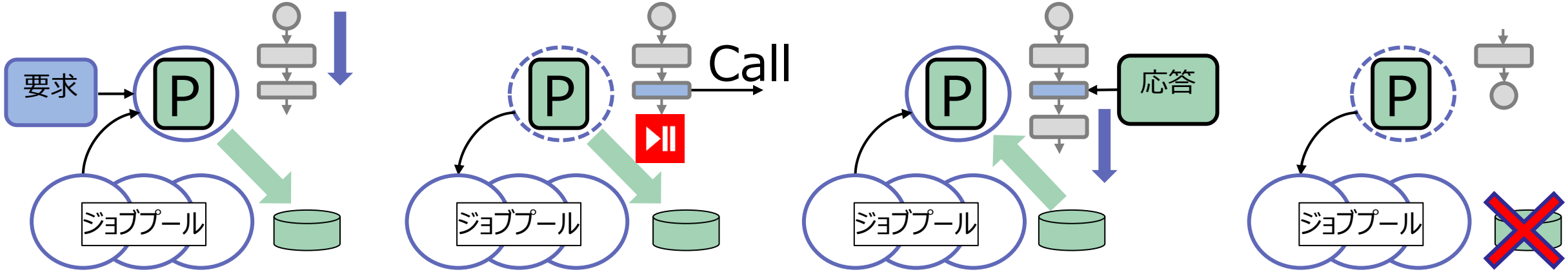
プロダクション

ビジネスサービス

ビジネスプロセス

ビジネス
オペレーション

BPの仕組み#1/ライフサイクル



1. 要求を受信すると、ジョブ(O/Sプロセス)が割り当てられ、BPのインスタンスが作成・保存される
2. BPのシーケンスが開始し、以降シーケンスに従って、処理が進む
3. Callアクティビティ実行時に、その時点の状態が保存され、処理は中断する。ジョブは返却される
4. 自分宛ての応答が存在する場合、再度ジョブが割り当てられ、シーケンス実行を再開する
(自分宛ての要求が存在する場合は1.を実行する)
5. End到達時に、ジョブが解放される。ディスク上のBPレコードも削除される。

BPの仕組み#2/状態の保持

- 状態とは？

メールでユーザ群にイベント案内を出すケース

日時	送信先	送信ステータス	ユーザ選択
2023/x/x	aaa@foo	成功	参加
2023/x/x	bbb@bar	失敗	N/A
2023/x/x	ccc@hoge	成功	不参加
		成功	オンライン参加

BPが送信した
メッセージの状態

BPが保持する
アプリケーションの状態

送受信、DBへの登録

BPのシーケンスで記述

- 状態テーブルをライフサイクルを通じて維持管理する仕組み
- Call/Delay実行時に、その状態を保存し、実行O/Sプロセスをプールに返却
- 状態値は、<CODE>を通じたオブジェクト操作の際に利用可能

BPの仕組み#2/状態の保持(続き)

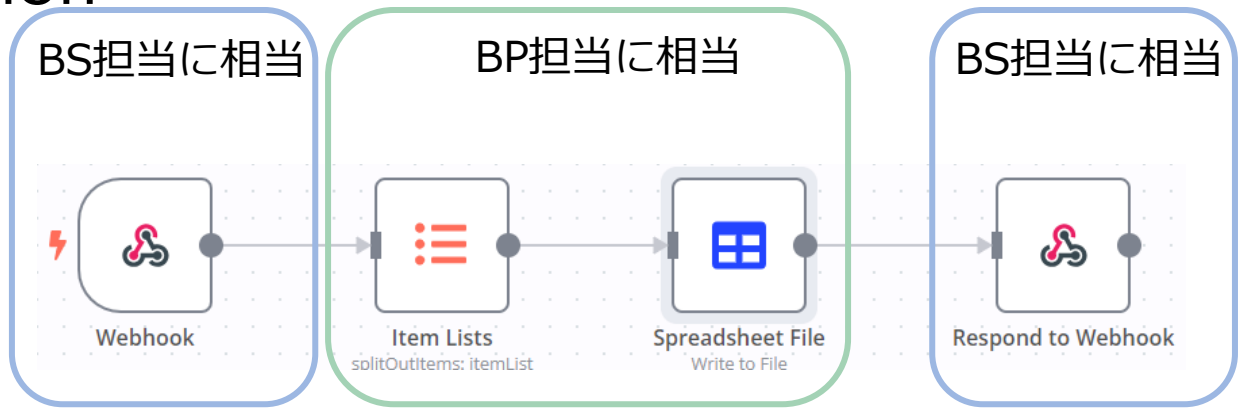
SELECT * FROM Ens.BusinessProcess

BPの仕組み#3/アクティビティ

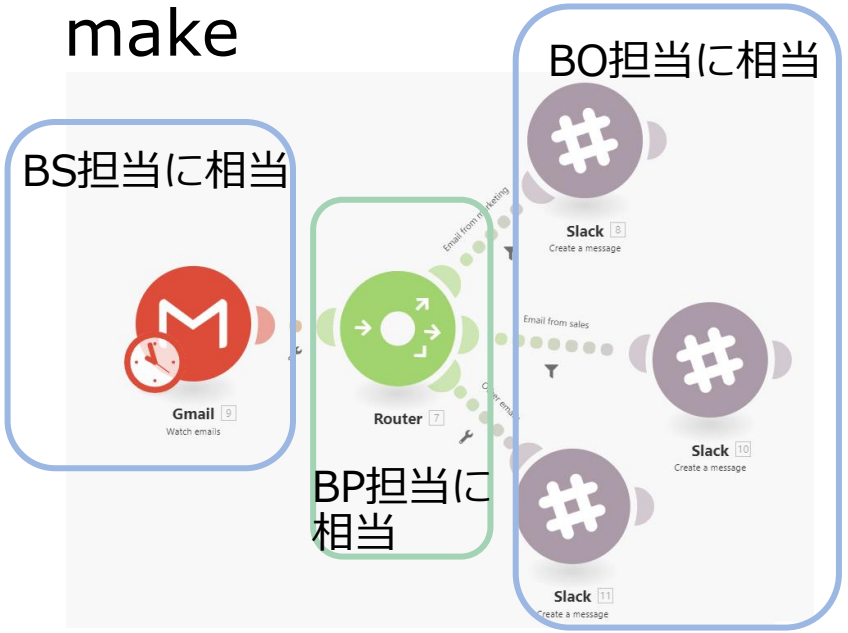
- <call>以外に、フロー制御のために<if><switch><for>などのアクティビティが用意されています
- 外部システムからの入力はBSからの要求,出力はBOコールという形で抽象化されています

Zapier,n8n,makeのようなサービス連携製品は、これほど明確に外部I/F部分と処理(プロセス)部分を分けていません

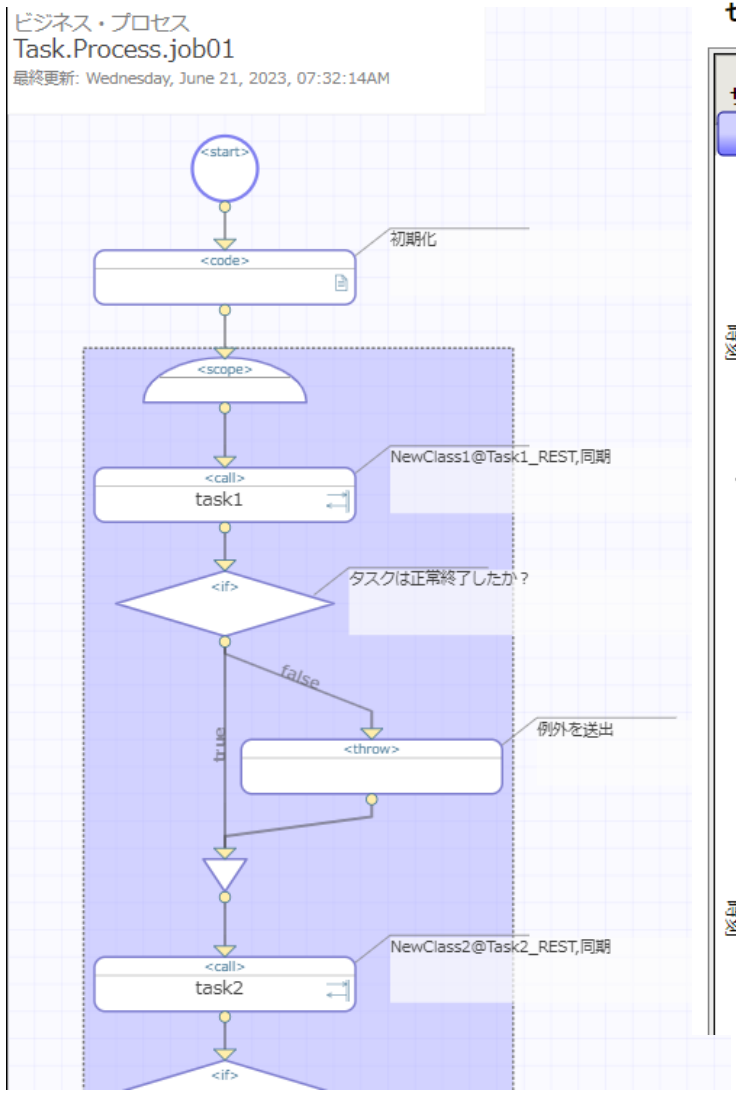
n8n

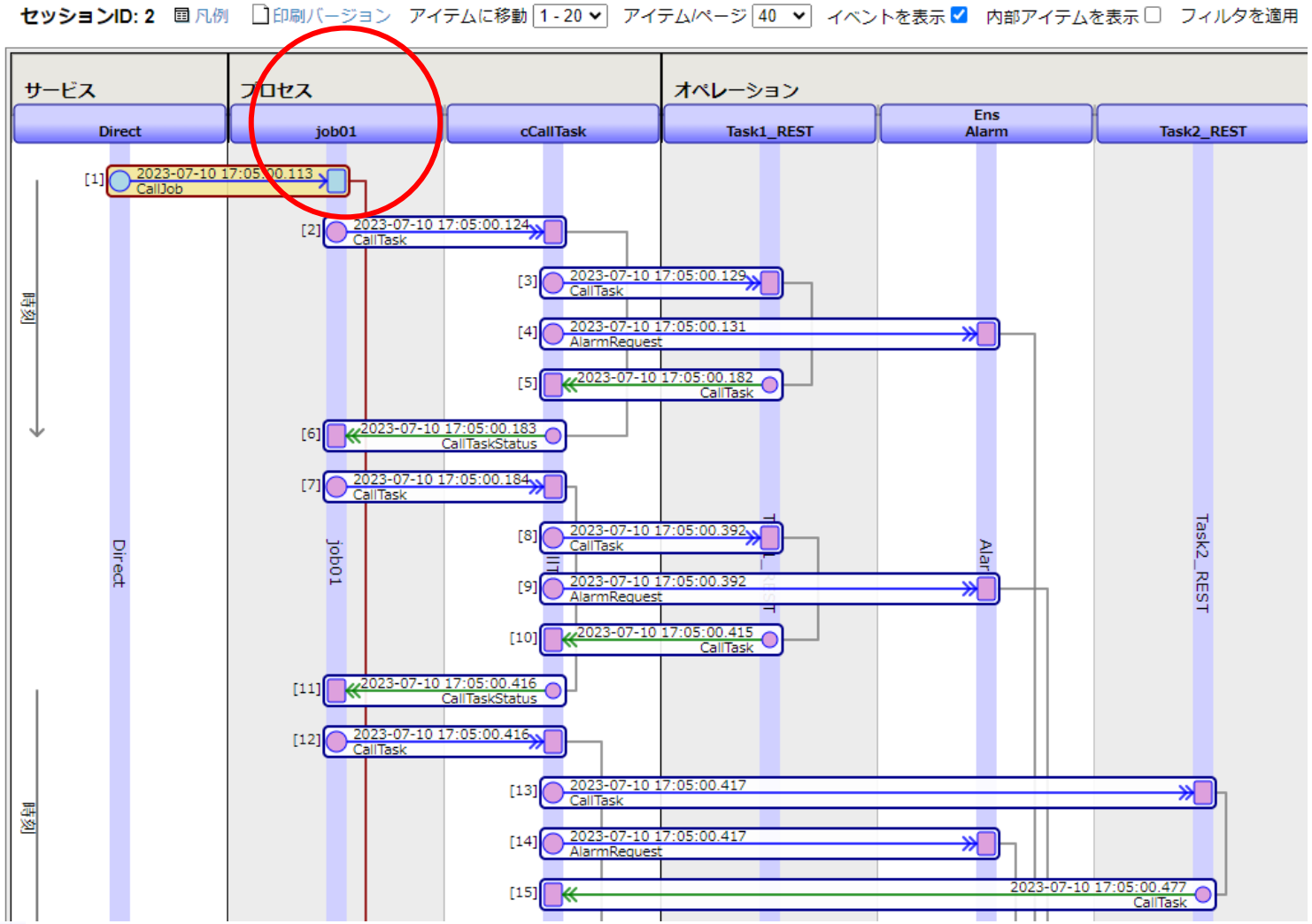


make



BPの仕組み#3/アクティビティ



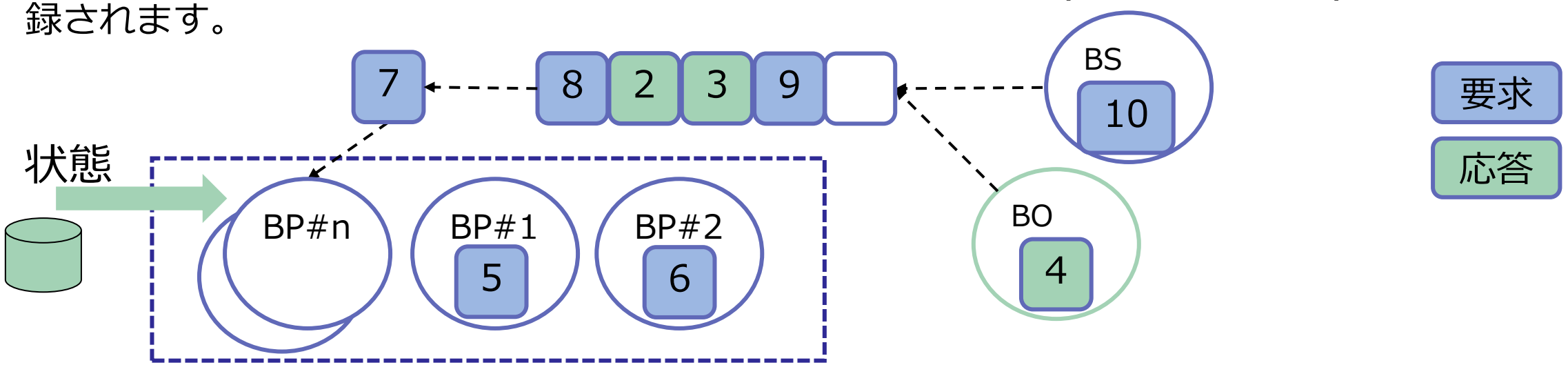


BPの仕組み#4/キュー

- BPはキュー経由で他のホストアイテム(BS,BP,BO)と連携します。
- キューと、そのキューを処理するために投入可能なO/Sプロセス(Job)の数であるプール値には密接な関係があります。

  Pool=0の時:共通のキューとジョブプール(Actor Pool)を使用

  Pool>＝1の時:専用のキューと専用のジョブプールを使用
- キューには、他ホストアイテムからの要求、BPが発行したCall(BO,BPが送信先)からの応答が登録されます。



- プロダクションのランタイムは、先頭から順次メッセージを取り出し、プールからBP実行用のO/Sプロセスを割り当て、状態(BPコンテキストなど)を新規作成/復元し、実行します

BPの仕組み#4/キュー

Interoperability > キュー

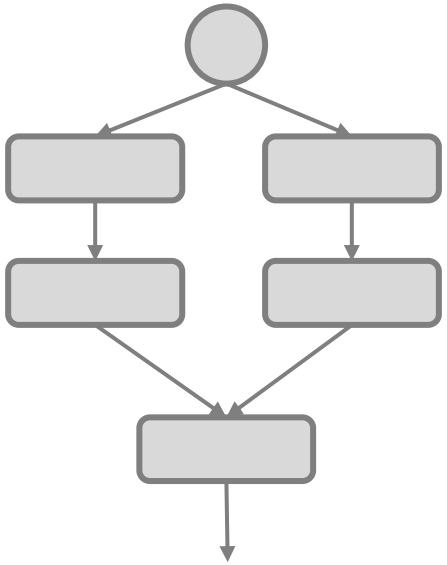
キュー　　更新:

フィルタ:　　結果: 17

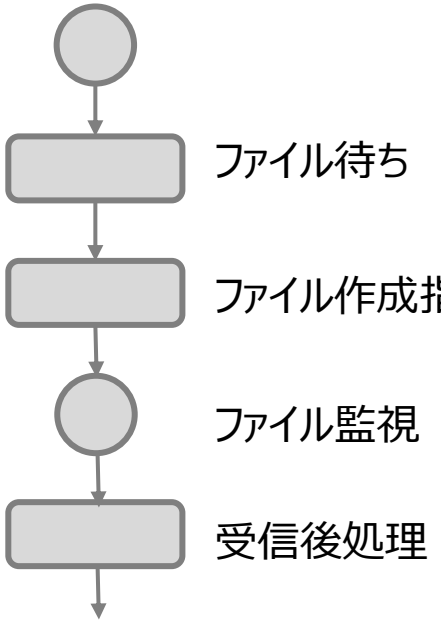
名前	カウント «	有効	作成日時
cWaitFolder	0	0	2023-06-20 15:48:00.375
cWaitFile	0	0	2023-06-20 15:48:00.371
cCallTask	0	0	2023-06-20 15:48:00.369
Task2_SFTP	0	0	2023-06-20 15:48:00.356
Task2_REST	0	0	2023-06-20 15:48:00.340
Task1_SFTP	0	0	2023-06-20 15:48:00.553
Task1_REST	0	0	2023-06-20 15:48:00.326
Task.Operation.DefaultAlert	0	0	2023-06-20 15:48:00.323
SFTP	0	0	2023-06-20 15:48:00.624
Operator	0	0	2023-06-20 15:48:00.318
EnsLib.Testing.Process	0	0	2023-06-20 15:48:00.312
EnsLib.EMail.AlertOperation	0	0	2023-06-20 15:48:00.312
Ens.ScheduleHandler	0	0	2023-06-20 15:48:00.306
Ens.Alert	0	0	2023-06-20 15:48:00.301
Ens.Alarm	0	0	2023-06-20 15:48:00.297
Ens.Actor	0	0	2023-06-20 15:48:00.293
Archive	0	0	2023-06-20 15:48:00.290

バッチジョブ管理に求められる要件

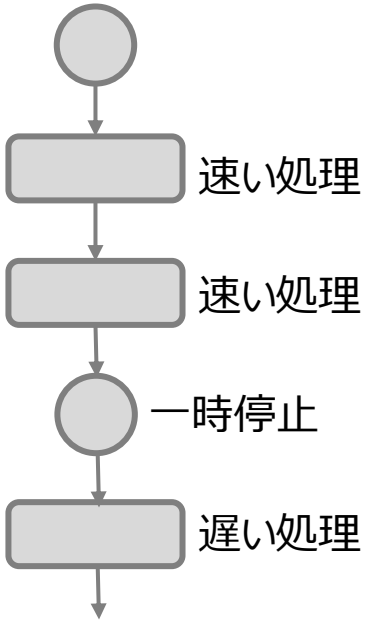
- 複数の処理を直列実行、もしくは並列実行する機能
- 複数の処理完了の応答を待ち合わせる機能
- いつ完了するか予測しづらい
- 失敗時のオペレータへの通知と再実行指示耐障害性



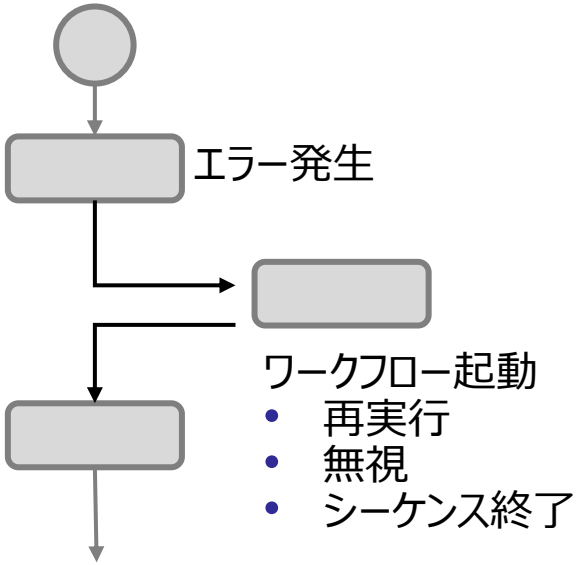
並列実行および完了待ち



ファイル受信を待つ



待ち方のバリエーション



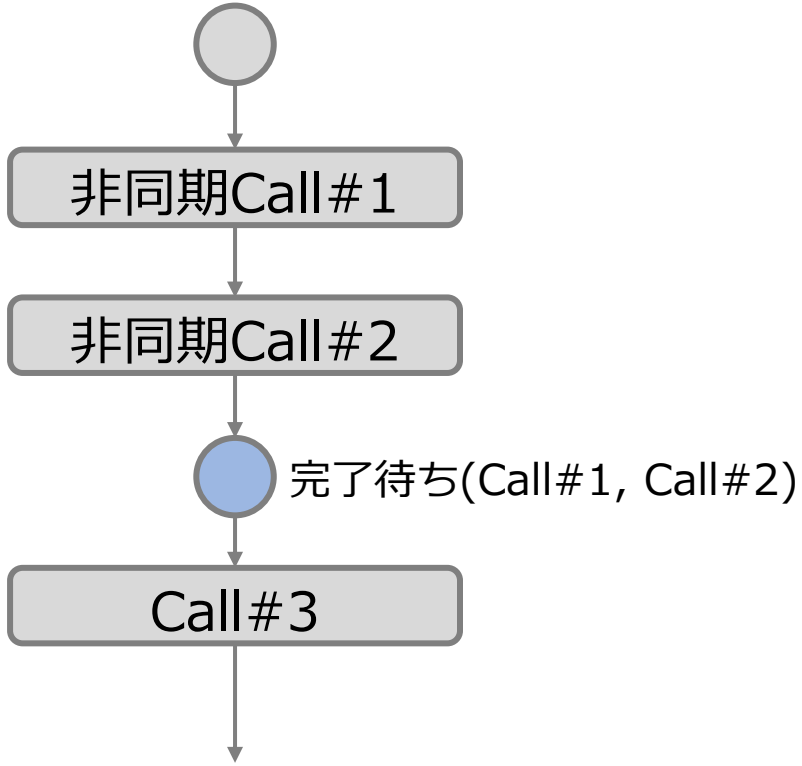
エラー発生時のワークフロー

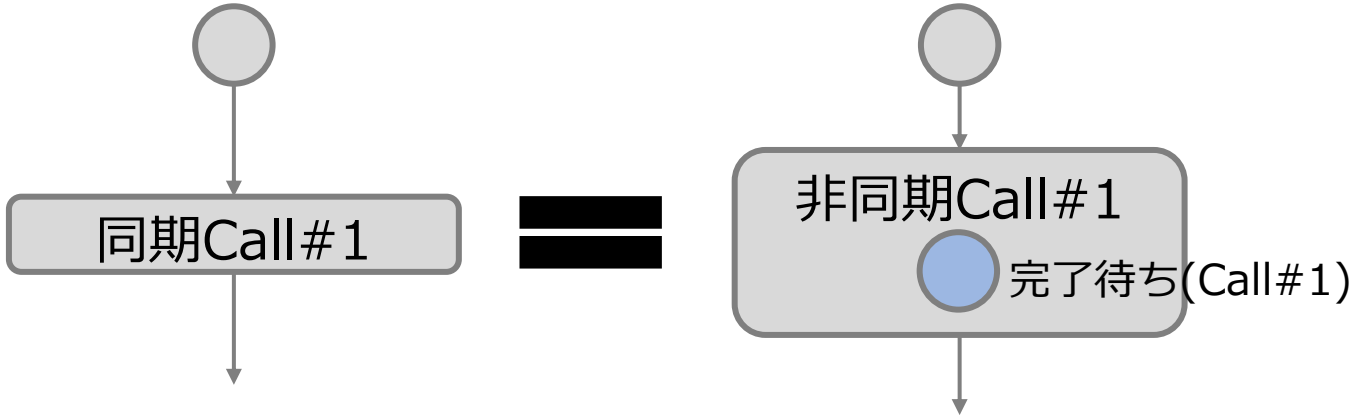
バッチジョブ管理に有用な機能

- 非同期コール
- 遅延送信
- InProc
- BPコンポーネント(モジュール的な用法)
- BPコンテキスト
- ワークフロー(遅延送信を使用した組み込み機能)

非同期コール

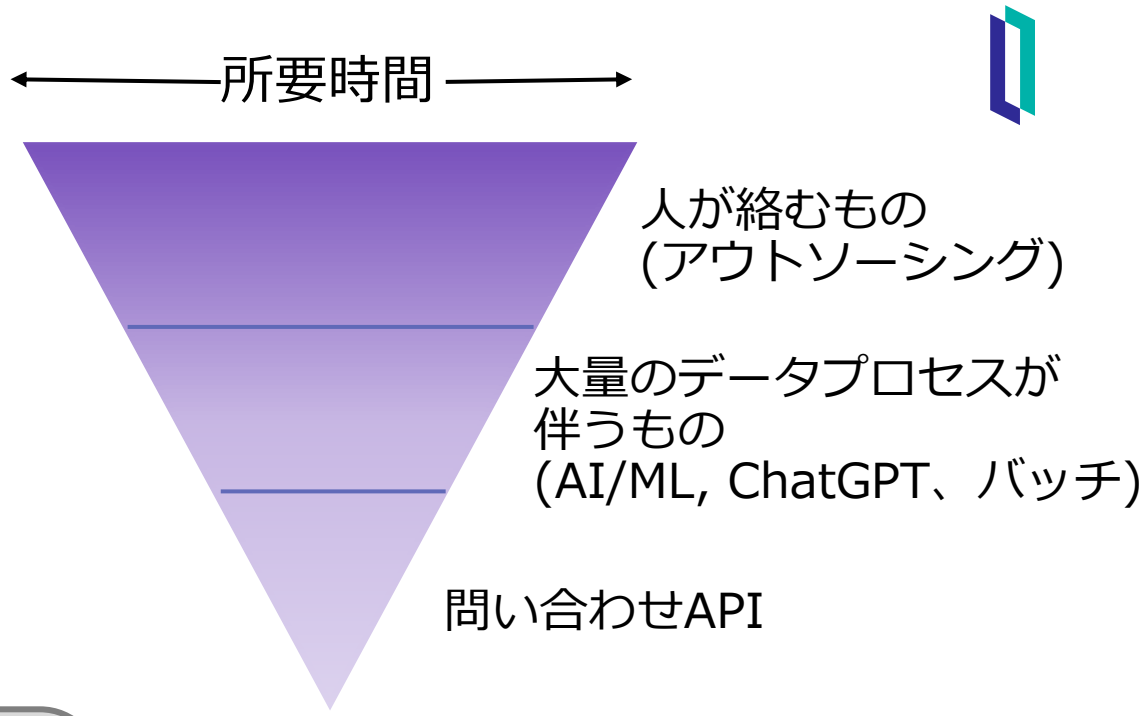
- 並列実行には不可欠です。
- BPが、Call先からの応答を待っている間に、次のアクティビティに進めるので、同時実行性が増します。
- BPLの同期コールは、実装上は非同期コール＋暗黙の完了待ちです。



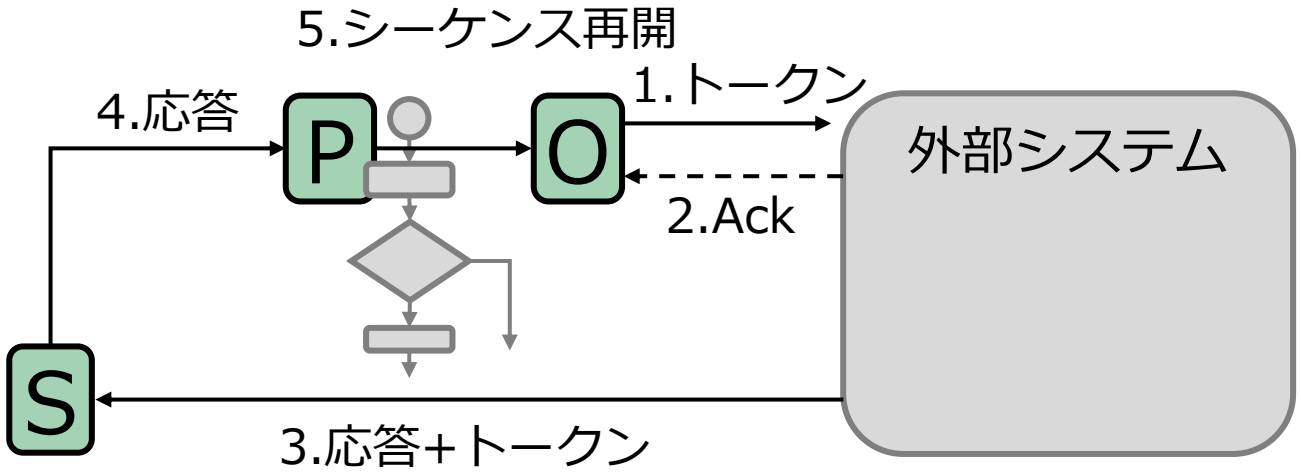


FIFOは保証されません

遅延送信



- オペレーションにかかる時間は様々
- 長時間のオペレーション実行は望ましくない
  - O/Sリソースの無駄使い
  - プロダクション停止時の「強制終了しますか？」
- 遅延送信

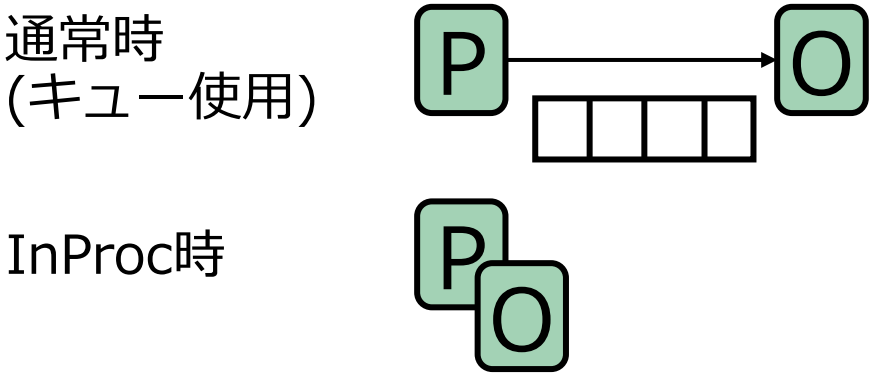


- 遅延応答となるイベントは、トークンを伝達できるものであれば何でも良い

mailのサブジェクトの一部, RESTのHTTP Header, WebServiceの引数...

InProc

- 外部との接続が無い(キューが不要な程度に安定性と即時性がある)場合に使用する仕組み
- ビジネスオペレーションをビジネスプロセス上で実行



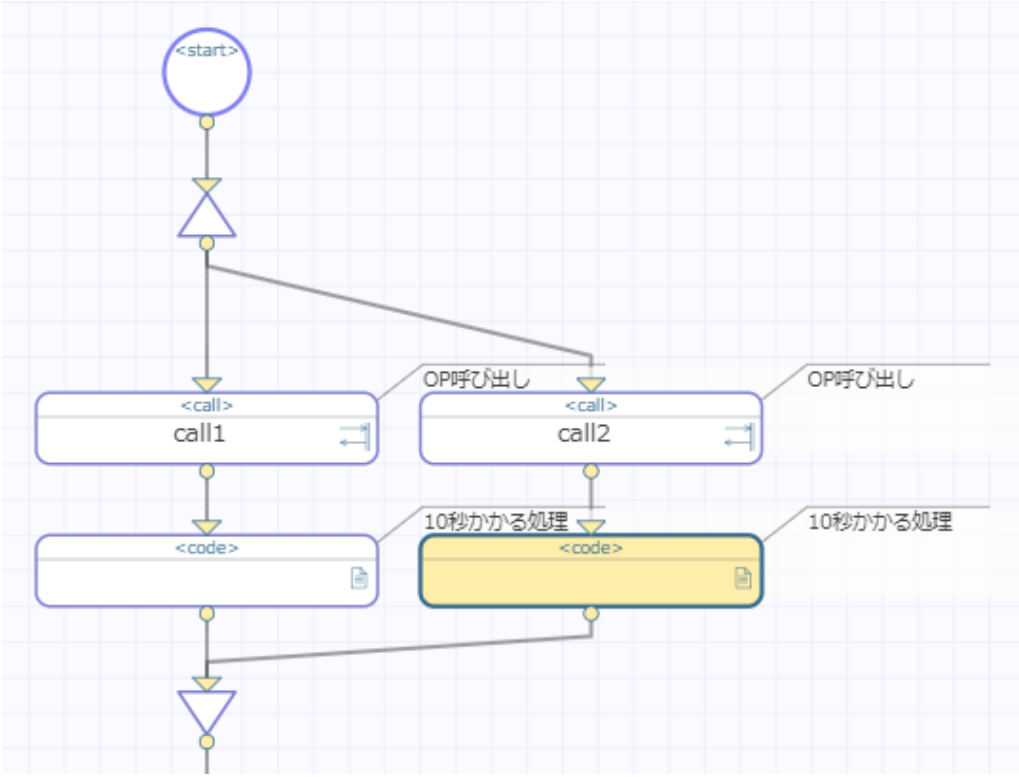
- 本例ではトークンの発行(BOのみが可能)とローカルテーブルへの保存に使用

BPコンポーネント

- 他のBPから呼ばれることを前提としたBP
  - エラー処理をコンポーネントで実現することで、繰り返し要素の出現を減らし、メインのBPの可読性を高める
  - 関数のモジュール化と同じ効果
- マルチコア環境であれば、アクティビティをパラレル実行出来る

- <flow>によるシーケンスの分岐とは異なる

<flow>が「良くない」という事ではありません。
こういう動きをする、ということです。



<flow>によるシーケンス分岐の例

BPコンテキスト

- BPの状態を保持するためのユーザ定義のオブジェクト
- BP毎に指定可能。
- BPのライフサイクルを通じて利用可能な任意の情報の保存領域
- 主に各コールの戻り値の一時保存のために使用するが、自由に使って良い
- 標準のアクティビティで表現できない(あるいは効率が悪い)ロジックを、<code>に記述する代わりに、メソッドとして実装して<code>から呼び出すのに最適な場所
- contextという予約変数でアクセスできます

<code>Do context.MyMethod(request)</code>

ワークフロー

- シーケンスの中に人を介在させる仕組み
- 本例では、エラーが発生したタスクの処理を
  - 再実行
  - (次のアクティビティから)継続実行
  - (BPでの処理そのものを)中止

  といった判断をオペレータに委ねるために使用

ワークフローについては、ウェビナーシリーズ
「ワークフローコンポーネントの使い方
～自動処理にユーザからの指示を統合する方法～ 」
をご覧ください。

FIFOについて

- メッセージの入力順と出力順が同じになる機構
- 処理の並列化(プロダクションでいうと、Pool>1を設定すること)を行うと、順序が入れ替わることがある。
- 医療系のシステムに見られる、リアルタイムにメッセージを転送するような用途では困ったことが起こります。

> 例. オーダとオーダキャンセルが、ほぼ同時に発生したような場合、順番が入れ替わると受信した側の最終状態が入れ替わる

- 処理するジョブがひとつであれば、このような現象は起きません。スループット向上のために、プロダクションに複数のキュー(プール)を用意し、それらを複数のジョブで処理する場合は注意が必要です。
- 同様の現象はキュー構造を持つどのようなソフトウェア(＊)も抱えており、その実現にはパフォーマンスの犠牲を伴います(アムダールの法則)。

（＊）Kafka, AMQP, MQTT, AWS SQS...などなど

FIFOについて

- 方法１

ドキュメントから抜粋

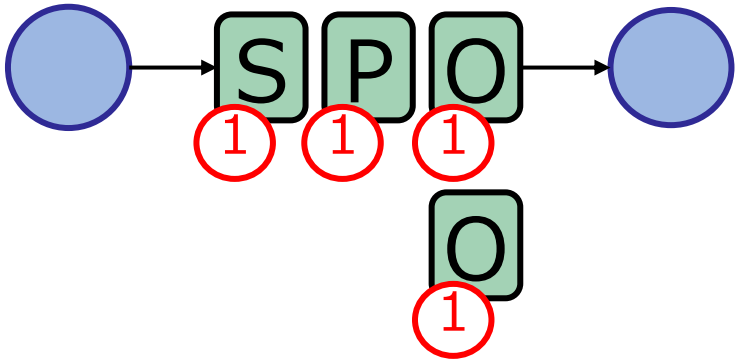
> ビジネス・プロセス・ロジックの条件分岐から呼び出しを行うことを避けて、かつ、それら自体が FIFO である要素の呼び出しのみを行います。

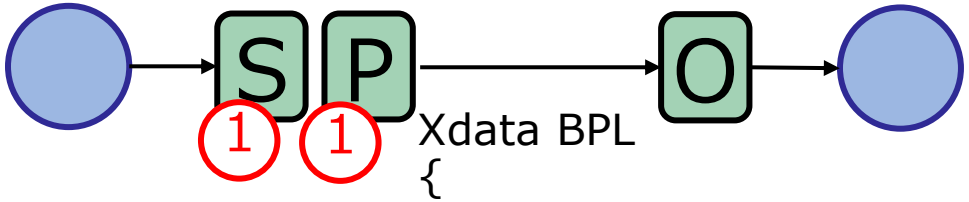
関連するBS,BP,BOのPool=1に設定し、かつBPのCALL実行は条件分岐無し、例えば入力別に、あるBOを呼んだり呼ばなかったり、ということがなければFIFOが維持される、ということです。

- 方法２

BS,BPのPool=1に設定し、<CODE>を使ってSendRequestSync() 呼び出しをする。

SendRequestSync()はジョブの解放を伴わず、呼び出し先からの応答を処理するまで、ジョブがそのBPの処理に占有される(他のメッセージの処理をしない)ため、FIFOが保たれる。






```
Xdata BPL
{
 <code>Do .. SendRequestSync()
</code>
}
```

FIFOについて

- 方法３

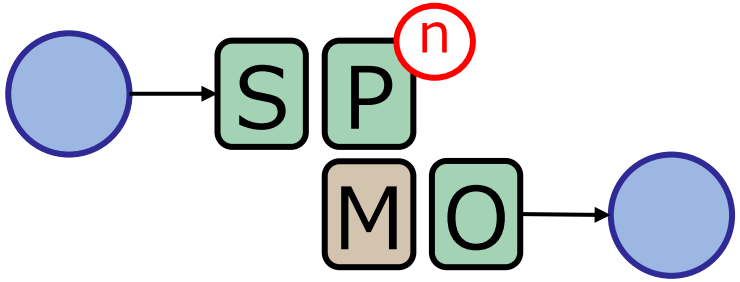
並列実行によって失われたFIFOを最終的に、制御するアプローチストアアンドフォワード方式。

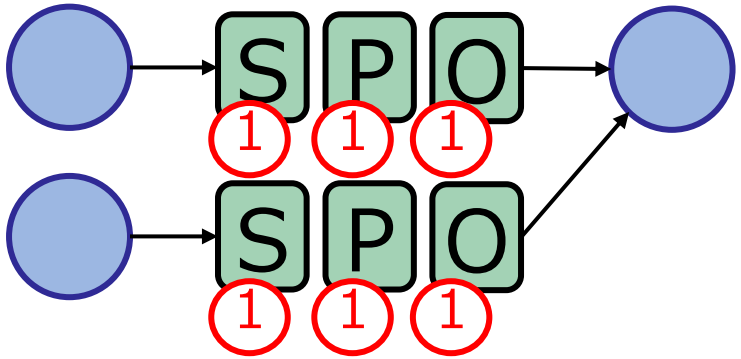
EnsLib.HL7.SequenceManagerはその例。

- 方法４

順番が入れ替わっても良いメッセージ群を分けて処理するために、BS/BP/BOのPool=1の組み合わせを複数用意する。

> 例えば、送信元の組織が複数あり、FIFOは同一組織からのメッセージ間でだけ保たれていればよい場合、同じ組織からの要求は同じBS/BP/BOのセットに振り分ける。





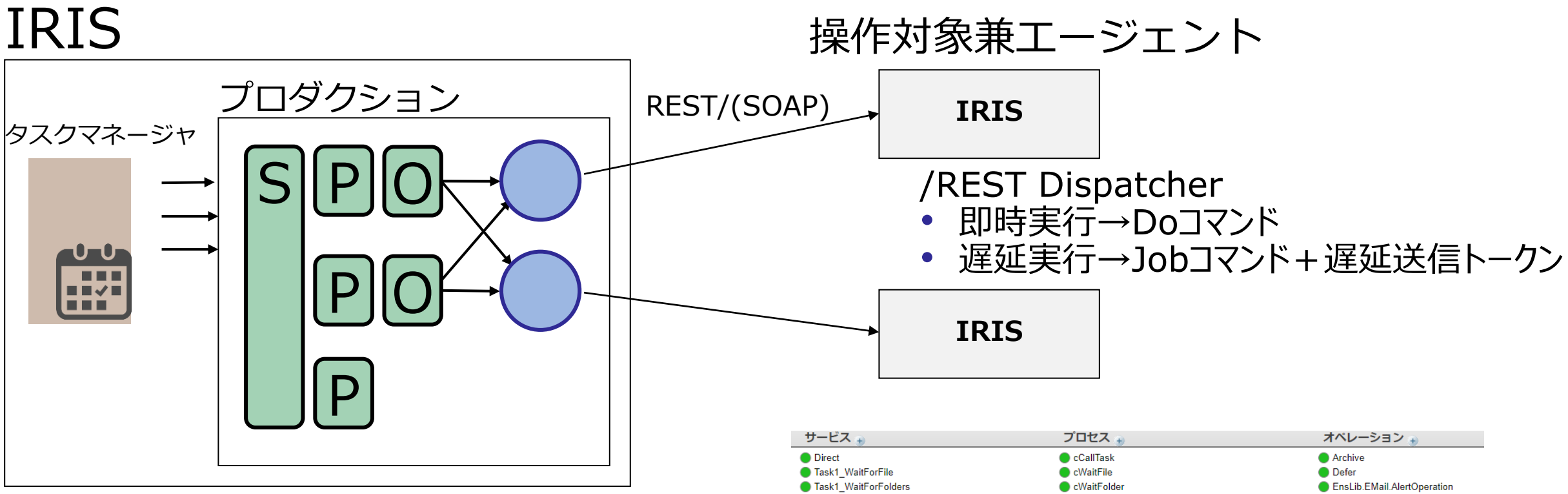
サンプル実装のご紹介

https://github.com/IRISMeister/jobmanagement

- 実現していること
  - 複数JOBの応答の待ち合わせ
  - エラー発生時のJOBの中止、再実行、無視(継続)の制御
  - ファイル受信待ち
  - フォルダ受信待ち
  - 遠隔(sftp)ファイル受信待ち
  - 遠隔(sftp) フォルダ受信待ち

  オンプレミス環境に加え、クラウドのオブジェクトストレージとの相性もよいため遠隔ファイル操作にはsftpを採用

バッチジョブ管理への適用



サービス	プロセス	オペレーション
Direct	cCallTask	Archive
Task1_WaitForFile	cWaitFile	Defer
Task1_WaitForFolders	cWaitFolder	EnsLib.EMail.AlertOperation
Task2_WaitForFile	Ens.Alert	GeneralWS
Task2_WaitForFolders	Ens.Alerting.NotificationManager	Operator
TaskComplete	job01	SFTP
TaskCompleteWS	job02	Task.Operation.DefaultAlert
WaitForFile	job02a	Task1_REST
WaitForFolders	job02b	Task1_SFTP
	job03	Task2_REST
	job04	Task2_SFTP
	job05File	
	job06Files	
	job07Folders	
	job08RemoteFile	
	job09RemoteFiles	
	job10RemoteFolders	

サンプル実装のご紹介(続き)

job01

起動方法

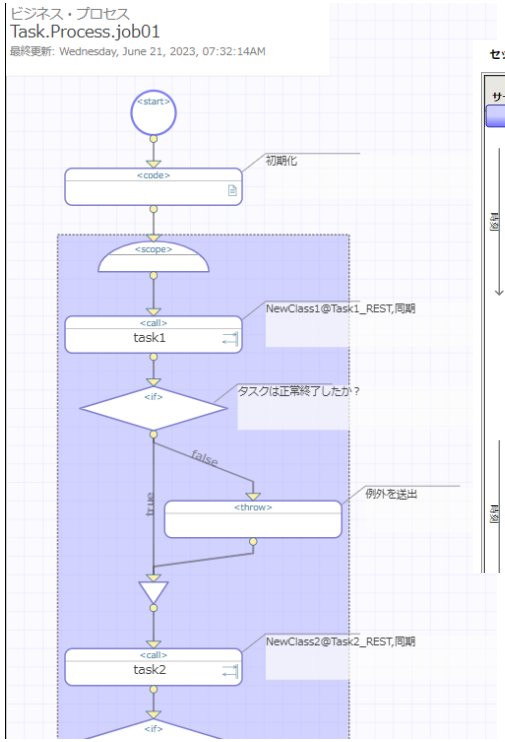
job01は、コンテナ起動直後よりタスクマネージャ->BP/Direct経由で5分間隔で自動実行されています。

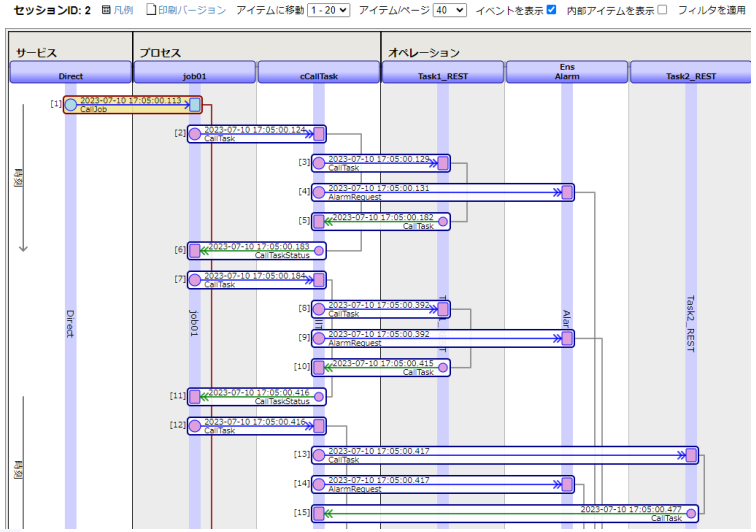
処理内容

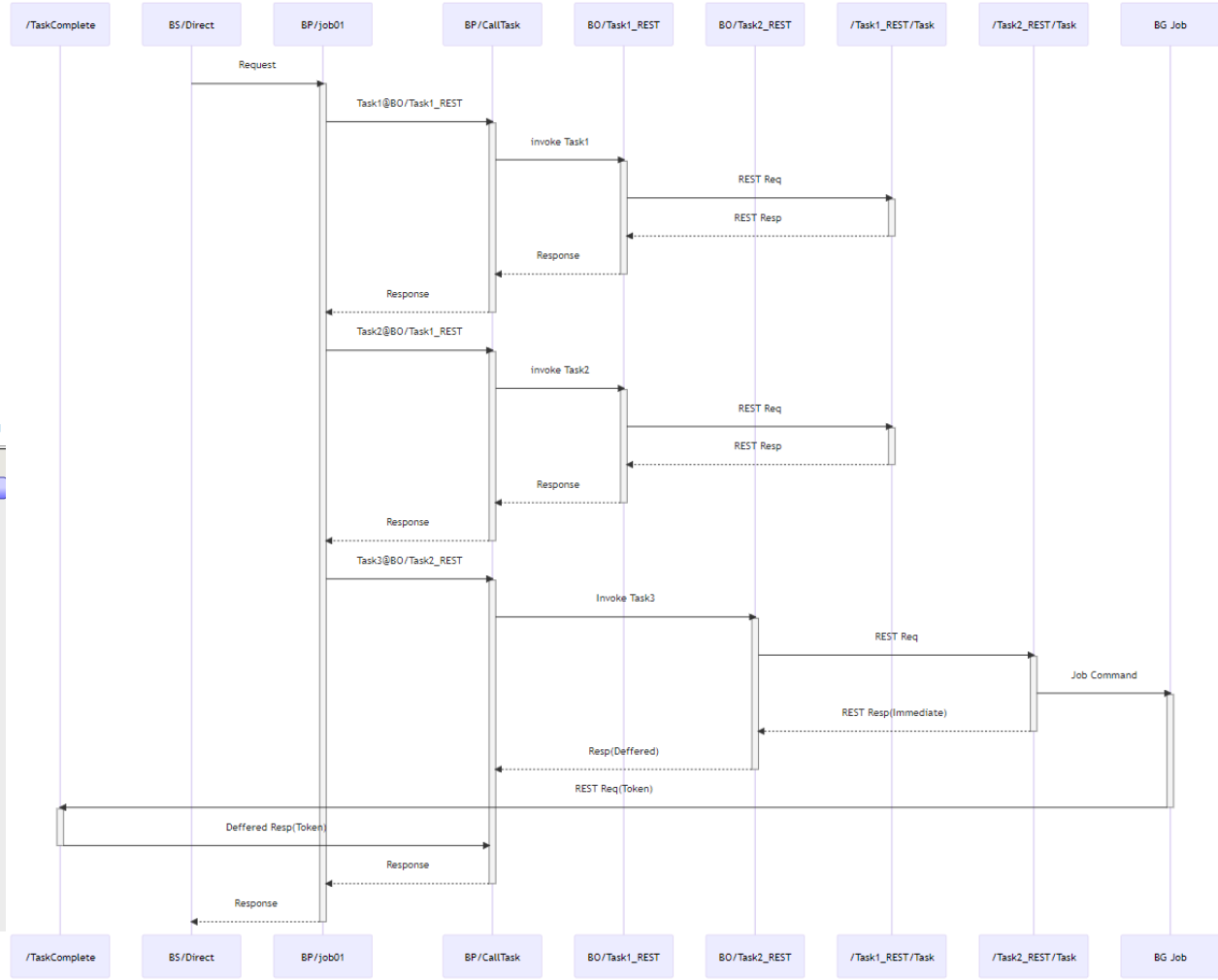
BP/CallTask経由でtask1(ターゲット:BO/Task1_REST,MyTask.NewClass1), task2(ターゲット:BO/Task1_REST,MyTask.NewClass2), task3(ターゲット:BO/Task2_REST,MyTask.NewClass3)を順番に同期実行。ただし、task3だけは遅延実行(taskインスタンス上でのタスクをJOBコマンドで実行し、遅延応答(トークン)を返却します)を行っています。

BO/Task1_RESTはRESTクライアントを使用して、IRISサーバ#1のRESTサービスを起動します。 その結果、IRISサーバ#1ではMyTask.NewClass1とMyTask.NewClass2が、各々実行されます。その動作結果はグローバルに保存されます。

BO/Task2_RESTはRESTクライアントを使用して、IRISサーバ#2のRESTサービスを起動します。 その結果、IRISサーバ#2ではMyTask.NewClass3が実行されます。その動作結果はグローバルに保存されます。







ふり返り

1	**動機**
2	**プロダクションとは**
3	**ビジネスプロセスとは**
4	**バッチジョブ管理の特徴**
5	**利用した機能**
6	**FIFO**
7	**実装コードのご案内**