# オブジェクト指向ペトリネットとルールに基づく業務プロセスの理解支援

# Support for Understanding Business Processes based on Object-oriented Petri Nets and Business Rules

飯島 正 †, 秦 良平 †, 金子 良太 ‡
Tadashi IIJIMA†, Ryohei HATA, and Ryouta KANEKO‡
† 慶應義塾大学　理工学部
‡ 慶應義塾大学大学院　理工学研究科
†Faculty of Science and Technology, Keio Univ.
‡Graduate School of Science and Technology, Keio Univ.

**要旨**

近年，**業務プロセス管理 (BPM; Business Process Management)** と並んで，**業務ルール管理 (BRM; Business Rule Management)** が重視されている．本研究は，業務ルールをとり入れた業務プロセスの理解性 (understandability) 向上を目指すものである．そのために，著者らの研究グループは，まず第一に，組織間にまたがるワークフロー (inter-organizatonal workflow) における組織間の協調と相互のやり取りを扱うこと，ならびに，アクター (実行者/サブシステム) を資源として扱うこと等を目的に，業務プロセスの表記法としてオブジェクト指向ペトリネット (Object-oriented Petri Net) を導入した．次に，そこに業務ルール記述を導入し，不完全なプロセスの部分的な補完や，不適切なプロセスの部分的な変換を扱ってきた．本報告では，上記に加えて新たに導入した，(1) 大規模な業務プロセスの理解性向上のための段階的詳細化 (Step-wise Refinement) と**リファクタリング (Refactoring)**，ならびに，(2) 業務ルールの理解性向上のための DSL(Domain Specific Language; ドメイン固有言語もしくはドメイン特化言語) 表現，の二点に関して述べる．

## 1. はじめに

近年，**業務プロセス管理 (BPM; Business Process Management)** と並んで，**業務ルール管理 (BRM; Business Rule Management)** が重視されている．本研究は，**業務ルールをとり入れた業務プロセスの理解性 (understandability) 向上**を目指すものである．そのために，著者らの研究グループは，まず第一に，**組織間にまたがるワークフロー (inter-organizatonal workflow)** における組織間の協調と相互のやり取りを扱うこと，ならびに，アクター (実行者/サブシステム) を資源として扱うこと等を目的に，業務プロセスの表記法としてオブジェクト指向ペトリネット (Object-oriented Petri Net) を導入した．次に，そこに業務ルール記述を導入し，不完全なプロセスの部分的な補完や，不適切なプロセスの部分的な変換を扱ってきた．本報告では，上記に加えて，さらに導入した，

(1) 大規模な業務プロセスの理解性向上のために導入した**段階的詳細化**と**リファクタリング**
(2) 業務ルールの理解性向上のために導入した **DSL (Domain Specific Language; ドメイン固有言語もしくはドメイン特化言語) 表現** [4, 5]

の二点に関して述べる．

続く第 2 節では業務プロセス管理と業務ルール管理の関係，さらに第 3 節ではオブジェクト指向ペトリネットに関して記述した後，第 4 節で段階的詳細化による大規模な業務プロセスの理解性向上，第 5 節で DSL による業務ルールの理解性向上に関して述べる．

## 2. 業務プロセス管理と業務ルール管理

業務の進捗を把握し，必要に応じて (人的資源を含む) リソースの再配分やワークフローの再調整をするためには，まず業務ロジックのプロセス面を定義しなければならない．そこでは，その直感性の高さや，曖昧な段階から記述を開始できる点から，BPMN 等の業務プロセス記述（業務フロー，業務プロセス表現）が一般に使われることが多い．一方で，業務ロジックは，ルールの形で記述し，ルールエンジンを使って実行することもできる．しかし，業務ロジックの全体をルールの形で記述すると，プロセス表現と比較して，却って見通しが悪くなって全体像を把握しにくくしたり，イメージしにくくすることも多い．もちろん，局所的にはプロセス表現よりも，分かり易く表現できるものもありうる．たとえば業務ポリシーにあたるようなものなどは，ルール表現の方が書きやすいことが多い．そこで，折衷案として，基本的なプロセス表現に加えて，ルール表現を適切な部分にのみ補助的に併用して，うまく使い分けることができれば一番都合がよい．プロセス表現に対し，あえて別種の表現であるルール表現を併用することのメリットを，まとめると以下の 2 点となる [7].

(1) 不完全なプロセス表現の一部を**補完**したいとき → 補完部分をルール表現で記述する
(2) 不適切なプロセス表現の一部を**変換**したいとき → 変換の目標や変換方法をルール表現で記述する

これらのメリットを活かすルール表現の使い方として，具体的には，以下のような用途がありうる [7].

(1-1) パラメータを外部化 · · · プロセス中に埋め込まれてしまうパラメータを外部化する
(2-1) フローの構造変換 · · · 適応的な例外処理, 横断する関心事に関する処理を記述し挿入する
(2-2) リソースの再配分 · · · 進捗が遅れているタスクや，外部要因に依ってそのままでは実行可能ではなくなった場合でフローを変更できないとき，(人的資源を含む) リソースを再調整し，できるだけ業務ポリシー (制約) を遵守ように変更する

これらのうち，別途，(2-1) は [7] で，(2-2) は [8] で扱っているので，ここで詳細に触れることは省き，第 5 章において (1-1) への DSL の導入に関して述べる．

## 3. オブジェクト指向ペトリネット

### 3.1. オブジェクト指向ペトリネットとは

本研究では，業務プロセスの表現としてオブジェクト指向ペトリネットを用いている．オブジェクト指向ペトリネットとはオブジェクト指向概念に基づいてモジュール性を取り入れたペトリネットの拡張モデルであり，多くのものが提案されている．ここでは，nets-in-nets 意味論に基づく参照ネット (Reference Net) [6] の考え方を採用したオブジェクト指向ペトリネットを採用している．ペトリネットで業務プロセスを記述する先行事例に YAWL [1] があるが，本研究グループの提案は，業務プロセス記述に，いわゆるペトリネットではなく，オブジェクト指向ペトリネットを利用している点に特徴がある．

本研究で，オブジェクト指向ペトリネットを導入した理由には，以下のようなものがある:
(1) 組織間にまたがるワークフロー (inter-organizatonal workflow) における組織間の協調を扱うため
(2) アクター (実行者/サブシステム) を資源として扱うため

以下では，それらの目的に関連付けて，簡潔に位置づける．

#### 3.1.1. 組織間にまたがるワークフロー

一つの業務フローは，一部門/一組織内の一つの業務内のローカルな，業務アクティビティ(作業，タスク)，ないし，呼び出される外部サービスの間の制御フロー (すなわち業務ロジック) を表現する．すなわち，アクティビティやサービスの**オーケストレーション (Orchestration)** を規定するものと位置づけられる．そうしたオーケストレーションに対し，業務間 (部門間/組織間) のメッセージのやり取りに規定するものを一般に**コレオグラフィ (Choreography)** と呼ぶことがある．この二つの区別は概念的には理解できるが，現実の業務を考えた場合には，境界を厳密に規定することは難しい．組織，部門，担当者というように細分化していくと，これらの境界をどこに設けるかは，ケースによって異なる．

UML の**アクティビティ図**や，**BPMN** [2, 3] は，いわゆる業務フロー図であるが，そこに，さらに，スイムレーン (swimlane) を導入し，業務担当者 (担当部署/部門もしくは組織) ごとに業務フローを分離し，それらの間の相互作用を明確にしている．これによって，各スイムレーン内でオーケストレーションを表現し，スイムレーン間でコレオグラフィを表現することができる．

オブジェクト指向ペトリネットでは，業務担当者 (担当部署/部門もしくは組織) ごとにペトリネットの形で業務プロセス（オーケストレーション）を表現することができる．さらに，それらを協調させるペトリネット（ここではシステムネットと呼ぶ）を導入したり，業務担当者 (担当部署/部門もしくは組織) の間でやり取りするメッセージを表現するペトリネットを導入したりすることが可能である．これによって，同じくペトリネットの形で，コレオグラフィをも表現させる点に特徴がある．

#### 3.1.2. アクターを資源として扱う

参照ネットにおいては，トークンには 2 通りある．一つは，通常の P/T ネット (Place/Transition ネット) におけるトークンである単純トークン (Black Token) とよぶ．プレースに存在する単純トークンは，トークン数で示す．もう一つは，別のサブシステムを表現するサブネットへの参照 (reference) に相当する参照トークン (Reference Token) である．

プレースは，単純トークン用の単純プレースと，参照トークン用の参照プレースに分類できる．

- 単純トークン (Black Token)[1]
- 参照トークン (Reference Token)

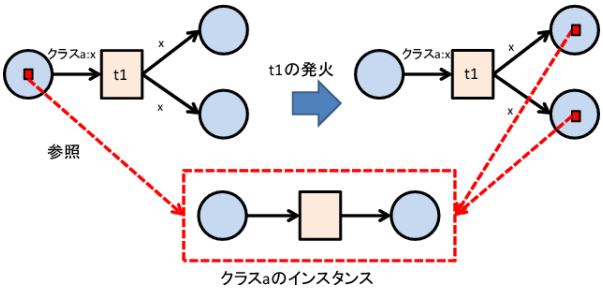

図 1: 参照トークンの例

ここで，参照トークンが参照しているサブネットは，オブジェクト・ネットと呼ばれ，一つのオブジェクトとしてのアイデンティティをもつ．参照トークンも，単純トークンと同様，ネットワーク中を遷移することができるが，一つの参照トークンがトランジションの発火によってコピーされることもある．その際には，あくまで共通のオブジェクトネットへの参照がコピーされる点に注意されたい．

後述する (図 2 左) ように，各トランジションの発火の際には，そのトランジションと同期する (interaction 関係にある) ペトリネットにおいては，同名のメソッドが実行される．すなわち，そうした同期関係 (interaction 関係にある) ペトリネットが実行主体 (アクター) として位置づけられることになる．受け渡しされるパラメータは，基本的に，そのトランジションへの入力アーク上の weight で表現される．それに加えて，同期関係にある他のネットから受け渡される参照が付け加えられる．それらは，?(入力パラメータ) ならびに !(出力パラメータ) として受け渡される (入力も出力もいずれもオブジェクトへの参照である)．

アクター (実行主体) に相当するオブジェクト・ペトリネットはモデルの中で new 演算子によって生成され，その参照は参照トークンとして扱われるため，モデル中で「資源 (resource)」として取り扱うことができる (図 2 右)．センサーに対応するペトリネットと同期して，オブジェクト・ペトリネットを生成することもある．

### 3.2. オブジェクト指向ペトリネット OPeN

以下では，具体的なオブジェクト指向ペトリネットのモデルとして，筆者の主宰する研究室で継続的に開発を行ってきたオブジェクト指向ペトリネット OPeN ファミリー (the Object-oriented Petri Net family) の中でも，特に人と人の連携して行う協調作業や，SOA におけるサービス間連携を記述するためのビジネスモデル記述 (ワークフロー) に特化した OPeN/WF を用いている．OPeN はオブジェクト指向ペトリネットの一つの基本モデル (メタモデル) であり，それをベースに多様な用途に応用すべく特化した応用モデル (プロファイル) である[2]．

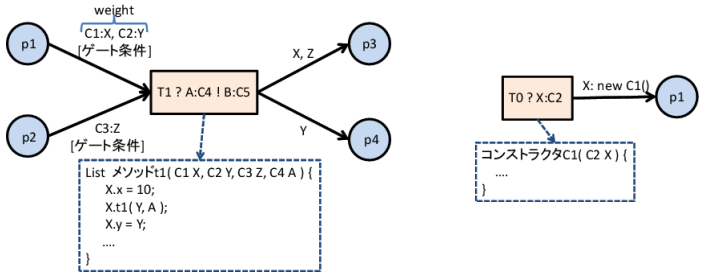

図 2: OPeN におけるアークとトランジション

OPeN では，個々のオブジェクト・ネットは，クラス記述から生成されるインスタンスに相当する．クラス記述は，オブジェクトを定義するものであり，本稿執筆時点では，opn(もしくは従来からの慣習から pnmlx) という拡張子を持つ XML ファイルとして記述する．クラスが定義するオブジェクト・ネットは，そのオブジェクトの振る舞い (ライフサイクル) を記述する一つのペトリネット，そのオブジェクトが持つプロパティ (インスタンス変数) 群と，ライフサイクルを遷移する際に使われるメソッド群である．メソッドは，主に，トランジションの発火条件であるゲート条件の記述や，トランジション発火時に起動されてプロパティの値の更新に使われる．図 2 に OPeN におけるアークとトランジションのアノテーションを示す．

トランジション $t_1$ の発火によって，同じ名前をもつメソッド t1 が起動される[3]．現時点のプロトタイ

[1]慣習的にブラック・トークンと呼ばれるが，ここでは分かり易さのために単純トークンと呼ぶ

[2]正確にはメタモデルアーキテクチャによる定義は未完成であり，基本モデルとしての表記法と意味論を共通とする応用モデル群がツール群を共有している段階である．

[3]後述するオブジェクト指向ペトリネットの発火則「Interaction(相互作用)」により，$t_1$ と同期するトランジション $t_{s1}$ が，このペトリネットのサブネット中に存在し，かつそれが発火可能であれば，その $t_{s1}$ も同時に発火する．$t_{s1}$ が発火可能でなければ $t_1$ も発火できない．トランジションの入力アークがメソッドの入力パラメータに対応する．

プ仕様では，プロパティの型やメソッドの定義のための言語仕様は，厳密には定義していないが，図 2 に準拠し，基本的なリフレクション機能を備えた一般的なオブジェクト指向言語 (たとえば Java や Scala) のクラス定義 (ネットと同名のクラス定義) に外部化しており，相互に対応付けている．これは，現時点での OPeN は，いろいろな応用先でのモデル記述言語としての表現力を確認している段階であるためである (そのために多様な応用先に組み込んで使用し表現力を試すことを優先しているが，その際に，分散並行システム記述としてのペトリネットのセマンティックスを忠実に実現してはいない．オブジェクト・ネットのインスタンス化の際には，その対応する言語のオブジェクトもインスタンス化しライフサイクル上の遷移に応じてメソッドを起動する．同時発火可能なトランジションはランダムに選択した順序で逐次的に，その動作をシミュレートしている)．

複数のオブジェクト・ネットから構成されるシステムは階層的に表現されるが，これには，プロジェクトという単位を用いる．しかし，ここでは，まず分かり易さを優先して多階層の構造ではなく，2 階層の構造に限定して，その意味論を概念的に説明する．これは，[6] における EOS(初等オブジェクトシステム) に相当する．2 階層の構造では，全体的な振る舞いを統制するオブジェクト (ペトリネット) と，それに規定された振る舞いを行うサブオブジェクト (ペトリネット) 群から構成される．ここでは，全体的な統制を記述する方を EOS の慣習に従いシステム・ネットと呼び，統制されたサブオブジェクトの側をオブジェクト・ネットと呼んで説明することにする[4]．

システム・ネットとオブジェクト・ネットの間では，一部のトランジションの発火が同期的に行われる (それによって，システム・ネットはオブジェクト・ネットの動作をオーケストレートする)．システム・ネットとオブジェクト・ネットの同期関係は，双方のトランジションの対の集合として表現される．システム・ネットにおいて，あるトランジション $T$ が発火するためには，以下の三条件が成り立たねばならない[5]．

(a) システム・ネットにおいて，単純トークン $B_i$ に関して発火条件を満たしていること，
(b) システム・ネットにおいて，参照トークン $R_j$ に関して発火条件を満たしていること，
(c) $T$ の発火に寄与している参照トークン $R_j$ が参照しているオブジェクト・ネット中で，$T$ と同期関係にあるトランジションが発火可能であること．

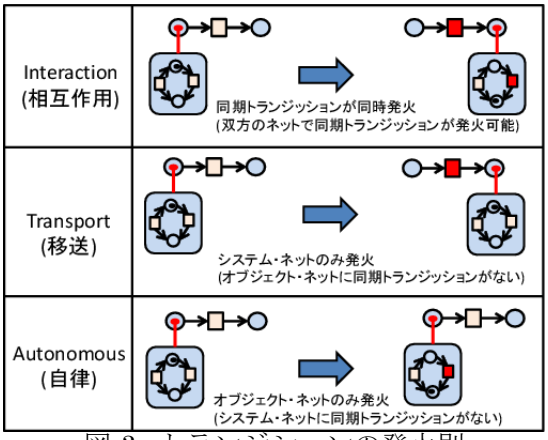

図 3: トランジションの発火則

こうした発火のメカニズムから，nets-within-nets 意味論 (ないし参照意味論)，およびその拡張に基づくオブジェクト指向ペトリネットにおいては，(a) interaction(相互作用)，(b) transport(移送)，(c) autonomous(自律)，という 3 通りの発火則が考えられる．まず，システム・ネットとオブジェクト・ネットが同期して遷移するタイプであり，interaction(相互作用) と呼ばれる．次にシステム・ネットの発火に際し同期して発火するオブジェクト・ネットのないタイプがあり，transport(移送) と呼ばれる．これは，オブジェクト・ネット内部には何の状態遷移も起こっていないまま，システム・ネット上にある参照トークンが移動する様子が，モバイル・エージェントの移動に対応づけられることから名づけられている．最後に，システム・ネットとは独立に，オブジェクト・ネット内のトランジションが発火するものであり，autonomous(自律) と呼ばれる．その呼称は，システム・ネットに制御されることなく，オブジェクト・ネット内部で自律的にトークンの遷移（すなわちそのオブジェクト・ネットの状態遷移）が引き起こされていることに由来する (図 3 )．

[4]階層的なシステムの場合には，ここでいうオブジェクト・ネットもまた，より下位のサブペトリネットからみれば相対的にシステム・ネットに相当する．OPeN は EOS の概念を多階層に拡張しているが，混乱のない限り，最上位ネットに限らず，隣り合う階層のネット間の関係を，システムネットとオブジェクト・ネットというように称することとする

[5]但し，OPeN の場合には，二階層の EOS ではなく，多階層を許容しているので，条件 (c) は再帰的に，より下位層にあるオブジェクト・ネットに対して適用されていくことになる．

## 4. 段階的詳細化とリファクタリングの導入による業務プロセスの理解性向上

一般には，多くの業務プロセスは，画面一枚に収まる規模ではなく，その全体を一度に把握するのは困難である．それは，既に設計された業務プロセスを理解するときばかりではなく，新規にプロセス設計時にも影響する．そこで，モジュール性を導入することが不可欠である．オブジェクト指向ペトリネットの利用は，アクター毎に分解することで，そうしたモジュール性を導入する試みの一環ともいえるが，大規模な制御フローに対しては，必ずしも有効ではない．

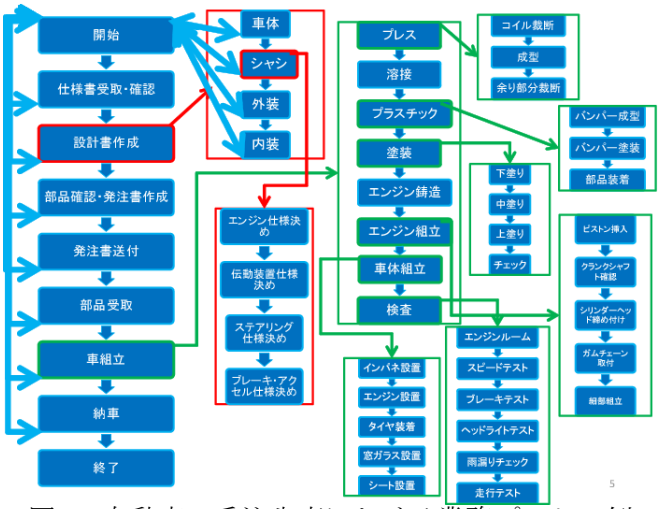


図 4: 自動車の受注生産における業務プロセス例

たとえば，自動車の受注生産における業務プロセスは複雑で大規模なものとなる．図 4 は，[9] を参考に作成した業務プロセスの一部である．ここでは，設計書作成，車組立といった一部のアクティビティは，さらに，サブアクティビティとして表現しており，そうした入れ子構造は階層的に繰り返されている．こうした階層構造は，**段階的詳細化 (stepwise refinement)** という形で，設計時に導入される．

そこで，その段階的詳細化の過程を，オブジェクト指向ペトリネットのエディタに導入した (図 5)．トランジションを詳細化して，別のビューでサブネットとして記述することができる．すなわち，詳細を省いた抽象度の高いネットから，段階的に詳細化していくことができる．このエディタは，シミュレータと連動しており，詳細を省いた抽象度の高いレベルのままで，実行経過を眺めることもできる．

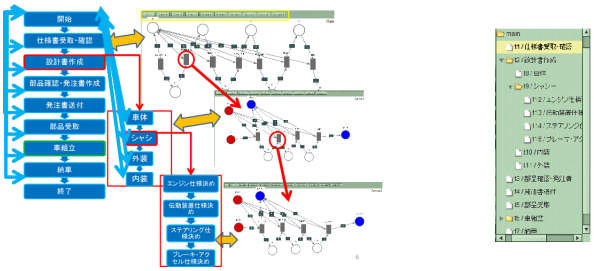

図 5: 段階的詳細化エディタ/シミュレータ

段階的詳細化の過程は，設計時の階層構造として XML 表現として内部的に管理しており，それに木構造状のインタフェース (図 5 右) でアクセスすることができる．編集を繰り返していくと，必ずしも当初の段階的詳細化の過程が，適切な階層構造から遊離していくことがありうる．そこで，オブジェクト指向ペトリネットエディタには，さらに展開 (unfolding)，再帰的展開 (allunfolding)，畳み込み (folding) という 3 つの意味保存変換を導入し，階層構造のリファクタリングが行えるようにしている．

## 5.DSL の導入による業務ルールの理解性向上

業務ルールとは，業務が満たすべき要件や規約のことである．たとえば，航空券のインターネット購入の場合には，以下のような業務ルールが考えられる:

(1) インターネットで航空券を予約した場合，3 日以内に購入手続きを完了しなければならない
(購入手続きを完了しなかった場合，予約が取り消される)
(2) 航空券の払い戻しには，運賃の 50%の手数料が発生する

一般に，ある業務プロセスの中で，オブジェクトのプロパティの値 (たとえば，航空券の代金) は，各種の割引や，時限的なキャンペーンなどにより，業務プロセス全体よりも短いサイクルで変更されがちである．そうした変更に迅速かつ低コストで追従することは重要である．そうした値の決定は，決定木ないし決定表の形で表現されることも多いが，それは容易にルール形式へ変換することができる．そうしたパラメータの値を決定する業務ルールを，ルールとして外部化することは，保守性を向上させる上で重要である．通常，業務システムの改修作業のためにはシステム開発の専門家の手を煩わせる必要があるが，そうしたルールがシステム開発の専門家ではないドメイン専門家にとっても扱いやすく書き換

えが可能なものであるならば，その必要がなくなる．

そこで，(a) トランジションの発火と同時に起動されるメソッドから，ルールエンジンを呼び出す機能を提供する．それによって，たとえば上記の例のような航空券の運賃計算を，ルールで扱うことが可能となり，プロセスから外部化させることができるようになる．

さらに，ルールの中で，特定ドメイン（ここでは運賃計算）に専門家にとって，日常的に使用する表現を導入することができればなお保守性が高まる．そこで，(b) ルール中で使用できる DSL(Domain Specific Language; ドメイン固有言語もしくはドメイン特化言語) を導入する．

今回は，プログラミング言語 Java と同じ Java 仮想機械 (Virtual Machine) のバイトコードにコンパイルされる Scala 言語を用いることで，上記 (a)(b) の 2 点を実現した．Scala は，いわゆる内部 DSL を構築しやすい言語の一つとして知られており [4]，Scala で実装されたオープンソースのルールエンジン hammurabi[10] を利用することができる (実際には，一部のキーワードに日本語が使えるように手を加えている)．実際の業務ルールの記述例を下記に示す．

**業務ルール例 (1)：東京福岡便の基本運賃は 36000 円で付与ポイントは 600pt**

```
ルール ("福岡便の基本運賃とポイント") は {
    val customer = any(kindOf[Customer])
    もし {
        route of customer is "福岡"
    } ならば {
        fare of customer is 36000
        point of customer increase 600
    }
}
```

**業務ルール例 (2)：年齢が満 12 歳未満の乗客の運賃割引率**

```
ルール ("12 歳未満の乗客の運賃割引") は {
    val customer = any(kindOf[Customer])
    もし {
        age of customer isLowerThan 12
    } ならば {
        discountRate of customer is 0.5
    }
}
```

## 6. まとめ

オブジェクト指向ペトリネットによる業務プロセス表現に対して，(1) 段階的詳細化 (Step-wise Refinement) と**リファクタリング (Refactoring)**，ならびに (2) DSL 表現による業務ルールを導入した．これにより，業務プロセスの理解性を向上させることができた．

## 謝辞

本研究は，当研究室・Net 班 (業務プロセスグループ) による蓄積の上に成り立っている．同班の卒業生 (片山 輝彦 (2012 年度修士了)，高橋 貴大 (2012 年度学部卒)，新間 理貴 (2010 年度修士了)，孫 騰涛 (2009 年度修士了)，塚原 浩太 (2009 年度学部卒)) 各位に感謝します．

## 参考文献

[1] W.M.P. van der Aalst and A.H.M. ter Hofstede: “YAWL: Yet Another Workflow Language,” QUT Technical Report, FIT-TR-2002-06, Queensland University of Technology, Brisbane, 2002.
[2] OMG: “Business Process Model And Notation (BPMN), Version 2.0,” formal/2011-01-03/, Release Date: January 2011, http://www.omg.org/spec/BPMN/2.0/PDF
[3] Bruce Silver, Bruce Silver, (監訳) 岩田 アキラ : “BPMN メソッド & スタイル 〜 第 2 版 BPMN 実装者向けガイド付き,” 日本ビジネスプロセス・マネジメント協会, 2013.
[4] Debasish Ghosh, (監訳) 佐藤 竜一: “実践プログラミング DSL 〜 ドメイン特化言語の設計と実装のノウハウ,” 翔泳社, 2012.
[5] Martin Fowler, ウルシステムズ株式会社 (監訳): “ドメイン特化言語 〜 パターンで学ぶ DSL のベストプラクティス 46 項目,” ピアソン桐原, 2012.
[6] Rüdiger Valk: “Object Petri nets? Using the nets-within-nets paradigm, ” LNCS 3098, pp.819–848, Spriner, 2004.
[7] 飯島 正: “アスペクト指向ワークフロー変換 〜 オブジェクト指向ペトリネットによるワークフロー表現への適用 〜,” 技術報告 (知能ソフトウェア工学研究会), Vol.112, Vol.165, pp.1-6, 電子情報通信学会, 2012.
[8] 飯島 正, 片山 輝彦, 金子 良太, 高橋 貴大: “オブジェクト指向論理ペトリネットを使った業務プロセス/業務ルール管理,” 第 8 回 全国大会・研究発表大会, 情報システム学会, 2012.
[9] 本田技研工業株式会社: “バーチャル工場見学 〜 車ができるまで,” http://www.honda.co.jp/kengaku/auto/press.html
[10] Mario Fusco: “hammurabi The Scala Rule Engine,” https://code.google.com/p/hammurabi/