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

Business Process and Business Rule Management by using a Logic Object-oriented Petri Net

飯島 正 †, 片山 輝彦 ‡, 金子良太 ‡, 高橋 貴大 †
Tadashi Iijima†, Teruhiko Katayama‡, Ryouta Kaneko‡, and Takahiro Takahashi†
† 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
最近，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．本報告は，従来より継続して開発してきたオブジェクト指向ペトリネット (Object-oriented Petri Net) によるワークフロー管理手法に対して，BRM を導入する試みに関して述べるものである．

1. はじめに

最近，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．本報告は，従来より継続してツール開発をしてきたオブジェクト指向ペトリネット (Object-oriented Petri Net) によるワークフロー管理システムを中心とした BPM に対し，BRM の考え方を導入する試みに関して述べるものである．

次の第 2 節では BPM と BRM の間の関係について述べる．そこで，BPM をベースに BRM を導入することにメリットのある具体的な用途として，(1-1) パラメータを外部化，(2-1) フローの構造変換，(2-2) リソースの再配分，の三つを取り上げる．次の第 3 節において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 (業務プロセス表現) について説明した後で，第 4 節，第 5 節，第 6 節において，上記の 3 つの具体的な用途に関して順に取り上げる．

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を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．

(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ことができる．
(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とができる．

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．

(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

これらについての具体的な議論は，第 4 節，第 5 節，および第 6 節で，より詳しく述べる．

3. オブジェクト指向ペトリネット

3.1. 基本概念

本研究では，エージェントのモデルの内部表現としてオブジェクト指向ペトリネットを用いる．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る．nets-in-nets 意味論に基づく参照ネット (Reference Net) [3] において，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)

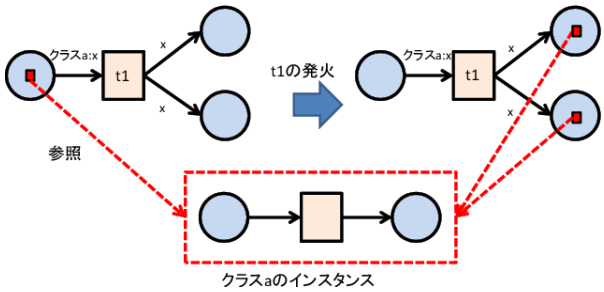

図 1: 参照トークンの例

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ッ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．

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，筆者らの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(the Object-oriented Petri Net family) の中で，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]．

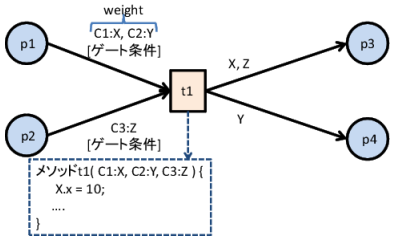

図 2: OPeN におけるアークとトランジッションのアノテーション

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，本校執筆時点では，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，トランジッションの発火条件であるゲート条件の記述や，トランジッ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ッションのアノテーションを示す．

トランジッション 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プとしての仕様では，プロパティの型やメソッドの定義のための言語仕様は，定義していない．基本

[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ッション 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．トランジッ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．この際，両方のペトリネットの間でデータをやり取りする表現？と！もあるが，ここでは省く．また，新しいオブジェクト・ネットの生成は，トランジッションの出力アーク上に，new キーワードを付けて記述されるが，その詳細も省く．

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．後述する論理ペトリネットの場合は論理型言語) のクラス定義に外部化しており，相互に対応付けることで代用している．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してみる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て，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ッ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する形で動作を確認している)．

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，[3] 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，一部のトランジッ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，双方のトランジッ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，あるトランジッション 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ッションが発火可能であること．

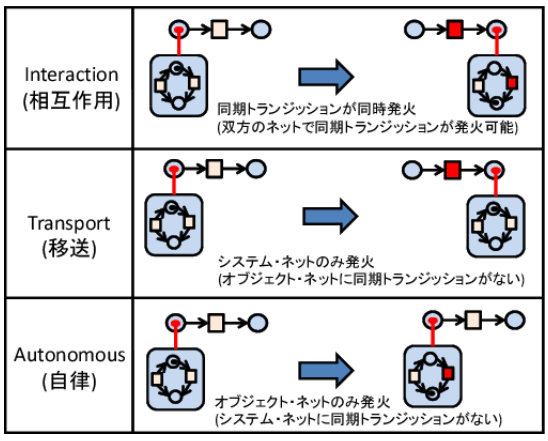

図 3: トランジッションの発火則

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ッ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．

3.2. 論理ペトリネット

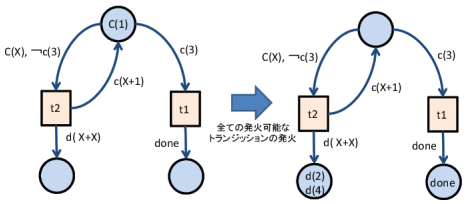

図 4: 論理ペトリネットの例

単純論理ペトリネット (SLPN; Simple Logic Petri Net) は，論理プログラミングと対応のよいペトリネットである．その例を図 [7] に示す [7]．トークンは論理項に対応し，トランジッションへの入力アーク上に書かれた発火条件のチェックの際に，発火条件の述語とトークンとの単一化によって変数に値が代入される．その変数を使って，トランジッションからの出力アークに計算を書くこともできる．こうした単純論理ペトリネットは，解集合を計算することで論理プログラムへ変換できる．これにより BDI アーキテクチャに基づくマルチエージェントシステム記述言語 AgentSpeak(L) [8] の有限状態版である AgentSpeak(F) [9] のモデル検査も試みられている [7]．

この単純論理ペトリネットを，オブジェクト指向拡張し，OPeN の枠組みに適合させたモデルを L-OPeN と呼ぶ．基本的に単純論理ペトリネットにおいて使われている項表現のトークンは，参照トークンによってオブジェクト・ネットで取り扱うことができるため，メソッドの表現力を限定 (論理変数の束縛に対応付けるため単一代入性を付与する) した OPeN に帰着させることができる．OPeN はメソッドの表現を外部化していることから，その表現言語の持つ Colored Petri Net を越える表現力と引き換えに，形式記述の持つモデルの分析能力を損なっている面があった．そこで，論理プログラミングとの対応を高める制限を付け加えることで，そのモデルの分析能力を高めることが，L-OPeN の目的である．

4. パラメータの動的な変更

一般に，ある業務プロセス中のゲート条件部分に含まれるパラメータの値は，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．その値の決定は，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．そこで，そうしたパラメータの値を決定する業務ルールを外部化することは，保守性を向上させる上で重要である[6]．

5. フローの動的な変更

5.1. 適応的例外処理

業務プロセス，特に，SOA のような外部サービス呼び出しで連携的にタスクを実行する部分と，人が関与してタスクを実行する部分とが混在している場合，その自動化 (automation) において，遅滞なくトランザクションを完了させるには，例外処理ないし補償処理が重要である．ただ，そうした例外処理は，業務プロセスの背景にある文脈に依存して，特定の文脈に適応した処理を選択することが望ましいことがある．そのとき，そうした選択をルールとして記述し外部化することは，保守性を向上させると期待できる[7]．

5.2. 横断的な関心事に対する処理

アスペクト指向プログラミング (AOP; Aspect-Oriented Programming) [5] ならびにアスペクト指向ソフトウェア開発 (AOSD; Aspect-Oriented Software Development) におけるアスペクト指向の概念は，ソフトウェアを構成する複数のモジュールに横断的に，共通の視点から見た関心事 (Crosscutting Concerns) が点在してしまう場合に，それらを統一的に取り扱うための考え方として，ソフトウェア開発のプログラミングやデバッグの工程から，より上流工程にわたるまで適用される考え方である．アスペクト指向の考え方は，プログラミング工程においても有用であるが，問題領域 (Problem Domain) に近い上流工程でその効力を発揮するものと期待している．これを，ペトリネットの変換に応用したものが，オブジェクト指向ペトリネット変換 [4] である．

[6] 具体的な業務ルール表現は，紙数の都合上，割愛する．
[7] 具体的な業務ルール表現は，紙数の都合上，割愛する．

ここでは，アスペクト，ジョインポイント，ポイントカット，アドバイス，ウィービングを一般的によくつかわれる用語をそのまま使用する [5]．アスペクト指向では，横断的関心事について扱うために，各モジュールに統一的に，何らかの記述（アスペクト指向プログラミングならプログラムコード）を埋め込む．その埋め込む作業のことをウィービングといい，埋め込まれる記述をアドバイスと呼ぶ．記述を埋め込むための候補の場所をジョインポイントと呼び，実際に埋め込む条件を満たしたジョインポイントの部分集合の地点に記述が埋め込まれる．その条件をポイントカットと呼ぶ．ポイントカットとアドバイスを定義するモジュールがアスペクトと呼ばれる．

アスペクト指向ワークフロー変換においても，これと同様に，プロジェクトファイルで規定されるオブジェクト指向ペトリネット群に対して，与えられたアスペクト群に基づいたウィービングを行い，変換されたワークフロー記述を得ることになる．但し，アスペクト指向プログラミングと違いペトリネットが対象であることから，あまり複雑なポイントカットやアドバイスを書くことは難しいという問題点がある．現在，記述例を増やしつつ，どのようなポイントカットやアドバイスが便利なのかを検討中である．現時点で，実現できているアスペクト指向ワークフロー変換は，単に，文字列で指定されたトランジッションの前，もしくは後に，指定されたトランジッションを挿入するといったものに限られている．トランジションの指定としては，トランジション名（トランジッション ID とは別で，ワークフロー記述では，トランジション発火とともに起動されるタスクを意味する）に関して，正規表現で検索を行ない，ヒットしたものを選択する．たとえば,「完了」で終わる文字列などを指定することが可能である[8]．

6. リソースの動的な再配分

リソースの動的な再配分とは，あるプロジェクト内のタスクにおいて，何らかの事情によってプロジェクト内のリソースであるタスクの実行者 (人的リソースや外部サービス) を変更しなければならない際に，他にタスクを実行可能なリソースを自動的に選出し，プロジェクトに割り当てることを指す．

リソースの状態や能力が不十分な際にも，プロジェクトを止めることなく進行することや，進捗が遅れているときに従来の配分よりもリソースを増強して遅れを取り戻すことがその目的である．具体的な状況としては，様々なケースが想定できある．例えば，担当者の急病によって，急遽タスクが実行できなくなってしまった場合や，担当者に対する事前の確認不足で，スキル (能力) 面でタスクが実行出来ないことが，判明した場合などもありうる．

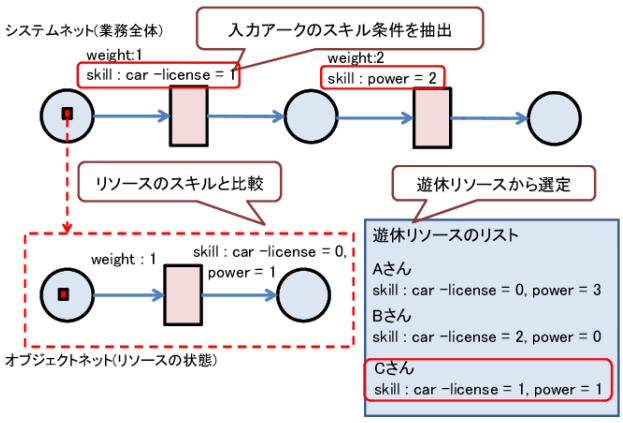

図 5: リソースの動的再配分

本報告では，後続するタスクの実行に必要なスキルが，担当者（リソース）には不足していることが，ワークフローの実行中に判明するというケースである．そこでポイントとなるのは，リソース不足をどう判定し，代替リソースとして，どのようなリソースを割り当てるかを決定するという 2 点である[9]．

オブジェクト指向ペトリネットでのワークフロー表現において，業務全体を表すシステムネットと各リソースの状態を表すオブジェクトネットが存在する．あるリソースがスキルを持っているかどうかは，そのリソースのオブジェクトネットを参照することで把握することができる．一方，ある業務に必要なスキルは業務全体を表すシステムネット上に記述されているので，タスク実行時にシステムネット上のタスク $task_i$ が必要とする (ゲート条件に記載される) スキルとリソース res のオブジェクトネットのスキルを比較すること

[8] 具体的な業務ルール表現は，紙数の都合上，割愛する．

[9] 実際には，あるタスクを担当するリソースを交代した場合，リソース間での引継ぎに要するコストを評価する必要もあるが，今回はその点に関しては省略する

で，タスクが実行可能かどうかを判断することができる (式 (1))．ここで，スキルの有無とレベル (熟練度) は 6 段階 (0〜5) で表現することで同時に表現する (レベルの値が大きいほど能力が高く，0 はスキルを持たないことを意味する)．

$$capable(res, task_i) = \begin{cases} 1 & \cdots \quad if\ task\_capability(res, task_i) > 0 \\ 0 & \cdots \quad otherwise \end{cases} \tag{1}$$

$$task\_capability(res, task_i) = \prod_{x \in needed\_skill\_labels(task_i)} skill\_capability(has\_skill\_level(res, x), needed\_skill\_level(task_i, x))$$

$$skill\_capability(skill\_level, needed\_skill\_level) = possible(skill\_level, needed\_skill\_level) \times (skill\_level / needed\_skill\_level)$$

$$possible(skill\_level, needed\_skill\_level) = \begin{cases} 1 & \cdots \quad if\ skill\_level \geq needed\_skill\_level \\ 0 & \cdots \quad otherwise \end{cases}$$

さらに，代替可能リソースの候補集合 $R_{candidate}$ は，決して最適解とはいえないが，より適切なものを選び出すために後続するタスク (長さ n) を一部考慮に入れ[10]，式 (2) の $R_{candidate}$(遊休リソース集合 R_{wait} の部分集合) で与える[11]．ここで，$\arg\max_x f(x)$ は，$f(x)$ を最大にする x の集合を意味する．

$$R_{candidate} = \arg\max_{res \in R_{wait}} n\_capability(res, task_i, n) \tag{2}$$

$$n\_capability(res, task_i, n) = capable(res, task_i) \times \sum_{j=0}^{n} task\_capability(res, task_{i+j})$$

7. まとめ

本報告では，業務プロセス管理に業務ルール管理を部分的に導入して使い分けることで，主に，プロセスの動的変更 (プロセス中のパラメータの変更と，フロー自体の変更) に対応し，保守性を高める手法を簡単に示した．紙数の都合上，具体的な業務ルール表現を割愛している部分も多いため，概念的な説明に留まっているが，現在，こうしたプロセスの動的変更を導入するための業務ルール表現と変換系を順次構築中である．

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] Rüdiger Valk: “Object Petri nets? Using the nets-within-nets paradigm, ” LNCS 3098, pp.819–848, Spriner, 2004.

[4] 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.

[5] Gregor Kiczales, J. Hugunin, E. Hilsdale, M. Kersten, J. Palm, C. Lopes, B. Griswold, and W. Isberg, “Aspect-Oriented Programming,” Palo Alto Research Center, Technical Report, AFRL-IF-RS-TR-2003-173, July 2003.

[6] Sean Luke: “Multiagent Simulation And the MASON Library, First Edition,” Online Version 1.0, August, 2011, http://cs.gmu.edu/ eclab/projects/mason/manual.pdf

[7] Behrens, Tristan M. and Dix, Jürgen: “Model Checking with Logic Based Petri Nets,” IfI Technical Report, IfI-07-02, Clausthal University of Technology, 2007.

[8] A. S. Rao. : “AgentSpeak(L): BDI agents speak out in a logical computable language,” In W. Van de Velde and J. Perram (eds), Proc. Seventh Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW ’ 96), No. 1038 in LNAI, pp.42-?55. Springer-Verlag, 1996.

[9] Rafael H. Bordini, Michael Fisher, Carmen Pardavila, and Michael Wooldridge: “Model Checking AgentSpeak, ” Proc. of AAMAS ’03, pp.409-416, 2003.

[10]但し，紙数の都合上，ここでは簡単のために分岐やループのない直線的なタスク系列に限定する

[11]実際には，タスクを実行するのに必要十分なレベルのスキルを持っているリソースが存在しない場合は，そのスキルを持っているリソースの中で求められるレベルに最も近いリソースを抽出したい．また，より複雑となるが，遊休リソースだけでなく全リソース集合を対象とすることでリソースの交代を考えたい