社団法人 情報処理学会 研究報告
IPSJ SIG Technical Reports

2007－ＤＳＭ－45（21）
2007／5／11

柔軟な業務プロセス実行方式におけるモデリング手法

増田 健† 中島 一† 野末 晴久† 大石 晴夫† 山村 哲哉†

†日本電信電話株式会社 NTTアクセスサービスシステム研究所 〒261-0023 千葉市美浜区中瀬１－６
E-mail: †{ tmasuda, nakajima.hajime, nozue, oishi, yamamura }@ansl.ntt.co.jp

あらまし 柔軟な業務実行を可能とするプロセス制御技術としてフレキシブルプロセス制御技術（ＦＰＣ）が提案されている．ＦＰＣの能力を活かしながら，実際の業務における柔軟性をひきだす新しいモデリング手法を提案する．プロセスの多面性を開発環境上で陽に扱うことにより，これまでのワークフローエンジンやルールベースエンジンでは記述や実行が困難な処理を実現できることを示す．

キーワード ワークフロー，ビジネスプロセス，プロセスモデリング，制約充足，プランニング

A Modeling Method of Business Process with the Flexible Process Control Engine

Takeshi MASUDA†, Hajime NAKAJIMA†, Haruhisa NOZUE†, Haruo OISHI†, and Tetsuya YAMAMURA†

† NTT Access Network Service Systems Laboratories, NTT Corporation 1-6 Nakase, Mihama-ku, Chiba, 261-0023 Japan
E-mail: †{ tmasuda, nakajima.hajime, nozue, oishi, yamamura }@ansl.ntt.co.jp

Abstract The Flexible Process Control (FPC) technique was proposed to enable enterprise systems to perform the business task sequences as flexibly as possible. In this paper, we will present a new method of business process modeling, which is implemented in the FPC, to elicit inherent flexibility from on-the-scene workflow. The process modeling method enables one to treat many aspects of the business process in a positive way on our development environment. We will show how this development environment successfully fulfills a process definition and a process execution that have difficulty with most conventional process design tools for the workflow engine.

Keyword Workflow, Business Process, Process Modeling, Constraint Satisfaction, Planning

1. はじめに

SOA や NGOSS におけるシステム連携では，協調プロセス制御方式として BPEL(Business Process Execution Language)を用いたワークフロー記述とその実行処理系が適用されている．また，より業務知識に近いビジネスプロセス記述法として，BPMN(Business Process Modeling Notation)が規定され，それに準拠した多数のビジネスプロセス定義ツール（開発環境）が存在する．実際のシステム開発では，複雑な処理の流れを記述する場合限界があるため，組み合わせ的に考えられる膨大な数のフローから多くの選択肢を削って，典型的かつ定型的なフローに絞った設計・実装を行うのが一般的である．しかし，多様な業務ドメインの中には，必ずしもこの方針が理想でないものも存在する．また，近年注目されている内部統制や業務改善の観点を考慮すると，業務要件の変化へのシステムの対応能力は，可能な限り高いことが望ましい．

筆者らは，このようなドメインに対し，ワークフロー記述に比して抜本的な柔軟性を有するプロセス制御（Flexible Process Control: FPC）技術[1]を提案し，その実行エンジンならびにフレームワークを実現してきた[2][3]．FPC 技術による実行時の柔軟性を十分に発揮しつつ，実際の開発場面での実用性を高めるには，業務要件の変更に対して影響範囲や副作用を的確に把握でき，変更内容を設計意図どおりにコントロール可能とするプロセスモデリング手法が重要である．

本稿では，まず２章で FPC におけるプランニング処理系の位置づけを説明し，それに由来する FPC の課題を，モデリング手法の観点から明確にする．３章では開通サービスオーダー業務を例に，ワークフロー形式上でのプロセスモデリングの課題を説明する．４章において，FPC フレームワークを前提としたプロセス表現とモデリング環境を提案し，前記課題に有効に対処できることを示す．５章では，モデル操作の内部計算シミュレーションをもとにその実現性を示す．最後に６章で本モデリング手法の今後の方向性を述べる．

2. フレキシブルプロセス制御（FPC）の概要

2.1. 基本構成と適用効果

リソース割り付け，故障診断，コンフィグレーションなどの非定型処理は，多数のデータ項目の値を決定していく性質のプロセスであると捉えることができる．

これに対しては，制約充足処理系（制約ソルバー）を応用することで，高い柔軟性が達成できる[1]．一方，

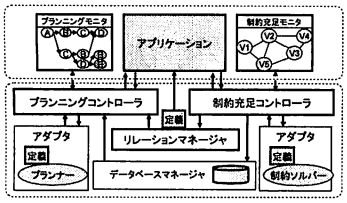

図１．FPC フレームワーク

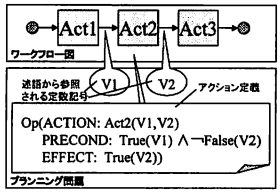

図２．ワークフロー図からのプランニング問題生成

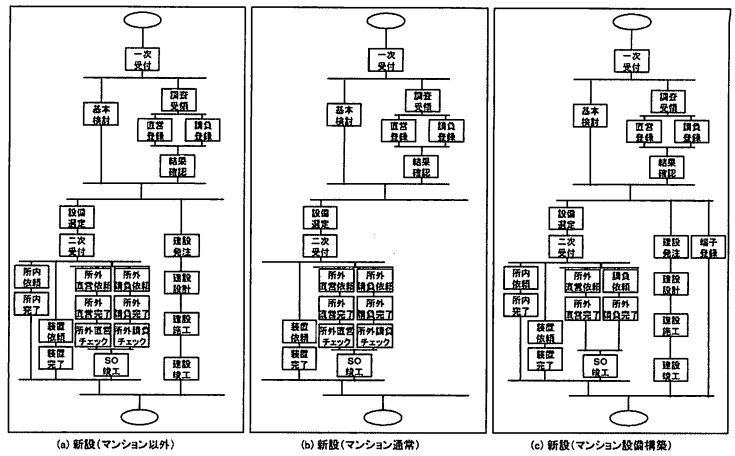

図３．光ブロードバンド開通ＳＯ業務（サンプル）

非定型処理の前後に接続する関連業務や，パラレルに進行しながら一部相互作用をもつ周辺業務との間には，全順序ではなく，半順序性が存在する場合が多い．リソース割り当てなどの非定型業務を含むさらに上位の業務（例えば，開通サービスオーダー業務）に広く FPC を適用する場合，次の課題があった．

- 非定型側面と定型側面が混在するプロセスに対して，両面を同時に満足する処理方式がない．非定型部の自由度の高さを損なわずに，境界部分の半順序性を順守する方法により，業務全体に対する効果を引き出すことが望ましい．
- 操作対象間の依存関係や操作種類間の前後関係によって，次に何を実行すべきかが場面ごとに複雑に変わる．ユーザに高いスキルが要求され，それを補うナビゲーション機能の実装も通常の方法では困難となる．

これらに対しては，制約充足処理系にプランニング処理系（半順序プランナー）を組み合わせてハイブリッド化することで，課題が解決できる[2][3]．図１に FPC フレームワークを示す．２つの問題解決器（制約ソルバーと半順序プランナー）を連携させることにより，結果として得られるデータ値が所定の制約を満足すること（整合性）ならびに，そのデータ値の操作過程が所定の条件を満足すること（順序性）が保障される．また，上記保障を得るために必然的に満たすべき状態は自動的に計算され（制約伝播，正例生成），データ値候補や推奨操作の形で動的に導出・提示される．つまり，ワークフローやルールベースのように，データ値や操作過程を開発時点で直接的・限定的に指定する必要がない．このように FPC は，一般的な処理記述方法に比べ，実行時の状況に応じた柔軟性・厳密性を提供できるという利点がある．

2.2. 処理記述における FPC の利点と課題

業務要件から FPC のプログラム仕様を検討する工程は，制約充足問題のモデリングならびにプランニング問題のモデリングに相当する．制約充足問題のモデリングには，通常の手続き型プログラムとは異なる考え方が必要であり，その作業を支援する方法（デバッグ方法）もいくつか提案されている[4][5]．一方，プランニング問題については，明確な課題の提示や整理は行われてこなかった．次にプランニング問題の利点（一部）とモデリングにおける課題を挙げる．

プランニング問題においては，個々のアクションの前置条件・後置条件を個別に定義するため，複数のフローを同期実行させる場合，そのためのフロー記述を必要としない．単にそれらを構成するアクションの和集合を，プランニング問題の候補アクション集合とすればよい．これは，前置・後置条件が，順序的に関連のある他アクションに対してではなく，状況（状態の集合）に対して記述されるためである．このため，相互に影響を及ぼしあうフロー間であっても，状況を介した作用（同期もこの中に含まれる）であれば，特別な配慮を必要としない．

また，既にワークフローとして定義されたプロセスを，等価なプランニング問題に変換することも容易である．図２に示すように，ワークフロー定義上の個々のフローリンクについて，プロセストークンが通過したか否かを，プランニング問題における状況を表す述語 [6]として内部表現すれば良い．すなわち，フローアクティビティをプランニング問題におけるアクションとして表し，前置条件にトークンの通過状況を記述する．ただし，ここでは議論を簡潔にするため，ループがない場合を仮定する．

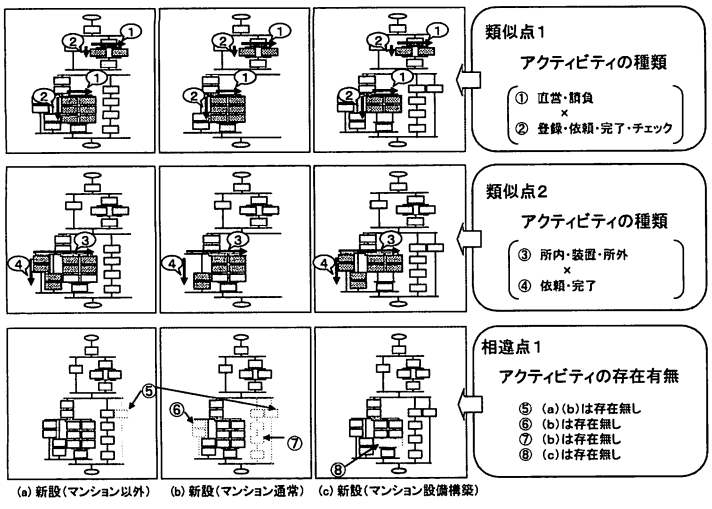

図４．分岐や実行有無の観点の類似・相違箇所

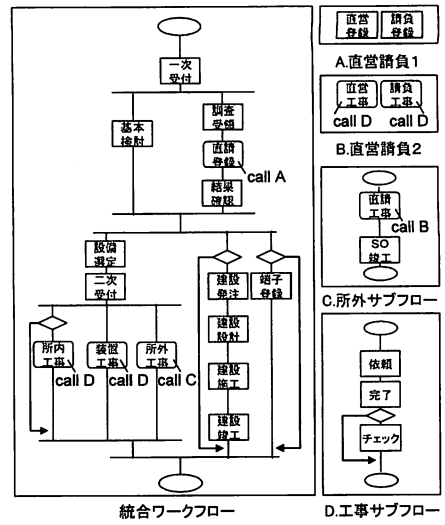

図５．ワークフロー記述の複雑化

一方，課題の側面は，上記利点と表裏一体の関係にある．FPC への適用を前提とした場合の課題認識を次に示す．

＜FPC のモデリング（順序性の側面）の課題＞
- プランニング問題では，個々のアクションの前置条件・後置条件を個別に定義する必要があるが，この結果として得られる全体のプロセスが設計意図どおりになっているか否かがわかりにくい．

例えば，前節の方法でワークフローから変換して得られたプランニング問題を，より柔軟なプロセスに改編しようとした場合，前置・後置条件の局所的変更がもとのワークフローにどのように影響するかを把握するのは，問題の規模が大きくなるに従い困難になる．

3. 業務プロセスモデリングの課題

3.1. 開通サービスオーダー業務プロセスの特徴

開通サービスオーダー業務（開通ＳＯ業務）の事例を用いて，業務プロセスモデリングの課題を説明する．図３は光ブロードバンドサービスを仮想した開通ＳＯ業務プロセスのサンプル例である．各ワークフロー図は，サービスの提供形態（業務区分）によって別々に定義されている．各図の類似・相違箇所を図４に示す．

1. 所外，所内，装置の３ドメインから構成される
2. 直営，請負の２種類の施工形態が存在する
3. 場合により実行不要なアクティビティがある

上記以外のサービスや提供形態，さらには廃止ＳＯや切り替えＳＯなど他の種類のＳＯを考慮すると，業務のバリエーションが多数存在することがわかる．さらに，オペレーションシステム開発にあたっての要件定義・仕様検討では，より詳細なレベルのプロセスを記述する必要がある．業務プロセスモデリングに際しては，これらを全てワークフロー図として明確に定義しなければならない．業務要件変更への対応を考えると，次が問題になると考えられる．

＜ワークフロー形式上のモデリングの課題＞
- 記述が冗長化してしまう
  （記述要素間の関連情報が散逸）
- 処理の抽象化ができない
  （処理内容を体系的に記述する機構が不十分）

3.2. ワークフロー形式上での記述例と課題

業務プロセス定義上で，サブフロー呼び出しを用いて，冗長な記述をできるだけ少なくすることを考える．サブフローはワークフロー記述におけるモジュール化機構に相当する．業務区分情報（図３のサービスの違い）をプロセス変数として内部表現し，条件分岐やサブフローで整理したものが図５である．業務上の場面ごとの動きがプロセス変数の値やフロー間の呼び出し関係に置き換えられるため，一見しただけでは具体的な流れが理解しにくくなる．これとは逆に，条件分岐やサブフローを具体的な動きに展開すると，場面を特定した場合は理解しやすいが，全体としての流れの把握がしにくくなる．また，いずれの場合も業務要件変更に対して，影響範囲や副作用を知ることは煩雑で時間のかかる作業となる．さらに，このようなフロー整理はやり方によって何通りも考えられるため，モデル検討を難しくしている．特に似た条件分岐が多数発生し，記述が冗長化する場合は顕著になると考えられる．

これらはプロセス定義上に見出されるパターンの重ね合わせや局所的な例外箇所を，柔軟かつ一意に表現できないことに起因すると考えられる．プログラム

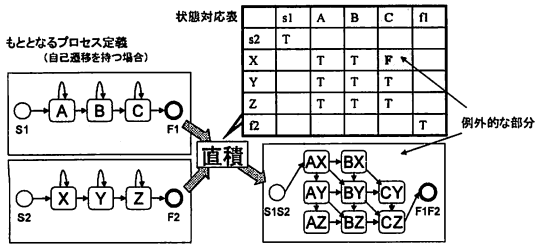

図６．一部重ねあわせ状態が存在しない場合

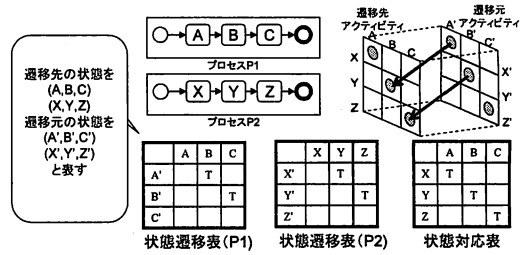

図７．遷移行列どうしの直積

言語のクラス継承やプロトタイピングと比較すると，ワークフロー記述のモデル化能力が低いことが問題である．

一方，より詳細なレベルのプロセスを記述する際，受付業務や選定業務など非定型業務の順序任意性を，最大限許容することを考える．非定型処理の前後に接続する関連業務やパラレルに進行しながら一部相互作用をもつ周辺業務との間に半順序性が存在するが，これに対しては，FPC のもともとのねらいどおり，プランニング問題として表現するのが適している．例えば 2.2.項の「利点」のように，個々のアクションの前置条件・後置条件を状況に対して定義するため，ワークフローにみられた記述の冗長化が発生しない．しかし，システム上の業務ロジックを直接，プランニング問題の前置・後置条件として記述することは，「課題」として示したように一般的に困難である．

4. 提案するプロセスモデル

4.1. 基本的な考え方

業務フローに関する要件の変更に柔軟に対応しつつ，同時に影響範囲や副作用を的確に把握できるコントロール性を実現するために，業務上の観点（アスペクト）ごとにプロセスを定義し，これらの直積として，所望のプロセス定義（業務全体の流れをあらわす）を作成する方法を提案する．2.2.項で説明したように，FPC には複数プロセスの合成が容易であるという特徴があり，ワークフロー形式上でのモデリングは業務の場面を限定した場合（アスペクトを絞った場合），比較的やりやすいという特徴がある．提案手法は両者をマッチングするアプローチといえる．

ここで注意すべきなのは，実際の業務プロセスは，アスペクトごとのプロセス定義の直積に厳密に一致するわけではない点である（複数の独立したプロセスの単純な重ね合わせではない）．特定のアクティビティどうしの一部重ねあわせ状態が，実際には存在しない場合がある（あるいはプロセスとして一部重ねあわせ遷移を許さない場合がある）．そこで，このような例外的な部分を任意に指定できるように，多次元配列を用意し，その要素のブール論理値によって，状態が定義されるか否か，遷移が許されるか否か，を表現することが考えられる．具体例を図６に示す．ここでは，「C」と「X」の重ね合わせの状態だけを除去している．

4.2. プロセスの内部表現方法

前項の考え方にもとづき，多次元配列をモデリング環境の内部表現として導入する．ここでの多次元配列は次元数が定義内容に対して非常に大きくなるが，多くの場合疎行列となるため，実装上の問題は回避可能であると考える．これら次元軸は業務を捉える場合のアスペクトに対応し，各次元軸の添え字のインデックス値は，当該アスペクトで業務を見た場合の所定のアクティビティに対応する．

多次元配列上の値には２つの意味がある．複数の観点によるプロセスの直積において，特定の部分だけ０（False）が設定されていれば，その部分は「①『状態』が存在しない」もしくは「②『遷移』が存在しない」ことを意味する．

なおここで，ワークフローにおける「遷移」は，図７に示す形の状態遷移表として表現できることを補足しておく．各アスペクトについて遷移元と遷移先に対応する次元軸をそれぞれ定義すれば，「状態」の存在有無を表すのと同様に，「遷移」の存在有無も多次元配列として表現できる．

4.3. モデリング手法と開発環境アーキテクチャ

上記で説明した内部表現の多次元配列を，開発環境上において，どのように操作するかを説明する．

１）プロセスのブラウジング

多次元配列の内容をオペレータが直接見て，対応するプロセスモデルをイメージすることは通常困難である．そこで多次元配列のブラウジング機能を提供する．一般の OLAP 集計（Online Analytical Processing）と同様に，多次元配列のドリリング，スライシング，ダイシングの操作を行って２次元配列化し，これを遷移行列と見なして対応するワークフローを表示する．例を

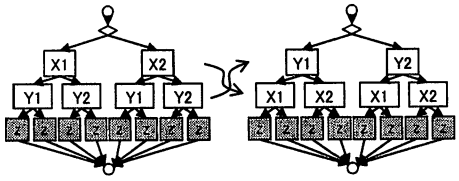

図８．ダイシング操作によるフロー表示の変化

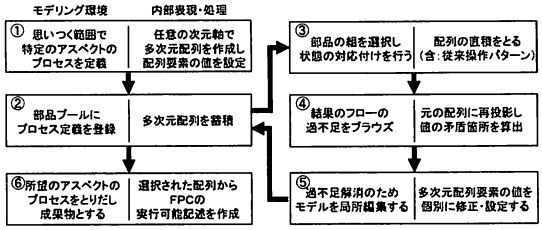

図９．編集作業の流れと内部処理

挙げて説明する．

ドリリングは，複数のアクティビティを縮退させ，抽象化されたアクティビティとする見方である．あるアスペクトにおける状態や遷移を同一視する，すなわち当該次元軸方向に多次元配列値を集計する（AND もしくは OR をとる）ことで実現できる．

スライシングは，指定のアスペクト上の特定の状態に固定した場合のシステム全体の振る舞いを抽出する見方である．すなわち指定次元軸の特定インデックスに相当する断面で多次元配列を切り出すことで実現できる．

ダイシングは，条件分岐やサブフロー呼び出しの入れ子の順序を変える見方である．図８に具体例を示す．ここで，X1,X2 と Y1,Y2 の実行履歴の組み合わせによって，Z の種類を切り替えるのが重要な要件と仮定し，Xn と Yn との間には順序的要請がないものとする．

以上示したとおり，多次元配列上にプロセスを表現しそれをブラウズする機能を提供することで，業務の流れの見方の多様性に応えつつ，記述の一意性を実現できることがわかる．

２）プロセスの作成・編集

また，ブラウジング機能のもとであっても，次元数が大きい配列の内容を最初から直接編集するのは，事実上不可能である．そこでオペレータに対し，グラフィカルな開発環境上での編集手段を提供する．図９に編集作業の流れ（矩形左側）と内部での多次元配列処理（矩形右側）を示す．まず業務上の場面や観点を限定して，単純なプロセスを定義し（①），部品プールに登録する（②）．このときプロセス定義ごとに小規模な多次元配列が作成される．次に任意の部品（プロセス）の組を選択し，２つの観点を同時に考慮した場合のプロセスを定義する．このとき，多次元配列上で２つの配列の直積をとることによって，新しいプロセスに対応する配列を生成する．直積をとる際は，単純な重ね合わせだけではなく，サブフロー呼び出しに相当する対応づけを行うなど，従来のメタファに相当する操作を利用できる．これは配列要素への値設定を，一定のパターンで行うことで実現する（③）．得られたプロセスをワークフロー形式で視覚的に確認し（④），上記操作では細かくコントロールできない例外箇所を，グラフィカルな操作により個別に指定する．作成・編集された新たなプロセスは，部品プールに追加され（⑤），他のプロセスを作成する場合に利用される．この操作を繰り返して業務の全体像を表すフローが得られたら，それをもとに FPC の実行可能記述を生成する（⑥）．

以上示したとおり，考えやすいプロセス定義から着手し，インクリメンタルにプロセスの重ね合わせを行っていくことで，冗長な記述を廃した体系的なモデルを作成できることがわかる．

３）プロセスの整合性チェック

プロセスのブラウジングや作成・編集は，オペレータが考えやすい範囲のアスペクトを指定して行う．常に全てのアスペクトを考慮することは困難なので，全体としての整合性がとれているかどうかを自動的にチェックする機能が有用と考えられる．

上記プロセス定義の作成過程における依存関係を，上半束構造（特殊なツリー構造）として記憶することで，モデル上の記述要素どうしの相関関係を扱うことが可能と考えられる．例えば，あるプロセス定義上のアクティビティに対応する状態が全体のプロセス上に存在しない（生成されない）場合は，オペレータの設計意図に反する状況（あるいは設計意図そのものが矛盾している状況）が考えられる．このような状況を，多次元配列要素間の論理演算により検出する方法が期待される．

以上示したとおり，業務要件の変更に対して影響範囲や副作用を的確に把握でき，変更内容を設計意図どおりにコントロール可能なモデリング環境が実現可能と考えられる．

5. モデル操作コア処理部の評価

5.1. 実装概要

本稿で提案する開発環境を実現するには，実際のプロセス定義を多次元配列で表現した場合の計算コスト（必要な記憶量，実際の計算量）が重要となる．そこで，モデル操作と同等の計算を行う簡易な処理系を作成し，実測を行った．使用言語は Java(J2SE5.0)，計算機環境は，CPU: Xenon 1.7GHz×2, Memory: 1GB, OS:

WindowsXP, JVM: build 1.5.0 である．多次元配列の内部表現や直積計算は自製のビット計算ライブラリにより実装した．原理確認用という位置づけから，シンプルな処理系にとどめ，記憶領域が指数的に増大する傾向に対し，これを回避する工夫は特に行っていない．

5.2. 評価結果

図４．（ａ）で示した例は，20 種類のアクティビティ要素から構成されており，最終的なプロセスはこれら要素の重ねあわせによって構成されている．図４．（ｂ），（ｃ）のサービス追加により，プロセス定義のバリエーションを作成した場合も，新たに付け加えるべきアクティビティは１種類で，他は既存のアクティビティの存在有無や業務区分に応じた条件分岐による違いである．対象とする業務分野に依存するが，一般的に業務バリエーションの追加として扱われる変更は，論理変数の数にして 20~80%の増大と予想される．また，「建設発注」から「建設竣工」に至る部分フローを１つのアクティビティに抽象化すれば，フロー上のまとまりをグループとして扱うことができ，モデリングの際に同時に考慮すべきアクティビティ要素の個数を減らすことができる．今後，さまざまな事例について上記観点で実質的数量の検証が必要である．

本例で用いた多次元配列のメモリ消費量と記述不整合チェック処理の計算時間を，論理変数の個数との対比で示したのが図１０である．上流設計レベルのアウトライン検討であれば，本環境上で十分作業可能である．今後はモデリング手法の基本評価を行う際に利用していく予定である．

一方，実際の要件定義や仕様検討では，本例におけるアクティビティをさらに詳細なプロセスに展開する必要がある．例えば，本例の倍程度の深さのモデル階層を扱う能力が最低限必要と想定される．実装上の工夫として，多次元配列に商用あるいは公開の OLAP ライブラリを利用し，整合性チェックアルゴリズムに分割統治法を取り入れるなどの対応が考えられる．

6. おわりに（今後の方向性）

本稿では，開通サービスオーダーに代表される，非定型と定型の両方が混在する業務プロセスについて，FPC 実行系を前提とした独自のモデリング手法を提案した．多次元配列による内部表現を基本とすることで，モデリング環境に要求される多様なモデル情報やモデル操作を統一的に扱えるという特徴がある．

一般に SOA をはじめとするシステム連携でのプロセス制御に関して，BPEL や BPMN が有効とされているが，３章で示した課題は共通していると考えられる．本稿で提案したプロセスモデリング手法は，特定の業務ドメインに限らず，広く適用が可能である．今後，スケーラビリティの問題を解決し，ワークフローエンジン並みに高速・大量の処理が可能な FPC 実行系が実現できれば，従来手法に代わる生産性の高い（変更に対するコントロール性の高い）プロセスモデル開発環境としての応用が期待できる．

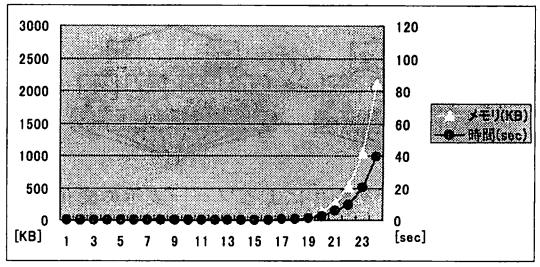

図１０．多次元配列処理の記憶量・計算量

文　献

[1] 田山健一，山村哲哉，"制約充足アプローチによる運用多様化に対応した設備選定アルゴリズムの提案，"電子情報通信学会技術研究報告，Vol.103, No.575, pp.85-90, Jan.2004.

[2] 野末晴久，中島一，大石晴夫，増田健，山村哲哉，"プランニングと制約充足を利用した業務の多様性に対応するプロセス制御技術，"電子情報通信学会技術研究報告，Vol.106, No.600, pp.71-76, Mar.2007

[3] 増田健，大石晴夫，野末晴久，中島一，山村哲哉，"制約充足とプランニングを併用した柔軟なプロセス制御フレームワーク，"2007 信学全大 B-14-5

[4] N. Jussien and V. Barichard: The PaLM system: explanation-based constraint programming, In Proc. of Techniques for implementing Constraint programming System, a post-conference workshop of CP2000, pp. 118-133 (2000).

[5] 中島一，大石晴夫，山村哲哉，"モデル導出にもとづく制約充足の妥当性の検査手法，"人工知能学会研究会資料，SIG-KBS-A504, pp.135-140, Mar.2006

[6] S. Russell, P. Norvig, “エージェント　アプローチ人工知能，” 共立出版，古川康一（監訳），pp.339-367, 1997.