情報処理学会論文誌　Vol. 51　No. 9　1751–1764 (Sep. 2010)

プロセス間競合を考慮した自己適応システムの形式仕様構築

中川博之[†1]　大須賀昭彦[†1]　本位田真一[†2,†3]

近年，ソフトウェアの利用環境が広がり，状況の変化に応じて自発的に振舞いや構成を変化させる自己適応システムの実現に対する期待が高まっている．自己適応システムは様々な関心事に対応するための並行プロセスにより構成されるが，状況によって振舞いやそれを構成するプロセス群が変わる可能性があることから，プロセス間で発生する競合に対しては，対処法の検討だけでなくその検出さえも困難である．そこで本研究では，システム開発の早期段階における分析・検証技術に着目し，自己適応システムに対する要求モデルと形式仕様を用いた競合検出法と，競合を検出するための形式仕様構築法を提案する．また，提案手法の有効性を仕様構築実験の実験結果から評価する．本提案手法により，自己適応システム構築に対する早期段階での競合への対応が期待できる．

Constructing Formal Specifications for Self-adaptive Systems with Handling Conflicts among Processes

Hiroyuki Nakagawa,[†1] Akihiko Ohsuga[†1]
and Shinichi Honiden[†2,†3]

Self-adaptive systems have recently attracted attention as flexible software because they can change their own behaviors to react to changes in their environments. However, these systems usually have multi-processes within them and developers are forced to design and construct these processes deliberately not to induce conflicts. This paper describes our approach to developing self-adaptive systems utilizing a requirements model and formal specification to detect conflicts and design the behavior keeping away the conflicts. The paper also discusses our evaluation of the effectiveness of our development process through a case study.

1. はじめに

近年，ソフトウェアの活躍する場面が広がり，環境の変化に応じて，みずからの構成や振舞いを変化させることのできる**自己適応システム**[1),2)] の実現に対する期待が高まっている．自己適応システムとは，みずからの目的と与えられた制約を管理し，環境や内部状態の変化に応じて振舞いや構成を切り替えることのできるシステムであり，QoS が要求されるアプリケーションサーバやネットワークシステムに対してだけでなく，組み込みシステムやロボットに代表されるユビキタス環境下のシステム，さらには日々新しい攻撃方法が出現しているセキュリティ分野においても，その適用が期待されている[3)]．自己適応システムにおいては通常，与えられた要求を満足するために，様々な関心事に基づいた機能を実現するプロセス群と制約を遵守するプロセス群がシステム内で並行に動作することとなる．このような並行プロセス群を内部に持つシステムでは，プロセス間で**競合**が引き起こされる可能性があり，開発段階で競合が発生しうる箇所を特定し，競合への対処法を検討しておかなければならない．しかしながら，特に自己適応システムにおいては，適応により生じるプロセス間競合を回避するためのメカニズムをあらかじめ組み込んでおくことが必要であるうえに，適応の状況やタイミングによって起動プロセス群や各プロセスの実行状態が異なるため，競合を発見することさえも困難であることが多い．また，自己適応システムは組み込みシステムや移動型ロボットなど実世界環境におけるハードウェアと相互作用するソフトウェアとしての実装も想定されるため，従来のソフトウェアと比較して実装に要するコストは高く，実装されたシステムに対するテストからの競合発見は現実的ではない．したがって，自己適応システムにおいては，プロセス間で生じる競合をシステム開発の早期段階で発見し，その対処法をシステム実装の前段階で決定しておく必要がある．

そこで本研究では，自己適応システムにおけるプロセス間競合の効果的な検出と競合対処法の確立を目的として，ゴール指向要求モデルを用いたプロセス間競合の検出法と，競合の具体的な検出と対処手段検証のための形式仕様構築法を提案する．本論文では特に，ゴール指向要求モデルからの競合情報の抽出法と形式仕様の構築を支援する仕様テンプレート生

†1 電気通信大学
The University of Electro-Communications
†2 国立情報学研究所
National Institute of Infomatics
†3 東京大学
The University of Tokyo

成法を提案する．これらは本研究において実装したツールにより，自動化が可能である．また，仕様のテストが可能である形式仕様記述言語 VDM++に着目し，プロセス間の競合や競合への対処法を検証するための形式仕様構築法も提案する．これらの手法により，開発の早期段階でシステム内に潜在する競合を検出してその対処法を検討することが可能となり，プロセス間競合を回避した頑強な自己適応システムの形式仕様が構築可能となる．

本論文の構成は以下のとおりである．まず 2 章で自己適応システム構築におけるプロセス間競合への対応の難しさについて議論し，競合対応のために開発プロセスに求められる要件を抽出する．続く 3 章では，自己適応システムにおける競合解決のための要求記述法と形式仕様構築法について述べる．4 章では，提案手法を利用した自己適応システムの形式仕様構築実験とその結果から，本手法の実現可能性と有効性を評価する．最後に，5 章で適用範囲と関連研究について言及し，6 章でまとめと今後の展望について述べる．

2. 自己適応システムにおけるプロセス間競合

2.1 競合解決の難しさと要件

自己適応システムは通常，提供すべき機能を実現するためのプロセス群と制約を遵守するためのプロセス群がシステム内で並行動作し，相互に連携することで環境変化への適応を実現している．このような複数プロセスが存在するシステムの構築においては，プロセス間で発生する競合に特に考慮する必要がある．しかしながら，自己適応システムは適応の状況やタイミングによって起動プロセス群や各プロセスの実行状態が異なる高度に状況依存なシステムであり[4]，競合の対処法を検討するどころか，どのような状況でどのプロセスが起動し，どのリソースに対して競合を引き起こすかといった競合の検出さえも容易ではない．つまり，自己適応を実現するためには，プロセス間で生じうるすべての競合の可能性のある状況を開発時に網羅的に把握し，それらに対応するためのメカニズムをあらかじめシステムに組み込んでおかなければならない．

また，自己適応システムの構築においては，組み込みシステムやロボットなど実装プラットフォームに大きく依存したシステムも対象となるため，従来のシステムよりも実装コストが高い場合が多い．その結果，実装コードに対するテスティングはコストの面からも，また，状況依存である競合をうまく再現させてエラーを検知しなければならないという検出の難しさの面からも現実的ではない．したがって自己適応システムにおいては，競合の検出や対処法の検討は実装フェーズよりも前の段階で実施されていることが求められる．

このような背景から，本研究では，自己適応システムのプロセス間競合に対処するために開発プロセスに求められる要件として以下の 2 つを定義する．

［定義 1］プロセス間競合対処のために自己適応システムの開発プロセスに求められる要件

- **要件 1：** 適応状況により発生状態が異なるプロセス間競合に対して，網羅的な検出が可能である．
- **要件 2：** システムを実装することなく，競合に対する対処法を検討することができる．

本研究では以降，定義 1 の要件を満たす自己適応システム開発プロセスについて議論する．

2.2 本研究のアプローチ

まず要件 1 に対しては，状況により実行プロセス群の異なる自己適応システムにおいて，各局面における起動プロセス群を同定し，他プロセスとの競合の可能性を網羅的に検査することは容易ではない．また，自己適応システムが直面するすべての状況を考慮し，システムがとりうる状態を網羅的に列挙することも現実的ではない．したがって，要件 1 を実現するためには，競合状態を網羅的に探索するのではなく，各プロセスがとりうる状態をもとに，競合の可能性がある組合せを抽出する手段が有効と考えられる．一方で要件 2 を満たすためには，実装フェーズの前段階，つまり設計フェーズ終了時までに競合発生個所を特定し，これに対処するための手段を決定する必要がある．しかしながらその一方で，競合に対処できることを確認するためには，対処手段に対する何らかの検証が必要である．

そこで本研究では，まず要件 1 を満たすために，プロセス単位での競合発生状況の抽出を目的として，要求モデルを利用した競合の検出法を提案する．特に本研究では，要求間の関係を階層構造で表現した**ゴール指向要求分析法**[5] に着目し，ゴール間関係を利用した競合の検出法を検討する．また要件 2 を満たすために**形式手法**[6] を導入し，設計段階での仕様テストによる競合の検出と対処手順の検証を実現する．仕様をテストするには実行可能な形式仕様を構築する必要があるため，自己適応システムの内部プロセス記述に必要な形式仕様の構造とその構築法を検討し，要求モデルを利用した形式仕様の生成法も提案する．要求分析法と形式手法の 2 つの手法を用いることで，要求モデルにおいて競合が発生する可能性のある箇所を特定し，その後仕様テストにより詳細な競合の検出と対処手段の妥当性を検証するという 2 段階の競合対応手段を実現する．

3. 競合を考慮した形式仕様の構築

2 章で示した要件を満たす自己適応システムの開発プロセスを実現するために，本研究ではゴール指向要求分析法 KAOS[7),8)] と形式仕様記述言語 VDM++[9),10)] を利用した競合検出プロセスを提案する．**図 1** は提案する開発プロセスを示したものであり，大きく以下の

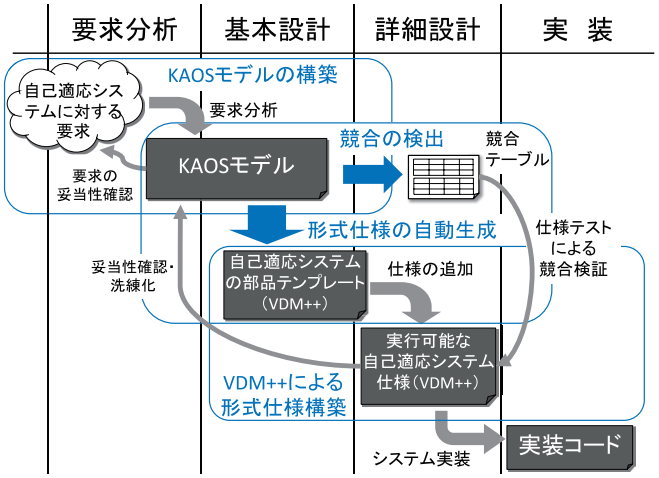

図 1　提案する自己適応システムの開発プロセス
Fig. 1　Proposed development process for self-adaptive systems.

3 つのフェーズにより構成される．

KAOS モデルの構築： 開発者はまず，自己適応システムに対する要求を KAOS により分析する．この分析は通常の KAOS 分析法に従うが，提案手法では KAOS モデルの記述制約を付与することにより，KAOS モデル上で構築すべきプロセスを同定し，競合が発生する可能性のある状況を特定する．

競合の検出および形式仕様の自動生成： 次に本研究で提案する競合検出ツールを用いて，構築した KAOS モデルからプロセス間競合が発生する可能性のある状態を示す競合テーブルを自動生成する．また，仕様生成ツールを用いて，KAOS モデルから VDM++の仕様テンプレートも自動生成する．この仕様テンプレートは，自己適応システムの仕様を構成するクラス群に対応する．

VDM++による形式仕様構築・競合検証： 開発者は得られたテンプレート群に振舞いに関する記述などを追加することで，自己適応システムの実行可能な形式仕様を完成させる．その後，構築した形式仕様に対して仕様テストを実施することで，競合の発生状態を確認するとともに競合対処に関する仕様の妥当性を検証する．

本章では以降，自己適応システムの例として文献 11) などでも用いられている清掃ロボットの構築を例にあげ，提案手法を説明する．

3.1　KAOS モデルの構築

KAOS は Lamsweerde らが提唱するゴール指向要求分析法であり，識別したゴールを達成するための機能やオペレーションを系統的に導出することができる分析手法である．KAOS 分析では要求はゴールツリーを用いて具体的な機能や非機能要求に分解され，さらにこれらを実現するために満たさなければならない個々の状態が抽出される．提案手法では，ゴール間の階層構造が記述可能であり，エンティティやオペレーションなどのシステム設計モデル要素との関係も記述できることから，ゴール指向要求分析法の中でも特に KAOS を用いる．加えて，競合検出と形式仕様へのモデル変換を実現するために，KAOS モデルに対して以下の記述制約を付与する．

ゴール記述と責務の割当て： KAOS では通常，ゴールモデルによりゴールをシステムやユーザなど個々のアクタ*1が単独で達成可能な粒度にまで分解し，分解されたゴール（要件と呼ばれる）を各アクタの責務として割り当てる．提案手法では，まずゴールモデルからシステムが持つべきプロセスに対応するゴールあるいは要件*2を抽出し，自己適応システムに該当するエージェントに割り当てる．その後，各プロセスがとりうる状態群を抽出するために，抽出した各ゴールに milestone-driven refinement [12] を適用して，ゴール達成のために経由しなければならない状態群へと分解し，これらをゴールモデル内で左から右へと達成すべき順に配置する．たとえば，**図 2** の清掃ロボットに関する KAOS ゴールモデルからは，ごみ処理プロセスとバッテリー管理プロセスが清掃ロボットの持つべきプロセスと考えられるため，これらを過不足なく包含するゴール「ごみが処理されている」とゴール「バッテリーが維持されている」を「清掃ロボット」エージェントの責務として割り当てる．その後，たとえばごみ処理に対しては，ごみを発見し，ごみに近づき，清掃するという状態を経由する必要があるため，これらに該当するサブゴールを記述し，達成すべき順に従って配置する．

エンティティの抽出： 従来の KAOS ではシステムが対象とする環境中のオブジェクトを抽出し，エンティティとして記述するが，提案手法では競合検出のために，ハードウェアなどのシステム構成要素も明示的にエンティティとして記述し，関連するゴールとの間に `Concerns` 関係を定義する．たとえば図 2 の清掃ロボットの例では，ゴール「ごみが見つかっている」を達成するには，対象とする「ごみ」と，ごみを検知するための「センサモジュール」が関連エンティティとして考えられるるため,「ごみ」と「センサモジュール」と

⋆1 KAOS ではアクタに対応する概念をエージェントと呼ぶ．
⋆2 以降，特に断らない限り，ゴールと要件のことを総称してゴールと呼ぶ．

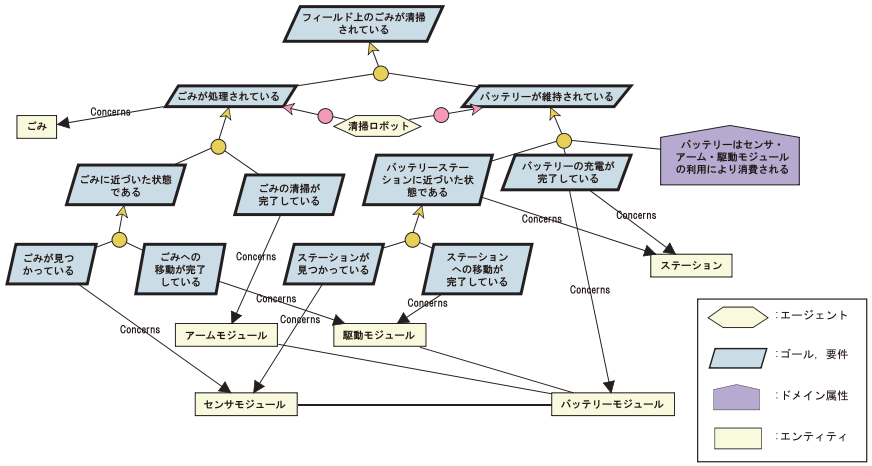

図 2　清掃ロボットに対する KAOS ゴールモデル
Fig. 2　KAOS goal model for cleaning robot.

の間に関連を定義する．ただし,「ごみ」に関しては，他のサブゴールや親ゴールも関与するため，図 2 では上位ゴールからの関連に集約している．提案手法ではこのように，各プロセスを milestone-driven refinement によりとりうる状態群に分解し，それらとエンティティ間の関連を定義することで，競合発生の可能性のある状態とエンティティを特定する．

オペレーションの定義： 提案手法では自己適応システムが割り当てられたすべてのゴールとそのサブゴールに対して，ゴールが表現する状態を達成するためのオペレーションを定義する．**図 3** はゴール「ごみへの移動が完了している」と「ステーションへの移動が完了している」を達成するオペレーションとして「移動する」を定義した例である．定義されたオペレーションは，自己適応システムを構成するコンポーネントとして次フェーズで形式仕様テンプレートとして自動生成される．従来の KAOS 分析ではゴールツリー上で葉として記述される機能（要件）に対してのみオペレーションを定義するが，提案手法では自己適応システムが割り当てられたすべてのゴールに対してオペレーションを定義する．これは，自己適応システムでは割り当てられた機能（要件）だけでなく，振舞いの実行順序や代替となる振舞いの切替えを制御する機構も必要であり，これらを実現するオペレーションを，子ゴールを集約する中間ノードの責務として抽出するためである．

以上の記述規則に従った KAOS モデルを構築し，次節で説明する 2 種類のツールを導入

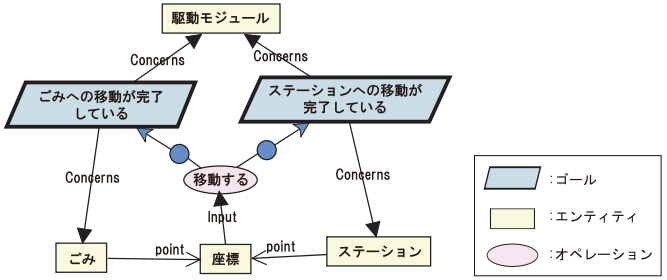

図 3　オペレーションの記述例
Fig. 3　An example of operation definition.

することで，競合が発生する状況を表現する競合テーブルと，形式仕様のテンプレートを生成することが可能となる．なお，本研究ではツールによるモデル情報の自動取得のために，KAOS のモデル情報を XML 形式で出力可能なモデリングツール Objectiver [13] を用いて KAOS モデルを記述する．

3.2　競合の検出および形式仕様の自動生成

提案プロセスでは，構築した KAOS モデルをもとに，競合発生の可能性がある状態を示す競合テーブルと自己適応システムの形式仕様を構築する．本研究では競合検出ツールと仕様生成ツールの 2 種類のツールを導入することで，競合テーブルと仕様テンプレートの自動生成を実現する．

3.2.1　競合テーブルの生成

まず，競合検出ツールは定義 2 に示す競合テーブル生成アルゴリズムを実装したものであり，3.1 節の記述制約に従った KAOS モデルから，競合が発生する可能性のある状態とその原因となるリソースを特定する競合テーブルを csv 形式で生成する．

［定義 2］競合テーブル生成アルゴリズム：

(1)　エージェントに割り当てられている各ゴール g_i（$1 \leq i \leq N$）に対して，g_i を根とするサブツリーの葉 $g_{i1}, ..., g_{im}$ を左から順にならべた状態の遷移リスト $[g_{i1}, ..., g_{im}]$ をプロセス $process_i$（$1 \leq i \leq N$）として定義する．

(2)　親ゴールに定義されている `Concerns` 関係を子ゴールへ継承させ，すべてのゴール–エンティティ間の `Concerns` 関係 concerns (goal, entity) の集合を $\mathcal{C}oncerns$ として抽出する．

(3)　$\mathcal{C}oncerns$ 内に異なるプロセスに属するゴール群と `Concerns` 関係を持つエンティ

ティがあれば，このエンティティconfRes を**競合リソース** とし，競合リソースと Concerns 関係を持つゴールのペア (g_{ik}, g_{jl}) を**競合状態**とする．競合リソースの集合 $\mathcal{C}onfResources$ と，競合状態の集合 $\mathcal{C}onflicts$ は以下に定義するとおりである．

$$\mathcal{C}onfResources = \{\text{confRes} \mid \text{concerns}(g_{ik}, \text{confRes}) \in \mathcal{C}oncerns \\ \& \,\text{concerns}(g_{jl}, \text{confRes}) \in \mathcal{C}oncerns \,\&\, i \neq j\}$$

$$\mathcal{C}onflicts = \{(g_{ik}, g_{jl}) \mid \text{concerns}(g_{ik}, \text{confRes}) \in \mathcal{C}oncerns \\ \& \,\text{concerns}(g_{jl}, \text{confRes}) \in \mathcal{C}oncerns \,\&\, i \neq j\}$$

(4)　(3) で得られた各競合状態 (g_{ik}, g_{jl}) に対して，それぞれのプロセスを $process_i = [g_{i1}, ..., g_{ik}, ..., g_{im}]$，$process_j = [g_{j1}, ..., g_{jl}, ..., g_{in}]$ としたときに，競合状態以降にとりうるすべての状態 (g_{ip}, g_{iq})（$k \leq p \leq m$，$l \leq q \leq n$）を**事後競合状態**として抽出する．

(5)　各競合状態に対して，競合状態と事後競合状態の情報をもとに競合テーブルを構築する．競合テーブルは各セル c_{ij}（$1 \leq i \leq m$，$1 \leq j \leq n$）に次の値をとるものである．

$$c_{ij} = \begin{cases} - & (\text{競合なし：}\ i < k \vee j < l) \\ ◎ & (\text{競合状態：}\ i = k \wedge j = l) \\ ○ & (\text{事後競合状態：}\ i \leq k \wedge j \leq l \wedge \neg(i = k \wedge j = l)) \end{cases}$$

(6)　もし，2 つのプロセスが同一リソースに対して複数の競合状態を持つ場合は，(5) で構築した競合テーブルを 1 つの競合テーブルに集約する．集約後の競合テーブルにおける各セル c'_{ij} の値は次のとおりである．

$$c'_{ij} = \begin{cases} - : & \text{すべての競合テーブルにおいて競合なし}(c_{ij} = -) \\ ◎ : & \text{いずれか 1 つの競合テーブルにおいて競合状態である}(c_{ij} = ◎) \\ ○ : & \text{1 つ以上の競合テーブルにおいて事後競合状態}(c_{ij} = ○)\ \text{であり，} \\ & \text{かつ，いずれの競合テーブルにおいても競合状態ではない}(c_{ij} \neq ◎) \end{cases}$$

□

この競合テーブル生成アルゴリズムは，たとえば図 2 のゴールモデルに対しては，センサモジュールと駆動モジュールが複数プロセスにまたがって Concerns 関係により関連付けられているため，これらを競合リソースとして判断し，関連付けられているゴールの対を競合状態として抽出する．これは，たとえば駆動モジュールに関しては，2 つのプロセスにおいて，それぞれがごみが存在する地点への移動と，ステーションへの移動を達成しようとした場合に競合発生の可能性があることを示したものである．

表 1　競合テーブルの例（実際に生成されるテーブルは csv 形式）
Table 1　Example of conflict table.

競合リソース：駆動モジュール

	ステーションが見つかっている	ステーションに向かって移動することができる	バッテリーを充電することができる
ごみが見つかっている	–	–	–
ごみに向かって移動することができる	–	◎	○
ごみを清掃することができる	–	○	○

駆動モジュールに対する競合テーブルの例を**表 1** に示す．定義 2 に示したアルゴリズムでは，競合状態の要素となるゴール以降の状態の対も事後競合状態として抽出する．事後競合状態は，競合リソースが原因で不具合が発生する可能性のある状態に対応し，環境の変化により起動プロセスが変化する自己適応システムにおいて，競合状態のゴール対を経由しなかった場合においても競合リソースによる潜在的な競合が発生しうる状態を示すためのものである．この事後競合状態は，仕様テスト時にエラーを検出した状態から，どのリソースにより引き起こされた競合であるかを特定するためにも用いることができる．

3.2.2　仕様テンプレートの生成

もう一方の仕様生成ツールは，KAOS モデルを VDM++の形式仕様へと変換するものである．VDM++は VDM-SL[14),15)] のオブジェクト指向拡張であり，本研究では，仕様のテストが可能であり，自己適応システムを構成するコンポーネント群をクラスの概念を用いて定義できるという特質から VDM++を用いる．仕様生成ツールは筆者らがすでに提案している KAOS 記述から VDM++形式仕様への変換ツール[16),17)] を拡張したものであり，形式仕様への変換は**表 2** に示す対応関係に従う．

仕様生成ツールは仕様のテンプレートとして，システムクラス，オブジェクトクラス，コンポーネントクラスの 3 種類のクラスファイルを生成する．このうち，システムクラスは自己適応システムの本体に対応するクラスであり，KAOS モデル中のエージェントに関する情報をもとに生成する．オブジェクトクラスは，ハードウェアなどのシステム構成要素や環境中に存在するオブジェクトに対応するクラスであり，KAOS モデル中の各エンティティに対してエンティティ間の関連とエンティティの属性情報をもとに生成する．また，共通の

表 2　KAOS モデルと VDM++仕様との対応関係
Table 2　Correspondence between KAOS model and VDM++ specification.

KAOS モデル	VDM++仕様
エージェント	自己適応システム
エンティティ	・システム構成要素（H/W など） ・環境に存在するオブジェクト ・型
オペレーション	コンポーネント

型に関する定義を集約したクラスもオブジェクトクラスとして生成する．これらのクラスの生成は，文献 16)，17) に示した変換ツールの生成法に従ったものである．

一方で，コンポーネントクラスは KAOS モデルの各オペレーションに対応付けられたクラスであり，提案手法では KAOS モデル上で定義したオペレーションに対して 1 対 1 の関係でコンポーネントクラスを生成する．生成されるコンポーネントクラスはそれぞれがスレッドとして記述され，起動方法により並行動作が実現可能であることが特徴である．たとえば，図 3 で定義したオペレーション「移動する」に対して生成されるコンポーネントクラスのテンプレートは**図 4** に示すとおりである．この例では，14〜19 行目がオペレーションの定義部に該当し，24 行目以降がオペレーションのスレッド化に該当する．また，`action` メソッド内にはコンポーネントの処理開始と終了を示す出力文が含まれているが（16，18 行目），これらは仕様テストの実行結果からシステムの状態遷移を把握するために用いる．

3.3　VDM++による形式仕様構築・競合検証

3.3.1　仕様の追加

仕様生成ツールにより各クラスファイルのテンプレートが生成されると，生成された VDM++の各テンプレートに仕様を追加することで仕様を完成させる．VDM++では，関数や操作の本体を実行可能なステートメントで記述した陽仕様と，関数や操作の本体に実行可能なステートメントを記述せず，事前・事後条件により満たすべき条件を宣言的に記述した陰仕様の 2 種類の記述スタイルを利用することができる．提案手法では仕様実行によるテストで競合を検出し，対処法を検討するため，陽仕様を構築する．仕様構築手順は，まず環境や構成要素をモデル化しているオブジェクトクラスのテンプレートに仕様を追加してオブジェクトクラスを完成させ，その後，環境や構成要素を監視，制御するコンポーネントクラスの仕様をプロセス単位で完成させる．特にコンポーネントクラスのテンプレートでは図 4 の 16〜18 行目に示されるように操作の本体が標準出力へ実行状態を印字するもののみであるため，具体的な処理内容を書き加える．ゴールツリー上の葉の部分に該当するコンポーネントにはエンティティを操作する記述を加え，内部ノードに該当するコンポーネントには関連コンポーネントの制御手順を記述する．オブジェクトクラスやコンポーネントクラスの仕様構築にあたっては，操作（operations）を実行する前に期待する状態である事前条件と，操作実行後に期待される状態である事後条件を可能な範囲で記述する．これは，仕様テスト実行時に事前・事後条件に違反するエラーも抽出することで競合検出の可能性を

```
 1 class 移動する
 2
 3 instance variables
 4 private att 清掃ロボット: 清掃ロボット;
 5
 6 operations
 7 -- constructor
 8 public 移動する:清掃ロボット ==> 移動する
 9 移動する (a 清掃ロボット) == (
10     att 清掃ロボット:=a 清掃ロボット;
11     return self;
12 );
13
14 public action:座標 ==> ()
15 action(a 座標) == (
16     let - = new IO().echo("Start: 移動する") in skip;
17     -- TODO: describe process statement --
18     let - = new IO().echo("End: 移動する") in skip;
19 );
20
21 public finish:() ==> ()
22 finish() == skip;
23
24 sync
25     per finish => #fin(action) > #fin(finish);
26     per action => #active(action) = 0;
27
28 thread
29     action();
30     -- TODO: add argument --
31
32 end 移動する
```

図 4　生成されるコンポーネントクラスの例
Fig. 4　An example of generated component classes.

高めるためである．

3.3.2　コンポーネントの実行制御

自己適応システムでは複数のプロセスが並行動作する必要があるため，提案手法では仕様テストにおける各コンポーネントの実行形態として，逐次実行と並列実行の 2 種類を考慮する．まず，VDM++においてはスレッドを起動した場合，明示的に適切なタイミングでスレッドを停止させる必要があるため，提案手法では図 4 のテンプレートにも示されるように，スレッドを明示的に終了させる `finish` メソッドをすべてのコンポーネントに用意する[*1]．一方，スレッドの起動には VDM++で提供される `start` メソッドを利用し，スレッド本体の処理内容を `action` メソッド内に定義する．図 4 の 28，29 行目の記述は，スレッドとして `action` メソッドを実行するためのものである．

また，提案手法ではコンポーネントの同期制約として以下のスレッド制御記述を導入する．

- `per finish => #fin(action) > #fin(finish);`
- `per action => #active(action) = 0;`

`#fin` と `#active` は VDM Tools が提供するスレッドの実行状況を返す履歴関数であり，`#fin` は実行が完了したスレッド数を，`#active` は現在実行中のスレッド数を返す．したがって，1 つ目の制約は `finish` メソッドよりも `action` メソッドの実行終了数が多い場合にのみ `finish` メソッドが発動可能，つまり 2 つのメソッドの実行順序を制御するためのものであり，2 つ目の制約は `action` メソッドが実行中でなければ同メソッドが即座に実行可能であることを示すものである．

以上の記述を用いて各コンポーネントを記述することで，コンポーネントの逐次実行と並列実行の双方が統一的な記述により可能となる．逐次実行の場合は，一方のコンポーネントの `start`，`finish` メソッドを呼び出した後にもう一方のコンポーネントの `start`，`finish` メソッドを呼び出せばよく，並列実行の場合は，並列実行させたいコンポーネント群のすべての `start` メソッドを呼び出した後に `finish` メソッドを呼び出せばよい．

3.3.3　仕様テストによる競合の検証手順

各プロセスの仕様が完成すると，仕様テストを用いて競合を検出し，対処法を検証する．提案手法では，VDM++のインタプリタである VDM++ Toolbox[18] を用いて仕様テストを実行する．本研究における仕様テストの目的は，競合の検出と対処法の検証であり，仕様テストは競合の発生を確認することから始める．提案手法では，まず機能を実現するプロセス群から統合，つまりこれらのプロセス群の仕様を並列に実行させ，競合を検出し，対処法を検証する．その後，制約を遵守するためのプロセスを弱い制約に対するプロセスから強い制約に対するプロセスの順に統合する．これらのプロセスは 1 つずつ統合し，その都度仕様テストを実行することで，競合の検出・対処法の検証とともに，追加した機能や制約が実際に反映されているかどうかを確認することが容易となる．

仕様テストにおいては，まず競合が発生するかどうかを検査し，その後対応法を検討する．競合が発生すると，仕様テストの出力から各プロセスの状態を判断し，この状態を競合状態あるいは事後競合状態として持つ競合テーブルを検索することで競合リソースを同定する．競合への対処手段を仕様に追加して競合が発生しなくなったことを確認すると，競合テーブルの該当箇所に競合解決のチェックをつける．競合への対処手段を追加することで他リソースに対して新たな関連が生じる場合や，プロセスが達成すべき状態集合に変更が生じる場合は，KAOS モデル上で情報を変更し，競合テーブルを再構築する．その一方で，対処手段を追加することで状態遷移先としてとりえない事後競合状態が生じた場合は，それらの状態にも競合解決のチェックをつける．競合テーブルのすべての競合状態，事後競合状態に対して競合解決のチェックがつけられるまで，これを繰り返す．

4. 評　　価

提案手法の適用可能性と有用性を検証するために，本研究では 2 種類の自己適応システムの開発を対象とした形式仕様構築実験を実施した．本章では実験内容と実験結果を示し，提案手法の有効性を評価する．

4.1　実験 1：清掃ロボット

4.1.1　実 験 概 要

まず，3 章で例題として用いた清掃ロボットに対して，提案手法に従った競合の検出と対処方法の検討を実践した．本実験で対象とした清掃ロボットに対する要求は以下のとおりであり，清掃ロボットに対する自己適応システムとしての要求は，ごみ清掃中における環境変化，つまりバッテリー残量低下への対応である．

［清掃ロボットに対する要求］ 清掃ロボットはフィールド上のごみを検知して清掃することができる．フィールド上にはごみと清掃ロボットのバッテリーを充電するためのバッテリーステーションが存在する．清掃ロボットはバッテリーステーションで随時バッテリーを充電しながら，フィールド上のすべてのごみを清掃することができる．

*1 ただし，動作結果として値を返すコンポーネントでなければ，何も処理をしない “`skip`” をテンプレートどおりに定義しておくだけでよい．

表 3　センサモジュールに対する競合テーブル
Table 3　Conflict table for sensor module.

競合リソース：センサモジュール

	ステーションが見つかっている	ステーションに向かって移動することができる	バッテリーを充電することができる
ごみが見つかっている	◎	○	○
ごみに向かって移動することができる	○	○	○
ごみを清掃することができる	○	○	○

実験ではまず，清掃ロボットに対して KAOS による要求分析を実施し，提案手法で追加した記述規則をもとに KAOS モデルを構築した．続いて，提案手法で導入する 2 種類のツールを用いて，清掃ロボットに関する競合テーブルと仕様テンプレートを生成した．最後に，生成されたテンプレートを利用して順次仕様を構築し，競合テーブルを利用した仕様テストによる競合の検出・対処手段の検証を通じて仕様を完成させた．なお，本実験では以下の前提条件に基づいて仕様を構築している．

- センサモジュールは現在地から最も近くにあるごみ，あるいはステーションを発見することができる．
- ロボットが所持するバッテリーは，周囲の探索，移動，清掃を実施することで消費される．バッテリーはステーションにより最大容量まで充電される．

4.1.2　実験結果

本実験で構築した KAOS ゴールモデルは図 2 に示したものであり，競合検出ツールにより生成された競合テーブルから，センサモジュールと駆動モジュールが競合リソースであることと，それぞれの競合状態，事後競合状態を確認した．駆動モジュールとセンサモジュールに対する競合テーブルは，それぞれ表 1，**表 3** に示すとおりであった．

次に，仕様生成ツールを用いて，構築した KAOS モデルから VDM++ 仕様のテンプレートを生成した．**表 4** は本実験で構築した清掃ロボットに対する形式仕様の構成である．このうち標準ライブラリである IO クラス，MATH クラスと，仕様テストのために構築したテストケースである Test クラスを除いたすべてのクラスのテンプレートが KAOS モデルから生成されることが確認できた．

仕様の構築手順は提案手法に従い，はじめにオブジェクトクラスを構築し，その後各プロセスごとにコンポーネントクラスの仕様を構築した．**図 5** は，ごみを発見してごみの場所

表 4　構築した VDM++仕様の構成
Table 4　Composition of formal specifications described in VDM++.

分類	クラス名	記述内容	初期行数	最終行数
システムクラス	清掃ロボット	自己適応システム本体	21	55
オブジェクトクラス	センサモジュール	センシング機能（システム構成要素）	28	100
	駆動モジュール	移動機能（システム構成要素）	28	64
	アームモジュール	清掃機能（システム構成要素）	28	69
	バッテリーモジュール	バッテリー管理機能（システム構成要素）	17	34
	ごみ	フィールド上に存在するごみ	36	37
	ステーション	フィールド上に存在するステーション	36	37
	フィールド	清掃ロボットが配置されるフィールド	37	58
	座標	2 次元座標（オブジェクトの場所指定に利用）	36	36
	CommonType	共通の型を定義	11	11
コンポーネントクラス	対象を発見する	センサを利用してオブジェクトを発見	36	46
	移動する	駆動モジュールを利用して目標の座標へ移動	32	46
	ごみに近づく	ごみを見つけて移動	31	51
	ごみを清掃する	アームを利用して現在地点のごみを清掃	31	34
	ごみを処理する	フィールド上のすべてのごみを清掃	31	45
	ステーションに近づく	ステーションを見つけて移動	31	47
	バッテリーを充電する	現在地点のステーションでバッテリー補給	31	32
	バッテリーを維持する	バッテリー残量を定常管理	31	52
テスト仕様	Test	テストケースを記述	–	118
標準ライブラリ	IO	入出力ライブラリ	–	–
	MATH	数学ライブラリ	–	–

まで移動する責務を持つコンポーネント「ごみに近づく」の `action` メソッド記述である．本コンポーネントでは，まず最も近いごみを探し出し，結果として得られたごみのある場所（座標）に向かって移動するといった動作の逐次実行が必要であるが，5〜10 行目の記述のように 3.3.2 項で示した実行制御記述に従った `start`，`finish` メソッドの呼び出しにより逐次実行が実現できることも，仕様の実行により確認できた．

各プロセスの仕様が構築できると，続いて，機能実現プロセスに該当するごみ清掃プロセスに対して，制約遵守プロセスに該当するバッテリー管理プロセスの仕様を統合した．競合への対処手順の記述に先立って，ごみ処理とバッテリー管理に関するプロセスを並列実行させた仕様テストの結果を**図 6** に示す．このテスト結果は，清掃ロボットがごみに近づこうとするのと並行してステーションにも近づこうとするため，ロボットが移動終了後に現在地点にごみが存在することを検査する「ごみに近づく」の事後条件（図 5 の 14〜16 行目）がエラーとなり異常終了したことを示すものである．本エラーは仕様テストの実行ログと競

```
1   public action:() ==> ()
2   action() == (
3       let - = new IO().echo("Start: ごみに近づく") in skip;
4       att 対象を発見する. 目的 := "ごみ";
5       start(att 対象を発見する);
6       let dest = att 対象を発見する.finish() in (
7           ごみの場所 := dest;
8           att 移動する. 目的地 := dest;
9           start(att 移動する);
10          att 移動する.finish();
11      );
12      let - = new IO().echo("End: ごみに近づく") in skip;
13  )
14  post
15      (att 清掃ロボット. 現在地.x = ごみの場所.x
16       and att 清掃ロボット. 現在地.y = ごみの場所.y);
```

図 5　コンポーネントの順次実行（「ごみに近づく」クラスの action メソッド）
Fig. 5　Components Sequential execution (in action method of "ApproachDust" class).

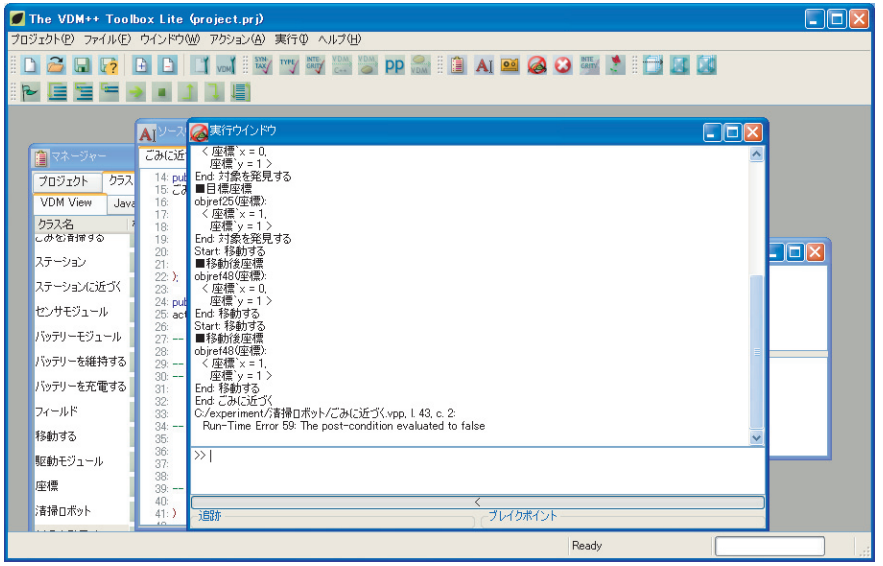

図 6　事後条件エラーによる競合の検出
Fig. 6　Conflict detection by post-condition error.

合テーブルから，状態「ごみに向かって移動することができる」と状態「ステーションに向かって移動することができる」の競合状態において発生したエラーであり，駆動モジュール利用時に発生した競合により引き起こされたエラーであることが分かった．なお，エラーを検知した事後条件記述を仕様から削除してテストを再実行したところ，後続のごみ清掃状態において，「アームモジュール」の事前条件エラーが発生することが確認できた．この場合も，競合テーブルから事後競合状態であることが判断でき，競合の同定が可能である．

同様に，センサモジュールによる競合も仕様テストにより検出され，これらの競合への対処として，バッテリー管理プロセスを構成するコンポーネント「バッテリーを維持する」内にプロセス中断および再開許可のシグナルを送る処理を追加し，ごみ処理プロセスに対応するコンポーネント内にシグナルに応じた処理の中断，再開処理を追加した．これらの制御仕様を追加することで，清掃ロボットがバッテリーを維持しながらフィールド上のすべてのごみを清掃することが仕様テスト上で確認され，競合に対処できる仕様が構築されたことが確認できた．

本実験では，仕様構築コストの評価を目的として，仕様生成ツールが生成した仕様と完成した仕様との比較も実施した．表 4 中の「初期行数」は仕様生成ツールが生成した仕様の行数を，「最終行数」は記述を追加して最終的に完成した仕様の行数を示したものである．表 4 と記述追加前後の仕様の比較により，オブジェクトクラスに関してはセンサモジュールなどのシステム構成要素や環境オブジェクトの振舞いを記述する必要があるため，手作業により追加しなければならない記述が多いことが分かった．一方でプロセスを構成するコンポーネントクラスに関しては，多くは action メソッドの本体記述，事前事後条件記述と関連コンポーネントの参照追加のみで仕様を構築できることが確認できた．逐次実行と並列実行も 3.3.2 項で示した制御記述に従うことで可能であり，仕様テンプレートの構造を利用した効果的な仕様構築が可能であることが確認できた．

4.2　実験 2：Web サーバ管理システム

4.2.1　実 験 概 要

本研究ではさらに，提案手法の汎用性を評価するために Web サーバ管理システムの開発を想定した仕様構築実験も実施した．本実験における Web サーバ管理システムに対する要求は以下のとおりであり，自己適応システムとしての要求は，同時接続数増加への適応としてのコンテンツの画質変換と，サーバの CPU およびメモリ利用率増加への適応としての

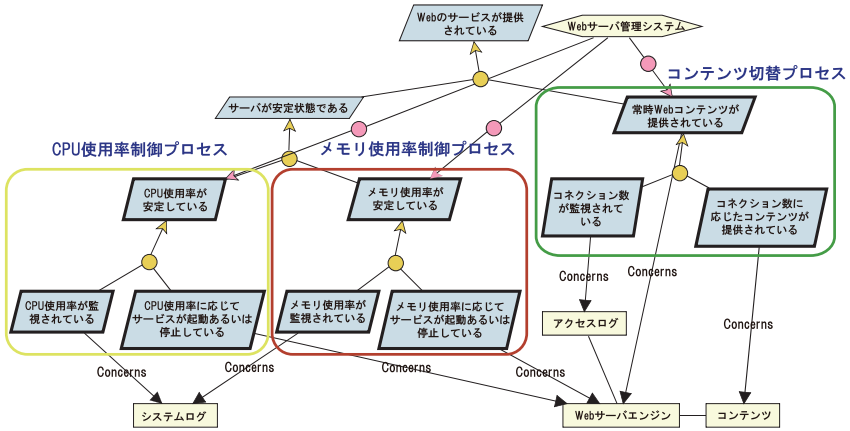

図 7　Web サーバ管理システムに対する KAOS ゴールモデル
Fig. 7　KAOS goal model for web server management system.

Web サービスの一時的な停止である．

[Web サーバ管理システムに対する要求] Web サーバ管理システムは，安定した Web サービスの提供を実現するために Web サーバ上に存在する管理システムである．本システムは，クライアントの接続数を監視し，接続数が急増した場合にも，提供コンテンツを低画質版に切り替えることで，接続数増加にともなうレスポンスの低下やサービス停止を回避する．一方で，Web サーバ本体の CPU やメモリの使用率も監視し，サーバが過負荷状態となった場合には，一時的に Web サーバエンジンを停止する．

4.2.2　実 験 結 果

本実験において構築した KAOS ゴールモデルは**図 7** に示すとおりであり，本 KAOS モデルから，CPU 使用率を制御するプロセス（CPU 使用率制御プロセス）と，メモリ使用率を制御するプロセス（メモリ使用率制御プロセス），さらにコネクション数に応じてコンテンツを切り替えるプロセス（コンテンツ切替プロセス）の 3 つのプロセスを同定した．このうち，コンテンツ切替プロセスは機能実現プロセスとして，その他 2 つのプロセスは制約遵守プロセスとして判断した．

続いて，競合検出ツールにより 4 種類の競合テーブルが生成された．CPU 使用率制御プロセスとメモリ使用率を制御プロセスに関しては，システムログと Web サーバエンジンに対する競合テーブルが生成され，コンテンツ切替プロセスに対しては，他の 2 つのプロセスに対して Web サーバエンジンを競合リソースとする競合テーブルがそれぞれ生成された．**表 5** は，CPU 使用率制御プロセスとコンテンツ切替プロセスに対する Web サーバエンジンを競合リソースとした競合テーブルである．

表 5　Web サーバエンジンに関する競合テーブルの 1 つ
Table 5　One of generated conflict tables for web server engine.

競合リソース：Web サーバエンジン

	コネクション数が監視されている	コネクション数が閾値を超えた場合，低画質のコンテンツが提供される
CPU 使用率が監視されている	–	–
CPU 使用率が閾値を超えた場合，サービスが起動・停止される	◎	◎

表 6　Web サーバ管理システムに対する VDM++仕様の構成
Table 6　Composition of VDM++ specifications for web server management system.

分類	クラス名	記述内容	初期行数	最終行数
システムクラス	Web サーバ管理システム	自己適応システム本体	21	47
オブジェクトクラス	システムログ	サーバが出力するログ	18	40
	Web サーバエンジン	Web サービスを提供するエンジン	36	73
	アクセスログ	Web サーバエンジンが出力するアクセスログ	18	37
	コンテンツ	Web サーバエンジンが提供するコンテンツ	18	26
	CommonType	共通の型を定義	6	6
コンポーネントクラス	CPU を監視する	システムログを参照して CPU 使用率を監視	32	37
	サービスを起動・停止する	Web サーバエンジンを起動，停止	32	34
	CPU 使用率を制御する	CPU 使用率が一定値を超えないよう管理	31	51
	メモリを監視する	システムログを利用してメモリ使用率を監視	32	34
	メモリ使用率を制御する	メモリ使用率が一定値を超えないよう管理	31	49
	コネクション数を監視する	アクセスログを参照してコネクション数を監視	32	34
	コンテンツを置き換える	高画質版および低画質版コンテンツを置換	31	36
	提供コンテンツを制御する	コネクション数によりコンテンツの画質を変更	31	54
テスト仕様	Test	テストケースを記述	–	235
標準ライブラリ	IO	入出力ライブラリ	–	–
	MATH	数学ライブラリ	–	–

仕様生成ツールを用いて生成された仕様テンプレートと最終的な仕様の構成は**表 6** に示すとおりである．仕様上の競合検出に関しては，まず，CPU 使用率制御プロセスとコンテンツ切替プロセスを統合し，表 5 に示した Web サーバエンジンを競合リソースとする競合

を仕様テストにより検出した．仕様テストでは，CPU 利用率増加にともない Web サーバエンジンが停止されることで常時 Web コンテンツを提供するという機能が実現できなくなるという競合を，コンポーネント「提供コンテンツを制御する」に記述した事後条件エラーを検出することで確認できた．本競合に対しては，同コンポーネントの制約として，サーバの負荷が高い（CPU 使用率およびメモリ利用率が高い）場合は例外とすることで競合を回避できることを確認した．

CPU 使用率制御プロセスとメモリ使用率制御プロセス間の競合に対しては，まずシステムログに関しては，両プロセスともに参照するのみであり，対処すべき競合として扱う必要はないと判断した．一方で Web サーバエンジンに関しては，仕様テストから CPU 利用率とメモリ利用率がともに増加した場合，一方が Web サーバエンジンを停止した後に他方も停止しようとする競合を，Web サーバエンジンクラスの操作「サーバを停止する」に記述した事前条件エラーを検出することで確認できた．本競合に対しては，各プロセスが Web サーバエンジンを起動，停止すべきと判断した際に，他方のプロセスの監視状況と現在の Web サーバエンジンの状態チェック処理を追加することを競合対処手段とすることで仕様テストにおいてエラーが解消されることが確認できた．以上の仕様変更と再仕様テストにより，競合テーブルにより抽出されたすべての競合状態，事後競合状態において対処法が有効であることを確認した．

4.3 考　　察

以上の 2 種類の形式仕様構築実験の結果をもとに，提案手法の有効性を 2 章で定義した各要件に対して評価する．

まず，要件 1 の「適応状況により発生状態が異なるプロセス間競合に対して，網羅的な検出が可能である」に関しては，提案手法では KAOS モデルと仕様テストを用いた 2 段階の競合検出プロセスを適用している．提案手法では，まず KAOS モデル上でプロセス集合を同定し，同定されたプロセス群とリソースとの関連から競合テーブルを生成することで，競合が発生しうる状態を特定する．このような分析手段は，状況によって起動プロセス群が異なる自己適応システムに対して，システムがとりうる状態の網羅的なチェックによる検出を避けるという観点から，1 つの有効な競合分析手段であると考えられる．

ただし KAOS モデル上で競合を特定するには，競合リソースとなりうるエンティティとゴールとの関連が KAOS モデル上で記述されていなければならない．特に，競合発生の状況を特定するためには，プロセスがとりうる状態が KAOS モデル上に記載されている必要がある．提案手法では，親ゴールの達成に必要な状態集合を抽出する milestone-driven refinement を利用することで，プロセスの目的から状態集合への分解を支援している．ゴール指向分析の難しさは一般にゴールの抽出および分解にあるが，筆者らはこの難しさに対してはトップダウンとボトムアップの双方向のアプローチの提供が有効であると考えている．提案手法においては，ゴールモデルにおけるサブツリー単位でのプロセス分割と，milestone-driven refinement による各プロセスから達成すべき状態への分解がトップダウンアプローチに該当し，競合リソースに対応するエンティティとゴールとの関連の定義がボトムアップアプローチに該当する．また，提案手法では KAOS モデルと仕様記述とを関連付けているため，仕様構築段階での検討内容による KAOS モデルの洗練化も期待できる．

一方で，実験 2 におけるシステムログのように，KAOS モデル上で特定された競合が自己適応システム上で必ずしも起こりうるわけではない．開発者は得られた競合可能性の情報をもとに，実際に自己適応システム上で考慮すべき競合であるかを判断する必要がある．ただし，競合リソースに該当するエンティティと状態に該当するゴールとの関連が KAOS モデル上に定義されている限りにおいて，提案手法では競合リソースと競合状態を漏れなく抽出することが可能である．これは，提案手法では状態遷移による発見ではなく，エンティティ（リソース）を対象として，それを利用するゴール（状態）の組合せを形式的に取得するためである．しかし，提案手法では，ゴールの分解粒度により特定できる競合状態の粒度が異なる．たとえば，ゴールの分解粒度が大きいと，リソースの利用状況を正しく把握できない恐れがある．一方で，ゴールの分解粒度が小さいと，KAOS モデル上で競合状況を詳細に特定することは可能であるが，後継フェーズで構築すべきコンポーネント数が増加し，仕様構築のオーバヘッドが大きくなってしまう．筆者らはこのゴールの粒度に関しては，1 つの独立した操作として抽出できる大きさ，つまりシステム内部変数や環境変数を用いた状態遷移記述ができる粒度が適しているのではないかと考えている．

2 段階目の競合検出プロセスである仕様テストに関しては，競合テーブルを利用することで，実験結果が示すように仕様テスト結果からの競合状態の特定を支援することが可能であるといえよう．また，競合テーブルを仕様構築前の段階で獲得できることは，リソースが競合する状態や競合の影響を受ける事後状態の事前把握を可能とするため，競合対処を意識した仕様構築という観点からも有効であると考えられる．ただし，仕様の記述内容によっては事後競合状態として示される状態に遷移しない場合もある．提案手法では，競合が発生しうる状態を部分集合とする状態集合を競合テーブルにより獲得するが，競合テーブルから実際に競合が発生しうる状態を特定するには，開発者の判断や効率的な仕様テストの実行が必要となる．

要件 2 の「システムを実装することなく，競合に対する対処法を検討することができる」に対しては，提案手法では仕様テストを採用することで，実装プラットフォームを利用することなく競合の検出と対処法の検証を実現している．特に提案手法では，実験結果が示すように，プロセスの並列実行のための制御メカニズムを仕様テンプレートに埋め込むことで仕様構築のコストを軽減し，また，仕様の構築順序を定め，競合テーブルを事後競合状態の把握に利用することで，仕様テストにおける対処手段の効率的な記述と，その効果の適用範囲の把握を支援しているといえる．

提案手法では仕様テストにより競合を検出するため，あらゆる状況における競合の検出を保証するわけではない．このため，競合検出の精度を高めるために，競合状態を効率的に検出できるテストケースが必要となる．しかしその一方で，2 つの実験結果が示すように，仕様テストにおける事前条件・事後条件によるチェックは効果的であるといえる．従来の実装コードに対するテストであれば，競合が発生した箇所でエラー終了するとは限らず，エラーを検出するために十分に状態が特定された複数のテストケースが必要であるが，仕様テストでは各クラスの操作記述に事前条件・事後条件を設定することで，期待した状態から逸脱した時点でのエラー検知が可能である．実験 1 では，エラーを発生した事後条件を削除しても，後続の事後競合状態において事前条件エラーからの競合検出が可能であったが，このように事前・事後条件と競合テーブルの効果的な利用により，競合を検出できる可能性は高まると考える．

5. 適用範囲と関連研究

最後に本手法の適用範囲と関連研究について論じる．まず，本手法では自己適応システムの要求分析に KAOS を利用し，形式仕様の記述と仕様のテストに VDM++を利用している．提案手法では競合を抽出するために，KAOS のようなゴール指向要求記述を用いて，各プロセスとプロセスがとりうる状態が記述できることが前提である．また，提案手法では要求記述とシステム設計モデルとの対応関係を形式仕様テンプレートの生成に利用しているため，状態として記述されたゴールとエンティティやオペレーションとの関係が記述できることも前提となる．KAOS はゴール指向の要求分析法であるだけでなく，概念間の関係を定義するオブジェクトモデルやシステムの動的側面を記述するオペレーションモデルを提供し，ゴールモデルとの明確な対応関係を記述することができるため，提案手法で利用する要求記述法として適している．ゴール指向要求分析法の代表的な手法としては KAOS のほかに i* [19] があるが，i*はステークホルダ間の要求を分析するといった初期の要求フェーズに重点を置いた手法であり，エンティティとの関係まで厳密に定義することができないことから，提案手法においては KAOS の利用が適しているといえよう．

一方で，提案手法での形式仕様記述言語に対する要件としては，仕様をテストできる環境が提供されているという点と，オブジェクト指向に基づいた仕様記述が可能であるという点があげられる．前者は提案する競合検証に必要な特性であり，後者は提案手法に限らず，コンポーネントの切替えにより適応を実現することが主流となっている自己適応システムにおいて，仕様を記述するための必要条件といえる．したがって，仕様のテストをサポートしていない B [20] や Z 記法[21] などの仕様記述言語や，オブジェクト指向に基づいた仕様記述が困難である VDM-SL などは，提案手法で利用する仕様記述言語としては適していない．

提案手法で利用する仕様テストに関しては，競合が検出できるテストケースが用意できることが前提となる．このためには，各プロセスの状態遷移を考慮したテストケースの構築が必要であるが，テストケースの構築には従来のソフトウェアテストの分野[22] での研究成果が利用できるであろう．また，VDM に対しても，テストフレームワークである VDMUnit [10] が提案され，文献 23) など仕様テストのためのテストケース自動生成の試みもある．これらの研究成果を取り入れることで，提案手法における仕様テストの効率化，競合検出の精度向上が期待できる．

仕様に対するテスティングは，自己適応システムに限らず従来のソフトウェアに対しても期待されている[24]．これは，定理証明による検証はコストがかかることと，定式的証明は必ずしも操作的な正しさを保証するものではないことによるものである．VDM++の仕様テスト機能を利用した開発プロセスの提案としては，鵜林らのコンテキストを考慮したプロダクトライン開発[25] などをあげることができる．鵜林らは環境をコンテキストとしてモデリングし，仕様として記述することで，コンテキストの変化にも対応できるシステム開発を実現している．提案手法でも環境オブジェクトを仕様として記述するが，検出の対象はシステム内の競合であり，システム側の構成要素を並行動作させ，その競合を仕様テストにより検証するという点で異なる．

仕様テストを利用する以上，提案手法は自己適応システム内に潜む競合すべての検出を保証するものではない．このような厳密な検出には，SPIN [26] などを用いたモデル検査が有効である．自己適応システムへのモデル検査法の適用に関しては，いくつかの研究がある．Zhang ら[27] は，適応に関与する振舞いとそれ以外の振舞いとに分離した状態遷移モデルによるモデル検査法とモデル駆動開発法を提案している．また，Schaefer ら[28] はスライシングによる状態数の抑制手段を検討している．モデル検査においては，検査を適用する箇所の

特定と，設計モデルや実装コードとの整合性維持が難しさとなっているが，本研究においては，仕様テストで検証できない箇所に対してモデル検査を実施するといった 2 段階適用の可能性がある．

現時点では，自己適応システムにおける競合を扱った研究は少ない．Carzaniga ら[29] は，競合のみを扱うものではないが，コンポーネントの振舞いを形式的な仕様として記述し，システムの動作履歴と障害検知シグナルを利用することで障害の回避手段を自動的に決定する手法を提案している．ただし，システムが障害を検出できることや障害発生後に復旧機能が動作することなどを前提としているため，実現に向けてはまだまだ解決すべき問題が多い．また，障害を扱うことから，障害発生後のプロセスに着目しているという点において，競合発生の可能性をシステム構築中に排除しようとする本研究のスタンスとは異なる．

最後に，筆者らはすでにゴール指向要求分析法を用いた自己適応システムのアーキテクチャ決定法と，決定したアーキテクチャに対する実装ガイドラインを提案している[30]．この手法においても，自己適応システムを構築するコンポーネントをゴール指向記述中の各ゴールに対応付けているため，本研究で構築する仕様をシステム構築に利用することができる．実装ガイドラインをもとに，形式仕様から実装コードへの変換ルールを定義することにより，効果的な統合が実現可能であると考えている．

6. ま と め

本研究では，自己適応システム内に潜在するプロセス間競合の検出とその対処を目的として，要求モデルと形式仕様を用いた競合の検出法と，競合を検証するための仕様構築法を提案した．また，提案する開発プロセスを支援するために，競合検出ツールと仕様生成ツールを提案し，これらの有効性を仕様構築実験を通じて評価した．本手法により，自己適応システムの仕様構築と，実装の前段階におけるプロセス間競合の検出・解消が期待できる．

今後はまず，より確実な競合検出を実現するために，仕様テストケースの構築支援法について検討を進める予定である．その後，仕様から実装コードへのモデル変換手法を検討することで，文献 30) の先行研究と統合した自己適応システム開発法に発展させたい．

参 考 文 献

1) Oreizy, P., Gorlick, M.M., Taylor, R.N., Heimbigner, D., Johnson, G., Medvidovic, N., Quilici, A., Rosenblum, D.S. and Wolf, A.L.: An architecture-based approach to self-adaptive software, *IEEE Intelligent Systems*, Vol.14, No.3, pp.54–62 (1999).
2) Kramer, J. and Magee, J.: Self-managed systems: An architectural challenge, *Future of Software Engineering* (*FOSE '07*), pp.259–268 (2007).
3) Martin-Flatin, J.-P., Sventek, J. and Geihs, K.: Self-managed systems and services, *Comm. ACM*, Vol.49, No.3, pp.37–39 (2006).
4) Cheng, B.H.C., de Lemos, R., Giese, H., Inverardi, P., Magee, J., et al.: Software engineering for self-adaptive systems: A research road map, *Dagstuhl Seminar Proceedings 08031* (2008).
5) van Lamsweerde, A.: Goal-oriented requirements engineering: A guided tour, *5th IEEE International Symposium on Requirements Engineering* (*RE'01*), Toronto, Canada, pp.249–262 (2001).
6) 中島　震：ソフトウェア工学の道具としての形式手法，Technical report, NII テクニカル・レポート (2007).
7) Dardenne, A., van Lamsweerde, A. and Fickas, S.: Goal-directed requirements acquisition, *Science of Computer Programming*, Vol.20, No.1-2, pp.3–50 (1993).
8) Letier, E.: Reasoning about Agents in Goal-Oriented Requirements Engineering, Ph.D. thesis, Universite Catholique de Louvain (2001).
9) CSK. VDM tools – The VDM++ Language Manual. http://www.vdmtools.jp/uploads/manuals/langmanpp_a4E.pdf
10) Fitzgerald, J., Larsen, P.G., Mukherjee, P., Plat, N. and Verhoef, M.: *Validated Designs for Object-oriented Systems*, Springer (2005).
11) Morandini, M., Penserini, L. and Perini, A.: Towards goal-oriented development of self-adaptive systems, *Proc. International Workshop on Software Engineering for Adaptive and Self-managing Systems* (*SEAMS2008*), Leipzig, Germany, pp.9–16 (2008).
12) Darimont, R. and van Lamsweerde, A.: Formal refinement patterns for goal-driven requirements elaboration, *Proc. 4th ACM SIGSOFT Symposium on Foundations of Software Engineering*, pp.179–190 (1996).
13) CEDITI. Objectiver. http://www.objectiver.com/
14) Jones, C.B.: *Systematic Software Development Using VDM, 2nd Edition*, Prentice Hall (1990).
15) Fitzgerald, J. and Larsen, P.G.: *Modelling Systems, Practical Tools and Techniques in Software Development*, Cambridge University Press (1998).
16) 中川博之，田口研治，本位田真一：モデル変換に基づく要求記述を利用した形式仕様の構築，情報処理学会論文誌，Vol.49, No.7, pp.2304–2318 (2008).
17) Nakagawa, H., Taguchi, K. and Honiden, S.: Formal specification generator for KAOS, *Proc. 22nd IEEE/ACM International Conference on Automated Software Engineering* (*ASE '07*), Atlanta, Georgia, USA, pp.531–532, ACM (2007).
18) CSK. VDMTools. http://www.vdmtools.jp/

19) Yu, E.S.K.: Towards modelling and reasoning support for early-phase requirements engineering, *Proc. 3rd IEEE International Symposium on Requirements Engineering* (*RE'97*), pp.226–235 (1997).
20) Abrial, J.-R.: *The B-Book: Assigning Programs to Meanings*, Cambridge University Press (1996).
21) Spivey, J.M.: The Z notation: A reference manual. http://spivey.oriel.ox.ac.uk/mike/zrm/index.html
22) 深谷直彦，古川善吾，西　康晴（編）：ソフトウェアテストの最新動向，Vol.49, pp.125–173, 情報処理学会 (2008).
23) Maury, O., Ledru, Y., Bontron, P. and du Bousquet, L.: Using TOBIAS for the automatic generation of VDM test cases, *VDM Workshop*, Copenhagen, Denmark (2002).
24) Bowen, J.P., Clark, J.A. and Hierons, R.M.: Fortest: Formal methods and testing, *Proc. 26th Annual International Computer Software and Applications Conference* (*COMPSAC '02*), pp.91–101 (2002).
25) 鵜林尚靖，金川太俊，瀬戸敏喜，中島　震，平山雅之：コンテキストベース・プロダクトライン開発と VDM++の適用，情報処理学会論文誌，Vol.48, No.8, pp.2492–2507 (2007).
26) Holzmann, G.J.: The model checker SPIN, *IEEE Trans. Software Engineering*, Vol.23, No.5, pp.279–295 (1997).
27) Zhang, J. and Cheng, B.H.C.: Model-based development of dynamically adaptive software, *Proc. 28th International Conference on Software Engineering* (*ICSE '06*), New York, NY, USA, pp.371–380, ACM (2006).
28) Schaefer, I. and Poetzsch-Heffter, A.: Slicing for model reduction in adaptive embedded systems development, *Proc. 2008 International Workshop on Software Engineering for Adaptive and Self-managing Systems* (*SEAMS '08*), Leipzig, Germany, pp.25–32 (2008).
29) Carzaniga, A., Gorla, A. and Pezzè, M.: Self-healing by means of automatic workarounds, *Proc. 2008 International Workshop on Software Engineering for Adaptive and Self-managing Systems* (*SEAMS '08*), New York, NY, USA, pp.17–24, ACM (2008).
30) 中川博之，大須賀昭彦，本位田真一：ゴール指向要求分析を用いた self-adaptive システムの構築，情報処理学会論文誌，Vol.50, No.10, pp.2500–2513 (2009).

(平成 22 年 1 月 7 日受付)
(平成 22 年 6 月 3 日採録)

中川　博之（正会員）

1974 年生．1997 年大阪大学基礎工学部情報工学科卒業．同年鹿島建設（株）に入社．2007 年東京大学大学院情報理工学系研究科修士課程修了，2008 年同大学院博士課程中退．同年より電気通信大学助教，現在に至る．要求分析，形式手法，エージェントおよび自己適応システム開発手法の研究に従事．電子情報通信学会，IEEE CS 各会員．

大須賀昭彦（正会員）

1981 年上智大学理工学部数学科卒業．同年（株）東芝入社．同社研究開発センター，ソフトウェア技術センターなどに所属．1985〜1989 年（財）新世代コンピュータ技術開発機構（ICOT）出向．2007 年より電気通信大学大学院情報システム学研究科教授．工学博士（早稲田大学）．主としてソフトウェアのためのフォーマルメソッド，エージェント技術の研究に従事．1986 年度情報処理学会論文賞受賞．電子情報通信学会，日本ソフトウェア科学会，人工知能学会，IEEE CS 各会員．

本位田真一（フェロー）

1953 年生．1978 年早稲田大学大学院理工学研究科修士課程修了．(株) 東芝を経て 2000 年より国立情報学研究所教授，2004 年より同研究所アーキテクチャ科学研究系研究主幹を併任，現在に至る．2008 年より同研究所先端ソフトウェア工学・国際研究センター長を併任，現在に至る．2001 年より東京大学大学院情報理工学系研究科教授を兼任，現在に至る．現在，早稲田大学客員教授，英国 UCL 客員教授を兼任．2005 年度パリ第 6 大学招聘教授．工学博士（早稲田大学）．1986 年度情報処理学会論文賞受賞．日本ソフトウェア科学会理事，情報処理学会理事を歴任．ACM 日本支部会計幹事，情報処理学会フェロー，日本ソフトウェア科学会編集委員長，日本学術会議連携会員．