Vol.2016-SE-191 No.11
2016/3/14

KAOSからのモデル変換に基づく形式仕様構築の一般化

岩本 侑也[1]　大森 洋一[1]　林 信宏[1]　日下部 茂[1]　荒木 啓二郎[1]

概要：仕様の曖昧さによる不具合は下流工程で発見されるほど修正コストが大きくなる. そのため, ソフトウェア開発の上流工程で厳密な仕様を記述することが重要である. 厳密な仕様を記述する手段として形式手法は有効であるが, 自然言語による要求記述からの直接の仕様記述は容易ではない. 形式仕様の構築を容易にするための要求分析作業を予備形式化 (プレフォーマル) と呼ぶ. 既にゴール指向要求分析法 KAOS により記述された要求記述から形式仕様記述言語 VDM++により記述された仕様を構築する手法が提案されている. 本研究では, この手法をより一般的な場合に適用できるよう拡張する. 本研究により, 複数のエージェントを含む場合にも, 要求と形式仕様の整合性を保証した, ソフトウェア開発を実現する.

1. はじめに

1.1 背景

ソフトウェア開発では, 日本語などの自然言語で作成する仕様書の記述が曖昧なことが原因でプログラム不具合などが起こり, 開発コストが増加することが問題視されている [1]. 仕様の曖昧さによる不具合は下流工程で発見されるほど修正コストが大きくなる. そのため, ソフトウェア開発の上流工程で厳密な仕様を記述し, 曖昧さやバグを取り除くことが重要である. 厳密な仕様を記述する手段の一つとして形式手法がある. 形式手法は形式仕様記述言語を使って仕様を厳密に書き, 数理的性質を検証するソフトウェア開発手法である. 形式仕様記述言語で仕様を記述することは日本語などの自然言語による仕様の曖昧な点を排除するのに有効である [2]. しかし, 自然言語による要求記述から直接, 形式仕様記述言語による仕様記述を作成することは容易ではない [3].

1.2 研究の目的

本研究では、形式仕様記述を容易にするための要求分析工程である予備形式化 (プレフォーマル) の一例として, ゴール指向要求分析を利用する手法を検討する. 具体的には, 先行研究 [3] で提案されている「ゴール指向分析手法 KAOS で表現された要求記述から, VDM++による形式仕様記述を生成する手法」をより一般的な場合に適用できるように拡張する. すなわち, 複数のエージェントから構成されるシステムの KAOS モデルを VDM++の形式仕様記述へと変換するアルゴリズムと, 変換を実現するための KAOS の記述制約を提案する. 本提案手法では, 開発者は, まずシステムの要求仕様書やユースケース記述を KAOS によって分析し, KAOS モデルを構築する. 次に, 提案する変換アルゴリズムによって VDM++による形式仕様記述を作成する. 最後に, 操作に対して具体的な処理を追記する.

1.3 本論文の構成

第 2 章では, 形式手法とゴール指向分析について説明する. また, それぞれを記述するツールを紹介する. 第 3 章では, ゴール指向要求分析手法 KAOS により記述された要求記述から VDM++による形式仕様記述を生成する手法について述べる. 第 4 章では, 第 3 章で述べた手法を自動販売機システムに適用し, 有用性を検証した結果について述べる. 第 5 章で結果に対する考察と今後の課題を述べる.

2. 用語の説明

2.1 形式手法

形式手法とは, 明確で厳密な意味論を持つ数理論理学に基づく言語を用いて設計対象を表現することにより, 設計記述の正しさを系統的に示す技術である. 形式手法は, 自然言語を用いて記述していた仕様を, 形式仕様記述言語を用いて背景となる数理に則った厳密な記述をすることによって, システム開発の上流工程からソフトウェアの品質を高めることを目的としている.

2.1.1 VDM

VDM(Vienna Development Method) は形式手法の 1 つであり, 1960 年代に IBM ウィーン研究所で開発された [4].VDM には VDM-SL,VDM++,VDM-RT という 3 つ

[1] 九州大学
Kyushu University

ⓒ 2016 Information Processing Society of Japan

の形式仕様記述言語がある.VDM-SL は ISO 標準になっている.VDM++は VDM-SL を基礎にオブジェクト指向拡張と非同期並行処理に関する拡張を行った言語である.VDM-RT は VDM++にさらに実時間制御と同期並行処理を拡張した言語である.VDM は形式手法の中でもモデル規範型と呼ばれ, モデルの状態を記述していくことに重点を置いている.VDM を適用した開発の事例も多くあり, その有用性が実証されている [5]. 本研究では形式仕様記述言語として VDM++を利用する.

2.1.2 事前条件・事後条件・不変条件

VDM++は関数と状態変更を伴う機能を記述する操作に事前条件と事後条件と呼ばれる制約条件を記述することができる. 事前条件とは関数と操作の実行直前の状態が満足すべき条件, 事後条件とは関数と操作の実行直後の状態が満足すべき条件である. 入力が事前条件に従っている限り, 関数や操作は事後条件を保証しなければならず, 期待する正しさの基準となる [7]. また, VDM++は型と状態変数に不変条件と呼ばれる制約条件を記述することができる. 不変条件とは, 型や状態変数が常に満たさなければならない条件である. 不変条件は型や状態変数を特徴付け, インスタンスの生存中にずっと保持される特性を表す [7].

2.1.3 陰仕様と陽仕様

VDM++では関数と操作を陰定義と陽定義の二通りで記述できる. 陰定義は関数と操作のシグニチャ, 事前条件, 事後条件のみ記述し, アルゴリズムは記述しない. 一方, 陽定義はシグニチャ, 事前条件, 事後条件に加えて関数や操作が実行するアルゴリズムまで記述する. 操作や関数が陰定義で記述された仕様を陰仕様, 陽定義で記述された仕様を陽仕様という. 陰仕様は, 構文検証や型検証などの静的な検証を行うことができ, 陽仕様はそれらに加えて, ツールを用いて実行させて検証を行える.

2.1.4 VDM の検証ツール

VDMTools と OvertureTool はともに VDM 検証支援ツールである.VDMTools はデンマークの IFAD 社が開発した.2005 年に SCSK（旧 CSK システムズ）がデンマークの開発元から全ての権利を取得し, 全世界に向けて提供している [4].OvertureTool は統合開発環境である Eclipse のプラグインとして提供されているオープンソースのモデル支援ツールであり, Peter Gorm Larsen 博士を中心としたコミュニティにより開発が進められている.[8]. 本研究では, KAOS から生成した VDM++モデルをこれらのツールを用いて検証した.

2.2 ゴール指向要求分析手法

ゴール指向要求分析とはシステム開発の目標であるゴールを構造的に明確化することでシステム要求を定義する要求分析手法である [9]. ゴールを階層的に展開することによって, システム要求を論理的に理解しやすい形で構造化する. 階層化されたゴールグラフの最下位のゴールがシステム要求になる.

2.2.1 KAOS

KAOS(Knowledge Acquisition in autOmated Specifications) は Lamsweerde らによって開発されたゴール指向要求分析手法である [3].KAOS はゴールモデル, 責任モデル, オブジェクトモデル, 操作モデルからなる [10].

- ゴールモデル
  システムが達成すべきゴールをツリー構造で表現する. ゴールを達成するために何をするかをサブゴールとして分解する. ゴールを分解し, 要件に責任を持つエージェントを特定したところで分解を止める.
- 責任モデル
  どのエージェントがどの要件に対して責任を持つかを表現するモデルである. エージェント毎に作られ, ゴールモデルから抽出したエージェントが責任を持つ要件を記述する.
- オブジェクトモデル
  ゴールを達成するための操作で扱うエンティティの関係を表現するモデルである. エンティティ同士の多重度などを記述する.
- 操作モデル
  ゴールを達成するために必要なエージェント毎の操作を表現する. 操作と操作の入力, 出力となるエンティティを記述する.

2.2.2 KAOS 記述ツール

Objectiver は Respect-IT 社が提供する KAOS のモデリングツールである [11].KAOS モデルの構築に加え, KAOS 内の各モデル要素に対しての詳細情報をプロパティとして記述することができる. 本研究では, Objectiver を用いて KAOS モデルの構築を行った.

3. 提案手法

本章では, ゴール指向要求分析手法 KAOS により記述された要求記述から VDM++による形式仕様記述を生成する手法について述べる. まず, 先行研究とその問題点について述べる. 次に, 先行研究で提案されている手法の拡張について述べる.

3.1 先行研究とその問題点

文献 [3] では, KAOS モデルを VDM++による形式仕様記述に変換する手法と, 変換を実現するための KAOS の記述規則が提案されている. ただし, 文献 [3] の事例はエージェントが単一の比較的単純な事例であり, 実際の組み込みシステム開発でしばしば見られる, システム内部が複数のシステムからなるような, いわゆるシステム・オン・システムズの考察がない. すなわち, エージェントの記述の粒度がシステムと外部環境といった程度であり, 複数のエー

ⓒ 2016 Information Processing Society of Japan

ジェントから構成されるシステムを VDM++で記述する場合とはギャップがある. よって, 文献 [3] の KAOS の記述規則と変換手法では複数のエージェントから構成されるシステムを KAOS で記述し, VDM++による仕様記述に変換するのは難しい.

3.2 KAOS の制約

3.1 で述べた難しさを解決するため, KAOS の各モデルに対して以下のような制約を加える. この制約は文献 [12] で提案されている, マルチエージェントシステムを KAOS で記述することを目的とした KAOS の拡張と文献 [3] の KAOS 記述規則を組み合わせたものである.

- ゴールモデルの制約
  要件はシステムを構成する各エージェントが単独で請け負う責務のレベルまで分解する. すべての要件に対してそれぞれ 1 つのエージェントを割り当てる. その時点でゴールを要件まで分解できない場合は, ゴールに対してエージェントから, エージェントがゴールに責任を持つことを表す“Assignment”関係を定義する.
- 責任モデルの制約
  エージェントが責務を負うすべての要件について, 対応するオペレーションを定義する. オペレーションはエージェントが単独で実行可能なレベルで記述する. エージェントから操作へ, エージェントが操作を行うことを表す“Performance”関係を定義する. エージェントの内部変数になるエンティティに対しては, エージェントがエンティティを管理することを表す“Control”関係を定義する.
- オブジェクトモデルの制約
  抽出されたエージェント以外の概念はすべてエンティティとして記述する. エンティティの型が決まっていれば, 型をプロパティに記述する. エンティティ同士の関連は, あるエンティティが別のエンティティの一種であることを表す“is-a 関係”を定義するか, 多重度を表す“Binary Association”関係を定義する. エンティティとエージェントの関係は, 責任モデルで定義した“Control” 関係と“Binary Association ”関係で記述する. また, エンティティに対する不変条件をプロパティとして VDM++の記法にしたがって記述する.
- オペレーションモデルの制約
  事前条件と事後条件があれば, プロパティとして VDM++の記法にしたがって記述する.

3.3 変換手法

KAOS モデルを VDM++による形式仕様記述へと変換する手法について述べる. まず, エンティティをオブジェクト, 型, 集合型, システムの内部変数, 変数名のいずれの概念を表現しているかを判別し, その後, 共通定義のクラスと, 各オブジェクト, 各エージェントのクラスファイルを作成する. 変換アルゴリズムは付録に記載した. この変換アルゴリズムは, 文献 [3] をもとにして, エージェントが複数ある場合にも対応できるように拡張を行ったものである. 変更点は,

- エンティティやエージェントの“is-a” 関係のルールを加えた点
- エージェントクラスの内部変数に別のエージェントをとる場合に対応させた点
- エージェントクラスにコンストラクタを記述させる点

などである.

3.3.1 エンティティの判別

エンティティの判別法は文献 [3] の手法と同様である. エンティティに他のエンティティに対する“is-a” 関係が定義されていれば, そのエンティティを変数名とする. エンティティが単一のエンティティに対してのみ“Binary Association”関係が定義され, かつ多重度が*なら, エンティティを集合型とする. エンティティが他のエンティティを参照していないならば, エンティティを型とする. 以上の判別の経て残ったエンティティをオブジェクトとする.

3.3.2 クラスファイルの生成

VDM++による形式仕様記述として, 共通定義, オブジェクト定義, エージェント定義のための 3 種類のクラスファイルを生成する.

共通定義クラスは, システム全体で共有される型を定義するクラスである. すべての型と判定されたエンティティについて, プロパティに型が記述されていれば, エンティティの型として定義する. プロパティに不変条件が記述されていれば, エンティティの不変条件として記述する.

オブジェクト定義クラスは, オブジェクトと判定された各エンティティに対して生成されるクラスである. “Binary Association”関係が定義されているエンティティをインスタンス変数とする. インスタンス変数となったエンティティのプロパティに不変条件が記述されていれば, 不変条件を記述する. 操作として, オブジェクトの構成子であるコンストラクタと, インスタンス変数の値を取得する操作であるゲッタ, インスタンス変数の値を書き込む操作であるセッタを定義する.

エージェント定義クラスは, 各エージェントに対して定義されるクラスである. インスタンス変数として, “Control”関係を定義したエンティティとエージェントを記述する. このとき, 変数名と判定されたエンティティは, 変数名をエンティティの名前, 型を“is-a” 関係の参照先とする. エージェントの操作として, コンストラクタと“Performance”関係が定義された操作を記述する. 各操作について, 入出力エンティティから, 入出力インターフェースを記述する. 操作の処理内容は未定義を表す VDM++のキーワード is not yet specified と記述する. 操作のプロパティに事前条件, 事

ⓒ 2016 Information Processing Society of Japan

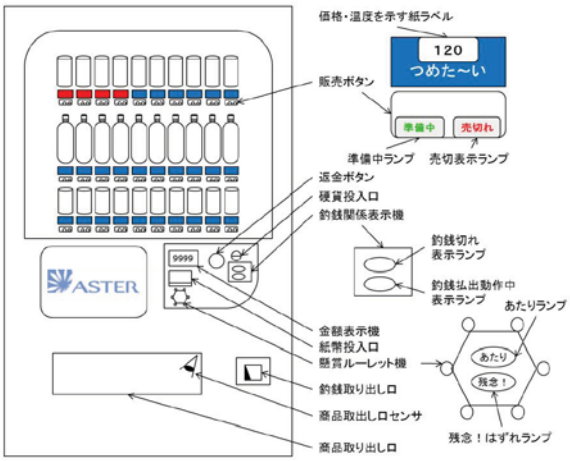


図 1 自動販売機ハードウェア構成, [ASTER, 自動販売機ハードウェア構成および販売者用機能仕様]

Fig. 1 Hardware of the vending machine

後条件が記述されていれば, それぞれ VDM++の操作の事前条件と事後条件として記述する.

4. 本手法の適用事例

第 3 章で提案した開発手法を, 自動販売機システムに適用して提案手法の有用性を検証した. 現在の自動販売機は複数の独立性の高いハードウェア部品がそれぞれ CPU を持ち, それらを組み合わせて実装される. したがって, 本質的に多くのエージェントが登場する仕様が対象となる.

4.1 記述対象の説明

本研究では「自動販売機ハードウェア構成および販売者用機能仕様」[13](以降「自動販売機機能仕様書」) と「ASTER 自動販売機ユースケース仕様書」[14](以降「自動販売機ユースケース記述」) をモデル化の対象とする. これは ASTER（特定非営利活動法人ソフトウェアテスト技術振興協会：Association of Software Test EngineerRing) が提供する自動販売機を題材にしたテスト設計コンテストの課題である [15].

4.1.1 自動販売機仕様書

自動販売機仕様書には, 自動販売機のハードウェア構成と各ハードウェアが行う処理が記述されている. ハードウェア構成を図 1 に示す.

4.1.2 自動販売機ユースケース記述

「自動販売機ユースケース記述」には次の 5 つのユースケースが記述されている.

- 代金投入ユースケース
- 商品選択ユースケース
- 返金ユースケース
- 懸賞ユースケース
- 故障ユースケース

それぞれのユースケースは「ゴール」, 「背景や理由」, 「事前条件」, 「事後条件」, 「処理 (メインフロー)」から構成されている.「ゴール」はそのユースケースで達成すべきこと, 「背景や理由」は、ゴールの内容についての背景やユースケースのゴールとして設定した理由など, 「事前条件」には, そのユースケースを行う前の状態, 「事後条件」にはユースケースを行った後の状態, 「処理 (メインフロー)」はユースケースを行うときの具体的なシステムの処理が書かれている. ただし, これらの「事前条件」,「事後条件」は形式的な記述ではない.

4.2 形式仕様記述の作成

自動販売機システムの商品選択ユースケースに対する KAOS モデルの構築, クラスファイルの生成, 陽仕様化を行った. 他のユースケースに対しても同様のプロセス適用することでシステム全体を記述することができる.

4.2.1 自動販売機の KAOS モデル構築

- ゴールモデル
  図 2 は商品選択ユースケースに対するゴールモデルである. 商品選択ユースケースが満たされることをトップゴールとした. ユースケースがユースケース記述の「ゴール」の欄から上位ゴールを構成した. 次に, 「処理」欄に書かれた自動販売機の機能がそれぞれ, どの上位ゴールを達成するためのものかを分析し, ゴールの分解を行った. そして, 機能ゴールに対して, それぞれ単一のエージェントが実行できる機能まで分解を行い, 要件を抽出した. 最後に, それぞれの要件をどのエージェントが担うかを「自動販売機仕様書」から抽出し, 責務関係を記述した. 例えば, 「購入者が商品を選択する」というトップゴールは, ユースケースの「ゴール」の欄から「当該商品の販売ボタンを点滅させる」などのサブゴールに分解される.「当該商品の販売ボタンを点滅させる」というゴールは, ユースケース記述の「処理」の欄に「当該商品の販売ボタンを点滅させる」と書かれているので, 「処理」欄の情報からは分解できないが,「自動販売機仕様書」から「販売可能ランプを点滅させる」と「当該商品の販売可能ランプの点滅を依頼する」という 2 つの要件に分解できる. これらは, 単一のエージェントで実行できる操作なので, 分解をやめ, 責任関係を記述した.
- 責任モデル
  ゴールモデルの責務関係から責任モデルを作成した. 図 3 は販売ボタンの責任モデルである. すべての要件に対して操作を定義した. 例えば, 「販売可能ランプを消灯する」という要件に対して「販売可能ランプを消灯」という操作が定義されている. また, 内部変数になるエンティティとして「商品」と「販売可能ランプ」を定義した.
- オブジェクトモデル

ⓒ 2016 Information Processing Society of Japan

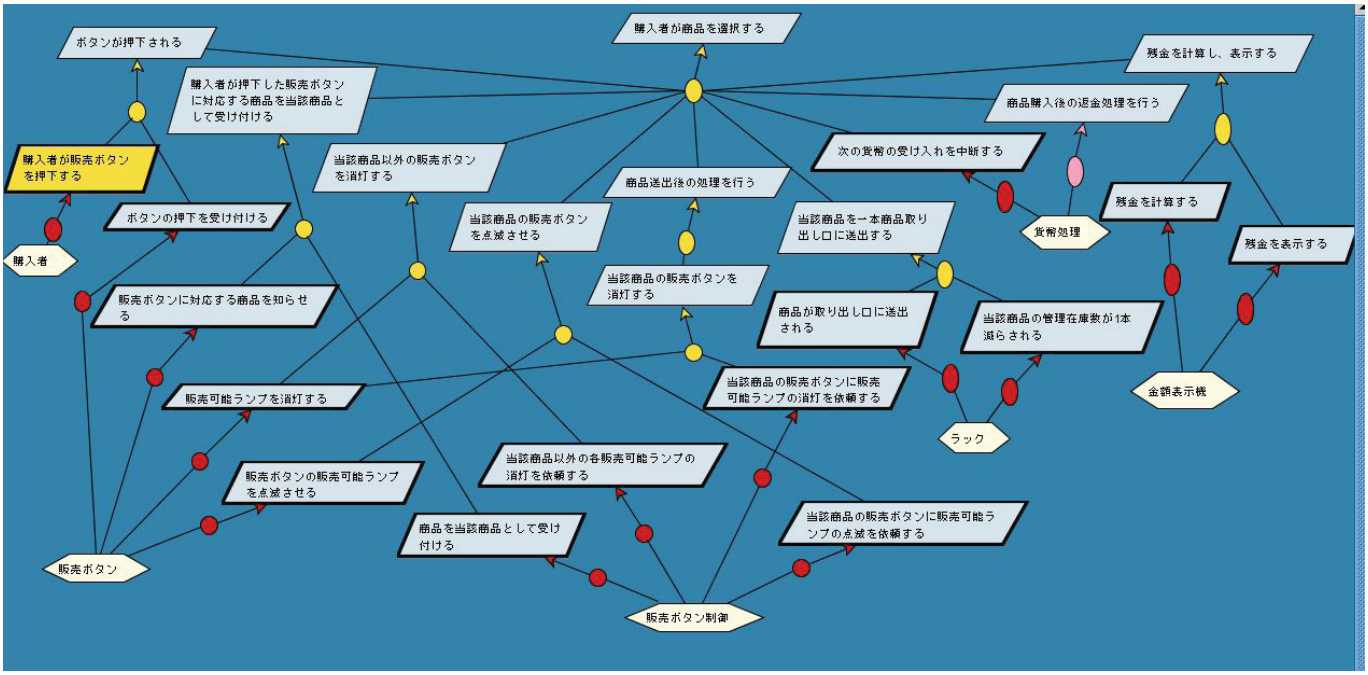


図 2　商品選択ゴールモデル
Fig. 2　Goal model for product selection

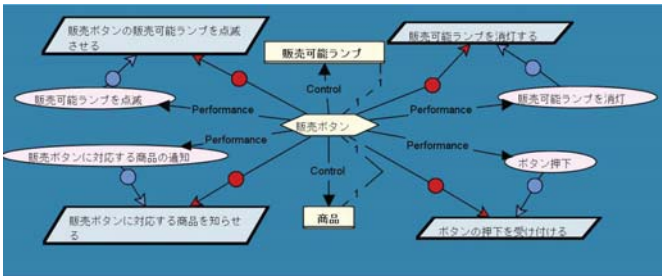


図 3　販売ボタン責任モデル
Fig. 3　Responsibility model for sell button

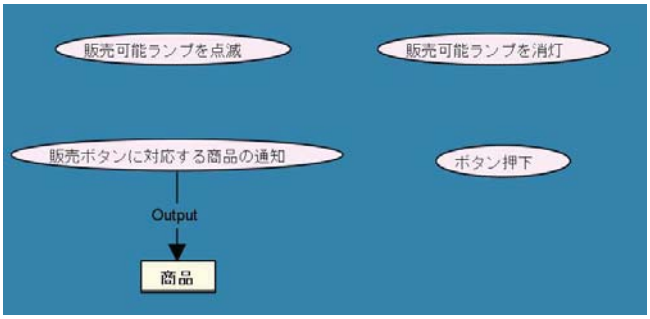


図 5　販売ボタン操作モデル
Fig. 5　Operation model for sell button

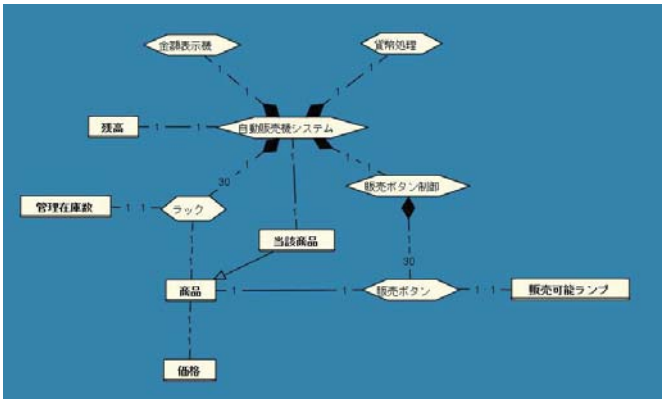


図 4　商品選択オブジェクトモデル
Fig. 4　Object model for product selection

商品選択ユースケースのゴールモデルから抽出されたエンティティとエージェント関係を記述した. 図??は作成したオブジェクトモデルである.

- 操作モデル

図 5 は販売ボタンの操作モデルである.「販売可能ランプを点滅」という操作は, 入出力エンティティを持たない.「販売ボタンに対応する商品の通知」という操作は入力を持たず, 出力としてエンティティ「商品」を返す.

4.2.2　生成されたクラスファイル

3.2 の変換手法により, 4.2.1 で構築した自動販売機の KAOS モデルから VDM++のクラスファイルを生成した. 共通定義クラス, オブジェクト定義クラスとして「商品」クラス, エージェント定義クラスとして「自動販売機システム」クラス,「販売ボタン」クラス,「販売ボタン制御」クラス,「ラック」クラス,「金額表示機」クラス,「貨幣処理」クラスの合計 8 クラスが生成された. 図 6 は生成された「販売ボタン」クラスである. インスタンス変数として「販売可能ランプ」と「商品」, 操作としてコンストラクタと「販売可能ランプを消灯」,「販売可能ランプを点滅」,

ⓒ 2016 Information Processing Society of Japan

```
class「販売ボタン」 is subclass of 共通定義

instance variables
  private 販売可能ランプ:「販売可能ランプ」;
  private 商品:「商品」;

operations
  /*コンストラクタ*/
  public 「販売ボタン」:「販売可能ランプ」*「商品」==>「販売ボタン」
  「販売ボタン」(a販売可能ランプ, a商品)==(
    atomic(
      販売可能ランプ:=a販売可能ランプ;
      商品:=a商品;
      );
    return self
    );

  public 販売可能ランプを消灯: ()==>()
    販売可能ランプを消灯()==  is not yet specified
    --TODO: 本操作の処理内容を記述してください
  pre 販売可能ランプ<> <on>
  post 販売可能ランプ=<off>;

  public 販売可能ランプを点滅: ()==>()
    販売可能ランプを点滅()==  is not yet specified
    --TODO: 本操作の処理内容を記述してください
  pre 販売可能ランプ<> <blink>
  post 販売可能ランプ=<blink>;

  public 販売ボタンに対応する商品の通知:()==>「商品」
  販売ボタンに対応する商品の通知()== is not yet specified
    --TODO: 本操作の処理内容を記述してください
    --pre   TODO: 事前条件があれば記述してください
    --post  TODO: 事前条件があれば記述してください
;

  public ボタン押下:()==>()
  ボタン押下()==  is not yet specified
  --TODO: 本操作の処理内容を記述してください
    --TODO: 本操作の処理内容を記述してください
    --pre   TODO: 事前条件があれば記述してください
    --post  TODO: 事前条件があれば記述してください;

end「販売ボタン」
```

図 6 生成された販売ボタンのクラスファイル
Fig. 6 Generated class file for sell button

「販売ボタンに対応する商品の通知」,「ボタン押下」が定義された.

4.2.3 クラスファイルへの追加・修正

生成されたクラスファイルの修正や, 必要なインスタンス変数, 操作などの追加を行った. 図 7 は修正した販売ボタンのクラスファイルである. 販売ボタンには「販売ボタン状態」という状態を表すインスタンス変数を追加し, それに伴うコンストラクタの修正を行い, 状態のゲッタを追加した. これは, KAOS で状態遷移を記述できないことに起因する. また, 販売ボタン制御は, KAOS のオブジェクトモデルの多重度を手動で反映させたため, カバー率が低かった. 修正した行数, 項目を表 1 に示す. また, それぞれのクラスファイルへの修正内容を表に示す.

生成されたクラスファイルが, 陰仕様作成のためにどの程度利用されているかを評価するため, カバー率を計測した. カバー率は次の式で定義される.

```
class「販売ボタン」 is subclass of 共通定義

instance variables
  private 販売可能ランプ:「販売可能ランプ」;
  private 商品:「商品」;
  private 販売ボタン状態:<通常>|<押下>;

operations
  /*コンストラクタ*/
  public 「販売ボタン」:「販売可能ランプ」*「商品」==>「販売ボタン」
  「販売ボタン」(a販売可能ランプ, a商品)==(
    atomic(
      販売可能ランプ:=a販売可能ランプ;
      商品:=a商品;
      販売ボタン状態:=<通常>;
      );
    return self
    );

  /*操作*/
  public 販売可能ランプを消灯: ()==>()
    販売可能ランプを消灯()== is not yet specified
  pre 販売可能ランプ<> <on>
  post 販売可能ランプ=<off>;

  public 販売可能ランプを点滅: ()==>()
    販売可能ランプを点滅()==  is not yet specified
  pre 販売可能ランプ<> <blink>
  post 販売可能ランプ=<blink>;

  public 販売ボタンに対応する商品の通知:()==>「商品」
  販売ボタンに対応する商品の通知()==  is not yet specified;

  public ボタン押下:()==>()
  ボタン押下()==  is not yet specified;

  public get状態:()==><通常>|<押下>
  get状態()==return 販売ボタン状態;

end「販売ボタン」
```

図 7 手動修正した販売ボタンのクラスファイル
Fig. 7 Corrected class file for sell button

表 1 形式仕様の追加・変更行数・カバー率
Table 1 Result of acquiring formal specification

生成されたクラス	生成行数	追加変更行数	カバー率
共通定義	10	0	1
ラック	21	4	0.81
販売ボタン	26	4	0.85
販売ボタン制御	15	4	0.73
金額表示機	12	0	1
貨幣処理	8	3	0.63
自動販売機システム	11	2	0.82
商品	12	0	1
合計	115	17	0.85

$$\text{カバー率} = \frac{\text{陰仕様記述の行数} - \text{追加変更行数}}{\text{陰仕様記述の行数}} \quad (1)$$

全体におけるカバー率は約 85 パーセントだった.

4.2.4 陽仕様化

変換手法によって生成されたクラスファイルは, 操作の内部処理が書かれていない陰仕様である. この時点で操作の処理内容が書ける操作には内部処理を書き, 部分的に陽仕様化を行った. 図 8 は陽仕様化した販売ボタンのクラス

ⓒ 2016 Information Processing Society of Japan

表 2　形式仕様の追加・変更項目
Table 2　Breakdown list of acquiring formal specification

生成されたクラス	追加変更項目
共通定義	
ラック	状態の追加, コンストラクタの修正, ゲッタの追加
販売ボタン	状態の追加, コンストラクタの修正, ゲッタの追加
販売ボタン制御	型の記述, コンストラクタの修正, 不変条件の追加
金額表示機	
貨幣処理	状態の追加, コンストラクタの修正, ゲッタの追加
自動販売機システム	型の記述, 不変条件の追加
商品	

```
class 「販売ボタン」 is subclass of 共通定義

instance variables
  private 販売可能ランプ:「販売可能ランプ」;
  private 商品:[「商品」];
  private 販売ボタン状態:<通常>|<押下>;

operations
  /*コンストラクタ*/
  public 「販売ボタン」:「販売可能ランプ」*「商品」==>「販売ボタン」
  「販売ボタン」(a販売可能ランプ, a商品)==(
    atomic(
      販売可能ランプ:=a販売可能ランプ;
      商品:=a商品;
      販売ボタン状態:=<通常>;
      );
    return self
    );

  /*操作*/
  public 販売可能ランプを消灯: ()==>()
    販売可能ランプを消灯()== 販売可能ランプ:=<off>
  pre 販売可能ランプ<> <on>
  post 販売可能ランプ=<off>;

  public 販売可能ランプを点滅: ()==>()
    販売可能ランプを点滅()== 販売可能ランプ:=<blink>
  pre 販売可能ランプ<> <blink>
  post 販売可能ランプ=<blink>;

  public 販売ボタンに対応する商品の通知:()==>「商品」
  販売ボタンに対応する商品の通知()== return 商品
;

  public ボタン押下:()==>()
  ボタン押下()== 販売ボタン状態:=<押下>
;
  public get状態:()==><通常>|<押下>
  get状態()==return 販売ボタン状態;

end 「販売ボタン」
```

図 8　陽仕様化した販売ボタンのクラスファイル

ファイルである.

4.2.5　ユースケースの追加

商品選択以外のユースケースについては, 商品選択ユースケースの場合と同様にしてゴールモデルを構築し, 抽出した要件をそれぞれ責任モデル, オブジェクトモデル, 操作モデルに反映させる. これにより, それぞれユースケースから抽出された要件に対応する要素が追加された KAOS モデルが構築される. 例えば, 代金投入ユースケースを図 9

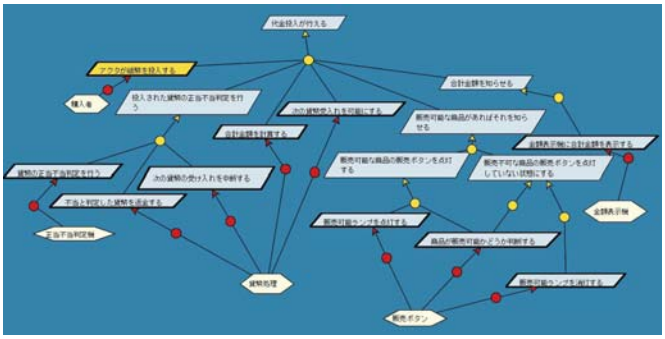


図 9　代金投入ゴールモデル
Fig. 9　Goal model for price input

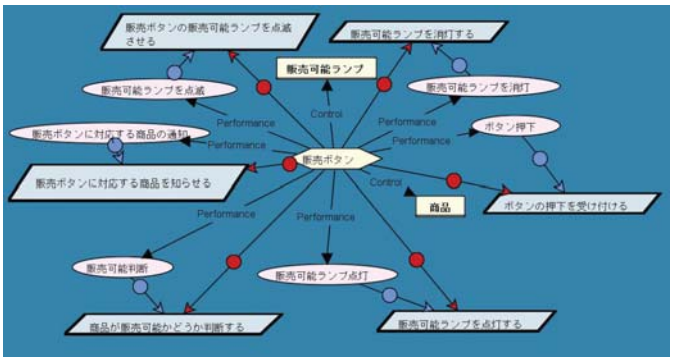


図 10　変更後の販売ボタン責任モデル
Fig. 10　Changed responsibility model for sell button

のように構築すると, 要件「販売可能ランプを点灯する」, 「商品が販売可能かどうか判断する」が販売ボタンの責務に追加される. これを責任モデルに反映させたものが図 10 である. 新たに加わった要件に対して, それぞれ操作が定義してある. 定義した操作を操作モデルに加え, 入出力を定義する. 要件から新たな概念が抽出されれば, オブジェクトモデルにエンティティとして記述し, 他のエージェントやエンティティとの関係を定義する. 構築された KAOS モデルを変換アルゴリズムで VDM++による記述に変換することにより, 変更が反映された VDM++のクラスファイルを生成することが可能である.

5.　おわりに

5.1　考察

第 4 章の事例から, 第 3 章の KAOS への制約と変換手法により, エージェントが複数ある場合でも, KAOS モデルから VDM++による形式仕様記述に変換できることが確認できた. また, ユースケース記述のゴールは, KAOS への制約により要件に分解され, VDM++による形式仕様記述へと変換されるため, ユースケースがどのように VDM++による記述に反映されたかが明確化した.

4.2.3 の追加変更項目の内容を集計した結果を表 3 に示す. 各エージェントの状態の追加, それにともなうコンストラクタの修正やゲッタの追加は開発者の判断で行わなけれ

ⓒ 2016 Information Processing Society of Japan

表 3　形式仕様の追加・変更行数
Table 3　The number of additional row

項目	行数
状態の追加	3
コンストラクタの修正	3
ゲッタの追加	6
変数名の変更	1
不変条件の追加	2
型の記述	2
合計	17

ばならなかった. これは, KAOS モデル上に状態の記述をすることができなかったことに起因する. また, 関連の多重度が 1 の場合か“*”(0 を含む任意の数) 以外の場合については, 変換アルゴリズムに記述されていないため, 開発者の判断で引数の個数や変数名の変更を行わなければならなかった. 文献 [3] の変換アルゴリズムに追加した“is-a”関係に関するルール, エージェントの属性に関するルールにより, KAOS モデルでの“is-a”関係, “エージェントが別のエージェントを管理する関係”も, VDM++の陰仕様に反映されていた.

5.2　今後の課題

本研究ではプレフォーマルの一例として, KAOS による要求分析を行い, 要求間の関連を明らかにすることで, VDM++形式仕様記述におけるモデルの要求整理を行う手法を提案した. ゴール指向分析手法である KAOS では, 各エージェントの状態を記述することが難しい. そのため, 状態遷移の分析が必要な場合, UML のステートマシン図との併用などを考える必要がある. そして, 変換アルゴリズムで多重度に対する KAOS と VDM++記述の対応について記述していないため, 多重度を考慮し, アルゴリズムを設計する必要がある. また, KAOS モデルは, 時相論理による検証を提案しているが, 妥当性を確認するためには時相論理式の記述が必要であり, 適切な検証条件の記述は容易でない. 対象が一階述語論理の範囲であれば VDM++を用いた検証はより容易である. 例えば, 実用的な観点からは変換後の VDM++仕様記述のテストなどによって検証するために, KAOS モデルから VDM++による記述のテストケースを作成する手法が有効であると考えられる.

参考文献

[1] 佐原伸: ソフトウェアトラブルを予防する 形式手法の技術講座, 株式会社ソフト・リサーチ・センター, (2008).
[2] IPA/SEC: 厳密な仕様記述入門, https://www.ipa.go.jp/files/000026829.pdf, (最終確認日 2016 年 2 月 6 日).
[3] 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).
[4] VDM information web site, http://www.vdmtools.jp/, (最終確認日 2016 年 2 月 6 日).
[5] IPA/SEC: 厳密な仕様記述における形式手法成功事例, http://www.ipa.go.jp/files/000026875.pdf, (最終確認日 2016 年 2 月 6 日).
[6] CSK SYSTEMS CORPORATION: VDM++言語マニュアル, http://www.vdmtools.jp/uploads/manuals/langmanpp_a4J.pdf, (最終確認日 2016 年 2 月 6 日).
[7] Bertrand Meyer, 酒匂寛訳: オブジェクト指向入門 第 2 版, 翔泳社, (2007).
[8] Overture Tool Formal Modeling in VDM, http://overturetool.org/, (最終確認日 2016 年 2 月 6 日).
[9] 山本修一郎: ゴール指向による システム要求管理技法, 株式会社ソフト・リサーチ・センター, (2007).
[10] Recpect-IT: A KAOS Tutorial, http://www.objectiver.com/fileadmin/download/documents/KaosTutorial.pdf, (最終確認日 2016 年 2 月 6 日).
[11] Engineer your requirements with Objectiver, http://objectiver.com/index.php?id=4, (最終確認日 2016 年 2 月 6 日).
[12] 中川博之, 吉岡信和, 本位田真一: IMPULSE:KAOS を利用したマルチエージェントシステムの分析モデル構築, 情報処理学会論文誌, Vol. 48, No. 8, pp.2551-2565, (2007).
[13] ASTER: ASTER 自動販売機ハードウェア構成および販売者用機能仕様書, http://aster.or.jp/business/contest/doc/2015tdc-v1_1.zip, (最終確認日 2016 年 2 月 6 日).
[14] ASTER: ASTER 自動販売機ユースケース仕様書, http://aster.or.jp/business/contest/doc/2015tdc-v1_1.zip, (最終確認日 2016 年 2 月 6 日).
[15] ASTER テスト設計コンテスト, http://aster.or.jp/business/contest/contest2015.html#agreement, (最終確認日 2016 年 2 月 6 日).

ⓒ 2016 Information Processing Society of Japan