MDAにおけるPIM作成支援を目的としたUMLダイアグラム洗練手法の提案

井上 陽* 片山 徹郎**

Proposal of a Refinement Method of UML Diagrams to Support Making PIM in MDA Development

Yo INOUE, Tetsuro KATAYAMA

Abstract

Recently, the development of software has come to be able to construct complex and large-scale systems. However, it has ploblems about productivity, portability, interoperability, and maintainability. In order to solve these problems, OMG(Object Management Group) has defined MDA(Model Driven Architecture). But, even if experts of modeling, it is difficult to make the models used in MDA called PIM(Platform Independent Model) or PSM(Platform Specific Model). In this paper, to reduce the burden in making PIM, it proposes a refinement method of UML(Unified Modeling Language) diagrams to support making PIM. The proposed method makes a more detailed UML class diagram from an original one by adding elements extracted from other behavioral UML diagrams in OCL(Object Constraint Language). The detailed class diagram can make PIM more easily. Consequently, it leads to improve productivity of software.

Key Words:
MDA(Model Driven Architecture), PIM(Platform Independent Model), UML(Unified Modeling Language), OCL(Object Constraint Language)

1. はじめに

ソフトウェア開発の分野は、オブジェクト指向技術や、UML(Unified Modeling Language)[1] などのモデリング言語を導入し、モデリング言語と、要求仕様書などの文書を併用することで、複雑で大規模なシステムを構築することができるようになった[2]。しかし、ソフトウェア開発の分野は、依然として生産性、移植性、相互運用性、及び、保守性の点に問題が存在している。これらの問題を解決するために、OMG(Object Management Group)[3] がMDA(Model Driven Architecture)[4] を定義した。MDA はソフトウェア開発プロセスのためのフレームワークで、ソフトウェア開発プロセスにおけるモデルの重要性に焦点を当てている。MDA を使用することで上述の問題を全て解決することができ、ソフトウェアの生産性と信頼性の向上につながる。MDA をシステム開発に採用する際、MDA に使用されるモデルはコンピュータによる自動解釈に適した明確な書式と意味、および十分な情報量を持っていなければならない。このようなモデルを、MDA では PIM(Platform Independent Model)[4] と呼ぶ。しかし、PIM の作成はモデリング熟練者であっても容易ではない。

PIM は、システムの分析段階で、モデル製作者が要求仕様書から作成したダイアグラムを洗練した結果、作成される。しかし、この作業は主に手作業で行われ

* 情報システム工学専攻大学院生 ** 情報システム工学科准教授

るため、非常に手間と労力がかかる。

そこで本研究では、PIM作成における作業量の削減を目的とし、分析段階におけるPIMの作成支援として、UMLダイアグラム洗練手法を提案する。具体的には、モデリング言語UMLで表されたダイアグラムから、分析した結果、作成されるPIMに使用可能な要素を抽出する。その後、抽出した要素を、モデリング言語OCL(Object Constraint Language)[5]を用いてUMLのクラス図に記述する。

提案する手法に基づいて作成されたクラス図は、分析段階において、ダイアグラムを洗練することで得られる情報を持つ。これにより、モデル開発者の分析における作業量を削減することができる。

本論文の構成として、第2章では、MDAの要素と、MDA実現のために使用されるモデリング言語について説明する。第3章ではUMLダイアグラムのアクティビティ図からの要素の抽出方法と、その要素のOCLを用いた記述法ついて述べる。第4章では適用例について述べ、第5章で本研究についての考察を行う。

2. MDA

MDA(Model Driven Architecture)[2)4)]とは、モデル駆動アーキテクチャの意味で、OMG (Object Management Group)[3)]によって定義されたソフトウェア開発のためのフレームワークである。MDAは、モデリング言語を、単なるデザイン言語としてではなく、プログラミング言語として使用することを目的としている。モデリング言語によるプログラミングは、ソフトウェアの生産性と品質を改善する。

MDAにおけるソフトウェア開発のプロセスと、プロセスにおける成果物を、図1に示す。MDAの中核となる2つのモデルとして、PIM(Platform Independent Model)[2)4)]とPSM(Platform Specific Model)[2)4)]が存在する。以下にPIMとPSMについて説明する。

- PIM
  PIM(Platform Independent Model)とは、MDAが定義するモデルの一つで、実装技術(プラットフォーム)から独立した抽象レベルの高いモデルである。ここで、プラットフォームとはOS、ミドルウェア、および、プログラム言語を指している。PIMは要求仕様をどのようにサポートするかという視点でシステムをモデル化する。
- PSM
  PSM(Platform Specific Model)とは、PIMから変換されて生成されるモデルであり、ある特定の実装技術で利用できる実装構造という視点で、システムを記述する。PSMは、特定のプラットフォームに関する知識を持つ開発者にだけ意味を成す。また、PSMは特定の技術プラットフォームごとの変換規則に従い、一つのPIMから複数生成される。

MDA開発プロセスでは、まず、要件から要求仕様書を作成し、次に要求仕様書の分析に入る。この分析では、要求仕様書からのダイアグラムの作成と、作成したダイアグラムの洗練を行う。この分析の結果、PIMが作成される。さらに、PIMはPSMへの変換規則に従い、一つまたは複数のPSMに自動変換される。PSMへの変換規則を定義することは大変な作業である。しかし、変換規則を定義する必要があるのは一度だけで、定義した後は、定義を繰り返し使用することができる。定義した変換規則を元にPIMからPSMを作成し、その後、PSMをコードに自動変換する。これがMDAの開発プロセスである。MDAの一つの特徴は、一つのPIMから複数のPSMを作成するステップを、自動化して行う点である。

MDAにおいて、モデルはモデリング言語を使って表される。本論文では、モデリング言語において、最も一般的なモデリング言語であるUMLを使用してモデルを作成する。

2.1 UML

UML(Unified Modeling Language)[1)]とは、統一モデリング言語の意味で、ビジネスや各種システムを対象として、その構造とダイナミクス(動的な振る舞いや挙動)を分かりやすく表現するためのビジュアルな言語である。UMLを導入することにより、ユーザと開発者、または、開発者同士のコミュニケーションギャップを解消することができる。また、ユーザからの要求を正確に把握できるようになるため、仕様の認識違いによる手戻りを削減できる。更に、UMLによるオブジェクト指向設計が効果的にモジュール化を促進し、保守コストを削減することも可能となる。

2.2 OCL

MDAにおける開発でUMLを使用することにより、システムの構造的な側面のモデリングが可能になり、PIMが完全にPSMに変換されるために必要な、PIMの完全性、一貫性、および、明確性を高めることができる。しかし、UMLにはPIMから完全なPSMへの変換を妨げる短所もいくつか存在する。UMLの短所は、各ダイアグラムの曖昧性にある。MDAにおいて、作成されたPIMの完全性、一貫性、明確性がきわめて高くなければ、PIMからPSMを作成するこ

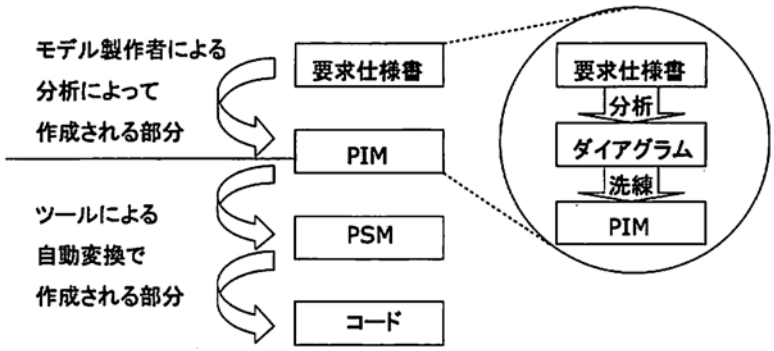

図 1: MDA 開発プロセスにおける成果物とその作成方法

とはできない。そこで使用されるモデリング言語がOCL(Object Constraint Language：オブジェクト制約言語)[5] である。

OCL とは、ソフトウェアのモデルを記述するためのモデリング言語の一つで、OMG のオブジェクト指向分析・設計のための UML の標準の追加機能として定義されている[3]。OCL で記述された式は、オブジェクト指向モデルや他のオブジェクトモデルのアーティファクト (成果物) に、条件や制約を追加することができる。この OCL と UML を組み合わせることにより、MDA で使用可能な完全性、一貫性、明確性を持った PIM を記述することができる。

3. ダイアグラム洗練支援ための UML からの要素抽出

本研究では、モデリング言語として最も一般的である UML の振る舞いを表すダイアグラムである、アクティビティ図、シーケンス図、コラボレーション図、ステートチャート図から、OCL を用いて条件として記述が可能な要素を抽出する。以下、アクティビティ図からの要素の抽出方法と、抽出した要素を OCL としてクラス図に付加するための記述法について述べる。図 2 に示すトランプゲームであるブラックジャックのアクティビティ図を、例として用いる。

3.1 クラス図と OCL

クラス図は、UML の中でも中心的なダイアグラムであり、概念や事象、それらの間の関連、各オブジェクトの持つ属性や操作を表現できるため、MDA で UML をモデリング言語として選択した場合、PIM はこのクラス図を使って表現されることが多い。そこで今回は、このクラス図に、システムの動的な側面を表す、アクティビティ図、シーケンス図、コラボレーション図、ステートチャート図から抽出した条件要素を、OCL の構文を用いて記述する。

クラス図を補強するために用いられる OCL の条件の定義には、以下のようなものがある。

- 不変条件
  不変条件は、条件が定義されている型のすべてのインスタンスが常に真でなければならない boolean 式
- 事前条件
  事前条件は、操作の実行を開始する瞬間に真でなければならない boolean 式
- 事後条件
  事後条件は、操作の実行が終了する瞬間に真でなければならない boolean 式

不変条件が破られた場合は、モデルが表すシステムに欠陥が存在することを意味する。よって不変条件はソフトウェアテストにも利用することができる。事前条件と事後条件は、アルゴリズムや実装を記述することなく、操作の適用可能性や結果を指定する制約である。事前条件と事後条件を記述することによって、モデルが表す、システムの仕様の完成度を高めることができる。ここで、不変条件はシステムが動作を実行している場合は、常に真であるべき必要はない (実行が終了した際に、再び真になればよい)。このため、動的な側面を表すダイアグラムから抽出した条件が不変条件なのかどうかの判断は困難であるため、今回は、事前条件と事後条件に絞って述べる。

3.2 アクティビティ図からの抽出

アクティビティ図は、一連の動作の処理の流れを表している。動作は一連の処理と対応しているため、直前の処理で変化が加えられた何らかの値を出力のガード条件とすることにより、それを次の状態への事前条件として利用することができる。また事前条件として直前の状態を参照することにより、システム全体の操

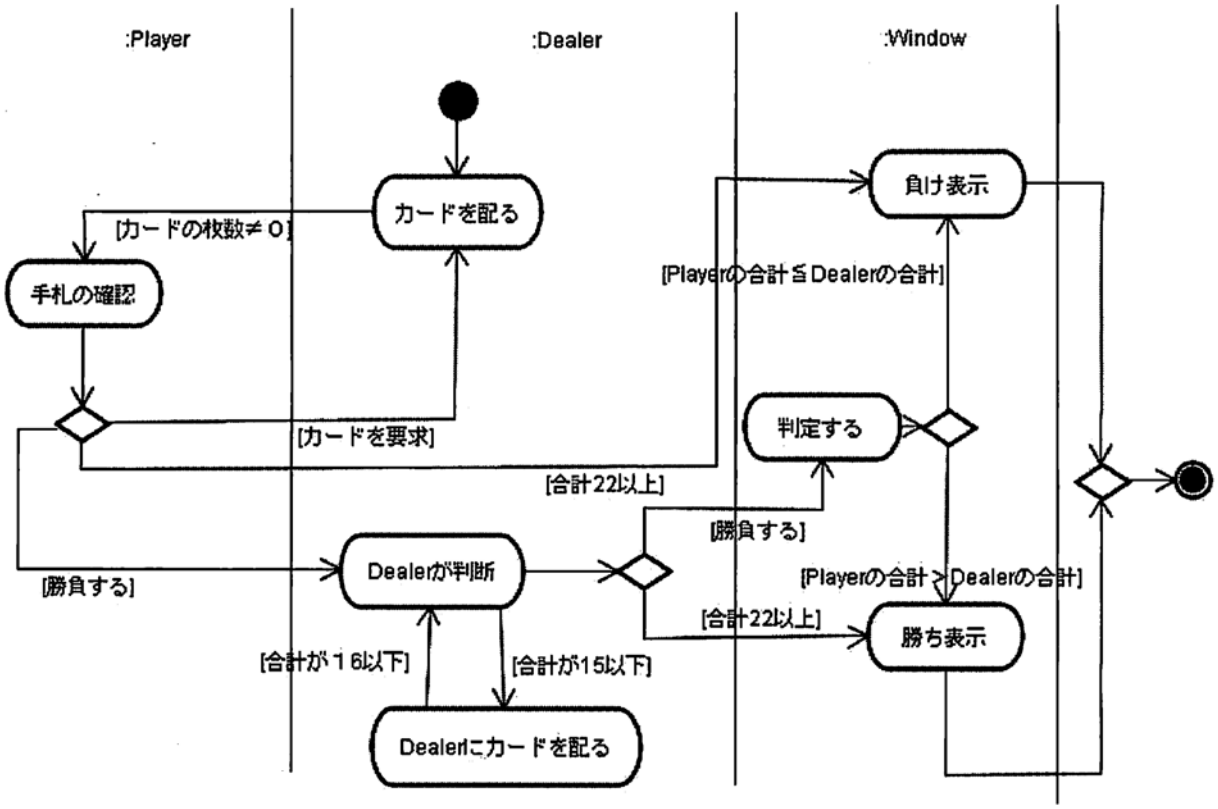

図 2: ブラックジャックのアクティビティ図

作の流れを、クラス図に明記することが可能である。さらに、ある動作によってオブジェクトが新たに生成された場合は、そのオブジェクトが生成されたという事象を事前条件、または、事後条件として利用できる。

各動作状態が記述されている部分をレーンといい、これはクラスに対応しているので、これによりレーン上の動作状態がどのクラスによって実行されるかを特定できる。さらに、ある動作状態の事前条件は、直前の動作状態の事後条件としても利用できる。

OCL を用いた事前条件と事後条件の構文は、

```
pre :  条件文
post :  条件文
```

のように記述する。「pre」は OCL で事前条件を表し、「post」は事後条件を表している。また、条件文でクラスや操作、属性の指定を行う場合はドット (.) 表記を使用し、クラス、操作、属性を指定する。ここで、条件として適用する要素が自身のクラス内のメソッドから発生する場合、自身を表すクラスを、「self」を用いることで表現する。

図 2 は、トランプゲームであるブラックジャックの全体の大まかな流れと動作状態とを示したアクティビティ図である。このアクティビティ図を用いて、アクティビティ図から抽出した要素と、その要素の OCL としての記述法を説明する。

まず Dealer クラスにおいて、開始状態から「カードを配る ()」という動作状態に遷移し、その動作を実行する。次に Player クラスの「手札の確認 ()」という動作状態に遷移する。しかし、「手札の確認 ()」への遷移の前にガード条件として書かれた [カードの枚数≠0] という事前条件を満たさなければならない。また、「手札の確認 ()」の動作状態からはデシジョン (分岐) によって 3 つの分岐に入る。この 3 つの分岐のそれぞれが動作状態「手札の確認 ()」の事後条件となる。以上を OCL の構文を用いると

```
context Player::手札の確認 ()
pre:Dealer. カードを配る ().[カードの枚数≠ 0]
post:self. 手札の確認 ().[カードを要求]
or
self. 手札の確認 ().[合計 22 以上]
or
self. 手札の確認 ().[勝負する]
```

と記述できる。ここで、context は属性、操作、関連端 (ロール名) を保持するクラスであり、Player クラ

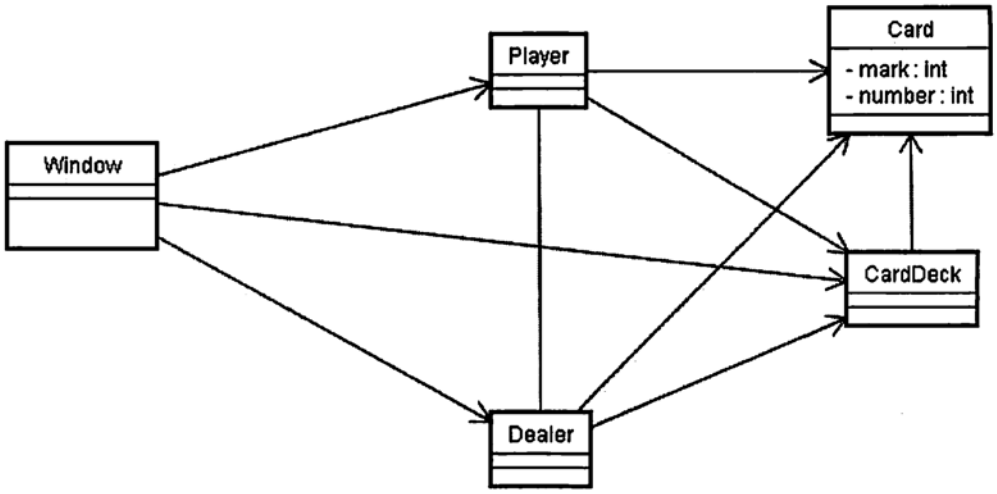

図 3: 要求分析段階で作成されたクラス図

スの「手札の確認 ()」という操作を指している。また、1 つの動作状態へ複数の動作状態からの遷移がある場合は、事前条件を「or」で結ぶことにより対応する、図 2 の例では、Window クラスの「負け表示 ()」、「勝ち表示 ()」の動作状態、Dealer クラスの「カードを配る ()」、「Dealer が判断する ()」の動作状態がこれにあたる。同様に 1 つの動作状態から複数の出力遷移が存在する場合は、事後条件を、前述の例の「カードを要求」、「合計 22 以上」、「勝負する」という 3 つの条件のように、「or」を用いてそれぞれの条件を結ぶ。

4. 適用例

第 3 章では、UML の振る舞いを表すダイアグラムからの、要素の抽出方法と、OCL を用いた要素の記述法について説明した。この章では、今回抽出した要素を OCL を用いて条件として付加することで、PIM として作成したクラス図の説明を行う。

図 3 に、要求段階のブラックジャックシステムのクラス図を示す。情報を付加するクラス図として、この図を使用する。

今回提案する手法を用いてアクティビティ図から作成したクラス図を、図 4 に示す。図 4 に示している各クラスの操作は、アクティビティ図から抽出したものである。その操作が持つ条件を OCL を用いて、図右側にテキストとして記述している。

ここで、Dealer クラスの「カードを配る ()」という操作に注目する。図 4 の右側の OCL の記述を見ると、この「カードを配る ()」という操作の事前条件に Player クラスの操作「手札の確認 ()」においての「カードを要求」という要素を適用している。また事後条件として、自身のクラスである Dealer クラスの操作「カードを配る ()」において「カードの枚数≠ 0」という要素を適用している。また、操作「カードを配る ()」の事前条件は、Player クラスの操作「手札の確認 ()」の事後条件の中の一つである。さらに、操作「カードを配る ()」の事後条件は、Player クラスの操作「手札の確認 ()」の事前条件になっている。以上から、Dealer クラスの「カードを配る ()」という操作と、Player クラスの「手札の確認 ()」という操作とは、特定の条件においてループするという関係があることが分かる。

以上のようにして、アクティビティ図から抽出した要素を、OCL を用いてクラス図に付加することで、本来クラス図には記述されない、各操作が持つ条件分岐とシステム全体の操作の流れとを追加することができる。

今回は、UML の動的振る舞いを表すアクティビティ図からだけでなく、シーケンス図、コラボレーション図、ステートチャート図からも、OCL として利用可能な要素を抽出した。抽出した要素と、OCL を用いた要素の記述法をまとめたものを、表 1 に示す。

5. 考察

本研究では、MDA プロセスにおける、分析段階のダイアグラムの洗練のための作業量の削減を目的として、PIM の作成に必要な、UML の振る舞いを表すダイアグラム中に存在する、OCL として記述が可能な要素の抽出と、抽出した要素の OCL としての記述法の提案を行った。本論文では、アクティビティ図、シーケンス図、コラボレーション図、ステートチャート図

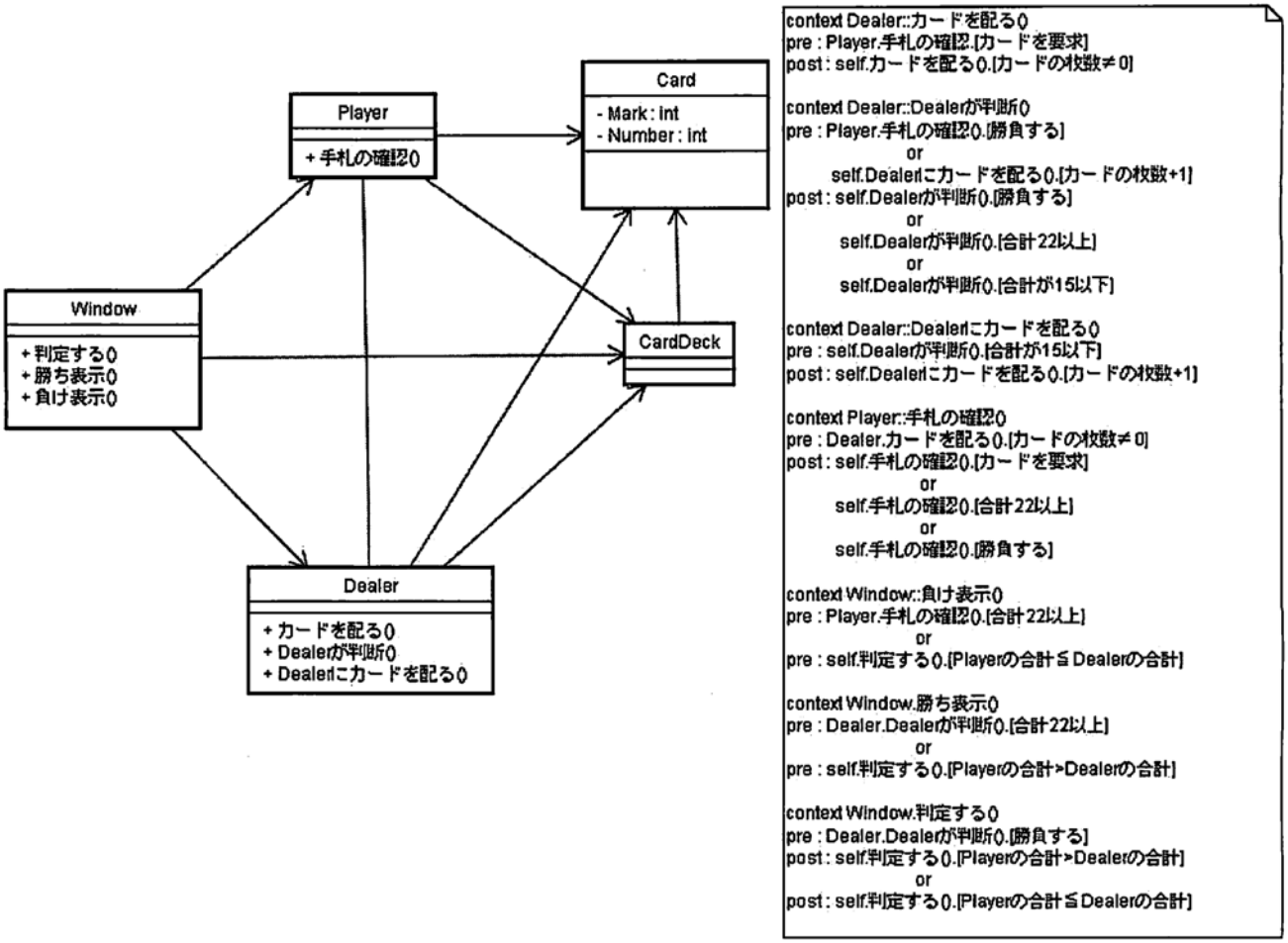

図 4: アクティビティ図から抽出した要素を基に作成したクラス図

を抽出対象とした。そして、抽出した要素を基にクラス図を作成することによって、以下のような情報をクラス図に付加することができた。

- アクティビティ図からの抽出
  アクティビティ図からの変換によって、システムの全体的な操作の流れと、各操作の持つ条件分岐を追加することができる。これらの情報は本来クラス図を記述する上では、ダイアグラム中には記述されない情報である。しかし、これらはシステムの動作を理解するために省略できないものである。この情報を、クラス図に OCL として追加することにより、クラス図により多くの情報を持たせることができる。

- シーケンス図、コラボレーション図からの抽出
  通常クラス図において、クラス間の関係は関連を用いて表される。しかし、その関連を使い、どの操作が通信を行うか、ということまでは記述されない。事後条件を用いて、メッセージの通信を表すことにより、クラス図にクラスが持つ操作が通信対象としているクラスを明記することができる。これにより、クラス図が持つ情報量を増加できる。

- ステートチャート図からの抽出
  本来クラス図を記述する上では、クラス図には操作が実行されるための必要な条件は記述されない。ステートチャート図のイベントを抽出し、OCL として記述することで、クラスの操作が持つ実行のための条件を記述し、クラス図の持つ情報量を増加できる。ここで、ステートチャート図において表現されるイベントは、そのイベント自体が、クラスの操作になっている場合がある。また、イベントにガード条件が付く場合があり、ガード条件はイベントの事前条件として利用が可能である。しかし、今回はイベント自体を条件として抽出したので、イベントが表す操作や、イベントのガード条件は抽出対象とはしなかった。

表 1: UML のダイアグラム内の要素と OCL を用いた記述法

ダイアグラム	要素	OCL を用いた記述法
アクティビティ図	ガード条件	出力遷移上のガード条件を事後条件、入力遷移上のガード条件を事前条件として利用できる。
	オブジェクトの生成	オブジェクトが生成されたという事象を、事前条件や事後条件として利用できる。
シーケンス図	メッセージ	メッセージ内のガード条件、繰り返し条件を事前条件や事後条件として利用できる。またメッセージの送信自体を事後条件として利用できる。しかしメッセージの送信は、OCL の構文の性質上、そのまま事前条件に用いることはできない。
コラボレーション図	メッセージ	メッセージ内のガード条件、繰り返し条件を事前条件や事後条件として利用できる。またメッセージの送信自体を事後条件として利用できる。しかしメッセージの送信は、OCL の構文の性質上、そのまま事前条件に用いることはできない。
ステートチャート図	イベント名	アクションを持つ状態へ遷移するイベントを、そのアクションを実行するための事前条件として、アクションを持つ状態から遷移するイベントを事後条件として利用できる。

表 2: クラス図に付加できた要素

ダイアグラム	付加できた操作数	付加できた条件数
アクティビティ図	7	20
シーケンス図	7	12
コラボレーション図	7	12
ステートチャート図	1	5

表 2 に、今回提案する手法を用いて、クラス図にどの程度、抽出した情報を付加できたかを示す。表 2 に示すクラス図に付加できた要素は、いずれも、分析段階において、モデル開発者がダイアグラムを洗練した結果得られる要素である。本論文で提案する手法を用いることで、モデル開発者は、分析段階におけるダイアグラムの洗練の手間を削減できる。これにより、より容易に PIM を作成することができ、ソフトウェアの生産性が向上する。

ここで、表 2 を見るとステートチャート図からクラス図に付加できた要素が、他のダイアグラムに比べ少ないことが分かる。この理由としては、使用したステートチャート図に記述された情報が少なかったためと考えられる。したがって、ダイアグラムが詳細に書かれていれば、クラス図に付加できる要素も増加すると考えられる。

MDA 支援ツールはいくつか存在している。しかし、そのほとんどが、PIM から PSM への変換、または、PSM からのコード変換が中心であり[6)7)]、MDA で使用可能な PIM の作成は、モデル開発者による手作業によって行われている。モデル開発者は、分析段階で、要求仕様書からダイアグラムを作成し、そのダイアグラムを洗練していくことで、PIM の作成を行う。この際に、本研究で提案した手法を用いることで、ダイアグラムの洗練の際の手作業の負担を減らし、MDA で使用可能な PIM の作成の支援を行うことができる。

また、UML をテストを目的に使用し、生産性、信頼性の向上を目的とした研究も最近行われてきている[8)]。第 3 章で述べたように、モデルが表すシステムの欠陥の有無を知ることができる要素として、不変条件

が存在する。この不変条件を抽出し、その値を参照することによってモデル上のシステムをテストする方法について、今後検討する。具体的には、モデルの動作をシミュレートし、不変条件の値を抽出する。抽出した不変条件の値を調べ、条件が破られたかどうかを確認する。もし、条件が破られた場合は、システムに欠陥が存在することを意味するので、これによりモデルに記述されたシステムをテストすることができると考えている。

6. おわりに

本研究では、分析段階のダイアグラムを洗練する際の作業量の削減を目的として、PIMの作成に必要な、UMLの振る舞いを表すダイアグラムから、OCLとして記述が可能な要素を抽出し、抽出した要素についてOCLを用いた記述法を提案をした。本論文では、アクティビティ図、シーケンス図、コラボレーション図、ステートチャート図を抽出対象とした。

現在UMLを用いたシステム開発が非常に増えてきており、一般的になりつつある。その中で、MDAという考え方が誕生した[4]。MDAにおけるPIM作成の際に、本研究で提案する要素の抽出方法を利用することで、PIMの作成に必要なダイアグラムの分析の手間を削減することができる。これは、MDAプロセスの支援にもなり、その結果、ソフトウェアの信頼性、および、生産性の向上につながると考えられる。

以下に今後の課題を示す。

- UMLの構造を表すダイアグラムからの要素抽出
  本研究では、UMLの振る舞いを表すダイアグラムからの要素の抽出を行った。しかし、UMLにはシステムの構造を示すダイアグラムとして、オブジェクト図、コンポーネント図、ユースケース図、配置図が存在する。これらのダイアグラムも分析段階で作成され、分析の対象となり得ると考えられる。そこで今後は、構造を表すダイアグラムからの要素の抽出についても考える必要がある。
- 抽出した要素の充分性の検証
  今回抽出した要素に対して、充分性を検証する必要がある。今回PIMとして作成したクラス図は、より詳細なモデルではあるものの、MDAにおいて使用できるほど、完成されてはいない。また、並行動作、ステートチャート図のイベント、ガード条件などに関しては、現段階では、要素として抽出することができなかった。今後は、より多くの事例を参考にすることにより、ダイアグラムからの更なる要素の抽出に努め、ダイアグラムの分析支援を目指す。
- ダイアグラム分析支援ツールの作成
  今回は、UMLの振る舞いを表すダイアグラムからの要素の抽出と、その利用法を提案した。今後は、今回提案した手法を基に、ダイアグラム分析支援ツールを開発し、ツールを実際に運用することによって、今回提案した手法の有用性を検証し、評価する。
- MDA上におけるUMLを用いたテスト手法の提案
  モデルが表すシステムの欠陥の有無を知ることができる要素として、不変条件が存在する。この不変条件を抽出し、その値を参照することによってモデル上のシステムをテストする方法について、今後検討する。具体的には、モデルの動作をシミュレートし、不変条件の値を抽出する。抽出した不変条件の値を調べ、条件が破られたかどうかを確認する。もし、条件が破られた場合は、システムに欠陥が存在することを意味するので、これによりモデルに記述されたシステムをテストすることができると考えている。

参考文献

1) UML Home Page: http://www.uml.org/
2) Anneke Kleppe, Jos Warmer, Wim Bast 著: 株式会社テクノロジックアート 訳:「MDA(モデル駆動型アーキテクチャ)導入ガイド UMLを基盤としたオブジェクト開発」, 株式会社インプレス (2003).
3) OMG ウェブサイト: http://www.omg.org/mda/
4) David S. Frankel 著: 日本アイ・ビー・エム株式会社 TEC − J MDA 分科会 訳:「MDA モデル駆動アーキテクチャ」, エスアイビーアクセス (2003).
5) Jos Warmer, Anneke Kleppe 著:竹村 司 訳: 「UML/MDAのためのオブジェクト制約言語 OCL 第2版」, 星雲社 (2004).
6) Jude ホームページ: http//objectclub.esm.co.jp/jude/jude.html
7) IIOS ホームページ: http://www.iios.org/index-j.html
8) U2TP ウェブサイト: http://www.fokus.fraunhofer.de/u2tp/