UMLメタモデル仕様とOCL制約に基づくUMLモデル検証環境の生成

浜口 優[†] 長井栄吾[†] 松下 誠[†]
岡野浩三[†] 楠本真二[†] 井上克郎[†]
我妻智之[††] 梅村晃広[††]

オブジェクト指向型言語を用いたソフトウェア開発において広く利用される UML (Unified Modeling Language) では，複数種類のダイアグラムが定義されている．しかしこれら複数種類のダイアグラムは完全に独立しておらず，そのためシステム設計の過程で不整合が生じる可能性がある．これに対して，UML メタモデルへ制約を付加することにより UML ダイアグラム間整合性検証を行う研究がある．しかし，それらの研究では UML メタモデルのバージョン変更に対応できない．そのため，整合性検証を行う場合は各バージョンの UML メタモデルに対して個別のモデル検証環境を用意する必要がある．本研究では，UML メタモデルとオブジェクト制約記述言語 OCL (Object Constraint Language) を用いて記述した制約を，モデル駆動型開発を支援するフレームワーク EMF (Eclipse Modeling Framework) に対して入力として与えることで，UML モデル検証環境を自動生成する手法を提案する．具体的には，EMF のモデル表現形式である Ecore モデルを UML メタモデルと整合性ルールを表現した OCL 制約から生成する．次に，生成された Ecore モデルを EMF に対して入力として与えることで UML モデル検証環境のソースコードを自動生成する．また提案手法に基づいて UML ダイアグラム間整合性検証を支援するツールを作成した．DrinkMaker の例題に対して適用した結果，提案手法による UML ダイアグラム間整合性検証が可能であることが確認できた．

Generating UML Model Verification Environment Based on UML Meta Model Specification and OCL Constraints

Yuu Hamaguchi,[†] Eigo Nagai,[†] Makoto Matsushita,[†]
Kozo Okano,[†] Shinji Kusumoto,[†] Katsuro Inoue,[†]
Tomoyuki Agatsuma[††] and Akihiro Umemura[††]

Several kinds of diagrams defined in UML (Unified Modeling Language) are widely used in Object-Oriented Software Development. They are, however, not completely independent from each other, and therefore inconsistencies between diagrams may occur in the process of the system design. For this reason, there are some studies to perform consistency verification between the UML diagrams by adding constraints to UML meta models. However, those studies cannot support the various versions of the UML meta models. Therefore, it is necessary to prepare each individual model verification environment for the corresponding UML meta models. In this paper, we propose a technique to generate UML model verification environment automatically by giving a UML meta model and constraints described in OCL (Object Constraint Language) as inputs for EMF (Eclipse Modeling Framework) that supports Model Driven Architecture. More precisely, we generate an Ecore model by using a UML meta model and the OCL constraints expressed consistency rules. Then, we generate automatically a source code of the UML model verification environment by giving the Ecore model as input for EMF. Based on the technique, we have developed a tool supporting the consistency verification based on the UML diagrams. As a result of the experiments, we confirmed that it was possible to verify UML diagrams consistency by our proposed technique.

1. まえがき

近年，ソフトウェア開発において，UML (Unified Model Language) がソフトウェアのモデル記述に幅広く利用されている[8]．UML モデルには，様々な視点からソフトウェアシステムを表現するために複数の

† 大阪大学 大学院情報科学研究科
Graduate School of Information Science and Technology, Osaka University
†† (株)NTT データ
NTT DATA Corp.

UML ダイアグラムが存在する．また，UML を用いたモデル駆動型開発では，段階的に UML ダイアグラムを修正，詳細化していく．

しかし，各 UML ダイアグラムは完全に独立していないために，段階的な各工程の中で名前要素の不一致などの不整合を起こす可能性がある．この問題に対して取り組んでいる既存研究やモデリングツールが存在する．Rational Rose[7] は UML ダイアグラム間の整合性を保つ機能を備えているが，通常のモデル記述に加えて，UML ダイアグラムの構成要素間に依存関係を定義する処理が必要になる．文献 18) の研究や UML/Analyzer[5] は不整合の修正を目的とし，修正動作の選択肢を提示する．しかし，UML メタモデルはよく変更されるにもかかわらず[16]，上記の研究で実装している UML メタモデルは変更することができない．したがって頻繁な UML メタモデルの変更に対して，逐一対応するためのコストが生じる．

そこで本研究では UML メタモデルと OMG[11] の標準規格である制約記述言語 OCL (Object Constraint Language)[15] で記述された制約式を入力として与えることで，UML モデル検証環境を自動生成する手法を提案する．具体的には，EMF (Eclipse Modeling Framework)[3] に基づいて，OMG より入手可能である Rose モデルで表現された UML メタモデルと，OCL で記述された整合性ルールを入力として与えることで EMF のモデル表現形式である Ecore モデルを生成する．次に Ecore モデルから EMF モデルを生成し，EMF モデルから UML モデル検証環境のソースコードを自動生成する．UML メタモデルから Ecore モデルへは変換可能であり，Ecore モデルは OCL 制約を付加することができる．ユーザは生成された UML モデル検証環境を用いて任意の UML ダイアグラムを編集し検証することができる．また提案手法に基づいて支援ツールを作成した．適用実験ではこのツールを用いてクラス図とシーケンス図間の整合性検証を行う．

以降，2 章で提案する手法の背景について述べ，3 章で UML モデル検証環境を構築する手法を述べる．4 章で適用実験を行い，5 章で実験の考察と評価を述べ，最後に 6 章でまとめと今後の課題を述べる．

2. 準　　備

本章では，提案する手法を説明するにあたり必要な背景を述べ，既存研究とその問題点について考察する．

2.1 UML モデルと UML メタモデル

OMG は UML を 4 階層のアーキテクチャを用いて設計している．

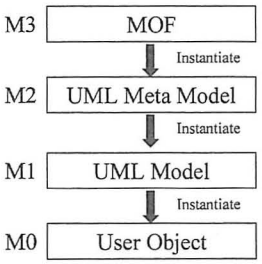

図 1　OMG のメタモデル 4 層
Fig. 1　Hierarchical structure of the metamodel of OMG

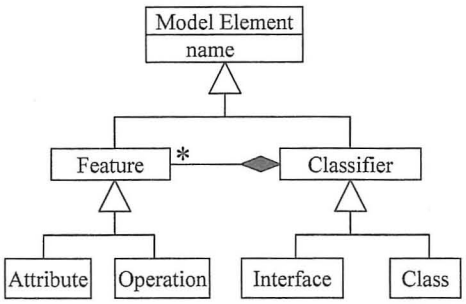

図 2　簡略化した UML メタモデル
Fig. 2　Simplified UML metamodel

(1)　MOF(Meta Object Facility)[12] : M3 層に相当し，UML メタモデルを記述するための言語を定義する．

(2)　UML メタモデル : MOF のインスタンス．M2 層に相当し，UML モデルを記述するための言語を定義する．

(3)　UML モデル : UML メタモデルのインスタンス．M1 層に相当し，オブジェクトモデルを記述するための言語を定義する．

(4)　オブジェクトモデル : UML モデルのインスタンス．M0 層に相当し，特定のオブジェクトを表現する．

UML メタモデルを定義することで UML モデルの仕様が決定される．したがって UML メタモデルに制約を付加することで UML モデルを制御することが可能になる．

図 2 はクラス図に関係する部分を簡略化した UML メタモデルである．クラス図のクラスは分類子を継承し，名前，属性，操作を持つことが定義されている．

2.2 オブジェクト制約言語 OCL

OCL は OMG 標準の 1 つであり，制約式を第 1 階論理で記述する．OCL は UML モデル内のモデル要素に対して正確に制約を与えることを目的に導入された．具体的には，OCL 式をクラスに付与することで，図 3 のようにクラスのインスタンスに対して制約を課

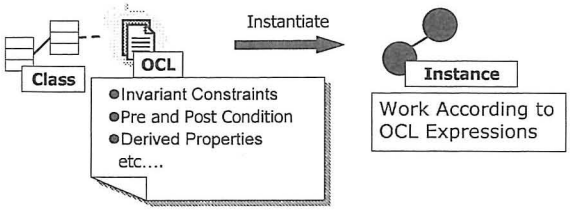

図 3 OCL 制約の実行
Fig. 3 Implementation of the OCL Constraints

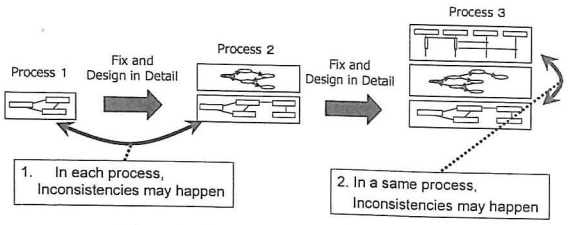

図 4 UML 開発工程における問題点
Fig. 4 Problems in the UML development process

すことが可能になる．

OCL は表 1 の条件式を宣言的な型付き言語で記述することにより，主にクラス仕様を表現する．OCL は，基本型，コレクション型，モデル型を持つ．基本型には，整数型，実数型，論理型，文字列型がある．コレクション型はクラスのインスタンスであるオブジェクト集合を表す．モデル型はクラスを表す．OCL 演算子では基本型やコレクション型に関する演算が定義されている．

2.3 UML ダイアグラム間の不整合

UML を用いたモデル駆動型開発では，段階的に UML ダイアグラムを修正，詳細化していく．しかし段階的な各工程の中で，図 4 のように UML ダイアグラム間には大別して 2 種類の不整合が発生し得る[6]．

(1) 各工程間において，同一種類の UML ダイアグラム間で発生する不整合
(2) 同一工程内において，異なる種類の UML ダイアグラム間と同一種類の UML ダイアグラム内で発生する不整合

(1) の不整合に対して文献 19) は各工程間に関する制約を与えて検証を行っている．本研究では (2) の不整合に対してユーザが行う検証を目的とする．

表 1 OCL の用途
Table 1 Use of OCL

分類	対象	説明
不変表明	クラス	すべてのオブジェクトが満たすべき条件を指定する
事前事後条件	操作	クラスの操作が実行前に満たすべき事前条件と，操作実行後に満たすべき条件を指定する
ナビゲーション	関連	互いに関連づけられているクラス間で，関連先に対する検索条件を指定する
派生	値	ある属性値が別の属性からどのように計算されるかを指定する
ガード条件	状態遷移メッセージ	状態遷移やシーケンス送信で，複数の選択肢がある場合，それぞれの選択肢が実行される条件を指定する

2.4 既存研究

UML モデルに対する検証手法はいくつか存在する．USE[17] は UML1.3 メタモデルの一部を実装し，UML モデルが OCL 式を満たすかをシミュレーションベースで検証する．OCLE[2] は UML1.5，OCL2.0 をサポートし，OMG が規定した WFR の整合性検証を行っている．

そして，2.3 で述べている同一工程内における UML ダイアグラム間整合性検証に関する研究も行われている．文献 18) の研究では，XMI[14] 形式で保存された UML モデルに対して，独自の記述文法で記述した整合性ルールを用いて整合性検証を行っている．開発したツールでは，整合性ルールと違反した場合に対応する解消動作の組を保存しておくことで，不整合要素の位置情報とそれを修正する解消動作系列の集合をユーザに提示する．UML/Analyzer[5] は Rational Rose[7] に統合されており，独自の記述文法で記述した整合性ルールに影響する全てのモデル要素をスコープとして保存する．これにより整合性ルールに違反した場合は，違反したモデル要素の属するスコープを提示し，不整合の修正を支援する．

しかし，上記の研究で実装しているツールの UML メタモデルは変更することができない．したがって過去の UML ダイアグラムのような設計資産の再利用が困難であり，また頻繁なバージョンチェンジを行う UML メタモデルに対して，逐一対応するコストが生じる．そこで本研究では，UML メタモデルの変更に柔軟に対応した UML ダイアグラム間整合性検証を目的とする．

3. 提案手法

本章では提案手法と適用した整合性ルールの詳細を述べ，実装したツールの詳細を述べる．

3.1 概　　要

本研究では EMF に基づいて，UML メタモデルと OCL で記述された UML ダイアグラム間整合性ルールを入力として与えることで，UML モデル検証環境

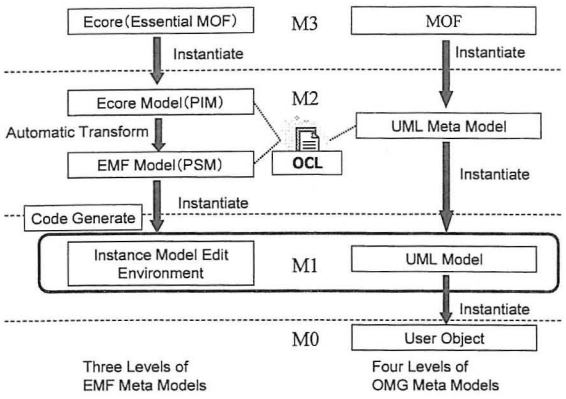

図 5 UML ダイアグラム間検証における EMF と UML の対応関係

Fig. 5 Correspondence between EMF and UML in Verification of UML Diagrams

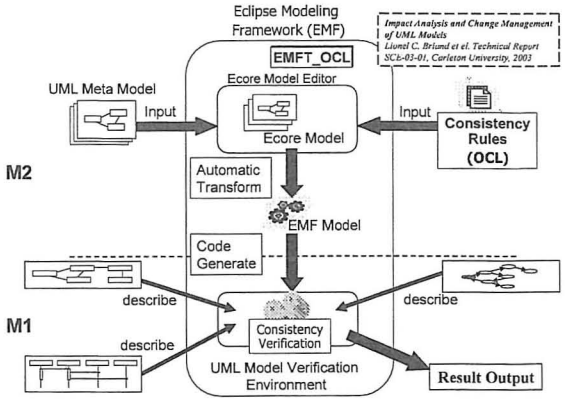

図 6 提案するシステム

Fig. 6 The Proposed System

を自動生成する手法を提案する．ユーザは生成された UML モデル検証環境を用いて任意の UML ダイアグラムを編集し検証することができる．

3.2 UML モデル検証環境生成手法の提案

EMF[3] は Eclipse Modeling Project が作成したプロダクトの 1 つであり，モデル駆動型開発をサポートするモデル記述とコード生成のためのフレームワークである．XML スキーマ，注釈付き Java コード，または Rose モデルを入力とすることで業務モデル (Platform Independent Model，図 5 では PIM) である Ecore モデルと，実装依存モデル (Platform Specific Model，図 5 では PSM) である EMF モデルを生成する．EMF は MOF に準拠しており，モデル変換に関して OMG の策定したモデル変換規格 QVT[13] に従っている．したがって MOF 準拠した Rose モデルから Ecore モデル，EMF モデルへの変換が可能である．Ecore モデルを編集することで対応する EMF モデルは自動更新され，EMF モデルからモデルのインスタンス編集環境をコード生成することができる．インスタンス編集環境では Ecore モデルのインスタンスおよびインスタンス間の関連が XMI[14] に基づく木構造で管理されている．またこの環境は木構造に基づくユーザインターフェースを提供している．また EMFT_OCL[4] プラグインを利用することで，Ecore モデルに付加された OCL 制約をインスタンスに対して検証することが可能になる．

図 5 の左側は EMF の 3 階層アーキテクチャである．Ecore モデル，EMF モデルは MOF[12] のサブセットである Essential MOF のインスタンスであり，インスタンス編集環境は Ecore モデル，EMF モデルのインスタンスである．

提案手法では図 5 のように，UML メタモデルと Ecore モデル，EMF モデルを対応させた．そうすることにより UML モデル編集環境が構築され，編集と OCL 検証が可能になる．図 6 は構築した UML モデル検証環境を表している．UML メタモデルを記述した Rose モデルを入力として与えると Ecore モデルが生成され，Ecore モデルから EMF モデルが生成される．Ecore モデルに対して整合性ルールを OCL 式で追加することで，EMF モデルに対しても同期して制約が追加される．最後に EMF モデルからコード生成することで UML モデル検証環境が構築される．

3.3 整合性ルール

UML モデル検証環境では，L. C. Briand らが定義した文献 1) に記述されている整合性ルールを利用して検証を行う．記述された整合性ルールは同一ダイアグラム内の整合性ルールとダイアグラム間の整合性ルールに大別されるが，本研究ではダイアグラム間の整合性ルールに着目して検証を行う．

3.4 実　　装

提案する手法を実現するために EMF の `org.eclipse.emf.ecore.presentation` パッケージ内に含まれる Ecore モデルエディタ (`EcoreEditor` クラス) を拡張した．拡張した Ecore モデルエディタにより，OCL 式の編集と，各 UML ダイアグラムに必要な UML メタモデル要素の管理を支援することができる．

インスタンス編集環境は Ecore モデル内で定義されたパッケージ毎に生成され，選択したパッケージ内のクラスのインスタンスとそのインスタンス間の関連を XMI に基づく木構造で管理する．よって，別パッケージ内に存在する関連を持たないクラスのインスタンス同士を同一インスタンス編集環境で管理すること

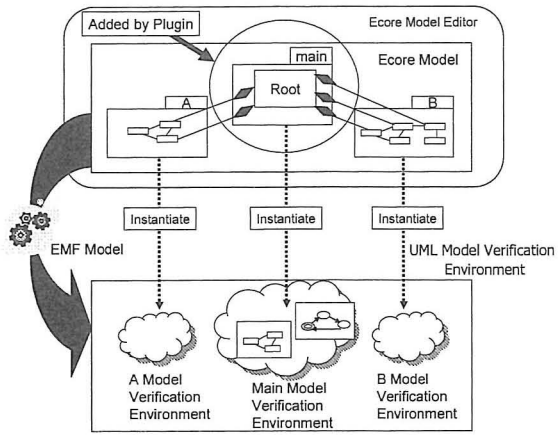

図 7 提案手法が行うインスタンス化
Fig. 7 Instantiation of UML Metamodel by the Proposed Tool

はできない．試しに EMF を用いて UML メタモデルから Ecore モデルを生成したところ，複数のパッケージに分割されており，その Ecore モデルから最終的に 46 個のインスタンス編集環境が生成された．実際の各 UML ダイアグラムを表現するために必要となる UML メタモデルのクラス間には，関連が存在しない場合がある．そのため，この分割されたインスタンス編集環境では適切に UML ダイアグラムを管理することができなかった．したがって，我々はこの問題を解決するために Ecore モデルを拡張することにした．具体的には，図 7 で示すように，Ecore モデルに main パッケージと Root クラスを追加する．Root クラスが UML ダイアグラムに必要な UML メタモデルを包含することにより，別パッケージにある UML メタモデルを同一インスタンス編集環境においてインスタンス化することが可能になる．これにより UML モデル検証環境において適切に UML ダイアグラムを管理することが可能になる．また，我々はこの一連の作業を支援するために Ecore モデルエディタを拡張することで提案手法をツール化した．

図 8 は実装したツールの UI である．UML ダイアグラムの選択と OCL 式の入力ができる．

図 9 はツールの出力結果である．Ecore モデルに対して main パッケージと Root クラスが追加されている．Root クラスでは以下の項目が定義される．

- ツールの UI で選択した UML ダイアグラムに対応する UML メタモデル内のモデル要素への包含関係
- ツールの UI で記述した OCL 式とその関数名

これにより図 7 で示している main モデル検証環境

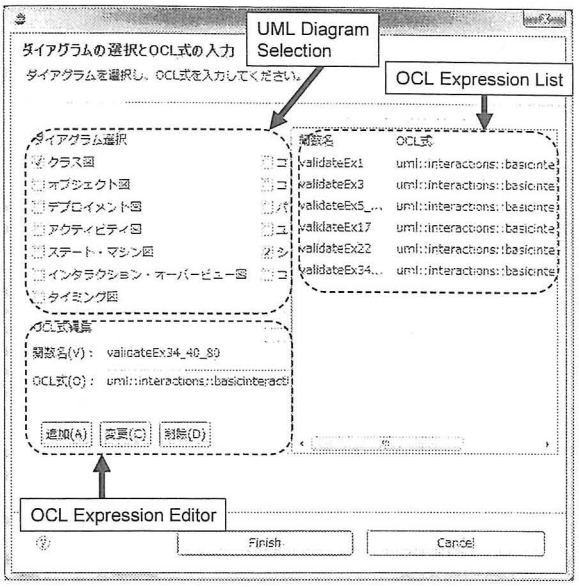

図 8 プラグインの UI
Fig. 8 UI of the Plugin

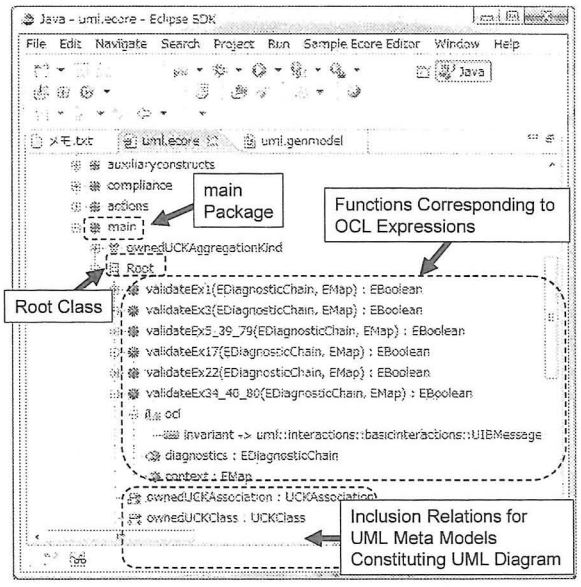

図 9 プラグインの出力
Fig. 9 Output of the Plugin

では，選択した UML ダイアグラムの表現に必要な UML メタモデルのインスタンスを生成することが可能になる．また，それぞれの UML ダイアグラムを構築するために必要な UML メタモデル内のモデル要素は UML の仕様書[16] から特定した．

図 10 は main モデル検証環境である．Root クラスを起点としてツリー構造のエディタで編集を行っていく．Root クラスを選択し validate ボタンを押すこと

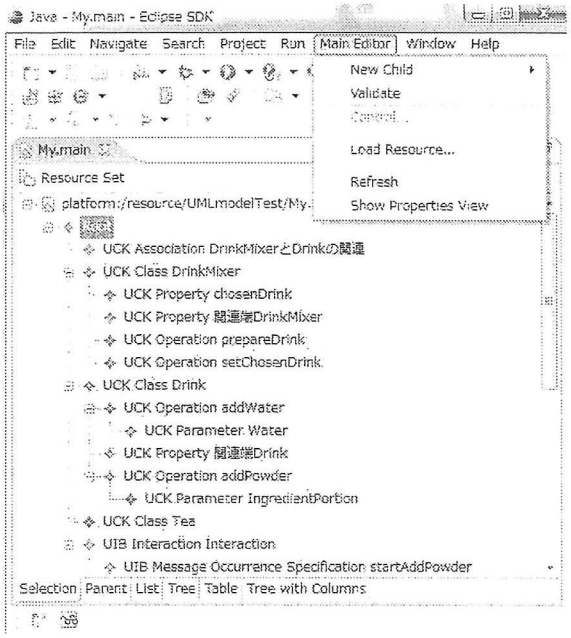

図 10 main モデル編集環境
Fig. 10 Main model Verification Environment

```
uml::interactions::basicinteractions::Lifeline.allInstances()->forAll(l |
  uml::classes::kernel::Class.allInstances()
  ->select(oclIsTypeOf(uml::classes::kernel::Class))->collect(Name)
  ->includesAll(l->collect(Name))
)
```

図 11 整合性ルールの OCL 表記例
Fig. 11 OCL Expression Example of the Consistency Rule

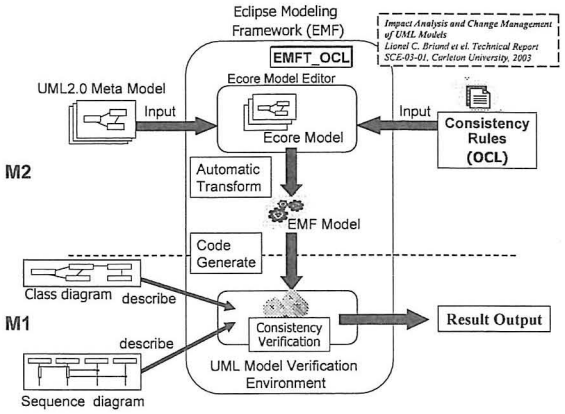

図 12 実験設定
Fig. 12 Experiment Setting

で，Root クラスに定義した OCL 制約を検証することができる．検証結果はダイアログで出力される．違反が検出された場合は，追加した OCL 式と対応する関数名が表示される．

4. 実　　験

本章では提案した手法が UML ダイアグラム間整合性検証に可能かどうかを評価するために行った実験とその結果について述べる．しかしこの実験は予備実験である．対象にした UML メタモデルは 1 つのみであり，本来は複数の UML メタモデルに対して適用実験を行う必要がある．

対象にした UML ダイアグラムは UML2.0[16] のクラス図とシーケンス図である．整合性ルールは文献 1) に記述された 115 個の整合性ルールの中からクラス図とシーケンス図間に関する全てのルール 10 個である．10 個のルールは重複する内容を考慮すると実際は 6 個である．以下が検証に用いた整合性ルールである．

(1) シーケンス図中において，protected なオペレーションはそのコンテナクラスを継承していないクラスに属するオペレーションからは呼び出されない．
(2) シーケンス図中において，private なオペレーションは他のクラスに属するオペレーションからは呼び出されない．
(3) シーケンス図中において，オブジェクトはクラス図中のクラスのインスタンスでなければならない．
(4) シーケンス図中において，2 つのオブジェクト間の各メッセージには，適切なパスがなければならない．
(5) シーケンス図中において，抽象オペレーションは呼び出すことができない．
(6) シーケンスメッセージ中で呼ばれる各オペレーションはクラス図で定義されていなければならない．

OCL 式は，まず UML2.0 仕様書より上記の整合性ルールに関係する UML メタモデル集合を特定し，次にそのインスタンスに関する不変式で表現する．評価結果は真偽値である．図 11 は整合性ルール (3) の OCL 表記例である．シーケンス図に存在する全てのオブジェクトはクラス図に存在するクラスと同じ名前を持たなければいけないことを示している．

図 12 は今回の実験で構築される検証環境である．UML2.0 メタモデルと上記の整合性ルールの OCL 式を用いて main モデル検証環境を生成し，クラス図とシーケンス図間の整合性検証を行った．実験環境は以下の通りである．OS : Windows Vista Business, CPU : Intel(R) Core(TM) 2 CPU 1.86GHz 1.87GHz, Memory : 2031MB RAM, Eclipse Ver-

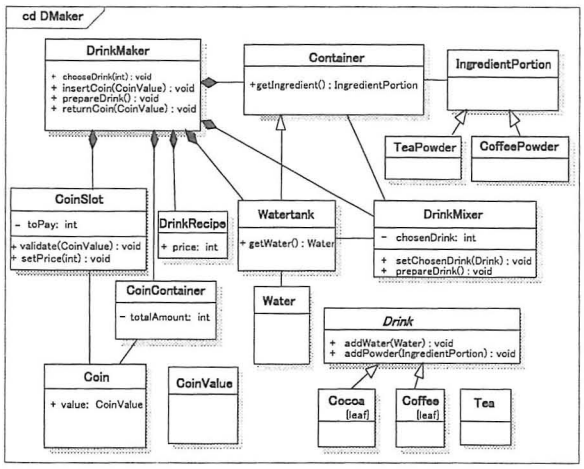

図 13 例題クラス図
Fig. 13 Class Diagram

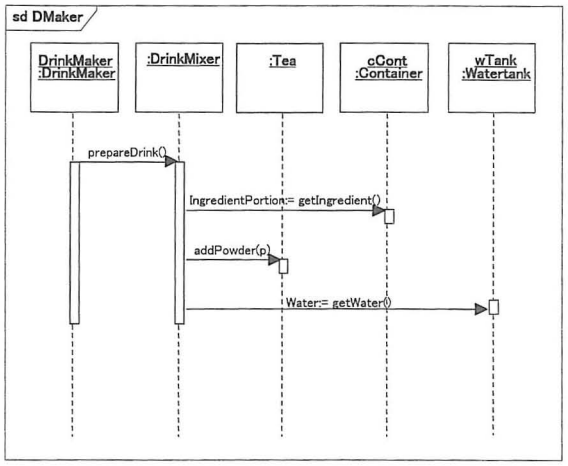

図 14 例題シーケンス図
Fig. 14 Sequence Diagram

sion : 3.2.0, EMF Version : 2.2.0

4.1 適用例題

ドリンクメーカー (DrinkMaker) はコーヒーや紅茶などの飲料を作る，レストランなどに置かれている機械である[10]．この例題に関するクラス図とシーケンス図を例題として提案手法の適用を試みる．図 13 と図 14 が例題の UML ダイアグラムであり，それぞれクラス図と，カップに注入された紅茶にパウダーを入れるというユースケースをモデル化したシーケンス図の一部である．また，元のクラス図には Tea クラスに root 特性が指定されていたが，UML2.0 ではクラスに root 特性の指定が行えなかったため省略している．

図の UML ダイアグラムの中には，最もよく起こりやすい「関連の定義されていないクラス間でメッセージのやりとりをしてしまう」という不整合が埋め込まれている[9]．例題として挙げた上記の 2 つのダイアグラムには，以下の 2 つの不整合を埋めこんだ．

(1) シーケンス図中の DrinkMixer クラスのオブジェクトと，Tea クラスのオブジェクトとの間でやりとりされているメッセージ addPowder が，その受信オブジェクトのクラスであるクラス Tea 中のオペレーションで定義されていない．

(2) シーケンス図中で，DrinkMixer クラスのオブジェクトと Tea クラスのオブジェクトとの間でメッセージがやりとりされているが，クラス図中でそれらのクラスの間に関連が引かれていない．

4.2 適用結果

例題を用いて main モデル検証環境において検証を行うと，整合性ルール (4)，(6) について違反しているという結果を出力できた．次に，Tea クラスが Drink クラスを継承し，DrinkMixer クラスと Drink クラス間に関連を追加するという修正を行ったところ違反は出力されなかった．したがって提案手法は正しく機能していると考えられる．なお検証時間は約 0.2 秒であった．

4.3 その他の実験結果

上記の例題を用いて，整合性ルール (4)，(6) 以外の 4 つの整合性ルールに対しても同様に実験を行った．全ての整合性ルールについて，違反する設計をした場合は正しく違反を出力するかどうかを確認し，違反を修正した場合は違反を出力しないかどうかを確認した．実験の結果，全ての整合性ルールの検証が成功した．検証時間も適用例題の結果とさほど変化しなかった．

5. 考察と評価

適用実験により，提案手法がクラス図とシーケンス図間の整合性検証に適用可能であることを確認できた．DrinkMaker は文献 1) で述べられている全ての不整合を含んでいるため，他のクラス図，シーケンス図間の整合性検証にも提案手法を適用可能であると考えられる．他の UML ダイアグラム間に対しても同様に整合性を検証可能だと考えられる．UML メタモデルの制限なしに UML モデル検証環境を構築できることは大きな利点である．

本研究と同様に文献 18) も DrinkMaker[10] を用いて整合性検証法の評価を行っている．本研究と文献 18) の実験結果を比較すると，本研究でも同等の検証を行えることがわかる．

短所として，検証者は UML メタモデル構造と OCL

記述文法を深く理解する必要があることがあげられる．UML ダイアグラムのモデル記述環境は UML メタモデルを入力として与えることでツールと EMF が自動で生成するが，OCL 式に関しては UML メタモデルのバージョン毎に検証者が書き直さなければならない可能性がある．制約を与える OCL 式は UML メタモデルのモデル要素にアクセスするので，モデル構造が変化すると OCL 式自体も変更する必要があるからである．また，文献 1) の中には OCL 式を適用できない整合性ルールも存在する．オペレーション内部の情報が必要な制約や，別の OCL 式の内容を参照する必要がある制約は適用が難しい，他にも大量の UML メタモデルにアクセスする必要がある制約を OCL 式で記述するのは現実的ではない．OCL 式で表現するのが適切な制約のみを実装するのが望ましい．

6. あとがき

本論文では，UML メタモデルと UML ダイアグラム間整合性ルールを記述した OCL 式を用いて UML モデル検証環境を構築する手法を提案し，適用実験を行った．提案手法では EMF に基づいて，UML メタモデルと OCL 制約を入力として与えることで Ecore モデルを生成し，Ecore モデルから UML モデル検証環境のソースコードを自動生成する．

今後の課題は，各工程間における同一種類の UML ダイアグラム間整合性検証や，UML ダイアグラム間不整合修正の支援である．また本研究の実験に用いた UML メタモデルはバージョン 2.0 のみであるので他のバージョンに対しても評価実験を行いたい．

参考文献

1) Briand, L. C., Labiche, Y. and O'Sullivan, L.: Impact Analysis and Change Management of UML Models, Technical Report SCE-03-01, Carleton University (2003). http://squall.sce.carleton.ca/pubs/tech_report/TR_SCE-03-01.pdf.
2) Chiorean, D., Pasca, M., Carcu, A., Botiza, C. and Moldovan, S.: *Ensuring UML models consistency using the OCL Environment*, Electr. Notes Theor. Comput. Sci., pp. 99–110 (2004).
3) Eclipse Foundation: Eclipse Modeling Framework. http://www.eclipse.org/modeling/emf/.
4) Eclipse Foundation: Model Development Tools. http://www.eclipse.org/modeling/mdt/.
5) Egyed, A.: In Proc. of ICSE 2007, *Fixing Inconsistencies in UML Design Models*, pp. 292–301 (2007).
6) Elaasar, M. and Briand, L. C.: An Overview of UML Consistency Management, Technical Report SCE-04-18, Carleton University (2004). http://squall.sce.carleton.ca/pubs/tech_report/TR-SCE-04-18.pdf.
7) IBM: Rational Rose. http://www-306.ibm.com/software/rational/.
8) Lange, C., Chaudron, M. R. V. and Muskens, J.: In practice: UML software architecture and design description, *IEEE Software*, Vol. 23, No. 2, pp. 40–46 (2006).
9) Lange, C., Chaudron, M. R. V., Muskens, J., Somers, L. J. and Dortmans, H. M.: In Proc. of Workshop on Consistency Problems in UML-based Software Development, *An Empirical Investigation in Quantifying Inconsistency and Incompleteness of UML Designs*, pp. 26–34 (2003).
10) Ludwik, K. and Miroslaw, S.: In Proc. of Workshop on Consistency Problems in UML-based Software Development, *Inconsistencies in Student Designs*, pp. 9–17 (2003).
11) Object Management Group: . http://www.omg.org/.
12) Object Management Group: Meta Object Facility (MOF) 2.0 Core Specification (2004). http://www.omg.org/docs/ptc/03-10-04.pdf.
13) Object Management Group: MOF QVT Final Adopted Specification (2005). http://www.omg.org/docs/ptc/05-11-01.pdf.
14) Object Management Group: UML 2.0 Diagram Interchange Specification (2005). http://www.omg.org/docs/ptc/05-06-04.pdf.
15) Object Management Group: OCL 2.0 Specification (2006). http://www.omg.org/cgi-bin/apps/doc?formal/06-05-01.pdf.
16) OMG Modeling Specifications: . http://www.omg.org/technology/documents/vault.htm#modeling.
17) Richters, M. and Gogolla, M.: In Proc. of UML2000, *Validating UML models and OCL constraints*, pp. 265–277 (2000).
18) 佐々木亨, 岡野浩三, 楠本真二: 制約指向に基づいた UML モデルの不整合検出・解消手法の提案, 電子情報通信学会論文誌 D, Vol. J90-D, No. 4, pp. 1005–1013 (2007).
19) Shen, W., Lu, Y. and Low, W. L.: In Proc. of Workshop on Consistency Problems in UML-based Software Development, *Extending the UML Metamodel to Support Software Refinement*, pp. 35–42 (2003).