ソフトウェア開発工程におけるUML整合性検証に関する研究

中西, 啓之 / NAKANISHI, Hiroyuki

(発行年 / Year)
2005-03-24
(学位授与年月日 / Date of Granted)
2005-03-24
(学位名 / Degree Name)
修士(工学)
(学位授与機関 / Degree Grantor)
法政大学 (Hosei University)

2004年度　修士論文

ソフトウェア開発工程における UML整合性検証に関する研究

STUDIES ON VERIFYING UML CONSISTENCY IN SOFTWARE DEVELOPMENT PROCESS

指導教官　三浦 孝夫 教授

法政大学大学院工学研究科
電気工学専攻修士課程

03R3229　中西（ナカニシ） 啓之（ヒロユキ）
Hiroyuki NAKANISHI

目 次

第1章　序論

1.1　研究背景

ソフトウェアは大型汎用コンピュータから家庭用電気製品まで至るところに組み込まれ，私たちの生活環境全般でソフトウェアの需要は増加する傾向にある．その様な膨大な要求に応えるために，ソフトウェア開発の現場はかつて織物などの製造方法が手工業から機械工業に変わったように，一定の品質を保った製品を効率的に開発するに足る分析・設計・構築の方法を必要としている．

ソフトウェア開発工程でも他の工業製品と同じように，まず利用者の要求をかなえるために必要な機能を備えたソフトウェアの設計図を書き，それに基づいて複数の開発者で協力して製品を作り上げていく．あいまいであったり複雑なものから余計なものを削ぎ落として単純で分かりやすい形にすることをモデル化と呼ぶ．オブジェクト指向開発では大規模で複雑な問題を扱うために，オブジェクトと呼ばれる概念の構成要素によってソフトウェアのモデル化を行う．それにより，開発者間にソフトウェアに対する共通の理解を与えることができ，口頭や文章をつかったコミュニケーションに比べて意思疎通をスムーズに行うことが可能となる．

現在，オブジェクト指向開発で広く世界中で使われているモデル言語に UML（Unified Modeling Language）がある．UML は 1990 年代オブジェクト指向開発方法論の分野をリードしていた OMT，Booch 法，OOSE の 3 つの手法の提案者達によって生み出されたモデル言語である．要求分析から分析・設計，そして実装に及ぶ開発工程全体を扱うために 9 つの図（diagram）が用意されており，それらを適用することによりシステムを複数の視点で捉えることができる．UML は特定のプログラミング言語や開発プロセスに依存することなく，一般的なプログラミング言語をすべてサポートする態度をとっている．UML はなんらかの開発プロセスで適用されるであろうが，特定の方法論に限定することはなくただモデルの表記法のみを提供するので，どの図をどのように組み合わせて使うのかは開発者しだいである [10].

UML は問題を直接的かつ簡潔に表現できる意味論と記法を提供していて，UML を用いて記述されたモデルをどのように解釈すればよいのかモデルを実際にどうやって記述すればよいのかをすべて定義している．システムの構造や動作を図で

記述することができるが，開発工程中で作成された図が UML が規定している正しい記法によって書かれているかどうかを確認する方法は確立されていない．あるソフトウェア開発工程で UML の図がシステム機能の詳細化や開発グループの共通理解のために使われる時，実際に書かれている図と記法との間の整合性が保障されていないとその図をもとに作られたシステムにも悪影響を及ぼしかねないので，図の整合性を即座に分析して検証する方法の確立を目指す必要がある．特に分析工程ではシステムの構造や動作を正確に捉えることが必要であるので，システムの基本構造やユースケースをもとにオブジェクト間にどのようなメッセージが行き来しているのかを記述している協調図が重要である．これらの理由から，本研究では UML 協調図の整合性を主に扱う．

本研究では知識表現・データベース分野で近年注目を浴びている記述論理 (Description Logics) を整合性検証の論理的な枠組みとして利用する．記述論理は一般的な第 1 階述語論理から変数や関数をなくし次数に上限を設けることで，健全かつ完全で決定可能な推論機構を備えている．記述論理の計算機による自動的な推論機構を実現するシステムとして RACER がある．RACER はフリーソフトとして公開されていて誰でも利用できるようになっているので，本研究では推論を実現する際にこれを利用する．

1.2　主な問題と解決のためのアイデア

本研究では，ソフトウェア開発工程で広く使われている UML の中でも協調図における整合性検証方法の確立とそれを利用した整合性検証システムの実現を目指す．

1.2.1　メッセージを利用した協調図の分析

あるソフトウェア開発工程で開発者によって作成された UML 図の整合性検証をするには，図に記述されている構成要素とその間の関係を詳しく分析することによって，まず図全体がどのようになっているのかを把握することが必要である．実際の開発現場で書かれる図は非常に大量になりそのままつぶさに人手で分析するのは困難であるので，開発者によって作成される協調図を何らかの形で要約することが必要となる．本研究では協調図の中心的要素であるメッセージに注目し，協調図の示す内容の要点を適格に捉えることを試みる．クラス図はクラス・オブジェクトの記述を目的としており，協調図はこれらが表すべき役割を表現している．協調図には個別の振舞いを示すインスタンスレベルでの記述と, クラス図との整合性検証を目的とする仕様レベルでの記述がある．特に前者は, オブジェクトと

それらの関連が作業を完了するためにどのように連絡しあうかを表す．複数のオブジェクトの対話はクラスとそれらの関連（汎化, 集約を含む）上でオブジェクトインスタンスが結びつきあうことで遂行され，クラス図上にリンクがない場合は協調動作は生じない．協調図の静的な特性は対応する UML クラス図を記述論理によって形式化することで得られる．状態図を用いた研究では状態 (state) を概念に, 事象 (event) と動作 (action) を概念と関連に対応させている．事象が実行できる条件 (guard) を記述論理式で与え，その充足性と状態の変化動作を推論することができる．しかし, この方法ではデータベース (クラス図) との関係をモデル化できず, 協調図の表現には適用できない．

本研究では, 動作が実行できる前提条件 P_1, 動作を発火させる発火条件 F, 動作完了後に成り立つ完了条件 P_2 を考え, これを記述論理で表現するアプローチをとる．P_1, P_2 は, この動作が生じるための条件であり, 例えばデータベース状態の変更, 利用者の要求や動作確認, 外部事象の発生などがある．データベース (クラス図) が与えられたときに，これら 3 つの条件の 1 つまたは複数が誤りとみなされる状況を全て調べあげ，記述論理の推論機構により判断することで協調図の整合性検証を行うことを試みる．また，協調図の分析ではソフトウエア各部の動作を検証するための 12 種類の協調動作リンクが考察されていて，精密な静的・動的条件を得ることができる．クラス・オブジェクトのメソッドを属性の動的計算と捉えることで, 属性値，戻り値を使って 7 種類の協調動作リンクに対応した前提条件 P_1 や完了条件 P_2 を記述論理式で機械的に生成できる．

A1. 協調ペア
クラス C から属性 A への関連は役割 $r \preceq C \times A$ で, 関連クラス R へのリンク r をたどる場合は $r \times R \times C$ で定義され, 定義の妥当性 (例えば, $D \models r \preceq R \times C$ などの) 検証を対応させる．これらを P_1 とする．

A2. 変数定義
P_1 として $C \sqcap \exists m.V$

A3. 変数参照
P_1 として $C \sqcap \exists m.C'$ P_2 として $C \sqcap \exists m.V \sqcap \Gamma$

A4. オブジェクト定義
P_2 として $C \sqcap \exists m.C'$

A5. オブジェクト生成
P_2 として $C' \sqcap \exists m^{-}.C$

A6. オブジェクト参照
P_2 は $C \sqcap \exists m.C' \sqcap \Gamma$

A7. オブジェクト削除
P_2 として $C \sqcap \neg \exists m.C'$

1.2.2　UML メタモデルを用いた整合性検証

全体の情報を 1 つの構成要素に関わる条件として要約している場合，そこに直接関わりがない情報や細かい情報というのは抜け落ちる可能性がある．より UML の思想に忠実な整合性検証を行うためには，協調図が有する膨大な量の情報を要約せずに一寸の漏れもないように扱う方法が必要である．

UML の構成要素間の関係や記法を規定しているものは UML メタモデルと呼ばれており，パッケージ (モデル要素のグループ化したもの) 群に体系化されていている．UML 協調図はその中の Collaborations パッケージによって細かく規定されている（Fig.1.1）.

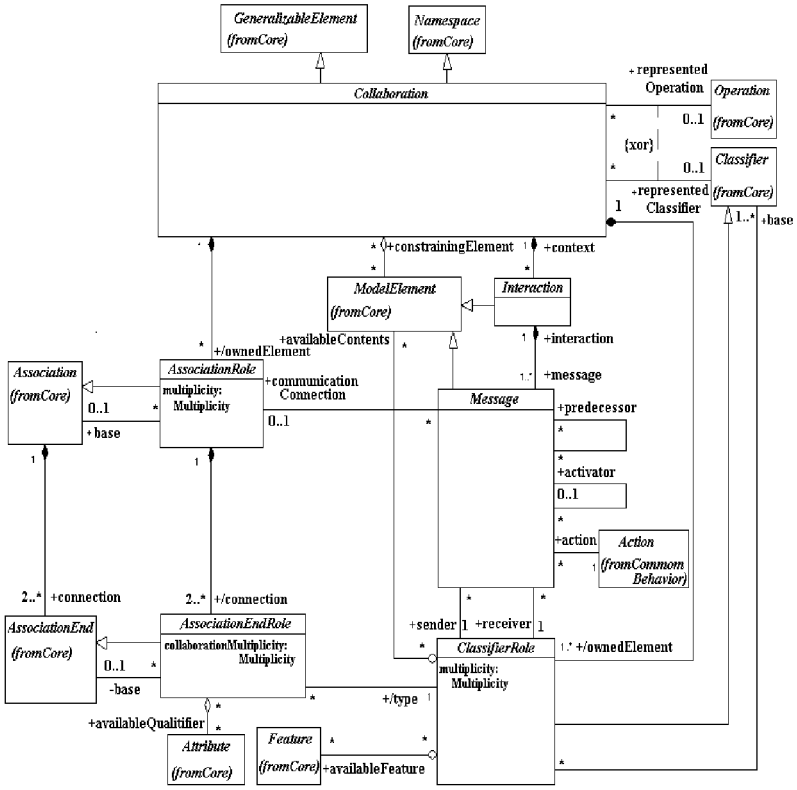

図 1.1: Collaborations パッケージ

Collaborations パッケージは，抽象構文，適格性規則，意味論といったもので構

成されていている．抽象構文はクラス図を用いて各図の構成要素とそれらの関係，適格性規則は OCL（Object Constraint Language：オブジェクト制約言語）を用いて UML モデルで満たすべき不変条件，意味論は主に自然言語を用いて構成要素の意味を定義する．クラス図を記述論理で捉え直す方法は提案されているが，それ以外の 2 つについては記述論理へ変換する方法が確立されていない．1.1 で表されたもの以外に協調図が満たすべき条件（協調図の構成要素やその間の関係や構成要素の解釈方法を形式的に定義するもの）が多数存在することが確認されている．これらの条件を 1 つ 1 つ分析して記述論理に置き換えることで，UML が定める協調図の満たすべき構文と意味論を記述論理式で表現することができる．これに対して開発者が作成した整合性検証の対象となる協調図は，Classifier や ClassifierRole，AssociationRole などの構成要素一つ一つを対応付けて，$ClassifierRole \preceq base.Classifier\sqcap$ のような先の Collaborations パッケージから知ることができる構成要素間の関係に基づいて記述論理式へ捉え直す．この 2 つの記述論理式を合せて記述論理エンジンに入力として与えて推論させる．エラーなどで止まることなく最後まで実行できれば，記述論理で捉えなおされた協調図の満たすべき抽象構文，適格性規則と開発者の書いた協調図との間に矛盾が存在しないことが確認される．また，この手法をシステムとして実現するにあたって，開発者がソフトウェア開発環境で利用する知識以外を必要とせずに協調図の整合性検証を行え，整合性検証の結果も容易に理解できるようにする必要がある．そこで，システムでは記述論理関係の作業は内部で行い利用者からは直接見えない構造にし（Fig.4.1），システムの出力も開発者にわかりやすいように UML の知識だけでわかるメッセージ形式で与える．

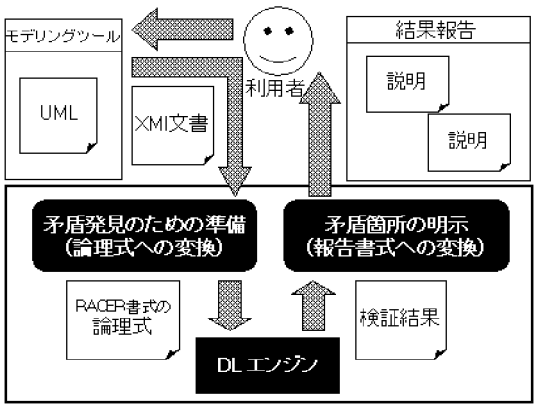

図 1.2: 整合性検証システム

1.3　論文の構成

第 2 章 メッセージの前提条件や完了条件，発火条件を利用した整合性検証方法を示す.

第 3 章 UML メタモデルに注目した協調図の形式化とその整合性検証方法を示す.

第 4 章 3 章の手法をシステムとして実現したものについて述べる.

第 5 章 本研究を通しての結論を示す.

1.4　発表論文

1. 中西啓之，三浦孝夫，塩谷勇：記述論理による UML の表現，夏のデータベースワークショップ（DBWS），2003
2. 中西啓之，三浦孝夫：記述論理による協調図の形式化，電子情報通信学会データ工学ワークショップ（DEWS），2004
3. Nakanishi,H., Miura,T. and Shioya,I.: Reasoning in Collaboration Diagrams by Description Logics, Computer and Their Applications（CATA），2004
4. Nakanishi,H., Miura,T. and Shioya,I.: Formalizing UML Collaborations by using Description Logics，IEEE International Conference on Computational Cybernetics（ICCC），2004
5. 中西啓之，三浦孝夫：記述論理を用いた UML 整合性の検証システムの実現，電子情報通信学会データ工学ワークショップ（DEWS），2005

第2章　記述論理によるUML の表現

UML (Unified Modeling Language) は, 情報システムの設計開発に関する言語であり, 事実上の業界標準的な手法として知られる．しかし, 定義の曖昧さが残るため, モデル変換, 等価性判定, 冗長性の検証, 無矛盾性の検査などの設計開発過程で要する知的な操作を行うことが極めて難しい．記述論理による UML 記述は, クラス図のモデル化と推論に試みられており, 従来直観的に扱われていた表現手法に形式的な枠組みを適用できる．本研究では, 協調図上のオブジェクトの振舞いに対し記述論理による表現を提案する．記述論理式で表現されたクラス図スキーマとの整合性や冗長性を推論することにより, システム仕様機能を検証できる．

2.1　前書き

UML (Unified Modeling Language) は, 情報システムの設計開発に関する言語であり, 事実上の業界標準的な手法として知られる． UML では, 情報システム全容の目的・機能を記述するユースケース, 必要となるオブジェクト情報の構造を表現するクラス図, オブジェクトの振舞いに関する役割を表現する対話図, オブジェクトの状態変化・振舞いを表す状態図やアクティビティ図などから構成される [16, 10]. このうち, 対話図 (協調図および順序図) は, 機能の振舞いを表現しており, ユースケースとともにシステム仕様の機能を示す．またクラス・オブジェクトの役割や実現の詳細を記述しており, 分析過程を通してソースコードやシステムテストの方法を検討できる．UML は記述の方法を定義する言語である．その意味は各種図, OCL (Object Constraint Language) および細部を規定する自然言語で表現されるため, 図の相互の関連性については曖昧さが残る [11]．この結果, モデル変換, 等価性判定, 冗長性の検証, 無矛盾性の検査などの設計開発過程で要する知的な操作を行うことが極めて難しい.

記述論理 (Description Logics) は, 健全かつ完全でしかも決定可能な推論機構を備えているため，知識表現・データベース分野で近年注目を浴びている．記述論理は構造化された情報を扱い, 変数や関数のない, 次数に上限を設けた第 1 階述語論理の部分クラスである．論理プログラムと異なり，選言 ($\vee$) や否定 ($\neg$) を自然に扱うことができる．データベースの立場からは，設計，管理，情報検索，情報

統合などで推論機構を導入し，同値性，無矛盾性，冗長性，充足可能性の検査・検証ができる．高度なモデル化機能により，実体関連データモデルやオブジェクト指向データモデル, あるいは UML クラス図では基本概念が対応しており，スキーマや一貫性制約の記述などに応用できる [5, 8, 9]．また様相論理機構を導入できるため，時制モデルへの応用が提案されている [2]．特に, データベーススキーマは $\mathcal{ALCQI}$ と呼ぶ記述論理の部分クラスで表すことができ, スキーマ・クラスの充足可能性・等価性・冗長性などの設計上重要な問題に対して, 決定可能な推論機構を利用できる．また，濃度制約の充足性判定問題は, 同等の記述論理式を整数線形計画法 (線形プログラミング）に帰着させ, 多項式時間で決定可能である．

記述論理による UML 記述は, クラス図のモデル化と推論に試みられており, 従来直観的に扱われていた表現手法に形式的な枠組みを与える [3, 4, 6]．反面, 論理系はスキーマやオブジェクト表現には効果的であるが, 状態の変化を捉えることが容易ではない．このため, 状態図の形式化やオブジェクト協調などの振舞いに対して適用することは容易でない [14].

本稿では, 協調図上のオブジェクトの振舞いに対し記述論理による表現を提案する．協調動作 (operation) はいくつかのメッセージ指令から構成され, 各指令を前提条件や完了条件 P_1, P_2 および発火条件 F で表す．記述論理式で表現されたスキーマと指令条件の整合性や冗長性を推論することにより, システム仕様機能をデータベースとの関連で検証できる．

次章では UML 協調図の果たす役割を述べ、3 章では記述論理の特徴を要約する．4,5 章では $\mathcal{ALCQI}$ によるクラス図の記述手法を示し, これを用いて協調動作を記述論理で表現する手法を示す.

2.2　UML と協調図

UML はソフトウエアシステムの仕様の策定, 構成, 視覚化, 文書化などを支援するための言語体系であり，多種類の図 (diagram) から構成される．とくに協調図 (Collaboration Diagram) はクラス・オブジェクトの果たす役割の相互関連性を表現している．

UML が想定している協調 (Collaboration) では, (OMG 1.3 によれば，) オブジェクトを想定したスロット, オブジェクト間の関連を表すリンク, および協調を記述するスロット（role という）がある [10]．クラス図はクラス・オブジェクトの記述を目的としており静的である．協調図における振舞いは，クラス・オブジェクトが果たすべき役割 (role) を表し動的な特性を示す．文献 [11] では，ClassifierRole によりオブジェクトが果たす役割を, AssociationRole により関連が他の役割と共同して果たす役割を示すとしている．リンク (Link) は関連 (Association) のイン

スタンスであり, クラス・オブジェクト同士がどのように結びつくかを表現する. 対話 (Interaction) とは振舞いの記述であり通信の半順序的列を言う. メッセージ (message) は 通信内容であり, 通常 sender/receiver オブジェクト, 指令 (変換, 質問, パラメタなど) が含れる. メソッドとはこれらの実現である.

協調図には個別の振舞いを示すインスタンスレベルでの記述と, クラス図との整合性検証を目的とする仕様レベルでの記述がある. 特に前者は, オブジェクトとそれらの関連が作業を完了するためにどのように連絡しあうかを表す. オブジェクトとその詳細,(オブジェクトに対する) 予め決められた操作, 他のクラスから呼び出された操作, 動作特性 (メッセージ順序) などが代表的である. 複数のオブジェクトの対話はクラスとそれらの関連（汎化, 集約を含む）上で, オブジェクトインスタンスが結びつきあうことで遂行され, クラス図上にリンクがない場合は, 協調動作は生じない.

協調図の分析は, 従来から研究されているデータフロー手法のそれと類似しているため, ソフトウエア各部の動作を検証するための精密な静的・動的条件を得ることができる. 検証のため次の 12 種類の協調動作リンクが考察されている [1].

A1. 協調ペア (クラス図で ClassifierRole や AssociationRole が定義されている)
A2. 変数定義 (局所変数を特定する)
A3. 変数参照 (局所変数を参照する)
A4. オブジェクト定義 (対応するオブジェクトの特定) A5. オブジェクト生成
A6. オブジェクト参照（メソッド呼び出し）
A7. オブジェクト削除
B1. 変数定義 (局所変数を定義・使用)
B2. オブジェクト定義 (オブジェクトを定義・使用)
B3. オブジェクト生成使用 (オブジェクトを生成・使用する)
B4. オブジェクト生成・削除 (オブジェクトを生成・削除する)
C1. メッセージ順序 (順序を表現)

A1,C1 は協調図リンクと構成が正しい定義であることを, A2,A3 はメソッドの局所変数に対する動作内容を, A4 は他のクラスのオブジェクト参照を示す. A5,A7 はデータベースの状態変化を生ぜしめる内容であり,A6 はメソッド呼び出しである. また B1,B2,B3,B4 は一時オブジェクト・局所変数に対するものであり, 新規オブジェクトやメソッドの解析に用いる.

例題 1 ここで扱うのは自動販売機の例である。

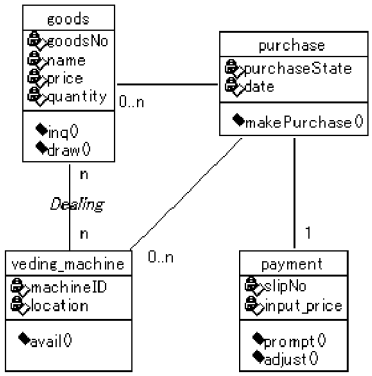

図 2.1: 「自動販売機」クラス図

ここでは 3 つのクラス vending-machine，goods，payment と 1 つの関連クラス purchase を考える．クラス vending-machine は自動販売機を表わし，該当地域にある販売機をオブジェクトと考える．管理のための ID (MachineID) と設置位置 (Location) が属性として与えられている．このオブジェクトの生成・削除・修正は無いと仮定する．ただ，自動販売機は在庫確認 (Avail()) を行うメソッドを有し，goods クラスと連携する．

自動販売機で扱う商品種別 (飲み物を想定) に対して goods クラスが対応する．個々の商品の ID (GoodsID), 名称 (Name)，価格 (Price)，在庫量 (Quantity) が属性として与えられ、販売されるつど在庫引き当てがなされる．在庫を検索する inq()，実際の引き当てを行う draw() メソッドを仮定する．

現金支払機の動作では payment クラスでオブジェクトが生成されるとし，個々の支払いをオブジェクトとみなす．ここでは受け取り書 ID (SlipID), 支払い日時 (Date)，支払い金額 (Amount) を属性値とする．現金支払機では支払い督促 (Prompt() を行い，つり銭処理に自動的に応答するとみなす．また，必要なら返金処理 (Adjust()) も行う．

個別の購入に関しては purchase 関連クラスのオブジェクトとで表現する．自動販売機，商品，支払いの 3 つのオブジェクトが関連を構成するとき成立する．購入処理 (MakePurchase()) では関連生成に必要な処理を行う．キャンセルは PurchaseState を変更し，履歴は保持する．

このクラス図では，各購入ごとに自動販売機，商品，支払いの各オブジェクトがただひとつ対応しており，またどの支払いも必ずちょうどひとつの購入取引によるものでなければならない．

アクタである顧客が希望商品を自動販売機上で指示すると，在庫を検索し，購入操作を開始する．この購入を顧客が確認し，現金を投入すれば支払い処理が完了する．キャンセルされれば，必要なら返金し，支払い金額をゼロとして検索し

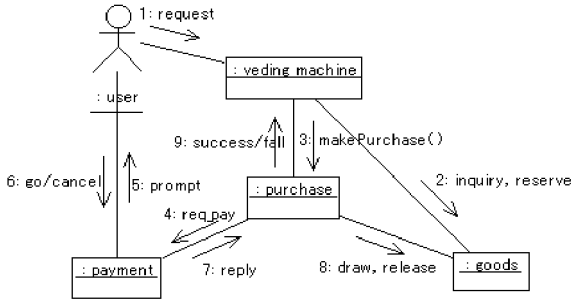

図 2.2: 協調図「自動販売機の使用」

た商品を解除する．いずれにせよ，購入情報 purchase は生成され，支払い情報も生成される．

ここで各動作を詳細に検討すると次のようになる．

1. request : vending-machine オブジェクト定義
2. inquery,reserve : goods オブジェクト定義・参照
3. MakePurchase : purchase オブジェクト生成
4. req-pay : payment オブジェクト生成
5. prompt : payment 変数参照
6. go/cancel : payment オブジェクト参照
7. reply : purchase オブジェクト参照
8. draw/release : goods オブジェクト参照
9. success/fail : purchase オブジェクト参照

本稿では,(クラス図を前提にして) クラス・オブジェクト, 関連とこれらの属性, クラスメソッドとその局所変数に対する分析を行う．動的な検証により，オブジェクト動作, 対話や実行結果の信頼性を高め誤りの原因を突き止めることが容易になる．動作が生じる条件や、各リンク上で前提・完了条件を分析し, メッセージ順序を監視することにより, これらの条件を記述することができる．

2.3　記述論理 (Description Logics)

記述論理は構造化された情報を扱う論理系であり, 変数や関数のない次数に上限を設けた述語論理の部分クラスである．第 1 階述語論理と違って, 充足性判定問題が決定可能であり, 主要な部分クラスでは多項式時間で処理できるという特徴を有する．記述論理では概念 (Concepts) と役割 (Role) から構成される．前者はオブジェクトクラスを意味しており，後者はオブジェクトインスタンスの属性 (2 項関連) を

意味する．基本概念 (primitive concept) によって記号が与えられ，$\sqcap$,$\sqcup$ などの構成子を用いて式 (expression) が定義される．限量作用子 $\forall,\exists$ は役割を介して定義される．基本概念 C, C' 上の役割 R に対して，$\forall R.C'$ とは C のオブジェクト x の任意の R 属性値 y に対して $y \in C'$ となることをあらわす．例えば，$Person$(人間), $Doctor$(医者) 概念と，$CHILD$(子供) 役割に対して，$Person \sqcap \forall CHILD.Doctor$ により，その子供すべてが医者である人物を表現している． $\exists R$ により何かの R 属性が存在することを表す．$C \preceq D$ により包摂関係を表す，すなわちすべての C オブジェクトは D オブジェクトでもある． 例えば $Parent$(親) 概念とは $Person$ で $CHILD$ 属性を有するオブジェクトであり，$Parent \preceq Person \sqcap \exists CHILD$ と表すことができる．

上述のような記述論理を $\mathcal{AL}$ クラスと呼ぶ．$\mathcal{L}$ スキーマとは，$\mathcal{C} \preceq \mathcal{E}$ または $\mathcal{C} \doteq \mathcal{E}$ の形の式の集合を言う．前者は基本概念の仕様とよばれ，$\mathcal{E}$ で指定された式を定義域とすることを意味する．後者は定義概念と呼ばれ，$\mathcal{C}$ 概念のオブジェクトであるための必要十分条件を表している．すべてのスキーマ内の式を充足する解釈をスキーマのモデルという．このとき $\mathcal{L}$ は 無矛盾 あるいは 充足可能であるという．$SAT(\mathcal{L})$ により $\mathcal{L}$ のモデル集合を表わす．式 $\mathcal{E}$ がスキーマと無矛盾とは，スキーマと $\mathcal{E}$ が共に空でないモデルを有するときを言う． $\mathcal{AL}$ では包摂関係を高速に推論することができる．実際多項式時間で決定可能 (decidable) であることが知られる [9].

記述論理には構成子の種類に応じて多数のクラスが存在する．図 2.3 にこれを示す.

構成子名		構文	解釈
概念名		C	$C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
トップ，ボトム		$\top, \bot$	$\Delta^{\mathcal{I}}, \phi$
原子否定		$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
連言		$E_1 \sqcap E_2$	$E_1^{\mathcal{I}} \sqcap E_2^{\mathcal{I}}$
全称限量		$\forall R.E$	$\{o \mid \forall o' : (o,o') \in R^{\mathcal{I}} \rightarrow o' \in E^{\mathcal{I}}\}$
非限定存在限量		$\exists R$	$\{o \mid \exists o' : (o,o') \in R^{\mathcal{I}}\}$
存在限量	$\mathcal{E}$	$\exists R.E$	$\{o \mid \exists o' : (o,o') \in R^{\mathcal{I}} \wedge o' \in E^{\mathcal{I}}\}$
選言	$\mathcal{U}$	$E_1 \sqcup E_2$	$E_1^{\mathcal{I}} \cup E_2^{\mathcal{I}}$
一般否定	$\mathcal{C}$	$\neg E$	$\Delta^{\mathcal{I}} \setminus E^{\mathcal{I}}$
数値制約	$\mathcal{N}$	$\exists^{\geq m} R$	$\{o \mid \sharp\{o' \mid (o,o') \in R^{\mathcal{I}} \geq m\}$
		$\exists^{\leq n} R$	$\{o \mid \sharp\{o' \mid (o,o') \in R^{\mathcal{I}} \leq n\}$
数値限量制約	$\mathcal{Q}$	$\exists^{\geq m} R$	$\{o \mid \sharp\{o' \mid (o,o') \in R^{\mathcal{I}} \wedge o' \in E^{\mathcal{I}} \geq m\}$
		$\exists^{\leq n} R$	$\{o \mid \sharp\{o' \mid (o,o') \in R^{\mathcal{I}} \wedge o' \in E^{\mathcal{I}} \leq n\}$
有限整合	$\mathcal{W}$	$wf(R)$	$\{o_0 \mid \forall o_0, o_1, ...$(無限列)$\exists i (o_i, o_{i+1}) \notin R^{\mathcal{I}}\}$
ルール値写像	$\mathcal{V}$	$(R_1 \sqsubseteq R_2)$	$\{o \mid \{o' \mid (o,o') \in R_1^{\mathcal{I}} \subseteq \{o' \mid (o,o') \in R_2^{\mathcal{I}}\}\}$
ルール名		P	$P^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
逆	$\mathcal{I}$	R^-	$\{(o,o') \mid (o',o) \in R^{\mathcal{I}}\}$
選言	$\mathcal{R}$	$R_1 \cup R_2$	$R_1^{\mathcal{I}} \cup R_2^{\mathcal{I}}$
結合	$\mathcal{R}$	$R_1 \circ R_2$	$R_1^{\mathcal{I}} \circ R_2^{\mathcal{I}}$
反射推移閉包	$\mathcal{R}$	R^*	$(R^{\mathcal{I}})^*$
同一	$\mathcal{R}$	$id(E)$	$\{(o,o) \mid o \in E^{\mathcal{I}}\}$
差	$\mathcal{D}$	$R_1 \setminus R_2$	$R_1^{\mathcal{I}} \setminus R_2^{\mathcal{I}}$

このうち特にデータモデリングでは $\mathcal{ALCQI}$ が適合する [8]. $\mathcal{ALCQI}$ では, 概念は, 基本記号 A または $\neg C$, $C \sqcap C'$, $C \sqcup C'$, $\forall R.C$, $\exists R.C$, $\exists^{\geq n} R.C$, $\exists^{\leq n} R.C$ の形式であり, 役割は P, P^- のいずれかである (A, P は基本概念･基本役割, C, C', R は概念表現および役割表現). また $C \Rightarrow C'$ を $\neg C \sqcup C'$ と, $C \equiv C'$ を $(C \Rightarrow C') \sqcap (C' \Rightarrow C)$ と表す.

例えば, 多項述語を許したモデル (実体関連モデルや UML クラス図) $\mathcal{DLR}$ は $\mathcal{ALCQI}$ の部分クラスであり, オブジェクト指向モデルは $\mathcal{ALCQ}$ で表せる. これらに応じて決定性判定問題の計算量が知られている.

クラス	充足性判定
$\mathcal{AL}$	P
$\mathcal{ALE}, \mathcal{ALR}, \mathcal{ALER}, \mathcal{ALU}, \mathcal{ALUN}$	NP
$\mathcal{ALC}$	$PSPACE$
$\mathcal{ALCQI}$	$EXPTIME$

データベースで扱うデータは有限である. これに対し論理系ではこの制限は通常ない. 充足可能ならば必ず有限モデルが存在するとき, その論理系は"有限モデル推論"であるという. すなわち, 充足性は有限モデルに限ってよく, 有限モデルがなければ本来充足不能であると断定できる. しかし $\mathcal{ALCQI}$ は有限モデル推論性を満たさない [7]. このため, 以下では (1) スキーマでは基本概念だけが高々 1 度しか左辺に表れず, (2) 概念を再帰的に定義せず, しかも (3) 有限性条件 (well-founded) $\mathcal{W}$ を仮定した $\mathcal{ALCQIW}$ に限定して議論を行う.

例題 2 記述論理による質問例を示す.
(1) 複数の販売機で扱われている商品：$goods \sqcap \exists^{\geq 2} Dealing^-$
(2) 自動販売機の扱う 100 円の商品：$goods \sqcap Price.\{100\} \sqcap (\exists Dealing^-.vendingmachine)$
(3) どの商品も 100 円である自動販売機：$vendingmachine \sqcap (\forall Dealing.(goods \sqcap Price.\{100\}))$
(4) どの自動販売機でも扱われていない商品：$goods \sqcap (\neg \exists Dealing^-)$

2.4　クラス図の記述論理による表現

記述論理による UML クラス図の形式化は, メタモデルやデータフロー等と異なり [13, 15], 決定可能な推論機構を利用することでクラス図の充足可能性・等価性・冗長性などの設計問題に対応することができる. この章では記述論理に基づく知識ベースによって UML が表現され, クラス図の設計・開発に有用であることを示す [3, 4, 8].

クラス C は概念に,C の属性 A (領域 C') については役割 R で $C \preceq \forall R.C', C' \preceq \forall R^{-}.C$ と解釈する．以下ではこれを $R \preceq C \times C'$ と表す．C が属性 $A_1, .., A_n$ を持ち，各属性 A_i に対応する役割 R_i は $R_i \preceq C \times C_i$ を満足するとき $C \preceq \forall R_1.C_1 \sqcap ... \sqcap \forall R_n.C_n$ なるスキーマ条件が対応する．クラス図では,実体関連モデルと同様に多項関連を扱う (n 項関連を導入し $\mathcal{DLR}$ クラスが提案されている) ため, 2 項関連のみを扱う記述論理ではモデルの変換が必要となる．クラス $C_1, .., C_n$ 上の関連クラス R は R を概念に対応させ, R と C_i とのリンク r_i を役割と見て $R \preceq \forall r_1.C_1 \sqcap ... \sqcap \forall r_n.C_n$ および $C_i \preceq \forall r_i^{-}.R$ と n 個の 2 項関連に分解する．濃度制約は (C と R 上の) r 上の数値限量と考え $(m..n)$ 対応を $C \preceq \exists^{\geq m} r^{-}$ および $C \preceq \exists^{\leq n} r^{-}$ と解釈する．$m = 0$ あるいは $n = \infty$ のときはこの制約を除外する．

クラス階層 (または汎化 generalization) C ISA C' あるいは包含制約 $C \subseteq C'$ は $C \preceq C'$ で表す．これは C, C' がブール表現されていても同様である。また排他制約 $C||C'$ ($C \cap C' = \phi$ を意味する) は $C \preceq \neg C'$ あるいは $C \sqcap C' \preceq \bot$ と表される．

UML クラス図では集約 (Aggregation) によって, クラス C のオブジェクトが (他のクラス C' の) 複数のオブジェクトから構成されることを表現できる．記述論理では 集約を表わす役割 Ag によって $Ag \preceq C \times C'$ と表す．C と C' の濃度制約 $(m..n)$ (どの $e \in C$ も m 個以上 n 個以下の C' オブジェクトで構成される) も先と同様に $C \preceq (\exists^{\geq m} A.C') \sqcap (\exists^{\leq n} A.C')$ と表せる．C' と C の濃度制約 (どの $e \in C'$ も m 個以上 n 個以下の C オブジェクトにしか参加しない) も逆役割 A^{-} を用いて同様に表わせる．

例題 3 例題 1 で示されたクラス図を記述論理スキーマで表現する．クラス・関連クラスのスキーマについては次のようになる．

(1) $goods \preceq \forall GoodsNo.INTEGER \sqcap \forall Name.STRING \sqcap \forall Price.INTEGER \sqcap \forall Quantity.INTEGER$
(2) $vendingmachine \preceq \forall MachineID.INTEGER \sqcap \forall Location.STRING$
(3) $vendingmachine \preceq Dealing.goods$
(4) $purchase \preceq purchase1.vendingmachine \sqcap purchase2.goods \sqcap purchase3.payment$

クラス goods，vending-machine は数値 (INTEGER) や文字列値 (STRING) 属性を定義域とし，さらに vending-machine は関連 $Dealing$ を介した goods のみを扱う．

関連クラス purchase は 3 項関連であるから独立したクラスとみなされ，3 つの 2 項リンク purchase1, purchase2, purchase3 でつながれる．

一方，このスキーマに課せられた制約は (1) 各購入 (purchase) で自動販売機, 商品，支払いの各オブジェクトがただひとつ対応すること，および (2) どの支払いも必ずちょうどひとつの購入取引によることである．この 2 つの濃度制約はそれぞれ次のように表現される．

(1) $purchase \preceq \exists^{=1} purchase1 \sqcap \exists^{=1} purchase2 \sqcap \exists^{=1} purchase3$
(2) $payment \preceq \exists^{=1} purchase3^{-}.purchase$

2.5　協調図の記述論理による表現

記述論理を用いた振舞いに関し, 状態図を用いた研究 [14] では $\mathcal{DLR}$ を用いて状態 (state) を概念に, 事象 (event) と動作 (action) を概念と関連に対応させている. 事象が実行できる条件 (guard) を記述論理式で与え, その充足性と状態の変化動作を推論することができる. しかし, この方法ではデータベース (クラス図) との関係をモデル化できず, 協調図の表現には適用できない. 本章では, クラス・オブジェクトの振舞いとメソッド内の局所変数あるいは一時オブジェクトの振舞いを直接扱う.

スキーマ D の解釈であるデータベースが, 具体的にどのような状態に遷移するかについて協調図では何も主張しない. このため記述論理の充足性は, 何かのモデルの存在を保証するだけであり, そのモデルの計算方法を示すものではない. 以下では 3 つの条件 $< P_1, P_2, F >$ によって対話のモデル化を行う. ここで, 前提条件 P_1 はスキーマ D のインスタンス d で動作が実行できるための制約条件を示す. 動作を発火させるためにはあるデータベースインスタンスで発火条件 F が充足されねばならない. 動作完了後に成り立つ完了条件 P_2 を考え, これを記述論理で表現する. 前提条件・完了条件は記述論理式で表されるもので, 一貫性制約と同様, 対話状況が生じるときにはどのようなデータベースに対しても成立すべき状況をいう. これに対し, 発火条件はあるひとつのデータベースで真となる状況にすぎず, $C(a)$ などのように個別オブジェクトに関する記述として与えられる.

どの動作もデータベースを変更する可能性があり, D, P_2 は充足可能であり, d が D, P_1 および F を充足していても, P_2 を充足するとは限らない. 協調図は状態遷移の詳細を表さず, その正当性は状態図などを用いて保障されねばならない. この動作の妥当性は次のように表せる：

もし $SAT(D, P_1, F) \neq \phi$ であれば $SAT(D, P_2) \neq \phi$

データベースのスキーマ D が与えられたとき, P_1, F に対応して, 記述論理の推論機構により判断すべき状況は D, P_1, F が充足可能かどうか, つまり $SAT(D, P_1, F) \neq \phi$ である. 記述論理では, 簡単にこれを検証することができる:

$$D, P_1, F \not\models \bot$$

検証できないときは P_1, F は D と整合せず誤りである.

更に, P_1, F および D を検査することにより妥当性や冗長性を検査できる.

(1) $D \models P_1$ ： 前提条件が常に成り立ち，P_1 は冗長
(2) $D \models P_1 \equiv \bot$ ： 前提条件は成り立たず，P_1 は誤り
(3) $D \models F \equiv \bot$ ： D からは決して発火せず，F は誤り
(4) $D, P_1 \models F$ ： 前提条件が成り立てば必ず発火し，F は冗長
(5) $D, P_1 \models F \equiv \bot$ ： 前提条件が成り立っても決して発火せず，F は誤り
(6) $D \models P_1 \sqcap F$ ： 常に発火する，P_1, F ともに冗長

完了条件 P_2 に関して，記述論理の推論機構により $SAT(D, P_2) \neq \phi$，つまり $D \not\models P_2 \equiv \bot$ であることを検査すべきである．また，つぎのような分析を行うことができる．

(1) $D \models P_2$ ： 完了条件が常に成り立つ．P_2 が冗長
(2) $D \models P_2 \equiv \bot$ ： 完了条件が成り立たず，R_2 は誤り

この (1),(2) は協調図上で何らかの誤りとみなされる．

データベースインスタンス d が変更されないとき，d が D, P_1, F を充足するならば P_2 を満たさねばならない，つまり $SAT(D, P_1, F) \subseteq SAT(D, P_2)$ が成り立つである．記述論理では $D, P_1, F \models P_2$ を意味するため，これが成り立たなければ P_2 は誤りとなる．

一般に，$d \in SAT(D, P_1, F)$ であっても $d \in SAT(D, P_2)$ であるとは限らない．ある動作が d を変更し新たな $d' \in SAT(D, P_2)$ になっても，どのようにして d' を得るのかを指定していない．知りえるのは $SAT(D, P_2)$ が空でないことだけであり，協調図は個々のオブジェクトがどのように変化するかを示すものではない．

2 章で示したように, 協調図では 12 種類の (C から C' への) リンクが存在しうる．クラス・オブジェクトのメソッド $m : C \to C'$ を属性の動的計算と考え, m を属性名, その戻り値を C' の要素とみなす．記述論理ではこれらに対応して完了条件 P_2 を次のように表現することができる．

A1. 協調ペア
クラス C から属性 A への関連は役割 $r \preceq C \times A$ で, 関連クラス R へのリンク r をたどる場合は $r \times R \times C$ で定義され, 定義の妥当性 (例えば, $D \models r \preceq R \times C$ などの) 検証を対応させる．これらを P_1 とする．

A2. 変数定義
変数は宣言・特定されねばならず，メソッド属性 $m :\preceq C \times V$ は P_1 として $C \sqcap \exists m.V$ を充足する必要がある．

A3. 変数参照
変数定義と同様に，$m : C \to V$ とする．変数参照・修正後に条件 Γ が成り立つとき，完了条件 P_2 は $C \sqcap \exists m.V \sqcap \Gamma$ として与えられる．

A4. オブジェクト定義
P_2 として $C \sqcap \exists m.C'$ が対応する．協調ペアを介して，戻り値がオブジェクト となるメソッド $m : C \to C'$ を用いる．

A5. オブジェクト生成
P_2 として $C' \sqcap \exists m^{-}.C$ を与える．クラス C' のオブジェクトを生成するメソッド $m : C \to C'$ と考える．

A6. オブジェクト参照
P_2 は $C \sqcap \exists m.C' \sqcap \Gamma$ で与えられる．メソッド $m \quad : C \to C'$ が参照するオブジェクトは式 Γ を満たす．

A7. オブジェクト削除
P_2 は $C \sqcap \neg\exists m.C'$ で与えられる．オブジェクト の削除メソッド $m : C \to C'$ を用いると考える．

これらの完了条件 P_2 はデータベーススキーマ記述，P_1, F とともに推論することにより，冗長性あるいは充足不能性を検査できる．なお B/C 規則は局所変数の操作あるいは一時オブジェクトの操作に関するものであり，プログラム的には興味深いがデータベースの整合性の観点を有さないため, 本研究では考察しない.

例題 4 想定している個々の動作を分析し，前提条件・完了条件の妥当性を検証する.

1. `request : vending-machine` オブジェクト定義
ここでは顧客 $(\top)$ が自動販売機を特定すると考えられる．顧客と自動販売機の関係 R (ここでは目前に立つ) を仮定すれば完了条件は $\top \sqcap \exists R.vendingmachine$ となる．スキーマから充足可能である.

2. `inquery,reserve : goods` オブジェクト定義・参照
`vending-machine` が `goods` に対するオブジェクト定義・参照動作を行う．ここでは予約操作が含まれ，完了条件は次のようになる：$vendingmachine \sqcap (\exists Dealing^{-}.\ (goods \sqcap Quantity \geq 0))$

3. `MakePurchase : purchase` オブジェクト生成
`vending-machine` が `purchase` オブジェクトを生成する．完了条件は $purchase \sqcap \exists purchase1.vendingmachine$ で与えられる.

4. req-pay : payment オブジェクト生成
   purchase が payment オブジェクトを生成する．上と同様に完了条件は $payment \sqcap \exists purchase3^{-}.purchase$ で与えられる．

5. prompt : payment 変数参照
   顧客に対する payment オブジェクトの問いかけである．支払額 InputPrice を通知し，実際に支払われた金額の過不足が処理される．$payment \sqcap \exists InputPrice \sqcap (InputPrice = \text{顧客支払額})$ が完了条件となる．

6. go/cancel : payment オブジェクト参照
   顧客 $(\top)$ が確認あるいはキャンセルを指示する．結果は局所変数に格納されるとすれば，データベースの完了条件は $\top \sqcap \exists R.payment$ である．

7. reply : purchase オブジェクト参照
   payment は purchase に対して処理結果を戻す．これは PurchaseState の属性値となる．$payment \sqcap \exists purchase3^{-}.(purchase \sqcap Purchase.\{YES/No\})$ が完了条件として得られる．

8. draw/release : goods オブジェクト参照
   正常であれば，予約された goods を引き渡すが、そうでなければ予約をキャンセルする必要がある．$purchase \sqcap \exists purchase2.(goods \sqcap \exists draw)$ または $purchase \sqcap \exists purchase2.(goods \sqcap \exists release)$ が完了条件となる．

9. success/fail : purchase オブジェクト参照
   同様に $purchase \sqcap \exists purchase1.(vendingmachine \sqcap \exists MSG)$ が完了条件になる．ただし MSG によって完了メッセージを表示する．

これらの式はいずれもスキーマ条件と整合するものであり、充足可能である．

上記 (8) において，release するとき purchase オブジェクトを削除する動作ならば，オブジェクト削除操作によって $payment \sqcap \neg \exists purchase3^{-}.purchase$ が完了条件となる．しかし，濃度制約条件 $payment \sqcap \exists^{=1} purchase3^{-}.purchase$ に矛盾し，完了条件が誤っている．

この結果は実行の結果 (振舞いの結果) が整合していることを保障するものでなく，充足可能性の検査にすぎないことに注意したい．スキーマ D の解釈であるデータベースが，具体的にどのような状態に遷移するかについて協調図では何も主張しない．記述論理の充足性は，何かのモデルの存在を保証するだけであり，そのモデルの計算方法を示すものではない．

例えば，A5,A6,A7 はメソッド内で他のオブジェクトの状態を変更するため, 実行の結果は D の解釈を変更するという意味で動的である．上記の範囲で局所的に充足していても，新たな解釈は後続する動作で D のモデルではなくなる可能性があり, この動作単独で判断することは合理性がない．これまでデータベース分野では, 解釈に不整合が生じた時点でこれを解消すべく他の部分への変更動作を伝播させるか, あるいはトランザクション理論に従って整合性検査をある時点まで延期する手法がとられている．後者の場合, メッセージ順序を表わすリンクはトランザクション制御にかかわる情報を提供する．

これらの考察は, 複数の連続する動作に対して適応できる．実際, $< P_1, P_2, F >$ と後続する $< P_1', P_2', F' >$ から，$< P_1, P_2 \sqcap P_1', F >$ および $< P_2 \sqcap P_1', P_2', F' >$ を構成し, 同様の考察を加えればよい．

2.6　結論

UML 記述のうちクラス図と協調図に記述論理を用いて形式的な枠組みを与えた．本研究による表現を用いれば, データベース (クラス図スキーマ) との整合性や冗長性を推論することにより, 協調図の整合性を検証することができる．オブジェクト状態の変更を伴う場合であっても, 協調図とクラス図から充足可能性の検証ができる．しかし，協調図だけでは, 充足性 (あるいは充足不能性) を判定できても，実際にどのようなモデルに変更すればよいかという分析はできない．このためには, 状態図やアクティビティ図などの，個々のオブジェクトの動作を追跡する推論機構が必要となる．

第3章　記述論理による協調図の形式化

UML(統一モデリング言語) は, オブジェクト指向分析や設計で広く利用されているが, 整合性を検査するための方法が確立されていない．他方，UML 協調図では属性値や状態の変化ではなく, オブジェクト間の関係と通信を宣言的に表現しているため分析には適している．本研究では協調図を記述論理を用いて形式化し, 記述論理の推論機構を用いて推論を行う.

3.1　前書き

対象問題について深い理解を得るために, その対象をある目的または観点から眺め, 本質的な部分を捉えるためにモデリング言語と呼ばれるものが用いられる．中でも,UML は, オブジェクト指向分析や設計等の分野で広く利用されているが, 整合性を検査するための方法が確立されていない．ただし，UML の中でも協調図においては, オブジェクト間の関係と通信を宣言的に表現しているため, 手続き的な順序を考慮する必要がなく, 論理との相性がよい．そこで, 論理を用いた UML 協調図の形式化を考えることにより, 協調図の整合性を検査する方法の確立を目指す.

関連する研究として, 仕様記述言語 Object-z を用いた協調図の形式化が行われ [17], UML の矛盾やあいまい性を指摘することに成功しているが, その分析には人の手が必要であり, 矛盾やあいまい性を自動的に検出する仕組みはない.

記述論理は一階述語論理のサブクラスであり, 協調図を論理で捉えることができる．また, その推論エンジンは, 計算機により自動的に包摂関係を推論することができる．本研究では, これらを用いることにより, 協調図の形式化の手法, 協調図の整合性の自動的な検査方法を示す.

2 章では UML とそのメタモデルについて述べ,3 章では記述論理とその推論エンジンについて述べる．4 章では UML 協調図の形式化について述べ,5 章で協調図の例を用いて推論を行い, その結果について考察し, 6 章で結びとする.

3.2　UML とメタモデル

3.2.1　UML の形式化

UML メタモデルは，UML を用いて UML 自身を記述するものである．表現するモデルの構造がどうなっているのか, モデルがどのような意味を持っていると解釈すべきかを示している．また,UML メタモデルは抽象的なモデルであり, 宣言的な意味論を中心としていてるので, この抽象的なメタモデルを用いた実装は, その意味論に従わなければならない.

UML メタモデルは, 抽象的なパッケージ (モデル要素のグループ化したもの) 群に体系化されていて, 静的構造を規定する Foundation(基盤), モデルの動的な振る舞いを規定する言語の上位構造である BehavioralElements(振る舞い要素), モデル要素がどのようにパッケージやサブシステムの中で組織されるかを規定する ModelManagement(モデル管理) 等の最上位パッケージに分解される [10]．各パッケージは, 抽象構文, 適格性規則, 意味論といったもので構成されている．抽象構文は構成要素とそれらの関係を定義するメタクラスを示す図で構成され, 関係の多重度用件からなる適格性規則を示している．適格性規則は,UML モデルで満たすべき不変条件を OCL を用いて定義し, メタモデルで定義された属性と関連に関する制約を規則として規定する．意味論は, 原則として自然言語で記述され, 構成要素の意味を定義する.

Foundation パッケージの下位には, 基本メタモデルに必要な基本概念の仕様を記述する Core パッケージがある（図 3.2）．その更に下位には, 振る舞い要素群に必要な核となる概念を規定する CommonBehavior, 特定タスクを実行するための振る舞いの文脈を規定する Collaborations 等のパッケージが含まれる.

3.2.2　協調図とメタモデル

協調図は, 役割およびそのリンクをまとめた 1 つの相互作用を示し, 異なる役割を演じるオブジェクト間の関係を表す．協調図では, オブジェクト間の関連と共に, メッセージの流れも表現できる．協調図でメッセージを表現するには, 2 つのオブジェクトをつなぐ実線 (関連) のそばに, 先端をメッセージ受信側に向けた矢印を付加すればよい．図 3.1 に協調図の例を示す．例では, 学生が講義を受けたい教師を指名し (メッセージ 1), その担当講義を 1 つずつ登録していき (メッセージ 1.1), その結果, 教師はどの学生が自分の担当講義を履修するのかを知る (メッセージ 1.i.1). 図 3.1 には, クラス Person,Course が存在し, クラス Person は,3 つの役割を演じている．また, メッセージ 1.i.1 は, メッセージ 1.1 が 1 回送信されるごとに送信される.

本研究では, 各 UML 図の基本概念を記述している Core パッケージ, UML 協調

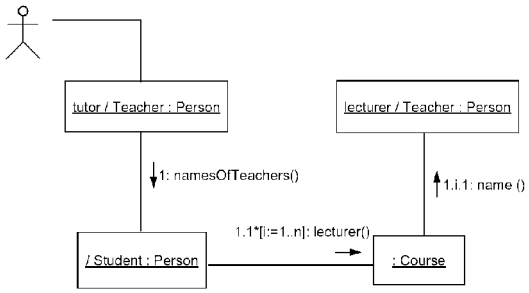

図 3.1: 協調図の例

図の意味論を記述している Collaborations パッケージに焦点をあてている．以下に Core パッケージと Collaborations パッケージの抽象構文を示す.

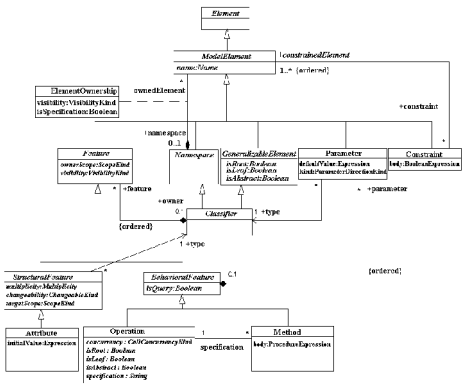

図 3.2: Core

Core パッケージは, オブジェクトモデルの開発に必要な基本的な抽象的構成要素および具象メタモデル構成要素を定義する．抽象的構成要素は, 重要な構成要素の具体化のために共通して使われ, インスタンス化できない．一方, 具象メタモデル構成要素は, オブジェクトモデル製作者が使用するモデル化構成要素を反映し, インスタンス化できる．Core パッケージに定義された抽象的構成要素には,ModelElement(モデル要素),GeneralizableElement(汎化可能要素),Classifier(分類子) があり, 具象メタモデル構成要素には,Class(クラス),Attribute(属性),Operation(操作),Association(関連) がある.

Collaborations パッケージは, モデル内のモデル要素の使い方を規定する．UML で,Collaboration は,Operation(操作) や Classifier(分類子) がどのように Classifiers と Associations(関連) の集合によって実現されるのか,Collaboration に関与するオブジェクト間のやりとりはどう定義するのかを記述する．例えば,Collaborations パッ

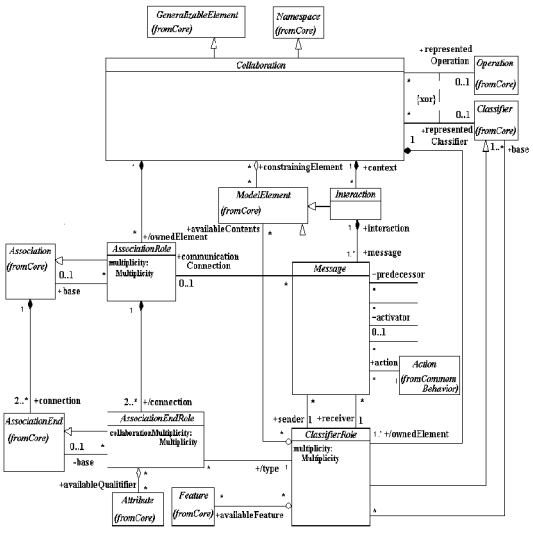

図 3.3: Collaborations

ケージでは,GeneralizableElement が Collaboration の汎化となっている．Collaboration は ClassifierRole に対して必ず ownedElement という関連をもつ．また,Collaboration は Operation に対して representedOperation,Classifier に対して representedClassifier の関連をもつ．各関連は多くとも一つしか存在できず, 二つのうちどちらか一つしか成り立たないことが示されている．ここで, 操作は振る舞いをもたらすためにオブジェクトから要求されるサービス, 分類子は振る舞いと構造上の特性を記述する要素, 関連は分類子間の意味的関係を定義する．

図??で表されたもの以外に, 協調図が満たすべき条件が多数存在する．これらは適格性規則と呼ばれ, 同じ構成要素間の関係に関する条件や, 他のパッケージの要素との間に成り立つ条件を表す．それらの条件には以下のようなものがある．[17]

1. Collaboration は,Classifier か Operation のどちらかである．
2. 全ての AssociationRole は,Collaboration に含まれる ClassifierRole だけに関連している．
3. Collaboration において, 全ての ClassifierRoles と AssociationRoles は,Namespace(名前空間) にある Classifiers,Associations に関連されている．
4. モデル要素に含まれている要素のみに,Constrain(制約) を設けられる．
5. もし 2 つの ClassifierRoles あるいは AssociationRoles が協調内で名前を持っていないなら, それらは異なった基盤を持っている．

6. 協調の親と子で, 同じ名前を持っている役割 (AssociationRole あるいは ClassifierRole) は, その役割の特化であるに違いない

7. 協調 (Classifier を表すことについてのケースで) 内のすべての Interaction 図は representedClassifier に送られたメッセージから始まる

8. ある協調が他の協調を特化したものである場合, 親協調が持つ全ての ClassifierRoles を含まなくてはならない.

9. ある協調が他の協調を特化したものである場合, 少なくともその Intarection(相互作用) の間, 親に存在しているすべてのメッセージを含んでいなくてはならない.

また, 意味論とは図の構成要素の解釈方法を形式的に定義するものであり, 抽象構文内の Instance(インスタンス),Stimulus(刺激) に基づいて規定されている．インスタンスは操作の集合が適用され, 操作の結果を格納する状態を持つ実体を定義し, 刺激はオブジェクト間の関係を示す．以下に協調図の構成要素のもち得る意味を示す. (a) オブジェクト間に関係が存在するか, (b) 他のオブジェクトに関与したか (c) オブジェクトの 1 つがパラメーターの通過によって他のオブジェクトを知っているかが存在する [17].

先の図 3.1 には,Classifier として Person,Course, ClassifierRole として Teacher, Student, AssociationEndRole として Tutor, lecturer, Message として namesOfTeachers（）, lecturer（）, name（）がある．

3.3　記述論理 (Description Logics)

3.3.1　記述論理の表現方法

記述論理は構造化された情報を扱う論理系であり, 変数や関数のない次数に上限を設けた述語論理の部分クラスである [9, 8, 7]．第 1 階述語論理と違って, 充足性判定問題が決定可能であり, 主要な部分クラスでは多項式時間で処理できるという特徴を有する．記述論理では概念 (Concepts) と役割 (Role) から構成される．前者はオブジェクトクラスを意味しており，後者はオブジェクトインスタンスの属性 (2 項関連) を意味する．基本概念 (primitive concept) によって記号が与えられ, $\sqcap, \sqcup$ などの構成子を用いて式 (expression) が定義される．限量作用子 $\forall, \exists$ は役割を介して定義される．基本概念 C, C' 上の役割 R に対して, $\forall R.\, C'$ とは C のオブジェクト x の任意の R 属性値 y に対して $y \in C'$ となることをあらわす．例えば, $Person$(人間), $Doctor$(医者) 概念と, $CHILD$(子供) 役割に対し

て, $Person \sqcap \forall CHILD.\, Doctor$ により, その子供すべてが医者である人物を表現している． $\exists R$ により何かの R 属性が存在することを表す．$C \preceq D$ により包摂関係を表す，すなわちすべての C オブジェクトは D オブジェクトでもある． 例えば $Parent$(親) 概念とは $Person$ で $CHILD$ 属性を有するオブジェクトであり, $Parent \preceq Person \sqcap \exists CHILD$ と表すことができる．

例題 5 記述論理式の例を示す.
(1) 構成要素 AssociationRole から構成要素 Association へ,
多くても 1 つの役割 base しか関連をもたない:
$AssociationRole \preceq \exists^{\leq 1} base.\, Association$
(2) Teacher,Student は, 構成要素 ClassifierRole のインスタンスである:
$Teacher \sqcap Student \preceq ClassifierRole$
(3) nameOfTeachers() は, 構成要素 Message のインスタンスである:
$nameOfTeachers() \preceq Message$
(4) メッセージ nameOfTeachers() の送信者は Teacher で, 受信者は Student である:
$nameOfTeachers() \preceq$
$sender.\, Teacher \sqcap receiver.\, Student$

上述のような記述論理を $\mathcal{AL}$ クラスと呼ぶ．$\mathcal{AL}$ では包摂関係を高速に推論することができる．実際多項式時間で決定可能 (decidable) であることが知られる [9].

データモデリングでは $\mathcal{ALCQI}$ が適合する [8]．$\mathcal{ALCQI}$ では, 概念は，基本記号 A または $\neg C,\ C \sqcap C',\ C \sqcup C',\ \forall R.\, C,\ \exists R.\, C,\ \exists^{\geq n} R.\, C,\ \exists^{\leq n} R.\, C$ の形式であり, 役割は P, P^{-} のいずれかである．ここで,A, P は基本概念・基本役割, C, C', R は概念表現および役割表現を表す.

データベースで扱うデータは有限である．これに対し論理系ではこの制限は通常ない．充足可能ならば必ず有限モデルが存在するとき，その論理系は”有限モデル推論”であるという．$\mathcal{ALCQI}$ は有限モデル推論性を満たさない [7]．このため, 以下では (1) スキーマでは基本概念だけが高々 1 度しか左辺に表れず，(2) 概念を再帰的に定義せず，しかも (3) 有限性条件 (well-founded) $\mathcal{W}$ を仮定した $\mathcal{ALCQIW}$ に限定して議論を行う.

3.3.2　RACER

RACER システムは, 記述論理 $\mathcal{ALCQHI}_{\mathrm{R}+}$のために最適化された計算を実行する推論エンジンであり, 記述論理式をユーザが入力することで, 計算機により包摂関係の推論が自動的にできる．また,RACER はフリーソフトとして公開されてい

る [18]．ここで,$\mathcal{ALCQHI}_{R+}$は, 数値制約, 役割階層，逆の役割と他動詞の役割で拡張された基本的な論理 $\mathcal{ALC}$ である．しかし, 本研究では UML モデルが記述対象なので,$\mathcal{ALCQIW}$ の枠組みだけを使用する．RACER では, 記述論理式を表すために, 記号を項と呼ばれる文字に置き換える必要がある．概念 C が C' に包摂される場合,implies を用いて,(implies C C') と表現する．限量作用子 ∀,∃ は all,some で表し,(some R C) と表現する．他に, 一般否定 ¬は,not, 逆役割は inv で表す．また, 概念 C のインスタンス I は,(instance I C) と表現し, インスタンス I と I' が役割 R で関連している場合,(related I I' R) と表現する．

例題 6 先の例を RACER で扱う式を用いて示す.
(1) 構成要素 AssociationRole から構成要素 Association へ,
多くても 1 つの役割 base しか関連をもたない:
(implies AssociationRole (at-most 1 base Association))
(2) Teacher,Student は, 協調図の構成要素 ClassifierRole のインスタンスである:
(instance Teacher ClassifierRole)
(instance Student ClassifierRole)
(3) nameOfTeachers() は, 協調図の構成要素 Message のインスタンスである:
(instance nameOfTeachers() Message)
(4) メッセージ nameOfTeachers() の送信者は Teacher で, 受信者は Student である:
(related nameOfTeachers() Teacher sender)
(related nameOfTeachers() Student receiver)

RACER には, 記述論理の内容を調査するための質問機能が備わっている．概念 C が充足可能かどうかを知るためには,$(concept - satisfiable?C)$ と表現し, 概念 C が C' に包摂されているかどうかを知るためには,$(concept - subsumes?CC')$ と表現する．

3.4　協調図の形式化

本研究では, 協調図の構文と意味を定義しているメタクラスを記述論理に変換することで, 図の形式化を行う．協調図の定義を直接行っているのは Collaborations パッケージであり, 協調図のメタモデル Collaborations とその上位である Common-Behavior, UML 全体の基本メタモデル Core を記述論理で捉えなおす必要がある．また, パッケージを記述論理を用いて表現するためには, 協調図の満たすべき構文と意味論とを記述論理に変換する必要がある．

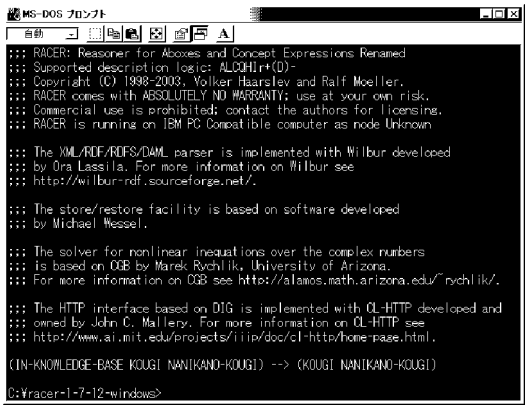

図 3.4: RACER

3.4.1　協調図の構文の形式化

協調図の構造の形式化

ここでは,Collaborations パッケージの抽象構文の要素間にある関係を記述論理で捉えなおしていく．例えば, 汎化関係は, 子クラス（概念 Collaboration）が親クラス（概念 GeneralizableElement）を包摂するという関係で捉えなおす.

$$\mathrm{GeneralizableElement} \preceq \mathrm{Collaboration}$$

クラス間の関連は, 例えば, 概念 Collaboration から概念 ClassifierRole へ役割 ownedElement で関連すると捉えなおす．また, 関連 Operation に対して役割 representedOperation, 概念 Classifier に対して役割 representedClassifier で関連すると捉えなおす. 要素間の多重度は限量子を用いて,$\exists^{\leq 1}$ と捉えなおす.

$$\begin{aligned}&Collaboration \preceq ownedElement.ClassifierRole \sqcap \\ &(\exists^{\leq 1} representedOperation.\ Operation \sqcup \\ &\exists^{\leq 1} representedClassifier.\ Classifier)\end{aligned}$$

上述の考え方を用いて,Collaboration パッケージの残りの要素を記述論理で捉えなおしたものを以下に示す.

$$\begin{aligned}&Namespace \preceq Collaboration \\ &ModelElement \preceq Interaction \sqcup Message \\ &Interaction \preceq context.Collaboration \sqcap \\ &message.\ Message \\ &Message \preceq\end{aligned}$$

$\exists^{\leq 1} communicationConnection.AssociationRole \sqcap$
$sender.ClassifierRole \sqcap receiver.ClassifierRole \sqcap$
$action.\ Action \sqcap \exists^{\leq 1} activator.\ Message$
$AssociationRole \preceq multiplicity.Multiplicity \sqcap$
$\exists^{\leq 1} base.\ Association \sqcap$
$\exists^{\geq 2} connection.\ AssociationEndRole$
$AssociationEndRole \preceq$
$collaborationMultiplicity.\ Multiplicity \sqcap$
$\exists^{\leq 1} base.\ AssociationEnd \sqcap type.\ ClassifierRole$
$ClassifierRole \preceq multiplicity.\ Multiplicity \sqcap$
$base.\ Classifier \sqcap$
$availableContents.ModelElement \sqcap$
$availableFeature.\ Feature$
$Operation \preceq Classifier$

記述論理による Collaboration の制約条件

ここでは, 適格性規則を記述論理に捉えなおす. 以下に適格性規則の変換結果を示す.

1. $((representedClassifier.\ Classifier) \sqcap$
   $(representedOperation.\ Operation)) \doteq \bot$

2. $\exists type^{-}.\ (base^{-}.\ (connection^{-}.\ AssociationRole)))$
   $\preceq \forall base^{-}.\ ClassifierRole$

3. $(name^{-}.\ (base^{-}.\ AssociationRole)$
   $\preceq \forall name^{-}.\ Association) \sqcap$
   $(name^{-}.\ (base^{-}.\ ClassifierRole)$
   $\preceq \forall name^{-} Classifier)$

4. $\forall base^{-}.\ constrainingElements \preceq ModelElement$

5. $((\neg(name^{-}.\ AssociationRole) \sqcup \bot) \sqcap$
   $(name^{-}.\ AssociationRole \sqcup \neg\bot)) \sqcup$
   $\exists_{\leq 1} base^{-}.\ AssociationRole$

7. $((\neg(representedClassifier^{-}.Classifier) \sqcup \bot) \sqcap$
$(\neg(\bot) \sqcup representedClassifier^{-}.Classifier)) \sqcup$
$(\neg(base^{-}.(receiver^{-}.(message0^{-}.Interaction)) \sqcup$
$representedClassifier^{-}.Collaboration) \sqcap$
$(\neg(representedClassifier^{-}.Collaboration) \sqcup$
$base^{-}.(receiver^{-}.(message0^{-}.Interaction)))$

9 $(message^{-}.(parent^{-}.(interaction^{-}.(parentInt^{-}.$
$(generalization^{-}.GeneralizationElement)))))$
$\preceq message^{-}.Interection$

適格性規則 1 は, 前節の記述論理式に存在する Classifier,Operation 双方から役割 representedClassifier, 役割 representedOperation で関連する Collaboration が, 存在しない ($\bot$) と捉えなおす．適格性規則 3 の上二行は, 概念 AssociationRole から役割 base で関連する概念が存在し, 更にその概念から役割 name で関連する概念が, 概念 Association から役割 name で関連する概念に包摂されると捉えなおす．適格性規則 7 は,representedClassifier への始めのメッセージを役割 message0 で捉えなおしている．このように, 基本的には先に述べたパッケージ内の要素間の関係を表わした記述論理式を用いて捉えなおす．完全には記述論理に置き換えられない部分に関しては, 必要ならば新たに記述論理の役割を定義する．また, 各要素の名前は, 概念 String へ関連する役割 name として捉えなおしている．適格性規則 6 は, 変数を用いた論理式で表わす必要があるが, 記述論理では変数を扱えず, 適格性規則 8 は RACER では表現できない．しかし, これらの条件が正しく成り立っているかどうかは, 後述する.

3.4.2　協調図の意味論の形式化

構文的な制約により Collaboration における役割を示すが, ここではインスタンスレベルで, 協調図における役割に従っているインスタンスとリンクを考える．オブジェクト間に存在する, 構文的制約がどういう意味に対応づけられるかを記述論理式で捉えなおし, 以下に示す．a,c では, 刺激の送信者（概念 sender）と受信者（概念 receiver）,b では, 役割 isCreateAction,isDestroyAction で関連している概念があると捉えなおす.

a オブジェクト間に関係が存在するか
$(sender^{-}.Stimulus \preceq$

$$instance^{-}.(connection^{-}.Link))\sqcap$$
$$(receiver^{-}.Stimulus \preceq$$
$$instance^{-}.(connection^{-}.Link))$$

b 他のオブジェクトに関与したか

$$(isCreateAction^{-}.(action^{-}.Stimulus2)\sqcap$$
$$sender^{-}.Stimulus2 = sender^{-}.Stimulus\sqcap$$
$$instantiation^{-}.(action^{-}.Stimulus2) \preceq$$
$$classifier^{-}.(receiver^{-}.Stimulus))\sqcap$$
$$(isDestroyAction^{-}.(action^{-}.Stimulus3)\sqcap$$
$$receiver^{-}.Stimulus = receiver^{-}.Stimulus3)$$

c オブジェクトの 1 つが, パラメーターの通過によって
他のオブジェクトを知っているか

$$(receiver^{-}.Stimulus2 = sender^{-}.Stimulus)\sqcap$$
$$receiver^{-}.Stimulus \preceq argument^{-}.Stimulus2$$

3.5　記述論理による推論

3.5.1　推論手順

ここでは, 前章までで記述論理により形式化された協調図の枠組みを用いて, 図 3.1 の協調図との間に矛盾が存在しないかを RACER で確かめる方法について述べる. 以下の作業が最後まで滞りなく進めば, 記述論理で捉えなおされた協調図の満たすべき抽象構文, 適格性規則と協調図の例との間に矛盾が存在しないことが確認される.

1. 図の要素間の関係を Tbox として記述する
2. 要素に対する適格性規則を Tbox として記述する
   (記述論理で表しきれないものは,
   質問機能によって無矛盾であることを確認する)
3. 協調図の例を Abox として記述する
4. RACER を起動
5. 記述内容に矛盾があれば停止
6. 2. で表現できなかったものを質問機能で確認

7. エラーが表示されなければ矛盾がないことがわかる

3.5.2　推論に用いる例

ここでは, 図 3.1 の協調図を用いて推論を行う．以下に, 図 3.1 に存在する概念や役割を記述論理で捉えなおしたものを示す.

$$
\begin{aligned}
&Person \sqcap Course \preceq Classifier \\
&Teacher \sqcap Student \preceq ClassifierRole \\
&/Teacher : Person \sqcap /Student : Person \sqcap \\
&\quad : Course \preceq AssociationEnd \\
&Tutor \sqcap Lecturer \preceq AssociationEndRole \\
&nameOfTeachers() \sqcap lecturer() \sqcap name() \\
&\quad \preceq Message \\
&Student \sqcap Teacher \preceq base.\, Person \\
&Tutor \sqcap Lecturer \preceq base./teacher : Person \\
&nameOfTeachers() \preceq \\
&\quad sender.\, Teacher \sqcap receiver.\, Student \\
&lecturer() \preceq \\
&\quad sender.\, Student \sqcap receiver.\, CourseCR \sqcap \\
&\quad predecesser.\, nameOfTeachers() \\
&name() \preceq \\
&\quad sender.\, CourseCR \sqcap receiver.\, Teacher \sqcap \\
&\quad predeseccer.\, (nameOfTeachers() \sqcap lecturer())
\end{aligned}
$$

上述の記述論理式は,3 章で述べた方法により RACER 内で使われる式に変換し, それを RACER に入力する.

3.5.3　協調図の推論

RACER 上で前述の例を記述論理の ABox として記述し, 推論を行わせる．RACER は, 利用者が記述した論理が矛盾ないものならば, 何も矛盾を知らせるメッセージが出ることなく動作を終了する．これをもって, 協調図の無矛盾性を確認できたと考える.

協調図の適格性規則の多くについては, 例を RACER 上で記述し推論させた結果, 矛盾は発見されない．例えば, 適格性規則 1 については,Classifier と Operation と

の両方に関連を持つ個体は存在しないので, 例では適格性規則 1 を満たす．適格性規則 2 についても,AssotiationRole が想定されていないので矛盾なく成立する．しかし, 適格性規則 7 については, 矛盾を知らせるメッセージが表示される．原因としては, 適格性規則 7 では, 図内での最初のメッセージのあり方に関する条件を述べているが, 最初のメッセージが設置されていないことが挙げられる．

3.5.4　質問機能による協調図の無矛盾性確認

4 章で述べた通り, 記述論理式で表現し切れなかった Collaboration の適格性規則を RACER のもつ質問機能によって確認する手法について述べる．

適格性規則 6 については, 設計者が記述した記述論理内で, 同じ名前を持つ概念が存在するかを検索する．それは,$(concept - satisfiable?\quad \exists_{\geq 2} name^{-}.\, r)$ という質問を RACER にすることで確認できる．そのような概念が存在する場合には, 一方が他方を包摂しているかを質問で確認する．概念 C1,C2 が存在する時,C1 が C2 を包摂するかを質問するためには,(concept-subsumes? C1 C2) と記述すればよい．その結果, 矛盾を知らせるメッセージが表示されれば, 適格性規則 6 に関する矛盾として判断する．

適格性規則 8 については, 継承関係が存在する協調がある場合に, 親側に存在する分類子役割を検索する．そして, 適格性規則 6 と同様に検索した結果に発見したものが, 子側に存在する分類子役割に包摂されているかを質問する．その結果, 矛盾を知らせるメッセージが表示されれば, 適格性規則 8 に関する矛盾として判断する．

発見された矛盾は, 適格性規則 7 に関するものである．この問題を解決するためには, 役割 representedClassifier で関連される概念を設定し, 協調図の最初のメッセージを設置すればよい
$(message0 \preceq Message,\quad message0 \preceq representedClassifier.\, Person)$．以上の結果から, 実際に記述論理を用いて協調図の整合性を半自動的に検査できることがわかる．

3.6　結び

本研究では, 推論機構が効率よく決定可能である記述論理を用いて,UML 協調図を形式化を行った．また, 実際に例を利用して, 形式化されたものについて RACER システムによる推論を行い, 協調図の矛盾を発見できることが確認できた．

謝辞

本研究の一部は文部科学省科学研究費補助金 (課題番号 14580392) の支援による.

第4章　記述論理を用いたUML整合性の検証システムの実現

4.1　前書き

ソフトウェア開発やオブジェクト指向分析・設計等の分野でUMLが広く利用されている[10]. しかし， よく知られているように，UMLによる記述の整合性を検査するための方法が確立されていない．UML記述手法のうち，協調図はオブジェクト間の関係と通信を宣言的に表現しており，手続き的な順序を考慮する必要がなく，論理との相性がよい．本稿では，論理を用いた協調図の形式化により，協調図による記述の整合性を検証する手法を提案する。実際，この検証過程の自動化を実現するために試作システムを開発した．

UMLメタモデルでは，協調図の満たすべき条件記述がUML図自体によって表現されている．後述するように，メタモデルはモデル構成に関するモデル記述であり，(個々ではなくて汎用的な) 協調図による記述を調べ，その整合性を知るには欠かすことができない。反面，メタモデルのためのUML図を用いて整合性を検証することは煩雑であり，検証そのものの精度も期待しにくい．

記述論理は一階述語論理の部分クラスであり，協調図を論理で捉えることができる[19]．推論のための処理系 (推論エンジン) を用いれば，計算機により自動的に包摂関係を推論することができる．

本稿では， 協調図によって表現された記述 (一般には個別の応用業務) とメタモデルに記述されている条件 (モデル構成条件) の2つを記述論理の式に変換し，双方の無矛盾性を確かめる手法を提案する．協調図で表された表現記述がUMLメタモデルに従っていなければ，推論によって異常が発見されることになる．本稿では，この整合性検査方法を計算機によって処理する方式を示し，実際に試作システムについて報告する．

2章では本研究で扱うUMLとメタモデルとUMLの形式化について，3章では整合性検証のために導入する記述論理について簡単に述べる．4章では記述論理を用いた整合性検証システムについて述べる．5章では例を用いて整合性検証システムの適用例を示し，6章で結びとする．

4.2　UML とメタモデル

協調図は，対象とする応用業務において，オブジェクトの持つ役割およびその関連をまとめて１つの相互作用を示し，異なる役割を演じるオブジェクト間の関係を表す．ここでは，オブジェクト間の関連と共に，メッセージの流れも表現できる．協調図でメッセージを表現するには，２つのオブジェクトをつなぐ実線 (関連) に，先端をメッセージ受信側に向けた矢印を付加すればよい．

UML メタモデルは，UML を用いて UML 自身を表した記述である．表現するモデル構造 (構文)，その意味・解釈を表現したものであり，この意味で応用業務に独立に定義される汎用性を持つ[1]．メタモデルは抽象的なモデルであり，宣言的な意味論に基づいて構築されているため，メタモデルを用いた実装はその意味論に従わなければならない．

UML メタモデルの構成は，抽象的なパッケージ (モデル要素のグループ化したもの) 群に体系化されている．各パッケージは，抽象構文，適格性規則，意味論からなる．抽象構文は構成要素とそれらの関係を定義するメタクラスを示す図で構成され，関係の多重度用件からなる適格性規則を示している．適格性規則は，UML モデルで満たすべき不変条件を OCL (オブジェクト制約記述言語) を用いて定義し，メタモデルで定義された属性と関連に関する制約を規則として規定する．意味論は，原則として自然言語で記述され，構成要素の意味を定義する．本研究では，UML 協調図の構文と意味論を記述している Collaborations パッケージを論じる [17]．Collaborations パッケージは, モデル内のモデル要素の使い方を規定する．Collaboration(協調) は,Operation(操作) や Classifier(分類子) がどのように Classifier と Association(関連) の集合によって実現されるのか, 協調に関与するオブジェクト間のやりとりはどう定義するのかを記述する．

協調図の構文と意味を定義しているメタクラスを記述論理に変換することで, 協調図の形式化を行う．協調図を定義している Collaborations パッケージと，その上位でありモデルの動的な振る舞いを規定する CommonBehavior パッケージ, UML 全体の基本である静的構造を規定している Core パッケージを記述論理で捉えなおす必要がある．また, パッケージを記述論理を用いて表現するためには, 協調図の満たすべき構文と意味論とを記述論理に変換する必要がある．協調図が満たすべき構成要素間の関係に関する条件や，他のパッケージの要素との間に成り立つ条件を以下に示す [17].

1．Collaboration は，Classifier か Operation のどちらかである．

[1]無論，UML という応用業務であると考えることもできるが、ここでは意識して区別する．

2. 全ての AssociationRole は，Collaboration に含まれる ClassifierRole だけに関連している．

3. Collaboration において，全ての ClassifierRoles と AssociationRoles は，Namespace(名前空間) にある Classifiers，Associations に関連されている．

4. モデル要素に含まれている要素のみに，Constrain(制約) を設けられる．

5. もし 2 つの ClassifierRoles あるいは AssociationRoles が協調内で名前を持っていないなら，それらは異なった基盤を持っている．

6. 協調の親と子で，同じ名前を持っている役割 (AssociationRole あるいは ClassifierRole) は，その役割の特化であるに違いない

7. 協調 (Classifier を表すことについてのケースで) 内のすべての Interaction 図は representedClassifier に送られたメッセージから始まる

8. ある協調が他の協調を特化したものである場合，親協調が持つ全ての ClassifierRoles を含まなくてはならない．

9. ある協調が他の協調を特化したものである場合，少なくともその Intarection(相互作用) の間，親に存在しているすべてのメッセージを含んでいなくてはならない．

UML メタモデルにおける意味論とは，図の構成要素の解釈方法の形式化である．具体的には，抽象構文内の”Instance”(インスタンス)，”Stimulus”(刺激) に基づいて規定されている．インスタンスによって，操作の集合が適用され，操作の結果を格納する状態を持つ実体を定義することができる．刺激はオブジェクト間の関係を示すために用いられる．以下に協調図の構成要素のもち得る意味を示す．(a) オブジェクト間に関係が存在するか，(b) 他のオブジェクトに関与したか (c) オブジェクトの 1 つがパラメーターの通過によって他のオブジェクトを知っているかが存在する [17].

4.3　協調図の記述論理による表現

協調図の構文と意味の定義を直接行っているのは，UML メタモデルの中に含まれている Collaborations パッケージである．本研究では，この Collaborations パッケージを記述論理に変換して，協調図の構文と意味との両方の満たすべき条件を

形式的に捉えることができる．パッケージを記述論理を用いて表現するために，協調図の満たすべき構文的制約と，オブジェクト間に存在する構文的制約の対応を記述論理式で表す [19].

記述論理は構造化された情報を扱う論理系であり，変数や関数のない次数に上限を設けた述語論理の部分クラスである [9, 8, 7]．第 1 階述語論理と違って，充足性判定問題が決定可能であり，主要な部分クラスでは多項式時間で処理できるという特徴を有する．記述論理では概念 (Concepts) と役割 (Role) から構成される．前者はオブジェクトクラスを意味しており，後者はオブジェクトインスタンスの属性 (2 項関連) を意味する．

基本概念 (primitive concept) によって記号が与えられ，$\sqcap, \sqcup$ などの構成子を用いて式 (expression) が定義される．限量作用子 $\forall, \exists$ は役割を介して定義される．基本概念 C, C' 上の役割 R に対して，$\forall R.\, C'$ とは C のオブジェクト x の任意の R 属性値 y に対して $y \in C'$ となることをあらわす．
例えば，　$Person$(人間), $Doctor$(医者) 概念と，　$CHILD$(子供) 役割に対して，$Person \sqcap \forall CHILD.\, Doctor$ により，　その子供すべてが医者である人物を表現している．　$\exists R$ により何かの R 属性が存在することを表す．

$C \preceq D$ により包摂関係を表す，すなわちすべての C オブジェクトは D オブジェクトでもある．
例えば $Parent$(親) 概念とは $Person$ で $CHILD$ 属性を有するオブジェクトであり，$Parent \preceq Person \sqcap \exists CHILD$ と表すことができる．

本研究では UML モデルを記述対象とする $\mathcal{ALCQIW}$ の枠組みを使用する．

4.4　整合性検証システム

あるソフトウェア開発工程で整合性検証の対象となる協調図は，構成要素一つ一つを対応付けて，Collaborations パッケージから知り得る構成要素間の関係に基づいて記述論理式へ捉え直すことができる．UML メタモデルの記述論理式と合せて記述論理エンジンに入力として与えて推論させ，エラーなどで止まることなく最後まで実行できれば，記述論理で捉えなおされた協調図の満たすべき構文的・意味的な条件と開発者の書いた協調図との間に矛盾が存在しないことが確認される．

本章では，応用業務に対応して作成された UML 協調図に対して，記述論理を用いる整合性検証システムをどのように構築するかについて述べる．

本システムでは，UML モデル情報は XMI 形式で表現されていると仮定する．これを入力として受け取り，協調図の構成条件の充足性判定を行う．XMI は XML 形式で記述されており，多くの UML モデリングツールによって利用され，特に UML モデリングツール間で情報交換をするための標準規格になっている．

本システムは，C++を用いて開発しており，XMI から論理式への変換機能，推論エンジン RACER とのインターフェイス機能，利用者への結果報告機能から構成される（図 4.1）．なお，RACER はフリーソフトとして公開されている [18].

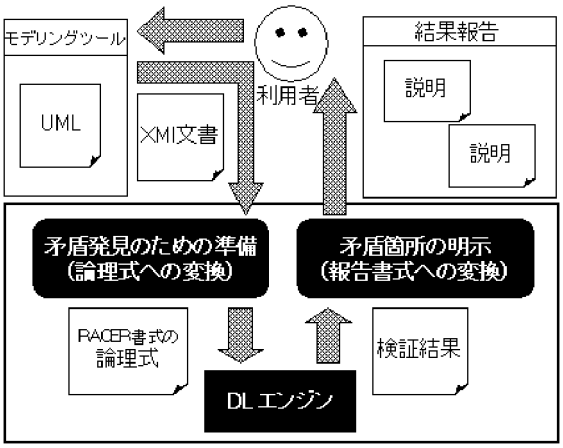

図 4.1: 整合性検証システム

4.4.1 RACER

RACER システム [18] は，記述論理 $\mathcal{ALCQHI}_{R+}$のために最適化された計算を実行する推論エンジンであり，記述論理式をユーザが入力することで，包摂関係の推論を行うことができる．ここで，$\mathcal{ALCQHI}_{R+}$は，数値制約，役割階層，逆の役割と他動詞の役割で拡張された基本的な論理 $\mathcal{ALC}$ である．しかし，本研究では UML モデルが記述対象なので，$\mathcal{ALCQIW}$ の枠組みだけを使用する．RACER では，記述論理式を表すために，記号を項と呼ばれる文字を使用する。本試作システムでは，この置き換え機能を備える必要がある．実際，概念 C が C' に包摂される場合，implies を用いて，(implies C C') と表わす．限量作用子 $\forall, \exists$ は all，some で表し，(some R C) と表現する．他に，一般否定 $\not$は，not，逆役割は inv で表す．また，概念 C のインスタンス I は，(instance I C) と表現し，インスタンス I と I' が役割 R で関連している場合，(related I I' R) と表現する．

概念 C が充足可能かどうかを調べるためには，$(concept - satisfiable?C)$ と表した記述を生成し，概念 C が C' に包摂されているかどうかを知るためには，$(concept - subsumes?CC')$ を生成する．これらはそのまま RACER への入力となる．

4.4.2　XMI から論理式への変換

記述論理を用いて推論するために，ArgoUML[21] や Rational Rose[22] のような UML モデリングツールによって作成された XMI を，RACER で利用可能な記述論理式へ変換する必要がある．そのためには，XMI 文書が協調図の構成要素をどのように表わしているのかを分析し，一つ一つ記述論理の概念や役割に対応させていかなければならない．

XMI 文書から協調図に関係する情報をやみくもに見つけてくるだけでは，構成要素間の関係を捉えることは難しい．本研究では，先の Collaboration パッケージで定義されている構成要素間の関係を利用し，実際に開発者の手で書かれた協調図の全体像を正確かつスムーズにつかむことで，記述論理式として捉えなおすことができる．

応用業務で必要とする協調図の整合性を検証するため，XML のタグ内容から構成要素の情報を獲得しなければならない．大規模なシステムではこの要素の数は膨大であり，メモリに一度で処理しきれる量ではない．本試作システムでは，XML に対してイベント駆動型のアクセス手法 SAX(The Simple API for XML) を使用し，処理容量を減じる．以下では，UML モデリングツールとして argoUML，C++ で XML を扱うために Xerces-C++ を仮定する．

図 3.1 に示す協調図の例を用いて変換の過程を示す．ここでは，Librarian(司書) が蔵書の情報を検索する動作を表す．まず，Librarian は User(利用者) からの質問を受け (メッセージ 1)，Worker(作業員) に調査を指示する (メッセージ 2).

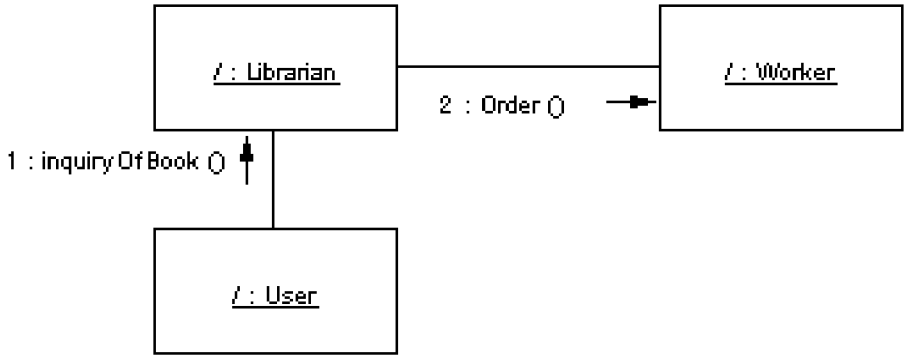

図 4.2: 協調図の例

図 3.1 で,「:Librarian」,「:User」,「:Worker」はそれぞれ Librarian，User，Worker Classifier(クラス) のオブジェクトであることを表わしている．Librarian オブジェクトの ClassifierRole(分類子役割) は，XMI 形式で図 4.3 のように表現される．XMI 形式では，各構成要素に固有の id 番号が振られている．Librarian オブジ

```
<Behavioral_Elements.Collaborations.ClassifierRole xmi.id="xmi.6"
xmi.uuid="127-0-0-1-823c6d:b06a321e26:-7feb">
 <Behavioral_Elements.Collaborations.ClassifierRole.base>
  <Foundation.Core.Classifier xmi.idref="xmi.7"/>
 </Behavioral_Elements.Collaborations.ClassifierRole.base>
 <Behavioral_Elements.Collaborations.ClassifierRole.message2>
  <Behavioral_Elements.Collaborations.Message xmi.idref="xmi.8"/>
 </Behavioral_Elements.Collaborations.ClassifierRole.message2>
 <Behavioral_Elements.Collaborations.ClassifierRole.message1>
  <Behavioral_Elements.Collaborations.Message xmi.idref="xmi.5"/>
 </Behavioral_Elements.Collaborations.ClassifierRole.message1>
</Behavioral_Elements.Collaborations.ClassifierRole>
```

図 4.3: ClassifierRole の例

ェクトには”xmi.6”，Librarian クラスには”xmi.7”，メッセージ 1，2 にはそれぞれ”xmi.5”，”xmi.8”が割り当てられている．User オブジェクトには”xmi.3”，Worker オブジェクトには”xmi.9”が付けられている．<Behavioral_Elements.Collaborations.ClassifierRole.base> タグから ClassifierRole の base(基盤) になっている Classifier が確認できる．Collaborations パッケージより，記述論理では概念 ClassifierRole から概念 Classifier へ関連 base が存在するので，この対応関係を維持したまま以下のように記述論理式に変換する．

(instance id6 ClassifierRole)
(related id6 Librarian base)

<Behavioral_Elements.Collaborations.ClassifierRole.message2>タグから”xmi.8”の Message2(2 番目のメッセージ) を送信していること，<Behavioral_Elements.Collaborations.ClassifierRole.message1>タグからは”xmi.5”の Message1(1 番目のメッセージ) を受信していることが判断できる．それぞれ，Collaborations パッケージの概念 Message から概念 ClassifierRole への関連 sender と関連 receiver の存在から，以下の記述論理式へ変換する．

(instance id8 Message)
(related id8 id6 sender)
(related id8 id9 receiver)

Librarian オブジェクトから Worker オブジェクトへ向かうメッセージ 2 は図 4.4 のように表わされる．

メッセージ 2 には”xmi.8”が割り当てられており，<Behavioral_Elements.Collaborations.Message.activator>タグから”xmi.5”である Message1 が一つ前のメッセージであることが確認できる．<Behavioral_Elements.Collaborations.Message.sender>

```
<Behavioral_Elements.Collaborations.Message xmi.id="xmi.8"
xmi.uuid="127-0-0-1-823c6d:b06a321e26:-7fe1">
 <Behavioral_Elements.Collaborations.Message.activator>
  <Behavioral_Elements.Collaborations.Message xmi.idref="xmi.5"/>
 </Behavioral_Elements.Collaborations.Message.activator>
 <Behavioral_Elements.Collaborations.Message.sender>
  <Behavioral_Elements.Collaborations.ClassifierRole xmi.idref="xmi.6"/>
 </Behavioral_Elements.Collaborations.Message.sender>
 <Behavioral_Elements.Collaborations.Message.receiver>
  <Behavioral_Elements.Collaborations.ClassifierRole xmi.idref="xmi.9"/>
 </Behavioral_Elements.Collaborations.Message.receiver>
 <Behavioral_Elements.Collaborations.Message.communicationConnection>
  <Behavioral_Elements.Collaborations.AssociationRole xmi.idref="xmi.14"/>
 </Behavioral_Elements.Collaborations.Message.communicationConnection>
</Behavioral_Elements.Collaborations.Message>
```

図 4.4: Message の例

タグから sender(送信者) が”xmi.6”の Librarian オブジェクトであること，<Behavioral_Elements.Collaborations. Message.receiver>タグからは受信者が”xmi.9”の Worker オブジェクトであることがわかる．<Behavioral_Elements.Collaborations.Message.communicationConnection>タグからは Librarian オブジェクトと Worker オブジェクトの間に Message2 が存在することがわかる．Collaborations パッケージより，概念 Message 同士の間にある関連 activator，概念 Association から概念 AssociationEnd への関連 connection，概念 Message から概念 AssociationRole への関連 communicationConnection の存在から，この対応関係を維持したまま以下のような記述論理式に変換する．

(related id8 id14 communicationConnection)
(instance id14 AssociationRole)
(related id14 id15 connection)
(related id14 id16 connection)

Librarian オブジェクトと Worker オブジェクトの間の関連は図 4.5 のように表わされる．

<Behavioral_Elements.Collaborations.AssociationRole.message>タグは Association(関連) 間に存在する Message が，<Foundation.Core.Association.connection>タグに囲まれた部分から Association につながっていることがわかる．この関連には”xmi.14”，関連端には”xmi.15”の Librarian オブジェクトと”xmi.16”の Worker オブジェクトが割り当てられている．Collaborations パッケージより，概念 AssociationEndRole から概念 ClassifierRole への関連 type の存在から，以下のような記述論理式に変換する．

(instance id15 AssociationEndRole)
(related id15 id6 type)

```
<Behavioral_Elements.Collaborations.AssociationRole xmi.id="xmi.14"
 xmi.uuid="127-0-0-1-823c8d:b06a321e26:-7fe6">
 <Behavioral_Elements.Collaborations.AssociationRole.message>
  <Behavioral_Elements.Collaborations.Message xmi.idref="xmi.8"/>
 </Behavioral_Elements.Collaborations.AssociationRole.message>
 <Foundation.Core.Association.connection>
  <Behavioral_Elements.Collaborations.AssociationEndRole xmi.id="xmi.15"
xmi.uuid="127-0-0-1-823c8d:b06a321e26:-7fe5">
   <Foundation.Core.AssociationEnd.association>
    <Foundation.Core.Association xmi.idref="xmi.14"/>
   </Foundation.Core.AssociationEnd.association>
   <Foundation.Core.AssociationEnd.type>
    <Foundation.Core.Classifier xmi.idref="xmi.6"/>
   </Foundation.Core.AssociationEnd.type>
  </Behavioral_Elements.Collaborations.AssociationEndRole>
  <Behavioral_Elements.Collaborations.AssociationEndRole xmi.id="xmi.16"
    xmi.uuid="127-0-0-1-823c6d:b06a321e26:-7fe4">
   <Foundation.Core.AssociationEnd.association>
    <Foundation.Core.Association xmi.idref="xmi.14"/>
   </Foundation.Core.AssociationEnd.association>
   <Foundation.Core.AssociationEnd.type>
    <Foundation.Core.Classifier xmi.idref="xmi.9"/>
   </Foundation.Core.AssociationEnd.type>
  </Behavioral_Elements.Collaborations.AssociationEndRole>
 </Foundation.Core.Association.connection>
</Behavioral_Elements.Collaborations.AssociationRole>
```

図 4.5: AssociationRole の例

(instance id16 AssociationEndRole)
(related id16 id9 type)

これらの情報を機械的に組み合わせることで，協調図上に現れる構成要素については記述論理式を生成できる．Collaborations パッケージの内容を利用して，構成要素間の関係を正確に捉えることができる．名前を設定されていない構成要素は，その id 番号を名前に設定する．例えば，”xmi.6”を割り振られた Librarian オブジェクトの ClassifierRole には，id6 という名前が付けられる．図 4.3，4.4，4.5 から，記述論理式を自動的に生成することができる．また，その他の要素からも同じ方法で記述論理式に変換することができる．

4.4.3　推論結果の報告

本機能は，推論エンジンによる推論結果を表示するための支援を行う．本来，推論エンジンを通して得られる推論結果は固有の形式で表され，必ずしも解釈しやすいものとはいえないので，推論結果を解釈可能な方法に変換し開発者に通知する必要がある．本システムでは，協調図の制約条件のうちどれが満たされているのかをメッセージ形式で示す．それにより，利用者は記述論理や RACER について深く知ることなくその恩恵を受けることが可能になる．もちろん，それらの知識をもっている利用者がシステムの生成した式を確認することもできる．

本試作システムで扱う通知内容とは，2 章で述べた協調図の満たすべき条件の充足性判定である．この結果に基づいて，利用者は指摘された制約条件の内容に従

い，協調図を修正することができる．

動作の実例を示す。ここでは図 3.1 の XMI を入力として与える．このとき論理式への変換とその論理式の推論が自動的になされ，推論結果を記したファイルが作成される．利用者はこの内容を確認して，協調図に矛盾が存在しないことを知ることができる．

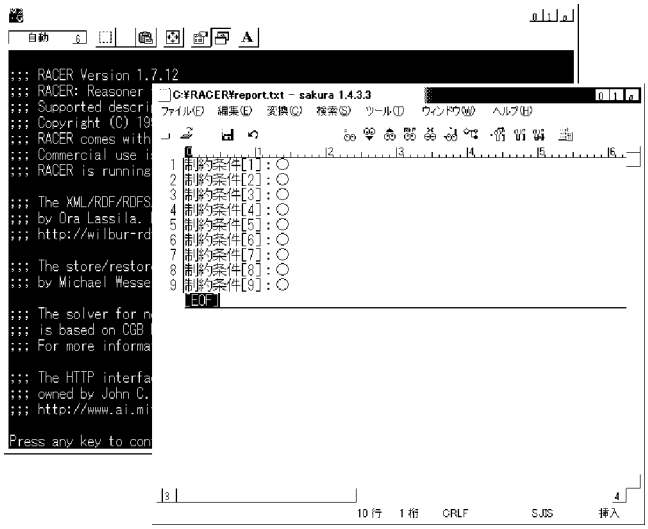

図 4.6: システムの実行例

4.5　整合性検証システムの適用事例

図 3.1 の協調図を考える．司書が蔵書を探し，求める情報がなかった場合，他の提携している図書館の情報を検索できると仮定する．このような処理は，図 4.7 のように表される．

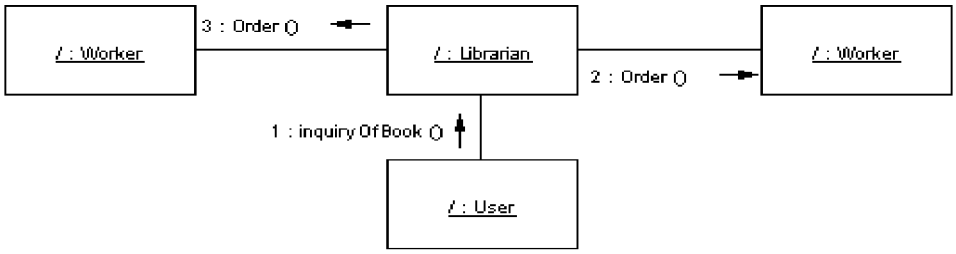

図 4.7: 協調図の例 (2)

整合性検証システムを用いて，図 4.7 の協調図の整合性を検証する．システムに

XMI 文書を入力し，4 章で述べた方法を用いて，記述論理式へ変換する．Classifier User，Librarian，Worker にはそれぞれ”xmi.4”，”xmi.7”，”xmi.11”が，メッセージ 1，2，3 にはそれぞれ”xmi.5””xmi.8””xmi.9”が割り振られている．

このとき検証システムは，整合性検証のための推論を開始する．この例では，制約条件 [5] を充足しないという結果を得た．実際，制約条件 [5] では，同じ base を持った ClassfierRole が存在してはならない．そこで，図 4.7 の 2 つの Worker のオブジェクトは，それぞれ別の役割を有することを明示し，新たに図 4.8 と表現すべきであると理解できる．

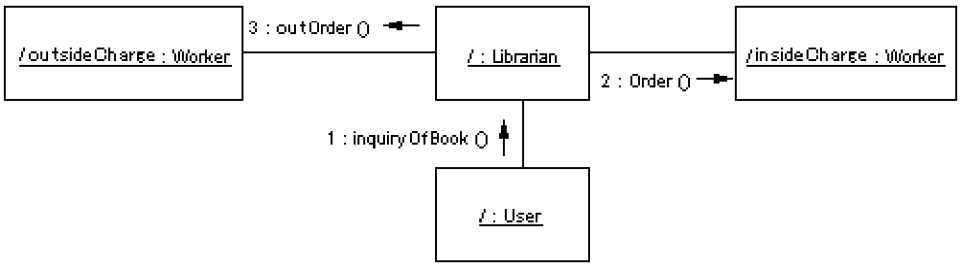

図 4.8: 修正後の協調図

また，図 4.8 において利用者が作業員を指定できるようにする場合，図 4.9 のようになる．これを XMI 文書形式でシステムの入力として与えて，記述論理式へ変換する．

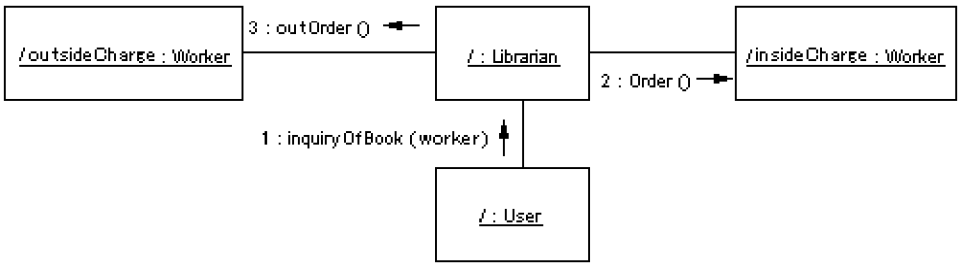

図 4.9: 協調図の例 (3)

この協調図の中に構文的な誤りはないが，本研究のシステムによって整合性を検証すると誤りが発見される．システムでは利用者から送信されるメッセージ 1 の意味も考慮しており，それにより作業者を指定しようとしていることがわかる．しかし，利用者がこの指定をするためには作業者への関連する役割が必要であるのにもかかわらず，そのような関連性は図 4.9 から読み取れない．したがって，本システムは協調図の整合性がないと判断を下すことができる．

このように，整合性検証システムを使うことで，構文的には誤りを認められないような場合いでも自動的に検出できる．協調図を XMI 文書で表現した記述を作成できれば，最終的な充足性判定の段階まではすべて自動化でき，利用者は記述論理や RACER についての知識を有さずとも検証可能である．

4.6　結び

本研究では UML 協調図の形式化と整合性検証方法によって，実際に現場で書かれた協調図が UML の定める構文と意味に従っているかどうかを確認する土台が築かれていることを示した．また，それらに基づいて，記述論理を用いた UML 協調図の整合性検証システムについて述べた．また，実際に例を利用して協調図の矛盾を発見できることが確認でき，整合性検証システムの有用性を示した．

4.7　謝辞

本研究の一部は文部科学省科学研究費補助金 (課題番号 16500070) の支援をいただいた．

第5章　結論

本研究ではソフトウェア開発工程で広く使われているUMLに注目し，その整合性を計算機上で自動的に検査する方法を考えた．UML協調図が成り立つための条件や図そのものを形式的に捉えることによって，図の詳しい分析を人手を使って行うのとは違い，大規模なソフトウェア開発にも対応可能で整合性の機械的な検証をするための枠組みも同時に得ることができた．

記述論理を利用することで，UMLメタモデルや開発者に作成された協調図，協調図から得た条件などをを曖昧さなく形式化でき整合性検証をするための論理的な仕組みが得られた．記述論理の健全かつ完全で決定可能な推論機構によって，UMLメタモデルが持つ多数の条件と開発者が作成した協調図から抽出した記述論理式が乱立する状態でも充足可能性を判断して，協調図の整合性検証を実現することができた．記述論理の推論エンジンであるRACERを利用することで，UMLメタモデルや開発者による協調図などを記述論理式化すれば人手を使うことなく自動的にそれらの間に存在しうる矛盾を発見し，比較的大規模なソフトウェア開発にも対応した協調図の整合性検証ができた．

協調図をメッセージに関わる前提条件 P_1, 発火条件 F, 完了条件 P_2 を用いて分析することで，協調図の満たさなければならない制約条件を端的に取り出すことができた． 協調動作リンクの種類に基づいて P_1,F などから P_2 を機械的に生成することもできた．取り出された条件は協調図内の様子をよく表しており，構成要素の存在の是非や構成要素間の矛盾等の協調図内の整合性を検証するのに役立った．

大規模で複雑なUMLに対応するための違う方法として，開発者の作成した協調図を機械的に置き換える方法を提案することで，協調図の含む全ての情報を要約することなく扱えるようにした．協調図を規定するUMLメタモデルの抽象構文，適格性規則，意味論が記述論理式へ置き換えられることを示した．開発者によって書かれた協調図も記述論理式で置き換えることができ，前述の記述論理式と合せることで，協調図に記載されている情報をよりUMLの思想に忠実な整合性検証を行うことができた．この手法をシステム化することで，協調図からの機械的な変換からUMLメタモデルに従っているかどうかの検証までを含めて自動的に行えることを示した．UMLの知識以外を必要とする作業をシステムの利用者が行わなくてもよいように，入力としてUMLモデリングツールで標準的に採用されている

XMI形式のデータを与えるだけで済むような手法を採用し，論理をあまり知らない人でも十分に扱えるようにした．

これらの研究により，UML協調図の持つ定義を形式化して構成要素間の関連を正確に表現することができ，開発者が作成した協調図の整合性を自動的に検証しうる方法を示した．UMLは多くのソフトウェア開発現場で使われており，UML図の構成要素とその間の関連などの矛盾はソフトウェア分析・設計工程中で簡単に起こりうる上に後の工程にも影響を及ぼす厄介なものである．本研究の手法を用いることで開発者が作成した協調図の矛盾に早い段階で気付け，ソフトウェアの品質を保つことに貢献できる．

謝辞

本研究を遂行するにあたり，日頃より数々のご指導をいただいた，法政大学工学部情報電気電子工学科 三浦孝夫教授に深く御礼申し上げます．

また，産能大学経営情報学科 塩谷勇教授にも多くのご指導をいただきました．深く感謝いたします．

データ工学研究室の先輩方，同輩，後輩たちにも，本研究の遂行にあたって数多くの助言と快適な研究環境の整備をして頂きました．御礼申し上げます．

よき理解者である友人達，兄弟にも幾度となく助けられました．修士論文として私の研究をまとめることができたのも，多くの皆様方の御支援，御協力の賜物であります．この場をお借りしまして，厚く御礼申し上げます．

最後に，今までの学生生活を支えてくださった私の両親に感謝したいと思います．

参考文献

[1] Abdrurazik, A. and Offutt, J.: Using UML Collaboration Digrams for Static Checking and Test Generation, IEEE UML, 2000, pp.383-395

[2] Artale, A., Franconi, E. et al: Description Logics for Modelling Dynamic Information, in *Logics for Merging Application of Databases*, Springer, 2003

[3] Bernardi, D., Calvanese, D. et al.: Reasoning on UML Class Diagrams using Description Logic Based Systems, *KI2001* Intn'l Workshop on Application of Description Loics, 2001

[4] Bernardi, D., Calvanese, D. etal: Reasoning on UML Class Diagrams, Technical Report 11-03, Dipartimento di Informatica e Sistemistica, Universita di Roma "La Sapienza", 2003

[5] Borgida,A : Description Logics in Data Management, IEEE TKDE 7-5, 1995, pp.671-682

[6] Cali, A., Calvanese, D. et al: A Formal Framework for Reasoning on UML Class Diagrams, Intn'l Symp. on Methodologies for Intelligent Systems (ISMIS02), 2002, pp.503-513

[7] Calvanese, D.: Finite Model Reasoning in Description Logics, *KR96*, 1996

[8] Calvanese, D., Lenzerini, M. et al: Description Logics for Conceptual Modelling, in *Logics for Databases and Information Systems*, Kluwer, 1998

[9] Donini, F.: Reasoning in Description Logics, in *Principle of Knowledge Representation*, CSLI Publication, 1996

[10] Object Management Group: OMG UML Specification 1.3, 1999, `www.omg.org.uml/`.

[11] Overgaard, G.: A Formal approach to Collaborations in the Unified Modeling Language, IEEE UML, 1999, pp.99-115

[12] Roger , M. Simonet, A. et al.: Bringing Together Description Logics and Databases in an Object Oriented Model, *DEXA*, 2002

[13] Shiroiwa, M., Miura, T. et al: Meta Model Approach for Mediation, IEEE proc.*COMPSAC*, 2003, to appear

[14] Van Der Straeten. R.: Using Description Logic in Object-Oriented Software Development, *DL02*, 2002

[15] Wieringa, R.: A Survey of Structured and Object Oriented Software Specification Methods and Techniques, ACM Comp.Surv. 30-4 (1998), pp.459-527

[16] ファウラー, M., スコット, K.: UML モデリングのエッセンス (第2版), 翔泳社, 2000

[17] Maria Agustina Cibran,Vanesa Mola,Claudia Pons,Wanda Marina Russo: "Rigorous description of the syntax and semantics of UML Collaborations", *ASSE2000*,Argentina

[18] RACER,http://www.sts.tu-harburg.de/ r.f.moeller/racer/

[19] Nakanishi, H. , Miura, T. and Shioya, I. : Reasoning in Collaboration Diagrams by Description Logics, *Computer and Their Applications* (CATA), 2004

[20] 中西啓之，三浦孝夫 :記述論理による協調図の形式化，DEWS2004

[21] argoUML, http://argouml.tigris.org/

[22] Rational Rose, http://www-6.ibm.com/jp/software/rational/