

記述論理を用いた UML 整合性の検証システムの実現

中西 啓之[†] 三浦 孝夫[†]

[†] 法政大学 工学研究科 電気工学専攻 〒184-8584 東京都小金井市梶野町 3-7-2

E-mail: [†]{i03r3229,miurat}@k.hosei.ac.jp

あらまし UML(統一モデリング言語)のうち,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整合性を検証する方法を提案する.また,この検証過程の自動化を実現するために UML 整合性の検証システムを開発する.

キーワード UML 整合性, 記述論理

Verifying UML consistency by using Description Logics

Hiroyuki NAKANISHI[†] and Takao MIURA[†]

[†] Dept.of Elect.& Elect. Engr., HOSEI University 3-7-2, KajinoCho, Koganei, Tokyo, 184-8584 Japan

E-mail: [†]{i03r3229,miurat}@k.hosei.ac.jp

Abstract In UML(Unified Modeling Language), we formalize collaboration diagrams in a framework of DLs(Description Logics) and describe how to reason consistency, validity and redundancy among them. By means of our logic framework, we can automate whole validation processes of diagram syntax and some relevant semantics soundly and completely. To show the usefulness of our framework, we show some experimental system and discuss how useful we can examine validity during the process.

Key words UML consistency, Description Logics

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[2]. 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[7]. 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.

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^{注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、関係の多重度要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 [1]。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 [1]。

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。

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 [1]。

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 [7]。

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 [4] ~ [6]。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。

$C \sqsubseteq D$ 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。

(注1): 無論、UML という応用業務であるとも考えることもできるが、ここでは意識して区別する。

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, $Parent \sqsubseteq Person \sqcap \exists CHILD$ 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 *ALCQIW* の枠組みを使用する。

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 1)。なお, RACER はフリーソフトとして公開されている [3]。

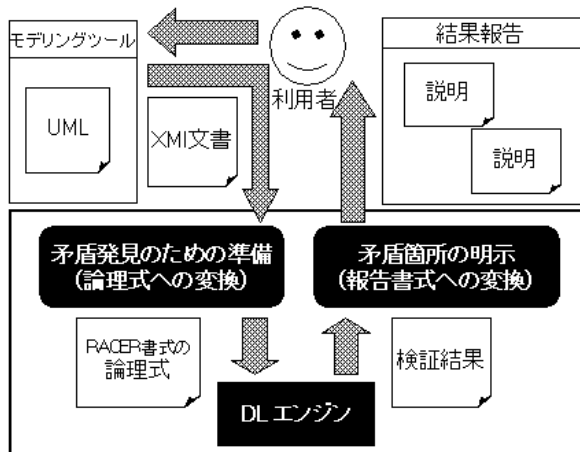


図 1 整合性検証システム

4.1 RACER

RACER システム [3] は, 記述論理 *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, *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 *ALC* である。しかし, 本研究では UML モデルが記述対象なので, *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 は, *not*, 逆役割割は *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。

4.2 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 [8] や Rational Rose [9] 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 文書から協調図に關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ることができる。

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ない量ではない。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 2 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)。

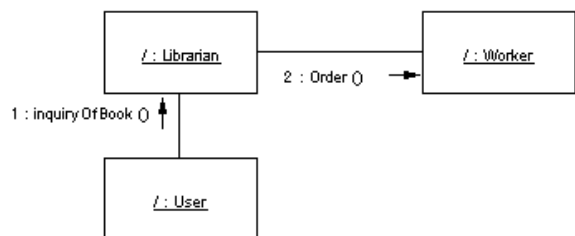


図 2 協調図の例

図 2 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 3 のように表現される。XMI 形式

```

<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 3 ClassifierRole の例

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 のように表わされる。

```

<Behavioral_Elements.Collaborations.Message xmi.id="xmi.8"
xmi.uuid="127-0-0-1-823c6d:b06a321e26:-7feb">
  <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 Message の例

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>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 5 のように表わされる。

```

<Behavioral_Elements.Collaborations.AssociationRole xmi.id="xmi.14"
xmi.uuid="127-0-0-1-823c6d:b06a321e26:-7feb">
  <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="127-0-0-1-823c6d:b06a321e26:-7feb">
      <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:-7feb">
    <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 5 AssociationRole の例

<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)
(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 3, 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。

4.3 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 2 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確認することができる。



図 6 システムの実行例

5. 整合性検証システムの適用事例

図 2 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 7 のように表される。

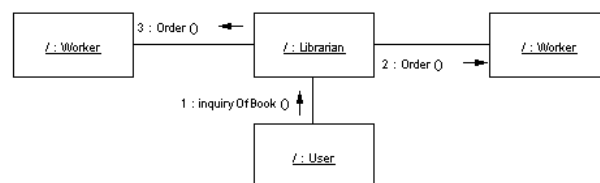


図 7 協調図の例 (2)

整合性検証システムを用いて、図 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 ClassifierRole が存在してはならない。そこで、図 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 8 と表現すべきであると理解できる。

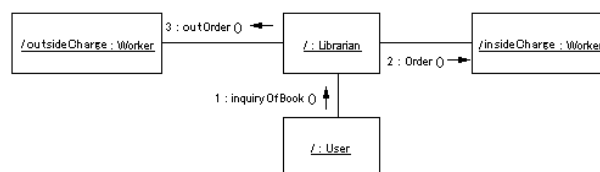


図 8 修正後の協調図

また、図 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 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。

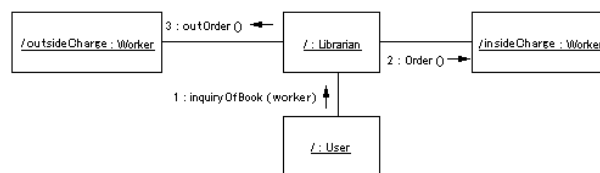


図 9 協調図の例 (3)

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、それにより作業員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員への関連する役割が必要であるにもかかわらず、そのような関連性は図 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．

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．

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．

文 献

- [1] Cibran, M. A., Mola, V., Pons, C., Russo, W. R.: "Rigorous description of the syntax and semantics of UML Collaborations", *ASSE2000*, Argentina
- [2] Object Management Group: "OMG Unified Modeling Language Specification", 1999
- [3] RACER, <http://www.sts.tu-harburg.de/~r.f.moeller/racer/>
- [4] Donini, F.: Reasoning in Description Logics, in *Principle of Knowledge Representation*, CSLI Publication, 1996
- [5] Calvanese, D., Lenzerini, M. et al: Description Logics for Conceptual Modelling, in *Logics for Databases and Information Systems*, Kluwer, 1998
- [6] Calvanese, D.: Finite Model Reasoning in Description Logics, *KR96*, 1996
- [7] Nakanishi, H., Miura, T. and Shioya, I.: Reasoning in Collaboration Diagrams by Description Logics, *Computer and Their Applications (CATA)*, 2004
- [8] argoUML, <http://argouml.tigris.org/>
- [9] Rational Rose, <http://www-6.ibm.com/jp/software/rational/>