記述論理を用いたUML整合性の検証システムの実現

Verifying UML Consistency by Using Description Logics

中西 啓之♦ 三浦 孝夫♠

Hiroyuki NAKANISHI Takao MIURA

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．

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.

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[1].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．本稿では，論理を用いたUML協調図の形式化により記述内容の整合性を検証する手法を提案し，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する．

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．

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．

第2章ではUMLメタモデルの特性を要約し，第3章では記述論理との関連を述べる．第4章ではこの枠組みを用いた検証システムを概説し，第5章で応用事例を示す．

2. UMLとメタモデル

UML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る．

UMLメタモデルは，表現するモデル構造(構文)，その意味・解釈をUML構文で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であり，宣言的な意味論に基づいて構築されている．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，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す．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)により定義し，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．

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．パッケージを記述論理によっ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[2].

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又は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の特化である場合，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，親に存在するすべてのメッセージを含まなければならない.

UMLメタモデルにおける意味論は，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．前者によって，操作集合の適用結果を格納する状態を持つ実体が定義される．後者はオブジェクト間の関係を示すために用いられる．協調図の構成要素が有する意味は次の3つと規定される[2]：(a) オブジェクト間に関係が存在するか，

♦ 学生会員 法政大学大学院工学研究科電気工学専攻

♠ 正会員 法政大学大学院工学研究科電気工学専攻

miurat@k.hosei.ac.jp

[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.

（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[2]．このように，UMLメタモデルを用いることで構文の構成・意味，構文の意図の解釈を表すことができる．

3. 協調図の記述論理による表現

本稿では，UML メタモデルのうち，協調図の構文と意味とを直接定義している Collaborations パッケージを記述論理に変換する．これにより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，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ことができる[3]．

記述論理は構造化された情報を扱う論理系である．変数や関数が無く，次数に上限を設けた述語論理の部分クラスである[4][5][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．これらは個々に記号が与えられ， u，tなどの構成子を用いて式(expression) が定義される．限量作用子∀，∃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，∀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． ∃Rにより何かのR属性が存在することを表す．R^{-1}によって逆向きの関連を表現する．

$C \sqsubseteq D$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．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* ⊓∃CHILD と表すことができる．

4. 整合性検証システム

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，正しく実行できれば，記述論理で捉え直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与えた協調図との間に矛盾がないことが確認できる．本稿では，UML モデル情報は XMI 形式で表現されていると仮定する．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，モデリングツールである ArgoUML[8] によって作成された 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[7]．

4.1 RACER

RACER システムは，記述論理クラス $ALCQHI_{R+}$ 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クラス ALC を表す．しかし，本研究では UML モデルが記述対象なので一部の枠組み(*ALCQIW*)だけを使用する．

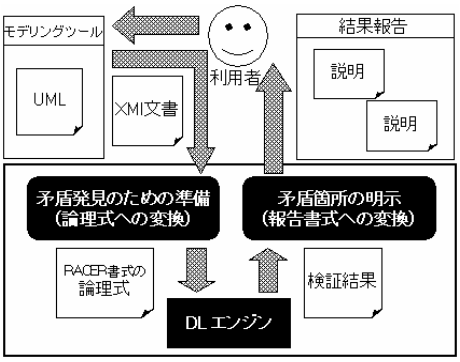

図 1 整合性検証システム

Fig.3 Verification System

RACER では，記述論理式の記号を文字に置き換えて表す．例えば概念 C が C'に包摂される場合，implies を用いて，(implies C C')と表す．限量作用子∀，∃は all，some で表し，∃R は (some R C) のように表現する．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 で関連する場合，(related I I' R)と表現する．

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，XMI を RACER で利用可能な記述論理式へ変換する．本研究では，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する．開発者が与え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．

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)．

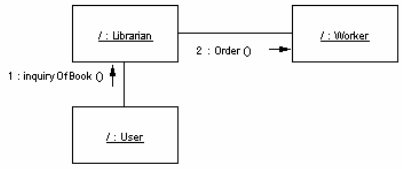

図 2 協調図の例

Fig.2 Example of Collaboration diagram

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す．Librarian オブジェクトの ClassifierRole(分類子役割)は，XMI 形式では図 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，User オブジェクトには"xmi.3", Worker オブジェクトに

```
<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

Fig.3 Example of ClassifierRole

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)である 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.message>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確認できる．これらはそれぞれ，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:-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 Messgae の例

Fig.4 Example of Message

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 オブジェクトであり，<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できる．

(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:-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="127-0-0-1-823c6d: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 5 AssociationRole の例

Fig.5 Example of AssociationRole

<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記述できる．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，他の要素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

推論エンジンから得られる結果は固有の形式で表されるため，必ずしも解釈が容易ではない．このため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．従って，利用者は記述論理や RACER を直接介さずに結果のみを受けることが可能になる．本試作システムでは，2 章で述べた協調図の満たすべき条件の充足性判定を扱う．判定結果に基づいて，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．

動作の状況を示す．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知ることができる．

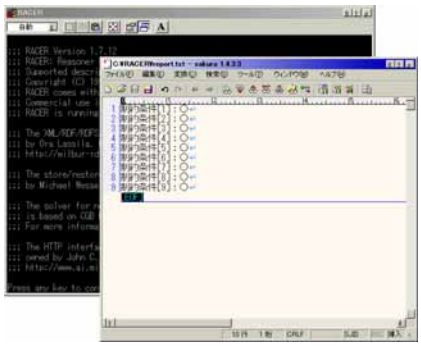

図 6　システムの実行例

Fig.6　Executive Example of System

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．

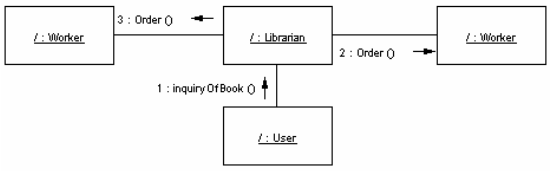

図 7　協調図の例（２）

Fig.7　Example of Collaboration diagram（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 ClassfierRole 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．

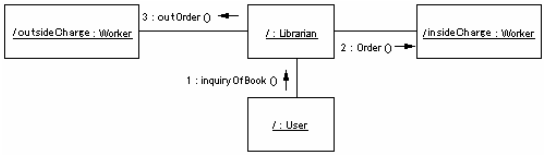

図 8　修正後の協調図

Fig.8　Revised Collaboration diagram

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．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，そのような関連性は図９から読み取れない．従って，本システムは協調図の整合性がないと判断する．

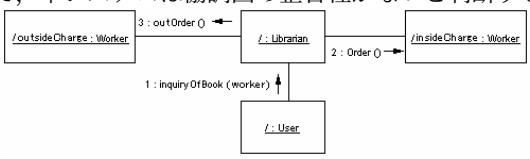

図 9　協調図の例(3)

Fig.9　Example of Collaboration diagram（3）

このように，本稿で提案する整合性検証システムを使うことで，構文上は正しいが解釈が誤っている状況も検出できる．この意味で単純な構文チェッカを超えた機能を有している．

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，利用者が与えた協調図が 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枠組みを提案した．それらは記述論理を用いているため，UML 協調図の整合性検証システムを構築することが可能であり，実際に試作システムを用いて応用例での協調図の矛盾を発見できることを示し，その有用性を示した．

[謝辞]

本稿に対して貴重なコメントをいただいた塩谷勇教授（産能大学）に感謝します．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] Object Management Group: "OMG Unified Modeling Language Specification", (1999).
[2] Cibran, M. A., Mola, V., Pons, C., Russo, W.R.: "Rigorous description of the syntax and semantics of UML Collaborations", ASSE2000, (2000).
[3] Nakanishi,H., Miura,T. and Shioya, I.: “Reasoning in Collaboration Diagrams by Description Logics”, Computer and Their Applications（CATA）, (2004).
[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] RACER,//www.sts.tu-harburg.de/~r.f.moeller/racer/
[8] ArgoUML, //argouml.tigris.org

中西 啓之　Hiroyuki NAKANISHI
法政大学大学院工学研究科修士課程修了．ソフトウェア工学の研究・開発に従事．

三浦 孝夫　Takao MIURA
京都大学理学部，工学博士(東京大学)．現在，法政大学工学部情報電気電子工学科教授．データモデル，知識表現，演繹データベース，複合オブジェクトなどの分野の研究に従事．電子情報通信学会，ACM 各会員．著書に"データモデルとデータベース"(全 2 巻，サイエンス社)