形式手法を用いたUMLの検証に関する研究

2000MT040 加藤 智也　　2000MT053 宮嶋 健至　　2000MT073 小川 行政
指導教員　野呂 昌満

1　はじめに

ソフトウェア開発では，仕様書・設計書を矛盾なく記述することが必要である．一般に，オブジェクト指向開発における仕様書・設計書は，UML[3] や自然言語を用いて記述される．UML や自然言語を用いた仕様記述には，以下の二つの問題点が挙げられる．

- 直観的には理解しやすいが，一意解釈を得ることができず，UML 図と UML 図間の整合性の検証を行うことが困難である．
- 各 UML 図は，対象システムを複数の視点から記述しているので，UML 図間に矛盾が発生する可能性がある．

本研究の目的は，UML 図を形式仕様言語を用いて記述する方法及び，UML 図と UML 図間の整合性の検証方法を提案することである．UML 図を形式的に記述することで，解釈が一意になり，検証可能となる．UML 図とそれらの間の整合性を検証することで，モデルに矛盾が無いことを示すことができると考える．

UML 図の形式記述の方法を定義し，整合性を提示する．形式仕様記述間で整合性の検証を行うことで，UML 図とそれらの間の整合性の検証ができると考える．我々の形式記述の特徴は，静的な関係を記述した UML 図はモデル指向を用いて，動的な関係を記述した UML 図はプロセス代数を用いて記述し，形式仕様記述間で整合性を検証することにある．

VDM-SL[1] と CSP[2] を用いることで，UML 図に対する形式記述の方法を示すことができた．定義した形式記述の方法に従って UML 図を記述し，形式仕様記述間で整合性を検証する方法を提案する．整合性の検証をエレベータシステムで行い，矛盾が無いことを確認した．

おもに加藤はシーケンス図の CSP 記述を担当し，小川はステートチャートの CSP 記述を担当した．宮嶋はクラス図の VSM-SL 記述を担当した．

2　対象とする UML 図と形式記述

本研究で対象とする UML 図と，各 UML 図の形式記述の方針を示す．

2.1　対象とする UML 図

本研究では，クラス図，シーケンス図，ステートチャートを対象とする．オブジェクト図，コラボレーション図，コンポーネント図，配置図等は以下の理由から対象から省く．オブジェクト図は，クラス図の具体的なインスタンスを記述した図であるので，クラス図を検証すれば十分である．コラボレーション図は相互作用を表した図である．表現力はシーケンス図と同等であるので，シーケンス図を取り扱う．コンポーネント図，配置図は，図式表現だけで十分であると考え，形式記述は行わない．

クラス図
本研究で用いるクラス図の構成要素として，クラス，属性，メソッド，関連を挙げる．属性は属性名と型を定義する．メソッドは，メソッド名，引数，返り値，各引数と返り値の型を定義する．関連では，関連があることだけを定義する．クラス間の関係には，関連の他に汎化や依存という関係もある．クラス図では，メソッドのシグネチャや，クラス間のメッセージのやりとりが重要であると考えるので，汎化や依存は考えない．

シーケンス図
本研究で用いるシーケンス図では，メッセージの実行順序が重要であると考え，構成要素をオブジェクト，メッセージ，時間的順序とする．

ステートチャート
本研究で用いるステートチャートの構成要素として，状態，イベントトリガ，アクション，ガード条件を挙げる．イベント，アクションの実行順序と状態遷移が重要であると考え，状態の構成要素は状態の名前だけとする．アクションでは，別のオブジェクトに対するトリガイベントだけを定義する．本研究では，一つのオブジェクトに対して一つのステートチャートを記述する．

2.2　形式記述の方針

クラス図の形式記述の方針
クラス間の静的な関係を記述するには，モデル指向言語が適しており，本研究では VDM-SL を用いる．

シーケンス図，ステートチャートの形式記述の方針
シーケンス図は，メッセージの時間的順序に焦点をあわせて，ステートチャートは，イベントによって変化する状態に焦点をあわせて，動的な側面を記述している．動的な振る舞いを記述するには，プロセス代数が適しており，本研究では CSP を用いる．

3　UML 図の形式的な記述法

クラス図，シーケンス図，ステートチャートの形式記述の方法を定義し，簡単な例としてスイッチとライトを用いて形式記述の方法を述べる．

3.1　スイッチとライトの概要

機能は点灯，消灯の二つである．スイッチをオンにするとライトが点灯し，オフにするとライトが消灯する．

(1) クラス図の形式記述法
クラス図の構成要素と VDM-SL との対応を以下に示す．

クラス図　　VDM-SL
- クラス　→　モジュール
- 属性　→　状態
- メソッド　→　操作
- 関連　→　輸入 (import)

クラス図の各クラスを VDM-SL ではモジュールとし，クラス名とモジュール名を対応させる．属性は状態として，各メソッドは操作として定義する．操作には事前条件，事後条件を記述し，具体的な処理内容も記述する．関連は他のモジュールから必要な定義を輸入する．

スイッチとライトのクラス図と VDM-SL **記述例**

クラス図と VDM-SL 記述例を**図 1** に示す．

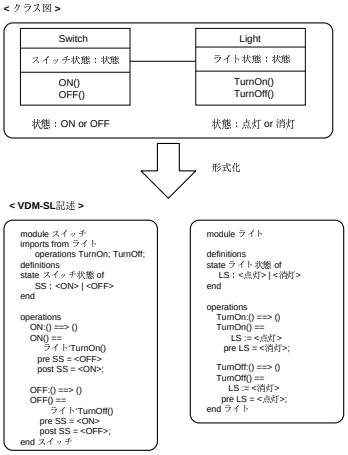

図 1: クラス図の VDM-SL 記述例

図 1 では Switch クラスと module スイッチ，Light クラスと module ライトが対応している．スイッチ状態，ライト状態という属性は状態として記述する．

(2) **シーケンス図の形式記述法**

シーケンス図の構成要素と CSP との対応を以下に示す．

シーケンス図　　CSP
- メッセージ　→　イベント
- 時間的順序 (縦軸)　→　軌跡

シーケンス図のメッセージを CSP ではイベントとして記述し，メッセージ名とイベント名を対応させる．時間的順序を CSP の軌跡を用いて記述する．CSP のイベント名は，送信元が A，送信先が B のメッセージが function(a) のとき，**'a_function_b_a'** とする．メッセージの前に送信元，後に送信先のクラス名と対応づけた記述を付随させる．送信元がオブジェクトでない場合は，送信元を省略する．

スイッチとライトのシーケンス図と CSP **記述例**

シーケンス図を**図 2** に示す．スイッチがオンになり，ライトが消えるまでのシーケンス図である．

図 2 の CSP 記述を以下に示す．

$$\langle on\_light, switch\_turnon\_light, \\ off\_light, switch\_turnoff\_light \rangle$$

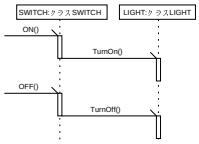

図 2: シーケンス図

(3) **ステートチャートの形式記述法**

ステートチャートの構成要素と CSP との対応を示す．

ステートチャート　　CSP
- 状態　→　プロセス
- イベントトリガ　→　イベント
- アクション　→　イベント
- ガード条件　→　条件分岐 (if、else 文)

ステートチャートの状態を，CSP ではプロセスとして記述する．イベントトリガ，アクションを CSP ではイベントとして記述する．ガード条件は CSP の条件分岐を用いて記述する．プロセス名は，オブジェクト名が A，状態名が ON のとき，**'A_ON'** と記述する．またイベント名は，イベント，またはアクションが function(a) であるとき，**'function.a'** と記述する．イベント名は，イベントトリガ名，アクション名と対応させる．

スイッチとライトのステートチャートと CSP **記述例**

スイッチのステートチャートを**図 3** に示す．

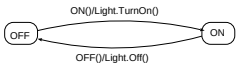

図 3: Switch のステートチャート

SWITCH のステートチャートの CSP 記述を示す．

$$SWITCH\_OFF = on \rightarrow turnon \rightarrow SWITCH\_ON$$
$$SWITCH\_ON = off \rightarrow turnoff \rightarrow SWITCH\_OFF$$

ライトのステートチャートを**図 4** に示す．

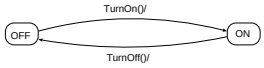

図 4: Light のステートチャート

LIGHT のステートチャートの CSP 記述を示す．

$$LIGHT\_OFF = turnon \rightarrow LIGHT\_ON$$
$$LIGHT\_ON = turnoff \rightarrow LIGHT\_OFF$$

4　整合性の検証

本研究で考える三つの UML 図とそれらの間の整合性を提示し，検証方法を提案する．検証の組合せは 6 通りあるが，シーケンス図間の整合性は以下の理由から省き，5 通りの検証を行う．シーケンス図はメッセージのやりとりの一例である．それぞれのシーケンス図は互いに独立しているので，シーケンス図間の整合性は省く．

4.1　クラス間の整合性

クラス間では，クラス間のメッセージのやりとりに誤りが無いという整合性が考えられる．我々の形式記述では，属性，メソッドの名前を対応づけて記述している．名前の対応づけをすることで，VDM-SL の Toolbox を

用いて検証することができる．構文検査，型検査を行うことで整合性の検証を行う．

4.2 クラスとステートチャート間の整合性

クラスとステートチャート間では，クラスのメソッドの定義とステートチャートの状態遷移との整合性が考えられる．クラス図のメソッドとステートチャートのイベントトリガが対応しており，イベントトリガに付随するアクションが他のクラスで定義されているかを調べる．我々の形式記述では CSP のプロセス名が状態名であり，CSP のイベントと VDM-SL の操作が対応している．**図 5** の対応を調べることで検証を行う．

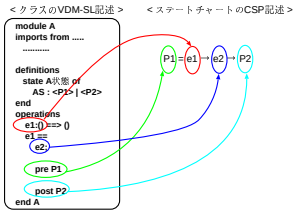

図 5: クラス図とステートチャート間の整合性

状態'P1' から状態'P2' へ遷移するステートチャートを考える．イベントを'e1'，アクションを'e2' とすると，状態遷移は CSP を用いて**図 5** のように記述できる．イベント'e1' は，VDM-SL では操作'e1' に，プロセス'P1' は事後条件'P1' に対応している．イベント'e2' は，具体的な処理内容である操作'e2' に対応している．プロセス'P2' は事後条件'P2' に対応している．名前を鍵に，**図 5** の対応関係を調べることで整合性の検証を行う．

4.3 クラス図とシーケンス図間の整合性

クラス図とシーケンス図間では，クラス図で表される関連に対して，シーケンス図のメッセージのやりとりに誤りが無いという整合性が考えられる．シーケンス図でメッセージのやりとりがあるオブジェクト間では，クラス図でも関連があるということを誤りが無いと言う．我々の形式記述ではクラス図の関連を VDM-SL の imports 記述を用いて記述する．シーケンス図におけるメッセージのやりとりは CSP のイベントで表現する．イベントに記述されるメッセージの送信元，送信先が，クラスの VDM-SL で imports 記述されているか調べる．

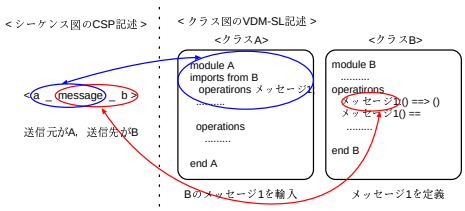

図 6: クラス図とシーケンス図間の整合性

シーケンス図において，' メッセージ 1' が' オブジェクト A' から' オブジェクト B' に送られているとする．CSP では，**図 6** のように記述する．クラス図の VDM-SL 記述では，' モジュール A' において，' モジュール B' の操作' メッセージ 1' を輸入し，' モジュール B' において操作' メッセージ 1' が定義されている．**図 6** の対応を調べることで整合性の検証を行う．

4.4 ステートチャート間の整合性

ステートチャート間ではオブジェクト間のイベント通信に誤りが無いという整合性が考えられる．送信されたイベントをあるオブジェクトが受け取ることができないことを誤りが無いという．我々の形式記述ではプロセスを合成した並行プロセスが停止しないことを示せばよい．

ステートチャートの CSP 記述がプロセス P，Q のとき，並行プロセス'$P \parallel Q$' を定義する．'$P \parallel Q$' が停止しないことを調べることで整合性の検証を行う．

4.5 ステートチャートとシーケンス図間の整合性

ステートチャートとシーケンス図間では，シーケンス図におけるメッセージの実行順序がステートチャートのイベント，アクションの実行順序に含まれるという整合性が考えられる．我々の形式記述では，シーケンス図を軌跡で記述する．ステートチャートをプロセスで記述し，並行プロセスを定義する．シーケンス図から記述した軌跡が，並行プロセスの軌跡に含まれるか調べる．

シーケンス図を CSP 記述した軌跡を's' とする．シーケンス図に現れるオブジェクトのステートチャートの CSP 記述が P，Q のとき，P と Q の並行プロセスの軌跡は'traces(P||Q)' と記述できる．整合性の検証は，'s ∈ traces(P||Q)' が成り立つかを証明すればよい．

5 考察

ソフトウェア開発では分析，設計，実現の抽象度があると考え，各段階での本方法の適用についてと，UML 図への形式手法適用による効果についての考察を行う．

5.1 各抽象度における UML 図の整合性の検証

前節までは実現段階における UML 図の形式記述及び，整合性の検証を行った．本節では分析段階，設計段階の UML 図に，我々の整合性の検証方法を適用する．抽象度に対応したクラス図の VDM-SL 記述，ステートチャート，シーケンス図の CSP 記述の定義を行い，整合性の検証を行う．分析段階では，ソフトウェアの構成要素の存在定義が重要であるので，ステートチャート，シーケンス図は記述しない．ステートチャート，シーケンス図における CSP 記述は，設計段階で行う．

(a-1) 分析段階における形式記述の定義

分析段階では，クラス図を VDM-SL を用いて記述する．形式手法のソフトウェアアーキテクチャの適用に関する事例研究 [4] で提案している．

(a-2) 分析段階における整合性の検証

・クラス間の整合性の検証

分析段階のクラス図では構成要素の存在定義が重要であり，メソッドの処理内容は記述しないので，クラス間の関連については検証不要である．

(b-1) **設計段階における形式記述の定義**

・クラス図

設計段階では各クラスの属性を，モジュールの状態として定義する．メソッドを状態に対する操作で記述し，操作の処理内容は記述せず，事前条件だけを記述する．

・ステートチャート

設計段階のステートチャートは，イベント・アクションを CSP のイベントとして記述し，イベント・アクションの実行順序に着目するので，引数は記述しない．非決定的な振る舞いを記述するのでガード条件は記述しない．

・シーケンス図

設計段階のシーケンス図は，実現段階と同様に，軌跡を用いて記述する．メッセージの実行順序に着目するので，引数の記述はしない．

(b-2) **設計段階における整合性の検証**

設計段階における整合性の検証を以下に挙げる．

・クラス間の整合性の検証

設計段階のクラス図も，分析段階と同様にメソッドの処理内容は記述しないので，クラス間の関連についての検証は不要である．

・クラス図とシーケンス図間の整合性の検証

クラス図とシーケンス図間の整合性は，4.3 節の整合性と同様である．設計段階においても，実現段階と同等の整合性の検証ができる．

・ステートチャート間の整合性の検証

ステートチャート間の整合性は，4.4 節の整合性と同様である．しかし，設計段階では非決定的な振る舞いを記述するので，実現段階でもデッドロックが起きないことを検査する必要がある．

・クラスとステートチャート間の整合性の検証

クラスとステートチャート間の整合性は，4.2 節の整合性の一部である．操作と事前条件の対応は検証できるが，アクション，事後条件の対応は検証できないので，実現段階で検証する必要がある．

・ステートチャートとシーケンス図間の整合性の検証

ステートチャートとシーケンス図間では，4.5 節と同様な整合性が考えられる．設計段階におけるステートチャートの CSP 記述は，非決定的なプロセスを用いて記述しているので，軌跡を調べた場合，ソフトウェアに期待する振る舞いとは合致しない．よって，整合性の検証は不要であり，実現段階で検証する必要がある．

分析段階，設計段階では上記の整合性の検証が可能である．各抽象度ごとに検証可能な整合性を**表 1** に示す．各抽象度ごとに整合性の検証を行うことで，UML 図だけを用いた仕様に比べ，より矛盾の無い仕様が記述できると考えられる．

5.2 仕様の検証

ステートチャートの記述に CSP を適用することによる効果について考察する．CSP では，仕様としてシステムがどのように振る舞うべきか，自由変数を含んだ述語を用いて記述することができる．記述した仕様 S をプロセス P が満たかを検証することができ，'P sat S' が成り立つかを証明する．

表 1: 各抽象度における整合性

	分析段階	設計段階	実現段階
クラス間	検証不要	検証不要	クラス間のメッセージのやりとり
クラスとステートチャート間		事前条件の検証 (アクション，事後条件の検証はできない)	クラスのメソッドの定義とステートチャートの状態遷移
クラス図とシーケンス図間		実現段階と同様	クラスの静的な関連とシーケンス図のメッセージのやりとり
ステートチャート間		オブジェクト間のイベント通信 (非決定的な記述なので，実現段階でも検証が必要)	オブジェクト間のイベント通信
ステートチャートとシーケンス図間		検証不要	ステートチャートのイベント，アクションの実行順序とシーケンス図のメッセージの実行順序

実現段階と異なる検証

本研究で取り挙げたスイッチとライトの例では，ライトをつけようとする利用者は，スイッチを ON にするとライトが点灯することを保証されるように望んでいる．仕様としては以下のように記述することができる．

$$LIGHTSPEC = 0 \leq ((tr \downarrow on) - (tr \downarrow turnon) \leq 1$$

tr をプロセスの任意の軌跡とする

4.4 節で記述した並行プロセスを P として，P が仕様 LIGHTSPEC を満たすか否かを検証することができる．この場合，'P sat LIGHTSPEC' を証明すればよい．CSP 記述を行うことで，システムが仕様を満たすかを検証することができる．

6 おわりに

本研究では，取り扱う UML 図の構成要素を定義し，UML 図に対する形式記述の方法を示した．UML 図と UML 図間の整合性を示し，形式仕様記述間で整合性を検証する方法を提案した．形式手法を適用することでシステムが仕様を満たすか検証できることを確認した．

謝辞

本研究を進めるにあたり，熱心にご指導をいただいた野呂昌満教授，張漢明助教授，蜂巣吉成講師，有益なアドバイスをいただいた大学院生の熊崎敦司さん，森貴彦さん，藤原泰昌さん，後藤修平さんに深く感謝致します．

参考文献

[1] 荒木 啓ニ郎，張 漢明：プログラム仕様記述論，オーム社 (2002)

[2] C. A. R. Hoare： Communicating Sequential Processes，Prentice Hall (1992)

[3] Grady Booch, James Rumbaugh, Ivar Jacobson：The Unifined Modeling Language User Guaide, Addison Wesley Professional (1999)

[4] 橋本 佳治，菅沼 かおり：形式手法のソフトウェアアーキテクチャの適用に関する事例研究 (2003)