ソフトウェア工学 126－5
（2000．3．9）

ユースケース定義のためのメタモデルの構築

中谷 多哉子[†] 安達 隆[†††] 山浦 直人[†††]
黒田 健司[†††] 玉井 哲雄[††]

オブジェクト指向の開発では，仕様書の表記にUMLが使われている．UMLは，要求分析段階で定義するユースケースやユースケース図の記法を提示し，その意味論も提供している．しかし，ユースケースの記述に関しては明確な構造を定義していない．そのため，定義自体が煩雑な作業の繰り返しとなることも少なくない．ユースケースに関する様々な解釈や記述項目の提案がなされる中で，ユースケースの構造を形式的に定義する研究が進められている．本稿では，定型的なユースケースについて，そのフレームワークを与える．これによって，記述者による内容の差異を少なくし，生産性を向上させることができることを示す．また，ユースケースフレームワークの構成要素に基づくユースケースのメタモデルを提案する．このモデルには，複数のユースケースから参照されるオブジェクトの属性や制約条件，利用者インタフェースを，ユースケースの構成要素として定義する．これによって，要求仕様書におけるユースケース間の一貫性と整合性を保つ構造を得ることができた．

Constructing Metamodel on Use Cases

Takako NAKATANI ,[†] Takashi ADACHI ,[†††]
Naoto YAMAURA ,[†††] Kenji KURODA [†††] and Tetsuo TAMAI[††]

We use UML for object-oriented development. UML does not provide precise structure of use cases and use case diagrams defined at the requirement analysis phase, but notation for them and their semantics. For lack of the formal structure, there are many different definitions of use cases and their structure. This paper proposes use case frameworks for the typical use cases to reduce the differences between expert engineers and novice engineers and to improve productivity. After that, we define metamodel of use cases for our use case frameworks. The metamodel defines relationships between use cases and objects with attributes, constraints, and user interfaces. The objects can be accessed and shared by multiple use cases. After applying the metamodel to use case frameworks, we come to the conclusion that it is useful for helping us maintain use cases.

1. はじめに

オブジェクト指向が開発現場に導入されるようになり，システムの要求定義にユースケースやユースケース図が用いられるようになってきた．ユースケースはJacobsonの開発方法論で提案された仕様記述形式であり，オブジェクト抽出，設計，テストケースなど，すべての後工程の情報源となる[10]．ユースケースは，システムとシステム外部のアクターとの相互作用を時系列で記述するため，システムの機能を列挙するというよりも，アクターの働きかけに対するシステムの動作が定義されていると言った方がよい．その記述には自然言語が用いられており，ユーザが要求を確認するのも容易だといわれている[8]．

Jacobsonが定義したユースケースは，事前条件，事後条件，基本系列と代替系列から構成される[8),9]．この構造はユースケースの記述項目を列挙したもので，ユースケースの数の爆発や基本系列と代替系列の対応関係などを管理するには不十分である．これらの問題に対処するためには，より詳細な構造化が必要である．Jacobsonの後，Cockburn[1]やLarman[12]，あるいはSchneiderとWinters[15]，吉田ら[18]によって，より詳細なユースケースの構造が提案された．

本稿でユースケースの新しい構造を提案する目的は，ユースケース記述の生産性と品質を向上させることである．

† SLagoon
e-mail: tina@slagoon.to
†† 東京大学大学院総合文化研究科広域科学専攻
Graduate School of Arts and Sciences,
University of Tokyo
e-mail: tamai@graco.c.u-tokyo.ac.jp
††† (株) 情報技術コンソーシアム
InformationTechnology Consortium
e-mail: adachi, yamaura, kuroda@itc.co.jp

ユースケースの記述には自己完結の原則があり，ひとつのユースケース記述だけで，システムの利用者とシステムとの相互作用を理解できるようになっている．複数のユースケースから参照される部分ユースケースも，それだけで独立したユースケースとして読むことができる．UML ver.1.1 では，ユースケース間の関係として，<< *extends* >>と<< *uses* >>関係を定義する[16]．前者は基本となるユースケースと，それに新たな系列を付加したユースケースとの間の関係で，後者は全体-部分関係に相当する関係である．

我々は，ユースケースの構造を二つの視点から整理した．第一の視点では，ユースケースの汎化-特殊化関係に着目する．第二の視点では情報を共有することによって生じる制約関係に着目する．

ユースケースの汎化-特殊化関係は，ユースケースの研究者によって，抽象ユースケース，具象ユースケースという名前で議論されてきた．しかし，特に抽象ユースケースには参考にできる例が少ない．個々のユースケースを自己完結するように個別に定義する際の問題点は，同種のユースケースに対しても，同じような記述を繰り返し定義しなければならない点にある．これでは記述の生産性を向上させることはできないし，レビュー効率も悪くなる．

実際に定義されたユースケースを見ると，多くのユースケースで同じ系列を持つものが存在し，いくつかの種別に分類できることに気づく[17]．本研究では，記述の繰り返しを避けるために，共通点をユースケースの種別によって分類し，ユースケース記述の枠組みを定義する．我々は，この枠組みをユースケースフレームワークと呼ぶ．ユースケース記述者は，ユースケースの種別によって選択したユースケースフレームワークに，具体的な情報を埋め込んでアプリケーション固有のユースケースを生成する．このユースケースをアプリケーションユースケースと呼ぶ．

ユースケース間の制約関係は，複数のユースケースで情報を共有しているときに生じる．たとえば，登録ユースケースで定義したオブジェクトが，参照ユースケースや更新ユースケースによって参照される場合，参照ユースケースで定義されるオブジェクトの項目は，登録ユースケースや更新ユースケースで定義された項目から導出可能でなければならない．このような制約を互いに与え合うユースケース間の関係が制約関係である．レビューでは，制約関係にあるユースケース間で，共有されるオブジェクトの一貫性が保たれているか否かを検証するが，この一貫性は，適切な構造をユースケースに与えることによって，効率的に保つことができる．我々は，この構造をユースケースのメタモデルに導入する．

本来，ユースケースとは，要求抽出の手段として用いられるものである．このような作業により多くの労力を割くためには，ユースケース間の記述の重複を解消して生産性を向上させ，共有される情報の一貫性を確認しやすいユースケースの構造を考える必要がある．本稿では，次の解決策を提案する．

- 共通の記述が現れるユースケースに対して，ユースケースフレームワークを提供する．
- ユースケース間の制約関係をメタモデルによって明確に定義する．

ユースケースのメタモデルを定義することによって，ユースケースを XML のような構造化文書言語を用いて記述することも可能となる．また，ユースケースをデータベースに格納し，再利用することも可能となる．

本稿では，次の第 2 項でメタモデルの基となるユースケースの構造を提案し，第 3 項でユースケースの構造を解析し，メタモデルを導く．第 4 項では，メタモデルに従うユースケースの例を示し，その有効性を議論する．

2. ユースケースの構造の検討

2.1 ユースケースの構造

ユースケースの重複記述を避け，ユースケースフレームワークとの差分だけの記述と，共有可能情報の定義ができるようにユースケースを構造化した．その結果を以下に示す．

- システム名：
- ユースケース種別：ユースケースが使用したユースケースフレームワーク名
- ユースケース名：
  多様性項目：
- アクター：
  多様性項目：
- 目的：
  多様性項目：
- 事前条件：
  多様性項目：
- 正常終了における事後条件：
  多様性項目：
- 例外終了における事後条件：
  多様性項目：
- ユースケース起動動作：
  多様性項目：
- ユースケース全体にかかる代替系列：
  多様性項目：
- 基本系列：
  基本系列 1．動作説明

基本系列 2. 動作説明
....
基本系列 n. 動作説明

主な項目について，以下に説明する．

- 多様性項目 (Variations)
  多様性項目とは，アプリケーションのユースケースで固有に定義される情報の多様性を許すために設けた項目である[1)]．多様性項目にユースケースの多様な情報を定義することによって，ユースケース数が爆発するのを回避できる．多様性項目は以下の項目からなる．ただし，多様性項目識別子は他のユースケースや他の項目から一度定義した多様性項目を参照しやすいように付加した属性である．
  - 多様性項目識別子
  - 内容記述：
    利用者インタフェース，操作対象のオブジェクトの情報，システムの動作制約，利用者インタフェース，他の基本系列で定義された多様性項目識別子（他のユースケースと共有される情報）など．
- 事後条件
  事後条件が二種類定義したのは，正常終了時と例外終了時に成立するユースケースの事後条件が異なる場合もあるためである[1)]．
- 基本系列と代替系列
  代替系列は，基本系列とは別の項目で定義する構造が一般的に使われている．しかし，ほとんどの代替系列で，「基本系列の i 番目において，XXXX の場合は..」という記述が現れており，基本系列の項目と深い関わりがあることがわかる．この記述が現れない代替系列は，ユースケース全体にかかって発生し得る例外処理を表す．前者の代替系列は，基本系列の項目内に代替系列を書くという入れ子の構造を定義すると，基本系列の項目と代替系列との間の関係を明示できる．後者は，ユースケース本体に関係付けて定義することで，代替系列が発生する例外事象が及ぼす範囲を明示できる．
  基本系列は以下の項目からなる．代替系列は基本系列を構成する項目と同じ項目を持つ．
  - 多様性項目：
  - 例外事象：
  - 代替系列（基本系列番号-代替系列番号）：
- 起動動作
  起動動作には，アクターのシステムへの働きかけ以外に，タイマー事象などが含まれる[2)]．通常，基本系列はアクターのシステムへの働きかけによって起動されるが，この働きかけを基本系列から分離して「ユースケース起動動作」の項目に定義する．すなわち，ユースケースは事前条件が満たされたもとで，起動動作が起きたとき，基本系列が走り始めると読む．起動動作は重要な項目であるため，記述から抜けないよう基本系列とは別項目として設定した．

2.2 ユースケースフレームワーク

情報処理学会ソフトウェア工学研究会の要求工学ワーキンググループで採用されている共通例題「プログラム委員長業務[11)]」で定義するユースケースは，図 1に示すように，データ登録，データ更新，データ参照，データ削除，一覧表参照といった定型的ユースケースを利用して定義できる．そこで，これらの定型的なユースケースに対して，アクターとシステムとの相互作用の流れ，代替系列，多様性項目などの記述の枠組みを与える．この枠組みをユースケースフレームワークと呼ぶ．これに対して，アプリケーション固有の情報を持つユースケースをアプリケーションユースケースと呼ぶ．

ユースケースフレームワークは，ユースケースのインスタンスであるから，先に示したユースケースの構造を持っている．しかし，部分的に具体的な情報が抽象化されている．アプリケーションユースケースは，この部分に具体的な情報を定義する．

オブジェクトを登録する登録ユースケースフレームワークの例を以下に示す．ゴチック文字で書かれた部分がアプリケーションユースケースで具体化する部分である．

- システム名：**システム名**
- ユースケース種別：登録ユースケースフレームワーク
- ユースケース名：**登録対象オブジェクト**登録ユースケース
  多様性項目：**登録対象オブジェクト**
- アクター：**オブジェクト登録希望者**
  多様性項目：**オブジェクト登録希望者**
- ゴール：**登録対象オブジェクト**を登録する
- 事前条件：アクターが**登録に必要な情報**を得ている
  多様性項目：**登録に必要な情報**
- 正常終了における事後条件：**登録対象オブジェクト**が登録済みとなったという結果をアクターが得ている
- 例外終了における事後条件：**登録対象オブジェクト**の登録が中止され，副作用が残っていない
- ユースケース起動動作：アクターはシステムに**登**

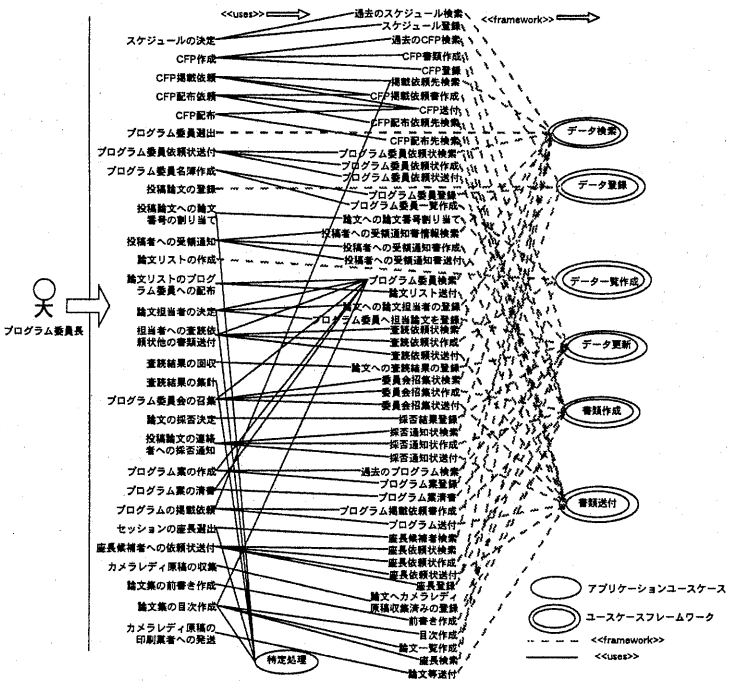

図 1 プログラム委員長業務のユースケース図とユースケースフレームワーク
Fig. 1 Use cases for a program chair and use case frameworks

録対象オブジェクトの登録希望を通知する

- ユースケース全体にかかる代替系列：
  アクターが登録希望を取りやめたときは，**登録対象オブジェクト**の登録を中止し，副作用を残さない
- 基本系列：
  1. システムはアクターに**登録対象オブジェクト**を登録するための**登録インタフェース**を提示する．
  多様性項目：**登録インタフェース**
  :::代替系列 1-1: **登録インタフェースが提示されない**
  2. アクターが入手した**登録インタフェース**を介してシステムに**登録対象オブジェクト**の登録に**必要な情報**を渡すと，システムは渡されたデータが正当であり，かつ既に同じデータが登録されていないことを確認し，**登録対象オブジェクト**を**永続化**する．
  多様性項目：
  データが正当である条件
  永続化
  :::代替系列 2-1: 正当なデータと判定できなかった場合は次のことを行う．
  :::::2e1-1. システムはデータが不当である理由を明らかにする．
  :::::2e1-2. システムはアクターにデータの不当個所と不当理由を通知する．
  多様性項目：
  データの不当個所と不当理由を通知するインタフェース
  :::::2e1-3. 基本系列 2 を繰り返す．
  :::代替系列 2-2: 登録しようとした**登録対象オブジェクト**が既に登録済みとなっている場合は，次のことを行う．
  :::::2e2-1. システムは，アクターに登録希望された**登録対象オブジェクト**が登録済みとなっていることを通知する．
  多様性項目：
  登録済みとなっていることを通知するインタフェース
  :::::2e2-2. 基本系列 2 を繰り返す．
  3. システムはアクターに**登録対象オブジェクト**が新たに登録されたことを通知する．
  多様性項目：
  登録対象オブジェクトが新たに登録されたことを通知するインタフェース
  :::代替系列 3-1: **通知に失敗した**
  4. アクターは，**登録対象オブジェクト**が登録されたことを確認する．
  多様性項目：

登録対象オブジェクトが登録されたことを確認するインタフェース

:::代替系列 4-1: **確認に失敗した**

ユースケースフレームワークを与えることによって，ユースケースの記述量が減り，生産性を高めることができる．また，レビュー関係者は，まず，ユースケースフレームワークの妥当性を検証し，その後で，ゴチック文字で書かれた部分を，個別のアプリケーションユースケースで検証すればレビューの効率も向上する．

ユースケースフレームワークを利用する場合，新しいユースケースを記述する前に，定義しようとするユースケースがどのユースケースフレームワークに具体的な情報を与えれば生成できるかを見極めなければならない．しかし，この労力は，すべてのユースケースを新たに記述する労力に比べれば大したことにはならない．ユースケースフレームワークは，プログラム委員長業務の例では 6 種類しかない．実システムでは 4 酒類であった[17]，この程度の数であれば，使うべきユースケースフレームワークを直感的に抽出するのは容易である．

2.3 ユースケース事例

登録ユースケースフレームワークに準拠して,「プログラム委員登録ユースケース」を定義する．最初に開発者が定義する事項を示し，その後，レビュー関係者が参照するアプリケーションユースケースを示す．記述量は減っているが，レビューに必要な情報は網羅されている．アプリケーションユースケースの表示方法は，様々な方法が考えられるが，これはユースケースのプレゼンテーションの問題であり，ここでは議論しない．

（開発者が定義する事項）

- システム名：**プログラム委員長業務**
- ユースケース種別：登録ユースケースフレームワーク
  - 登録対象オブジェクト=**プログラム委員**
  - オブジェクト登録希望者=**プログラム委員長**
  - 登録に必要な情報=**プログラム委員名（文字列），所属（文字列），役職（文字列）*，連絡先（文字列），専門分野（文字列），関係国（文字列）**(***はオプション**)
  - 登録インタフェース=**未定**
  - :::代替系列 1-1: **登録インタフェース**が提示されない=**考慮しない**
  - データが正当である条件= **関係国はすでに登録されている国名とする．また，重複して同じ人を登録しない．データの同一性は，名前と連絡先によって判断する．**
  - 永続化=XX **データベースへ登録する**
  - データの不当個所と不当理由を通知するインタフェース=**未定**
  - 登録済みとなっていることを通知するインタフェース=**未定**
  - 登録対象オブジェクトが新たに登録されたことを通知するインタフェース=**未定**
  - :::代替系列 3-1: 通知に失敗した=**通知の失敗を考慮しない**
  - **登録対象オブジェクトが登録されたことを確認するインタフェース=未定**
  - :::代替系列 4-1: 確認に失敗した=**通知の失敗を考慮しない**

（レビュー関係者が参照するユースケース）

- システム名：**プログラム委員長業務**
- ユースケース種別：登録ユースケースフレームワーク
- ユースケース名：**プログラム委員**登録ユースケース
- アクター：**プログラム委員長**
- ゴール：**プログラム委員**を登録する
- 事前条件：アクターが**プログラム委員名（文字列），所属（文字列），役職（文字列）*，連絡先（文字列），専門分野（文字列），関係国（文字列）**(***はオプション**) を得ている
- 正常終了における事後条件：**プログラム委員**が登録済みとなったという結果をアクターが得ている
- 例外終了における事後条件：**プログラム委員**の登録が中止され，副作用が残っていない
- ユースケース起動動作：アクターはシステムに登録対象の登録希望を通知する
- ユースケース全体にかかる代替系列：アクターが登録希望を取りやめたときは，**プログラム委員**の登録を中止し，副作用を残さない
- 基本系列：

  1. システムはアクターに **プログラム委員**を登録するための登録インタフェース**未定**を提示する．

  ::代替系列 1-1:登録インタフェース**未定**が提示されない=**考慮しない**

  2a. データが正当であるための条件：**関係国はすでに登録されている国名とする．また，重複して同じ人を登録しない．データの同一性は，名前と連絡先によって判断する．** … 以下省略

3. メタモデルの構築

3.1 UML によるユースケースのメタモデル

UML が与えているユースケースのメタモデルを図

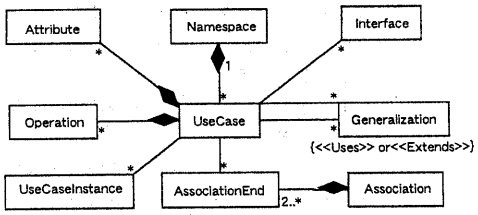

図 2 UML によるユースケースのメタモデル
Fig. 2 Use cases for Program Chair and Use case frameworks

2に UML. ver.1.1 に示す.

この図には，ユースケースが<< *Uses* >>，<< *Extends* >>という二つのステレオタイプによって関係を定義できること，複数の利用者インタフェースがユースケースで定義されることが示されている．UML では，黒く塗りつぶした菱形は，合成を表し，他のオブジェクトから参照されることはないことを意味する．

このメタモデルでは，Operation や Attribute の構造を定義していないため，我々のユースケースの構造で示したような系列の入れ子関係を比較することはできない．また，これらの情報との関係が合成によって定義されていることから，複数のユースケースが参照する属性が制約関係を生じさせていることも考慮されていないことがわかる．

3.2 ユースケースの構造解析

図 1を注意深く読むと，プログラム委員登録アプリケーションユースケースで定義される**プログラム委員**は，次のアプリケーションユースケースからも参照され，更新される．

- プログラム委員選出
- プログラム委員検索
- プログラム委員一覧作成
- プログラム委員検索
- プログラム委員へ担当論文を登録
- 委員会招集状作成：召集状には委員の名前や所属部署が記述される
- 委員会招集状送付：宛名には委員の住所が記述される
- 座長候補者検索：座長はプログラム委員の専門分野とセッションの内容によって決定される
- 座長依頼状作成：招集状作成と同様
- 座長依頼状送付：招集状送付と同様
- 座長登録：プログラム委員の名簿にどのセッションの座長を担当するかが登録される
- 座長検索：特定のセッションの座長を担当するプログラム委員の検索

プログラム委員の登録アプリケーションユースケースには，プログラム委員の登録項目が定義される．各項目はプログラム委員というクラスの属性となる可能性が高い．しかし，後で論文の査読を依頼したり，座長を検索する場面で該当者を検索するとき，十分な情報が登録されているとは限らない．そのため，新たな属性が他のユースケースで発見されたとき，プログラム委員の属性項目が追加されたとき，登録アプリケーションユースケースで参照するプログラム委員の属性項目も更新されることはない．

UML では，特定のオブジェクトに対する相互作用を記述したユースケースは，パッケージを用いてグループ化することを推奨している．しかし，まだオブジェクトが抽出されていない分析段階で，オブジェクトを中心としたユースケースのカプセル化を行うのには無理がある．我々のメタモデルには，論理的なオブジェクトとユースケースとの間に関連を与え，ユースケース間の制約関係を明示する．この制約関係は，論理的なオブジェクトの構造が変化したとき，その影響を関係するユースケースに伝播する機構となる．

利用者インタフェースでも同様の議論が成り立つ．ある利用者インタフェースが他のユースケースでも流用可能なものであれば，同じインタフェースにした方がよい．UML のメタモデルでは，Interface と UseCase の間には 1 対多の関係が定義されているが，共有や流用を考慮すると，利用者インタフェースとユースケースの間には，多対多の関連が存在する．利用者インタフェースもまた，それを参照するユースケースに制約を与える．

3.3 新しいメタモデル

アプリケーションユースケースは，その記述の枠組みとしてユースケースフレームワークを参照する．この参照関係は，ユースケースフレームワークが抽象化している箇所に対して，具体的な情報を与えるという関係である．具体的な情報の型はユースケースフレームワークの中で規定される．

ユースケースの構造で定義したメタモデルの構成要素に型を与えた構造を示す．

- システム名：システム
- ユースケース種別：フレームワーク
- ユースケース名：文字列
- アクター：
  多様性項目：内部オブジェクト
- 目的：
  多様性項目：文字列
- 事前条件：
  多様性項目：条件
- 正常終了における事後条件：
  多様性項目：条件
- 例外終了における事後条件：

多様性項目：内部オブジェクト

- ユースケース起動動作：
  多様性項目：内部オブジェクト
- ユースケース全体にかかる代替系列：
  多様性項目：系列
- 基本系列：契約の時系列集合
  基本系列 1. 動作説明：契約
  基本系列 2. 動作説明：契約
  ....
  基本系列 n. 動作説明：契約

契約は，事前条件と事後条件を定義できるプログラムを指し[13]，例外事象によって関連づけられる契約を集約した入れ子構造を持つ．ユースケースの基本系列と代替系列は契約の時系列集合である．契約には契約者と被契約者を定義できる[5]．ユースケースもまた，契約の一種である．ただし，ユースケースには，新しい属性として，起動動作と多様性項目を定義する．ユースケースでは，契約者をアクター，被契約者をシステムと呼ぶ．アクターやシステムには，開発対象のシステムそのものや外部のシステム，外部のオブジェクト，コンポーネントのいずれも該当することができる．

図 3にユースケースメタモデルの構造を UML のクラス図の表記に準拠して示す．

4. 考　　察

Collins-Cope は，ユースケースを<< 要求 >>，<< インタフェース >>，<< サービス >>に分類した[4]．<< インタフェース >>ユースケースとは，システムのインタフェースとアクターとの相互作用を記述したユースケースであり，<< サービス >>ユースケースは，<< インタフェース >>ユースケースから呼び出されるユースケースである．システム内部のオブジェクト間の相互作用もこれで記述される．<< 要求 >>ユースケースはシステム外部のアクターとシステムとの相互作用を記述したユースケースであるから，このユースケースは，<< インタフェース >>ユースケースと<< サービス >>ユースケースに分割できる．

要求分析では，Collins-Cope の言う<< 要求 >>ユースケースの記述やアクティビティの解析といった問題領域分析により多くの時間をかけるべきである．我々は，問題領域分析以降の要求の詳細化の生産性向上と品質向上を目指し，ユースケースフレームワークとメタモデルを提案した．ユースケースの品質を向上させるのは，レビューをいかに効率よく進めるかに依存している．ユースケースを契約として見ることによって，クリーンルーム手法のレビューも適用できる[6]．

我々のメタモデルでは，ユースケースの<< *uses* >>関係とを，ユースケースに定義される系列を契約というオブジェクトの入れ子構造で表した．これによって，Collins-Cope のユースケースや Jacobson のユースケースの構造を網羅でき，さらに Coleman が提案している連接，分岐，反復構造を持った系列を記述することも可能である[3]．

定型的なユースケースフレームワークは実際にどの程度有効であろうか．プログラム委員長業務の例題を解いたユースケース図からわかるように，実際のシステムで定義されるユースケースの多くは，定型的なユースケースを組み合わせて作ることができる．ユースケースを契約の一種とすることで，段階的に詳細化しなければ記述できないような複雑で大規模なユースケースも，本稿で提示したメタモデルに従って記述できる．最近のオブジェクト指向のパターンに関する議論[7),14)]を見習えば，将来，このような複雑な構造を持つユースケースに，定型的な構造を割り当てるためのユースケースパターンを発見できる可能性は高い．ユースケースパターンを定義できるようになると，さらに要求分析の作業は問題領域分析に時間を割くことができるようになるだろう．

本稿で定義したユースケースフレームワークに従ってユースケースを定義すれば，ユースケースのプレゼンテーションを自由に変えることも可能である．今後は，メタモデルに基づき，ユースケースフレームワークを使ったユースケース記述とレビューの支援システムを開発していきたい．

5. ま と め

本稿では，ユースケースに明確な構造を定義し，定型的なユースケースについて，そのユースケースフレームワークを与えた．ユースケースフレームワークを用いることによって，アプリケーションユースケースの記述に関する生産性を向上させることができる．また，ユースケースの構造に基づき，メタモデルを提案した．このメタモデルには，複数のユースケース間の制約関係が表現されている点で，UML が与えたメタモデルよりも強力である．メタモデルにユースケース間の制約関係を定義したことによって，要求仕様書におけるユースケース間の一貫性と整合性を保つ構造を得ることができた．

参 考 文 献

1) Cockburn, A. : “Structuring Use Cases with Goals,” http://members. aol.com/acockburn /papers/usecases.htm.
2) Cockburn, A. : “Basic Use Case Template,” http://members. aol.com/acockburn

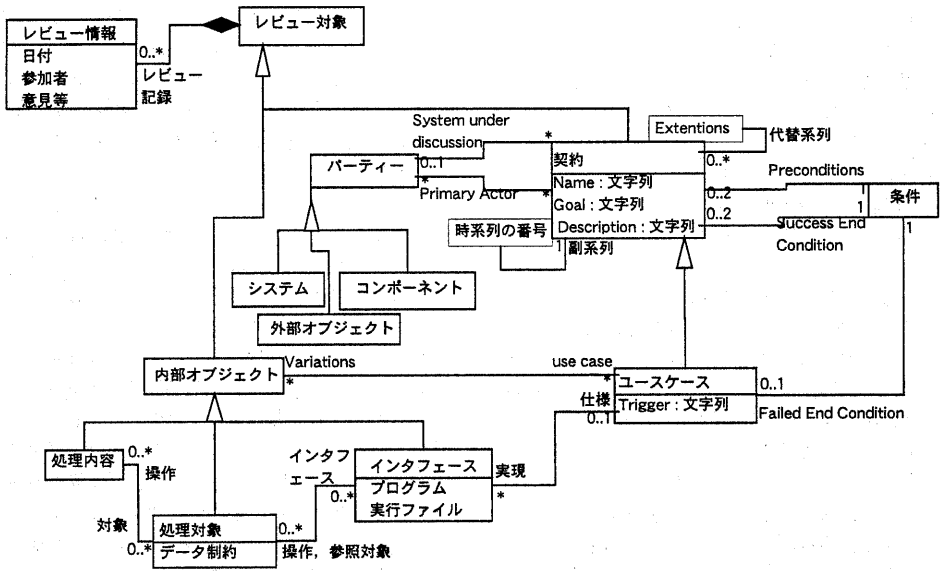

図 3 ユースケースのメタモデル
Fig. 3 Use case metamodel for Use case

/papers/uctempla.htm.

3) Coleman, D. : "A Use Case Template: draft discussion," *EDOC'99 (Tutorial material)*, May,1999.

4) Collins-Cope, M. : "The RSI Approach to Use Case Analysis," *C++ REPORT*, July-August 1990.

5) Fowler, M. : *Analysis Patterns: Reusable Object Models*, Addison-Wesley, 1997. (堀内一監訳, アナリシスパターン, アジソンウェスレイ, 1998.)

6) 二木厚吉監修 : ソフトウェアクリーンルーム手法, 日科技連, 1997.

7) Gamma, E. et al. : *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1994.

8) Jacobson, I., Christerson, M., Jonsson, P. and Overgaard, G. : *Object-Oriented Software Engineering: A Use Case Driven Approach*, Addison-Wesley, 1992.

9) Jacobson, I., Ericsson, M. and Jacobson, A. : *The Object Advantage*, Addison-Wesley, 1994.

10) Jacobson, I. , Booch, G. and Rumbaugh, J. : *The Unified Software Development Process*, Addison-Wesley, 1999.

11) 情報処理学会ソフトウェア工学研究会 要求工学ワーキンググループ共通問題 : http://www.selab.cs.ritsumei.ac.jp/ohnishi/RE/problem.html.

12) Larman, C. : *Applying UML and Patterns: an Introduction to Object-Oriented Analysis and Design*, Prentice Hall, 1998. (今野睦, 依田智夫監訳, 実践 UML, プレンティスホール, 1998.)

13) Meyer, B. : *Object-Oriented Software Construction, second edision*, Prentice Hall, 1997.

14) 中谷多哉子, 青山幹雄, 佐藤啓太編著 : ソフトウェアパターン (pp.42–43), bit 別冊, 共立出版, 1999.

15) Schneider, G. and Winters, J. P. : *Applying Use Cases*, Addison-Wesley, 1998.

16) UML Semantics version 1.1, Rational Software Corporation, September 1997. http://www.rational.com/uml/index.html

17) 山浦直人, 安達隆, 黒田健司, 中谷多哉子 : "上流工程における再利用部品としてのユースケース", 情報処理学会第 60 回全国大会論文集, 2000 (印刷中) .

18) 吉田裕之, 山本里枝子, 上原忠弘, 田中達雄 : UML によるオブジェクト指向開発実践ガイド, 技術評論社, 1999.