ユースケースフレームワークの検討

中谷　多哉子[†]　玉　井　哲　雄[††]

要求定義の道具としてユースケースが用いられるようになってきた．ユースケースは自然言語で記述されるため，理解が容易な反面，開発者にとっては，定義自体が煩雑な作業の繰り返しとなっている．特にビジネス系システムのユースケースでは，同様の系列を持ったユースケースが多数定義される傾向にあり，生産性と品質を向上させるために，明確な構造を決定する指針が必要となってきた．本稿では，ユースケースの構造を検討した後，ビジネス系システム用の定型的なユースケースに対して枠組みを与え，それをユースケースフレームワークとして定義する．ユースケースフレームワークを適用することによって，アプリケーションユースケースの構造を統一できただけでなく，記述する際に考慮すべき事項を明示的に示すことができるようになった．

The Use of Use Case Framework

Takako NAKATANI [†] and Tetsuo TAMAI[††]

We use use cases as a tool to define user requirements. As use cases are defined in natural language, the defining process itself tends to be a complicated work and make participants have patience with reading similar use cases many times. In business systems, there are many use cases with similar contents. In this paper, we propose a use case framework as a typical use case structure defined for business systems. The use case frameworks provide selective points of view under consideration when developers define the use cases.

1. はじめに

オブジェクト指向システムの要求定義ではユースケースやユースケース図が用いられる．ユースケースとは，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仕様記述形式であり，Jacobson の開発方法論で提案された．また，ユースケースはオブジェクト抽出，設計，テストケースなど，すべての後工程の情報源でもある[7]．その記述には自然言語が用いられるため，ユーザが要求を確認するのも容易だといわれている[5]．

実際に記述されたユースケースを調査してみると，多くのユースケースで基本系列の流れや代替系列に重複を発見することができる．ユースケースの記述の重複は，後で読み直し，同じ記述を発見して部分ユースケースにまとめ，関係するユースケースを書き直すといった作業を繰り返して解消するが，このような作業には多くの労力が必要である．

我々は，ユースケースの記述の重複を解消させるために，アクターとシステムとの間の相互作用，処理対象やインタフェースをユースケースの構成要素として抽出し，ユースケースの構造化を進め，定型的なユースケースの記述の枠組みを与える研究を行った．この枠組みをユースケースフレームワークと呼ぶ．アプリケーションシステムに固有なユースケースは，ユースケースフレームワークに対して個別の情報を差分として定義する．こうして定義されるユースケースをアプリケーションユースケースと名付ける．

本研究は，要求抽出により多くの時間を割けるような環境を整備することであるが，そのためには，まず，ユースケースの記述自体を統一させるような枠組みが必要である．枠組みがあることによって，ユースケースの記述の生産性が向上し，レビューの効率も向上させることができるようになる．このような環境が提供されれば，これまでよりも多くの時間を要求抽出の時間に当てることができるようになるだろう．本稿では，ユースケースの記述の枠組みを紹介し，その有効性について議論を行う．

† 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

本稿は，以下の構成になっている．次の第 2 節で，Jacobson によるユースケースの定義を示し，これまでに提案されたユースケースの構造を紹介する．第 3 節でユースケースの構造を，重複情報の削除という観点から再検討し，ユースケースフレームワークを提示する．第 4 節では，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によって採用されている共通問題「プログラム委員長業務[8]」に対して，ユースケースフレームワークを適用し，その有効性を検証する．

2. ユースケースとは

2.1 定　　義

UML にもユースケースとユースケース図が定義されているが，その厳密な書き方を規定しているわけではない．UML の意味論によると，ユースケースは，“状態機械や事前条件・事後条件などの振る舞いを記述する技術を用いて，アクティビティ図の操作を自然言語によって記述したものである” と定義されている[13]．このようなユースケースには，様々な分類がある．

たとえば，Jacobson は，ユースケースを次のように分類した[5]．

抽象ユースケース　複数のユースケース間での共通部分を記述するためにだけ意味を持つユースケース．

具象ユースケース　実際にインスタンス（シナリオ）を持つユースケース．

これらのユースケースの間には次の関係があり，UML では，逐次新しい関係の提案がなされている．

拡張関係　ひとつのユースケースの記述が他のユースケースにどのように組み込まれ，拡張されるのかを定義した関係．

利用関係　あるユースケースが他のユースケースによって利用されるという関係．たとえば，ユースケースのインスタンスが具象ユースケースの記述にしたがって動作し，ある時点から抽象ユースケースの記述にしたがって動作を継続することを表す．

ユースケース図には，これらの要素と要素間の関係を視覚的に表現することができる．ユースケース図は，要求の全体像を俯瞰するには便利であるが，要求抽出を行う道具には適さない．

2.2 主なユースケースの構造

以下に，これまでに提案された主なユースケースの構造を紹介する．

2.2.1 基本的な構造

Jacobson のユースケースは次の要素によって構成されている[5]．

- 名前
- アクター
- 事前条件
- 基本系列
- 代替系列
- 事後条件

2.2.2 系列の記述に関する提案

Larman は，基本系列と代替系列に，プライマリ，セカンダリ，オプショナルという開発の優先順位付けを行った[9]．このような分類は，Schneider と Winters によるユースケースでも採用されている[12]．

系列の優先順位付けを行うという発想から，基本系列に判断による条件分岐を定義するか否かにも賛否が議論がある．Schneider と Winters は，ユーザのわかりやすさを優先すべきであるとの立場からは条件分岐や繰り返しの記述を避けることを推奨しており，基本系列が開発の最優先の要求であるとしている．しかし，特に分岐が発生する個所で，どちらも同じように発生する可能性がある場合には，分岐の記述を基本系列に許可している[12]．

開発の優先順位やわかりやすさという立場がある一方，Coleman のユースケーステンプレートのように，基本系列に条件分岐と繰り返しの構文を明示的に導入した例もある[3]．

Cockburn が提案したユースケーステンプレート[2]の特徴は，責任駆動型のオブジェクト指向方法論を基本系列の記述に取り入れた点にある．システムがプライマリアクターに対して果たさなければならない責任はシステムのゴールとして設定され，システムからセカンダリアクターに対してメッセージを送って果たされる．また，系列の各項目には，＜時間要素，判断＞＜アクター＞＜動作＞＜制約＞を明記する．例えば，次のように記述する．

＜論文投稿の締め切り日が過ぎてから＞
　＜プログラム委員長は＞
　　＜担当委員を決める．＞
　　　＜担当委員は 1 論文につき 3 名で,… ＞

さらに，Cockburn は，オブジェクトや操作の多様性を明記する多様性項目を追加し，ユースケースの数の爆発を防止する方法も提案した[1]．

2.2.3 代替系列の記述に関する提案

吉田らのユースケースは，以下の構造を採用し

ている[15].

- ユースケースの名前と管理用 ID
- 概要 (Summary)
- アクター (Actors)
- 前提条件 (Preconditions)
- 処理 (Descriptions)
- 例外事項 (Exceptions)
- 後条件 (Postconditions)

例外事項には，代替系列の記述の指針として，入力の誤り，入力データ間の矛盾，オブジェクトの属性値自体の問題，属性値間の矛盾，入力データと属性値間の矛盾に起因する事項を挙げている．

3. ユースケースフレームワークの検討

3.1 従来のユースケース記述

Jacobson の基本的な構造に基づいてユースケースを記述した例を示す．

- システム名：プログラム委員長業務
- ユースケース名：プログラム委員登録ユースケース
- 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を登録するために必要な情報を得ている
- 事後条件：プログラム委員が登録済みとなる
- 基本系列：
  1. アクターはシステムにプログラム委員を登録する旨を通知する．
  2. 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情報を要求する．
  3. アクターはシステムにプログラム委員の登録に必要な情報を渡す．
  4. システムは渡されたデータが正当である条件を満たし，かつ既に同じデータが登録されていないことを確認し，プログラム委員を登録する．

  プログラム委員登録に必要な情報とその情報の制約は以下のとおりである．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)．ただし，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．
- 代替系列：
  - (1) 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
  - (2) 基本系列 4. において，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．
    - (a) システムはデータが不当である理由を明らかにし，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．
    - (b) 基本系列 2–4 を繰り返す．
  - (3) 基本系列 4. において，登録しようとしたプログラム委員が既に登録済みとなっている場合は次のことを行う．
    - (a) システムは，アクターに登録希望されたプログラム委員が登録済みとなっていることを通知する．
    - (b) 基本系列 2–4 を繰り返す．

このようなユースケースには，代替系列と基本系列との間の関係が明示されているが，ユースケースの構造として，両者の関係を管理する構造はない．また，ここに示した記述は，最初から記述するか，ソースレベルの切り貼りによって生成される．

明確な構造を持たないユースケースが持つ問題点をまとめると以下のようになる．

- 代替系列に，基本系列に対応する代替系列とユースケース全体にかかる代替系列とが混在する．基本系列の項目番号に変更があると，それに伴って代替系列に対応する基本系列の項目番号を修正しなければならない．
- 系列を構成する項目の粒度が不明確である．たとえば，代替系列で指定された繰り返しの単位から，基本系列 2–4 はひとつの項目として記述されるべきであるが，システムの動作単位で項目が記述されている．記述者の判断によって様々な粒度のユースケースが定義されることによって，レビューが煩雑となる．
- 要求に関するすべての事項がユースケースだけで参照できるわけではなく，最終的なテストケースを生成する材料としては不完全である．たとえば，利用者インターフェースの定義は後の開発工程で決定されるが，このユースケースを参照しただけでは，インタフェースが決定されているのか，まだ決定されていないのかがわからない．定義されているとしても，どこに定義されているのかがわからない．
- ユースケース間で共有される情報の一貫性を保つのが困難である．ここで定義したプログラム委員のデータ項目は，このユースケースの中に埋没しており，他のユースケースから参照するには手作業に頼るしかない．たとえば，このデータ項目がどのユースケースから参照されるかを管理するのは困難であり，データ項目の一貫性を保持するのも困難である．

これらの問題を解決するために，ユースケースの構造を解析する．

3.2 ユースケースの構造

記述の重複部分が発生する要因として，以下の事項を考えることができる．

- 同様の目的を達成するために記述されたユースケースにおいて，同様の事前条件，事後条件が定義される傾向がある．
- アクターとシステムとの相互作用は，同様の

目的であれば，同様の順番で定義される傾向がある．

- 特定の基本系列項目に対して，同様の代替系列が定義される傾向がある．
- 特定の処理に対しては同様の利用者インタフェースが開発される傾向がある．
- 複数のユースケースの処理対象が同じオブジェクトである場合，オブジェクトに関する記述が繰り返し説明される傾向がある．

特にビジネス系のシステムでは，データ登録／削除／更新／検索といった処理に関する要求が多い．そのため，ユースケースの記述にも重複が多く発生しやすいと考えられる．

我々は，複数のユースケースに現れる記述の重複を減らすという視点から，ユースケースの構造を検討する．ユースケースの重複を解消する作業はユースケース記述者（Use-Case Specifier）が行わなければならない[7)]と言われているが，定型的なユースケースであれば，その記述の枠組みを提供することによって，重複する記述を枠組みの中に定義できる．個々のユースケースの差分は，Cockburn が提案した多様性項目として後から付加すればよい．

枠組みの構造は，例えば基本系列に対しては，処理対象の関連データ項目，例外事象と代替系列の組み，利用者インタフェースを構成要素として与えることができる．以下にユースケースの構造を提案する．

- システム名：
- ユースケース種別：
- ユースケース名：
- アクター：
- 目的：
- 事前条件：
- 正常終了における事後条件：
- 例外終了における事後条件：
- ユースケース起動動作：
- ユースケース全体にかかる代替系列：
- 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

ユースケース名以降の各項目では，具体的なユースケース固有の情報をユースケースの多様性項目として定義する．基本系列では，以下の多様性項目を考えることができる．

- 利用者インタフェース，オブジェクト構造と制約，システムの動作制約，他の基本系列で予め定義するように指示された多様性項目など．
- 代替系列（基本系列番号-代替系列番号）：代替系列が起動するための例外事象

ユースケース種別には，ユースケースが記述の枠組みとして使用した枠組み名を定義する．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，事後条件を正常終了における事後条件と例外終了における事後条件の二種類に分けた[1)]．代替系列は基本系列を構成する項目と同じ項目を持つ．また，代替系列には，基本系列の項目に関連し，基本系列の記述の中に定義されるものと，ユースケース全体にかかるものとがある．

3.3 ユースケースフレームワーク

ユースケースフレームワークとは，アクターとシステムとの定型的な相互作用の流れ，例外事象と例外時の相互作用の流れ，処理対象のデータ項目，利用者インタフェースを構成要素として構造化したユースケースである．これに対してアプリケーションユースケースとは，個々のアプリケーションに固有の情報をユースケースフレームワークに与えて定義されたユースケースである．

先に提示したユースケースの構造に準拠して登録ユースケースフレームワークを記述した例を以下に示す．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多様性項目である．ただし，二度目以降の出現ではゴチック文字による表示を行っていない．

- 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
- アクター：**オブジェクト 登録希望者**
- 目的：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
- 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録対象オブジェクトの登録希望を**サービス要求インタフェース**を介して通知する
- 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．
  :::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**データ制約**を満たし，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．
:::代替系列 2-1: データが正当である条件を満たしていなかった場合は次のことを行う．
:::::2-1-1. システムはデータが不当である理由を明らかにする．
:::::2-1-2. システムはアクターにデータの不当個所と不当理由を**データ不当個所／不当理由通知インタフェース**を介して通知する．
:::::2-1-3. 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．
:::::2-2-1. 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**重複登録不許可通知インタフェース**を介して通知する．
:::::2-2-2. 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**登録対象オブジェクト 登録完了通知インタフェース**を介して通知する．
:::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．
:::代替系列 4-1: **確認に失敗した**

3.4 期待される効果

アクターとシステムとの相互作用は，同じユースケースフレームワークを具体化して定義したすべてのユースケースで統一される．異なる相互作用を持つユースケースは，標準的なユースケースフレームワークを拡張して新しいユースケースフレームワークを記述することもできる．ユースケースフレームワークを提供することによって期待される効果を次にまとめた．

- アプリケーションユースケースの構造を統一できる．
- 基本系列の定型的な流れをフレームワークに定義できるため，システムの動作に一貫性を持たせることが容易となる．
- システムとアクターの間で行われる同様の相互作用において，必要となる利用者インタフェースを統一でき，インタフェースのユースケースとともに，再利用部品として明示できる．
- アプリケーションユースケースで重複する部分を記述する必要がなくなり，ユースケースの記述の生産性が向上する．
- 個々のユースケースの差分は，明示的に示された多様性項目によって明記され，関係者によって重点的にレビューされるべき箇所の可視性が高まる．

4. 適　　用

4.1 適用プロセス

ユースケースフレームワークの適用プロセスは，Schneider と Winters のユースケース記述手順[12]を拡張させることで定義できる．アプリケーションの開発は，段階的な詳細化によって行われる．そこで，この詳細化のプロセスに対応させ，最初に定義されるユースケースを複合ユースケース，複合ユースケースを分割し，最も詳細化された粒度の小さいユースケースを基本ユースケースと定義する．現状では，単機能のユースケースフレームワークを適用対象としているため，アプリケーションユースケースは基本ユースケースの一種となる．

ユースケースフレームワークの適用プロセスは以下のようになる．

(1) アクターのアクティビティ図[13]を記述し，各アクティビティ毎にシステムの開発対象となるか否かを識別し，要求を抽出する．
(2) 開発対象となるアクティビティについて，自由な形式でユースケースを記述する．
(3) 自由な形式で記述したユースケースを，基本的なユースケースを組み合わせて複合ユースケースの形に再構成する．
(4) 複合ユースケースのレビューを行い，要求抽出／確認を行う．
(5) 複合ユースケースを構成する基本ユースケースについて，その内容を検討する．もし基本ユースケースにユースケースフレームワークを適用できるならば，ユースケースフレームワークのレビューを行い，フレームワークの妥当性を確認する．
(6) ユースケースフレームワークを適用できる基本ユースケースを，アプリケーションユースケースとして生成する．
(7) アプリケーションユースケースに固有な情報をレビューし，要求を確認する．
(8) ユースケースフレームワークを適用できない基本ユースケースを定義し，レビューする．

相互作用の系列を確認したいと考えるユーザは，ユースケースフレームワークに定義された相互作用の系列が妥当か否かをレビューすればよい．また，記述者は，ユースケースフレームワークを選択し，差分情報だけを記述することでアプリケーションユースケースを記述できる．

表 1 ユースケースフレームワークの効果

複合ユースケース数	29
基本ユースケース数	58
アプリケーションユースケース数	53
ユースケースフレームワーク数	6

このように，ユースケースフレームワークを導入することによって，ユースケースを記述する生産性の向上とレビューの効率の向上を実現できる．複合ユースケースに対してフレームワークを定義できれば，上記のプロセスから，複合ユースケースを基本ユースケースへ分割するプロセスを省略できる．

4.2 共通例題の適用

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[8]」を例に，ユースケースフレームワークの効果を検証した．

プログラム委員長業務に定義されたユースケース図を図 1に示す．UML のユースケース図にユースケースフレームワークの表記を追加した．二重楕円がユースケースフレームワークを表す．アプリケーションユースケースは複数のユースケースを利用することができる．ここでは，データの登録／検索／更新／一覧表作成／書類作成／書類送付という 6 種類のユースケースフレームワークを定義し，それを具体化して定義されるアプリケーションユースケースとの関係を点線で示した．実線は，利用関係を表す．

図 1からわかるように，58 の基本ユースケースのうち，ユースケースフレームワークを適用できたアプリケーションユースケースは 53 ある．ユースケースフレームワークの適用結果を表 1に示す．

以下にプログラム委員の登録アプリケーションユースケースの例を示す．最初に開発者が定義する事項を示す．

（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**
- 多様性項目
  - 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)
  - サービス要求インタフェース=**未定**
  - 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**
  - データ制約= **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こと**
  - 永続化=XX **データベースへ登録する**
  - データ不当個所／不当理由通知インタフェース=**未定**
  - 重複登録不許可通知インタフェース=**未定**
  - 登録対象オブジェクト登録完了通知インタフェース=**未定**
  - :::代替系列 3-1: 通知に失敗した=**通知の失敗は考慮しない**
  - 登録対象オブジェクト登録確認インタフェース=**未定**
  - :::代替系列 4-1: 確認に失敗した=**通知の失敗は考慮しない**

次に，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．ゴチック文字で表示された部分が，フレームワークのキーワードに対して与えられた多様性項目の情報である．

（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**
- 目的：**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**サービス要求インタフェース**を介して通知する
- 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**を提示する．
  ::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**
  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**プログラム委員**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**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こと**を満たし.. 以下省略

4.3 事例による検証

本ユースケースフレームワークは，(株) 情報技術コンソーシアムが情報処理振興事業協会より先進的情報処理技術促進に係わる「次世代コンポーネントウェア技術に関する研究開発」の発注を受けて実施した研究開発でも検証の協力を得ることができた．この研究では，データ登録，修正，削除，検索の 4 種類についてユースケースフレームワークを，

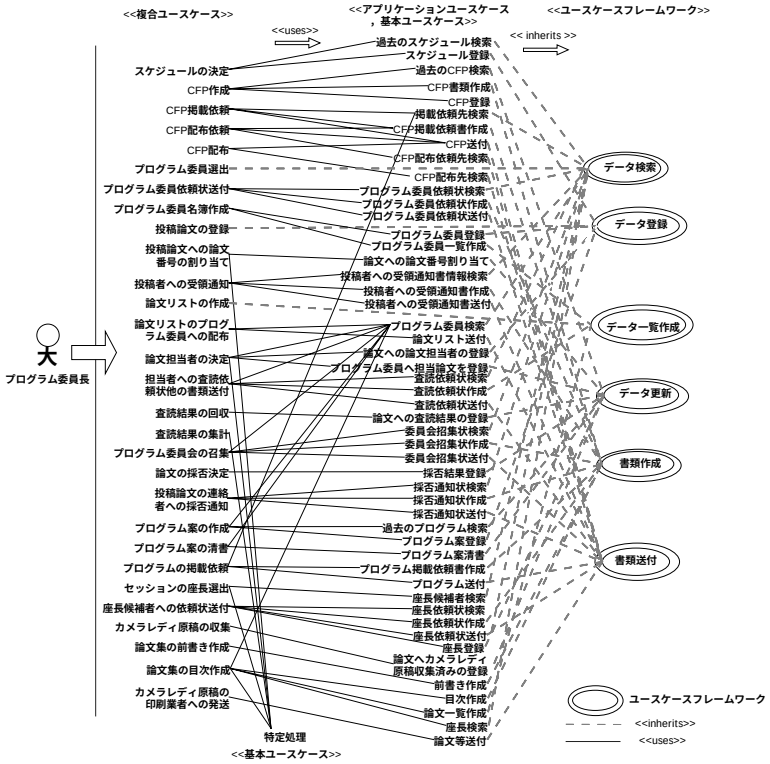

図 1　プログラム委員長業務のユースケース図とユースケースフレームワーク

表 2　ユースケースフレームワークの適用効果

システム名	ユースケース数	適用対象	対象外
システム 1	68	47	21
システム 2	60	55	5

ビジネス系アプリケーション開発への適用を検討した．その結果，表 2のような効果を期待できることが明らかとなった[14]．

対象外に分類されたユースケースは複合ユースケースである．先に示したユースケースフレームワーク適用プロセスに準拠した開発を行えば，適用対象はさらに増えるはずである．以上の調査から，ユースケースフレームワークを定義することによって，ビジネス系のソフトウェア開発においては，ユースケース記述の生産性向上を十分期待できると言える．

5. 考　　察

Cockburn は，TROUSERS モデルによってユースケースの系列の項目番号の組み合わせによってシナリオを表した[1]．このモデルは，ユースケースからテストケースを生成する指針を提供している．Jacobson が主張しているのは，ユースケースを開発のすべての工程の主導的道具として導入することである[7]．我々は，要求を抽出するための道具としてユースケースを活用するためには，記述自身の生産性向上が必要であると考える．現状のように構造化されていないユースケースを記述するには，多くの時間が必要であり，時間をかけても品質の高いユースケースを書くことができない．本稿では，この問題解決の糸口をユースケースフレームワークの提供という形で示した．

ユースケースフレームワークというユースケースの記述の枠組みによって，特定のユースケースでは，基本系列でどのような代替系列が発生し得るかといった熟練者のノウハウを提供することができる．実際のアプリケーション開発ですべての代替系列を考慮すべきでない場合もあるが，考慮すべきか否かを議論する必要はある．このような議論を経ることによって，ユーザと開発者は，要求と要求の背景について相互理解を深めることができると考える．

Collins は，ユースケースを要求，利用者インタフェース，システムのサービスの実現という三つの

カテゴリに分類し，それぞれについてユースケースの構造を定義した[4]．要求ユースケースは，他の二種類のユースケースと利用関係を持つ．さらに，複数のユースケースから利用されるインタフェースの統一やデータの共有によるユースケースの一貫性を保つためには，ユースケースのメタモデルが必要である．メタモデルを提供することによって，ユースケース記述の支援と管理が可能となる．我々はメタモデルの構築を現在進めているが，その中間成果が文献 11) にある．

今後，本稿で示したユースケースフレームワークを活用したユースケースの記述が実用されるようになると，例えば登録ユースケースフレームワークについて，様々なバリエーションが提案されるようになるだろう．ユースケースフレームワークは，記述の枠組みを決めるが，要求に制約を与える目的で使うべきではない．開発者は，ユースケースフレームワークから，ユースケースの記述方法を学び，その枠組みを実際の要求記述に適用できるか否かをユーザとともに検討し，新しいユースケースフレームワークを構築することを期待する．

実時間システム向けのユースケースフレームワークは，中谷が基本ユースケースとして検討した例がある[10]．実時間システム向けのユースケースフレームワークは，今後，事例を収集しながら検討を続ける．

6. ま と め

本稿では，これまでに提案されたユースケースの構造を整理し，ユースケースフレームワークを提案した．ユースケースフレームワークを利用することによって，ユースケースの生産性向上とレビューの効率向上も実現できる．今後は，ユースケースメタモデルによる記述支援とユースケースの管理を行い，ユースケースの品質向上を検討する．

謝　　辞

本研究の成果の検証には，(株) 情報技術コンソーシアムの安達 隆氏，黒田 健司氏，山浦 直人氏のご協力を頂きました．関係者の方々に感謝いたします．

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/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) Jacobson, I., Christerson, M., Jonsson, P. and Overgaard, G. : *Object-Oriented Software Engineering: A Use Case Driven Approach*, Addison-Wesley, 1992.
6) Jacobson, I., Ericsson, M. and Jacobson, A. : *The Object Advantage*, Addison-Wesley, 1994.
7) Jacobson, I. , Booch, G. and Rumbaugh, J. : *The Unified Software Development Process,* Addison-Wesley, 1999.
8) 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.
9) 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.）
10) 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.
11) 中谷多哉子, 安達隆, 山浦直人, 黒田健司, 玉井哲雄 : “ユースケース定義のためのメタモデルの構築”, 情報処理学会研究会報告, SE126-5, pp.33–40 (2000).
12) Schneider, G. and Winters, J. P. : *Applying Use Cases,* Addison-Wesley, 1998.
13) UML Semantics version 1.1, Rational Software Corporation, September 1997. http://www.rational.com/uml/index.html
14) 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.
15) 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.