データベースシステム 121－15
情 報 学 基 礎 58－15
（2000．5．26）

要求分析のためのメタモデルの提案

中谷 多哉子† 玉井 哲雄††

オブジェクト指向によるソフトウェア開発では，要求仕様記述にユースケースを用いることが多い．ユースケースの記述の生産性を上げ，品種を向上させるためには，ユースケースの明確な構造を定義する必要がある．記述の生産性は枠組みを定義することによって向上させることができる．また，ユースケースの品質を向上させるためには，ユースケースの矛盾を発見したり，新しい要求を抽出するための手法を開発しなければならない．本稿では，ユースケースのメタモデルに，動的な要求のモデルを追加し，ユースケースの記述の漏れや矛盾を発見する要求のメタモデルを提案する．

A Metamodel for Requirements Analysis

TAKAKO NAKATANI † and TETSUO TAMAI ††

In object-oriented system developments, use cases are applied for requirements analysis. In our previous works, we have proposed use case frameworks for typical use cases. To enhance requirements quality, we should elicit conflicts from use cases and extract undefined requirements from the problem domain. In this paper, we present the requirements metamodel which includes use cases and dynamic requirements for extracting conflicts in use cases and undefined requirements.

1. はじめに

オブジェクト指向では，システムの要求定義にユースケースが用いられる．ユースケースは，Jacobsonの開発方法論で提案された要求仕様記述形式であり，システムと，システム外部のアクターとの間の相互作用が時系列で記述される．ユースケースは，アクターとシステムの境界に存在するインタフェースオブジェクトと，システムの状態を保持し，協調動作するエンティティオブジェクト，それらの制御を行うコントロールオブジェクトを抽出する情報源となり，オブジェクト抽出，設計，テストケースなど，すべての後工程から参照される[6]．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であると言われている[4]．しかし，ユーザがユースケースを理解できるということで，簡単にユースケースの品質を向上させたり，記述の生産性を向上できるわけではない．実際の開発現場では，ユースケースの品質を高めるために多くの時間がレビューに費やされている．

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の4つの項目から構成される[4),5)]．我々は，より詳細なユースケースの構造化とユースケース間の関連を解析し，要求から，未定義の要求を抽出したり，要求の矛盾を発見する手段を研究している．

ユースケースの矛盾を発見するには，ユースケース間の関連を把握する必要がある．ユースケース間の関連を定義する視点として，二つの視点を考えることができる．第一の視点は静的なユースケース間の関係を観察する視点である．UML(Unified Modeling Language)[15)]のユースケース図には，汎化一特殊化の関係，呼び出し関係，拡張関係など，静的な関係を定義できる．第二の視点は，ユースケースが起動される順番やユースケース間の依存関係に基づく制約を観察する視点である．UMLのアクティビティ図や状態遷移図といった表記法を用いると，動的な仕様を記述することができる．

我々は，これまで，第一の視点に基づき，定型的なユースケースのフレームワークを検討し[9)]，ユースケースの構造を解析してきた[10)]．本稿では，ユースケースを時間軸上に配置する動的な視点からユースケースの構造を検討し，未定義要求の発見や要求の矛盾を発見するための要求メタモデルを提案する．

本稿の構成は，次のようになっている．第2節で，関連研究を紹介し，第3節では，オブジェクト指向の

† 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

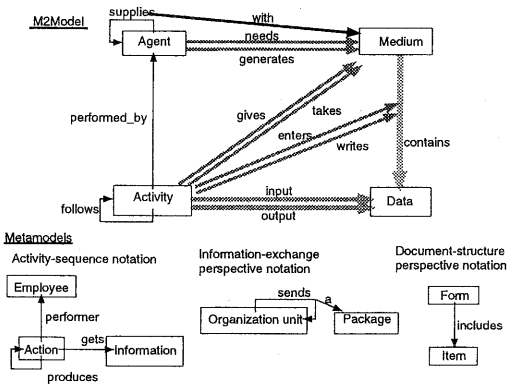

図 1　Nissen らの M2Model[11)]
Fig. 1　M2Model introduced by Nissen et al.[11)]

UML で提案されているユースケースとアクティビティ図との間の関係を解析する．第 4 節では，動的な要求を取り込んだ要求メタモデルを提示し，ユースケースに記述されるべき要求の発見と矛盾を発見する方法について議論する．

2. 関連研究

要求分析は，様々な視点から行われ，実体関連図，データフロー図，フローチャート，状態遷移図など，視点ごとに異なる仕様化技術が用いられる．Finkelstein と Nuseibeh らは，様々な視点で記述された仕様書間の矛盾を発見するために，仕様書間の制約を記述する枠組みを提案した[3),12)]．

仕様書の矛盾は避けなければならない問題であるが，要求抽出と要求分析の工程では，矛盾を発生させた原因を追及することによって，未定義の要求を抽出できることもある．Nissen らは，複数の視点で記述される仕様書について，そのメタモデルを解析し，さらに要求のメタメタモデル（M2Model) を構築した[11)]．M2Model によって，視点ごとに定義された仕様書の間の関係が明らかとなり，仕様書間の競合や矛盾を自動的に発見できると考えられている．図 1に，Nissen らの M2Model を示す．

M2Model は，要求を記述する様々な仕様化技術を，次の 4 つの視点から整理したモデルである．

- エージェント間のメディアの交換
- 情報構造
- 逐次活動
- プロセス間の関係

それぞれの視点によって，次の 4 つのオブジェクトとオブジェクト間の関係が定義されている．

Agent　システムの動作や振る舞いに対して，働きかけをする事物

Medium　情報交換が行われる媒体．紙や電子媒体など．

Activity　情報を取り扱う活動

Data　操作される情報

例えば，エージェント間のメディアの交換は，Agent 間の “提供する” という関連と，提供する Medium との間の関連によって説明することができる．Activity は，Agent との間に “実施される” という関連を持ち，その Activity がデータを生成／更新／参照する活動である場合には，Data との間に “入力” と “出力” の関連を持つ．また，それが情報媒体へ書き込む活動である場合には，媒体を構成するデータとの関連を “生成する” という関連を持つ．

我々は，ユースケースと，M2Model とを対応付けながら，ユースケースと動的な視点で定義される要求との関係を解析し，個々の仕様の依存関係を明らかにし，矛盾や未定義の要求を発見する指針を検討した．

3. ユースケース

Jacobson の記述方法に従って記述したユースケースの例を以下に示す．

- 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
- 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**プログラム委員**の登録希望を通知する
- 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．
  登録インタフェースは未定
  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**永続化**する．
  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．

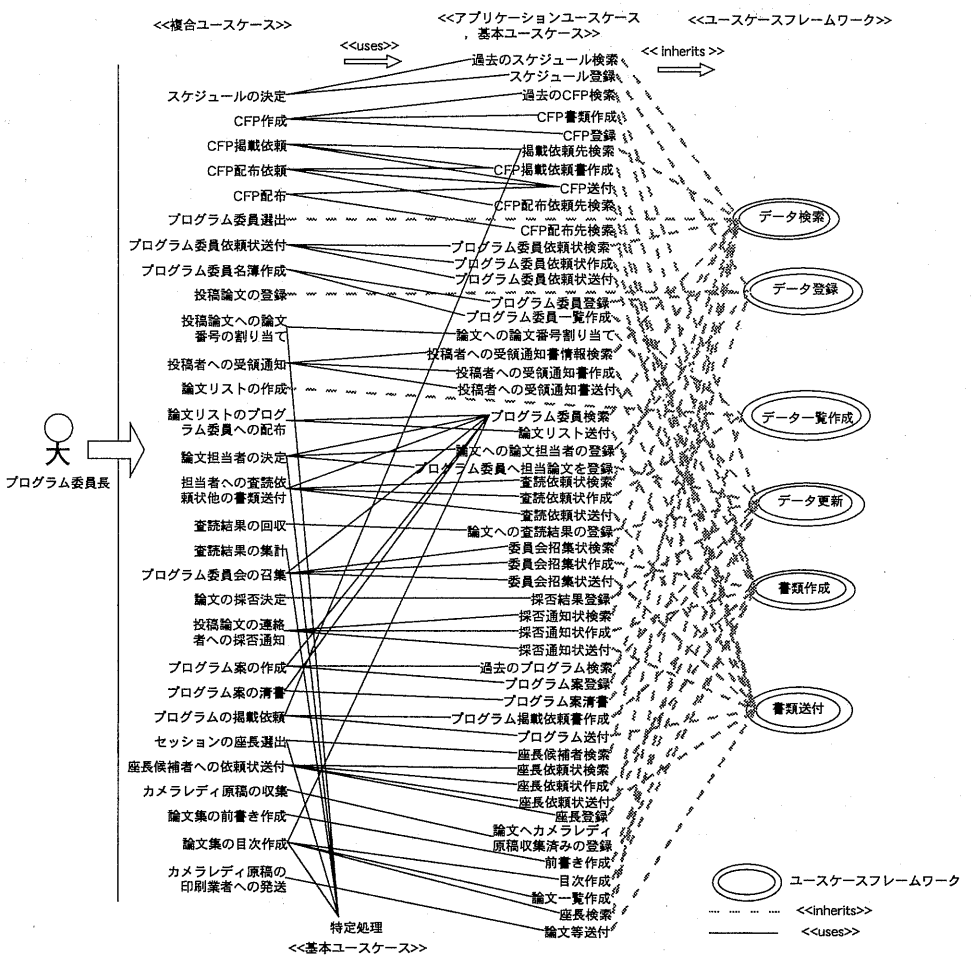

図2　プログラム委員長のためのユースケース図の例

Fig. 2　An example of a use case diagram for a program chairman

データの同一性は，名前と連絡先によって判断する．
永続先=XX データベース

3．システムはアクターに**プログラム委員**が新たに登録されたことを通知する．

プログラム委員が新たに登録されたことを通知するインタフェースは未定

4．アクターは，**プログラム委員**が登録されたことを確認する．

プログラム委員が登録されたことを確認するインタフェースは未定

- 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2.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する．
    - (b)　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．ただし，**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**は未定
    - (c)　基本系列2を繰り返す．
  - (3)　基本系列2.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．ただし，**登録済みとなっていることを通知するインタフェース**は未定

(b) 基本系列2を繰り返す．

このユースケースは，「国際会議のプログラム委員長の業務[7]」から抽出したものである．プログラム委員長の業務を解析するときに，上記のような形式のユースケースだけでは，要求の全体像を把握できない．全要求を俯瞰し，ユースケース間の静的な関係を概観するには，図2のようなユースケース図を用いる．このユースケース図には，

- プログラム委員長の業務を支援するためにどのようなユースケースが定義され，
- それらのユースケースは，どのような基本的なユースケースの組み合わせで定義されるか，
- それらの基本ユースケースはどのような定型的なユースケースを具体化したユースケースか

を定義することができる．しかし，ユースケース図は，システム開発に必要な要求が網羅されているか否かを検証する手段には適さない．

要求の網羅性を検証するには，アクティビティ図や状態遷移図が適している．例えば，アクターが人である場合，その人の活動や仕事の流れをアクティビティ図に表記することで，一連のシステム化の範囲を網羅した業務全体の流れを把握できる．

図3にプログラム委員長のアクティビティ図を示す．この図は，国際会議開催のスケジュール決定から，採録論文の印刷依頼に至るプログラム委員長の主な業務の流れを示している．ここでは，後の議論の参考となるように，各アクティビティで生成されたり，参照されたりするデータを注釈として記述した．アクティビティとデータとの間の点線の矢印はデータの流れを表す．

Schneiderらは，アクティビティ図から要求を仕様化する手順を，次のように定義している[14]．

(1) アクティビティ図から，システムで支援するアクティビティを抽出する．
(2) 各アクティビティに対応するユースケースを記述する．
(3) ユースケースをレビューする．
(4) ユースケースからオブジェクトを抽出し分析モデル化の作業に入る．

我々の問題は，ユースケースを記述しているとき，ユースケースとアクティビティ図との間の論理的な関係が切り離されてしまい，要求の抜けや矛盾を発見しにくくなることにある．プログラム委員長の業務の流れを解析し，ユースケース図を記述し，ユースケース

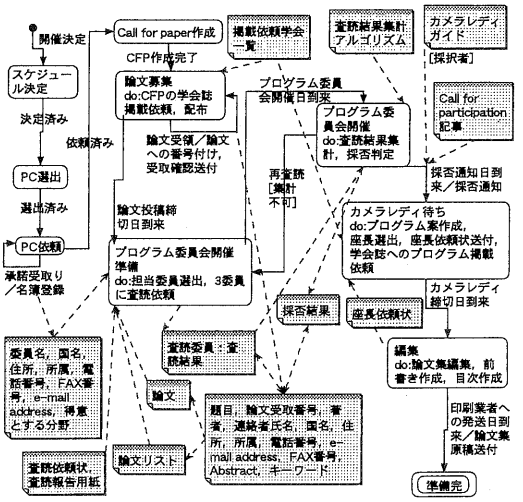

図3 プログラム委員長のアクティビティ図の例
Fig. 3 An example of an activity diagram for a program chairman

を定義するといった分析工程は，要求の詳細化を行う工程であり，様々なシステムのユーザの要求を見渡すという行為が不足しがちとなる．例えば，プログラム委員長の査読結果の取りまとめ作業では，プログラム委員の査読報告の遅れへの対処も考慮しなければならない．しかし，このような要求は，システムでどのように査読結果の取りまとめ作業を支援するかを考える視野からは外れやすく，問題領域の専門家でなければ見落としがちな要求である．

実際のシステム開発では，アクティビティ図とユースケース図やユースケースを見比べながら，要求の抜けがないか，必要な情報を見落としていないかを手作業で検証し，インタビューとレビューを繰り返す．場合によっては，これまでシステムのユーザ対象とは考えていなかった人にインタビューしなければならないこともある．これらの作業は繁雑であり，再現が困難であると言われてきたが，ユースケースと動的な要求との関係を明確に把握できれば，要求記述に内在している問題を抽出することができるはずである．

4. 動的視点とユースケースの関係

中谷らは，先に示したような定型的なユースケースについて，記述の枠組みを考案し[9]，そのメタモデルを開発した[10]．ユースケースのメタモデルをM2Modelに対応させると，

- ユースケースのアクター，システムといった関与者はAgentに，
- ユースケースの基本系列，代替系列はActivityに，
- ユースケースから抽出されるエンティティはData

に，

- ユースケースで定義される利用者インタフェースは Medium に，

それぞれ対応させることができる．したがって，ユースケースは，要求仕様で記述される多面的な情報を網羅できる可能性を持っていることがわかる．

ユースケースには，ユースケースの順序とユースケースをまたがるデータの流れといった情報が欠けている．前者は状態遷移図やアクティビティ図から解析でき，後者はデータフロー図によって表現される情報となる．

例えば，エンティティオブジェクトのライフサイクルから，ビジネス系ユースケースでは，エンティティオブジェクトの処理に関して，エンティティオブジェクト登録が最初に行われ，そのエンティティオブジェクトが削除されるまでエンティティオブジェクトの更新と参照の処理が繰り返されるという制約がある．

図 3には，注釈で表したデータが，アクティビティの流れに沿って定義され，参照されることを説明している．しかし，プログラム委員長がこのアクティビティに沿って作業を行うという保証はなく，また，手順どおりに作業を進めることができない状況が発生し得るかもしれない．そのときにシステムがどのような反応をすべきかも，要求として抽出しなければならない．図 3から，未定義の要求を発見できるのも，アクティビティの流れとデータの状態遷移とを同時に見ることができるためである．

このように，データ，すなわち，ユースケースから抽出されるエンティティオブジェクトの状態遷移を解析することによって，以下のような要求を抽出するヒントを得ることができる．

- エンティティオブジェクトを二度登録することを許すのか，許さないのか．
- エンティティオブジェクトの削除後にエンティティオブジェクトを参照するユースケースが定義されるのは誤りであるが，このようなことは起こり得ないのか．起こり得るとしたら，そのとき，どのような手続きを実行すべきか．
- 登録したエンティティオブジェクトが参照されない場合，そのエンティティオブジェクトを登録する要求は適切か．もし必要であれば，抽出されていないアクターが他にいるのか．エンティティオブジェクトを参照するユースケースが未定義なのか．

また，アクティビティ図からは，アクティビティ図に定義されている順序に従ってユーザが処理を実行しなかった場合，システムがどのように反応するかといった例外事象を検討するヒントを得ることもできる．アクティビティの順番には，業務のポリシーが関わる要求であるが，変更される可能性のあるポリシーと変わらないポリシーもある．

- アクティビティが実行される順序に制約はあるか．
- 順序づけが守られていることを確認する処理は，ユースケースに確実に定義されているか．
- 一人のアクターが起動するユースケースの間に同期や非同期といった制約を考慮しなくてもよいか．
- 複数のアクターが起動するユースケースの間について，同期や非同期といった制約の有無を検証したか

大西らは，データのフロー要求を抽出したり確認するためのシミュレーションシステムを開発している[13]．フロー要求を明らかにすることによって，複数のアクターがシステムを介して相互作用する様子を発見することができる．この情報は，複数のアクターが起動するユースケースの同期や非同期の制約に関する要求を抽出するヒントとなる．

以上の動的な視点から得られる要求は，ユースケース図のような静的なモデルだけを参照していたのでは発見できない要求である．要求のメタモデルは，静的なユースケースの構造だけでなく，これらの動的な制約を取り込んだモデルとなる．

5. 要求メタモデル

中谷らが提案したユースケースのメタモデルに，アクティビティ図で定義されるユースケース間の順序づけとデータの状態遷移を考慮したメタモデルを図 4に示す．

要求メタモデルの構成要素を以下に説明する．

5.1 静的な構造の反映

データ辞書項目　契約（後述）から共有される可能性のある情報をキー付きで保持するオブジェクトのクラス．

エンティティオブジェクト　操作対象となり，システムの状態を保持するオブジェクトのクラス．エンティティオブジェクトは分析モデルで抽出される可能性のあるオブジェクトでもある．Jacobson のエンティティオブジェクトと同義．データ辞書でキーによってシステムで管理される．

利用者インタフェース　ウィンドウやコンポーネントなどの利用者インタフェースオブジェクトのクラス．操作方法がユースケースで記述されることもあるため，複合契約（後述）と関連を持つ．データ辞書でキーによって管理される．

契約　事前条件，事後条件，アクター，記述内容を属性として持ち，例外状態をキーとして他の契約との関連を持つ抽象クラス．

条件　条件記述のクラス．実際の記述は，パラメー

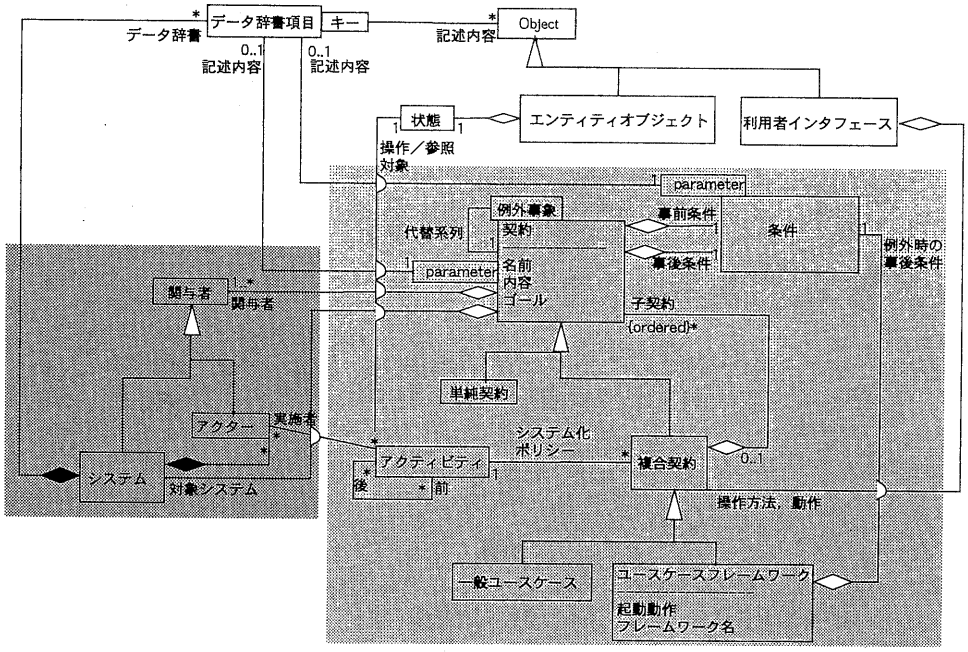

図 4　動的視点を導入した要求メタモデル
Fig. 4　Requirements metamodel with dynamic perspective

タにデータ辞書で管理されるオブジェクトへの関連づけを与えて完成する．

単純契約　契約のサブクラス．記述内容の系列を持たない契約の具象クラス．

複合契約　契約のサブクラス．記述内容の系列が二つ以上の子契約からなる契約の具象クラス．子契約には，単純契約や複合契約が該当し得る．

一般ユースケース　複合契約のサブクラス．記述内容を 1 から記述した一般的なユースケース．明確な構造を持たないため，そこに定義された情報がデータ辞書に管理されるとは限らない．

ユースケースフレームワーク　複合契約のサブクラス．実際の記述は，パラメータにデータ辞書で管理されるオブジェクトへの関連づけを与えて完成する．定型的なユースケースの記述の枠組み．フレームワークの名前，起動動作，例外時の終了条件[1]を属性として持つ．基本系列は，スーパークラスの複合契約に示されている子契約の時系列の連なりとして表現される．

関与者　契約の関与者の抽象クラス．

システム　開発対象のアプリケーションシステムのクラス．関与者の具象クラス．

アクター　契約の起動者となる関与者の具象クラス．

データ辞書の項目は，アプリケーションごとに定義され，契約や条件から参照される．

ユースケースフレームワークの起動動作には，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，システム自身がユースケースを起動するといったユースケースのトリガーを記述する[2]．通常のユースケースは，基本系列の最初の項目にトリガーを定義する．特に，定型的なユースケースでは，トリガーが共通であるものが多い．そこで，可読性を改善するために，このトリガーを「起動動作」の項目に定義し，他の基本系列の項目とは区別した．

Jacobson のユースケースで，基本系列と代替系列が別の項目に定義されているのは，レビュー時にレビューすべきシナリオを発見しやすくするためである．例えば，基本系列に分岐や繰り返しがなければ，基本系列が最初にレビューするシナリオとなる．次に代替系列に記述された項目に沿って，例外事象を選択し，その事象が発生したときのシナリオを基本系列から分岐させて生成し，レビューする[8]．ユースケースのメタモデルでは，このような基本系列の項目と，例外事象，代替系列の関連を明示した．

契約と例外事象を関連づけたことで，ユースケースには，基本系列の特定の項目に対応して発生する例外

事象に対して記述される代替系列と，ユースケース全体にかかる例外事象に対して記述される代替系列という，二種類の例外事象を定義できるようになった．図4では，前者の代替系列には，基本系列を構成する個々の契約に定義される例外事象をキーとする契約が対応し，後者には，複合契約が契約から継承している例外事象をキーとする契約が対応する．

5.2 動的な構造の反映

M2Model に対応させると，ユースケースは4つの視点を表現できる構造を持っていることがわかると前に述べたが，動的な視点が十分表現されているとは言えない．M2Model で表現されているアクティビティの順序付き関連は，ミクロな視点では，ユースケースを構成する基本系列の契約の順序付き関連に対応するが，マクロな視点では，ユースケース間の順序付き関連に対応させることもできる．これは，アクティビティ図に記述されるものである．

図4に以下の項目は，要求の動的な構造を反映したものである．

アクティビティ 複合契約の順序づけという制約を与えるオブジェクトのクラス．アクティビティには多対多の前後の関係があり，システム化のポリシーによって，個々のアクティビティにユースケースが定義される．

状態遷移 エンティティオブジェクトの状態遷移を保持するオブジェクト．状態を遷移させる操作はアクティビティである．状態遷移の順序によって，アクティビティは順序付けられる．

アクティビティの関連 アクティビティは，その操作対象のエンティティオブジェクトを介して，エンティティオブジェクトの状態を遷移させる他のアクターのアクティビティと間接的な関連を持つ．間接的に関連を持つアクティビティ間には，時間軸上の制約が存在する．

6. 考　　察

これまでの開発現場では，ユースケースの要求の漏れや矛盾をレビュー時に手作業で発見し，解決していた．本稿で示した要求のメタモデルは，手作業で行っていた検証事項をユースケース記述のためのメタモデルに取り込んだものである．Nuseibeh や Finkelstein らの多重視点に基づく要求仕様の矛盾を発見する研究は，仕様記述に矛盾発見のための構造を埋め込み，そうして定義された仕様書の中の矛盾を抽出しようとするものであった．本稿で紹介した要求のメタモデルに基づいて，ユースケースを記述することによって，これまでは切り離されていたアクティビティ図や状態遷移図との関連を把握できるようになり，仕様書の記述の漏れや矛盾を発見しやすくなると考える．

ただし，実際の要求抽出の場面で仕様化される要求は，問題領域の専門家の知識や経験，日常業務の一部に過ぎない．一部というのは，例えばアクティビティ図を記述したとき，そこに記述されるのは実際の業務の流れをモデル化した結果であるという意味である．そのため，システムを業務の現場に導入したとき，予想しないアクティビティによって予想しない使い方をされたり，システムが考慮しなかった事象の発生によって，現場で対応しなければならないといった問題が数多く発生する．

このような問題を完全に排除することはできないが，ユースケースの記述に，より上位の要求を把握するリンクを持たせておくことによって，未定義の要求を把握しようという意思は高まるのではないだろうか．ユースケース間の矛盾は，要求分析の誤りだけが原因ではなく，未定義の要求を把握できていない場合もある．要求メタモデルに基づいて要求を抽出することによって，アクティビティ図の作成からシステム化範囲の決定，ユースケースの記述といったトップダウンの作業展開を，平面的に拡大させることができる．

本稿では，定型的な要求の漏れや矛盾の発生を予期する手法を再考し，ユースケースに記述されない動的な要求と，ユースケースの記述との関連を求め，要求メタモデルにまとめた．

要求抽出はシステム開発の入口であり，現実世界と形式的世界との橋渡しをする役割を持つ重要な活動である．今後，ユースケースに関する研究をさらに進め，要求定義の現場へ適用することによって，より精緻な要求メタモデルに進化させていきたい．

7. ま と め

本稿では，要求の M2Model をもとに，ユースケースのメタモデルを再考し，ユースケースの記述に制約を与える動的な要求との関連を明らかにした．ユースケースのメタモデルには，ユースケースが本来持っている静的な視点と機能的な視点から明らかとなる要求が構造化されている．本研究では，ユースケースのメタモデルに動的な要求の構造を取り込み，要求メタモデルとして提示した．今後，要求分析の現場に要求メタモデルを適用し，要求メタモデルの精緻化を検討していきたい．

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 Tem-

plate," http://members.aol.com/acockburn/papers/uctempla.htm.
3) Finkelstein, A. , Gobbay, D., Hunter, A. , Kramer, J. and Nuseibeh, B. : "Inconsistency Handling in Multi-perspective Specifications," *IEEE Transactions on Software Engineering,* Vol. 20, No. 8 (1997), pp. 569–578.
4) Jacobson, I., Christerson, M., Jonsson, P. and Overgaard, G. : *Object-Oriented Software Engineering: A Use Case Driven Approach*, Addison-Wesley, 1992.
5) Jacobson, I., Ericsson, M. and Jacobson, A. : *The Object Advantage*, Addison-Wesley, 1994.
6) Jacobson, I. , Booch, G. and Rumbaugh, J. : *The Unified Software Development Process,* Addison-Wesley, 1999.
7) 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.
8) 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.)
9) 中谷多哉子, 玉井哲雄 : "ユースケースフレームワークの検討", ソフトウェアシンポジウム, 2000 年 6 月.
10) 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).
11) Nissen, H. W., Jeusfeld, M. A., Jarke, M., Zemanek, G. V. and Huber, H.: Managing Multiple Requirements Perspectives with Metamodels, *IEEE Software*, March 1996, pp. 37–48.
12) Nuseibeh, B., Kramer, J. and Finkelstein, A. : "A Framework for Expressing the Relationships Between Multiple Views in Requirements Specification," *IEEE Transactions on Software Engineering,* Vol. 20, No. 10 (1997), pp.760–773.
13) Ohnishi, A. and Tokuda, N. : "Visual Software Requirements Definition Environment," Proc. of the IEEE 21st International Computer Software and Applications Conference (COMPSAC), 1997, pp.624–629.
14) Schneider, G. and Winters, J. P. : *Applying Use Cases,* Addison-Wesley, 1998.
15) UML Semantics version 1.1, Rational Software Corporation, September 1997. http://www.rational.com/uml/index.html