Vol.2010-SE-168 No.15
Vol.2010-EMB-17 No.15
2010/6/1

プロトタイプ自動生成可能なモデルドリブン要求分析手法による要求仕様のトレーサビリティの向上

小形真平[†]　松浦佐江子[†]　酒井理江[††]
佐藤宏之[††]　小林透[††]

近年のクラウドコンピューティングに見られるように，複雑・大規模化されるシステムを複数サービスの複合体として効率よく安定して構築するためには，個々のサービスに対する要求をそのエンドユーザーの立場から分析し，開発することにより，各サービスとソースコードのトレーサビリティを確保することが重要である．われわれは，これまでユースケース駆動により UML(Unified Modeling Language)を用いてデータ・振舞い・シナリオ等の異なる観点から定義した要求分析モデルから，最終プロダクトの機能におけるシステムの内部処理および UI における外観を除いたシステムの模型であるプロトタイプを自動生成する手法を提案してきた．提案手法では，そのエンドユーザーの顧客または開発者が，サービス提供に必須なフローとデータをプロトタイプを通して確認することにより，モデルの整合を図っている．システムにおけるトレーサビリティを向上させるためには，要求仕様においては，使用される語句の一貫性とそのデータ構造および制約の明文化，サービスの操作手順の妥当性を検証することが必要である．本稿では，実際に企業が正式な手順に従って開発したアプリケーションの非形式要求仕様を提案手法の要求分析モデルで再定義し，結果として発見された従来仕様の曖昧性や矛盾点等の問題点を複数サービスの複合体としてシステムをモデルドリブン開発する観点から議論する．

Enhancement of Requirements Specification Traceability by Model Driven Requirements Analysis employing Automatic Prototype Generation

Shinpei Ogata[†]　Saeko Matsuura[†]　Rie Sakai[††]
Hiroyuki Sato[††] and Toru Kobayashi[††]

The cloud computing is a technique to efficiently develop a large and complex information system by reusing existing services. Then, two of keys to develop high-quality service reusing existing service are to sufficiently analyze each service from the viewpoint of end users and to ensure the traceability between each specification of existing service and each set of source code. To achieve these keys, we have proposed use-case-driven requirements analysis method using Unified Modeling Language (UML). The main feature of proposed method is to automatically generate prototype from UML requirements analysis model (RA model) defined in proposed method so that the customers and developers can easily validate the model and check the consistency between the models. The RA model represents several aspects such as the data and behavior of service and the scenario which represents usage of the service in a specified condition. Also, the prototype represents the internal process and the User Interface (UI) without its layout shown as a product image following the RA model. For enhancing the traceability, following three points are should be considered. Firstly, it should identify the meaning of data name uniquely. Secondly, it should clarify the structure and constraints of data. Thirdly, it should validate the steps of operating the service. In this paper, to discuss solutions of issues of informal specification which negatively affect the traceability such as ambiguity and inconsistency, from the viewpoint of model-driven development, we redefine the requirements specification of actual product using proposed method.

1. はじめに

開発コストを抑えつつ，高機能なサービスを，エンドユーザーの使用性向上という観点から提供するために，近年のクラウドコンピューティングに見られるように，システムを複数サービスの複合体として効率よく安定して構築できることが必要となっている．このような開発においては，個々のサービスに対する要求をエンドユーザーの立場から十分に分析することに加えて，トータルなサービスに対する要求を，新規開発サービスと既存サービスの適切な連携方法や，エンドユーザーの入力数に関する操作負荷の軽減の観点から，十分に分析できる必要がある．これらを実現するためには，要求分析段階からつぎの 2 点を達成することが重要である．

(1) 新規開発サービスが手続きおよびデータフローの観点から既存サービスとどのように連携するかを正確に仕様化すること
(2) 要件レベルから新規サービスへの既存サービスの再利用性を正確に検討するため，サービスの要求仕様とソースコードのトレーサビリティを確保すること

われわれは，これまで業務系 Web アプリケーションを対象に，ユースケース[1]駆動により UML(Unified Modeling Language)[2]を用いてデータ・振舞い・シナリオ等の異なる観点から定義した要求分析モデルから最終プロダクトの機能におけるシステムの

[†] 芝浦工業大学大学院　工学研究科
Graduate School of Engineering, Shibaura Institute of Technology
[††] 日本電信電話株式会社　NTT 情報流通プラットフォーム研究所
NTT Information Sharing Platform Laboratories, NTT Corporation

Vol.2010-SE-168 No.15
Vol.2010-EMB-17 No.15
2010/6/1

内部処理および UI(User Interface)における外観を除いたシステムの模型であるプロトタイプ[3]を自動生成する手法を提案してきた[4]．要求分析モデルでは，サービス間の連携やサービスとそのエンドユーザーのインタラクションをアクティビティ図，クラス図，オブジェクト図により定義する．さらに，提案手法では，そのエンドユーザーである顧客または開発者が，サービス提供に必須なフローとデータをプロトタイプにより確認することによりモデルの整合を図っている．

一般に，既存モジュールの再利用に着目した手法では，コンポーネントベース開発に見られるように既存モジュールを組み合わせる開発手法[5]が提案されており，ソースコードの再利用性の議論ではモジュール化の適切性が 1 つの焦点である[6]．その再利用性向上の 1 つの技術が，ソースコード内の手続きやデータの抽象化である．しかし，サービス連携によりアプリケーションを構築する場合においては，各サービスを実現するソースコード群はそのアプリケーションの範囲で要件や設計，実装技術に基づき既に最適化・抽象化されたものであるため，別のアプリケーションにそのようなソースコードをそのまま再利用するには限界がある．

そこで，本研究では，既存サービスにおける最終的なソースコードに対応づいた本質的かつ具体的で理解しやすい要件レベルの手続きやデータ（後述する UML アクティビティ図のアクションやオブジェクトノードおよびUML クラス図)を単位として，機能または非機能といった要件ごとにモデルを分割統治するような系統的な定義プロセスを含めた要求仕様を再利用することで，既存サービスにおける手続きやデータの柔軟な再利用方法の確立を目指す．その前提として，最終的なソースコードに表れる手続きを再利用するためにトレーサビリティを確保した要求仕様が不可欠となるが，本研究では，モデルドリブン開発の観点において，要求分析から設計・実装までのモデルを段階的に詳細化することでその実現を目指す．

本稿の焦点となるトレーサビリティの確保では，使用される語句の一貫性とそのデータ構造および制約の明文化，サービスの操作手順の妥当性検証が必要である．本稿では実際に企業が正式な手順に従って開発した Android アプリケーションの非形式要求仕様を提案手法により再定義し，発見された従来仕様の曖昧性や矛盾点等の問題点を複数サービスの複合体としてシステムをモデルドリブン開発する観点から議論する．

2. 要求分析とトレーサビリティの向上

2.1 要求と要求分析プロセス

要求の種類は，手続き条件や手続き順序等の機能要求と性能や安全性等の非機能要求に大別される．携帯アプリのように不特定多数のエンドユーザーへのサービス提供のためにセキュリティ等の非機能要件を重視することは当然であるが，機能要件がサービスの核であり,その妥当性や正確性を十分に検証することの重要性は周知である．

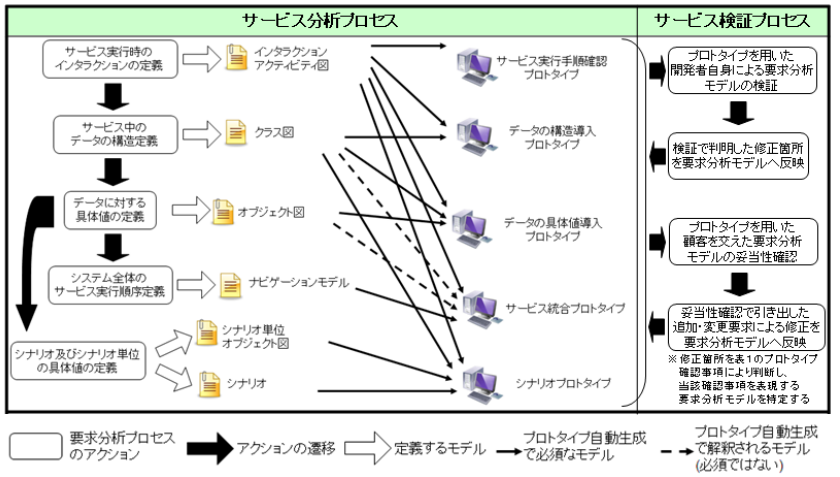

図 1　要求分析プロセス

そのため,提案手法は機能要求の分析を中心とした要求分析プロセスを提案している．

提案手法における要求分析プロセスでは，機能要件の妥当性や正確性を十分に確認することを目的として，定義と妥当性確認・検証の工程をそれぞれ，図 1 に示すようにサービス分析プロセスとサービス検証プロセスとして想定する．サービス分析プロセスでは，要求を振る舞いや構造の側面ごとに分析し，要求分析モデルを定義または洗練する．サービス検証プロセスでは，顧客または開発者が要求分析モデルの妥当性や正確性を確認する．後者のプロセスでは，詳細な確認方法までは規定しないが，側面ごとに UML モデル図を使い分けて定義した要求分析モデルから，提案手法が提供するツールを利用して自動生成される各側面を統合したビューであるプロトタイプを用いて，要求分析モデルの整合確認を図る．なお，要求分析モデルの詳細は 3 章で事例を示しながら説明する

2.2 インタラクションに着目した要求分析

サービスのエンドユーザーは，システムの可視部である UI(User Interface)を通して，操作手順や入力項目へ入力方法の十分性と容易性，入力/確認すべき項目の十分性，入力に対する出力の計算の正当性といった側面から機能の妥当性を確認する．一方で，開発者がエンドユーザーの操作負荷を考慮する場合，直接入力するデータ数やその機会を低減することが重要である．しかし，サービスの根幹をなすデータの低減は品質の低下を招く要因となり得るため，開発者は如何に既存サービスを活用して必要なデータを獲得できるかが重要となる．既存サービスを利用するためには，関数と同様に

Vol.2010-SE-168 No.15
Vol.2010-EMB-17 No.15
2010/6/1

サービス実行時の入力データとその構造，サービス実行後の出力データとその構造を明確に理解する必要がある．さらに，新規サービス本来の目的を達成するために，どのタイミングで既存サービスと連携すべきかを十分に検討しなければならない．

従って，エンドユーザーの立場やサービス連携の観点からそれらのインタラクションを明確に定義することが要件定義，ひいては将来のサービス開発におけるサービスの仕様の再利用性向上に対して重要な要素であると考える．そのため，提案手法ではインタラクションの分析に着目したユースケース駆動の要求分析手法を採用している．

2.3 要求仕様とシナリオ

要求仕様は，エンドユーザーの立場からサービスに要求される手続きやデータの要件を明確化したものであり，エンドユーザーが実際にサービスを実行する観点で具体的に理解できるものでなければならない．そこで，提案手法では以下の 2 点を同時に成立させることを目的として，UI 上の入出力データやシステム内部で処理されるデータに対する具体値を含めた特定のサービス実行パスを定義するシナリオを定義し，シナリオに基づくシナリオプロトタイプを提供する．

(1) エンドユーザーの実操作における負荷を確認するため，要求分析者がエンドユーザーの立場で特定の状況下のサービスを実操作する観点から理解する．
(2) システム内部の手続きによって，システム内部に保持されるデータの変化を実値を用いた具体レベルで理解する．

また，要求分析者がサービスの実現可能性を十分に検討する上で，サービス連携において，データがどのように変化するかの想定を後工程の開発者と正確に共有できなければならない．提案手法では，エンドユーザーの観点の UI だけではなく，システム内部の手続きに係わるデータを可視化するプロトタイプ生成も提供する．

2.4 非形式要求仕様の問題点

われわれはこれまで研究室内の図書管理システムとスケジュール情報共有システムの仮想開発事例を設け，要求仕様の品質の比較を目的に提案手法と従来手法の比較実験を行った[7]．本実験では，従来手法としてユースケースモデル（UML ユースケース図，非形式記述であるユースケース記述）とクラス図によりインタラクションの仕様を定義し，かつ対応する UI イメージを別途手動で作成するものを想定した．結果として，従来手法では UI イメージとユースケースモデルの間でデータ名の不整合が 4 割から 6 割程度発見された．また，クラス図にはエンティティ候補のクラスが定義されたが，ユースケース記述ではエンティティ候補以外のデータが多数存在した．例えば，図書のキーワード検索における“キーワード”といったシステム化して初めて登場する機能実行のパラメタのようなデータがクラスとして定義されなかったことが原因である．開発の初期段階では，洗練された構造よりも手続きで扱われるデータを全て列挙することが要求漏れを回避する観点から重要であり，このデータの定義漏れは致命的な問題である．さらに，フローの観点から UI イメージの作成は作業時間が多大にかかる関係上，その作成者はユースケース記述の一部のフローを UI イメージ化していたが，それらの対応関係を明記した仕様がなかったため，ユースケースモデルから UI イメージを正確にトレースできなかった．提案手法では，プロトタイプ上に表れる情報を意識的に要求分析モデルに定義することができたため，これらの問題点を解消することができていた．

2.5 トレーサビリティ向上の要点

本研究では，トレーサビリティとは要求仕様の項目全てを統合した情報が実際にサービスの満たすべき要件と過不足なく合致するかどうかの仕様の性質と定義する．そして，要件レベルのモデルではトレーサビリティ確保に以下の点が重要である．

(1) システムの側面ごとに分離・整理される仕様において，語句の一貫性や側面間の関連を正確に定義すること
(2) 実装技術に係わらない手続きや，データの構造や制約を明文化すること

(1)については，インタラクションの正確な定義実現のためには，振る舞いー構造間の関連や，呼び出しのタイミングを含めた振る舞いー振る舞い間の関連の正確な定義が重要である．この点が，非形式要求仕様で実現することが難しいことは前節で述べた通りである．また，語句の一貫性を明確にするためには，例えば「ユーザー」のインスタンスは「ログインユーザー」や「コンテンツ作成者」とコンテキストによって呼び名が変わることからも，言葉の整理だけではなく，その語句に対応づくデータの構造と制約や，特定の具体的な振る舞いのルールを対応付けなければならない．

(2)については，最終的には言語や抽象化方法に依らず，（設計段階では既存のフレームワークの機能として代替される可能性もあるが）式や条件，データとしてソースコード中に必ず登場する手続きやデータの構造や制約であり，要求仕様として明確に定義すべきである．また，これらの定義が不十分な場合は，一意に解釈できない要件定義となる可能性があり，意味的なトレーサビリティの保証は難しい．

また，単純な定義ミス等から，これらの全てを初定義の段階で明確に正確に定義することは難しいため，サービス検証プロセスにより十分に洗練する必要がある．

次章で説明するように，本稿では実際に企業が正式な手順に従って開発したアプリケーションの非形式機能設計書に対して，提案手法を用いたモデルドリブン開発の観点から，上記の両点における問題点を議論する．

3. 適用事例と要求仕様の再定義

提案手法が要求仕様のトレーサビリティの向上に貢献可能かを評価するため，実際に企業が一般ユーザーを対象として実証実験サービスに利用した Android アプリケーションの機能設計書を要求分析モデルで再定義する．なお，本アプリケーションは，当該企業の正式な開発手順を踏まえて作成されたものである．そして，発見された従

Vol.2010-SE-168 No.15
Vol.2010-EMB-17 No.15
2010/6/1

来仕様の曖昧性や矛盾点等の問題点に対して，モデルドリブン開発の観点から要求仕様定義時の要点を議論する．提案手法では，UML モデリングツールとして，ChangeVision 社の Astah*(旧 JUDE)[8]を利用する．

3.1 適用事例の概要

本稿の適用事例は，伊藤ら[9]が提案した時刻や場所等の複数のコンテキストからユーザーの嗜好を推定する行動支援サービスのためのユーザー理解モデルに基づき，エンドユーザーまたはそのグループごとに“お勧めの飲食店”を提供するロジックを実現したユーザー理解基盤エンジン（以降，単に“基盤エンジン”）である．この非形式機能設計書は，アプリケーション開発後に複数の開発者により定義された基本設計レベルのものであるため，通常の要求分析時の仕様以上に設計情報を含み，最終的なソースコードを正確に表現する可能性が高い．最終的には Android 端末ユーザーを対象としたサービスであるが，本設計書では UI の仕様は存在しない．また，基盤エンジンは複数の既存サービスと連携しており，既存サービスの外部インタフェース仕様が別途存在し，当該仕様も要求分析モデルの再定義対象に含める．

基盤エンジンの機能設計書の構成は，システム概要，用語集，システムの構成，ユースケースとその詳細，ユースケース中で参照される機能とその入出力データ及び方式詳細，データベースのテーブル定義である．ユースケースと機能の差異は，前者が，提供すべきサービスを中心に手続きを定義するものであり，後者は，ユースケース間で共通性のある振る舞いをまとめたものや，サービスの核である嗜好の履歴化や嗜好に基づく店舗算出のアルゴリズムをユースケースに比べて詳細に定義したものである．

本仕様では，手続きはユースケースの基本フローレベルの粒度であり，例外フローはほぼ未定義である．また，テーブルや入出力データおよび用語集の語句を利用することで，手続き中に扱うデータとの関連を非形式に定義している．

なお，本事例当初の関係者の知識背景としては，当該仕様に対する要求分析モデルの定義者（1 名）はアプリケーションの構成を概念レベルで理解しているが，詳細な技術や手続き，データは理解していなかった．また，本事例について定義者と同程度の知識を持つ経験豊富な開発者（1 名）がレビュアーとして参加した．

再定義に係わる実働期間は 2 カ月程度であり，レビュアーとのレビューを平均 2 時間で 5 回程度行い，実開発者と平均 1 時間のモデルの妥当性確認を含めた報告会を 2 回行った．また，対面形式とは別に，定義者は実開発者に対して機能設計書に対する疑問点をまとめた質問表による質問を 2 回行った．

3.2 適用事例における要求分析モデルの定義とプロトタイプ自動生成

本事例では，提案手法の特徴である段階的プロトタイプ自動生成を活かしながら，以下の手順に沿って定義と確認を繰り返しながら再定義を行った．

1. 機能設計書のユースケースと機能のそれぞれにおける基盤エンジンと既存サービスのインタラクションを UML アクティビティ図を用いてモデル化した．

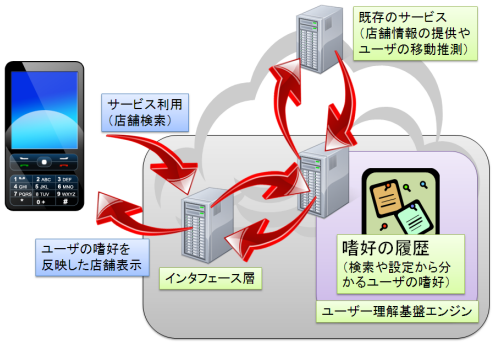

図 2 ユーザー理解基盤エンジンをベースとしたサービスの構成

2. ユースケースから機能への参照関係から，ユースケースと機能のインタラクションを接合し，そこから自動生成されるプロトタイプを用いてレビューを行った．
3. 機能の入出力データ，テーブル定義，既存サービスの外部インタフェース仕様からインタラクション中で処理されるデータの構造を UML クラス図を用いてモデル化した．この段階も，自動生成されるプロトタイプを用いてレビューを行った．
4. 手順 3 におけるプロトタイプを用いて実開発者との妥当性確認のための報告会を行った．さらに曖昧語とその統一，構造仕様，振る舞い仕様の疑問点を質問表としてまとめ，同開発者に質問した．
5. 質問回答を 2 回繰り返して要求分析モデルを洗練した後に，各ユースケースの基本となる正常フローの 1 パスに対し，データの具体値を含めたシナリオを 1 つずつモデル化した．この段階で，シナリオに基づいて自動生成されるシナリオプロトタイプを用いてレビューを行った．

3.2.1 ユースケース・機能からのインタラクションアクティビティ図の定義

手順 1 では，Android 端末を示す「クライアント」や，「ユーザー理解基盤エンジン」をシステムのパーティションとし，その間のインタラクションにおける手続きとデータのフローをアクションやオブジェクトノード，コントロールノードの系列により定義する．図 3 はユースケースまたは機能のフローを表すアクティビティ図の例である．

再定義においては，機能設計書に合致するように本図のアクションをユースケース詳細の 1 ステップや機能の 1 方式ステップに対応付けて記述した．ただし，複文で複数の手続きが 1 ステップに内包されている場合は，適宜分割して 1 アクションの複雑度を緩和した．また，オブジェクトノードとしてのデータ抽出は，手続き中に非形式にデータ名が登場するため，直前のアクションから登録・読込・更新・送信・計算等の対象となるデータ名をデータベース仕様のテーブル名も踏まえて適宜抽出した．色

Vol.2010-SE-168 No.15
Vol.2010-EMB-17 No.15
2010/6/1

付きのノートは，再定義中に挙げられた疑問点(赤)，修正点(緑)，メモ(黄)である．

この段階では，分岐条件があるにも係わらず直列ステップとして記述されるような明らかなフローの誤りや，機能中で参照されるテーブル名がデータベース仕様のものと合致しないような語句の不整合といった表面的に理解できる不整合を発見した．

手順 2 では，ユースケース詳細からユースケースとその参照機能を接合した．本仕様では，参照する機能を明記しているが，その“タイミング“は不明瞭であるため，定義者が適宜接合したが，最終的には実開発者によるその妥当性の確認を行った．手順 1 と手順 2 は図 1 の“サービス実行時のインタラクションの定義”に相当する．

3.2.2 データフロー・テーブル定義・外部インタフェース仕様からのクラス図定義

手順 3 では各オブジェクトノードに対して，構造としてクラスを定義し割り当てる．図 4 は最終的な要求分析モデルにおけるクラス図であり，データベース仕様（青色のクラス），2 種類の既存サービスの外部インタフェース仕様（赤色と黄色のクラス），前述以外の機能の入出力データやユースケース詳細および機能方式詳細に登場するデータ構造（緑色のクラス）に基づき名前と属性を決定する．緑色のクラスについては，既存設計書中に構造として明確に定義されていないものもあったが，これについては定義者が手続きを定義した文章を適宜解釈し，必要最低限のデータ構造を定義した．

このとき，インタラクションアクティビティ図における手続き中のデータ名と，クラス図中に表れるデータ名の整合性は，モデルの表示を切り替えながら確認するため，作業負荷が高い．そのため，図 5 に示すように手続きやデータのフローとデータ構造を統合したビューとして要求分析モデルから自動生成されたプロトタイプを用いて整合性を確認する．プロトタイプでは，インタラクションアクティビティ図のフローに沿ってアクションとオブジェクトノードをそれぞれメッセージ形式とテーブル形式により表現する．なお，プロトタイプは機能を持たない HTML 形式で生成される．手順 3 は図 1 の“サービス中のデータの構造定義”に相当する．

3.2.3 具体データを含むシナリオの定義と妥当性確認

手順 4 の質問回答や報告会により既存の機能設計書の疑問点を整理し，語句の統一や，手続きのフローとデータ名の修正の後に，手順 5 で各ユースケースに対する特定の状況下のシナリオを想定した．既存の機能設計書を十分に把握しきれていない定義者は，手続きに沿って具体レベルのデータの変遷過程を想定することでユースケースの理解を深めるとともに，機能設計書が最終的にエンドユーザーの得られるべき情報やその情報を提供するための手続きを十分に非曖昧に定義できているかを確認した．

提案手法では，インタラクションアクティビティ図から導出されるフローのパスをシナリオのパスとして生成する．このパスに沿って逐次出現するオブジェクトノードを単位とし，図 6 に示すように UML オブジェクト図を用いて具体値を定義することで，シナリオを完成する．そして，図 7 に示すようにシナリオに沿って具体値を挿入したシナリオプロトタイプを生成し，サービス実行時の手続きやデータ変化の想定に

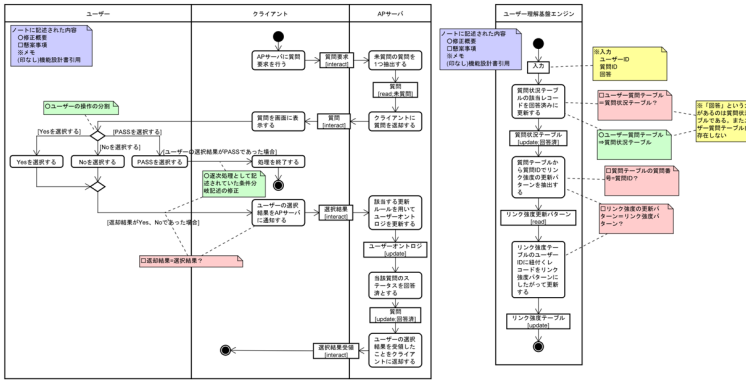

図 3　インタラクションアクティビティ図（手順 1）

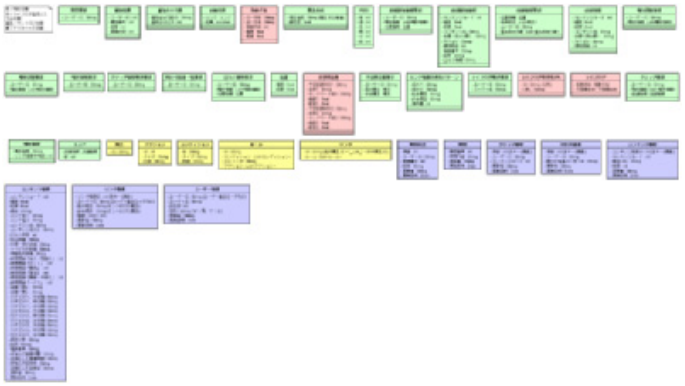

図 4　最終的な要求分析モデルにおけるクラス図

不備がないかを確認する．この段階では，具体値レベルのデータ変化の過程を明確にすることで，取り得る値の範囲とその形式の問題や，これに付随する手続きの欠如等の問題を発見した．また，定義者はレビュアーとのその問題に係わる誤解も解消した．手順 4 と手順 5 は図 1 の“シナリオ及びシナリオ単位の具体値の定義”に相当する．そして，これまで図 1 に対応づかなかった“データの具体値の定義”は，シナリオ定義の前段として行われており，“システム全体のサービス実行順序定義”は，既存設計書にユースケース間の繋がりを示す仕様が存在しなかったため行えなかった．

Vol.2010-SE-168 No.15
Vol.2010-EMB-17 No.15
2010/6/1

4. 適用事例における問題点と考察

本事例では，トレーサビリティを低下させる要因となる語句の不整合や曖昧記述を発見できた．本章では，問題点の量を結果として述べ，各問題点の種類と発見時期を説明し，最後にモデルドリブン開発の観点から利点と問題解決法を考察する．

4.1 結果

表 1　問題点の数と分類

分類	曖昧語	構造仕様	振る舞い仕様
数量（個）	28	22	15

発見した問題点の数を質問表の質問から集計した結果が表 1 である．「曖昧語」は，一見同一な意味に見える用語が複数存在する場合の分類である．例えば，“予測目的地”や“予測位置”といった一見同一データに見えるが表現が異なる場合である．「構造仕様」は，ユースケース詳細や機能方式詳細のみに登場するデータ名の表す構造が一意に解釈可能ではない場合や，データ構造を示す仕様に登場するデータについて，手続き中に登場しないために利用状況が不明確な場合の分類である．例えば，“店舗詳細情報”は，その名前から一意に解釈可能な構造であるとは言い難い．「振る舞い仕様」は，手続き中に登場するデータの取得時期が不明確な場合や，手続きに応じてデータに割り当てる既定値の種類が不明確な場合，ユースケースと機能の接合点において各仕様間の入出力データの想定に不整合がある場合の分類である．

本事例では，最終的にソースコードと要求分析モデルを比較した．ソースコードのエンティティクラスと一見対応づくクラスが不足なくモデルに存在したが，その設計過程が不明瞭なため，例えばソースコードに存在した制御クラスやデータベース接続クラスは，モデルでは未定義であった．従って，要求分析モデルがソースコードまでトレーサビリティを確保したと評価するまでには至っておらず，そのトレーサビリティを確保した最終的なモデル像を示す必要があり，この点は今後の課題である．

ただし，制御クラスはおよそユースケース単位にフレームワークに適合する形で定義され，データベース接続用クラスはおよそエンティティクラス単位で定義されていた．そして，制御クラスのメソッドは，再定義した手続きに沿って定義されていたため，フレームワークを根拠としたモデルの抽象化や，接合対象のフレームワークを表現したモデルを検討し，現状の要求分析モデルに導入することで，提案手法における要求分析プロセスを拡張した設計プロセスの実現の素地はあると考える．

4.2 問題点の種類と発見時期

4.2.1 分離定義された振る舞い間の関係の不整合

振る舞い仕様に係わる問題点として，ユースケースが参照する機能自体が間違っている例が 1 件あった．また，接合点において，ユースケース側には存在しないが，参照機能側に存在するデータが 2 件あった．この不整合の発見は，ユースケースと機能の接合時に，対応すべきデータの妥当性を確認し，機能で利用するデータのユースケ

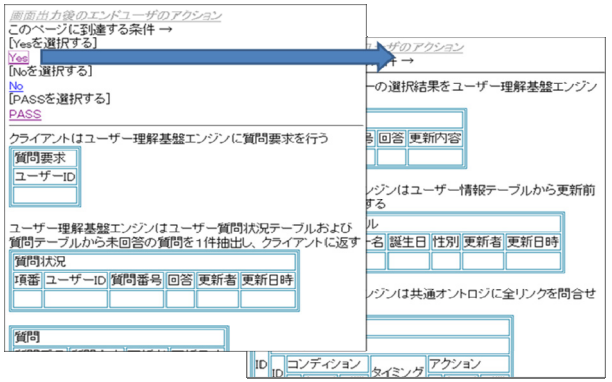

図 5　システムの内部処理を可視化したプロトタイプ（手順 3）

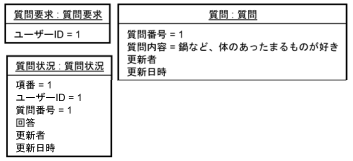

図 6　シナリオ中の具体値を表すオブジェクト図

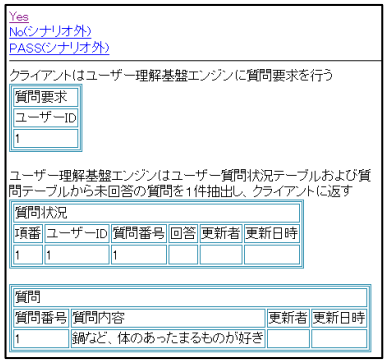

図 7　シナリオプロトタイプ

Vol.2010-SE-168 No.15
Vol.2010-EMB-17 No.15
2010/6/1

ース上での取得時期が不明瞭な点を質問したことによる．これは，ユースケースと機能の定義を分離したため，両者の接合時の妥当性を確認できていない例である．

4.2.2 抽象化における機能の曖昧性

ある 2 つのユースケースでは，特定の情報を既存サービスから取得する機能を参照していた．しかし，当該ユースケースが当該機能に要求するデータの構造が互いに異なっていたため，機能の定義に曖昧性があることが発見できた．この問題はデータの構造を明確化せずに手続きの共通性を判断し，抽象化したことにより生じたと考えられる．この発見は，各ユースケースや機能のデータフロー中のオブジェクトノードにクラスとして構造を割り当てたことによる．従って，データフローにおけるデータ構造を初期段階から明確化することが機能の正確な定義につながると考える．

提案手法では，各サービスの要求を段階的に定義する過程において，顧客が十分に確認できる具体的なフローの明確化が目的のため，手続きの共通性は要求分析段階では特に注意して議論はしない．しかし，設計段階では，これを整理する必要がある．

4.2.3 語の関連定義の欠如による誤解

種々のユースケースとその参照機能間の接合点において，対応すべき入出力データ名に不一致なものが多数確認できた．この理由として，つぎの 2 点が考えられる．第一に，機能は一連の手続きを抽象化したものであるため，複数ユースケースからの参照機能に登場するデータ名は，これらのユースケースに登場する個々のデータ名を意味的に内包するように抽象化したと思われる．例えば，“自動車”と“電車”を抽象化すると“乗り物”に呼称が変わる．このような抽象-具体の関係による命名の変化自体は問題ではないが，その対応関係が不明瞭な場合は誤解を生じさせる問題となる．

第二に，ユースケースと機能を分離したことで，両者に意味的な不整合が生じた点である．例えば，ある 2 つのユースケースが同一機能から取得したいデータがそれぞれ“店舗詳細情報”と“口コミ情報”であったが，機能の仕様ではこれらを抽象化したデータを“店舗情報”と名付けていた．当該エンジンの知識が浅い定義者は当初これらのデータが同一に見えたが，それぞれのデータ名に紐づくデータ構造を質問した結果，構造が互いに異なっていたような不整合が発見できた．このような語の解釈違いによる誤解の解消は仕様定義に不可欠であり，提案手法ではデータの構造定義とアクションとデータの関連を明示することで，こうした不整合を解消し，要求分析者の暗黙的に補完したユースケースと機能の関係を明確化に役立つ．

4.2.4 手続きルールの曖昧性

通常，一部の手続きはデータに副作用を与える可能性がある．しかし，実際に何の副作用がどのデータに与えられるかは明記されていなかった．例えば，「ユーザー理解基盤エンジンはユーザー情報テーブルから更新前ユーザー情報を取得する」に想定される「ユーザーの更新」という手続きのルールは明確化されていない．「ユーザーの更新」では，例えば現在日付による年齢の更新が考えられるが，このような記述の欠如から誤解が生じる可能性が高い．この発見は，定義者がシナリオ定義段階で手続きに沿ってデータの変化を具体値レベルで想定したことや，レビュアーがシナリオプロトタイプにより具体値レベルのデータフローから手続きのルールを想定したことによる．

提案手法では，サービス連携のためのサービス実行のタイミングや入出力データを明確にすることを重点に置き，手続きルールの定義方法を議論していない．この問題は，UML 標準の形式言語 OCL(Object Constraint Language)[10]を導入し，解決を図る．

4.2.5 値の制約の曖昧性

ある範囲を表すデータが「1～99」という制約があったが，この理由が不明確である．その制約がどの要件に応じて決定しているかを理解できなければ，その制約の要件へのトレーサビリティを確保できない．例えば「弱い」「普通」「強い」と言う 3 段階のように「段階的に強度を評価したい」という理由を明記することが必要であると考える．この発見は，エンドユーザーの立場から得られる情報について，シナリオプロトタイプを定義者とレビュアーが確認することによる．

4.3 提案手法の利点と課題

4.3.1 利点 1：データフローとデータ構造の定義の分離と形式的な連携

手続き中のデータ名に対して，構造といった側面から意味が明確化されないことを原因とした問題点が多数存在した. 提案手法では, UML のセマンティクスを活かして，データフローとデータ構造を分離し，かつオブジェクトノードとクラスをキーとして接点を小さくした形式的な連携を実現している．そして，提案手法では，モデルの切り替え表示することなくフローとデータ間の整合性を確認する方法として，プロトタイプの自動生成を提供しており，実際に不整合を発見できている．さらにシナリオ定義後のシナリオプロトタイプにより手続きと具体値レベルのデータの変化を対比し，その手続きが達成すべき内容の理解性を向上した．実際にレビュアーは「エンドユーザーの立場として必要な情報を想定してシステム内部の手続きやそれに係わるデータを具体値レベルで確認することで問題を発見しやすかった」と評価している．

4.3.2 利点 2：シナリオ定義における具体データの想定と曖昧箇所の明確化

データを構造，型，具体値の観点からその実体の定義を詳細化することで，実際は不適切に抽象化された手続きが存在した．提案手法では，特定の手続きのパスに沿って，具体値レベルの構造化されたデータを想定することで，その問題を引き起こす曖昧な記述を事前に回避することができる．

4.3.3 課題 1：条件や制約の形式定義とシナリオの妥当性の検証

提案手法では，手続きのルールの形式的な定義を実現していないため，フローのシナリオの妥当性を形式的に検証できず，シナリオの正確性や無矛盾性を保証する枠組みがない．この問題は，フローに登場する分岐条件や，型の制約，アクションの手続きルールを OCL により定義し, 具体値を含めたシナリオにおいてサービス間の境界を流れるデータのバリデーションを含む分岐条件の判定や，手続きルールに基づく手続

Vol.2010-SE-168 No.15
Vol.2010-EMB-17 No.15
2010/6/1

き前後のデータ変化の妥当性を形式的に検証する枠組みを検討することで解決を図る．

4.3.4 課題 2：語句の対応関係の明確化

今回，曖昧語の原因として，抽象－具体の関係にある語句間の対応関係の明記の欠如が挙げられた．提案手法では，その対応関係の明記方法は規定していないため，各データに対して OCL を用いてデータの派生関係の定義し，この問題の解決を図る．

5. 関連研究

サービス連携を重視する観点から既存のユースケース駆動開発手法[11]や UML を用いたモデルドリブン要求分析手法[12,13]と比較する．三部ら[11]は，ユーザーとシステムの UI 上のやりとりを記述した複数のシナリオパタンからユースケース記述を構成し，シナリオパタンに応ずる複数の画面パタンから画面を選ぶことでプロトタイプを自動生成する．しかし，本手法のユースケース記述では，システム間のやりとりは定義しないため，サービスの連携の観点からは恩恵を得られない．

Elkoutbi ら[12]は，クラス図の定義は前提として，コラボレーション図によりオブジェクトのメッセージングのシナリオを定義する．シナリオのセットからステートマシン図への変換を経て，UI オブジェクトとしてみなした特定のオブジェクトの状態遷移を基に UI プロトタイプを自動生成する．しかし，本手法では具体値をシナリオに反映できないため，値の変化といった具体値レベルのモデル理解が不可能である．また，コラボレーション図は，データフローを除いたメッセージのやりとりを定義するものであるため，複数のサービス連携時に流れるデータを明確に定義することはできない．

また，佐藤ら[13]はサービスをコンポーネントとして，その連携により最終的なサービスを提供するというアーキテクチャである SOA(Service Oriented Architecture)を対象として，UML モデルを用いたモデル駆動型のセキュリティ開発プロセスを提案しており，サービス分析プロセスとセキュリティ分析プロセスの分割統治を実現している．提案手法は，UML を用いた機能要求中心の分析手法であり，同様にクラス図を利用する本手法と定義制約および定義対象上競合しないため，手法連携の実現可能性がある．このように MDA(Model Driven Architecture)[14]に見られるように機能要件や非機能要件の各観点や，要求分析や設計の各段階から複数側面に分離したモデルを最終的に形式的に統合することで，同様な機能に対する要求がサービスごとに差異がある場合でも，定義プロセスを含めた既存仕様を柔軟に再利用できる可能性があると考える．

6. 結論

プロトタイプ自動生成可能なモデルドリブン要求分析手法により，手続きやデータの観点ごとに分離して定義し，かつ接点を小さく形式的に連携することや，具体値レベルで手続きやデータを既存の機能設計書から再定義することで非形式な要求仕様に含まれた不整合を発見することができた．これらの不整合を UML や OCL を活用して排除することで，要求仕様の正確性の向上，ひいてはトレーサビリティの向上を図ることができると考える．今後の展望として，これまで要求分析モデルに未規定である条件値やデータの制約，手続きのルールを OCL を活用した正確な定義方法を検討する．

参考文献

1) Jacobson, I., Christerson, M., Jonsson, P., et al.: Object-oriented software engineering: A usecase driven approach, Addison-Wesley Publishing (1992).
2) Object Management Group: Unified Modeling Language, Object Management Group (online), available from <http://www.uml.org/> (accessed 2010-04-12).
3) ACM SIGSOFT: Special Issue on Rapid Prototyping, ACM SIGSOFT Software Engineering Notes, Vol.7, No.5 (1982).
4) 小形真平, 松浦佐江子：UML 要求分析モデルからの段階的な Web UI プロトタイプ自動生成，ソフトウェアエンジニアリング最前線 2008，pp.79-86 (2008).
5) Chusho, T. and Yagi, N.: Modeling by Form Transformation for End-user Initiative Development, Proc. of the 32nd Annual IEEE International Computer Software and Applications Conference, pp.331-334 (2008).
6) 五十嵐淳：Generic Java：多相的型付けによる安全かつ再利用性の高いオブジェクト指向プログラミング，情報処理学会誌　情報処理，Vol.45，No.6，pp.610-617 (2004).
7) 小形真平，松浦佐江子：Web UI プロトタイプ自動生成ツールを用いたユースケース駆動要求分析の評価実験 Web UI プロトタイプ自動生成ツールを用いたユースケース駆動要求分析の評価実験，第 8 回情報科学技術フォーラム　講演論文集　第 1 分冊，Vol.1，pp.73-80 (2009).
8) ChangeVision : Astah*, ChangeVision (online), available from <http://astah.change-vision.com/ja/> (accessed 2010-04-12).
9) 伊藤浩二，飯塚京士，村山隆彦，小林透：行動支援サービスのためのユーザ理解モデルの検討，ライフインテリジェンスとオフィス情報システム研究報告(LOIS)，LOIS2009-58，pp.121-128 (2009).
10) Object Management Group: Object Constraint Language, Object Management Group (online), available from < http://www.omg.org/spec/OCL/2.2/> (accessed 2010-04-12)
11) 三部良太，河合克己，竹内拓也，石川貞裕，福士有二：Web アプリケーションのユースケース駆動プロトタイプによる要求獲得方法，情報処理学会論文誌，Vol.49，No.4，pp.1669-1679 (2008).
12) Elkoutbi, M., Khriss, I. and Keller, R.K.: Automated Prototyping of User Interfaces Based on UML Scenarios, Journal of Automated Software Engineering, Vol.13, No.1, pp.5-40 (2006).
13) 佐藤史子，中村祐一，小野康一：モデル駆動型開発に基づく SOA のセキュリティ開発プロセス，情報処理学会論文誌，Vol.49，No.7，pp.2292-2303 (2008).
14) Object Management Group: Model Driven Architecture, Object Management Group (online), available from < http://www.omg.org/mda/> (accessed 2010-04-12).