2007－SE－157 (13)
2007／9／28

UMLで記述された要求分析モデルからのプロトタイプ自動生成

小形 真平† 松浦 佐江子‡

†芝浦工業大学大学院 工学研究科電気電子情報工学専攻
‡芝浦工業大学 システム工学部電子情報システム学科

システム開発では，顧客がプロトタイプを通して要求分析モデルの妥当性を確認する方法がある．顧客が満足しない箇所に応じ開発者はモデルを修正するが，修正内容の妥当性が問題となる．そこで，Web システム開発を対象に，利用者とシステムのインタラクションや入出力データを定義した数種の UML モデルを要求分析モデルとし，そこからプロトタイプを自動生成する方法を提案する．その結果，開発者は画面遷移や入出力データといった顧客の要求レベルに応じてモデルを修正し，繰り返しプロトタイプを生成することでモデルの妥当性を確認することができる．本学で使用されているグループワーク支援システムを適用事例として，その有用性を議論する．

An Automatic Generation of Prototypes from Requirements Analysis Models in UML

Shinpei Ogata† Saeko Matsuura‡

†Graduate School of Engineering, Shibaura institute of technology
Department of electronic engineering and computer science
‡Shibaura institute of technology Department of electronic information system

Prototyping is known as an effective software development method for customers to confirm the validity of requirement analysis model in the early stage of system development. However, the development process requires guaranteeing the consistency between the system model and requirements of customers which are caused by the confirmation. This paper proposed a prototyping method for Web applications that generates a prototype system automatically from the requirement analysis model in UML which defines the interaction between user and a system and the input/output data. We discuss the expressiveness of the generated prototype in comparison with the existing group work support tool.

1 はじめに

システム開発では，顧客がプロトタイプを通して要求分析モデルの妥当性を確認する方法がある[1]．このとき，顧客はシステムの提供する業務手順や入出力項目が要求を反映しているか確認し，満足しない箇所に対して開発者はモデルを修正する．しかし，プロトタイプがモデルの内容を過不足なく反映していなければ，顧客の指摘に対するモデルの修正箇所の特定は困難となり，モデル自体を一から作り直す可能性が大きくなる．そこで，要求分析モデルからのプロトタイプ自動生成により内容の整合性を保証する．これにより，プロトタイプの箇所に対応するモデルの内容を参照・修正することが可能となるため，修正の妥当性が向上する．本研究では，オブジェクト指向による開発を前提にしており，要求分析モデルでは，既存の業務手順や業務用語を基礎にユーザとシステムのインタラクションや入出力項目を UML2.0 (Unified Modeling Language)を用いて定義する．UML では振る舞いと構造を区別して定義するため，業務フローと入出力項目を数種のモデルにより区分けして定義する．本研究の特徴として，

- 要求分析モデルは，顧客の要求レベルに応じたモデルの作成・修正を実現するため，顧客がプロトタイプで確認する事項を基準にモデルを定義する．
- 要求分析モデルに対し，プロトタイプを自動生成するために与える制約を少なくし，洗練したモデルをオブジェクト指向に則った開発の後工程へ適用しやすくする．
- UML モデルの情報からプロトタイプを自動生成するツールを開発し，プロトタイプ開発の負担を減らす．また，形式的なプロトタイプ生成方法を提供することで，モデルの洗練過程において反復的にプロトタイプを生成する場合にブレのない要求分析モデルの視覚化が行える．

本稿はプロトタイプ自動生成ツールを用いて，本学で稼動中の Web システムであるグループワーク支援システム(以下，GWSS)に適用し，有用性を議論する．

2 プロトタイプと要求分析モデルの定義

2.1 プロトタイプ

本研究では HTML(Hyper Text Markup Language)により構成されるユーザインタフェースプロトタイプを用いて，顧客が表 1 の事項を確認するものとする．

表1　プロトタイプにおける顧客の確認事項

確認事項	確認事項の詳細内容
業務手順	・サービス実行手順としての画面遷移 ・サービスに対する入出力項目の過不足
入出力項目	・入力項目の入力形式 ・グループ化すべき入出力項目

要求分析モデルをユースケース図，アクティビティ図，クラス図，オブジェクト図とし，本提案用に拡張する．各モデルと確認事項の関連は表2に示す．特に入出力項目は，顧客が直感的に理解できるように単に項目名を表示するだけではなく，その表示のタイミング，入出力項目のグループ化，入出力項目の具体例を考慮してプロトタイプを作成する必要がある．モデルとの対応は，表示のタイミングをアクティビティ図に定義し，入出力項目のグループ化をクラス図へ定義し，入出力項目の具体例をオブジェクト図へ定義する．

表2　モデルの定義内容と確認事項の対応

モデル名	確認事項
アクティビティ図	・サービス実行手順としての画面遷移 ・サービスに対する入出力項目の過不足 ・入力項目の入力形式
クラス図	・グループ化すべき入出力項目
オブジェクト図	・入出力項目の具体例

2.2　アクティビティ図の定義

アクティビティ図はユースケース単位で定義し，入出力操作レベルのユーザとシステムのインタラクションを表現する．また，以下に示すようなプロトタイプ自動生成ツール用の独自の制約がある．

- 生成するプロトタイプとユーザとの境界を決定するために，ユーザとシステムの境界として，パーティション名をアクター名としたパーティションを定義する．(図1-(1))
- サービス実行手順としての画面遷移の遷移条件を明確にするためにデシジョンノード直後の制御フローに，必ずガードを定義する．(図1-(2))
- プロトタイプ上の入出力項目を表現するために，アクション名の形式を「～を…する」に統一する．「～」の名詞部分に入出力項目名を定義し，「…」に動詞部分に処理動作を定義する．(図1-(3))
- プロトタイプ上の入出力項目に入力形式を与えるため，HTMLにおける入力形式に基づき，入力表現の動詞を予め決定し，その動詞を必要に応じてアクション名に定義する．(図1-(4)と4.2節で後述)
- 顧客の直感的理解を助けるためにグループ化した入出力項目を表示するが，その表示のタイミングをユーザ側の処理のオブジェクトノードが定義された位置と対応付ける．(図1-(5))

ただし，アクティビティ図では，例えばデシジョンノードとマージノードを別々のノードで表現するか1つのノードで同時に表現するかのように，1つの意味を複数の記述で表現できるが，本ツールで解析する上で，各要素にただ1つの意味を持たせるために要素の制御フローの入出力数に制約を設け，表現方法を統一する．

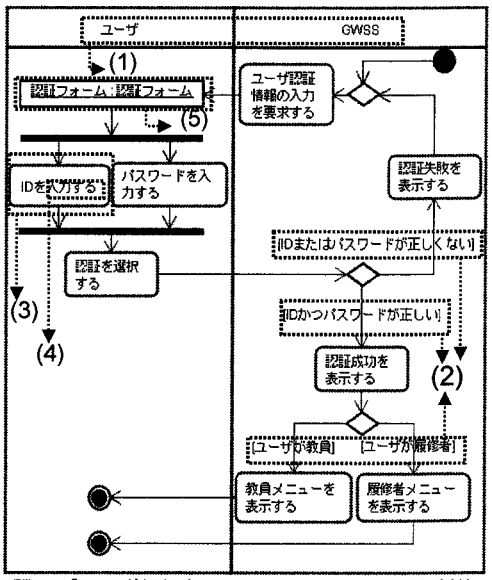

図1　「ユーザ認証する」のアクティビティ図と制約

2.3　クラス図の定義

クラス図では，プロトタイプ上で表現すべき入出力項目のグループ化を目的としてクラスを定義する．クラス候補やクラス内の属性候補は，アクティビティ図のオブジェクトノードの定義時に，そのオブジェクトに必要なデータを考えていくことで抽出する．記述内容として，関連や操作は定義せず，1つのクラスを入出力項目の1つのグループとみなし，クラスの1つの属性を1つの入出力項目として定義する．

2.4　オブジェクト図の定義

オブジェクト図では，2.3のクラスに対する具体例としてインスタンス仕様を定義する．なお，1つのスロットに複数の値を定義する場合は，カンマ区切りで定義する．

3　要求分析プロセスとプロトタイプの確認

本研究の要求分析プロセスを以下ように定義する．

1. 既存業務や顧客要求から必要サービスをユースケースとして導出し，ユースケース図に定義する．
2. 導出したユースケース単位でシステム化した業務手順をアクティビティ図に定義する．このとき，顧客が全ての入出力項目を確認できるように，入出力項目を詳細に定義することが重要となる．
3. 顧客が確認すべき入出力項目が不必要に分散しないようにクラスを用いてグループ化し，クラス図を定義する．
4. 3.で定義したクラスに対して，プロトタイプを用いて顧客に説明する際のシナリオを想定したインスタンス仕様をオブジェクト図に定義する．

5. システム化後のアクティビティ図，クラス図，オブジェクト図の情報からプロトタイプ自動生成ツールにより，プロトタイプを自動生成する．
6. 顧客がプロトタイプにより要求分析モデルの妥当性を確認し，満足しない箇所を指摘する．
7. 指摘の要求レベルを表 2 の確認事項にて判断し，対応するモデルを修正する．(以降，顧客の了承が得られるまで 5.-7.を反復的に繰り返す．)

本研究では，現状として 1.のユースケースの導出方法や 6.のプロトタイプの使い方は考慮していない．また，2.-4.のモデル定義では，順序的に行う必要は必ずしもない．

4 プロトタイプ自動生成ツールの設計と実装

4.1 アクティビティ図の解釈

プロトタイプ自動生成ツールでは，以下に示すような概念をアクティビティ図に設けており，この概念を用いて画面分割や遷移を表現している．

(1) ユーザ側へ処理が移行する直前のシステム側のアクションを「システム外部遷移トリガー」と呼ぶ．
(2) システム側へ処理が移行する直前のユーザ側のアクションを「画面遷移トリガー」と呼ぶ．
(3) 1 画面の区切りを画面遷移トリガー直後とする．
(4) システム側の処理で定義されるガードを，複数の遷移先を表現するための遷移条件とする．

図 2 に各番号とアクティビティ図との対応と分割された 1 画面の例を示す．

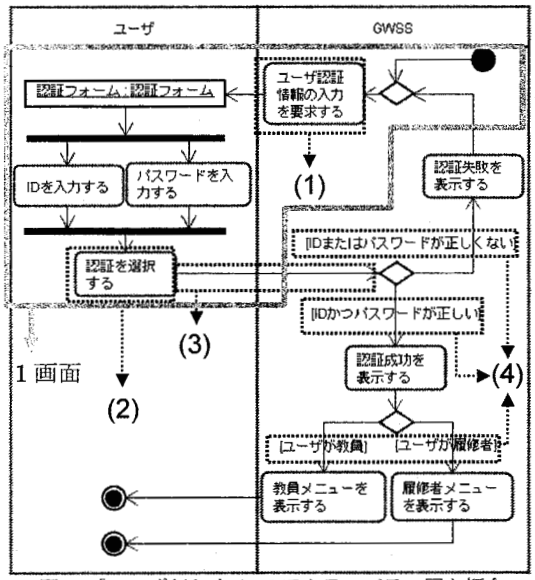

図 2 「ユーザ認証する」のアクティビティ図と概念

1 画面中の入出力項目は，ユーザ側の処理のオブジェクトノードとアクションによって決定し，図 2 の 1 画面の枠の例では，オブジェクトノードの認証フォームクラス，アクションの名詞部分より「ID」，「パスワード」，「認証」となる．

4.2 アクションの動詞と入力形式

2.2 節で述べた入力形式の表現用に用意した動詞とその動詞に対応する入力形式の種類を表 3 に示す．

表 3 動詞と入力形式の対応

動詞	入力形式の種類
入力する	テキスト入力形式
単数選択する	ラジオボタン選択形式
複数選択する	チェックボックス選択形式
選択する	リンク選択形式 (画面遷移トリガーに必要)
確認する	項目名表示(入力ではない)

例えば，図 2 の例では，図 3 のプロトタイプのように「ID」と「パスワード」はテキスト入力形式の入力項目であり，「認証」はリンク選択形式の入力項目となる．そして，「認証」のリンク先は，「ID またはパスワードが正しくない」，「ID かつパスワードが正しい&ユーザが教員」，「ID かつパスワードが正しい&ユーザが履修者」の 3 パターンの遷移条件に応じた遷移先が存在する．

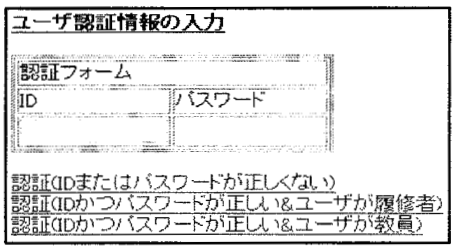

図 3 図 2 のプロトタイプ

4.3 クラス定義の入出力項目とテーブル表示

入出力項目のグループ化をクラス図で定義した場合，その構造が理解しやすいように，グループ化した入出力項目はプロトタイプ上ではテーブル形式で表示する．プロトタイプにおけるテーブル構成要素を図 4 に示し，その構成要素と要求分析モデルの要素の対応を表 4 に示す．

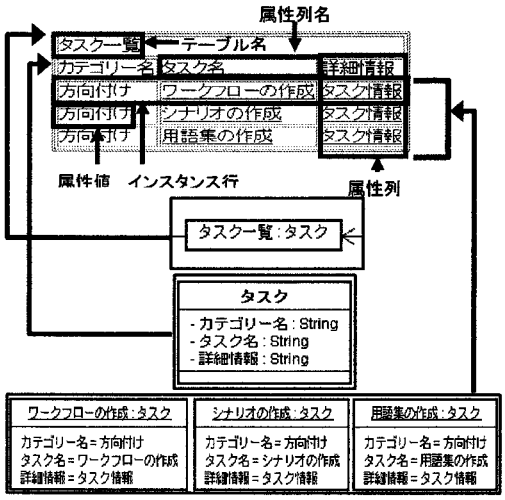

図4　プロトタイプのテーブル構成要素

表4　テーブル構成要素と要求分析モデル要素の対応

図4の構成要素	要求分析モデル要素
テーブル名	オブジェクトノードのオブジェクト名
属性列名	クラスの属性
属性列	クラスの属性およびインスタンス仕様のスロット
属性値	インスタンス仕様のスロット
インスタンス行	インスタンス仕様

属性値の表現形式について，属性列名を含んだアクションが定義された場合は表3に示した形式で表現される．逆にそのようなアクションが定義されていない場合は，インスタンス仕様のスロットをそのまま表示する．また，オブジェクトノードで指定したクラスに対応する全てのインスタンス仕様を取得し，インスタンス行を生成する．その際の属性列の数は指定したクラスの全ての属性の数と同値となる．

例えば図5では，図4のオブジェクトノードに対するアクションに「タスク名を選択する」と「詳細情報を選択する」があるため，対応する属性値はリンク形式になり，それ以外は通常表示となる．

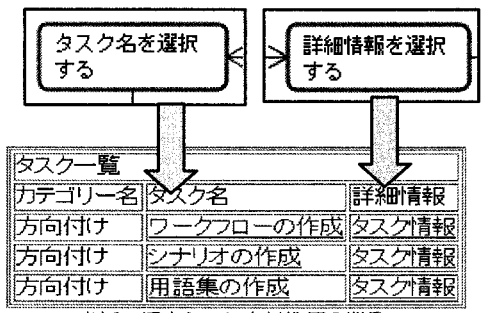

図5　アクションと属性値の関係

4.4　入出力項目データの表示制限

4.3の機能だけでは，プロトタイプは常にオブジェクトノードで指定したクラスの全属性を表示し，定義したインスタンス仕様を全て表示するため，状況に応じた入出力項目データの表示制限が行えない．例えば，図6のようなユーザ情報をプロトタイプ上では，パスワードだけ非表示にしたい場合がある．

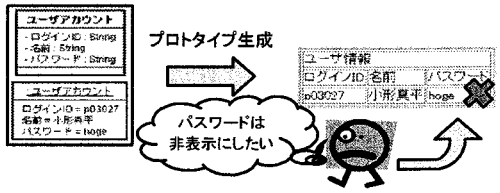

図6　特定の属性列自体を非表示にしたい例

また，プロトタイプ上で検索機能を表現する際に，ある属性値が同値のインスタンス仕様のみを見せたい場合がある．例えば，図7のような掲示板の記事について，B さんの記事のインスタンス仕様のみを取得し，インスタンス行を表示したい場合がある．

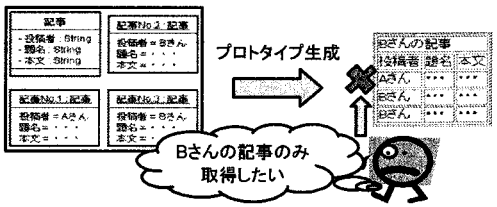

図7　インスタンス仕様の取得数を制限したい例

このように同一のクラスを指定したオブジェクトに対して，それがアクティビティ図に登場する場面によって見える情報を変える必要がある．そこでオブジェクトノードの状態を表5に示す本研究独自の命令により定義し，以下の情報の制限を実現する．

(1)特定の属性列自体を非表示にする．
(2)特定の属性列の属性値を非表示にする．
(3)インスタンス仕様の取得数に制限をかける．
(4)インスタンス行に新規欄として空欄を設ける．

この命令は原則として属性列名を指定して行う．

表5　オブジェクトノードの状態に対する命令

命令の種類	命令値
属性列の表示	表示　または　非表示
属性値の表示	表示　または　非表示
インスタンス行の絞込み	指定属性列の属性値が同値なインスタンス仕様数
新規欄	有　または　無

状態への命令は，表6の概念を踏まえた上で，図8に示す制約構文に則って記述する．

表 6　状態に対する制約構文の概念

概念	説明
属性	クラス(インスタンス仕様)における属性
属性列可視	属性列自体を表示するかどうか
属性値可視	属性の値を表示するかどうか
集合選択	取得するインスタンス仕様に関して，属性値が同値であるものの数が，指定した数値以下の最大のものを取得する
newly	新規欄として，空白のインスタンス行を表現するかどうか

```
状態 := 属性状態?
属性状態 := 対象属性名 "(" 状態命令 ( "," 状態命令)* ")" ( "; " 対象属性名 "(" 状態命令 ( "," 状態命令)* ")" )*
対象属性名 := 文字+
文字 := "[^ ();,]"  --文字"(" ")" "," ";" は使用不可
状態命令 := 属性列可視 | 属性値可視 | 集合選択
属性列可視 := "fieldVisible=" 真理値
属性値可視 := "valueVisible=" 真理値
集合選択 := "set=" 数値 ("newly")?
真理値 := "true" | "false"
数値 := "1" - "9" ( "0" - "9")*
```

図 8　オブジェクトの状態に対する制約構文

図 6 のユーザ情報表示の例，および図 7 の掲示板の記事検索の例について，図 8 の制約構文に則った命令を記述したオブジェクトノードは図 9 に示す通りとなる．

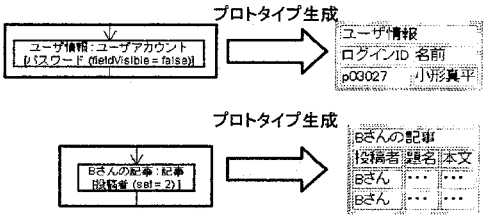

図 9　状態に対する命令によるプロトタイプの変化

インスタンス行の絞込みについて，インスタンス仕様の取得数制限を属性値に対する直接的な条件(例えば，投稿者の値が「B さん」であるという条件)からではなく，集合の要素数指定の考え方で行っている理由として，アクティビティ図のオブジェクト図に対する依存度を低くすることにある．アクティビティ図，クラス図は洗練していく過程で内容が固定されていくことに対し，オブジェクト図はデータの具体例を定義する位置づけにある．そのため，プロトタイプ確認時において，説明のために想定する状況の変化に応じて常に内容が変わる可能性がある．そこで，インスタンス仕様取得数制限の命令の条件では，値ではなく状態のタイプで定義している．

また，新規欄の命令はインスタンス行の絞込みの命令と併用することが原則である．これにより，この命令を記述された属性では，インスタンス行の絞込みにより属性値が一意に決定することができるため，新規欄として設けられた空白のインスタンス行に，その属性列の決定した属性値を表現することができる．もし，オブジェクトノードの状態に命令がない場合は，属性列は表示，属性値も表示，インスタンス行の絞込みは行わず，新規欄は無，となる．

4.5　実装

プロトタイプ自動生成ツールは Java により実装した．要求分析モデルが定義された JUDE5.0 の JUDE ファイルとシステム側のパーティション名をツールへ入力し，HTML で記述された Web ページをツールから出力する．図 10 にツールの処理の流れを示す．

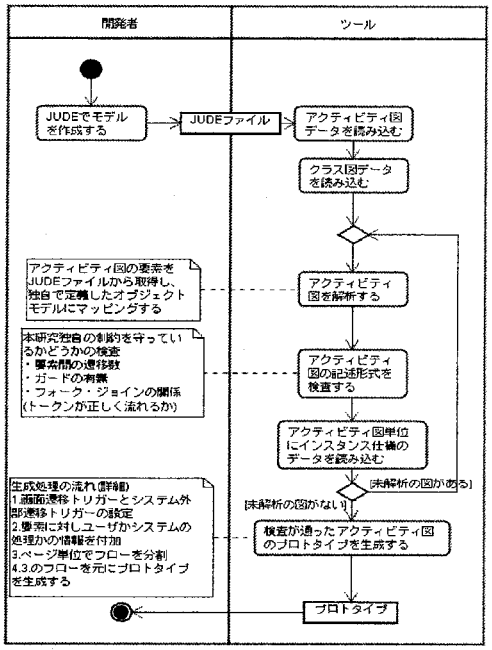

図 10　ツールの処理の流れ

本ツールの特徴は，アクティビティ図のフローの分割方法である．読み込まれるアクティビティ図には，フォークやデシジョンノードによるフローの分岐が存在する．そこで，フロー全体を分岐しない最長の直列フローの単位で分割し，さらに画面遷移トリガーとシステム外部遷移トリガーを分割点として分割する．本稿では，この分割された 1 つのフローを「ブロック」と呼ぶ．このブロック間を直列に連結することで，分岐のあるフローを強制的に直列フローとして読み込むことができる．また，ループのあるフローもブロック間の連結で容易に表現できる．これにより，ブロックの連結方法によって，条件分岐ごとのフローやループを含めたフローなどの種類を容易に表現することが可能となっている．

5　要求分析モデル間の連携

アクティビティ図，クラス図，オブジェクト図の 3 種のモデルを連携させた情報からプロトタイプを自動生成するので，開発者は 3 種のモデルをフローとデータを区別して定義できるため，分析内容の整理が可能となる．一方で，モデルの段階的な定義に応じたプロトタイプ生成を可能にするため，プロトタイプ自動生成ツールはアクティビティ図のみからでもプロトタイプが生成できる．

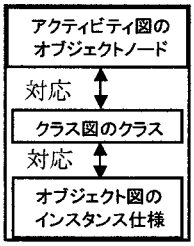

図 11 モデルの連携

図 11 のモデルの要素に関して，
・オブジェクトノードのクラス
・インスタンス仕様のクラス
のクラスを共通にすることで，モデルを連携させて情報取得することが可能となる．従って，オブジェクトノードの定義箇所において，グループ化した入出力項目とその具体例を表現したプロトタイプが生成可能となる．

図 12 に「ユーザ認証する」の認証フォームのアクティビティ(一部)，クラス，インスタンス仕様を示し，図 13 に連携の有無によるプロトタイプの変化を示す．図 13 よりわかるように，プロトタイプをアクティビティのみから生成する場合よりも，アクティビティ，クラス，インスタンス仕様を組み合わせた情報から生成する場合の方が，入出力データ間の関連が直感的に理解しやすい．

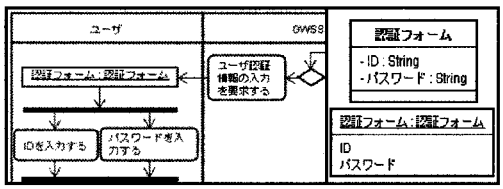

図 12　アクティビティ，クラス，インスタンス仕様

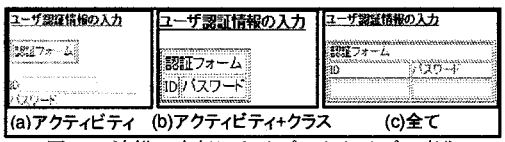

図 13　連携の有無によるプロトタイプの変化

6　開発の後工程におけるモデルの利用可能性

アクティビティ図では，入出力操作レベルのユーザとシステムのインタラクションを定義し，プロトタイプを通してモデルの洗練するため，シーケンス図のアクターが直接メッセージを送るライフラインとのインタラクションに適用しやすい．これは，アクターとバウンダリクラスのメッセージのやりとりと同義の意味をもつ．また，オブジェクトノードの命令記述は，シーケンス図で詳細なロジックを定義する上で，データに対する加工方法の指標となる．

本研究におけるクラス図では，サービスに対する入出力項目のグループ化を念頭おいたクラスの作成を行う．これを元にモノの性質や関連を考慮したクラス図へ洗練していくことで，業務上必要な用語や項目を漏らさず要求分析モデルに定義できる．

本研究におけるオブジェクト図では，システムのサービス実行手順に即した必要データを定義したものであるため，テストデータとして再利用できる．

7　適用事例と考察

7.1　GWSSへの適用

本学の GWSS はシステム開発実習におけるグループワーク支援目的ために開発された．システムの提供するサービスの内，ファイル共有機能に対し本提案を適用した．GWSS では，フェーズ→カテゴリー→タスク→作業項目の順でグループの行う作業を分類しており，ファイル共有機能では，作業項目単位に共有場所を提供することで，グループ内のファイルの共有を実現している．本稿ではプロトタイプの表現力に着目し，この機能の実行手順や入出力項目について，現行システムでの表現と自動生成されたプロトタイプの表現の差異の比較や新システムのためのプロトタイプを生成することで検証した．

7.2　適用結果

確認事項が確認できたかや確認事項に応じたモデル修正が可能だったかを表 7 に示す．

表 7　プロトタイプ確認とモデル修正の適用結果

確認事項	確認	修正
・サービス実行手順としての画面遷移	○	○
・サービスに対する入出力項目の過不足	○	○
・入力項目の入力形式	△	△
・グループ化すべき入出力項目	○	△
・入出力項目の具体例	○	△

○：可能，△：一部可能，×：不可能

画面遷移は，ファイルをアップロードする際の「投稿を選択する」のアクションから生成されたものを示した図 14 からわかるように，どのような遷移条件の元で遷移先が変わるのかを確認できる．実際の投稿という選択肢は 1 つだが，プロトタイプでは，「説明を行うこと」を考慮しており，遷移条件を括弧付きで表現している．

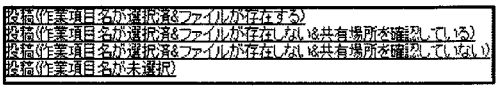

図 14　投稿の遷移条件

修正に関しては，アクティビティ図のフローを変更することや，システム側の処理内の条件分岐を追加・削除することによって修正できる．

入出力項目の過不足は，オブジェクトノードに対応するクラスの属性の変更やアクションの追加・削除により変更が行える．

入力項目の形式は，1 画面中に異なるクラスから生成されたテーブルがあり，同名の属性があった場合，現在のツールの仕様ではテーブルの異なりに関わらず常に同じ入出力形式に変換してしまう．例えば，図 15 は別々のクラスに定義された同名の属性が 1 画面に同時に現れる場合のアクティビティ図の一部の要素とクラスを示している．ここで，四角枠の各オブジェクトノードの指定クラスにはそれぞれ「作業項目名」という属性があるが，「作業項目名」に対するアクションは 1 通りしかない．

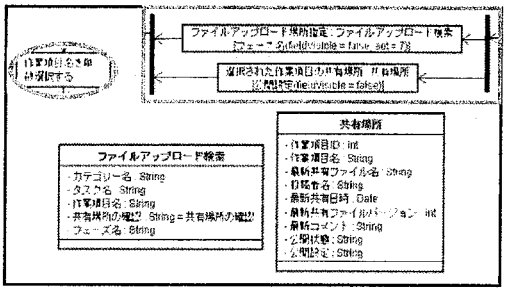

図 15　同名の属性を 1 画面で表示するモデル例

図 16 は図 15 を基に自動生成されたプロトタイプの一部であり，共有ファイルをアップロードするために，アップロード場所指定のテーブル(図中の上)と，指定場所のアップロード状況を確認するテーブル(図中の下)を表現している．ここで，アップロード場所指定のテーブルは「作業項目名」を選択することでアップロード場所指定を行う．一方で，指定場所のアップロード状況のテーブルでは，「作業項目名」は単に名前を表示すれば良い．しかし，生成されたプロトタイプではどちらの作業項目名も選択可能な状態で表示されてしまう．これは，現状のツールではクラス別にアクションと属性を対応付けられないことが原因である．

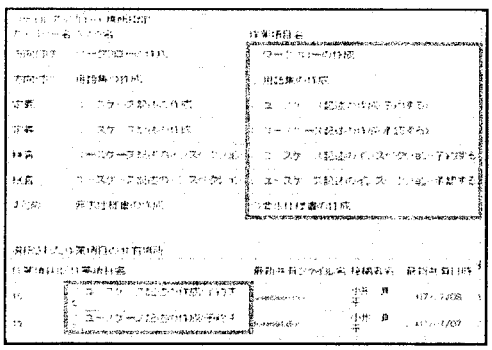

図 16　ファイルアップロード画面(一部)

入力形式に未対応な動詞は，ツールに動詞と対応する入力形式を追加することで修正が可能となる．

グループ化すべき入出力項目は確認ができ，グループ化される入出力項目を追加・変更する修正は行えるが，テーブルのレイアウトの修正が行えない．例えば，ファイルを共有するためのタスク選択画面を現行システムとプロトタイプの両方を図 17 に示す．

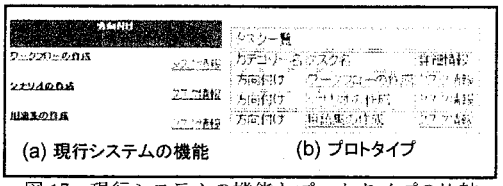

図 17　現行システムの機能とプロトタイプの比較

図 17 より，プロトタイプでは同値の属性値のセルを 1 つにまとめられていないことである．「同じ値ならまとめて 1 つで表示した方が見やすい」という顧客の要求は容易に想定できる．しかし，ツールではインスタンス行間の関連性は考慮していないため，まとめることは現段階では不可能となっている．

入出力項目の具体例は，確認および追加・変更による修正は行えるが，ツールのインスタンス仕様の取得順序に対して明確に順序を定義できないため，インスタンス行の順番を変更することや，特定のインスタンス行に対しての命令記述が行えない．

7.3　考察

7.2 の適用結果以外の問題として，図 17 から「カテゴリー名」，「タスク名」，「タスク情報」と言った列名が現行システムでは表示されないことに対し，プロトタイプでは表示されている．このように，プロトタイプでは現行システムよりも情報量が多い場合がある．これは，プロトタイプを確認してもらう上で「説明のための表示」が必要だからである．しかし，顧客は実際のシステムの情報量を確認したい場合があるため，「実際を想定した情報量による表示」を別途実現する必要がある．ユーザ認証を例としたプロトタイプと実際を想定した表示の差異を図 18 に示す．

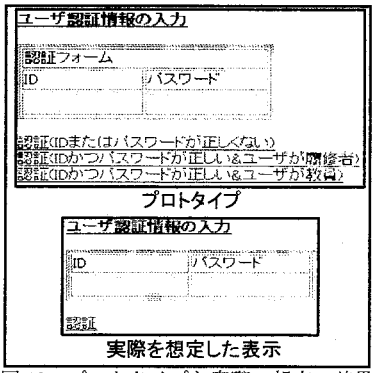

図 18　プロトタイプと実際の想定の差異

これを解決するために，要求分析モデル，または，ツールによるプロトタイプ自動生成過程において，表示方法の種類を指定することによって，プロトタイプの表現を変化させる必要がある．ただし，実際を想定した表示の場合は遷移条件の記述を省く必要があるため，全ての遷移については表現しきれない部分があり，開発者記述のサービス実行順序を表現したシナリオに基づいてプロトタイプ生成を行う枠組みが必要である．

8 関連研究

Juan ら[4]の研究では，ステレオタイプなどを追加することでシーケンス図を拡張した MSC (Message Sequence Chart)から，プロトタイプを自動生成している．大まかな開発の流れとして，

(1)ユースケースを割り出し，ユースケース単位でシナリオを記述することで，ユースケースモデルを作成する．

(2)ユースケースモデルを非曖昧な言語である MSC によって統合する．

(3)MSC からにユーザインタフェースの部品であるフォームとその画面遷移を表現した状態遷移図を生成し，そこからプロトタイプを生成する．

目的は本研究と同じところにあるが，Juan らの研究では MSC のメッセージの流れを振る舞いとし，メッセージの引数を入出力項目として定義していることに対し，本研究では振る舞いの定義とデータ構造の定義をアクティビティ図，クラス図，オブジェクト図として区別している．

MSC はシーケンス図ベースなため分岐や繰り返しなどの柔軟な表現力に乏しく，複雑なフローの場合に入出力項目の情報も含めて可読性が低くなる可能性がある．本研究では，分岐の表現力が高いアクティビティ図を用いているため，複雑なフローでもシーケンス図と比べて，可読性を保つことができ，内容量が多くなった場合でも，入出力項目は別途にクラス図やオブジェクト図に定義するため，アクティビティ図の記述が煩雑になることをある程度抑制することができる．

一方で，Juan らの研究では，全体のユースケースモデルに対し，アクター毎に対し必要なユーザインタフェースのコンポーネントの連携を遷移状態図に近いナビゲーションモデルで表現できる．本研究ではこのような全体の流れを提示するような機能は現段階ではサポートしていないが，これは全体の流れを確認する上で非常に重要なモデルと考えており，アクター毎の全体のユースケースに対するアクティビティ図の統合方法を検討している．

9 まとめ

本稿では，アクティビティ図，クラス図，オブジェクト図を連携させることにより，入出力項目の表示に具体的なデータを表示できるようになり，直感的な理解の容易さを向上する意味では，プロトタイプの表現力の向上が実現できた．しかし，適用事例により新たな問題も出てきた．それは，使用性に関わるデザインの問題や，プロトタイプの実際的な利用方法に関わる問題であった．

今後は既出の問題の解決も考慮しつつ，顧客が
(1)サービス実行手順としての複数サービス間の連携
(2)システムの利用権限別に実行可能なサービス
の 2 つを確認できるプロトタイプを生成することを目標とする．具体的には，(1)に対しては，ユースケース単位で表現していたアクティビティ図に対し，ユースケース自体の実行順序を表現するアクティビティ図を定義し，ツールにその関係を解釈するような処理を加えることで解決を目指す．また，(2)に対しては，権限をアクターに置き換えて考え，プロトタイプを生成する際にアクターを指定する．そして，ツールに，指定されたアクターがパーティション名として関わるアクティビティ図のみを解釈するような処理を加えることで解決を目指す．

10 参考文献

[1] 大西淳，郷健太郎，要求工学，共立出版，2002

[2] 小形真平，松浦佐江子，UML・プロトタイプを組み合わせた要求仕様の妥当性確認，第 69 回情報処理学会全国大会，6L-7，2007

[3] 山内亨和 監修，その場でつかえるしっかり学べる UML2.0，秀和システム，2006

[4] Juan Sánchez Diaz，Oscar pastor López，Juan J. Fons，From User Requirements to User Interfaces: A Methodological Approach，CAiSE 2001，LNCS 2068，pp. 60-75，2001