B-015

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の実験例

An Experiment on Stepwise Automatic Generation of Web UI Prototypes based-on a Requirements Analysis Model in UML

小形 真平† Shinpei Ogata　松浦 佐江子‡ Saeko Matsuura

1. はじめに

システム開発の初期段階では，顧客要求は顧客と開発者間で誤解が生じないように，非あいまいに定義し，反復的に要求の妥当性を確認することが肝要である．かつ，定義した要求は顧客と開発者の双方にとってシステムを想起できるものでなければならない．そのため，開発者が分析した要求とシステムの対応を開発者間で誤解なく理解でき，かつ，顧客がシステムイメージを正しく理解できなければならない．顧客がシステムイメージを理解するための有効な手段として，プロトタイピング[1,2]がある．そこで，我々は Web アプリケーションの業務系システムを対象に，UML(Unified Modeling Language)[3]により定義した要求分析モデルから UI(User Interface)プロトタイプを自動生成する手法を提案してきた[4]．本研究では，要求分析モデルとして一種類ずつ図を追加する各段階でプロトタイプが生成できる．そのため，開発者は短いサイクルでプロトタイプを通して定義モデルが自身の意図を反映しているかを確認できる．また，具体的なデータを取り込むことで明確なイメージを持つプロトタイプを生成できる．図 1 に提案手法の概要を示す．

しかし，モデル記述の難しさは同じ記述言語を用いたとしても、開発者により，モデル要素に当てはめる単位の基準（例えば、アクティビティ図中のアクションや遷移上のイベント・ガード・アクションへの要求の分割の仕方）や、モデル要素の記述に用いる用語は異なる。

今回の実験は、これまでの開発事例の開発者とは異なる開発者による異なる問題の開発実験により、このような問題や他の事例に対して、現在の方法が問題なく適用できるのかを確認する．

†芝浦工業大学大学院工学研究科電気電子情報工学専攻, Department of electronic engineering and computer science Graduate School of Engineering Shibaura Institute of Technology

‡芝浦工業大学システム工学部電子情報システム学科, Department of Electronic Information Systems College of Systems Engineering Shibaura Institute of Technology

2. 注文管理システム開発概要

注文管理システム開発は，大学の講義に用いられている架空の会社（S 製菓）における開発事例である．本実験は提案ツールの実装に関与していない開発のエキスパート 1 名の協力の下に行った．注文管理システムは，注文受理から商品の出荷まで，現状(システム未導入)の業務手順の内容を損なわない程度にシステムを導入することが主な要求となる．また，既存の顧客管理システム

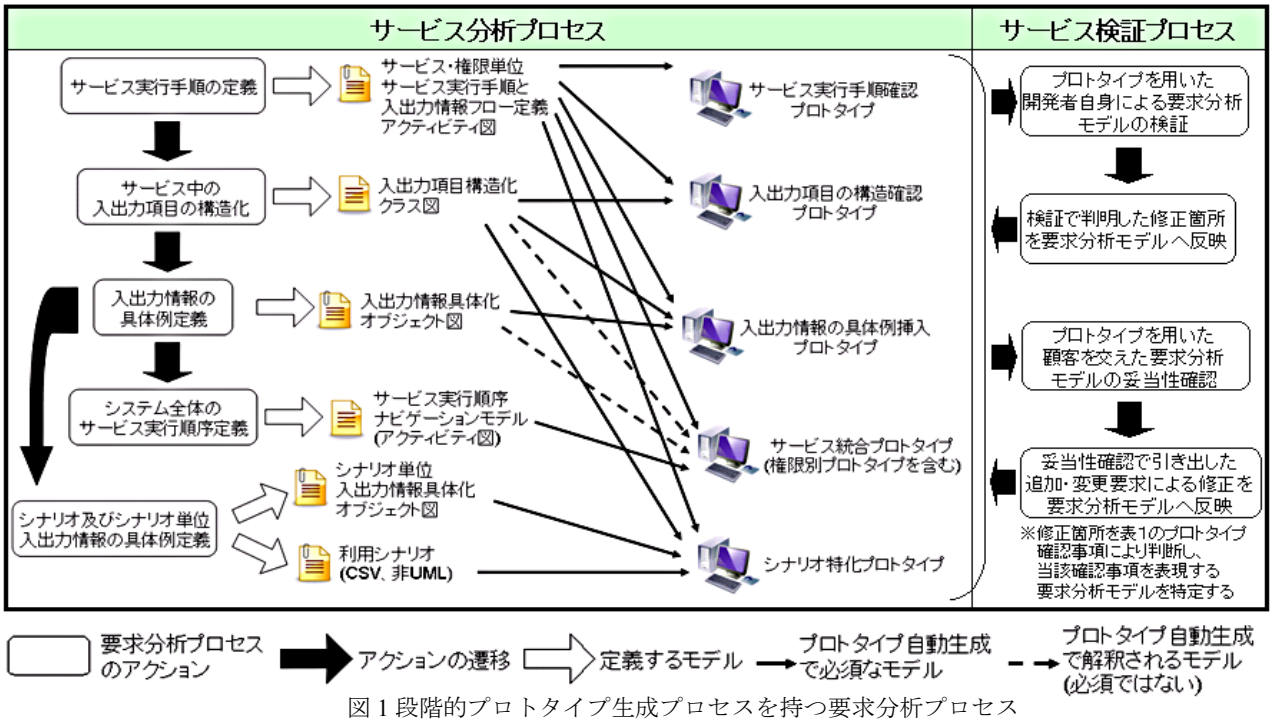


図 1 段階的プロトタイプ生成プロセスを持つ要求分析プロセス

ムと連携させて，現状ではそれぞれ別個に扱っていたシステムを統合しなければならない．

3. サービス導出プロセスと要求分析手順

本研究では，最初のステップとして，アクターとシステムのインタラクションをユースケース・権限単位でアクティビティ図に定義する．しかし，これまで初期要求からユースケース導出までのプロセスを明確に述べていない．その点を含めて以下のプロセスで行った．なお，モデルは全てJUDE[5]により定義する．

（1）顧客の主たる要求（商品を注文して購入する）に着目し，顧客とS製菓の振舞いの流れを定義する．顧客とS製菓の間でやり取りされるデータを定義する．

（2）役割ごとにS製菓の作業を詳細化する．

- 作業する人の役割を認識する．
  ⇒ アクターの候補を抽出する．
- （1）で定義したデータの状態を定義する．作業の結果データの状態が変化する点を探す．

（3）システム化する範囲を認識する．

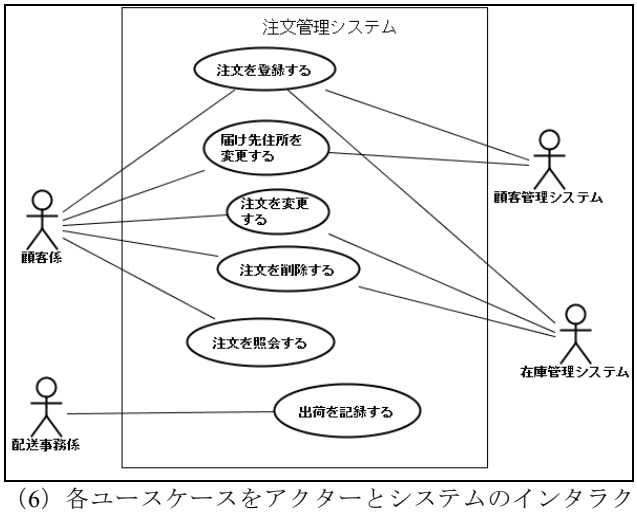
- システム化する作業としない作業・誰がシステムを利用するのかを明らかにする．
  ⇒ システムが提供するサービスを導出する．

（4）データの取得方法を検討する．外部システムの顧客管理システム，在庫管理システムとの連携を検討する．

- システムが提供するサービスに必要なデータを外部システムから得るか否かを検討する．
- 外部システムを使用しない場合には，内部に想定するデータとその取得方法を検討する．

（5）ユースケースを抽出する．

抽出したユースケースは下図の通りである．



（6）各ユースケースをアクターとシステムのインタラクションとしてアクティビティ図を定義する．ここではサービスを提供可能とする正当な入力値とサービスを構成するアクションがすべて成立するという前提のもと，基本フローと代替フローのみを分析する．

⇒ サービス成立に必要な入出力データを十分に洗い出し，サービスのアクション列を定義する．

（7）例外を分析する．例外はサービスロジックが不成立となるデータを排除し，適切な処理を施すことを目的とする．サービスロジックが不成立となるケースは

1) サービスへの外部からの入力が型および値の制約を満たさない場合
2) あるアクションが制約を満たさない場合

これらの場合も含めてサービス（ユースケース）のフローを分析すると煩雑になるため，上記の不成立になるケースのポイント（外部からの入力データ・アクションの制約を確認する位置）に着目し，アクターとシステムの間のインタラクション部を導入して分析する．

（8）アクティビティ図中のアクション列や，データであるオブジェクトノードを基にデータ構造としてクラス図を定義する．

（9）定義したデータ構造を基に実際のデータを想定したものとしてオブジェクト図を定義する．

（10）ユースケース・権限単位のアクティビティ図を統合するためにナビゲーションモデルを定義する．

（7）～（10）のプロセスにおいて，提案手法で示すプロトタイプ自動生成機能を用いて，各段階でプロトタイプを確認しながら，モデルを洗練した．一方で，本提案で考慮していなかった（1）～（5）までのプロセスでは，アクティビティ図を定義している．しかし，サービス・権限単位でアクターとシステムのインタラクションを定義するルールに沿って定義したものではないため，プロトタイプは生成できないものであった．また，（6）のプロセスでは，プロトタイプが生成できるルールで定義されたアクティビティ図を定義したが，この段階ではプロトタイプを生成することはなかった．この理由は，サービス単位で定義する基本・代替のみのフローは複雑にはならないため，開発者はアクティビティ図上で確認するのみで十分に確認できると判断したためである．現段階では，シナリオ及びシナリオ単位の具体例定義プロセスには至っていない．なお，プロセス（6）以降に定義したモデルの一部を図2，図3に示す．

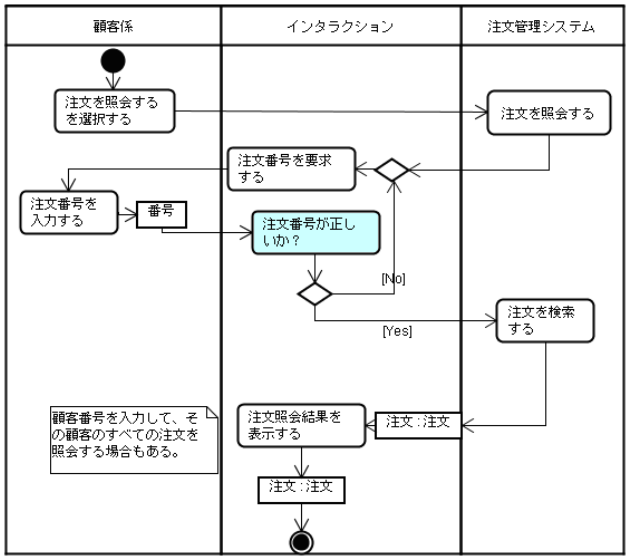


図2 「注文を照会する」アクティビティ図

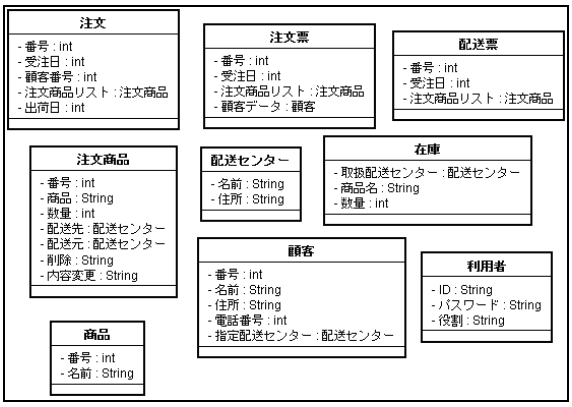


図 3　注文管理システムのクラス図

4. 実験結果と考察

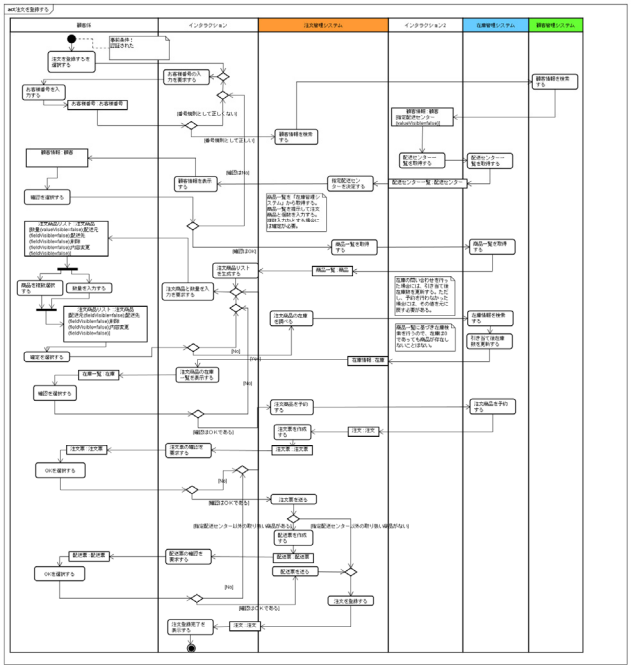
実験結果から，提案手法を用いて有効であった点を考察する．また，開発者として提案手法の利便性や要求定義段階として本質的に不足していた点について問題のあった点を考察する．

◆　提案手法利用時の有効点

● 段階的なモデルの定義と洗練

提案手法では，サービス実行手順の定義，データの構造化，データの具体例の定義，サービスの統合と言う各指標の下に段階的にモデルを洗練する．そのため，各 UML モデルに対し，明確な指標を持って定義できる．

例えば，アクティビティ図に定義されるフローは要求を十分に定義した場合，下図のように制御フローが 60 本になるような複雑なものがある．しかし，オブジェクトノードに注目しクラスを定義することで，アクティビティ図の煩雑さの影響を抑えた形でデータ構造をサービス間で整合性を保って定義できる．図 3 に掲載したクラス図は全てのサービスで整合性をデータ構造である．モデルを各観点で切り分けて定義し，また，プロトタイプ確認で各モデルを洗練することで，開発者のモデル定義の複雑さや混乱を軽減して定義することができた．



● 短期サイクルにおけるプロトタイプ確認

3 節で説明したように，例外フローが定義されて複雑なフローになったアクティビティ図からサービス統合までのモデル定義まで各段階でプロトタイプを生成できる．

プロトタイプによる確認の有効性は，開発者の意見として以下の点にまとめられる．

入力や選択といった必要なデータを提供する行為については、アクティビティ図の記述の際にも漏れがあることは少ない。これらの場合には、行為そのものではなくサービスに必要なデータの漏れを発見することが重要であり、これは具体例の提示により発見できた。

しかし、「確認する」といった行為は、実際にページを辿り、「ここで、このことを確認したい」ということを実感することによって発見できることがあり、アクティビティ図のフローでは気がつかなかったアクションを見つけることができた。

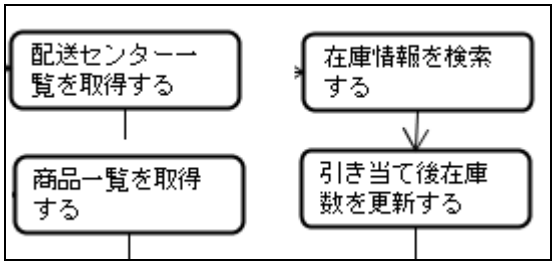
◆　要求定義時の問題点

● 外部システムと外部ユーザの区別

提案手法では，これまで外部ユーザと外部システムの 2 種のアクターは開発システムと相互作用するという点で共通なものとし，明確な区別を与えなかった．しかし，以下の観点において，プロトタイプ上で確認したい点が異なっていた．

1) 外部ユーザとのインタフェースでは入出力項目や形式が重要である．
2) 外部システムとのインタフェースではどのデータが変化して返戻されるのが重要である．

また，モデルの理解性向上では語の選定が重要な要素となるが，アクターの種類に依らず定義したルールとしての語を用いることへの違和感が問題となった．下図は実際に外部システムの振る舞いとして定義されたアクションの一部である．

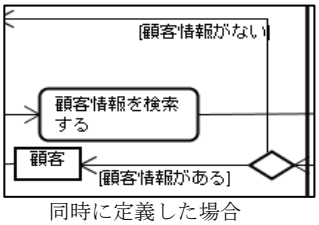


この問題は外部システムと外部ユーザを明確に区別し，要求分析段階で確認すべき事項を整理した上でルールを定義する．

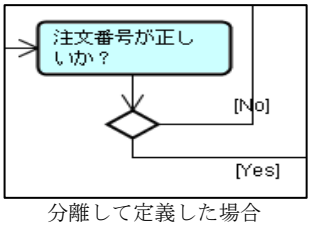
● 条件定義に対応するアクションの考慮

フローの分岐条件としてアクティビティ図のデシジョンノード直後の制御フローにガードを定義する．特に開発システム中のガードは生成されるプロトタイプ中では画面遷移の条件として表現される．提案手法は，これまで開発者がガードに条件内容と条件値を同時に定義する方式を想定していた．例えば，「注文商品と個数の入力が正しい」というガードの場合，「注文商品と個数の入力」という条件内容に対し「正しい」という条件値を示している．しかし，今回の開発者は「注文商品と個数の入力は正しいか？」という条件内容をアクションに定義し，その後の制御フローに「yes」または「no」の条件値をガードに定義した．前者の方式は，条件としての意図が明確になる反面，条件内

容を精査するアクションをガードに含めることになる．一方，後者の方式は，条件内容を精査するアクションを明確に定義できる反面，そのアクションと条件値を定義するガードを明確な対応付けが必要となる．下図に記述例を示す．この問題は開発者の考え方による方式の選択の差異と考えられるため，本研究では，どちらかの方式に選ぶよりも，両方の定義方法を吸収できる機構をツールに導入することを検討する．



同時に定義した場合



分離して定義した場合

- 階層の深いデータの構造化

データ構造を定義する際，階層の深い構造が定義できる必要がある．提案手法では，データ構造をクラスとして定義しているが，クラス間関連を考慮せずにプロトタイプを生成するため，1 階層のみのデータ構造しか表現できない．この問題はクラス間の関連，及びデータの具体例となるオブジェクト図のインスタンス仕様間の関連を解釈する機構をツールに導入する．そして、深い階層構造を持つデータ構造をテーブルとしてプロトタイプ上で表現することを検討する．

- 単一データと集合データの区別

インタラクション上で扱われるデータは大別して，ただ 1 つデータを含む単一データか，0 以上のデータを含む集合データとなる．その種別によって，インタラクション上で考慮すべきポイントが変わってくる．例えば，集合データは 1 つもデータを含まない場合に，システムの処理を考慮する必要がある．提案手法では，単一データと集合データを明示することができなかった．この問題はオブジェクトノードに対し，単一データか集合データの種別をステレオタイプとして定義することで識別することを検討する．

- 複数サービスの並列表現

複数のサービスは並列実行される場合がある．例えば，商品を検索しながら注文を行う場合，サービスとしては検索後のみに注文を行う場合もあるが，検索と並行して注文を行う場合も考えられる．提案手法では，後者を実現するための手段がない．この問題はサービス統合時に用いるナビゲーションモデルに並列表現を導入することと，プロトタイプに反映可能な画面遷移に特化したモデルを導入することを検討する．

◆ ツールの利便性に対する問題点

- 段階的に洗練したモデル間の整合性検査

提案手法では，自動生成されるプロトタイプを用いて段階的にモデルを定義・洗練できるという特徴がある．そのため，開発者は 1 つのサービスに対するモデルにバージョンを付記し，管理する場合がある．そしてバージョンが上がるごとに，定義した要素を削除する等の手戻り箇所が散在する可能性は高い．そのため，意図しない要素削除が起こっていないかどうかをバージョン間で確認する必要がある．そこで，モデルのバージョン間の整合性を UML 本来の記法と提案手法で定義したルールから検証する方法を検討する．

5. まとめと今後の課題

本稿では，開発ツールの実装に関与していない開発者の観点で提案手法の有効性を検証し，問題点を抽出した．3 節で述べたように，アプリケーションドメインを段階的に整理しながら，分析することで開発者にとって解りやすく分析を進めることができた．

今後の課題として，前述の問題点を解決するとともに，要求分析時に定義したモデルが開発の後工程でどのように扱われるかを検証する．

参考文献

[1]Sommerville, I., Sawyer, P., 富野壽 監訳, 要求定義工学プラクティスガイド, 共立出版, 2000
[2]大西淳, 郷健太郎, 要求工学, 共立出版, 2002
[3]Object Management Group：http://www.omg.org/
[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シンポジウム 2008, 掲載予定
[5]JUDE：http://www.change-vision.com/