2004－IS－88　(4)
2004／5／21

日本語要求記述に基づくプロトタイプ作成支援ツールの開発

小林　孝弘[†1]　　中鉢　欣秀[†2]　　大岩　元[†3]

日本語による要求記述から，システムを特徴付ける機能およびデータを提供する部分の Java プログラムを生成することにより，プロトタイプ作成を支援するツール，Toriaezer を開発した．この要求記述とは，ユーザのシステム導入後の業務手順と，その中でのシステムに対する操作から成る「業務手順書」である．プロトタイプ作成者は，ユーザとの密な協調により，ユーザの満足する業務手順書を作成する．そしてその中で使用した言葉について実装方法を定義し，Toriaezer によりプログラムを生成する．これにより，要求に忠実なシステム本質部分のプログラムを素早く生成することが可能になる．

A Support Tool for Prototype Development based on Japanese Requirement Description

Takahiro Kobayashi[†1]　　Yoshihide Chubachi[†2]　　Hajime Ohiwa[†3]

Toriaezer is a tool which generates the Java program of the essential portion of a system from Japanese requirement description. This requirement description is an operating procedure document which describes the operating procedure after system introduction, and the operation to a system in Japanese. A prototype maker draws up an operating procedure document in harmony with a user, and defines implementation of the words used in it. By making them read into Toriaezer, the program of a system essence portion faithful to a requirement can be obtained quickly.

1　はじめに

システムのプロトタイプを利用してユーザの要求を収集する開発アプローチがある [1]．このアプローチの目的は，ユーザにシステム利用の実際的な体験をさせることで，より確かな要求を得ることである．システムを評価する際，文書や抽象的な図解などで説明されるよりも，システムそのものを操作してみた方がはるかに分かりやすく，質の高い要求が得られる可能性が高い．

プロトタイプの完成度は高ければ高い程よいが，そのようなプロトタイプの作成には時間がかかり，限られた時間の中での開発において現実的でない．そのため，ユーザに評価させたい部分のみ入念につくり，残りは間に合わせ的につくるという方針をとることになる．

このとき，プロトタイプについてユーザに必ず評価させるべきは，システム本質部分に関する要求である．これはシステムにとって最も重要な部分であり，本番のシステム開発の際に後々の変更が最も難しい部分であるからだ．これを確実に捕捉するために，プロトタイプはその時点のシステム本質部分に関わるユーザの要求を確実に反映したものである必要がある．

Toriaezer は，日本語で記述された要求記述から，システムを特徴付ける機能およびデータを提供する部分（以降システム本質部分と呼ぶ）の Java プログラムを生成するためのツールである．

要求記述は，ユーザのシステム導入後の業務手順，およびシステムに対する操作から成る「業務手順書」である．プロトタイプ作成者（以降作成者と呼ぶ）は，ユーザとの密な協調の中でユーザの納得する業務手順書を作成する．そしてその中で使用した言葉の実装方法を定義し，Toriaezer によりプログラムを生成する．これにより，要求に忠実なシステム本質部分のプログラムを素早く生成することが可能になる．

2　背景

プロトタイプを一般的なオブジェクト指向開発アプローチによって開発しようとする場合，以下のような手順を踏む．

手順 1　システム要求定義（ユースケースの作成）

手順 2　システム静的側面の設計（クラス図の作成）

[†1]有限会社小原メディカルサービス
Obara Medical Service Inc.
[†2]慶應義塾大学政策・メディア研究科
Graduate School of Media and Governance, Keio University
[†3]慶應義塾大学環境情報学部
Fuculty of Environmental Information, Keio University

手順 3 システム動的側面の設計

手順 4 システム実装

この手順によって前述のようなプロトタイプを作成しようとした場合，以下のような問題がある．

1. **作成者の恣意性が混入しやすい**
2. **作成の効率が悪い**

1 について，上記の手順にはユーザ要求がプロトタイプとして作成されるまでの間に作成者の解釈が多分に介在している．

例えば，上記手順の中でシステム要求定義を行う際にはユースケースが用いられる．ユースケースはユーザの視点によるシステムの振る舞いを識別し要求を定義する [3] ものであるが，この過程において要求の表現や単位の切り出し方は作成者の解釈によって行われる．

要求からシステム静的側面の設計を導き出す際には，要求に含まれる名詞をクラス候補として抜き出し，それを吟味することによってクラス図を作成する [3] ということが行われる．このとき，そのクラス候補がクラスとして妥当であるかどうか，およびその構成が適切であるかどうかの判断は作成者が行う．作成者の力量によっては冗長な設計を行ってしまったり，無駄に高度な設計を行ってしまうことがある．

このようにして作成されたユースケースおよびクラス図をもとに，動的側面の設計，更には実装が行われるため，作成されるプロトタイプは必然的に恣意性混入の可能性が高いものとなる．

2 について，システムの実装を行う際，プログラムの随所に似たようなロジックを実装することがある．特にそれはシステム本質部分において顕著である．

例えば，属性の取得または設定といったメソッドは，システム本質部分を構成するほとんどのクラスに実装される．これらのメソッドは処理対象が異なるだけでアルゴリズムは同じであるため，そのようなメソッドをひとつずつ実装していくのは非常に効率が悪い．

プログラムの書き方を工夫することで，同じロジックは一度しか書かないといったことは不可能ではないが，無理にこれを行おうとすると恣意的な実装を行うこととなり，1 の問題を助長することとなる．

3 Toriaezer

要求に忠実なプロトタイプを素早く作成することを支援するツール，Toriaezer を開発した．Toriaezer は「業務手順書」を解析することによりシステム本質部分の Java プログラムを出力することができる．Toriaezer は，以下の 2 点を実現する．

1. **要求に忠実なシステム本質部分の実装を生成する**
2. **迅速なプロトタイプ作成を可能にする**

1 について，Toriaezer は、業務手順書から直接システム本質部分の実装を生成する．業務手順書は日本語の簡単な記法に基づいて記述されるため，ユーザによる評価が可能である．ユーザと密に協調し，ユーザが満足する業務手順書を用意すれば，要求に忠実な実装が自ずと得られる．

2 について，Toriaezer によるプログラムの生成は半自動的である．作成者が業務手順書の中で使用した言葉について実装方法を定義することにより，Toriaezer はそれに従ったプログラムを生成する．これにより作成者がシステム本質部分を実装する手間が削減されるとともに、バグ混入を防ぎ，プロトタイプを素早く作成することを可能にする．

Toriaezer を利用してシステム本質部分のプログラムを得るには，以下のような手順を実行する必要がある．本章では，この手順に従って，Toriaezer の仕組みについて説明する．

手順 1 業務手順書の作成

手順 2 名詞・動詞の実装定義

手順 3 システム本質部分のプログラム生成

3.1 業務手順書の作成

業務手順書とは，システムサポート対象業務におけるシステム導入後の業務の手順（業務手順）と，その業務手順の中でユーザが行うシステムに対する操作（システム操作）を日本語で記述したものである．

業務手順書は日本語の簡単な記法による記述が可能であるため，ユーザによる評価が可能である．ユーザと開発者が協調して作成した業務手順書をもとに Toriaezer を利用することで，要求に忠実なシステム本質部分のプログラムを素早く得ることができる．

以下では業務手順書を構成する，業務手順とシステム操作の記述の仕方について述べる．

3.1.1 業務手順の記述

業務手順については，一行単位で記述すること以外に記法上の制約はない．従って，自由な表現が可能であり，ユーザに書かせることも可能である．

ここで記述する業務手順とは，システム導入後にその業務においてユーザが行う行動である．従って，この中には当然システムを利用する手順が含まれている．このような手順に対し，次節に述べるようなシステム操作を記述していくことになる．

～ 省略 ～
5. 受付係は患者に問診表の作成を求める.
6. 受付係はシステム上で診察受付を行う.
「患者名簿」に「患者」を「追加する」
「ID」を「設定する」
「氏名」を「設定する」
～ 省略 ～

図 1: システム操作の記述例

メソッド名	説明
環境	名詞と，環境を示す助詞（に，から 等）で成り立つ名詞句の型
目的	名詞と，目的を示す助詞（を 等）で成り立つ名詞句の型
引数	名詞と，引数を示す助詞（と，で，により 等）で成り立つ名詞句の型
操作	動詞の型

表 1: 型の種類

3.1.2 システム操作の記述

システム操作は，入出力方式を無視した，ユーザのシステムに対する意味的な操作について記述するものである．システム操作は業務手順に対して階層構造で記述される．図 1 はその例である．

システム操作は名詞，助詞，動詞の組み合わせで構成される．このうち名詞と動詞は ”「 ”, ”」”でくくる必要がある．

助詞は，名詞に「型」を与える働きをする．型には「環境」,「目的」,「引数」,「操作」の 4 種類が存在し，表 1 のように定義される．なお，それぞれの型を示す助詞の値は，Toriaezer の設定ファイル上で任意の設定が可能である．

システム操作の記述は，型が環境，目的または引数である名詞句を任意の順番に並べたものの末尾に，動詞を一つ配置する形で行う．このとき，引数型の名詞句は任意数記述することができる．その他の名詞句は一つしか記述できない．

特に環境型の名詞句に関しては，システム操作記述の階層構造の中で最上位階層のものについてのみ記述する．すなわち，業務手順の直下に配置されるシステム操作に関してのみ環境型の名詞句を記述しなければならず，その他の階層においては記述してはならない．その理由は，親階層のシステム操作における目的型の名詞句が，暗黙に子階層の環境型の名詞句として引き継がれるからである．

これはシステム操作を記述する際に，思考を整理するための工夫である．例えば図 1 のシステム操作記述（太字部分）において，業務手順直下に

- 「患者名簿」に「患者」を「追加する」

とシステム操作を記述した場合，その子階層では親階層から引き継いだ「患者」を暗黙の環境として

- 「氏名」を「設定する」

というような記述を行うことになる．これは

- 「患者」に「氏名」を「設定する」

を記述することと同義である．すなわち，まず患者名簿に患者を追加し，その患者に対して氏名を設定する，と読むことができる．

このように，システム操作の記述は，ユーザのシステムに対する操作手順に従ったツリー構造で構成することができるため，論理的な思考が行いやすい．また，このような構成は直感的で全体像を掴みやすく，業務手順書の可読性向上にもつながる．

但し，システム操作の記述は，暗黙的にシステムの内部的な構造を記述する必要があるため，ユーザに書かせるのは難しい．例えば，図 1 における最上位階層のシステム操作における環境型名詞句として,「患者名簿」を発想することをユーザに期待することは難しい．しかし，システム操作記述が，入力方式を無視したシステムに対する意味的な操作であることを説明すれば，ユーザによる評価は十分可能である．

システム操作をユーザに評価してもらい了承を得ることは，作成するプロトタイプから作成者の恣意性を排除するために非常に重要である．Toriaezer は，システム操作を解析することによりプログラムを生成するため，ユーザの納得するシステム操作を用意するれば，自ずとそれに忠実なシステム本質部分のプログラムを得ることができるからである．

3.2 名詞・動詞の実装定義

Toriaezer を利用してプログラムを生成するには，業務手順書の中で使用した名詞および動詞について，その実装方法の定義を行う必要がある．

本節ではその方法について述べる．

3.2.1 名詞実装定義ファイルの作成

Toriaezer は，システム操作記述の中で使用された名詞から，クラスまたは変数を生成する．

名詞実装定義ファイルは，システム操作記述中で使用された名詞について，その実装情報を定義する．作成者はこれをテキストファイルとして作成する必要がある．表 2 はその定義の内容である．

3.2.2 動詞実装テンプレートの作成

Toriaezer は，システム操作記述において使用された動詞から，フィールドまたはメソッドの実装を生成

メソッド名	説明
名詞	名詞の字面
継承する名詞	継承する名詞の字面
クラス名	名詞の実装時クラス名
変数名	名詞の実装時変数名
唯一性	この名詞が実装されるクラスがこの変数により唯一に識別できるか

表 2: 名詞定義の内容

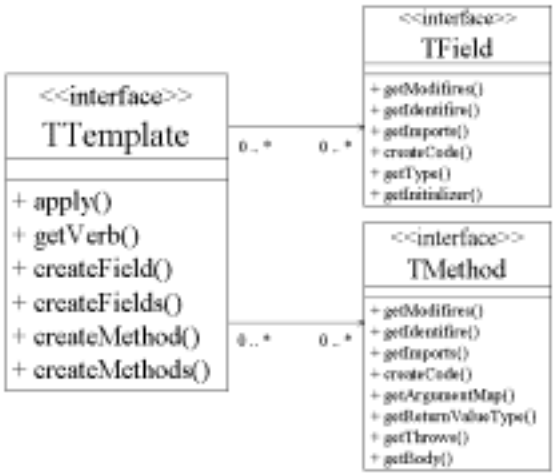

図 2: 動詞実装テンプレートを定める Java インターフェース

する．

動詞実装テンプレート（以降テンプレートと呼ぶ）は，動詞が所属するシステム操作中の名詞に応じた，メソッドおよびフィールドの実装方法を定義する．作成者はこれを，Toriaezer が定義する仕様に従った Java プログラムとして作成する必要がある．

Toriaezer はそのライブラリの中で，テンプレートのプログラムが満たすべき仕様として，Java インターフェース（TTempalte，TMethod，TField）を提供している．図 2 は，その Java インターフェースの構成を表すクラス図である．

図中において TTemplate はテンプレートに対応し，TMethod および TField は，テンプレートが生成するメソッドおよびフィールドに対応する．

Toriaezer は，メソッドおよびフィールドの生成を行う際，動詞に対応するテンプレートに，関係する名詞の実装定義を作用させる．このために，TTemplate インターフェースの apply() というメソッドが用いられる．このメソッドは以下の引数をとる．

- 目的型の名詞の実装定義
- 引数型の名詞の実装定義リスト

作成者はテンプレートを作成する際，apply() メソッドが呼び出されることによって得られる目的型および

メソッド名	説明
apply()	このテンプレートに目的型の名詞および引数型の名詞リストの実装定義を適用する
getVerb()	このテンプレートが実装を生成する対象となる動詞を取得する
createField()	フィールド (TField 実装クラスのオブジェクト) を作成して返す
createFields()	フィールドを複数作成して返す
createMethod()	メソッド (TMethod 実装クラスのオブジェクト) を作成して返す
createMethods()	メソッドを複数作成して返す

表 3: TTemplate インターフェースの仕様

引数型の名詞の実装定義をもとに，メソッドおよびフィールドを生成するようテンプレートのプログラムを記述する必要がある．

なお，表 3 は TTemplate が定めるメソッドの仕様である．TMethod および TField については，それが定める全てのメソッドが，メソッドあるいはフィールドのプログラム要素を文字列として返す．例えば TMethod，TField の両方が持つ getModifier() は，メソッドあるいはフィールドの修飾子を文字列として返すメソッドである．

3.3 システム本質部分のプログラム生成

作成者は，作成した業務手順書，名詞実装定義ファイル，動詞実装テンプレートを Toriaezer に読み込ませることにより，システム本質部分のプログラムを得ることができる．

本節では，Toriaezer による業務手順書の解析およびプログラムの生成の仕組みについて説明する．

3.3.1 業務手順書の解析

Toriaezer は，業務手順書を一行ずつ読み込み，個々のシステム操作記述について，それを構成する名詞あるいは動詞を取り出し，その型を判別する．

図 3 はその仕組みを表している．まず Toriaezer は，名詞の型を判別するための助詞を自身の設定ファイルから読み取り，環境，目的，引数のそれぞれの型に割り当てる．そして，1 行ずつシステム操作記述を読み取り，名詞の後につく助詞によって名詞の型を決定し，抽出する．

動詞に関しては，それがシステム操作記述の末尾に配置されていれば，自動的に操作として型判別され，抽出される．

なお，システム操作記述の記法上，最上位階層のシステム操作以外は環境型の名詞句を記述してはならな

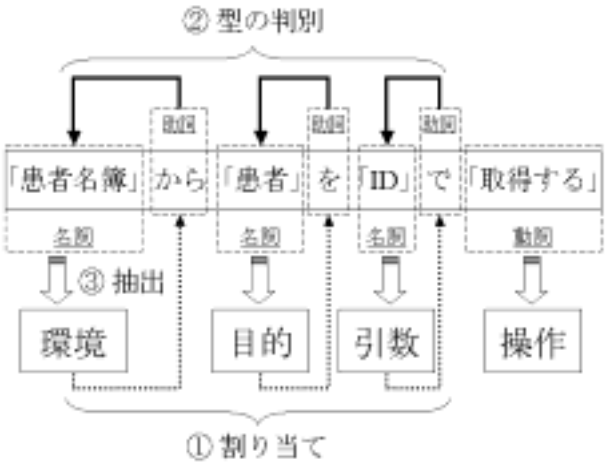

図 3: Toriaezer によるシステム操作記述の解析

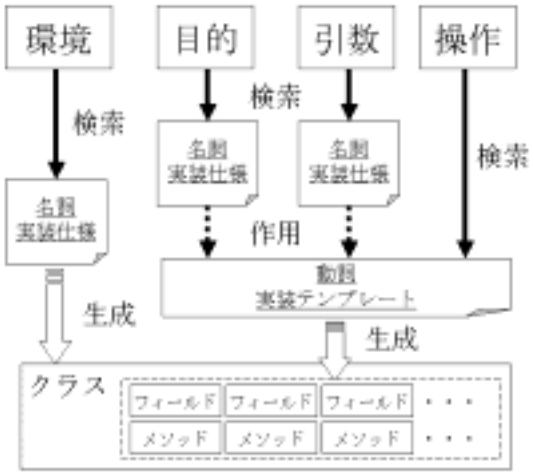

図 4: Toriaezer によるプログラムの生成

い．子階層のシステム操作は，親階層の目的型の名詞を自身の環境型として暗黙的に引き継ぐ，と先に述べた．Toriaezer が業務手順書を解析する際，この解決は自動的に行われる．すなわち，最上位階層でない全てのシステム操作について，親階層の目的型の名詞が，自身の環境型として抽出される．

3.3.2 プログラムの生成

Toriaezer は，システム操作を一行読み込む度に，前述の過程で抽出されたそれぞれの型に対応する名詞または動詞について，その実装定義を得る．このとき，名詞についてはその実装定義を名詞実装定義から，動詞については，所定のパスからその動詞に対応する動詞実装テンプレートを得る．Toriaezer はこうして得た実装定義を，その名詞または動詞の型によって使い分け，プログラムを生成する．

図 4 はその様子を表している．Toriaezer は，一行のシステム操作の実装を生成する際，それぞれの型の名詞または動詞についてその実装定義を検索する．そして環境型の名詞からは，その実装定義をもとにクラスを生成する．目的型および引数型の名詞については，動詞の実装定義であるテンプレートに，それらの実装定義を作用する．するとメソッドおよびフィールドが生成され，環境型の名詞によって生成されたクラスに対して実装される．

このようにして業務手順書における全てのシステム操作の処理が終了すると，システム本質部分のソースコードが得られる．

4 Toriaezer の利用

筆者がその開発に携わっている A 病院の医事会計システムにおける要求収集の過程において，Toriaezer を実際に利用した．

A 病院は約 100 床の病床を持つ中規模総合病院であり，医事会計業務を支援するシステムの開発・導入に取り組んでいる．筆者はこの中で，外来業務を支援するシステムの要求を収集するためにプロトタイプを利用した．そして，プロトタイプの作成には Toriaezer を活用した．

本章では，Toriaezer を利用して新患受付業務を支援するシステムのプロトタイプを作成したときの様子について述べる．

4.1 新患受付業務手順書の作成

業務手順書の作成は，ユーザと密に協調する形で行った．

業務手順の記述の際は，まずユーザに現状の業務手順を記述してもらい，それをもとにユーザと議論しながらシステム導入後の業務手順を記述した．

システム操作の記述については，システム導入後の業務手順におけるシステムと関係のある手順に対し，筆者がシステム操作を記述した．そしてそれをユーザに評価してもらうというアプローチをとった．

本節では，そのときの様子について述べる．

4.1.1 業務手順の記述

図 5 はユーザが作成した新患受付業務についての業務手順書である．

1. 受付係は患者の症状を聞き受診科を確認する.
2. 受付係は患者に保険証の提示を求める.
3. 受付係は患者に診療申込書の作成を求める.
4. 受付係は保険証類の有効期限を確認する.
5. 受付係は患者に問診表の作成を求める.
6. 受付係は患者のカルテを作成する.
7. 受付係は患者の診察券を作成する.
8. 受付係は待合札大・小を作成する.
9. 受付係は診察券をカルテにはさむ.
10. 受付係は伝達事項をカルテに添付する.
11. 受付係はカルテを診察室の看護婦に届ける.
13. 受付係は待合札大を待合掲示板に掲げる.
14. 受付係は保険証類を患者に返却する.
15. 受付係は待合札小を患者に渡す.

図 5: システム導入前の「新患受付」業務手順

1. 受付係は患者の症状を聞き受診科を確認する.
2. 受付係は患者に保険証の提示を求める.
3. 受付係は患者に診療申込書の作成を求める.
4. 受付係は保険証類の有効期限を確認する.
5. 受付係は患者に問診表の作成を求める.
6. 受付係はシステム上で診察受付を行う.
7. 受付係はこの患者の診察券を作成する.
8. 受付係は問診表の内容をシステムに登録する.
9. 受付係は保険証類を患者に返却する.

図 6: システム導入後の「新患受付」業務手順

図 6 は，図 5 をベースにユーザとともに作成したシステム導入後の業務手順書である．ここでは，システム導入前の業務手順書（図 5）におけるカルテに関係する手順 6，9，10，11 が削除されている．新システムでは既存のカルテを全て電子化することを前提としているためである．同様に，診察室における待合管理についても，診察室前の掲示板に患者 ID の書かれた待合札を掲げる方式から，電子掲示板に切り替えることを前提としているため，図 5 における待合札に関係する手順 8，13，15 は削除された．

一方，図 6 における手順 6，8 はシステム導入後の業務手順として新たに追加されている．これはカルテの電子化に伴い、これまでカルテに記載あるいは添付されていた情報をシステムに対して入力する必要があるからである．

システム導入後の業務手順書の作成は，ユーザと議論しながら行った．このとき，ユーザはシステム導入前の業務手順を記述することで現状の業務手順についてよく考えた後であったため，現状業務における問題点をきちんと整理できており，議論はスムーズに進んだ．

1. 受付係は患者の症状を聞き受診科を確認する.
2. 受付係は患者に保険証の提示を求める.
3. 受付係は患者に診療申込書の作成を求める.
4. 受付係は保険証類の有効期限を確認する.
5. 受付係は患者に問診表の作成を求める.
6. 受付係はシステム上で診察受付を行う.
   「患者名簿」に「患者」を「追加する」
      「ID」を「設定する」
      「氏名」を「設定する」
      「生年月日」を「設定する」
      「性別」を「設定する」
   ～ 省略 ～
7. 受付係は問診表の内容をシステムに登録する.
   「患者名簿」から「患者」を「ID」で「取得する」
      「問診表」を「追加する」
         「作成日」を「設定する」
         「様態」を「設定する」
         「発熱開始日」を「設定する」
         「頭痛の有無」を「設定する」
   ～ 省略 ～
8. 受付係はこの患者の診察券を作成する.

図 7: システム導入後の「新患受付」業務手順にシステム操作を記述したもの

4.1.2 システム操作の記述

次にシステム導入後の業務手順書において，システム操作の記述を加える作業を行った．図 6 において，システムを扱う手順は 6，8 であるため，それに対してシステム操作を記述した．この作業は筆者が行い，後にユーザに評価してもらった．

図 6 における手順 6 では，診察受付を行うために，受付対象患者の情報をシステムに対して入力する必要がある．それにはまず患者名簿に新しい患者を追加し，追加された患者に対して氏名，生年月日などといった情報を入力していく．

同じく手順 8 では，これまでは問診表はカルテに添付していたが，カルテは電子化されるため，代わりにシステムに入力しなければならない．これにはまず患者名簿から，問診表入力対象となる患者を取得する必要がある．このときの取得の手がかりとして患者の ID を使用する．次に，取得した患者に対して問診表を追加する．問診表は，患者来院時に毎回作成されるため，一人に対して複数の問診表情報を登録できなくてはならないからである．最後に，患者に対して追加された問診表に作成日，様態などの情報を入力する．

このようにしてシステム操作を記述し、最終的に図 7 に示す業務手順書が得られた．

システム操作記述についてユーザの評価を受ける際，ユーザの関心は最初，入出力方式の方に向いてしまっていた．しかし，システム操作記述とはシステムに対する意味的な操作であり，入出力方式を無視して考え

```
ID, , int, id, true
患者, , Patient, patient, true
患者名簿, , PatientList, patientList, true
氏名, , String, name, false
生年月日, , Date, dateOfBirth, false
性別, , int, gender, false
保険者番号, , String, assuranceNumber, true
記号・番号, , String, entitlementNumber, true
～ 省略 ～
```

図 8: 名詞実装定義ファイル

るようユーザに説明したところ，ユーザはこれを理解し，適切な評価を行ってくれた．すなわち，扱うべき情報の不備や，必要な操作に関する指摘をいただいた．

4.2 使用した名詞・動詞の実装定義

次に，システム操作記述において使用した名詞と動詞について，その実装方法を定義する．Toriaezer は，業務手順書中のシステム操作記述で利用されている名詞および動詞を洗い出す機能を備えているため，実装定義の際にはこれを利用すると便利である．

名詞の定義に関しては，図 8 に示したような名詞実装定義ファイルを作成し使用した．

動詞に関しては，図 6 のシステム操作記述中で使用した動詞が「設定する」,「取得する」,「追加する」であったため，これらについて動詞実装テンプレートを，Toriaezer のライブラリが提供する Java インターフェースに準じて実装を行った．

4.3 システム本質部分のプログラム生成

これまで作成してきた業務手順書，名詞実装定義ファイルおよび動詞実装テンプレートをもとに，Toriaezer を利用してプログラムを生成した．図 9 は生成されたプログラムの一部である．

4.4 Main プログラムの作成

Main プログラムとは，システム本質部分のプログラムを利用し，ユーザに対して実際にそのサービスを提供する部分のプログラムである．作成者はプロトタイプを完成させるために，Toriaezer が生成したプログラムを用いて Main プログラムの設計・実装を行わなくてはならない．

Main プログラムの設計には，Toriaezer が生成したプログラムから既存の CASE ツールを利用してリバースエンジニアリングを行い，生成された図 10 に示すクラス図を利用した．

```
import java.io.Serializable;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;

public class Patient implements Serializable {

      private int id = 0;
      private String name = null;
      private List diagnosticInquirys = new ArrayList();
     ～ 省略 ～

      public String getName() {
          return name;
      }
      public int getId() {
          return id;
      }
      public List getDiagnosticInquirys() {
          return new ArrayList(diagnosticInquirys);
      }
     ～ 省略 ～
}
```

図 9: 生成されたソースコード（Patient.java の一部）

この作業は素早く行うことができた．Toriaezer によって生成されたシステム本質部分のプログラムが要求に忠実であるという前提に立って Main プログラムを作成できるからである．また、業務手順書により業務の流れが定義されており，それに添った形で Main プログラムを作成すればよく，設計および実装作業はほぼ機械的であった．

5 Toriaezer の評価

本章では，A 病院の事例に基づき，3 章において掲げた 2 点についてその達成度合いを評価する．

5.1 得られたシステム本質部分について

Toriaezer を利用すると，業務手順書から直接システム本質部分のプログラムが得られる．従って，このプログラムが要求に忠実であるかどうかは，業務手順書に対してユーザが納得しているか否かによることになる．

業務手順書を軸としたプロトタイプ作成のための要求収集作業では，ユーザとの密な協調が可能であるということが A 病院の事例によって明らかになった．

特に，最初にユーザにシステム導入前の業務手順を記述させることで，業務手順についてよく考えたという前提を作ったことは効果的であった．これを行うことによって，ユーザの中で現状の業務手順が整理され，問題点等を良く認識できるようになったようである．結果，ユーザとともにシステム導入後の業務手順を作

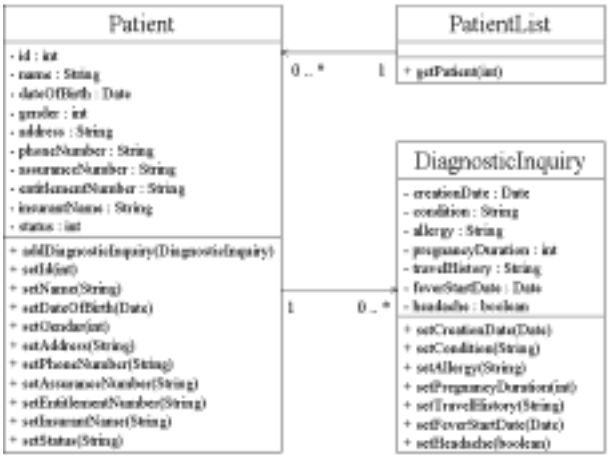

図 10: リバースエンジニアリングにより得られたクラス図

成する際に，円滑な議論を行うことができた．

システム操作記述に関しては，自分で記述することはできないにしても，評価は十分可能であったようである．記法は直感的であり，階層構造の意味も理解できているようであった．そのため，ユーザにとって直感的でないと思われる表現や，抜け落ちている操作等に関する的確な指摘を受けることができた．

このように，業務手順書の作成にはユーザの深いコミットメントが期待できるため，ユーザ要求を忠実な反映した業務手順書が得られる可能性が高い．従って，Toriaezer が生成するシステム本質部分のプログラムは，必然的にユーザ要求に忠実なものとなる．

5.2 プロトタイプ作成の効率について

Toriaezer は，業務手順書から実装を半自動的に生成する．すなわち，業務手順書の中で，同じ名詞や動詞は複数回用いられることがあるが，個々の名詞・動詞について一度だけその実装方法を定義しておけば，Toriaezer はそれを業務手順書中の全ての重複した名詞あるいは動詞に適用する．更に，この名詞および動詞の実装定義は再利用が可能であり，一度作成した実装定義を蓄積しておけば，その後のプロトタイプ作成時に実装定義を作成する手間がなくなる．

また，従来手法において，システム本質部分の設計（クラス図作成）は最もコストのかかる工程であった．要求から直接的に設計を導き出す術がないので，クラスの抽出および構成は言わば発見的であり，試行錯誤する必要があったためである．

Toriaezer は，業務手順書から直接プログラムを生成するので，明示的な設計作業を行う必要がない．すなわち、システム本質部分に関しては従来手法における要求定義がそのまま設計になるため，その分の時間が節約できる．

このように Toriaezer は，システム本質部分の実装を半自動化し，明示的な設計の工程を不要にすることによりプロトタイプ作成の効率化に貢献する．

6 おわりに

本稿では，日本語要求記述に基づくプロトタイプ作成支援ツール Toriaezer について述べてきた．

Toriaezer の入力となる要求記述は，システム導入後の業務手順，およびシステムに対する操作から成る業務手順書である．要求に忠実なシステム本質部分のプログラムを得るために，作成者は業務手順書作成の過程において，ユーザと密な協調のもとに作業を行う必要がある．ユーザが満足する業務手順書を得ることができれば，あとは作成者の介在なしに Toriaezer が直接プログラムを生成するからである．日本語の簡単な記法に基づき記述可能な業務手順書は，ユーザとの協調を支援する．

Toriaezer によるプログラムの生成は半自動的である．すなわち，作成者は業務手順書中のシステム操作記述において使用した名詞および動詞について，その実装を定義しておけば良い．しかもその定義は再利用が容易であるため，以降のプロトタイプ作成時に便利に利用できる．

このように，Toriaezer を利用すると，要求に忠実なプロトタイプを素早く作成することが可能になる．

今後の課題として，プロトタイプのみならず本番システム開発で利用できるような出力を生成できるようにすることを検討している．近年，O/R マッピングツールとして Torque[4] というフレームワークが利用されている．Toriaezer がそのスキーマファイルを生成出来るようになれば，業務手順書からデータベース，それに対するラッパクラス群が生成できるようになり，本番システム開発において有用性が期待できる．

このように，プロトタイプ作成のみならず，本番システム開発において有用なツールとなるよう，Toriaezer を進化させていこうと考えている．

参考文献

[1] R.Vonk 著、黒田純一郎訳『プロトタイピング CASE ツールの有効利用』(共立出版株式会社、1992 年)

[2] 有澤誠著『ソフトウェアプロトタイピング』(近代科学者、1986 年)

[3] ペルディタ・スティーブンス、ロブ・プーリー著、児玉公信監訳『オブジェクト指向とコンポーネントによるソフトウェア工学 ― UML を使って―』(ピアソン・エデュケーション、2002 年)

[4] Apache Torque http://db.apache.org/torque/