日本ソフトウェア科学会第 27 回大会 (2010 年度) 講演論文集

日本語一貫プログラミングの実践〜プロデルを用いて

馬場 祐人　筧 捷彦

現在の日本におけるソフトウェア開発では母国語である日本語で仕様書を作成し，Java に代表されるような欧米発想のプログラミング言語を使って実装している。本来日本語と欧米言語とは表現や意味にギャップがあり，そのことで仕様と実装に違いが生じている。我々は日本語で一貫してソフトウェア開発が行えることが理想であると考えている。本研究で我々は仕様書から実装まで日本語で一貫したプログラム作成を実践した。比較的小規模なソフトウェア開発を例に，設計段階では UML で日本語を用いてモデルを作成し，実装の段階では日本語プログラミング言語プロデルを用いてソフトウェアを開発した。本稿ではソフトウェア開発プロセスをすべて日本語で行うことによる利点や浮かび上がった問題点について報告する。

1　はじめに

現在，日本におけるソフトウェア開発では母国語である日本語で仕様書を作成し，Java に代表されるような欧米発想のプログラミング言語を使って実装することが行われている。ソフトウェア開発において，要求分析段階では，仕様書に日本語で機能名やデータ構造を表記する。詳細設計や実装段階では，識別子 (クラス名，関数名，変数名) を仕様書の日本語表記から英単語表記へ置き換える。この置き換えを行うことで仕様書に書かれた言葉 (日本語) と，実装コードの識別子 (英単語) との間にギャップが起こる。実装段階においても仕様書にある表現をそのままプログラムを実装することができれば，このようなギャップを解消することができると考えられる (図 1)。

我々は日本語で一貫してソフトウェア開発を行うことができる環境が求められると考えている。その一つの方法として，日本語プログラミング言語を積極的に活用することが考えられる。日本語プログラミング言語は，変数や関数を日本語で表記し，また日本語の語順に近い文法でプログラムを書くプログラミング言語である。日本語でプログラムを書くことで，日本語で書かれた仕様書の表現を実装段階でそのまま用いることができ，仕様書の表現を直にプログラムで表現できる。本研究の目的は，日本語で仕様書を作成し，日本語プログラミング言語で実装することで，日本語で一貫したソフトウェア開発を実践し，その利点や問題点を明確にすることである。

筆者らは，日本語プログラミング言語 “プロデル [1]” を開発している。プロデルを用いて，仕様書から実装まで日本語で一貫したソフトウェア開発を実践している。その一例として比較的小規模なソフトウェア開発を例に，設計段階では UML で日本語を用いてモデルを作成し，実装の段階では日本語プログラミング言語プロデルを用いてソフトウェアを開発した。本稿では，ソフトウェア開発プロセスをすべて日本語で行うことによる利点や浮かび上がった問題点について報告する。

A Practice of Japanese Programming

BAMBA Yuto, 早稲田大学 基幹理工学研究科, Graduate School of Fundamental Science and Engineering,Waseda University.

KAKEHI Katsuhiko, 早稲田大学 基幹理工学部 情報理工学科, Faculty of Science and Engineering, Waseda University.

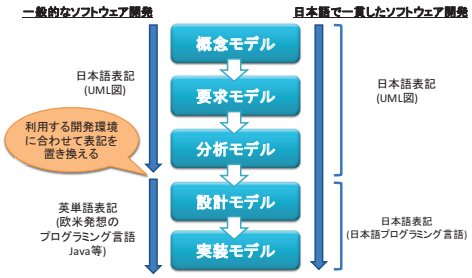

図 1　日本語一貫のソフトウェア開発

2　日本語一貫プログラミング

2.1　期待される点

日本語で一貫したソフトウェア開発を行うことで期待される点について述べる。利点として考えられることは，ソフトウェア設計者とプログラム作成者との意思疎通がよくなるという点がある。プログラム作成者は，日本語で書かれた設計書の言葉を直にプログラムで表現できる。ソフトウェア設計者も，プログラムからその構造やロジックを直感的に理解できる。また，日本語を使ってプログラム書くことによる効果についての研究 [2] では,『仕様書とプログラムの対応の良さはプログラムの可読性や保守に寄与』すると結論づけられている。このような点から概念モデルから設計モデルまで日本語で一貫して作成し，日本語プログラミング言語を使って実装することは有意義であると考えられる。

Java や C#のように最近のプログラミング言語の処理系は日本語の識別子を使うことができる。しかしながら予約語や API 群には英単語が使われていることから，ソフトウェア開発者が実装するクラスやメソッド，変数だけを日本語で名付けても，英単語と日本語が混在したプログラムとなり可読性に欠ける。日本語プログラミング言語は，プログラムを日本語らしく書けることを目標としており，標準ライブラリも日本語で用意されている。そのため表記が日本語で統一され可読性も向上すると考えられる。

2.2　日本語プログラミング言語

日本語で一貫してソフトウェア開発を実践するにあたり筆者らが開発している日本語プログラミング言語 “プロデル” を使った。プログラムを日本語で書くことができる処理系は，以前から存在する。実用的なソフトウェア開発を目的とした日本語プログラミング言語の処理系として “まほろば [3]”，“Mind [4]” が挙げられる。これらはプログラムを自然な日本語で書けることを目指している。まほろばでは，自身の処理系を使ってセルフコンパイラが作成されている。Mind は，市販ソフトウェアの開発に利用された事例が報告されている。また，“言霊 [5]” は，JavaVM で動作する日本語プログラミング言語であり，文系大学生のプログラミング教育に使われている。ほかに日本語表記の言語仕様を採用している教育用プログラミング言語として “ドリトル [6]” や “PEN [7]” が挙げられる。ドリトルはプロトタイプ型のオブジェクト指向プログラミング言語である。さらにフリーソフトとして “なでしこ [8]” が広く公開されており，豊富なライブラリを持ち，事務処理等，実用的なソフトウェア開発に利用されている。

このようにプログラムを日本語で書く処理系は，多く発表されており，プログラミング入門や単純なバッチ処理を行うことを目的としている。しかしながら，日本語一貫のソフトウェア開発を実践することができる，現在のソフトウェア開発で主流のオブジェクト指向プログラミングを行え，かつ，実用的なソフトウェア開発を行えるだけのライブラリが整備された日本語プログラミング言語は見当たらない。

プロデルは，Java や C# (.NET Framework) と同程度のオブジェクト指向プログラミングの言語機能および，ライブラリを整備することを目指している。またプロデルを実際に使い，ソフトウェア開発を実践していく過程で，言語仕様やライブラリ設計の改良を行っている。

プログラミングの習熟者にとっては，日本語表記のプログラムに対する魅力が少ないという意見がある。その理由の一つに日本語入力は英単語に比べてタイピング数が多くなることが挙げられる。その点については，開発環境の入力補完を用意するなどして，日本

語プログラミング言語向けの開発環境を整備することで解消すると考えられる。

2.3 関連研究

仕様書からプログラムを作成するための研究は，古くから行われている。和田らは，モジュール仕様書からプログラムを自動生成するシステム [9] を開発した。これは，モジュール仕様書をもとに仕様記述言語によって仕様を書き，COBOL のプログラムを生成するものである。畠山らは，オブジェクト指向一貫記述言語 OOJ [10] を提案している。これは，日本語で一貫して理工系専門分野のドメインユーザ向けのプログラム開発を行うものである。オブジェクト指向によるソフトウェア開発において，分析，設計および実装の各段階のモデルを作成するためにそれぞれ記述言語を用意している。小林らは，日本語による要求記述から Java で書かれたプロトタイプを作成する支援ツール [11] を開発している。

このように日本語で書かれた仕様書からプログラムを作成する試みは，多く報告されている。これらの研究目的は，仕様書から最終的に C++や Java で書かれたプログラムを自動生成することである。またこれらは共通して，仕様書で使われる言葉と，出力先のプログラミング言語にあった英単語との対応表を辞書として持たせている。本研究の最終目標は，仕様書から最終的に生成されるプログラムも日本語プログラミング言語によって日本語で書かれることである。

3 プロデルの言語仕様

プロデルは，Java に似た一般的なオブジェクト指向プログラミング (以下 OOP と呼ぶ) 言語の性質を持つ日本語プログラミング言語である。本稿で比較，議論するにあたってプロデルの特徴と，Java との違いについて述べる。

3.1 プログラム例

プロデルの言語仕様について，プログラム例を挙げて説明する。表 1 は，プロデルにおけるメソッド呼出しのプログラム例と，それに相当するプログラムを Java で書いたものである。なお，以下プロデルのプログラムでは，メソッド名を**太字**で表し，キーワードを 下線 で表す。また Java でメソッドに相当するものを，プロデルでは “手順” と呼ぶ。

表 1 Java とプロデルで書かれたプログラム

```
//Java
public class Person {
  private static string name;

  public static void setName(string value) {
    this.name = value;
  }

  public static void greet() {
    System.out.println(“こんにちは”+name+”です”);
  }

  public static void main(String[] args) {
    Person.setName(“太郎”);
    Person.greet();
  }
}
```

```
//プロデル
人を「太郎」と名付ける
人が挨拶する

人とは
  -名前

  +【自分】を、【値】と、名付ける手順
    名前は、値
  終わり

  +【自分】が、挨拶する手順
    「こんにちは」&名前&「です」を表示する
  終わり
終わり
```

表 1 の Java のプログラムで示すように，一般的にメソッド呼出しは，メソッド名および実引数 (必要な場合) を書く。プロデルのメソッド呼出しも同様にメソッド名と実引数を書くが，Java と異なる点として，実引数と仮引数を結びつけるために “助詞” を書く点がある。実引数の直後に添えられた助詞によって，実引数と仮引数が対応づけられる。この対応付けによって，メソッド呼出しで，補語 (実引数+助詞) をメソッド定義で定義された順番に書かなくても，メソッドへ正しく引数を渡すことができる。また表 1 を見るように，プロデルの文は複雑な構造ではなく，補語と動詞で構成されている単純な文で書く。

なお，Java では主プログラムを main メソッドに

書くが，プロデルでは主プログラムをソースファイルの先頭に書く。またプロデルで仮引数や局所変数を定義する場合は，変数を【 】で囲むことで定義する。手順の宣言部において “【自分】” と書かれている部分がある。これは，メソッド呼出しにおいて，“【自分】” の直後にある助詞によって対応づけられる実引数が，呼出す手順のメッセージレシーバであることを示す。

3.2 共通点

プロデルは，クラスベースの OOP に対応した日本語プログラミングである。OOP の性質である継承，カプセル化および多態性を持つ。

3.3 相違点

プロデルは，スクリプト言語ライクな性質を持っている。Java とプロデルの違いについて述べる。

3.3.1 明示的な型宣言が不要

プロデルでは変数や仮引数に対して明示的に型を宣言する必要がない。

3.3.2 単一インスタンスの自動生成

プロデルではクラス名と同名のインスタンス変数が自動的に生成される。Java において唯一単一のインスタンスを作るために，Singleton パターンを利用する。一方，プロデルでは同名のインスタンス変数を自動生成する機能があるから，特にプログラムを書くことなく，唯一単一のインスタンスを作ることができる。

3.3.3 日本語らしい言語仕様

プロデルでは，違和感のない日本語で書くことができるように言語仕様が用意されている。例えば表 2 の Java プログラムで示すようなプログラムを考える。ここで示した setInfomation メソッドは 3 つの引数を持つ。次に同じように動作するプログラムをプロデルで作成する場合を考える。プロデルではメソッドの引数の直後に助詞を付けなければならない。この例のように 3 つ以上の引数を持つメソッドを定義する場合に，日本語として自然で適切な助詞を付けることが難しくなる場合がある。そこで，Java では 3 つの引数を渡したが，プロデルでは 1 つの配列にまとめて渡す。そして，受け取った引数を “みなす” 文によって局所変数に展開することで工夫している。このように，日本語として自然な文でメソッド呼出しを書くことができるように言語仕様を用意している。なお，みなす文は，Ruby における “多重代入” に相当する操作である。

表 2 Java プログラムとプロデルのみなす文

```
//Java
showInfomation(“H10D8100”, “山田太郎”, 2010);

public static void setInfomation(String id, String name, String date) {
   System.out.println(id + “(” + name + “):” + date);
}
```

```
//プロデル
{「H10D8100」,「山田太郎」, 2010}を学生情報として表示する

【情報】を、学生情報として、表示する手順
        【学籍番号】,【名前】,【入学年度】
        情報を、{学籍番号, 名前, 入学年度}と、みなす
        名前&「(」&学籍番号&「):」&入学年度を表示する
終わり
```

4 日本語一貫プログラミングの実践

4.1 航空便管理システム

本研究で我々は，仕様書から実装まで日本語で一貫したプログラム作成を実践した。本章では，航空便予約システムの開発を例に，仕様書作成から実装まで日本語で一貫してソフトウェア開発を行った過程について述べる。

航空便予約システムは，早稲田大学情報理工学科におけるソフトウェア工学の講義において教材として使われたものである。講義資料 [12] には，UML で書かれた仕様書と Java で書かれたソースコードが用意されている。概念設計から要求分析まで日本語で仕様書を作成し，実装環境 (Java) に合わせて，基本設計および詳細設計を英単語の組み合わせの名前に置き換え，実装する流れで資料が用意されている。

4.2 仕様

仕様書は UML2.0 に従って書かれた UML 図である。日本語で一貫して書くにあたり新たに基本設計

および詳細設計を，クラス図，シーケンス図および状態遷移図を使って日本語で作成した。図 2 および図 3 は，航空便予約システムにおける “座席検索” と “座席予約” のシーケンス図である。

4.3 実装

航空便予約システムの仕様書をもとにプロデルで動作するプログラムを実装した。また比較のため同じ要求仕様を使って航空便管理システムを Java で実装した。Java のソースコードの文字コードは Unicode であり，識別子に日本語を使うことができることから，識別子に日本語を用いてプログラムを実装した。3 章で述べたプログラミング言語仕様の違いによるものを除いてプログラムのロジックは同じである。

4.4 ケース A プロデルで実装

概念設計から詳細設計まで日本語で一貫して仕様書を作成し，プロデルで実装した。現段階で用意されているプロデルの構文，ライブラリを利用した。

4.5 ケース B Java 上で識別子は日本語で実装

概念設計から詳細設計まで日本語で一貫して仕様書を作成し，Java で実装した。なおデータ構造に関するクラスなど Java で提供される標準ライブラリは，日本語に置き換えずにそのまま利用した。

5 日本語による命名

5.1 英単語表記と日本語表記

英単語と日本語では，表記による書き分け方が異なる。英単語を組み合わせてプログラム中の名前とする場合は，大文字・小文字の使い分けでその表す対象を区別することが行われている。例えば，Java の仕様書では，クラス名に Pascal 表記を使い，インスタンス名やメソッド名に Camel 表記を使うように推奨している。Java API 群では，実際にこの推奨規則に則った命名が行われている。

日本語の名前を使う場合は，識別対象を書き分けるのに，大文字・小文字の書き分けが使えないから，これとは別の工夫をするほかはない。本章では，日本語の名前を使う場合に，命名対象をうまく区別するために採った書き分け方の工夫を紹介する。

5.2 クラス名とインスタンス名

自然言語では，クラスとインスタンスをともに同じ名前で呼ぶことが多い。一方，プログラムの上ではこれらを区別したいので，実質的に同じ名前となることがある。そのために，それらを書き分ける工夫が必要となる。日本語で書き分けるのに，すぐに思いつくのは，同じ名前を漢字で書くか，かなで書くかで区別する方法である。ところが，かな綴りの名前は，日本語プログラミング言語でのプログラムの中では，構文を表すための部分と区別がつきにくく読み取りにくい。名前としては，やはり，漢字列やカタカナ列を使うのが視認しやすい。

そこで，名前には漢字列やカタカナ列を原則として当てることにした，その上で，クラス名には簡潔なものを選び，インスタンス名には個を示すものをクラス名に前置してできる名前を用いることにした。

例：“ボタン” クラスのインスタンスの場合
“開始ボタン”，“キャンセルボタン”

5.3 配列の名前

配列の名前も，そこに並ぶデータの型（クラス）と密接な名前としたいことが多い。名前に英単語を使っている場合には，単数・複数の書き分けを準用する方法をよく用いられる。

例：“Airplane” クラスの配列 “airplains”
“Seat” クラスの配列 “seats”

日本語の名前を使う場合，これにならって “航空機” クラスの配列に “航空機たち” とか “航空機ら” とする方法が考えられる。ただ，こうした接尾詞をつけた名前は普段使わないだけになんとなく落ち着かない。一般に，こうした配列は，何らかの条件を満たすものを列挙するのに使うことが多いことから，“一覧” を後置する方法を優先して採用した。

例：“航空機” クラスの配列 “航空機一覧”
“座席” クラスの配列 “座席一覧”

5.4 実装時に導入されるものの名前

クラスやメソッド，フィールドなどは，設計の時点に導入される。これに対して，カウンタ，イテレータ，バッファなどは，詳細設計や実装の段階になって導入される。実装の段階では，メソッドの局所変数や，各種作業の一時変数も導入される。

与えられた全候補それぞれについて作業を行う反復処理に使われるイテレータ変数については，その候補となるクラス名の前に “対象” を添えた名前をつけることにした。

例：“航空機一覧” に対するイテレータの場合
“対象航空機”
“座席一覧” に対するイテレータの場合
“対象座席”

メソッドの仮引数と，その仮引数の値に更新するフィールドとは，関連付けがわかる名前を与えておきたい。しかし，わざわざ名前を考え出すのも面倒である。そこで，仮引数名を更新対象のフィールド名の省略形にすることが多い。英単語を用いた名前を使う場合に，例えば，次のように名付ける類いである。

例：フィールド名 “departingAirport” に対する
仮引数名 “deptAirport”

省略形を用いる方法は，日本語の名前にも適用可能であった。

例：フィールド名 “出発空港” に対する
仮引数名 “発港”
フィールド名 “到着日時” に対する
仮引数名 “着日時”

6 評価

6.1 ソースコードの比較

航空便予約システムを実装した 3 つのソースコード (ケース A，ケース B および講義資料にある Java で英単語を用いた実装) を比較する。表 3 は，ソースコードの行数，文字数および識別子数を示したものである。文字数には，スペース文字やタブ文字を含まない。識別子数とは，識別子として使用された単語の数を表し，名前が重複する場合は 1 つとして数えた。なお類似度については今後の課題の章で述べる。

表 3 を見るとおり，ソースコードの行数や識別子数に関して見ると，日本語で書いた場合と英単語で書いた場合とで大きな違いはない。プロデルで実装した場合に，行数や識別子数が少ないのは，3 章で述べたプロデルと Java の言語仕様との違いによるものである。また，文字数に関しては日本語で書いた方が少ない。この理由は，英単語を用いる場合よりも日本語の文字数が少なくて済むからである。

表 3 航空便予約システムのソースコードの比較

実装言語	行数	文字数	識別子数	類似度
プロデル	359	5,855	89	0.102407407407407
Java (日本語)	372	8,754	98	0.065242538752217
Java (英単語)	370	11,330	95	0.002123453174196

6.2 プロデルで日本語を使った場合

表 5 は，図 2 および図 3 をもとに航空便予約システムの座席検索および座席予約を行うプログラムをプロデルで書いたものの一部である。

プロデルでは，Java における ArrayList に相当する標準クラスが用意されている。プロデルの標準ライブラリはクラスやメソッドが日本語で名付けられている。そのことから表 5 に英単語を組み合わせた表現は，見当たらない。

6.3 Java で日本語を使った場合

表 6 は，図 2 および図 3 をもとに航空便予約システムの座席検索および座席予約を行うプログラムを Java で書いたものの一部である。

Java は Unicode に対応しており，クラスや変数の名前に日本語を用いることに問題はなかった。

今回 ArrayList クラスなどの Java にある標準クラスライブラリを，そのまま利用した。Java で日本語を用いた場合であっても，Java のキーワード，データ型，例外クラスおよび Java の標準クラスのメソッドや属性は，英単語表記のままである。単に，仕様書に書かれた日本語を実装に用いるだけでは，英単語表記と日本語表記が混在したソースコードとなる。例えば，表 6 の “検索結果文字列を取得する” メソッドの 4 行目では ArrayList クラスの add メソッドを呼出しているが，仕様書では “リストへ加える” といっ

た日本語表現で書かれている操作である。

6.4 Java で英単語を使った場合

さらに比較のため，一般的なソフトウェア開発として多い Java 上で英単語の組み合わせを用いて実装した場合について述べる。

仕様書にある日本語表記を実装する言語にあわせて英単語を用いると，仕様書の言葉と実装時の識別子との変換表を用意しなければならない。プログラムに改変を加える時や動作テストの時に，ソースコードからだけで仕様書との対応を探すことは難しい。そのことから，単語の対応を確認するために対応表を用意する必要がある。より大規模なソフトウェア開発を行う場合には，対応表が膨大になると考えられる。

表 4 は，航空便管理システムを Java で英単語を用いて実装する場合における，仕様書の日本語表記と，実装時の英単語表記との対応表の一部である。この例では 111 対の単語の対応があった。表 3 の識別子数よりも対応単語数が多いのは，英単語の省略表現で名付けている変数を，日本語名では省略せずに名付けるなどして，同じ名前でも定義する場所によって対応する日本語名を変更したからである。ケース A およびケース B では，仕様書の表現をそのまま用いることから，このような対応表は不要であった。

表 4 仕様書の日本語と実装の英単語の対応表 (一部)

日本語名	英単語名
管理処理	managementControl
検索座席リスト	foundSeatClassList
座席を検索する	findSeats
出発地	departingLocation
到着地	arrivingLocation
出発日	departingDate
航空便一覧	plane
対象航空便	allAirplanes
検索出発日時	queryDepDate
空席一覧を作成する	getResultSeatStrings
結果	foundSeatClassStringList
座席	seat
航空便	plane
航空便情報一覧	foundSeatClassStrings

日本語名	英単語名
座席を予約する	reserveSeat
座席番号	seatNumber
座席	seat
値段を取得する	getPrice
座席番号	seatNumber
座席	seat
航空便の空席を確認する	findMatchedVacancies
出発地	departingLocation
到着地	arrivingLocation
出発日	plane
検索出発日	queryDepDate
対象出発日	targetDepDate
座席一覧	seats
座席	seat

6.5 例外処理

表 5 の “**座席を、予約する**” 手順では，例外処理が書かれている。“例外監視” の直後から “発生した場合” の直前までの間に書かれたプログラムにおいて例外が発生した場合，“発生した場合” 直後から “監視終わり” 直前のプログラムが実行される。Java の try-catch 文に相当する。この例外処理は，ユーザから不正な入力が与えられた際に，プログラムが予期せず終了してしまうことを回避するものである。

このようなプログラムは，プログラムを仕様書として読むと，違和感がある。仕様書に書かれたソフトウェアそのものの動作の記述と，計算機やプログラミングの都合上，必要な記述とが，混在しているからである。仕様書からプログラムを一貫して書く仕組みを考えた場合，ユーザドメイン固有の流れと計算機上の処理を分けて書くなどの仕組みが必要である。

7 今後の課題

本研究を踏まえて，日本語一貫プログラミングを容易に実践することできるように，日本語プログラミング言語の開発環境を整備する必要があると考えている。辞書などの前持って用意した情報を使用せずに，UML 図から日本語プログラムを生成する仕組みを検討することが今後の課題である。

また評価では表 3 に類似度を挙げた。これは，仕様書にあるユースケース記述の文章と，ソースコードを比較した時の類似度を求めたものである。類似度は，String::Trigram [13] を使用して文字単位の類似度 (tri-gram) で求めた。類似性が高ければ日本語プログラムが仕様書の文章をそのままプログラムとして書けることが示せると考えている。しかし，ユースケース記述とソースコードとの文書の類似度については，いくつかの指標があり，より明確な方法で行うべきであると考えている。評価方法についても課題がある。

8 まとめ

本研究では仕様書から実装まで日本語で一貫したソフトウェア開発を実践した。本稿では，日本語で一貫したソフトウェア開発を行う利点と問題点を報告した。利点としては，仕様書の表現をそのままプログラムとして書くことで，仕様書と実装との名前の対応表が不要であることである。一方，仕様書にあるユーザ

ドメイン固有のソフトウェアのメインの動作と，プログラミングの都合上に必要な動作とを，別に書くための仕組みが求められる。

筆者らは，本例に限らず，実際に業務アプリケーションや Web アプリケーションなど実際に作られる，様々なソフトウェア開発において仕様書から実装まで日本語で一貫したソフトウェア開発を実践し，また同時に日本語らしい日本語プログラムを書くための日本語プログラミング言語の言語仕様やライブラリ設計を模索する必要があると考えている。

謝辞 早稲田大学情報理工学科 鷲崎弘宜先生には，研究題材を提供していただき，また本研究に関するご意見を頂きました。感謝いたします。

参 考 文 献

[1] ゆうと: 日本語プログラミング言語「プロデル」, http://rdr.utopiat.net/, (2010 年時点).
[2] 中川正樹ほか: 日本語プログラミングの実践とその効果, 情報処理学会論文誌 Vol.35,No.10(1994)
[3] 今城哲二ほか: 日本語プログラム言語 “まほろば” の言語仕様, 情報処理学会研究報告 (SE)(2001).
[4] 木村明, 片桐明: 日本語プログラミング言語 Mind について, 情報処理学会第 16 回プログラミング言語研究会, 1988, pp. 25–32.
[5] 岡田健，中鉢欣秀ほか: “ プログラミング言語としての日本語 ”，第 44 回プログラミングシンポジウム報告集 (2003).
[6] 中谷多哉子, 兼宗進ほか: オブジェクトストーム：オブジェクト指向言語による初中等プログラミング教育の提案. 情報処理学会論文誌 Vol.43 No.6 (2002).
[7] 西田知博ほか: 初学者用プログラミング学習環境 PEN の実装と評価. 情報処理学会論文誌 Vol.48 No.8 (2007).
[8] クジラ飛行机: 日本語プログラム言語「なでしこ」, http://nadesi.com/, (2010 年時点).
[9] 和田考ほか: プログラム自動生成のための日本語仕様記述言語. 情報処理学会研究報告 プログラミング (PRO)(1988).
[10] 加藤木和夫, 畠山正行ほか: オブジェクト指向プログラム設計記述言語 OOPD とその記述環境. 情報処理学会論文誌 Vol.43 No.5 (2002).
[11] 小林孝弘ほか: 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. 情報処理学会研究報告 (IS)(2004).
[12] 鷲崎弘宜ほか: ソフトウェア工学講義資料, 早稲田大学情報理工学科ソフトウェア工学講義資料 (2008).
[13] Tarek Ahmed: String::Trigram. CPAN, http://search.cpan.org/dist/String-Trigram/Trigram.pm(2010 年時点).

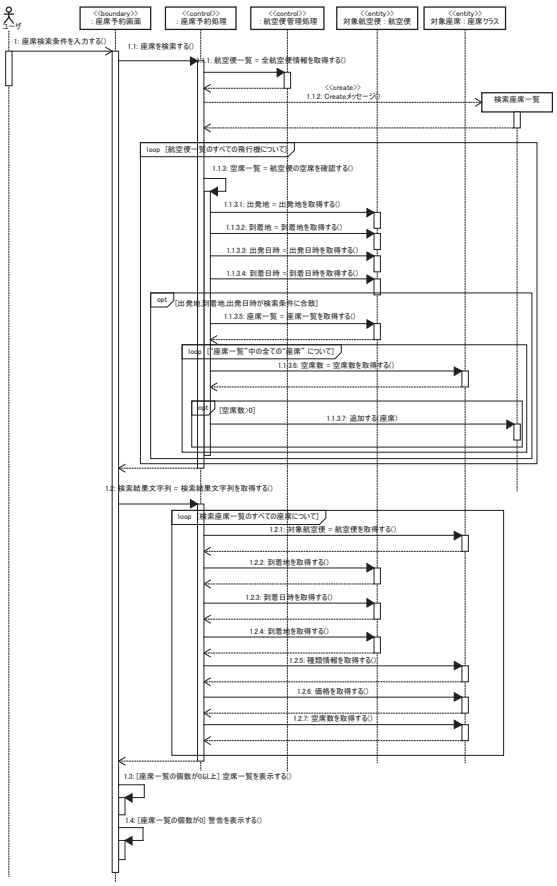

図 2　座席検索シーケンス図

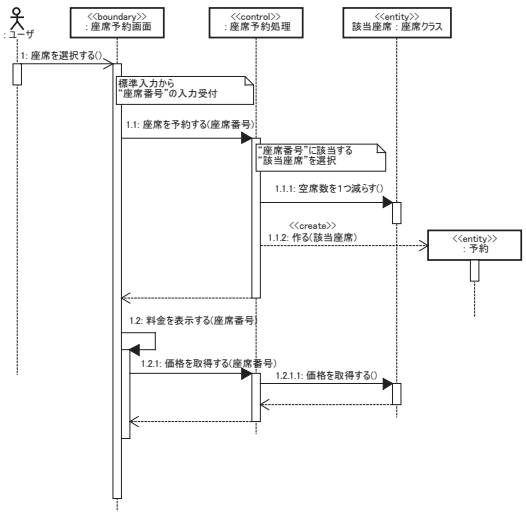

図 3　座席予約シーケンス図

表 5　航空便予約システムプロデルソースコード一部

```
座席予約処理とは
    -検索座席一覧:配列

    +【自分】から、【検索条件】で、座席を、検索する手順
        【出発地】、【到着地】、【出発日】
        検索条件を{出発地, 到着地, 出発日}と、みなす

        【航空便一覧】は、航空便管理処理の全航空便
        検索座席一覧は、{}
        航空便一覧のすべての対象航空便について、それぞれ繰り返す
            例外監視
                検索条件で、対象航空便から空席を確認する
            監視終わり
        繰り返し終わり
    終わり

    +【自分】から、検索結果文字列を、取得する手順
        【航空便情報一覧】は、{}
        検索座席一覧のすべての座席について、それぞれ繰り返す
            【対象航空便】は、座席の航空便
            航空便情報は、「[対象航空便の出発空港の地名],」&
                    「[対象航空便の到着空港の地名],」&
                        「[対象航空便の出発日時],」&
                            「[対象航空便の到着日時],」&
                            「[座席の種類情報],」&
                            「[座席の値段],[座席の空席数]

            航空便情報一覧へ航空便情報を加える
        繰り返し終わり
        航空便情報一覧を返す
    終わり

    +【自分】で、【座席番号】へ、座席を、予約する手順
        【該当座席】
        例外監視
            該当座席は、検索座席一覧(座席番号)
            該当座席から空席数を減らす
            予約(該当座席)を作る
            (座席番号から値段を取得したもの)を返す
        発生した場合
            0を返す
        監視終わり
    終わり

    +【自分】から、【座席番号】で、値段を、取得する手順
        【該当座席】
        例外監視
            該当座席は、検索座席一覧(座席番号)
            (該当座席の値段)を返す
        発生した場合
            0を返す
        監視終わり
    終わり

    -【検索条件】で、【対象航空便】から、空席を、確認する手順
        【出発地】、【到着地】、【出発日時】
        検索条件を{出発地, 到着地, 出発日時}と、みなす
        【検索出発日時】は、出発日時を日時形式化したもの
        【対象出発日時】は、対象航空便の出発日時

        もし(出発地が対象航空便の出発空港の地名)かつ
                到着地が対象航空便の到着空港の地名かつ
                検索出発日時の年が対象出発日時の年かつ
                検索出発日時の月が対象出発日時の月かつ
                検索出発日時の日付が対象出発日時の日付なら

            【座席一覧】は、対象航空便の座席一覧
            座席一覧のすべての座席について、それぞれ繰り返す
                もし座席の空席数が0以上なら
                    検索座席一覧へ座席を加える
                もし終わり
            繰り返し終わり
        もし終わり
    終わり
終わり
```

表 6　航空便予約システム Java ソースコード一部

```
public class 座席予約処理 {

  private 航空便管理処理 管理処理 =
    航空便管理処理.航空便管理処理を取得する();
  private ArrayList<座席クラス> 検索座席種類リスト;

  public void 座席を検索する(String 出発地,
      String 到着地, String 出発日時) {
    航空便[] 航空便一覧 = 管理処理.全航空便を取得する();
    検索座席種類リスト = new ArrayList<座席クラス>();
    for (航空便 航空便 : 航空便一覧) {
      Date 検索出発日時 = null;
      try {
        検索出発日時 = new Date(出発日時);
      } catch(IllegalArgumentException iae){
        return;
      }
      空席を確認する(出発地, 到着地, 航空便, 検索出発日時);
    }
  }

  public String[] 検索結果文字列を取得する() {
    ArrayList<String> 航空便情報リスト = new ArrayList<String>();
    for (座席クラス 座席 : 検索座席種類リスト) {
      航空便 航空便 = 座席.航空便を取得する();
      航空便情報リスト.add(
          航空便.出発地を取得する().地名を取得する()
          + "," + 航空便.到着地を取得する().地名を取得する()
          + "," + 航空便.出発日時を取得する()
          + "," + 航空便.到着日時を取得する()
          + "," + 座席.種類情報を取得する()
          + "," + 座席.価格を取得する()
          + "," + 座席.空席数を取得する());
    }
    String[] 航空便情報一覧 = new String[航空便情報リスト.size()];
    return (String[]) 航空便情報リスト.toArray(航空便情報一覧);
  }

  public void 座席を予約する(int 座席番号) {
    座席クラス 座席;
    try {
      座席 = (座席クラス) 検索座席種類リスト.get(座席番号 - 1);
      座席.空席数を1つ減らす();
      new 予約(座席);
    } catch(IndexOutOfBoundsException ioobe){
    }
  }

  public int 値段を取得する(int 座席番号) {
    座席クラス 座席;
    try {
      座席 = (座席クラス)検索座席種類リスト.get(座席番号 - 1);
      return 座席.値段を取得する();
    } catch(IndexOutOfBoundsException ioobe){
      return 0;
    }
  }

  private void 空席を確認する(String 出発地, String 到着地,
      航空便 対象航空便, Date 検索出発日時) {
    Date 対象出発日 = 対象航空便.出発日時を取得する();
    if(出発地.equals(対象航空便.出発地を取得する().地名を取得する())
        && 到着地.equals(対象航空便.到着地を取得する().地名を取得する())
        && 検索出発日時.getYear() == 対象出発日.getYear()
        && 検索出発日時.getMonth() == 対象出発日.getMonth()
        && 検索出発日時.getDate() == 対象出発日.getDate()
    ){
      座席クラス[] 座席一覧 = 対象航空便.座席一覧を取得する();
      for (座席クラス 座席 : 座席一覧) {
        if(座席.空席数を取得する() > 0) {
          検索座席種類リスト.add(座席);
        }
      }
    }
  }

}
```