社団法人 情報処理学会　研究報告I
PSJ SIG Technical Report

2004－SE－144　(4)
2004／3／18

アクティビティ図主導のモデリング支援ツールの開発

野村幸司　　小飼 敬　　上田賀一

茨城大学 工学部 情報工学科

〒 316-8511 茨城県日立市中成沢町 4-12-1

オブジェクト指向ソフトウェア開発において, 近年, UML が普及している. UML に対応するモデリング支援ツールでは, UML モデルの正当性や整合性のチェックを行う必要がある. 一方, プログラム生成の観点で各種ダイアグラムを捉えてみると, クラスの構造情報を表すクラス図は必要不可欠である. また, クラスの操作情報の抽出には, 対象システムの相互作用を表すシーケンス図を利用できる. しかし, ユーザがシナリオに従って, 1 つの操作の実行順序をモデリングするにはアクティビティ図の方が適していると言える. そこで, 本研究ではアクティビティ図を中心に利用した開発に着目し, 作成されたクラス図及びアクティビティ図の正当性や整合性をチェックし, 基本的なプログラムコードの自動生成を行うモデリング支援ツールを開発した.

A Modeling Support Tool Under Activity Diagram Leading

Koji NOMURA, Kei　　I, and Yoshikazu UEDA

Ibaraki University

4-12-1 Nakanarusawa, Hitachi, Ibaraki, 316-8511, JAPAN

In object-oriented software development, UML spreads recently. The correctness and the consistency of UML models must be checked by a UML modeling tool. On the other hand, when thinking about various diagrams with a viewpoint of the program generation, the class diagram which shows the structure information of the class is indispensable. In addition, an activity diagram is more suitable than a sequence diagram when a user models the executive order of operation in accordance with the scenario. Then, in this research, we developed the modeling support tool that checks the correctness and the consistency of models and generates program code of the system from class diagram and activity diagram models.

1　はじめに

オブジェクト指向によるソフトウェア開発において, 近年, その問題分析や仕様記述のための UML (Unified Modeling Language)[1] が標準のモデル表記法として広く普及している. これにともない, UML に対応した多くのモデリングツールや開発支援環境 (以下, UML ツール) が開発されており, その利用も広まってきている [2],[3],[4],[5].

UML ツールは描画ツールと異なり, ダイアグラムを単なる図形としてではなく, UML のモデルとして意味を解釈し, ダイアグラムの正当性を保たなければならない.

UML 記法に基づいてダイアグラムを作成するだけで, モデル間の整合性は必ず保証されるわけではなく, モデルを作成した開発者に委ねられることになる. 整合性の保証されない複数のモデル間をそのまま用いて開発を進めることは, 開発過程において重大なバグや手戻りを生じる原因となる.

また, UML ツールを用いた開発では, 作成したダイアグラムからプログラムの自動生成を行うことで, 単純なプログラミングミスの削減が期待できる.

本研究ではプログラム生成とモデルの正当性や整合性のチェックに着目した支援を考える. プログラム生成の観点で UML の各種ダイアグラムを捉えてみると, プログラムコードにはクラスの構造情報が必要であるため, クラス図は必要不可欠なものである. また, クラスの操作情報を抽出する必要もあり, 対象システムの相互作用を表すシーケンス図を利用できる. しかし, シーケンス図はクラス単位

でまとめられておりプログラム生成には向いていると言えるが, ユーザがシナリオに従って, 1 つの操作の実行順序をモデリングするにはアクティビティ図の方が適していると言える.

以上を踏まえた上で, 本研究では UML のダイアグラムの中でもアクティビティ図を中心に利用した開発に着目し, 開発過程において作成されたクラス図及びアクティビティ図の正当性や整合性のチェックを行い, ダイアグラムから対象システムの基本的な java プログラムコードの自動生成を行う UML ツールを開発した.

2 アクティビティ図

アクティビティ図は各処理状態の順序関係を明示するものである. そのため, アクティビティ図は操作の内部処理の手順を表すために用いられる.

本研究で用いるアクティビティ図の構成要素を表 1 に示す. アクティビティ図の構成要素としては, 他に同期バーやオブジェクトフローがあるが, 本研究では表 1 の基本的な要素だけを扱う. また, 分岐や合流を特に明記しないアクティビティ図の記述の仕方があるが, この場合, 分岐や合流に相当するものをダイアグラムから判断する必要がある. 本研究では分岐や合流を要素として用いるものとする.

表 1: アクティビティ図の要素

要素	意味
開始	制御の始まりを示す.
終了	制御の終了を示す.
アクティビティ	何らかの処理を行っている状態
分岐	条件による制御の流れの変更を行う.
合流	複数の制御を 1 つにする.
遷移	制御の方向を示す.
レーン	アクティビティの存在範囲を表す

アクティビティ図中のあるアクティビティの詳細を別のアクティビティ図によって表現することができる[1]. 言い換えればすべてのアクティビティ図は何らかのアクティビティを表していると考えられる.

また, シーケンス図が複数の処理間の相互作用を表すのに対して, アクティビティ図は 1 つの処理の内部作用を表す. しかし, アクティビティ図中のアクティビティを示すアクティビティ図が存在するとき, 処理間の相互作用として抽出することができる.

[1] アクティビティを表すアクティビティ図は, もとのアクティビティ図に対してサブアクティビティ図と呼ばれることがある.

3 正当性チェック

UML ツールはダイアグラムを単なる図形としてではなく UML のモデルとして扱い, UML の文法やモデルの構造上の矛盾をチェックする必要がある. 本研究ではこれらを正当性チェックとして作成されたダイアグラムのチェックを行う. アクティビティ図に対する正当性チェック項目を以下に示す.

3.1 状態要素と遷移の関係

「開始」,「終了」,「アクティビティ」,「分岐」,「合流」を状態要素と呼ぶものとする. 状態要素はそれぞれ決められた数の入遷移 (前の要素) と出遷移 (次の要素) を持つ. また,「遷移」要素は必ずスタートの状態要素 (前の要素) とゴールの状態要素 (次の要素) とを結ぶ. 次の表 2 に要素が持つ前次の要素の数を示す. アクティビティ図中のすべての要素がこれを満たさなければならない.

表 2: 要素の持つ前次要素数

要素	前の要素数	次の要素数
開始	-	1
終了	1	-
アクティビティ	1	1
分岐	1	2 以上
合流	2 以上	1
遷移	1	1

3.2 制御の流れ

- 開始位置の特定

制御の開始位置を調べる. 図中の「開始」要素の数が 1 つであれば制御の開始位置が特定できる.

- 分岐における制御

「分岐」要素後の制御は必ず 1 つの「遷移」に移る. どの「遷移」に移るかは「分岐」からの「ガード条件」による. つまりある「分岐」に「ガード条件」$g1, g2, ..., gn$ があるとき, gi と gj は互いに素 $(1 \leq i < j \leq n)$ である. よって制御分岐ではこれが成り立っているかをチェックしなければならないが,「ガード条件」は自然言語によって記述される

ため証明は困難である. 本ツールでは, ある「分岐」から発生する「ガード条件」を単なる文字列として等しいかといった簡易なチェックを実装している.

3.3 非実行要素

それぞれの要素に対して制御が移るかどうかを判断する必要がある. その例として図 1 に非実行要素を含むアクティビティ図を示す.

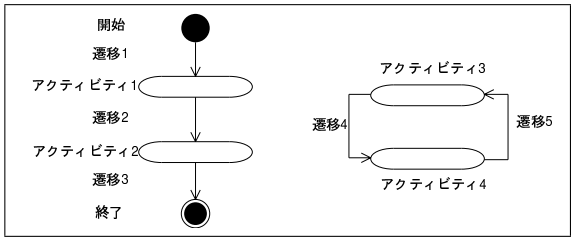

図 1: 非実行要素例のアクティビティ図

図 1 のすべての要素のうちで, アクティビティ3,4 及び遷移 4,5 は制御の移ることのない要素であると判断できる.

4 整合性チェック

UML では対象とするシステムを異なる視点からモデリングするために数種類のダイアグラムの記法を定めている. しかし, UML 記法に基づいただけではモデル間の整合性は開発者に委ねられているため, 必ず保たれるとは言えない. モデル間に整合性のない複数のモデルによって開発を進めると, 開発過程において重大なバグや手戻りを生じる原因となる. これらの障害を未然に防ぐには, 対象システムをモデリングしたすべてのダイアグラムが互いに整合性を保っていることをチェックする必要がある.

本研究ではアクティビティ図を中心にクラス図と併用しての開発を考えているため, クラス図とアクティビティ図との間で整合性を保つ必要がある.

また, 複数のダイアグラム間の整合性チェックを行う際には, 3 節で述べた正当性がすべてのダイアグラムでチェックされてなければならない.

以下にクラス図とアクティビティ図間の整合性チェック項目を示す.

• クラスの一致
アクティビティ図中のクラスがクラス図中から判断できるか.

• 操作の所属
アクティビティ a と操作 b が対応するとき, a の属するクラスと b を持つクラスが一致するか判断する.

• 操作の可視性
アクティビティ図において, あるクラス A のアクティビティで他クラス B のアクティビティ b を呼び出しているとき, クラス図において, クラス A からクラス B の b に対応する操作の呼び出しが可能であるかを判断する.

5 プログラムの自動生成

5.1 クラス図からのプログラム生成

クラス図からのプログラム生成機能は UML ツールの基本的な機能となっている. クラス図は対象システムの静的な側面を示しているためにクラス名, 属性名, 操作名といった 1 つのクラスに記述される情報に加え, クラス同士の関係からもプログラム記述のための情報を得ることができる. これらの情報はプログラムコードを作成する上で必要不可欠であり, またクラス図から容易に取得できる.

5.2 アクティビティ図からのプログラム生成

広く普及しているモデリングツールの中にはクラス図に加えてシーケンス図からの情報をプログラムに (操作として) 変換するものも存在する. しかし, シーケンス図はあくまで相互作用をモデリングするためダイアグラムであり, 1 つの操作を詳しくモデリングするためにはアクティビティ図を用いる方がより適している. アクティビティ図はある操作が複数の処理を行うことがわかっている場合にそれらの実行順序をモデル化することができる. この実行順序を作成したダイアグラムから読み取り, 構造を解釈することによってシナリオやプログラムの生成が可能である.

5.2.1 要素の制御順の導出

アクティビティ図により示される制御は「開始」要素から始まる. また「終了」,「アクティビティ」,「分岐」,「合流」の要素への制御の移り変わりは「遷移」の示す方向により明示されている. 制御は「合流」要素を用いることでループ制御を表すことができる.「分岐」要素後の「遷移」の分岐は図の作成段階であらかじめ決定するものとする. 次の図 2 にアクティビティ図の例を示す.

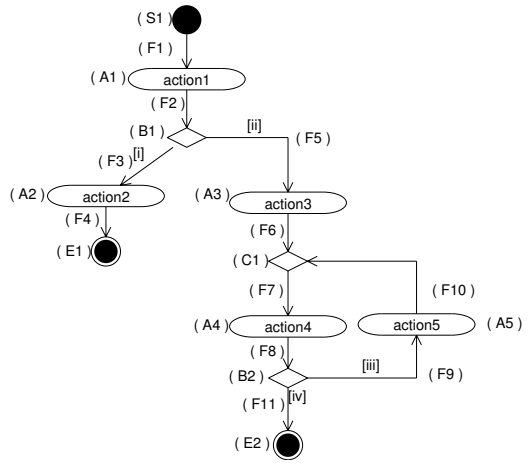

図 2: アクティビティ図の例

なお, 図中の記号の対応は次の通りであり, 数字は要素ごとの通し番号である.

S: 開始　　B: 分岐
E: 終了　　C: 合流
A: アクティビティ　F: 遷移

図 2 から得られる要素の制御順のリストは,

S_1,F_1,A_1,F_2,B_1,F_3,A_2, F_4,E_1,B_1,F_5,A_3,
F_6,C_1,F_7,A_4,F_8,B_2,F_9,A_5, F_{10},C_1,B_2,F_{11},E_2

となる.

5.2.2 アクティビティ図の解析

アクティビティ図の制御構造を 5.2.1 節で求めた要素の制御順序から導く. アクティビティ図として正当性の保たれている図であれば, 一意に実行順を得ることができるはずである. 次の表 3 に解析ノードを, 図 3 に解析の規則を示す.

*join*X は自身以下の階層の中で同一の合流 C を持つ *join with*Y がある時に反復処理を表している.

表 3: 解析ノード

ノード名	意味
begin	制御の開始
selection	選択処理
selection part	選択処理の一部
join	同一となる制御処理
action	1 つの動作
finish	制御処理の終了
join with	同一となる制御処理の繰り返し

```
<begin> ::= S <action>*
            <selection>|<join>|<finish>
<selection> ::= F <selection part>*
                                  (*:2 以上)
<selection part> ::= B <action>*
     <selection>|<join>|<join with>|<finish>
<join> ::= F C <action>*
     <selection>|<join>|<join with>|<finish>
<action> ::= F A
<finish> ::= F E
<join with> ::= F C
```

図 3: 解析の規則

5.2.3 シナリオの導出

アクティビティ図の示す処理手順をシナリオとして導出する. 5.2.2 節での構造の解釈ができれば, 各ノードが対応する文章を生成するだけで実現できる.

図 4 に図 2 のアクティビティ図から導出されるシナリオを示す.

1. action1
2. [i] なら (3.) へ, [ii] なら (5.) へ,
3. action2
4. 終了
5. action3
6. action4
7. [iii] なら (8.) へ, [iv] なら (10.) へ,
8. action5
9. (6.) 以降を繰り返す.
10. 終了

図 4: 図 2 から導出されるシナリオ

5.2.4 プログラムの導出

シナリオの導出と同様に, プログラムの導出を行う. この際, アクティビティは操作として扱う. 図 5 に図 2 のアクティビティ図から導出されるプログラムを示す.

```
//開始
action1( );
if ( true /* [i] */ ){
    action2( );
    // 終了
}else
if ( true /* [ii] */ ){
    action3( );
    do{
        action4( );
        if ( true /* [iii] */ ){
            action5( );
            // ループ先頭に戻る
        }else
        if ( true /* [iv] */ ){
            // 終了
        }
    }while(false);
}
```

図 5: 図 2 から導出されるプログラム

6 開発例

本 UML ツールの全体像を図 6 に示す.

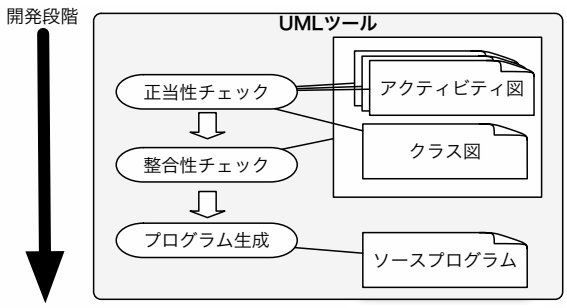

図 6: ツールの全体像

例を用いて本 UML ツールを用いた開発の流れに沿いながら実行の様子や結果を示す. ここでは文献 [8] のビデオレンタルシステムを例に取り上げる.

6.1 モデルの作成

6.1.1 ユースケース図・ユースケース記述

図 7 にビデオレンタルシステムのユースケース図を示す.

次に, 各ユースケースについてユースケース記述を作成する.

ユースケース名：会員を登録する
1. 受付係は, 会員情報を入力する.
2. システムは, 会員番号を発番する.

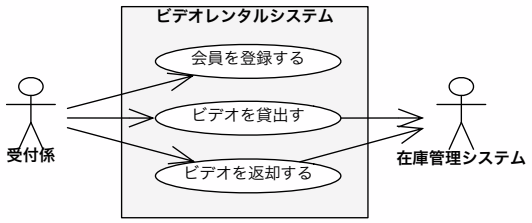

図 7: ユースケース図

ユースケース名：ビデオを貸出する
1. 受付係は, 会員番号を入力する.
2. システムは, 会員名を表示する.
3. 受付係は, ビデオ番号を入力する.
4. システムは, 映画番号などを表示する.
5. システムは, 在庫管理システムに出庫指示する.
6. 追加貸出ビデオがあれば 3〜5 を繰り返す.
7. 受付係は, 貸出し日数を入力する.
8. システムは, 貸出番号・レンタル料金を表示する.

ユースケース名：ビデオを返却する
1. 受付係は, 貸出番号を入力する.
2. システムは, 貸出内容を表示する.
3. 受付係は, 返却を登録する.
4. システムは, 在庫管理システムに, 入庫指示する.

6.1.2 アクティビティ図

各ユースケースについてユースケース記述より, アクティビティ図を作成する. 作成したダイアグラムを図 8, 9, 10 に示す. 図は本 UML ツールによって作成され, 各ダイアグラムについて 3 節で述べた正当性項目がチェック済みである.

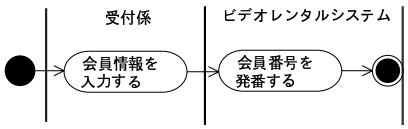

図 8: 会員を登録する

さらに必要であれば, あるアクティビティを表すアクティビティ図を作成する. ここでは例として図 11 に図 9 中のアクティビティ「在庫管理システムに出庫指示する」を表すアクティビティ図を示す.

また, 本 UML ツールでのアクティビティ図の作成画面を図 12 に示す.

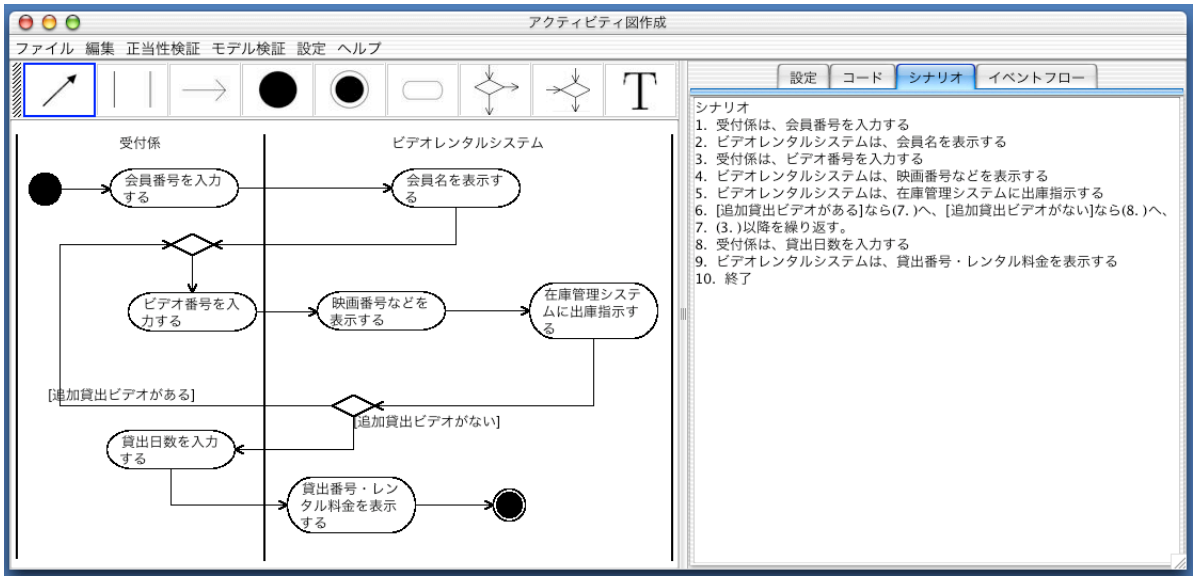

図 12: アクティビティ図作成画面

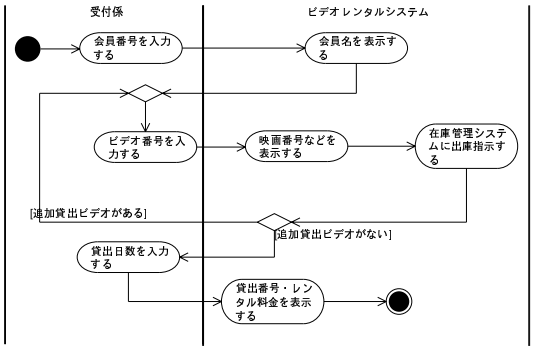

図 9: ビデオを貸出する

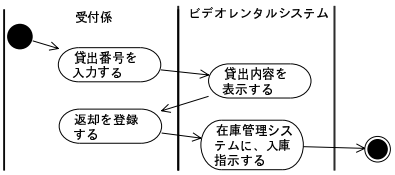

図 10: ビデオを返却する

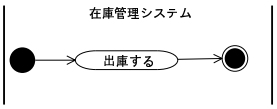

図 11: 在庫管理システムに出庫指示する

表 4: クラス図の要素候補

クラス	操作
受付係	会員情報を入力する
	貸出番号を入力する
	返却を登録する
	会員番号を入力する
	ビデオ番号を入力する
	貸出日数を入力する
ビデオレンタルシステム	会員番号を発番する
	貸出内容を表示する
	在庫管理システムに入庫指示する
	会員名を表示する
	映画番号などを表示する
	在庫管理システムに入庫指示する
	貸出番号・レンタル料金を表示する
在庫管理システム	出庫する

6.1.3 クラス図

次に, クラス図を作成する. レーンとクラス, アクティビティと操作が対応するとしたとき, 図 8, 9, 10, 11 のアクティビティ図を元に, クラス図を作成する上で, 表 4 に示す要素が必要となる.

また, 図 8, 9, 10 それぞれのアクティビティ図も対応するアクティビティを表しているため, そのアクティビティに対応する操作,

- 会員を登録する
- ビデオを貸出する
- ビデオを返却する

が得られる. これらの操作の所属するクラスは, 今の段階では不明である.

表 4 をもとに作成したビデオレンタルシステムのクラス図を図 13 に示す.

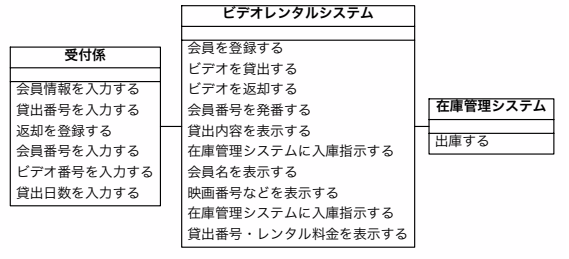

図 13: ビデオレンタルシステムのクラス図

ここでは, 所属が不明だった操作をクラス「ビデオレンタルシステム」の操作としている.

6.2 モデルの統合

独立して作成を行ったモデルを 1 つのプロジェクトとしてまとめる. 対象となるレンタルビデオシステムのクラス図をプロジェクトに取り入れる.

プロジェクト管理では作成したクラス図を登録し, 各クラスの操作に対して対応するアクティビティ図を割り当てる. この時点でモデル間の整合性チェックを行い, 問題があれば作成したモデルに手直しを加える.

6.3 プログラム生成

本 UML ツールのプログラム生成機能から java プログラムを自動生成する. クラス図からはクラスの構造上のプログラムが生成され, アクティビティ図からは操作の内部処理にあたるプログラムが生成される. 例として生成されるクラス「ビデオレンタルシステム」のプログラムを図 14 に示す.

図 14 の 6 行目から 26 行目の操作「ビデオを貸出する ()」のようにアクティビティ図と対応している場合は, アクティビティ図に示される処理内容がプログラム化される.

6.4 実装・テスト

6.3 節で得たスケルトンプログラムに実装を加えていく. 内部処理が表されている操作については,

```
1  public class ビデオレンタルシステム
2  {
3    private 受付係 t 受付係; // 関連のコード化
4    private 在庫管理システム t 在庫管理システム; // 関連のコード化
5    public ビデオレンタルシステム ( ){}
6    public void ビデオを貸出する ( )
7    {
8      System.out.println("begin : ビデオを貸出する");
9      //開始
10     ( new 受付係 ( ) ). 会員番号を入力する ( );
11     ( new ビデオレンタルシステム ( ) ). 会員名を表示する ( );
12     do{
13       ( new 受付係 ( ) ). ビデオ番号を入力する ( );
14       ( new ビデオレンタルシステム ( ) ). 映画番号などを表示する ( );
15       ( new ビデオレンタルシステム ( ) ). 在庫管理システムに出庫指示する ( );
16       if ( true /* [追加貸出ビデオがある] */ ){
17         // ループ先頭に戻る
18       }else
19       if ( true /* [追加貸出ビデオがない] */ ){
20         ( new 受付係 ( ) ). 貸出日数を入力する ( );
21         ( new ビデオレンタルシステム ( ) ). 貸出番号・レンタル料金を表示する ( );
22         // 終了
23       }
24     }while( false );
25     System.out.println("end : ビデオを貸出する");
26   }
27   ...
28 }
```

図 14: クラス「ビデオレンタルシステム」のソースプログラム

クラスの属性などを考慮して他の操作呼び出し元のクラスや条件文を定めていく (処理の手順を変えてしまっては, アクティビティ図で表した手順が無駄になってしまうため). 内部処理が空の操作は適宜コーディングしていく.

作成したプログラムをコンパイルし, 実行する. 実行結果を図 15 に示す.

```
   > java VideoLendingSystem
   ビデオを貸出する処理を行う
 1 begin : ビデオを貸出する
 2 begin : 会員情報を入力する
 3 end : 会員情報を入力する
 4 begin : 会員名を表示する
 5 end : 会員名を表示する
 6 begin : ビデオ番号を入力する
 7 end : ビデオ番号を入力する
 8 begin : 映画番号などを表示する
 9 end : 映画番号などを表示する
10 begin : 在庫管理システムに出庫指示する
11 begin : 出庫する
12 end : 出庫する
13 end : 在庫管理システムに出庫指示する
14 end : ビデオを貸出する
```

図 15: 「ビデオを貸出する」を行った実行結果

図 15 において, 10〜13 の部分を見るとわかるように, 操作の相互作用がプログラム上で表現できたといえる.

7　関連研究

文献 [6] では整合性検証についての研究を行っている. この研究では UML の 6 つのモデルを対象に, モデル間の整合性検証を行っている. 本研究で扱うクラス図とアクティビティ図間での整合性チェックについては, 4 節で示した項目の「クラスの一致」や「操作の所属」の他に,「関連の一致」を挙げている. しかし, 本研究で扱うアクティビティ図からは関連と断定できる要素や記述方法は存在しないため, ここでは考慮していない. ただし, 今回は対象としていないが, 複数のアクティビティ図からクラス間の操作の呼び出しが判断できるとき, クラス図において操作の呼び出し元/先となるクラス間に何らかの関係 (関連, 依存, 集約など) があると考えられるため, 今後はチェックすべき項目として考慮する必要がある.

また, 既存の UML ツールではモデルからプログラムの生成だけでなく, プログラムからモデルの生成といった機能も備えている [2],[3]. 加えて, モデルとプログラムがリアルタイムに対応する UML ツールも存在する [4],[5]. これらの機能は基本的にクラス図を対象にしており, UML ツールとして標準となりつつある. 本 UML ツールの中心はアクティビティ図であるため, 対象とはしなかったが, 追加機能として検討の価値があると考えている.

8　まとめ及び課題

本研究ではプログラム生成とモデルの正当性や整合性のチェックに着目した支援を考え, アクティビティ図を中心に利用した開発に着目し, 作成されたクラス図及びアクティビティ図の正当性や整合性をチェックし, ダイアグラムから対象システムの基本的な java プログラムの自動生成を行う UML ツールを構築した.

• 成果

本 UML ツールを用いることにより, 作成される UML モデル間の矛盾によるバグやモデルからコーディングする際の人為的なミスの削減, さらにモデルの正当性・整合性チェックにより UML の学習支援としての効果も期待できる.

• 課題

今回はアクティビティ図の特に基本的な要素を用いた. 他の要素を加えた場合の正当性や整合性のチェック, またプログラム生成においてどのような効果があるか見極める必要がある.

また, シナリオやユースケース記述からのアクティビティ図の生成, 作成したアクティビティ図とシーケンス図などの他の UML モデルの生成を実現することで, 開発におけるアクティビティ図の利用方法と効果を確立し, アクティビティ図主導のソフトウェア開発のさらなる価値を高めることができると考える.

参考文献

[1] UML(Unified Modeling Language), http://www.omg.org/, Object Manegement Group (OMG)

[2] IIOSS(**I**ntegrated **I**nter-exchangeable **O**bject-modeling and **S**imulation **S**ystem for Opensource Software Environment), http://www.iioss.org/

[3] Jude(Java and UML Developer's Environment), http://objectclub.esm.co.jp/, 株式会社永和システムマネジメント

[4] Together, http://www.techmatrix.co.jp/together/, Borland

[5] Rational Rose, http://www-6.ibm.com/jp/software/rational/, IBM

[6] 大西淳, UML におけるモデル整合性検証支援システム, 電子情報通信学会論文誌 2001/6 Vol.J84-D-I No.6, 2001

[7] P.Stevens,R.Pooley, 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エア工学 –UML を使って –, ピアソン・エデュケーション, 2000

[8] 河合昭男, UML がわかる, 技術評論社, 2002

[9] SDMetrics, http://www.sdmetrics.com/LoM.html

[10] T.Katayama, Proposal of a Supporting Method for Diagrams Generation with the Transformation, Asia-Pacific Softw. Eng. Conf.(APSEC 2002), pp.475-484(2002)

[11] 青山幹雄, 中谷多哉子 編著, オブジェクト指向に強くなる, 技術評論社, 2003