2008－SE－161 (11)
2008／9／26

GUIプログラミング課題の自動採点方式について

内藤広志[†] 齊藤 隆[††] 水谷泰治[††]

JAVA言語で書かれたGUIプログラムの実行テストを効率的におこなうための方式とその評価について報告する．本方式では，AWTをエミュレーションするために擬似AWTを開発した．擬似AWTは，AWTと同様にGUIコンポーネントの内部状態を保持するが，スクリーンへの描画の変わりに要求されたメソッドの名前と引数の値を出力ログへ書き込む．実行テストでは，出力ログを調べることでプログラムが正しいかを検査する．また，本方式の評価のために，類似システムとの機能や実行時間の比較をおこなった．

Automated Assessment Method for GUI Programming assignments

NAITO HIROSHI,[†] SAITO TAKASHI[††] and MIZUTANI YASUHARU[††]

This paper describes a Java GUI program testing method which perform the testing efficiently, and a evaluation of Hercules Computer Aided Assessment (CAA) system with this method. To emulate the Java AWT, a dummy AWT was developed. The dummy AWT maintains internal states of the GUI components like the AWT. When the AWT is requested to draw some objects to a screen by a program, the dummy AWT generates an output log with the requested method names and its arguments values. In the execution test, an assessment of a program can be obtained by examining the output log. To evaluate this method, the function and peformance are compared with related systems.

1. はじめに

情報処理学会情報処理教育委員会が発行した「日本の情報教育・情報処理教育に関する提言2005」[1]に提言されているように，プログラミング教育は『「手順的な自動処理」の構築』のために重要である．当学部では，C演習I，C演習II，Java演習などの多くのプログラミング演習科目を設け，学生が『「手順的な自動処理」の構築』の方法を理解し，実際にプログラムを作成できるように図っている．特にJava演習では，オブジェクト指向プログラミングを理解するためのコマンドラインインタフェース (CLI) のプログラムを作成するだけでなく，ボタンやマウス操作を用いたグラフィカルユーザインタフェース (GUI) をもつアプレットを作成できることを学習目標に上げ，5回の単元を割り当てGUIプログラミングの演習を実施している．このようにGUIプログラミングを重視している理由は，1) GUIプログラムはGUIコンポーネントのライブラリを利用して作成するためクラスの継承やコンストラクタなどのオブジェクト指向プログラミングの重要な概念を具体的な例を通して理解できる，2) GUIアプリケーションを日常的に使用しているため，GUIを実現する方法 (イベント駆動プログラミング) を理解することが実用的なプログラムを作成する上で重要である，3) CLIプログラムで中心となる数値計算やテキスト処理では演習が単調となるため，複雑な図形を描画したり，マウスやボタンなどを用いたGUIプログラムを作成することで学生の意欲を高めることができるためである．

本学部のプログラミング演習では，効果的なプログラミング教育をおこなうため，少数の大きなプログラムを作成する代わりに多数の小さいプログラムを作成する課題を学生に課している．例えば，2コマのJava演習の場合は毎回7個のプログラミング課題を出題している．しかし，プログラミング演習の履修者が400名から600名と非常に多いため，プログラミング課題を手作業で評価していたのでは採点ミスが発生しやすく，非常に時間がかかる．その上，CLIプログラムの評価は，テストデータを指定してプログラムを実行する処理はshell言語を用いて容易に自動化できるた

† 大阪工業大学 情報科学部 情報メディア学科
Department of Media Science, Faculty of Information Science and Technology, Osaka Institute of Technology
†† 大阪工業大学 情報科学部 情報システム学科
Department of Information Systems, Faculty of Information Science and Technology, Osaka Institute of Technology

め，出力データが正しいかを確認する部分を手作業でするだけでよいが，GUIプログラムの評価は，マウスやキーボードを使ってプログラムを実際に操作し，画面の表示を確認する必要があるため，更に時間がかかり，その上正確な評価は不可能である[2]．

これらの問題を解決するために，ソフトウェアテスト技術を用いて学生のプログラムを採点するコンピュータ支援評価（Computer-Aided Assessment, CAA）システムが開発され，実際の授業で利用されている[3),4)]．本学部でも，C言語，Java言語，XML言語の課題を評価するHerculesというCAAシステムを開発し，2003年度より機能を順次拡張しながらプログラミング演習で利用している[5)]．Herculesシステムは，課題の提出期限を過ぎた後に学生の提出プログラムを採点するだけでなく，演習時間内に学生の進捗状況をモニタリングするためにも使用する．そのため，コンパイラエラーを含むプログラムも評価でき，かつ，採点結果のキャッシュをもちいることで多数のプログラムを効率よく評価できる．

本報告では，Java言語で書かれたGUIプログラムの実行テストを効率的におこなうための採点方式について，2章でGUIプログラミング課題の例を上げ，GUIプログラムを評価するために必要な項目と問題点について述べ，3章でHerculesのシステム構造と検査内容の記述法について述べ，4章で本方式の処理フローと実行テストの記述法を述べ，5章で類似システムとの比較をすることで本方式の評価をおこない，6章でまとめを述べる．

2. GUIプログラミング課題の評価

GUIプログラムは，GUIライブラリが提供するボタンやテキストフィールドなどのGUIコンポーネントを利用してイベント駆動プログラムとして記述する．本報告の演習科目では，Java言語とGUIライブラリのAWTを用いてGUIプログラミング課題を作成している．

ソフトウェアのテストは，テストケースを実行して検出された障害(failure)からプログラムの欠陥(fault)を識別し，その原因となっているエラー(Error)を同定するプロセスである[6)]．GUIプログラムがイベント駆動で処理されるため，GUIプログラムの実行テスト(GUIテストと省略)もイベント駆動となる[7)]．つまり，テスト対象プログラム(System Under Test, SUT)を起動した後，(A)必要なGUIコンポーネントがあるか，GUIコンポーネント間の親子関係やレイアウト(位置と大きさ)が期待するものと等しいかを検査した後，(B)イベントをSUTに通知することでSUTの機能を検査する．この検査処理は，(1) GUIコンポーネントを特定する，(2) そのGUIコンポーネントにイベントを通知する，(3) 期待するイベント処理が実行されたかを検証するの3つの処理を繰り返すことでおこなわれる．(3)の検証処理は，(a) GUIコンポーネントの内部状態が期待される値と等しいかの検証と，(b) 期待される図形などのイメージがウィンドウに描画されているかの検証に分類される．

図1 GUIプログラムの例

図1は，テキストフィールドにテキストを入力した後で"描画"ボタンを押すと，そのテキストを長方形の中に表示するアプレットである．このアプレットの実行テストでは，(A)でテキストフィールドとボタンがあるかを検査し，(B)で，(1) テキストフィールドとボタンを特定し，(2) テキストフィールドに"Hello"と入力するイベントと"描画"ボタンを押すイベントを通知し，(3) テキスト"Hello"と長方形がウィンドウに描画されたかを検証する．

GUIプログラミング課題の評価も同様な処理となるが，多数の学生が作成したバリエーションの多いプログラムを検証するため，1つの製品を検証するためのGUIテストとは異なる機能が必要である．まず，多数のプログラムを検証する必要があるため，効率よく検証できる方式が必要である．また，一般にGUIテストはシステムの回帰テストや受け入れテストとして実施されため，(3)の検証処理では期待される値と完全一致するかを調べる．しかし，学生は様々なやり方で座標計算をおこなうため1，2ビット程度の誤差がよく発生するため，描画される図形の位置や大きさの値の検証では部分一致の機能が必要である．

3. Herculesシステムの概要

Herculesを用いたJava演習では，学生はLinuxマシン上でJavacコンパイラ，FireFoxブラウザ，emacsエディタなどの基本的な開発ツールを使ってプログラムを作成する．完成したプログラムは電子メールやHTMLフォームを使って提出するのではなく，各学

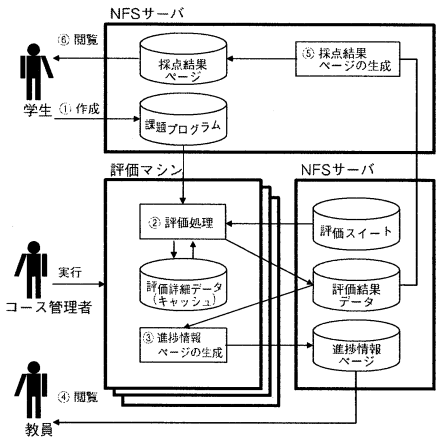

図 2　全体処理フロー

生のホームディレクトリ下の課題の説明文で指定されたファイルに保存する．その結果，Hercules は学生が作成中のプログラムを評価できる．

Hercules は，1) 演習時間中に学生が作成中のプログラムを評価して進捗情報を生成し，その情報を教員に提示するモニタリング機能と，2) 課題提出の締切後に学生が完成したプログラムを評価し，その結果を学生に提示する採点機能をもつ．

図 **2** に示すように，モニタリング機能では，学生が課題プログラムを作成したとき (①)，課題の検査方法を記述した評価スイートを実行して学生の課題プログラムを検査し評価結果データを作成する (②)．次に，評価結果データから全クラスの進捗状況を示す HTML ページと，クラス毎の進捗状況を示す HTML ページ (図 **3**) を生成する (③)．なお，学生の課題プログラムに変更がない場合は検査をおこなわず，評価マシンのローカルなディスクに格納された評価詳細データ（キャッシュ）より評価結果データを生成する．教員はブラウザを使って進捗情報ページを閲覧して，進捗状況を見て指導をおこなう (④)．

採点機能では，モニタリング機能で作成した評価結果データより学生の採点結果ページを各の学生のホームディレクトリ下に生成し (⑤)，学生が閲覧できるようにする．

なお，学生のホームディレクトリ，評価スイート，採点結果データは NFS サーバーに格納されているので，複数の Linux マシンを使って評価処理を並行に処理している．

Hercules では，プログラムを実行せずにソースコードを解析する静的な評価と，プログラムを実行してプログラムの機能を検査する動的な評価の両方を実行す

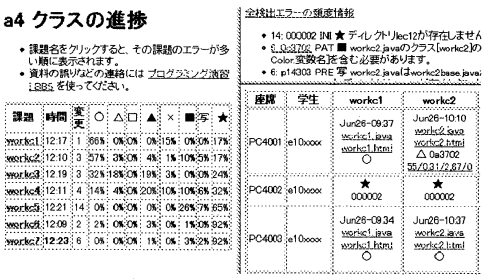

図 3　クラスの進捗表示ページ

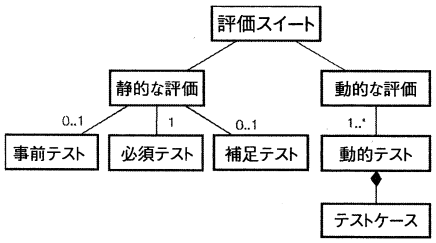

図 4　評価スイートの構造

る．図 **4** の評価スイートの構造が示すように複数のテストに分けて記述する．静的な評価は，例題プログラムから変更があるか，コンパイルエラーの原因となるコードがあるかを検査する事前テスト，課題の仕様に合ったプログラム構造であるかを検査する必須テスト，冗長なコードがないかを検査する補足テストに分けて記述する．動的な評価の実行テストの各テストケースでは，プログラムの出力が課題の仕様とおりか，メソッドの呼び出し関係や回数が正しいかを検査する．

学生のプログラムの評価結果は，より致命度の高いものから★（ファイルが存在しない），写（例題プログラムから変更がない），■（課題の学習目標を満たしていない），×（コンパイルエラー，プログラムの構造に誤りがある），▲（実行エラー），□（エラーテストケースの誤り），△（冗長なコードがある，出力データが正確でない），○（正解）の 8 種類の評価値として表す．

4. GUI プログラミング課題の採点方式

GUI コンポーネントを実際に生成してウィンドウに描画する処理は OS のリソースを消費し処理時間がかかる．GUI プログラムの実行テストを効率的におこなうために，本方式では，AWT をエミュレーションする擬似 AWT を開発した．擬似 AWT は，AWT と同様に GUI コンポーネントの内部状態を保持するが，描画メソッドが呼び出されたときは，スクリーンへの描画をせずにメソッドの名前と引数の値を出力ロ

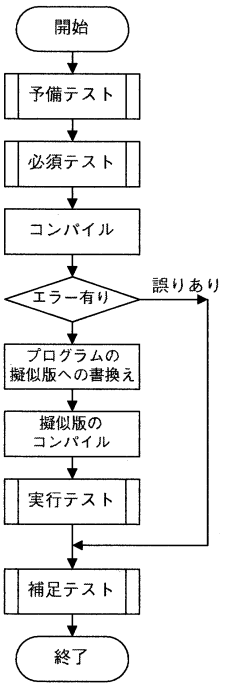

図 5　テストの実行順番

グへ書き込む．擬似 AWT を用いて実行テストをするために，図 **5** に示すように，コンパイルが成功した場合だけ，AWT の代わりに擬似 AWT を使用するようにプログラムを書き換えた擬似版プログラムをテストケースに書かれた入力イベント列を指定して実行し，擬似 AWT がどのように利用されたかを示す実行ログを処理結果として生成する．実行ログをパターン検索によって必要なメソッドの呼び出しがあるかを調べることでプログラムが正しいかを検証する．以降，擬似 AWT の構成と機能，入力イベントの表現，実行ログの表現，検証処理の記述法を順に述べる．

4.1　擬似 AWT の構成と機能

本 Java 演習では，アプレットを用いた基本的な GUI プログラムを作成し，フレームやメニュー，レイアウトマネージャを使用したプログラミング課題は出題していない．そのため，図 **6** に上げる AWT の GUI コンポーネント中の括弧で囲まれていいないクラスだけを擬似 AWT でサポートしている．

擬似 AWT の GUI コンポーネントは，AWT の GUI コンポーネントの public メソッドとコンストラクタをほぼサポートし，メソッドで変更可能なラベル，親子関係，選択状態などの内部状態を保持する．GUI コンポーネントの位置と大きさはラベルの文字列の長さなどからおおよその値を計算する．また，GUI コンポーネントのイベント処理をおこなうために，主要なイベントクラスの ActionEvent，InputEvent，MouseEvent，ItemEvent と対応するイベントリスナーインタフェースを実装している．

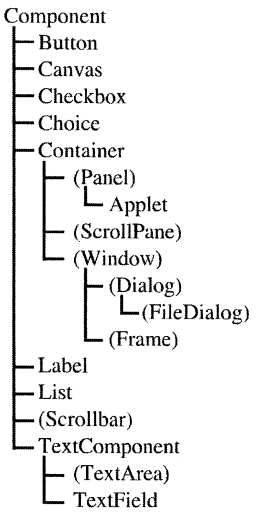

図 6　擬似 AWT の GUI コンポーネント

これらの GUI コンポーネントクラスの他に，フォント情報を表すクラスの Font と FontMetrics や，色情報を表すクラスの Color や，描画をおこなうためのクラスの Graphics を実装している．FontMetrics クラスでは，代表的なフォントのメトリックス情報を擬似的に計算する．Graphics クラスでは，setFont メソッドと setColor メソッドで設定されたフォント情報とカラー情報を保持する．

4.2　入力イベントの表現

実行テストの入力データとして指定するイベントは，GUI コンポーネントのクラス名，GUI コンポーネントの名前または序数，イベント記述の 3 つの値で表現する．最初の 2 つの値によって，次のようにイベントを通知する GUI コンポーネントを指定する．

(1)　クラス名と名前：指定したクラス名の指定した名前をもつ GUI コンポーネントを指定する．なお，ボタンなどではラベルの値が名前となる．

(2)　クラス名と序数：GUI コンポーネントの階層構造をたどり，序数で指定した番目の指定したクラス名の GUI コンポーネントを指定する．

イベント記述には，各イベントクラスのコンストラクタの表現と類似したものを使い，イベントクラス名

の後の括弧の中にイベントの詳細情報を書く．ラベルが"描画"の Button に ActionEvent イベントを通知するイベントは次のように書く．

```
Button("描画",ActionEvent())
```

TextField に文字列"Hello"を設定するイベントは次のように書く．

```
TextField(0,setText("Hello"))
```

マウスイベントは，mouseClicked，mousePressed，mouseReleased などのイベント名の後にクリックした位置の X 座標と Y 座標，キー修飾子，マウスボタンを書く．例えば，アプレット上の点 (10, 20) でマウスの右ボタンをクリックしたイベントは次のようになる．

```
Applet(0,mouseClicked(10,20,0,1))
```

複数のイベントを通知したいときは，空白で区切りイベントの記述を通知したい順に書く．

4.3 実行ログの表現

擬似 AWT が出力する実行ログは 3 つの形式がある．GUI コンポーネント，Font，Color のインスタンスを生成したときは，「クラス名. クラス名 (引数の値)」の形式のメッセージを出力する．

```
Button.Button([描画])
Font.Font(Serif, 0, 60)
Color.Color(56, 141, 199)
```

GUI コンポーネントのメソッドが呼び出されたとき，「クラス名 (名前). メソッド名 (引数の値)」の形式のメッセージを出力する．GUI コンポーネントの内部状態を変更しないメソッドのログは出力しない．なお，Applet は 1 個しか存在しないので名前を省略する．

```
Button(描画).addActionListener(Applet())
Applet.add(Button(描画))
TextField(textfield3).setText([Hello])
```

ウィンドウにどのようにが描画されたかを検証できるようにするため，Graphics クラスのメソッドが呼び出されたとき，「Graphics. メソッド名 (引数の値) with コンテキスト」の形式のメッセージを出力する．コンテキストには，図形の描画用のメソッドの場合は，setColor メソッドで設定された色情報が，文字列の描画用のメソッドの場合は，setFont メソッドで設定されたフォント情報が書かれる．

```
Graphics.drawRect(20, 40, 100, 40)\\
  with Color.gray
Graphics.drawString([Hello], 30, 60)\\
  with Font(Dialog,0,12) Color.black
```

図 1 のアプレットで，テキストフィールドに"Hello"と入力してから"描画"ボタンを押した場合，図 **7** のように実行ログが出力される．先頭が###の行はイベントの通知を表し，先頭が数値の行はイベントを処理するメソッド起動を表す．数値はメソッド起動の順番を示す連番である．先頭が空白の行はメソッド起動から呼び出される処理を表す．

4.4 検証処理の記述法

実行テストの検証処理は，擬似 AWT が出力した実行ログ中に特定のパターンがあるかを調べることで可能である．Hercules の実行テストは，表 **1** の検査コマンド☆とシェル言語を使って記述する．複雑な検証処理の記述を容易にするため表 **2** のパターン式をサポートし，fpatternInOutput.sh 検査コマンドの引数にパターン式を指定して検査したい条件を記述する．

実行テストは図 **8** のようにテストケースごとに 1 つのファイルに記述する．検査コマンドの-N オプションは検査項目を識別するための番号を指定するもので，-e オプションは検査項目の評価値を指定するものである．例としてあげた図 8 は図 1 の実行テストの init メソッドの処理を検査する部分を記述したものである．

5. 関連研究との比較

5.1 GUI テストツールとの比較

GUI テストには Capture/Replay 方式と，JUnit を GUI テスト用に拡張した方式 (フレームワーク方式と省略) がある[8]．Capture/Replay 方式は，まず，実際に SUT のプログラムを実行してテスターがおこなった GUI 操作とその結果の画面の描画イメージを記録する．次に，テストの実行のときには，記録された GUI 操作を自動的に実行し，SUT の実行結果のウィンドウの表示と記録された描画イメージとを比較することで SUT の機能を検証する．Capture/Replay 方式は，GUI 操作だけでテストケースを作成できる点が優れているが，描画イメージのビットマップデータを比較することで描画処理を検証するため時間がかかる[9]．また，描画イメージ中のどの図形オブジェクトの描画が誤っているかを検出できないためプログラミング課題の評価方式として適さない．

☆ 静的な評価用の検査コマンドは文献[5] を参照．

```
1 sworkf2.init()
  1 1 TextField.TextField([], 10)
  1 2 Applet.add(TextField(textfield0))
  1 3 Button.Button([描画])
  1 4 Applet.add(Button(描画))
  1 5 Button(描画).addActionListener(Applet())
2 sworkf2.paint(Graphics)
  2 1 Graphics.setColor(Color.gray)
  2 2 Graphics.drawRect(20, 40, 100, 40) with Color.gray
### TextField(textfield0).setText("Hello")
### Button(描画).ActionEvent()
3 Button(描画).actionPerformed(ActionEvent(Button(描画)))
  3 1 TextField(textfield0).getText()
  3 2 Applet.repaint(1)
4 sworkf2.paint(Graphics)
  4 1 Graphics.setColor(Color.gray)
  4 2 Graphics.drawRect(20, 40, 100, 40) with Color.gray
  4 3 Graphics.setColor(Color.black)
  4 4 Graphics.drawString([Hello], 30, 60) with Font(Dialog, 0, 12) Color.black
```

図 7　図 1 のアプレットの実行ログの例

表 1　実行テスト用の検査コマンド

検査コマンド	機能
fexecute.sh	プログラムを実行する
fhaveOutput.sh	実行ログの有無，サイズの上限および下限を検査する
fpatternInOutput.sh	実行ログ中にパターンがあるかを検査する
fgetPatternInOutput.sh	実行ログ中の何行目にあるパターンを取り出す
fcheckEventRepaint.sh	指定したイベントの後の再描画処理にパターンがあるかを検査する
fcallgraph.sh	関数が呼ばれた回数，どの関数から呼ばれたかを検査する

表 2　パターン式の種類

パターン式		検査内容
INCLUDE	*pattern* [MORE|EQUAL|LESS] *occurence*	正規表現 *pattern* の出現回数を *occurence* と比較する
LINE	*pattern from to*	正規表現 *pattern* が *from* 行から *to* 行までにあるか調べる
AND	*pattern1 pattern2*	正規表現 *pattern1* と *pattern2* を含む行があるか調べる
OR	*pattern1 pattern2* ...	いずれかの正規表現を含む行があるか調べる
NOT	*pattern1 pattern2* ...	いずれかの正規表現を含む行がないか調べる

JUnit の拡張方式では，2 章で述べた (B) の (1) から (3) に対応して，GUI コンポーネントを特定する機能，GUI コンポーネントにイベントを通知する機能，イベント処理の結果を取得する機能をもつ GUI 自動化ライブラリを提供し，このライブラリを使ってテストプログラムを作成するためのテスティングフレームワークを提供する．この方式の Java 言語用のオープンソースのツールに，Abbot[10] と，GUI 自動化ライブラリだけを提供しフレームワークには JUnit を使用するする Jemmy[11] などがある．両者とも AWT と Swing のテストを可能にする．フレームワーク方式は，Java 言語を使ってテストプログラムを作成するため，非常に作成コストが高い．特に，GUI コンポーネントを特定する処理はリフレクション機能を用いるため，JUnit を使用する場合と比べて，より高度なプログラミング知識を必要としテストプログラムもより複雑となる．また，描画処理の検証も画面イメージを用いるため，Capture/Replay 方式と同じ問題がある．

両方式ともテストをおこなうときに SUT を起動してスクリーン描画をおこなうため，多数のプログラムを評価する必要のある CAA システムでは時間がかかり効率が悪い．

処理効率を比較するために，同じイベントを実行するテストケースを本方式の擬似 AWT と Jemmy で作成し，実行ログを出力するために必要な処理時間を計測した．テストに用いたのは図 **9** の手動信号機のアプレットで，ボタンを押すと対応する色が変わる．テストケースは「Green ボタンを 2 回押す」と「Green, Yellow, Red, Red の順にボタンを押す」の 2 つで，擬似 AWT では 300 ミリ秒がかかり，Jemmy では約 4

```
setenv HTITLE   "'Hello' を TextField に入力し、ボタンを押す"
setenv HPRARGS  'TextField(0,setText("Hello"))  Button("描画",ActionEvent())'
fexecute.sh

### 出力があるかを調べる。
fhaveOutput.sh     -N "A1"

### init メソッドの処理の検査
# 「描画」というラベルの Button を生成しているか？
fpatternInOutput.sh -N "A2" -e "▲" INCLUDE "^${S}1.*Button\.Button\(\[描画\]\)" || exit 1
# 1個の Button をアプレットに登録しているか？
fpatternInOutput.sh -N "A3" -e "▲" INCLUDE "^${S}1.*Applet\.add\(Button\(描画\)\)" 1 EQUAL || exit 1
# Button にイベントリスナーを登録しているか？
fpatternInOutput.sh -N "A4" -e "▲" INCLUDE "^${S}1.*Button\(描画\)\.addActionListener\(Applet.*\)" || exit 1
# TextField を生成しているか？
fpatternInOutput.sh -N "A5" -e "▲" INCLUDE "^${S}1.*TextField\.TextField\(.*\)" || exit 1
# 1個の TextField をアプレットに登録しているか？
fpatternInOutput.sh -N "A6" -e "▲" INCLUDE "^${S}1.*Applet\.add\(TextField\(textfield0\)\)" 1 EQUAL || exit 1
# TextField にイベントリスナーを登録しているか？
fpatternInOutput.sh -N "A7" -e "▲" NOT "^${S}1.*TextField\(描画\)\.addActionListener\(Applet\(\)\)" || exit 1

### init メソッドの直後の paint メソッドの処理の検査
fpatternInOutput.sh -N "A8" -e "▲" INCLUDE "^${S}2.*Graphics\.drawRect\(.*\)" || exit 1
fpatternInOutput.sh -N "A9" -e "▲" NOT "^${S}2.*Graphics\.drawString\(.*\)" || exit 1

exit 0
```

図 8　実行テストの記述例

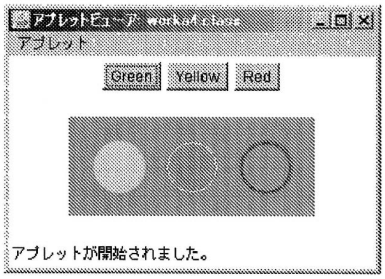

図 9　評価用 GUI プログラムの例：手動信号機

倍の 1300 ミリ秒がかかった．この結果，擬似 AWT を用いる本方式の処理効率が高いことがわかった．

5.2 GUI プログラムの採点システムとの比較

前節の GUI プログラムのテストツールの他に，プログラミング課題の採点用ツールも存在する．次に上げる全ツールは Java 言語で書かれた GUI プログラムを検査するものである．

AJGUITE[12] は，XML 言語で記述した GUI イベントテスト仕様から採点用の Jemmy プログラムを生成する．Java でプログラムをテストプログラムを作成する必要がないため，テストケースの作成が容易である．描画イメージの検証をテスト仕様で記述できないため，図形を描画するアプレットのテストが不可能である．また，Jemmy を使用しているため Jemmy と同じ欠点をもつ．

GUI_Grader[13] は，ユーザインタフェース設計の演習をおこなうことを目的としたツールで，AJGUITE と同様に Jemmy を使用したテストプログラムを生成する．GUI コンポーネントのクラス名，ラベル名，順番が模範プログラムと同一でなければならないという AJGUITE の制限を解消するために，GUI コンポーネントクラスを機能によって抽象化したグループ名を導入することで，より多様なデザインのプログラムを採点できる．しかし，機能は高くなっているが，GUI テストとしては AJGUITE と同じ欠点をもつ．

GuiG[14] は，既存の CAA システム Ceilidh に GUI プログラムの採点機能を追加することで，開発コストを少なくすることを目的に開発されたツールである．GuiG も Jemmy を使用しているが，テストプログラムは自動生成ではなく教員がプログラミングする．CAA システムに組み込まれているため，学生へのフィードバック機能や，課題の再提出機能などの CAA システムのコース管理機能が使え，実際の演習科目で利用する上で有用である．また，仮想フレームバッファを用いることで，スクリーンに描画することなくテストできるため，サーバサイドでの GUI テストが可能である．ただし，Jemmy を使ってテストプログラムを作成するため，テストケースの作成にコストがかかり，

柔軟なテストケースの作成が難しい．

JEWL[15] は，上の 3 つのツールとは異なるアプローチを取っており，教育用に設計された，機能が AWT/Swing より制限された独自の GUI ライブラリを使う．採点をおこなう場合は，擬似 AWT と同じく GUI のエミュレーションモードで実行し，Java 言語で書かれたテストプログラムから GUI コンポーネントの内部状態をアクセスして機能の検証をおこなう．ただし，図形の描画も独自で，キャンバスに描画オブジェクトを格納することでおこなわれる．そのため，描画イメージの検証はキャンバスに格納された描画オブジェクトを比較することで実現できるが，効率はよくない．

6. おわりに

Hercules を利用することで，課題の締切後にプログラムを採点するだけでなく，演習時間中に学生の進捗状況に合わせて指導が可能となっている．本方式を用いることで，通常の CLI プログラムだけでなく，GUI プログラムの評価もおこなえるようになり，出題できるプログラムの範囲も広がり，効果的なプログラミング教育が実施できている．

しかし，品質の高い評価スイートを作成するためには，スキルや経験をもった人材が必要である．テストケースの作成技法や shell プログラミングのスキルが必要であり，学生の誤りの多い条件を推測することが必要なため 2 年以上の経験が必要である．更に品質を高めるには，複数の作成者による相互レビューを実施することが重要である．

必要なコストは作成時間と高速のコンピュータである．GUI プログラミング課題の評価スイートを作成するのは難しいため 1 から 2 時間がかかる．特にマウス操作の課題は時間を必要とする．品質を高めるために，課題の締切後に検査結果を調べて評価スイートのバグを取るが，課題によっては更に 1，2 時間を必要とする．本方式によって，GUI プログラミング課題の検査は手作業と比べると効率化されたが，1 個の GUI プログラミング課題を評価するには 4 秒から 8 秒を必要するため，600 名の学生の 7 個の課題を採点するのに複数台のコンピュータを用いても数時間がかかる．

今後の研究として，より分散処理を進めて評価処理の高速化を図るとともに，評価スイートの開発コストを少なくするための方法を検討したい．

参考文献

1) 情報処理学会情報処理教育委員会：日本の情報教育・情報処理教育に関する提言 2005(2006.11 改訂/追補版) (2006).
2) Memon, A.M.: GUI Testing: Pitfalls and Process, *Computer*, Vol.35, No.8, pp.87–88 (2002).
3) Ala-Mutka, K.: A Survey of Automated Assessment Approaches for Programming Assignments, *Computer Science Education*, Vol. 15, No.2, pp.83–102 (2005).
4) Douce, C., Livingstone, D. and Orwell, J.: Automatic test-based assessment of programming: A review, *Educational Resources in Computing*, Vol.5, No.3 (2005).
5) 内藤広志，斉藤　隆：プログラミング演習のための進捗モニタリングシステム，情報処理学会研究報告 コンピュータと教育研究会報告，No.13, pp.33–40 (2008).
6) 松本吉弘 (訳)：ソフトウェアエンジニアリング基礎知識体系—SWEBOK2004，オーム社 (2005).
7) Jorgensen, P.: *Software Testing: A Craftman's Approach 3th Edition*, Auerbach Pub. (2008).
8) Li, K. and Wu., M.: *Effective GUI Test Automation: Developing an Automated GUI Testing Tool*, SYBEX (2005).
9) Takahashi, J.: An Automated Oracle For Verifying GUI Objects, *ACM SIGSOFT Software Engineering Notes*, Vol. 26, No. 4, pp. 83–88 (2002).
10) Wall, T.: Getting Started with the Abbot Java GUI Test Framework, `http://abbot.sourceforge.net/` (2008).
11) Netbeans.org: Jemmy Module, `http://jemmy.netbeans.org/`.
12) Sun, Y. and Jones, E.: Specification-driven automated testing of GUI-based Java programs, *42nd annual Southeast regional conference*, pp. 140–145 (2004).
13) Feng, M. and McAllister, A.: A Tool for Automated GUI Program Grading, *Frontiers in Education, 36th Annual Conference*, pp. 7–12 (2006).
14) Surakka, S., Auvinen, J. and Ihantola, P.: Automatic grading of graphical user interface programs using Jemmy, *5th Finnish/Baltic Sea Conference on Computer Science Education*, pp.49–56 (2006).
15) English, J.: Automated Assessment of GUI Programs using JEWL, *ACM SIGCSE BULLETIN*, Vol.36, No.3, pp.137–141 (2004).