Engenharia Reversa de Sistemas Interactivos Desenvolvidos em Java2/Swing

João Carlos Silva[1,2] José Creissac Campos[1] João Alexandre Saraiva[1]
[1] Departamento de Informática/CCTC, Universidade do Minho
[2] Escola Superior de Tecnologia, Instituto Politécnico do Cávado e do Ave
jcsilva@ipca.pt, {jose.campos,jas}@di.uminho.pt

Resumo

A manutenção e evolução de sistemas interactivos, mantendo um elevado nível de usabilidade, dá origem a problemas importantes que afectam a eficência e eficácia dos sistemas. Pelas suas características este tipo de sistema é bastante vulnerável aquando da execução de alterações. As metodologias e técnicas actuais não abordam de forma satisfatória estes processos. Neste trabalho pretende-se combinar a programação funcional com programação estratégica, code slicing e modelos com semântica formal na tentativa de fortalecer a tese de que a aplicação destas metodologias e tecnologias no processo de engenharia reversa de sistemas interactivos permite melhorar significativamente o grau de flexibilidade e suporte à manutenção e evolução do sistema.

Palavras-Chave

Interface, Programação Funcional, Code Slicing, Programação Estratégica, Modelos, Haskell, Java/Swing

1. Introdução

Um dos problemas que afectam o sucesso dos projectos de engenharia de software que envolvem uma componente de interacção com o utilizador é a usabilidade [Cam04]. Vários estudos têm sido realizados na área de métodos formais aplicados à análise da usabilidade [dSGD98, BA95]. A norma ISO DIS 9241-11 define a usabilidade de um sistema como a eficácia, eficiência e satisfação com que utilizadores atingem determinados objectivos em ambientes específicos. A eficácia diz respeito à possibilidade (ou não) de o utilizador poder atingir os seus objectivos utilizando o sistema num dado contexto. A eficiência tem a ver com o maior ou menor esforço que o utilizador deverá aplicar para atingir esse objectivo. A satisfação é uma medida subjectiva do grau de agradabilidade na utilização do sistema. Avaliar a qualidade de um sistema interactivo implica avaliar a facilidade de utilização do mesmo.

Para que as interfaces tenham uma boa usabilidade é necessário realizar previamente um desenho e uma implementação adequada. Existem ferramentas para auxiliar o desenvolvimento rápido de interfaces através da programação visual das componentes gráficas. Todavia, as interfaces continuam muitas vezes difíceis de entender para os utilizadores finais. Em muitos casos, os utilizadores têm dificuldades em identificar todas as funcionalidades oferecidas pelo sistema, ou dificuldades em alcançar tais funcionalidades.

Tradicionalmente, a área da interacção homem-máquina tem-se preocupado com aspectos relacionados com o desenho de interfaces, enquanto os engenheiros de software têm principalmente considerados aspectos de implementação. Existe um desencontro entre estas duas áreas, o que leva a que a qualidade do software não seja a melhor. Torna-se necessário que os engenheiros de software tenham em mente preocupações acerca do desenho das interfaces aquando do desenvolvimento de sistemas interactivos.

Hoje em dia é também comum falar-se dos custos na manutenção de sistemas interactivos. Segundo [eES80], 70% do esforço de programação concentra-se na manutenção. Acontecem sempre situações em que existem problemas com software já desenvolvido. Por exemplo, os requisitos iniciais e as tecnologias estão em constante mudança o que implica a alteração do software. Nos primeiros desenvolvimentos de soluções informáticas, a disciplina da engenharia de software concebia o processo de desenvolvimento dos sistemas através de metodologias pouco flexíveis e orientadas fundamentalmente para projectos novos. Rapidamente se observou que os sistemas informáticos têm um período de vida bastante extenso, sendo necessário adaptar os referidos sistemas às diversas evoluções, tanto funcionais como tecnológicas. A engenharia de software tem respondido a estas necessidades adaptando e promovendo metodologias e processos que visam o desenvolvimento de soluções informáticas num processo em espiral [Boe88], tentando aumentar a rapidez com que são incorporados novos requisitos (ou alterações de requisitos) nas soluções informáticas.

Nestas situações a utilização de engenharia reversa pode

contribuir para a resolução de tais problemas [MBN03b, ESS03]. Tendo em vista explorar este aspecto, pretende-se com este trabalho desenvolver uma ferramenta para a engenharia reversa de sistemas interactivos.

O projecto IVY[1], no qual se enquadra este trabalho, tem como objectivo desenvolver ferramentas para suportar uma abordagem ao desenvolvimento de sistemas interactivos em que se procura facilitar a comunicação entre duas comunidades: a Interacção Humano-Computador (IHC) e a Engenharia do Software [CH03]. O trabalho já realizado nesta abordagem é baseada em modelos e pretende possibilitar aos engenheiros de software uma maior autonomia na consideração de aspectos de usabilidade relacionados com o comportamento do sistema, bem como identificar os pontos em que é necessário recorrer ao auxílio de peritos em IHC.

Neste artigo, apresentamos os resultados do trabalho de investigação realizado na área da engenharia reversa de sistemas interactivos. O nosso objectivo é produzir um protótipo funcional de engenharia reversa combinando várias funcionalidades, nomeadamente: programação estratégica, slicing de programas e modelos abstractos. Pretende-se que a ferramenta seja capaz de contemplar os seguintes pontos:

- gerar modelos do comportamento dos sistemas interactivos a um nível adequado de abstracção;
- modelar a interacção entre o sistema e o utilizador e não a arquitectura dos sistemas;
- ser capaz de gerar modelos para sistemas com elevado grau de dinamismo.

Como se verá, a partir de um qualquer programa JAVA, conseguimos já extraír dois tipos de modelos: modelos de interactores [SCS06] e máquinas de estados. Cada tipo de modelo simula o comportamento interactivo induzido por invocações de rotinas da biblioteca gráfica SWING a um dado nível de abstracção.

Na secção 2 descrevemos sucintamente o projecto IVY. A seguir, na secção 3, descrevemos alguns trabalhos alternativos. Na secção 4 explicamos as técnicas utilizadas no processo de engenharia reversa de interfaces gráficas. A secção 5 descreve duas representações distintas para abstrair o comportamento de sistema interactivos. A secção 6 apresenta os resultados obtidos com a aplicação do protótotipo a um sistema de reduzida dimensão. Finalmente, a secção 7 aponta algumas conclusões assim como algumas orientações para trabalho futuro.

2. O Projecto IVY

O projecto IVY surge na sequência do desenvolvimento da ferramenta I2SMV [CH01], a qual permite verificar modelos de sistemas interactivos através do *model checker* SMV [McM93]. O objectivo do projecto consiste no desenvolvimento de uma ferramenta para a análise do comportamento dos sistemas interactivos na fase de desenho. Pretende-se que a ferramenta dê suporte ao processo de modelação e análise através de editores para modelos e propriedades, e visualizadores para a análise dos resultados obtidos no processo de verificação (cf. Figura 1).

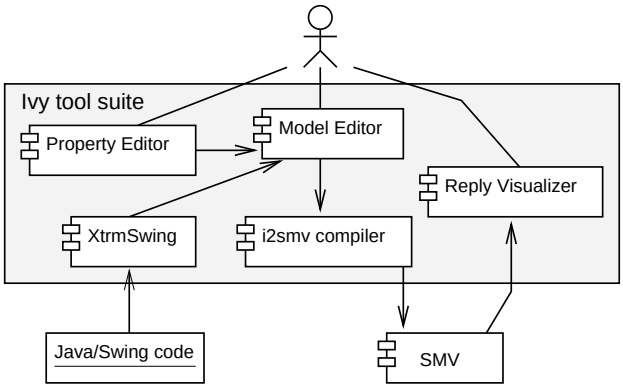

Figura 1. IVY **architecture**

Através do projecto IVY pretende-se criar uma abstracção na qual os modelos dos sistemas interactivos possam ser mais facilmente desenvolvidos e analisados. Encontramo-nos presentemente a explorar o uso de técnicas de engenharia reversa para permitir a geração de modelos a partir do código fonte. Deste modo, pretende-se analisar as aplicações já existentes bem como dar suporte ao processo de re-engenharia aquando de uma manutenção ou migração. Neste último caso, pretende-se garantir que o novo sistema tem as mesmas características do que o anterior.

3. Trabalhos Alternativos

Nos últimos anos, vários autores investigaram a análise da usabilidade dos sistemas interactivos utilizando modelos e técnicas formais de raciocínio. O uso de engenharia reversa tem sido explorada no sentido de extrair tais modelos directamente a partir dos sistemas interactivos legados. Várias técnicas têm sido apresentadas. Uma abordagem típica consiste em executar o sistema interactivo e automaticamente armazenar os seus estados e acções. Por exemplo, Memon [MBN03a] descreve uma ferramenta que extrai automaticamente dados relacionados com os componentes gráficos, suas propriedades e valores. Chen [CS01] propõe uma técnica baseada em especificações para testar interfaces. Os utilizadores manipulam especificações de teste representadas por máquinas de estados finitas as quais são obtidas através da execução do sistema. Por sua vez Systa estuda e analisa o comportamento em tempo de execução de código Java através de um processo de engenharia reversa. A partir da execução do software sobre um *debugger*, Systa consegue gerar um digrama de estados, o qual é utilizado para analisar o comportamento de uma classe, um objecto ou um método [Sys01].

Uma alternativa que surge no âmbito da engenharia reversa de sistemas interactivos é a análise estática. O pro-

[1] http://www.di.uminho.pt/IVY

cesso baseia-se na análise do código de uma aplicação em vez da sua execução, tal como acontece no caso anterior. Por exemplo, d'Ausbourg [dDR96] investigou a análise estática do sistema de janelas X11 [HF94]. Nesse trabalho, os modelos salientam os eventos que podem surgir para um dado componente da interface, como por exemplo pressionar uma tecla.

Moore [Moo96] descreve uma técnica para automatizar parcialmente o processo de engenharia reversa de sistemas interactivos. O resultado deste processo é um modelo para testar a funcionalidade das interfaces gráficas. O trabalho desenvolvido permite, com base num conjunto de regras, detectar componentes interactivos de código legado. Merlo [MGG+95] propõe uma abordagem similar. Em ambos os casos é utilizada uma análise estática.

A nossa abodagem baseia-se em análise estática tal como [dDR96, Moo96, MGG+95]. No entanto, ao contrário de [MGG+95] ou [Moo96], que trabalhavam sobre interfaces por caracteres, nós pretendemos fazer engenharia reversa de interfaces gráficas. Tal como já foi afirmado, neste momento estamos a utilizar código Java/Swing como caso de estudo. No entanto, o objectivo de médio prazo é desenvolver uma abordagem tanto quanto possível genérica e independente da linguagem. Adicionalmente, o nosso objectivo é obter modelos que reflictam o diálogo gerado pela interface, e não simplesmente a arquitectura do código que a implementa. Assim, os modelos que pretendemos gerar estão a um nível de abstracção mais elevado do que aqueles gerados por [MBN03a] ou [dDR96].

4. Técnica de Engenharia Reversa de Sistemas Interactivos

As contribuições deste trabalho consistem, então, num modelo e numa arquitectura aplicacional que suportem a engenharia reversa de sistemas interactivos a partir da análise estática do código dos sistemas. Tal como já afirmado, o processo de reengenharia deverá permitir gerar modelos do diálogo induzido pelas interfaces implementadas por esse código.

A técnica apresentada nesta secção permite construir uma abstracção da interface de um qualquer código *Java* legado. Este processo identifica os dados e acções envolvidas na interface, bem como as interacções entre os vários componentes da interface. Estes componentes incluem objectos e acções da interface. Para construir uma abstracção da interface, a partir da sua implementação em JAVA, utilizamos uma combinação de várias técnicas, nomeadamente: programação estratégica, slicing de programas e modelos abstractos. O objectivo é detectar componentes na interface através de estratégias funcionais e modelos com semântica formal.

4.1. O Processo

O protótipo especificado neste trabalho (cf. Figura 2) tem como ponto de partida código fonte JAVA2/SWING de sistemas interactivos desenvolvidos através do ambiente integrado de desenvolvimento NETBEANS. Tal como referido na secção 1, existem três objectivos essenciais que o protótipo deve satisfazer:

1. Engenharia reversa do código para um nível adequado de abstracção;
2. Modelação de interacção;
3. Modelação de interfaces com elevado grau de dinamismo.

A estrutura deste protótipo é composta por vários passos (cf. Figura 2). Foram utilizadas várias funcionalidades disponibilizadas pela UMINHOHASKELLLIBRARY[2], de entre as quais destancam-se as seguintes:

- Gramática no formato SDF da linguagem JAVA2 e geração de um *parser* para a linguagem;
- Geração automática de um conjunto de tipos de dados abstractos em HASKELL [JHA+99] para a representação de código JAVA2/SWING segundo uma árvore abstracta de sintaxe;
- Extracção do código SWING através de técnicas de slicing construidas utilizando programação estratégica para efectuar travessias da árvore original [Vis03].

A aplicação de algumas funcionalidades da UMINHOHASKELLLIBRARY permite, deste modo, obter uma plataforma funcional para a extracção e manipulação de quaisquer expressões contidas em código JAVA2/SWING. Sobre este modelo, foi desenvolvido um protótipo o qual permite inverter código JAVA2/SWING para um nível adequado de abstracção.

4.2. Slice de Interfaces

Tal como apontado na secção anterior, para extrair um modelo de uma interface a partir do código é necessário construir uma função de *slicing* de modo a isolar as instruções relacionadas com a interface. A abordagem habitual consiste em escrever uma função recursiva para executar uma travessia da árvore abstracta de sintaxe (AST) de um programa JAVA, obtendo como resultado uma sub-árvore com as instruções da interface. Para simplificar o desenvolvimento das funções de travessia, utilizamos uma abordagem diferente, isto é, aplicamos programação estratégica. Neste estilo de programação, existe um conjunto pre-definido de funções genéricas para a travessia de qualquer AST incorporando várias estratégias de travessias possíveis (i.e. *top-down*, *left-to-right*, etc). Estas funções permitem ter em conta somente os aspectos de interesse, permitindo assim considerar unicamente as partes relevantes (no nosso caso a linguagem SWING).

[2] Conjunto de bibliotecas desenvolvidas na linguagem de programação HASKELL pelo Departamento de Informática da Universidade do Minho

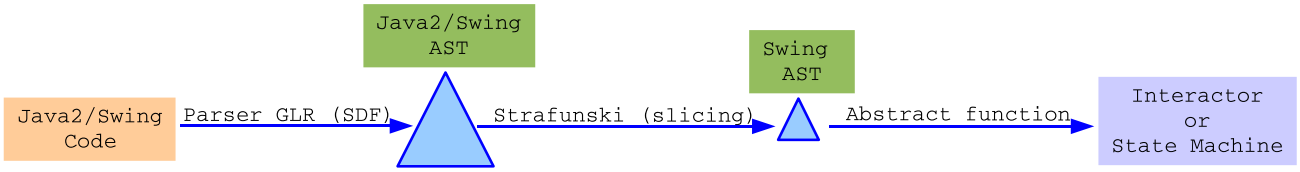

Figura 2. O processo de engenharia reversa

4.3. A Arquitectura Aplicacional

O protótipo foi desenvolvido na linguagem de programação funcional HASKELL [JHA+99] e através da plataforma descrita no ponto anterior, permite realizar *slicing* [Tip95] de qualquer árvore representando código JAVA2/SWING, possibilitando assim extrair sub-partes da árvore.

O protótipo extrai e manipula em primeiro lugar o método INITCOMPONENTS o qual define todos os componentes gráficos da aplicação.

A seguir protótipo extrai do respectivo método toda a informação relacionada com a parte gráfica da aplicação em estudo. Por exemplo objectos das classes *JButton*, *JLabel*, *JTextField*, *JProgressBar*, *JComboBox*, *JSlider*, etc. Para cada tipo de objecto extrai-se somente a informação relevante para este processo de engenharia reversa.

A título de exemplo, descreve-se brevemente como se pode efectuar o slicing em *Strafunski* de sub-expressões SWING de um qualquer programa JAVA para o caso dos objectos definidos através da classe *JButton*. Pretende-se extrair da árvore somente as instruções responsáveis por criar esses objectos bem como todas aquelas que invocam estes métodos:

1. Instâncias da classe *JButton*: As instâncias da classe *JButton* implementam-se via instruções com o seguinte padrão:

   $$VarObjName = new\ javax.swing.JButton();$$

   A partir da árvore, extrai-se então todas as expressões que satisfazem este padrão e guarda-se para cada expressão o nome da variável utilizada.

   ```
   getJButtons :: (Term t) => t -> [[Id]]
   getJButtons types = jButton
     where
     assigments = getAssig types
     jButton = [a | (a,b) <- assigments,
       (b==["javax","swing","JButton"])]
   ```

2. Método *setText* das instâncias da classe *JButton*: A invocação do método *setText* em objectos definidos com a classe *JButton* implementa-se via instruções com o seguinte padrão:

   $$VarObjName.setText(text)$$

   A partir da árvore, extrai-se então todas as expressões que satisfazem este padrão e para cada expressão guarda-se o nome da variável utilizada bem como o método invocado e o texto colocado sobre o objecto.

   ```
   getJButtonsSetText :: (Term t) =>
     [[Identifier]] -> t -> [([Id],[Id])]
   getJButtonsSetText vjid types = jBText
     where
     nmi = getNamesMethodsInvocation types
     jBText = [(a,["setText"]++(getStr d))
               | a <- vjid, (c,d) <- nmi,
               (a++["setText"]) == c]
   ```

3. Método *addActionListener* dos objectos das instâncias da classe JBUTTON: A invocação do método *addActionListener* em objectos definidos com a classe *JButton* implementa-se via instruções com o seguinte padrão:

   $$VarObjName.addActionListener(expression)$$

   A partir da árvore, extrai-se então todas as expressões que satisfazem este padrão.

   ```
   getAddActionListener :: (Term t) =>
     [[Id]] -> t -> [([Id],[Id])]
   getAddActionListener vjid types = jba
     where
     nmi = getNMI types
     jba = [(a,["addActionListener"]++
          (concat (map fst (getNMI d))))
          | a <- vjid, (c,d) <- nmi,
          (a++["addActionListener"]) == c]
   ```

   A seguir determina-se o fecho transitivo de todos os métodos invocados a partir do método ADDACTIONLISTENER. Com base no fecho obtido, determina-se assim recursivamente todos os objectos gráficos com os quais o botão está relacionado.

   ```
   getTMB ::(Term t)=>
    [(Header,Body)]->t->[(Header,Body)]
   getTMB t tall =
    t++(concat (exp2++exp3))
    where
    nmi = getNMI t
    exp2 = [getMethod a tall | (a,b) <- nmi]
    exp3 = [getTMB [b] tall | b <- exp2]
   ```

5. Relacionando todos os Objectos Gráficos

Após extracção dos dados dos objectos gráficos assim como todos os dados dos métodos sobre estes executados, é então possível gerar representações abstractas das interfaces. Nesse sentido, o protótipo permite criar duas abstracções diferentes: uma máquina de estados e um interactor.

5.1. A máquina de Estados

A máquina de estados permite abstrair o comportamento dos sistemas interactivos. Com esta representação pretende-se abstrair todos os estados gráficos de um sistema interactivo bem como as acções que interligam esses estados. A representação consiste num grafo. Nesse sentido optou-se por utilizar a ferramenta GRAPHVIZ[3] para a geração do grafo. Em GRAPHVIZ, os grafos são elaborados com base numa linguagem própria pelo que foi desenvolvido um módulo auxiliar responsável por produzir o código necessário na linguagem GRAPHVIZ.

Toda a informação traduzida para a linguagem GRAPHVIZ é obtida com base nos resultados dos *slices* da secção anterior. Assim, cada estado gráfico deduzido da árvore dá origem a um estado gráfico na máquina de estados e cada acção sobre a aplicação dá origem a uma transição entre dois estados. Acrescentam-se também na máquina outros dados específicos do código fonte como por exemplo a inicialização de valores, as condições associadas às acções, etc.

5.2. A Linguagem dos Interactors

Os interactors, tais como desenvolvidos em [DH93], são um mecanismo de estruturação de modelos de sistemas interactivos e auxiliam a aplicação de linguagens de especificação de âmbito genérico à modelação desses mesmos sistemas. Os interactors não prescrevem uma linguagem de especificação. Em vez disso, propõem uma estruturação dos modelos que é adequada à modelação de sistemas interactivos e independente da linguagem de especificação utilizada. Utilizando interactors, os modelos são estruturados recorrendo à noção de um objecto que é capaz de apresentar parte do seu estado ao exterior. Assim, cada interactor possui um estado (definido como um conjunto de atributos), um conjunto de eventos a que pode responder ou que pode originar (definido como um conjunto de acções) e uma relação de apresentação que define quais os atributos/acções que são apresentados ao utilizador (marcados com [vis]).

No nosso caso particular, o comportamento dos interactors é definido utilizando uma Lógica Modal de Acções (MAL - Modal Action Logic). Em MAL, temos quatro tipos base de axiomas para definir comportamento:

- axiomas modais permitem definir o efeito das acções no estado do interactor, por exemplo, o axioma $[add\_action]\ consult' = true\ keep(numero, nota)$ expressa o facto de que, depois da acção add_action, o valor do atributo $consult$ passa a ser $true$. As plicas são utilizadas para referir o valor de um atributo no estado após a ocorrência da acção (o valor de atributos sem plica é calculado no estado anterior à ocorrência da acção). O operador $keep$ é utilizado para indicar que os valores dos atributos passados como parâmetros não mudam.

- axiomas de permissão permitem definir, para cada acção, em que condições ela é permitida, por exemplo, o axioma $per(add) \rightarrow \neg total$ expressa o facto de que a acção add pode ocorrer apenas quando o atributo $total$ tem o valor falso.

- axiomas de obrigação permitem definir que, em determinadas condições, uma determinada acção tem obrigatoriamente que ocorrer, por exemplo, o axioma $state = open \rightarrow obl(add)$ expressa o facto de que quando o valor do atributo $state$ é $open$ então a acção add tem que ocorrer algures no futuro. Note-se que o axioma não força a acção a ocorrer imediatamente, mas sim que deve ocorrer eventualmente.

- axiomas de inicialização permitem definir o estado inicial do interactor, por exemplo, o axioma $[]add = false, consult = true$ define os valores dos diferentes atributos no estado inicial do modelo.

O modelo e a arquitectura aqui apresentados definem assim uma abordagem no contexto da engenharia reversa de sistemas interactivos desenvolvidos em JAVA via NETBEANS IDE. A seguir pretende-se descrever a aplicação desta metodologia através de um caso de estudo concreto.

6. Caso de Estudo

A experimentação do modelo e da arquitectura aplicacional sobre um pequeno sistema interactivo desenvolvido com o ambiente integrado de desenvolvimento NETBEANS em Java2/Swing permite validar e demonstrar, na prática, que o modelo e a arquitectura aplicacional descritos na secção anterior suportam efectivamente a engenharia reversa de sistemas interactivos desenvolvidos através do NETBEANS.

6.1. Descrição da Aplicação JTURMA

A aplicação em estudo designa-se por JTURMA (cf. Figura 3). Por este ser demasiado extenso, omitimos aqui o código JAVA/SWING da aplicação e apresentamos apenas a interface gráfica que ele implementa.

Esta aplicação permite armazenar e manipular as classificações teóricas e práticas de uma turma. O sistema disponibiliza uma funcionalidade que permite registar os números, os nomes, as classificações teóricas e as classificações práticas de todos os alunos da turma. Para além do armazenamento dos dados, encontram-se disponiveis funções para consultar ou remover os dados de um aluno bem como uma função para terminar a aplicação. No desenvolvimento desta aplicação foram utilizados vários

[3] http://www.research.att.com/ erg/graphviz

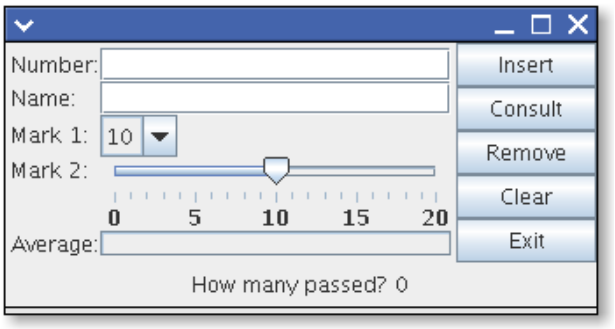

Figura 3. Aplicação *JTurma*

objectos gráficos JAVA2/SWING, nomeadamente objectos das classes JBUTTON, JTEXTFIELD, JLABEL, JPROGRESSBAR, JCOMBOBOX, JSLIDER, JPANEL e GETCONTENTPANE.

A execução do protótipo sobre o sistema JTURMA permite gerar de forma automática uma máquina de estados e um interactor. Considerando a Figura 2, quer o interactor quer a máquina de estados correspondem ao resultado do último passo no referido processo de engenharia reversa de sistemas interactivos. A partir da sub-árvore com as expressões SWING e aplicando um conjunto de funções de abstracção, gera-se uma máquina de estados ou um interactor.

6.2. A Máquina de Estados

Na figura 4 apresenta-se parte da máquina de estados gerado pelo protótipo quando executado sobre a aplicação JTURMA.

Nesta máquina de estados, cada estado descreve o conteúdo de uma janela da aplicação num determinado instante. A figura 4 permite-nos visualizar o comportamento da aplicação JTURMA à volta de dois estados gráficos. Assim a máquina de estados especifica dois estados (conteúdos da janela em dois tempos distintos): à esquerda um estado gráfico com os atributos *consult* e *remove* iguais a *true* e à direita um segundo estado gráfico com os referidos atributos iguais a *false*.

Estes dois estados gráficos encontram-se relacionados por transições, as quais simbolizam as acções que podem ser executadas sobre um determinado estado. A cada acção encontra-se associada uma sequência de condições (entre parêntesis rectos). Estas devem ser satisfeitas para se poder executar a transição relativa à acção. Por exemplo, por análise da máquina de estado apresentada, deduz-se que a remoção de um aluno (a partir do estado esquerdo) implica transitar para o outro estado caso o número de alunos após a remoção seja igual a zero (*this.turmaquantos()==0*), caso contrário a remoção retorna o mesmo estado gráfico.

A partir da máquina de estados pretende-se aprofundar a análise do comportamento de sistemas interactivos. Por exemplo, calculando automaticamente todas as frases possíveis (conjunto das sequências de transições da máquina de estados) na interface (até um tamanho dado), torna-se possível verificar até que ponto a interface respeita ou não um dado modelo de tarefas.

6.3. O Interactor

Para além da máquina de estados, o protótipo pode também construir um interactor que captura a informação e acções presentes na interface, bem como o seu comportamento em resposta a acções do utilizador. Para o exemplo apresentado, o interactor contém um conjunto de atributos:

```
interactor JClass
attributes
number,
name: String
mark1,
mark2,
average: Integer
addEnabled,
consultEnabled,
removeEnabled,
clearEnabled,
exitEnabled: Boolean
```

um para cada componente gráfico capaz de apresentar ou aceitar infromação presente na interface, e um por cada um dos dos botões para representar o seu estado. Os nomes dos atributos são retirados dos nomes das variáveis no código relativas aos componentes gráficos respectivos.

O interactor contém também um conjunto de acções:

```
actions
  add,
  open,
  close,
  consult,
  remove,
  clear,
  exit,
  setText_name(String),
  setSelectedItem_mark2(Integer),
  setValue_mark1(Integer),
  setValue_average(Integer),
  setText_number(Integer)
```

uma por cada botão, e uma por cada componente gráfico de *input*.

E finalmente um conjunto de axiomas:

```
[] number="" & name="" &
 mark1=10 & mark2=10 & average=0
[] addEnabled=true & clearEnabled=true &
 exitEnabled=true & consultEnabled=false &
 removeEnabled=false & number="" &
 name="" & mark1=10 & mark2=10 & average=0
[add] number'=number & name'=name &
 mark1'=mark1 & mark2'=mark2 &
 average'=average & consultEnabled'=true &
 removeEnabled'=true &
 addEnabled'=addEnabled &
 clearEnabled'=clearEnabled &
 exitEnabled'=exitEnabled
[consult] number'=number & name'=?ref1? &
```

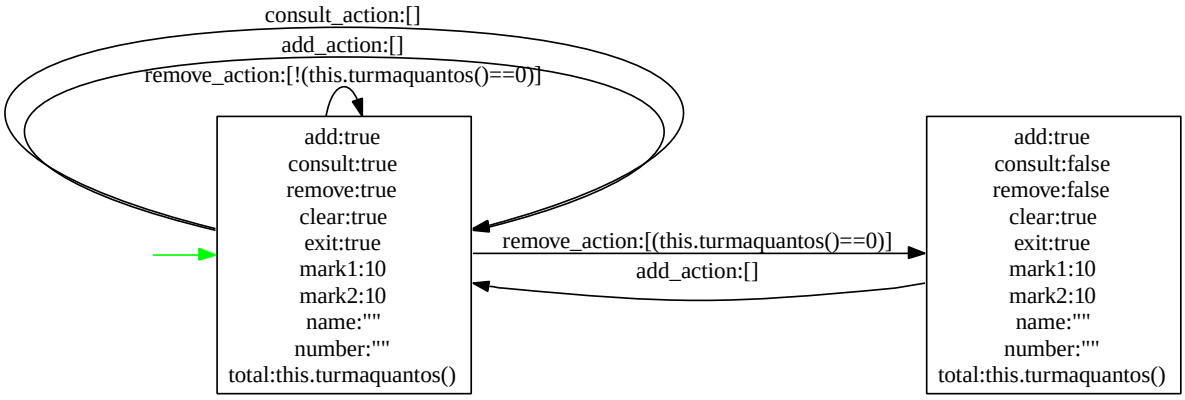

Figura 4. Parte da máquina de estados associada à aplicação *JTurma*

```
 mark1'=?ref2? & mark2'=?ref3? &
 average'=?ref4? & addEnabled'=addEnabled &
 consultEnabled'=consultEnabled &
 removeEnabled'=removeEnabled &
 clearEnabled'=clearEnabled &
 exitEnabled'=exitEnabled
[remove] number'=number name'=name &
 mark1'=mark1 & mark2'=mark2 &
 average'=average & addEnabled'=addEnabled &
 clearEnabled'=clearEnabled &
 exitEnabled'=exitEnabled
[clear] number'=?ref5? & name'=?ref6? &
 mark1'=?ref7? & mark2'=?ref8? &
 average'=?ref9? & addEnabled'=addEnabled &
 consultEnabled'=consultEnabled &
 removeEnabled'=removeEnabled &
 clearEnabled'=clearEnabled &
 exitEnabled'=exitEnabled
[setText_name(a)] name'=a & number'=number &
 mark1'=mark1 & mark2'=mark2 &
 average'=average &
 consultEnabled'=consultEnabled &
 removeEnabled'=removeEnabled &
 addEnabled'=addEnabled &
 clearEnabled'=clearEnabled &
 exitEnabled'=exitEnabled
...
```

Os dois primeiros axiomas definem o estado inicial do sistema. Os quatro seguintes definem o efeito dos botões na interface. As expressões `?refX?` representam valores que devem ser preenchidos através do editor IVY. Para apoiar o preenchimento do modelo, cada expressão é um apontador para o código JAVA que define o valor a atribuir. O último axioma define o efeito das acções de *input* na caixa de texto $name$.

Ainda que incompleto, este interactor incluí dados relevantes no que diz respeito ao comportamento do sistema JTURMA. A título de exemplo, o quarto axioma, exprime o estado interactivo após executar a acção *consult*. Podemos constatar que os atributos *number*, *addEnabled*, *consultEnabled*, *removeEnabled*, *clearEnabled*, *exitEnabled* mantêm-se inalterados. Por outro lado, os atributos *name*, *mark1*, *mark2* e *average* recebem novos dados.

Após instanciar totalmente o modelo, este poderá ser usado pela ferramenta IVY para verificação do seu comportamento.

Através dos resultados obtidos, demonstra-se que a abordagem descrita na secção anterior, ainda que actualmente com limitações ao nível do tipo de objectos gráficos que consegue tratar, suporta a engenharia reversa de sistemas interactivos desenvolvidos em JAVA/SWING.

7. Conclusão

Neste artigo, foi descrita a forma como a programação estratégica e técnicas de *slicing* podem ser aplicados à engenharia reversa de interfaces a partir do código das aplicações. O resultados deste trabalho evidenciam a possibilidade de inverter interfaces directamente a partir do seu código. Um protótipo foi desenvolvido permitindo extrair automaticamente o comportamento de uma aplicação à partir do seu código fonte.

Actualmente a ferramenta permite extrair um sub-conjunto de componentes gráficos e de acções, a partir do qual gera quer uma máquina de estados quer um interactor. Estes modelos permitem então raciocinar acerca de aspectos da usabilidade e da qualidade da implementação do sistema em análise. Neste momento, o protótipo considera somente um conjunto limitado dos componentes SWING. Como trabalho futuro, pretende-se extender a implementação de modo a contemplar interfaces mais complexos.

8. Agradecimentos

Este trabalho é parcialmente suportado pela FCT (Portugal) e FEDER (União Europeia) através do contrato POSC/EIA/56646/2004.

Referências

[BA95] Peter Bumbulis and P.S C. Alencar. A framework for prototyping and mechaniacally verifying a class of user interfaces. *IEEE*, 1995.

[Boe88] B. Boehm. *A Spiral Model for Software Development and Enhancement*, volume 21, pages 61–72. Computer edition, Maio 1988.

[Cam04] José C. Campos. The modelling gap between software engineering and human-computer interaction. In Rick Kazman, Len Bass, and Bonnie John, editors, *ICSE 2004 Workshop: Bridging the Gaps II*, pages 54–61. The IEE, 2004.

[CH01] José C. Campos and Michael D. Harrison. Model checking interactor specifications. *Automated Software Engineering*, August 2001.

[CH03] J. C. Campos and M. D. Harrison. From HCI to Software Engineering and back. In Rick Kazman, Len Bass, and Jan Bosch, editors, *Bridging the Gaps Between Software Engineering and Human-Computer Interaction, ICSE '2003 workshop*, pages 49–56, Portland, Oregon, USA, May 2003. IFIP.

[CS01] J. Chen and S. Subramaniam. A gui environment for testing gui-based applications in java. *Proceedings of the 34th Hawaii Internacional Conferences on System Sciences*, january 2001.

[dDR96] Bruno d'Ausbourg, Guy Durrieu, and Pierre Roché. Deriving a formal model of an interactive system from its UIL description in order to verify and to test its behaviour. In *Design, Specification and Verification of Interactive Systems '96*, pages 105–122. 1996.

[DH93] D. J. Duke and M. D. Harrison. Abstract interaction objects. *Computer Graphics Forum*, 1993.

[dSGD98] Bruno dAusbourg, Christel Seguin, and Pierre Rochk Guy Durrieu. Helping the automated validation process of user interfaces systems. *IEEE*, 1998.

[eES80] B. Lientz e E. Swanson. *Software Maintenance Management*. Addison-wesley edition, 1980.

[ESS03] P. Iglinski E. Stroulia, M. El-ramly and P. Sorenson. User interface reverse engineering in support of interface migration to the web. *Automated Software Engineering*, 2003.

[HF94] Dan Heller and Paula M. Ferguson. *Motif Programming Manual*, volume 6A of *X Window System Seris*. O'Reilly & Associates, Inc., second edition, 1994.

[JHA+99] Simon Peyton Jones, John Hughes, Lennart Augustsson, et al. Report on the Programming Language Haskell 98. Technical report, February 1999.

[MBN03a] Atif Memon, Ishan Banerjee, and Adithya Nagarajan. Gui ripping: Reverse engineering of graphical user interface for testing. Technical report, 2003. Department of Computer Science and Fraunhofer Center for Experimental Software Engineering, Department of Computer Science, University of Maryland, USA.

[MBN03b] Atif Memon, Ishan Banerjee, and Adithya Nagarajan. GUI ripping: Reverse engineering of graphical user interfaces for testing. Technical report, Department of Computer Science and Fraunhofer Center for Experimental Software Engineering, Department of Computer Science University of Maryland,USA, 2003.

[McM93] Kenneth L. McMillan. Symbolic model checking. *Kluwer Academic Publishers*, 1993.

[MGG+95] Merlo, Gagne, Girard, Kontogiannis, Hendren, Panangaden, and De Mori. Reverse engineering and reengineering of user interfaces. *IEEE Software, 12(1), 64-73*, 1995.

[Moo96] M. M. Moore. Rule-based detection for reverse engineering user interfces. *Proceedings of the Third Working Conference on Reverse Engineering, pages 42-8, Monterey, CA*, november 1996.

[SCS06] João Carlos Silva, José Creissac Campos, and João Saraiva. Combining formal methods and funcional strategies regarding the reverse engineering of interactive applications. *DSVIS-2006, Design, Verification and Validation of Interactive Systems*, July 2006. (accepted for publication).

[Sys01] T. Systa. Dynamic reverse engineering of java software. Technical report, 2001. University of Tampere, Finland.

[Tip95] Frank Tip. A survey of program slicing techniques. *Journal of Programming Languages*, september 1995.

[Vis03] Joost Visser. *Generic Traversal over Typed Source Code Representations*. PhD thesis, University of Amsterdam, February 2003.