Análise de usabilidade baseada em modelos

José Creissac Campos
Departamento de Informática
Universidade do Minho
Braga
jose.campos@di.uminho.pt

Sumário

A norma ISO DIS 9241-11 define usabilidade de um sistema como a eficácia, eficiência e satisfação com que utilizadores determinados atingem objectivos determinados em ambientes específicos. A análise de usabilidade de um sistema deve então ter em consideração os utilizadores e o contexto de utilização. Isto coloca problemas pois tipicamente os engenheiros de software não estão motivados, nem tem os conhecimentos necessários, para analisarem o sistema desta perspectiva. Neste artigo apresenta-se a arquitectura de uma ferramenta que suporta uma abordagem ao desenvolvimento de sistemas interactivos em que se procura facilitar a comunicação entre as comunidades da Interacção Humano-Computador e da Engenharia de Software.

Palavras-chave

Usabilidade, verificação de modelos (model-checking).

1. INTRODUÇÃO

No desenvolvimento de sistemas interactivos cruzam-se as áreas da Interacção Humano-Computador (IHC) e da Engenharia do Software. Estudos têm mostrado que o sucesso de tais sistemas depende, em grande medida, da sua usabilidade. *Usabilidade* é definida em [Rosson02] como a qualidade de um sistema relativamente à facilidade de aprendizagem, à facilidade de utilização e à satisfação dos seus utilizadores. Trata-se portanto de uma medida de qualidade de um sistema interactivo em que o factor humano é fundamental.

A norma ISO DIS 9241-11 identifica mais claramente os factores relevantes para a usabilidade de um sistema ao defini-la como a eficácia, eficiência e satisfação com que utilizadores determinados atingem objectivos determinados em ambientes específicos. Eficácia tem a ver com a possibilidade (ou não) de o utilizador poder atingir os seus objectivos utilizando o sistema num dado contexto. Eficiência tem a ver com o maior ou menor esforço que o utilizador terá que despender para atingir esse objectivo. Satisfação é uma medida subjectiva do grau de agradabilidade na utilização do sistema.

A análise da qualidade de um sistema interactivo, no que diz respeito à sua usabilidade, tem portanto que ter em consideração tanto os utilizadores do sistema como o contexto em que este é utilizado. Isto coloca problemas pois tipicamente os engenheiros de software não estão motivados, nem tem os conhecimentos necessários, para analisarem o sistema desta perspectiva. O *Software Engineering Book of Knowledge* [SWEBOK01], por exemplo, considera a concepção de interfaces com o utilizador como uma área relacionada (mas distinta) da engenharia de *software*, não fazendo menção à área da IHC. Torna-se, portanto, necessário recorrer a peritos na área da IHC para ter em consideração factores de usabilidade. Na prática, no entanto, as duas áreas tem vivido relativamente afastadas e os métodos e técnicas de uma não são conhecidos de forma generalizada na outra.

Neste artigo apresenta-se a arquitectura de uma ferramenta que suporta uma abordagem ao desenvolvimento de sistemas interactivos em que se procura facilitar a comunicação entre as duas comunidades. A 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.

2. ANÁLISE DE USABILIDADE BASEADA EM MODELOS

Apesar do reconhecimento de que a análise de usabilidade deve começar tão cedo quanto possível no processo de desenvolvimento, a verdade é que a separação entre HCI e Engenharia de Software não facilita esse objectivo. Torna-se necessário estabelecer mecanismos de comunicação entre as duas áreas. Neste contexto a utilização de modelos tem vindo a ser explorada. Estes modelos deverão ser desenvolvidos de um ponto de vista da interacção entre o sistema, os seus utilizadores e o contexto que os rodeia e podem ser utilizados nas fases iniciais de desenvolvimento para ajudar a definir o sistema que deverá ser implementado.

O grau de formalidade dos modelos utilizados varia desde simples esquiços do aspecto gráfico da interface. até modelos matemáticos do seu comportamento. A utilização de modelos matemáticos possibilita a análise rigorosa das propriedades dos mesmos. Isto torna-se particularmente relevante quando se pretende analisar o comportamento de sistemas interactivos complexos e/ou de risco e quando se pretende garantir a confiabilidade de tais sistemas. Neste contexto, a utilização de técnicas de

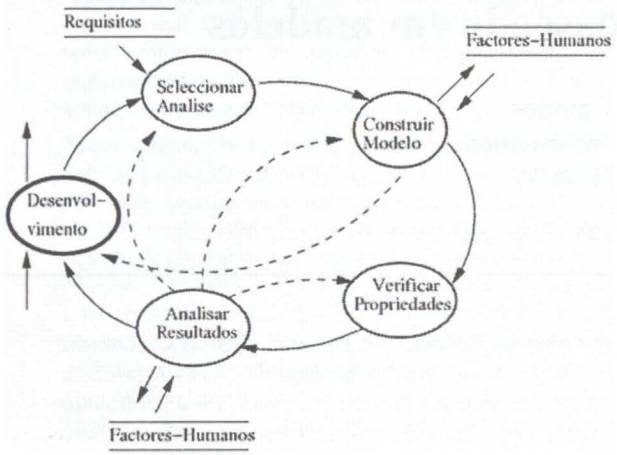

Figura 1 - Processo de verificação

model-checking tem vindo a ser investigada desde há já alguns anos [Paterno95, Campos98].

Em [Campos99, Campos01] é apresentada uma abordagem à análise de modelos interactivos baseada no utilização do *model-checker* SMV [McMillan93] para a análise de modelos expressos em Modal-Action Logic (MAL) [Ryan91]. Os modelos são estruturados utilizando a noção de *interactor* de [Duke93]. Para uma descrição da linguagem de modelação utilizada consultar [Campos01]. De forma sumária, no contexto deste trabalho, um interactor consiste em (ver figuras 2, 5 ou 7): um conjunto de atributos tipados que definem o estado do *interactor*; um conjunto de acções que definem alterações ao estado do *interactor*; e um conjunto de axiomas escritos em MAL que definem a semântica das acções em termos do seu efeito no estado do *interactor*.

A interface fornecida pelo *interactor* é definida à custa de anotações que definem qual a modalidade através da qual os atributos/acções podem ser acedidos (por exemplo, a modalidade vis aplicada a um atributo indica que este é visível para o utilizador). Estão ainda definidos mecanismos de composição (cláusula *including*) e de reutilização (cláusula *imports*) de *interactors*.

A figura 1 apresenta a abordagem proposta. Em cada momento do processo de concepção do sistema interactivo é possível desencadear um processo de análise. Tal ficar-se-á normalmente a dever a dúvidas sobre se algum aspecto da interface obedece aos requisitos e/ou princípios de usabilidade definidos para o sistema. Depois de identificado o aspecto a analisar é desenvolvido um modelo do sistema e são identificadas as propriedades relevantes que o modelo deverá possuir recorrendo à ajuda de peritos de IHC. Procede-se então à análise do modelo e os resultados desse processo são então analisadas com a ajuda de peritos em IHC tendo em vista validar os resultados da análise.

O objectivo primordial da abordagem é facilitar a integração da análise de questões relacionadas com usabilidade no processo de desenvolvimento. Por outro lado, os papeis e responsabilidades tanto dos engenheiros de software como dos peritos em usabilidade ficam

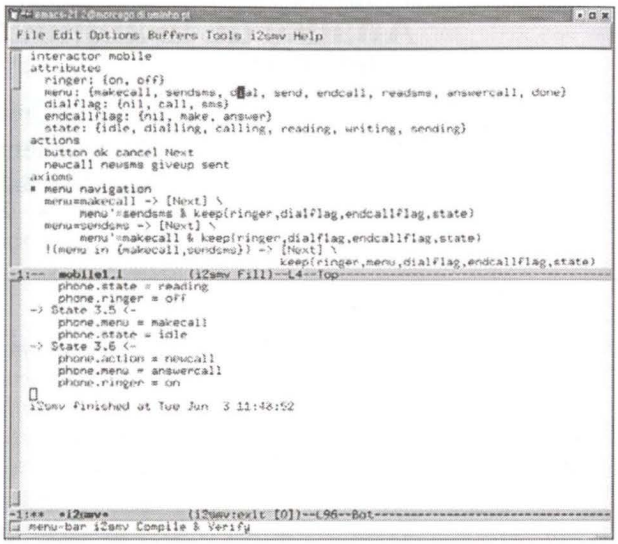

Figura 2 - Versão inicial do ambiente

claramente identificados. Para uma descrição mais detalhada da abordagem consultar [Campos99].

A abordagem tem vindo a ser testada através da aplicação a diversos casos de estudo. Uma crítica comum entre a comunidade IHC tem a ver com a complexidade dos modelos utilizados. Tal deve-se, em grande medida, ao tipo de modelos aqui utilizado não ser comum naquela área. Por sua vez, na comunidade da Engenharia de Software não é muitas vezes compreendido que os modelos se referem apenas à componente de interacção entre o sistema, os utilizadores e o contexto em que interagem. Isto leva a que seja criticado o *exagerado* nível de abstracção a que os modelos são expressos. Torna-se claro que, para este tipo de abordagem poder ter aceitação, é necessário fornecer suporte ao processo de modelação e análise através do desenvolvimento de ferramentas adequadas, por forma a que a modelação seja facilitada e os objectivos do processo clarificados (cf. [Loer04]).

Em [Campos99] é apresentado um primeiro protótipo de uma ferramenta para a modelação e análise de sistemas interactivos. Esta permite a compilação dos modelos MAL para a linguagem do SMV e fornece já algum suporte à edição e análise dos modelos. Neste artigo é apresentada a arquitectura de uma ferramenta que fornece um maior suporte a todo o processo anteriormente descrito.

3. VERSÃO ORIGINAL DA FERRAMENTA

O protótipo inicialmente desenvolvido em [Campos99] era basicamente um compilador da linguagem de modelação para a linguagem de descrição de máquinas de estado do SMV. Um suporte mínimo ao processo de modelação e análise foi fornecido na altura através do desenvolvimento de um modo para o editor Emacs.

O ambiente fornecido é o apresentado na figura 2. A janela aparece dividida em duas áreas principais: na área superior podem ser editados o modelo e as propriedades a verificar; na área inferior é apresentado o resultado do processo de verificação realizado com recurso ao SMV.

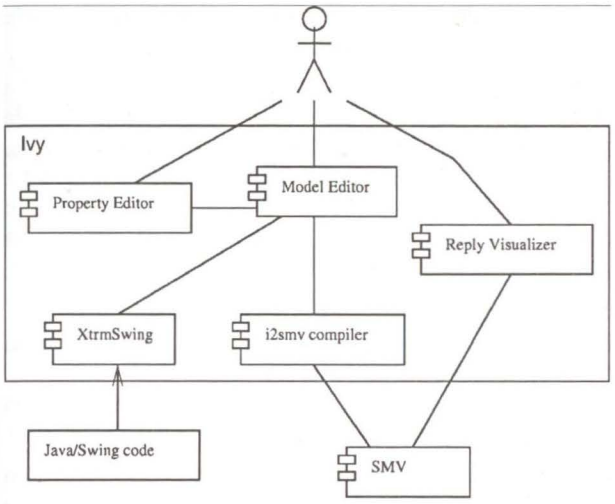

Figura 3 - Arquitectura da ferramenta IVY

Na barra de menus foi adicionado um novo menu que permite aceder às funcionalidades específicas do i2smv: compilação do modelo e verificação do modelo.

Este protótipo permitiu a modelação e análise dos diferentes casos de estudo que foram sendo levados a cabo, mas não fornece verdadeiro suporte ao processo. Em particular fornece pouco suporte ao desenvolvimento e edição dos modelos (permite apenas a identificação dos diferentes componentes do modelo através de *syntax highlighting),* à expressão das propriedades a verificar (as propriedades têm que ser expressas directamente em CTL [Clarke86], a lógica utilizada pelo SMV para expressão de propriedades sobre o comportamento da máquina de estados) e à análise dos resultados do processo de verificação (o que é apresentado é o resultado produzido pelo SMV, isto implica que seja necessário conhecimento sobre o processo de compilação para a compreensão dos resultados apresentados).

Tendo em vista resolver estes problemas, foi iniciado o desenvolvimento de uma nova ferramenta. Esta deverá fornecer um ambiente integrado para a modelação e análise de aspectos comportamentais de sistemas interactivos.

4. A NOVA FERRAMENTA — O IVY

A nova ferramenta toma o nome de IVY e deverá ser capaz de suportar as diferentes actividades identificadas anteriormente. A sua arquitectura é apresentada na figura 3. A arquitectura base consiste nos seguintes componentes: um editor de modelos (model editor); um compilador (i2smv compiler); e uma ferramenta para visualização dos traços de comportamento produzidos pelo SMV (reply visualiser).

Seguindo as ideias de [Loer03] é adicionada à arquitectura do sistema uma ferramenta para a edição de propriedades (property editor). A principal responsabilidade deste componente prende-se com o auxílio à tradução de propriedades relevantes de um ponto de vista da usabilidade para fórmulas lógicas que possam ser verificadas pelo SMV.

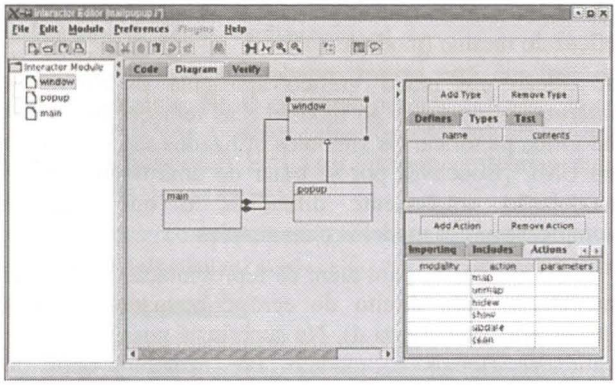

Figura 4 - Componente de edição (modo gráfico)

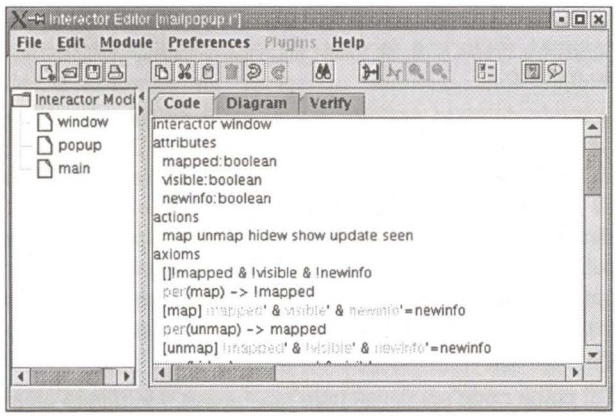

Figura 5 - Componente de edição (modo texto)

A aplicação do tipo de abordagem que aqui está a ser proposto ao desenvolvimento de novos sistemas é apenas umas das suas possibilidades de utilização. Irão sempre acontecer situações em que existirão problemas com software já desenvolvido. Nestas situações a utilização de engenharia reversa pode contribuir para a resolução de tais problemas. Tendo em vista explorar este aspecto, foi acrescentado um último componente à arquitectura IVY: um componente para engenharia reversa (XtrmSwing).

Apresentam-se de seguida cada um dos componentes em mais detalhe.

4.1 Componente de edição

Este componente é responsável pela prestação de auxílio no desenvolvimento dos modelos e fornece dois modos de edição: um modo gráfico que permite manipular a estrutura geral do modelo bem como as características de cada um dos seus componentes (ver figura 4), favorecendo uma rápida compreensão da estrutura global do modelo; e um modo de texto em que é possível trabalhar directamente sobre o texto fonte do modelo (ver figura 5), favorecendo uma edição eficaz do mesmo. Adicionalmente, ambas os modos de edição disponibilizam também uma visão sumária do modelo, apresentada em forma de árvore.

Independentemente do modo de edição a ferramenta fornece as funcionalidades de edição habituais: *copy/ paste*, *undo/redo*, inserção de novos *interactors*, etc. A disponibilização dos dois modos de edição tem como objectivo permitir uma rápida compreensão da estrutura

global do modelo (modo gráfico) bem como uma edição eficaz do mesmo (modo texto);

A edição em modo gráfico apresenta os diferentes *interactors* presentes no modelo e as relações entre eles de forma pictórica. Os símbolos utilizados são inspirados no UML [Booch98] por se tratar de uma linguagem de modelação amplamente difundida, o que torna a compreensão dos modelos mais simples.

Em modo gráfico, para além da representação gráfica do modelo, o lado direito do ecrã é ocupado por dois *inspectores* (ver figura 4). No superior é possível definir entidades globais ao modelo. Por exemplo, tipos ou propriedades a testar. No inferior é possível editar o *interactor* seleccionado em cada momento. Aqui é possível adicionar, remover e alterar os atributos, acções, axiomas, *imports* e *includes* de cada um dos *interactors* do modelo.

O modo de texto permite, tal como já foi dito, trabalhar directamente sobre o texto do modelo. Deste modo, utilizadores mais experientes conseguem editar o modelo de forma mais rápida, procurando os aspectos a alterar directamente no texto e não nos campos disponibilizados pelos *inspectores*. Utilizadores menos experientes, por outro lado, poderão optar pelo modo gráfico, onde usufruem de um grau de auxílio e assistência à edição mais elevado.

Tal como pode ser constatado nas figuras 4 e 5, para além das *panes* 'Code' e 'Diagram' o editor fornece, neste momento, uma terceira intitulada 'Verify'. Esta, é utilizada para visualizar os resultados da verificação do modelo e será substituída pelo componente de visualização logo que este esteja finalizado.

4.2 Componente de compilação (i2smv)

Este componente é responsável pela compilação dos modelos desenvolvidos para a linguagem do *model-checker SMV*. O algoritmo de compilação a utilizar é o descrito em [Campos99, Campos01]. Existe já uma versão deste componente implementada em Perl [Wall96].

4.3 Componente de visualização

Este componente é responsável pela apresentação dos resultados do processo de verificação.

Quando uma dada propriedade não se verifica, o SMV procura fornecer um traço de comportamento que demonstre a falsidade da propriedade em questão. Este componente terá duas responsabilidade essenciais: pré-processar os traços produzidos pelo SMV por forma a torná-los consistente com a notação utilizada na escrita do modelo e fornecer animações visuais dos traços por forma a facilitar a sua compreensão.

Os traços produzidos fazem, tal como seria de esperar, referência às variáveis e estados existentes ao nível do código SMV. Uma vez que o processo de compilação para SMV introduz uma série de variáveis auxiliares, torna-se necessário reverter esse processo por forma a que as entidades referidas passem a ser as existentes ao nível do modelo original. Um exemplo típico será a eliminação do atributo 'action', substituindo-o pela indicação da acção nas transições entre estados.

Estes traços, no entanto, podem atingir tamanhos na ordem das dezenas de estados, dependendo da complexidade do modelo, e incluem, muitas vezes, ciclos. Através de uma representação visual procura-se facilitar a sua compreensão bem como a da sua relação com o modelo, por forma a tornar mais claro qual o problema que está a ser apontado e possíveis soluções.

4.4 Componente de edição de propriedades

Este componente é responsável pelo auxílio à definição das propriedades que deverão ser verificadas. Este auxílio é prestado a dois níveis: codificação das propriedades em CTL e compilação da propriedade para código SMV.

Relativamente ao primeiro aspecto, o desenvolvimento deste componente tem por base as ideias apresentadas em [Loer03]. É disponibilizada uma lista de padrões de propriedades do comportamento dos modelos (ao estilo de [Dwyer99]) e facilidades para a instanciação dos padrões com acções e/ou atributos do modelo em análise. Deste modo, será possível ao utilizador da ferramenta seleccionar e compor aquelas propriedades em que tem mais interesse. Para cada uma dos padrões existe uma codificação pré-definida em CTL. Desta forma, as fórmulas CTL são criadas de forma transparente. Tal como para o editor, existe sempre a possibilidade de decidir editar as fórmulas manualmente.

4.5 Componente de engenharia reversa

Este é um componente experimental em que se está a estudar a aplicação de engenharia reversa a código Java2/Swing. Um aspecto que interessa referir prende-se com o nível de abstracção a que se pretende trabalhar. O objectivo não é obter um modelo da estrutura do código que implementa a interface. Quais as classes/objectos envolvidas(os) e suas relações. Em vez disso, o que se pretende é obter um modelo da interface que esse código implementa. Que informação está presente na interface, e que acções podem ser executadas sobre ela. Trata-se, portanto, se um salto de abstracção maior que o necessário para obter um modelo do código.

Considere-se por exemplo, a interface apresentada na figura 6. O que se pretende deduzir por análise do código, não é quais os objectos presentes na interface (campos de texto, botões, etc.) e como estão organizados, mas antes que informação está presente na interface (o número, o nome, etc.) e as acções disponíveis na mesma (adicionar, consultar, etc.). Na figura 7 é apresentado o modelo correspondente.

O processo de engenharia reversa assenta num certo número de assunções e heurísticas que permitem tornar o processo viável. Estas abordam basicamente dois aspectos: aspectos relacionados com a arquitectura da aplicação a reverter e aspectos relacionados com o nível de abstracção a utilizar. Relativamente aos primeiros, assume-se que o código está organizado segundo um modelo semelhante à arquitectura MVC, utilizando para isso o padrão Observer-Observable do Java. Isto permite a separação entre o código da interface e o código da

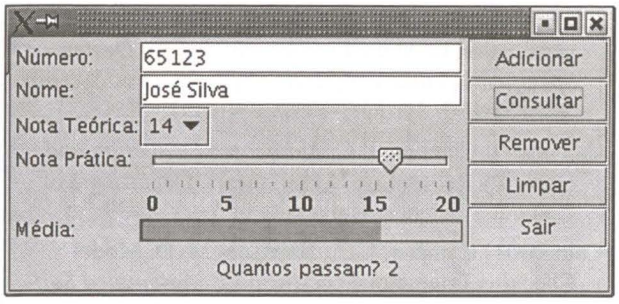

Figura 6 - Uma interface de teste

```
interactor JTurma
includes
        Turma via turma
attributes
        nome: String
        numero: String
        notaTeorica: 0..20
        notaPratica: 0..20
        media: 0..20
actions
        set_nome(n) set_numero(n) set_notaTeorica(n)
        set_notaPratica(n)
        adicionar consultar limpar remover sair
axioms
        [set_nome(n)] nome'=n
        [set_numero(n)] numero'=n
        [set_notaTeorica(n)] notaTeorica'=n
        [set_notaPratica(n)] notaPratica'=n
        [adicionar] ...
        [consultar] ...
        [limpar] ...
        [sair] ...
```

Figura 7 - Modelo para a interface de teste

camada computacional da aplicação. Para além desta assunção básica outras mais específicas são utilizadas. Por exemplo, relativamente ao modo como os *listeners* são codificados. Relativamente aos segundos é necessário decidir, por exemplo, como devem os botões presentes na interface ser modelados: simplesmente como acções, ou deverá um *interactor* ser utilizado para os modelar explicitamente. Neste case específico, e nesta fase, optou-se pela primeira solução, tanto por ser mais simples como por parecer ser mais útil.

O aspecto mais complexo do processo é realização de engenharia reversa sobre o código da camada computacional da aplicação. Nesta fase este aspecto serão apenas identificadas algumas das acções disponíveis na camada computacional a partir de heurísticas aplicadas aos *listeners* presentes no código. Para colmatar esta questão está planeado um modo interactivo em que o utilizador poderá guiar o processo de aplicação de heurísticas bem como completar o modelo.

5. ESTADO DE DESENVOLVIMENTO

Os diversos componentes da arquitectura estão neste momento em desenvolvimento. Uma versão do compilador i2smv existe já implementada em Perl. A sua re-implementação em Java utilizando AntLR está a ser considerada. Uma primeira versão do editor está também já implementada em Java. O conjunto das duas ferramentas está a ser testado na modelação e análise de diferentes alternativas de interface para um sistema de tratamento de águas residuais.

O componente de engenharia reversa está em fase de desenvolvimento. Neste momento permite já a geração da parte estática dos modelos (identificação dos atributos e das acções possíveis). Está a ser iniciado, neste momento, o tratamento dos axiomas.

Os restantes componentes (editor de propriedades e componente de visualização) estão em fase de concepção.

6. CONCLUSÕES

A usabilidade é claramente um dos factores determinantes para a qualidade e sucesso de um sistema interactivo. As abordagens baseadas em modelos permitem começar a analisar a usabilidade dos sistemas desde muito cedo no processo de desenvolvimento. Quando se trata de modelos rigorosos e formais torna-se possível a aplicação de tecnologias de raciocínio automatizado (*model-checkers* e demonstradores de teoremas) como auxiliares ao processo de análise. Este aspecto é particularmente relevante quando consideramos o comportamento de sistemas complexos. Isto tem levado ao estudo da aplicação destas técnicas a modelos de sistemas interactivos.

Para este tipo de abordagem ter sucesso torna-se necessário fornecer apoio ao processo de modelação e análise. Esse apoio deverá ser concretizado na forma de ferramentas que suportem esse processo. Neste artigo foi apresentada a arquitectura de uma ferramenta para a análise de usabilidade baseada em modelos. Os modelos são escritos em Lógica Modal de Acções e estruturados utilizando a noção de *interactor*. A ferramenta fornece suporte à fase de escrita dos modelos e permite analisar propriedades do comportamento dos modelos através da tradução desses modelos para o *model-checker* SMV. É ainda fornecido suporte à análise dos resultados produzidos pelo processo de verificação.

A ferramenta está neste momento em desenvolvimento e o estado actual de cada um dos seus componentes foi descrito. Uma primeira versão do editor e do compilador estão já disponíveis e permitem já a modelação e análise de sistemas interactivos. Uma primeira versão do módulo de engenharia reversa deverá ficar disponível em breve.

No que diz respeito a trabalhos relacionados, IFADIS [Loer03] é uma ferramenta para a análise de usabilidade baseada em modelos que possui semelhanças com a que aqui se descreve. No caso do IFADIS, no entanto, não existe um componente de edição de modelos. Os modelos são desenvolvidos utilizando Ofan Statecharts recorrendo à ferramenta Statemate [Harel88].

O facto de o Statemate ser uma ferramenta proprietária e não especificamente adaptada à modelação de sistemas interactivos torna a adopção do IFADIS como ferramenta de modelação e análise deste tipo de sistemas mais complicada. Com efeito, está neste momento a ser realizada uma análise comparativa que envolve a modelação de um mesmo sistema com as duas notações (Statecharts e *interactors*/ MAL). Os resultados preliminares apontam para uma maior adequação da

combinação *interactors*/MAL à modelação de sistemas interactivos ao nível de abstracção que se pretende.

A utilização de modelos formais tem também os seus inconvenientes. Um dos problemas usualmente mais mencionado prende-se com a necessidade (e, por vezes, dificuldade) de apreender uma nova linguagem de modelação. O desenvolvimento da ferramenta permite atenuar este problema, uma vez que todo o processo passa a ser efectuado com assistência por parte da mesma.

Um outro problema relaciona-se com o facto de o tipo de análise que é possível realizar sobre os modelos ser limitado em relação a abordagens menos formais. Trata-se de uma questão real e no caso presente a análise que é possível realizar tem a ver essencialmente com o comportamento do sistema interactivo (por exemplo, questões relacionadas com a existência de modos na interface, ou com a previsibilidade do sistema — ver [Campos99] para alguns exemplos). Assim, questões relacionadas com a representação de informação na interface não são facilmente analisáveis neste contexto.

Alguns pontos devem no entanto ser realçados: (a) A análise, apesar de limitada no âmbito, é efectuada com maior rigor e de forma mais exaustiva do que o que é normalmente possível em processos menos formais. (b) A experiência tem demonstrado que os resultados da análise, apesar de esta ser apenas efectuada sobre o comportamento, levantam questões quem tem depois que ser respondidas considerando também aspectos de representação, possivelmente com recurso a outras técnicas de análise. (c) Para além da análise propriamente dita, o processo de modelação em si mesmo obriga a olhar para o sistema em desenvolvimento com particular atenção, permitindo identificar problemas e desenvolver uma maior compreensão das questões logo nessa fase.

Como conclusão pode dizer-se que a abordagem proposta não pretende ser uma panaceia para todos os problemas que surgem no desenvolvimento de sistemas interactivos. No entanto, ela poderá ser útil na análise de questões relacionadas com o comportamento de tais sistemas e pode ser integrada com técnicas mais tradicionais de avaliação de usabilidade por forma a criar um processo de desenvolvimento em que questões de usabilidade são consideradas desde as primeiras fase de trabalho. Alguns estudos preliminares apontam na direcção da validade de ferramentas do género da aqui proposta [Loer04]. Este é, no entanto, um aspecto que só poderá ser totalmente validado quando uma primeira versão da ferramenta estiver disponível.

AGRADECIMENTOS

O autor agradece a Fernando Salgueiro e a José Mira o trabalho realizado na implementação dos componentes de edição e de engenharia reversa.

REFERÊNCIAS

[Booch98] Booch G., Rumbaugh J., Jacobson I. *The Unified Modeling Language User Guide*. Addison-Wesley, 1998.

[Campos98] Campos, J. C., Harrison, M. D. The role of verification in interactive systems design. *Design, Specification and Verification of Interactive Systems '98*, 155-170. Springer-Verlag/Wien, 1998.

[Campos99] Campos, J. C. *Automated Deduction and Usability Reasoning*. D.Phil. thesis, Department of Computer Science, University of York, 1999.

[Campos01] Campos, J. C., Harrison, M. D. Model Checking Interactor Specifications. *Automated Software Engineering*, 8(3-4), 275-310, 2001.

[Clarke86] Clarke, E. M., Emerson, E. A., Sistla, A. P. Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications. *ACM Transactions on Programming Languages and Systems*, 8(2), 244-263, 1986.

[Duke93] Duke, D. J., Harrison, M. D. Abstract Interaction Objects. *Computer Graphics Forum*, 12(3), 25-36, 1993.

[Dwyer99] Dwyer, M. B., Avrunin, G. S., Corbett, J. C. Patterns in property specifications for finite-state verification. *21st International Conference on Software Engineering (ICSE'99)*, 411-420, 1999.

[Harel88] Harel, D., Lachover, H., et al. Statemate: a working environment for the development of complex reactive systems. *10th International Conference on Software Engineering: ICSE '88*, 396-406. IEEE Computer Society Press, 1988.

[Loer03] Loer, K. *Model-based Automated Analysis for Dependable Interactive Systems*. PhD. thesis, Department of Computer Science, University of York, 2003.

[Loer04] Loer, K., Harrison, M. Integrating Model Checking with the Industrial Design of Interactive Systems. *ICSE'04 workshop: Bridging the Gaps II*, 9-16. The IEE, 2004

[McMillan93] McMillan, K. L. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.

[Paterno95] Paternò, F. D. *A Method for Formal Specification and Verification of Interactive Systems*. D.Phil. thesis, Department of Computer Science, University of York, 1995.

[Rosson02] Rosson, M. B., Carroll , J. M. *Usability Engineering: Scenario-Based Development of Human-Computer Interaction*. Morgan Kaufmann, 2002.

[Ryan91] Ryan, M., Fiadeiro, J., Maibaum, T. Sharing Actions and Attributes in Modal Action Logic. *Theoretical Aspects of Computer Software*. LNCS, vol. 526, 569-593. Springer-Verlag, 1991.

[SWEBOK01] *Guide to the Software Engineering Book of Knowledge, trial version 1.0* . IEEE, Maio 2001.

[Wall96] Wall, L., Christiansen, T., Schwartz, R. L. *Programming Perl (2nd edition)*. O'Reilly, 1996.