Uma abordagem formal à Engenharia da Usabilidade

José Creissac Campos
Departamento de Informática, Universidade do Minho
Campus de Gualtar, 4710-057 Braga, Portugal
tel.: + 351 253 60 4447
E-mail: Jose.Campos@di.uminho.pt

ABSTRACT
The quality of an interactive system can be measured in terms of its usability. Empirical approaches to usability evaluation attempt to assess the system under real usage conditions. This type of approach can be very expensive. Analytical approaches have been proposed as a means of reasoning about usability issues from early in development. These approaches use models to focus the analysis in specific usuability issues. In this context, the aplication of (mathematically) formal notations and tools has been proposed.

This paper presents a formal approach to the analysis of interactive systems. The analysis can be carried out taking into account all possible behaviours of the device, or it can be guided by the tasks the device is supposed to support.

RESUMO
A qualidade dos sistemas interactivos pode ser medida em termos da sua *usabilidade*. Abordagens empíricas à avaliação de usabilidade procuram analisar os sistemas sob condições reais de utilização mas, tipicamente, são dispendiosas. Abordagens analíticas têm sido propostas como um meio de raciocinar sobre questões de usabilidade desde as fases iniciais do desenvolvimento. Estas abordagens socorrem-se de modelos para focarem a análise em aspectos específicos da usabilidade. Neste contexto, a utilização de notações e ferramentas (matematicamente) formais tem sido proposta.

Este artigo apresenta uma abordagem formal à análise de sistemas interactivos que tem vindo a ser desenvolvida ao longo dos últimos sete anos. A análise tanto pode ser realizada tendo em conta apenas o comportamento do artefacto, como permite a integração de um modelo de tarefas por forma a restringir o comportamento do artefacto a um subconjunto adequado de todos os seus comportamentos possíveis.

Palavras chave
Análise de usabilidade, métodos formais, *model checking*

INTRODUÇÃO
É um facto reconhecido que o desenvolvimento de software é normalmente efectuado de modo iterativo, num processo incremental de tentativa e erro. Quanto mais avançado estiver o processo de desenvolvimento, tanto maior será o custo de introduzir alterações no sistema. Deste modo, é importante que a análise da qualidade do sistema a ser desenvolvido seja iniciada o mais cedo possível.

A qualidade dos sistemas interactivos pode ser medida em termos da sua *usabilidade*. Abordagens empíricas à avaliação procuram avaliar os sistemas sob condições reais de utilização mas, tipicamente, são dispendiosas. Isto acontece, tanto porque a análise é efectuada em fases avançadas do processo de desenvolvimento, como porque a montagem do processo de análise requer muitos recursos e tempo. Abordagens analíticas têm sido propostas como um meio de raciocinar sobre questões de usabilidade desde as fases iniciais do desenvolvimento. Estas abordagens socorrem-se de modelos para focarem a análise em aspectos específicos da usabilidade. Neste contexto, a utilização de notações e ferramentas (matematicamente) formais tem sido proposta [17,7].

Neste artigo apresenta-se um tal tipo de abordagem que tem vindo a ser desenvolvida, ao longo dos últimos anos, em colaboração com M. D. Harrison da Universidade de York [4,5,6,7,3]. Nesta abordagem, modelos dos artefactos[1] interactivos e informação sobre as tarefas que estes deverão suportar são utilizados para raciocinar sobre a usabilidade do sistema interactivo pretendido. A análise é realizada tentando provar que o comportamento dos modelos exibem determinadas propriedades consideradas desejáveis.

Os modelos são estruturados utilizando a noção de *interactor* [12,10] e são expressos através de uma lógica modal de acções. A análise é realizada utilizando a ferramenta SMV, um *model checker*, tendo sido desenvolvida uma

[1] Neste artigo utilizaremos o termo artefacto para referir o software/hardware a ser desenvolvido e reservaremos o termo *sistema interactivo* para referir a combinação do artefacto com o utilizador.

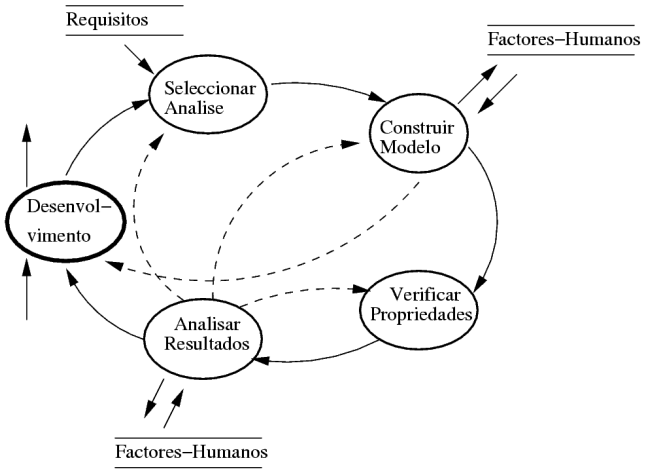

Figura 1. Processo de verificação

ferramenta para traduzir os modelos escritos na linguagem dos *interactors* para a linguagem do SMV. As propriedades a provar são escritas em CTL. O artigo apresenta também um exemplo de aplicação, tendo em vista ilustrar a abordagem.

VERIFICAÇÃO DE SISTEMAS INTERACTIVOS

O objectivo de uma abordagem formal à verificação de sistemas interactivos não é provar a correcção do sistema que está a ser desenvolvido. A noção de correcção implica a definição de uma medida absoluta de qualidade que possa ser utilizada como padrão. Ao tentarmos definir tal medida padrão somos confrontados com a questão da sua própria correcção. Tal como Henzinger afirma [14]:

> O único objectivo sensato dos métodos formais é detectar a presença de erros e fazê-lo desde cedo no processo de desenho.

Assim, o papel da verificação é por um lado descobrir potenciais problemas no sistema e por outro garantir que determinadas propriedades consideradas desejáveis são garantidas por esse mesmo sistema. Para atingir estes objectivos o processo de verificação necessita estar intimamente ligado ao processo de desenvolvimento, guiando-o e servindo como ajuda no processo de tomada de decisão relativamente a diferentes alternativas de desenho do sistema.

As primeiras abordagens à utilização de técnicas e ferramentas formais na análise de especificações de sistemas interactivos pressupunham o desenvolvimento prévio de um modelo do sistema que o caracterizasse de forma total. Este tipo de abordagens apresentava alguns dos inconvenientes das abordagens empíricas, uma vez que é necessário definir todo o sistema antes de iniciar o processo de análise. Uma abordagem mais flexível pode ser conseguida através do desenvolvimento e análise de modelos parciais do sistema, em que cada um dos modelos é uma representação do sistema a um nível adequado de abstracção, para a tarefa de verificação que se pretende levar a cabo. Quatro etapas principais podem ser identificadas para uma abordagem deste tipo (ver figura 1):

- selecção de aspectos a analisar;
- construção de um modelo apropriado;
- realização da verificação;
- análise dos resultados.

As próximas secções analisam cada um destes passos.

Selecção

Uma instância do processo de verificação começa com a identificação de um aspecto particular do sistema que é considerado suficientemente relevante para merecer uma verificação rigorosa das suas propriedades. Ao mesmo tempo, os objectivos do processo de verificação são também identificados.

Várias situações podem dar início ao processo de verificação, no entanto duas possibilidades principais merecem referência:

- requisitos de usabilidade pré-estabelecidos — a engenharia da usabilidade pressupõe a definição de requisitos de usabilidade que devem ser garantidos ao longo do processo de desenvolvimento; a sua validação para uma determinada proposta de desenho poderá passar por um processo de verificação formal.
- necessidade de tomar uma decisão relativamente ao desenho do sistema — um processo de análise formal pode ser utilizado para avaliar diferentes alternativas de desenho. Deste modo, é possível tomar decisões mais informadas, sem ter que esperar até fases mais adiantadas do desenvolvimento por forma a ter um protótipo para avaliação.

Ao aproximar a decisão sobre que aspectos verificar do processo de desenvolvimento, torna-se mais fácil identificar quais as propriedades realmente relevantes e utilizar os resultados da verificação para auxiliar o processo de desenvolvimento.

Modelação

Tendo identificado um aspecto do sistema que deve ser sujeito a verificação, torna-se necessário construir um modelo por forma a que a análise seja levada a cabo. Um primeiro ponto a considerar é o tipo de análise que se pretende realizar pois isso terá influência no tipo de modelo que deverá ser desenvolvido. Neste artigo iremos considerar apenas análises realizadas com recurso a *model checkers*. Este tipo de ferramenta é especialmente útil na verificação de propriedades relacionadas com aspectos comportamentais dos sistemas, no entanto, a utilização de outro tipo de ferramentas tem sido também considerada (ver [9] para um exemplo de aplicação de demonstradores de teoremas à análise de aspectos relacionados com a representação de informação na interface).

Um ponto importante a considerar, relativamente ao modelo a desenvolver, é a sua completude. O modelo deve apenas incluir a informação considerada relevante relativamente à análise que se pretende realizar. Mas como saber exactamente o que é relevante e o que não o é? A solução para este problema passa por identificar claramente quais as abstracções que estão a ser utilizadas. Estas abstracções poderão, caso seja necessário, serem elas próprias objecto de análise para demonstrar a sua validade. Deste modo, a abordagem suporta um tipo de análise composicional.

Obviamente, a possibilidade de realizar análise formal é apenas um dos benefícios da construção de modelos formais do sistema. O próprio processo de modelação permitirá obter informações relevantes sobre o sistema e melhorar a compreensão do mesmo.

Verificação

Nesta fase o modelo construído é validado relativamente às propriedades identificadas anteriormente. Para que tal seja possível é necessário expressar tais propriedades na lógica apropriada à análise que vai ser efectuada. Este processo será ilustrado mais adiante.

Análise

Finalmente os resultados devem ser avaliados. De um ponto de vista da análise, as situações interessantes acontecem quando a verificação falha. Nesse caso, será necessário identificar as causas de tal situação. Nas situações em que a análise é efectuada com *model checkers*, um contra-exemplo poderá ser fornecido pela ferramenta, mostrando uma situação em que a propriedade não se verifica.

A análise dos resultados deve ser efectuada com cuidado. Em particular duas questões devem ser colocadas:

- O problema encontrado é um problema do sistema, ou deve-se ao modo como o modelo foi desenvolvido? A utilização de abstracções pode significar que o modelo possui mais comportamento que o exibido pelo sistema final. Por outro lado, pode também acontecer que o modelo não inclua toda a informação relevante relativamente à propriedade em análise.
- O problema encontrado é realmente relevante? Pode acontecer que o problema encontrado, apesar de possível, não seja significativo em situações reais de utilização.

Uma vez atingida uma conclusão relativamente ao significado do insucesso da tentativa de prova, os resultados da análise podem ser incorporados numa nova iteração do processo, quer redefinindo os aspectos do sistema a considerar, quer refinando o modelo, quer alterando a codificação da propriedade.

Vantagens da abordagem

O processo iterativo de verificação, descrito nesta secção, apresenta diversas vantagens quando comparado com abordagens monolíticas em que a verificação é adiada até ser possível obter um modelo mais completo do sistema a ser desenvolvido:

- disponibilidade imediata de resultados — uma vez que a verificação é realizada durante o processo de desenho, os resultados passam a estar disponíveis mais cedo e potenciam um processo de decisão mais informado;
- maior relevância da análise — uma vez que a selecção do que deve ser modelado e verificado se foca nas características do sistema a ser desenvolvido, há maiores garantias de que o processo de verificação seja utilizado para validar características do sistema e não apenas do modelo (como pode acontecer em abordagens baseadas na produção inicial de um modelo completo do sistema);
- escolha da ferramenta — uma vez que os modelos são desenvolvidos conforme necessário, torna-se possível escolher a ferramenta mais adequada a cada situação de verificação (embora no presente artigo apenas o SMV seja utilizado, outras ferramentas têm sido exploradas);
- controlo da complexidade — uma vez que os modelos focam aspectos específicos do sistema, serão mais simples e fáceis de desenvolver e analisar, o que permite a análise de propriedades que, de outro modo, seriam demasiado complexas para verificar; o facto de se utilizarem modelos parciais levanta algumas questões relativas à completude e consistência dos diversos modelos, mas estes aspectos podem ser controlados recorrendo a uma abordagem de estilo composicional;
- reutilização — uma vez que os modelos se centram em características específicas dos sistemas, torna-se possível reutilizar formalizações quando a mesma propriedade está a ser verificada em diferentes modelos, ou mesmo reutilizar modelos quando os mesmos aspectos estão a ser analisados em sistemas diferentes (ver [2] para um exemplo);

QUE MODELOS?

Na secção anterior foi referida a necessidade de desenvolver modelos de artefactos interactivos de modo a analisar as suas características de usabilidade. No entanto, analisar a usabilidade de um artefacto implica ter em consideração aspectos relacionados com o utilizador. No caso corrente, isto reflecte-se na escolha do tipo de modelo a desenvolver, nas propriedades a provar e na interpretação dos resultados. Neste artigo iremos apenas considerar modelos do comportamento do sistema, para questões relacionadas com aspectos da representação da informação ver [9].

No que diz respeito a modelos do comportamento, uma primeira possibilidade consiste em modelar apenas o comportamento do artefacto e considerar, durante a verificação, todos os comportamentos possíveis que ele pode

exibir. A vantagem desta abordagem é que força a análise a considerar comportamentos imprevistos que tenham o potencial de deixar o artefacto em estados não desejados. Na prática, no entanto, isto pode originar falsos negativos que têm que ser investigados e tratados individualmente durante a análise. Como já foi mencionado, alguns destes comportamentos anómalos serão descartados, quer por se ficarem a dever ao modo específico como o modelo foi construído (por exemplo, a utilização de abstracções pode levar o modelo a exibir mais comportamento do que o artefacto real), quer por, mesmo tratando-se de comportamentos possíveis do artefacto, eles não serem considerados plausíveis/relevantes do ponto de vista dos factores-humanos (por exemplo, se estivermos a considerar o modo como é possível utilizar um telefone móvel para efectuar chamadas telefónicas, provavelmente não vamos querer considerar situações em que o utilizador cancela o processo de marcação do número).

A filtragem de comportamentos não desejados pode ser feita, quer alterando o modelo, quer alterando a propriedade a ser provada. Introduzir alterações no modelo, de forma a filtrar os comportamentos indesejados (mas sem alterar o seu restante comportamento), pode ser uma tarefa difícil quando estamos a lidar com sistemas complexos. Alterar a propriedade, por seu lado, pode também ser complexo e originar propriedades difíceis de ler e cujo significado se torna menos claro. A lógica CTL (*Computational Tree Logic*) é uma lógica apropriada para expressar propriedades acerca dos estados que um sistema pode ou não atingir, mas já não é tão apropriada para expressar propriedades sobre os comportamentos que levam a tais estados. Por exemplo, é fácil expressar que um estado que verifica *prop* pode ser alcançado (EF *prop*), todavia não é tão fácil escrever propriedades acerca dos comportamentos que levam a tal estado.

Uma solução para este problema passa por adicionar modelos do utilizador ou das tarefas aos modelos dos artefactos [21,11,18,13,3]. Isto aumenta a complexidade do modelo final, mas permite uma análise mais centrada em aspectos específicos da interacção entre o utilizador e o artefacto que se pretendam analisar. Apenas aqueles comportamentos que sejam consistentes com o modelo do utilizador/tarefa são considerados durante a análise. Uma diferença importante entre as duas abordagens é que enquanto os modelos do utilizador procuram definir como o comportamento do utilizador é *gerado*, os modelos de tarefas definem o comportamento que os utilizadores supostamente deverão ter. Os modelos do utilizador podem ser especialmente úteis na análise do comportamento de utilizadores inexperientes. Infelizmente, estes modelos são difíceis de desenvolver e tendem a ser bastante complexos. Os modelos de tarefas são especialmente úteis em situações onde existam comportamentos correctos previamente definidos e que devem ser seguidos pelos utilizadores.

Um inconveniente de utilizar um modelo do utilizador ou de tarefas é que o âmbito da análise é reduzido. Se um modelo do utilizador for utilizado, ele definirá tipicamente o comportamento dito *racional* dos utilizadores e, possivelmente, classes típicas de erros do utilizador. Isto pode levar a que alguns comportamentos anómalos, com potencial para terem um impacto negativo na usabilidade do sistema, não sejam considerados. Se um modelo de tarefas for utilizado, então apenas aqueles comportamentos definidos como correctos, de acordo com a descrição das tarefas, serão considerados durante a análise. Também neste caso, comportamentos anómalos, com impacto negativo na usabilidade, podem passar despercebidos. É possível considerar a hipótese de erros no modelo de tarefas, para obviar este problema, mas uma cobertura total de todos os erros possíveis não pode ser garantida.

As abordagens que não consideram directamente um modelo do utilizador ou das tarefas, procuram garantir que todos os comportamentos anómalos do artefacto são considerados durante a análise. Isto permite a identificação de comportamentos do sistema inicialmente não considerados e que possam prejudicar a usabilidade do sistema. O ponto fraco da abordagem situa-se no insuficiente suporte à análise de comportamentos específicos do utilizador e de como eles são suportados pelo artefacto. Para superar este problema optou-se pela inclusão, na análise, de informação relativa às tarefas que o artefacto deve suportar. A utilização de tarefas, em vez de modelos do utilizador, ficou a dever-se à maior complexidade deste últimos.

ANÁLISE DE INTERACTORS

Esta secção descreve a linguagem utilizada para modelar sistemas interactivos e a ferramenta que foi desenvolvida para permitir a sua análise com a ferramenta SMV. Mais detalhes podem ser encontrados em [7,13].

A linguagem dos interactors

Os *interactors*, tais como desenvolvidos em [10], 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 (ver figura 2). Assim, cada *interactor* (ver apêndice A para um exemplo) 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 (ρ) que define quais os atributos/acções que são apresentados ao utilizador (marcados com [vis]).

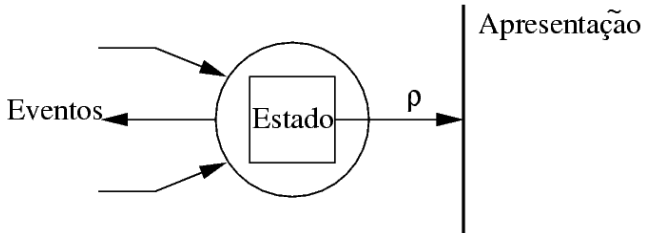

Figura 2. Interactor

No nosso caso particular, o comportamento dos *interactors* é definido utilizando uma Lógica Modal de Acções (MAL — *Modal Action Logic*) [19].

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 [newcall] ringer'=on $\wedge$ menu'=answercall $\wedge$ keep(dialflag, endcallflag, state) expressa o facto de que, depois da acção newcall, o valor do atributo ringer passa a ser on, o valor do atributo menu passa a ser answercall e os atributos dialflag, endcallflag e state não mudam de valor. 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(newcall) $\rightarrow \neg$ringer expressa o facto de que a acção newcall pode ocorrer apenas quando o atributo ringer 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=sending $\rightarrow$ obl(sent) expressa o facto de que quando o valor do atributo state é sending então a acção sent 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 [] ringer=off $\wedge$ menu=makecall $\wedge$ dialflag=nil $\wedge$ endcallflag=nil $\wedge$ state=idle define os valores dos diferentes atributos no estado inicial do modelo.

A ferramenta i2smv

Foi desenvolvida uma ferramenta que permite a verificação automática dos modelos utilizando o SMV. O SMV [15] é um *model checker* que utiliza CTL [8] como lógica para expressar propriedades. Os modelos são definidos como máquinas de estados finitas, e o CTL é utilizado para escrever propriedades acerca dos comportamentos exibidos pelos modelos.

As propriedades que podem ser escritas/verificadas têm a ver essencialmente com os estados que podem ou não ser atingidos pelo modelo. Propriedades típicas incluem:

- X é um invariante — AG(X) (X verifica-se em todos os estados de todos os comportamentos exibidos pelo modelo);
- X é inevitável — AF(X) (para todos os comportamentos possíveis do modelo, X verifica-se num dos estados desse comportamento);
- X é possível — EF(X) (para pelo menos um dos comportamentos possíveis do modelo, X verifica-se num dos estados desse comportamento);
- X até Y é um invariante — A[X U Y] (para todos os comportamentos do modelo, X verifica-se em todos os estados até que um estado em que Y se verifique seja alcançado);
- X até Y é possível — E[X U Y] (para pelo menos um dos comportamentos do modelo, X verifica-se em todos os estados até que um estado em que Y se verifique seja alcançado);

Estes operadores podem ser combinados para expressar propriedades mais complexas. Por exemplo, a propriedade AG(X $\rightarrow$ AF(Y)) define que é um invariante do modelo que sempre que X se verifique é inevitável que Y se venha a verificar.

A ferramenta i2smv traduz modelos escritos com a linguagem dos *interactors* para a linguagem da ferramenta SMV. Para tal foi definida uma tradução que permite expressar cada um dos axiomas da MAL como uma relação de transição entre estados. A única situação especial refere-se aos axiomas de obrigação para os quais é necessário utilizar condições de justiça (*fairness*). Aplicando esta tradução, torna-se possível verificar propriedades do comportamento dos *interactors* utilizando a lógica CTL e o SMV. Uma vez que o SMV não possui a noção de acção, o algoritmo de tradução introduz uma acção especial ``action'' que é utilizada ao nível do CTL para referir a acção que provocou a transição para o estado actual.

A ferramenta é composta por um compilador, que efectua a tradução dos modelos, e um modo Emacs que fornece um ambiente integrado de edição e tradução de modelos (ver figura 3). Para mais detalhes sobre a ferramenta ver [7].

Modelação do artefacto

Como exemplo de modelação e análise utilizando *interactors* vamos apresentar a modelação da funcionalidade básica de um telemóvel desenvolvida originalmente em [3]. Considere-se, então, um telemóvel com a capacidade de receber e efectuar chamadas telefónicas e de receber e enviar mensagens SMS. A utilização do telefone baseia-se

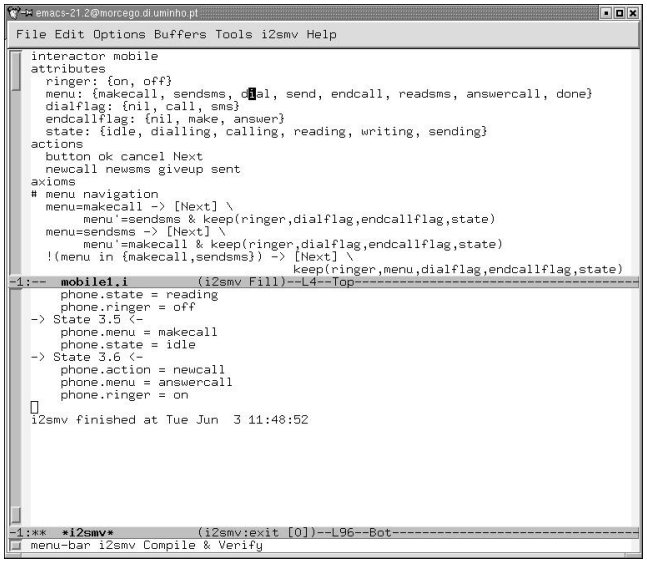

Figura 3. Modo Emacs do i2smv

num menu e é a interacção entre o utilizador e esse menu que se pretende analisar. Em cada momento, o menu apresenta uma única opção ao utilizador. Para além das teclas numéricas usuais, o telemóvel possui uma tecla OK (para seleccionar a opção do menu que está a ser apresentada), uma tecla Cancel (para reinicializar o menu) e uma tecla "→" para navegação no menu (avançar para a próxima opção).

A estrutura da navegação nos menus é apresentada na figura 4. Em modo normal de operação, a opção apresentada alterna entre ``make call'' e ``send sms''. Se, por exemplo, o utilizador selecciona ``make call'' (premindo a tecla OK enquanto essa opção está a ser apresentada no écran), o menu muda para ``dial''. O utilizador deverá digitar o número que pretende marcar e pressionar OK para efectuar a ligação. Uma vez estabelecida a ligação, o menu muda para ``end call''. Um processo semelhante é utilizado para o envio de mensagens SMS.

Existem duas excepções ao comportamento descrito acima. Quando uma chamada é recebida, o menu muda para ``answer call''. Esta opção permanecerá activa até que o utilizador aceite a chamada (premindo OK) ou rejeite a chamada (premindo Cancel), ou até que o telefone deixe de tocar. Quando uma mensagem SMS é recebida, o menu muda para ``read sms''. Esta opção permanecerá activa até que o utilizador aceite ou rejeite ler a mensagem.

Um modelo do telemóvel foi desenvolvido (ver apêndice A). Esse modelo reflecte a estrutura de menus descrita. O estado interno do telemóvel é modelado por dois atributos principais: ringer e state. O atributo ringer é utilizado para modelar o comportamento do telemóvel em termos de estar ou não a tocar. O telemóvel começa a tocar sempre que uma nova chamada ou mensagem SMS é recebida (ver

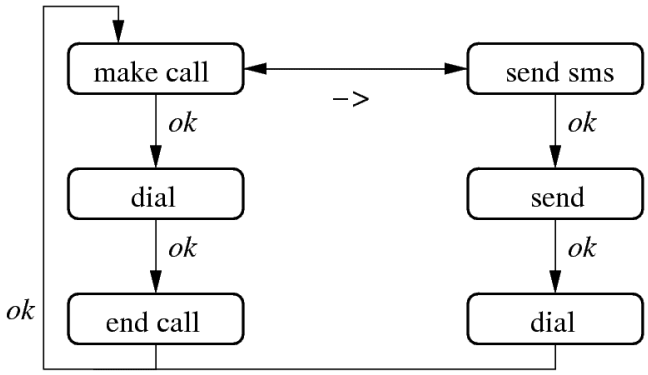

Figura 4. Estrutura de menus

axiomas 15 e 18) e pára de tocar quando o utilizador aceita a chamada/lê a mensagem. Se o utilizador não aceita a chamada/lê a mensagem, o telefone acaba por parar de tocar (ver axiomas 21 a 23). O atributo state modela o estado geral do telemóvel. Esta modelação é efectuada a um nível elevado de abstracção. É no, entanto, suficientemente detalhada para permitir a análise que se pretende. Os estados possíveis são:

- idle — o telemóvel está em descanso;
- dialling — o telemóvel está a ligar um número;
- calling — o telemóvel está a efectuar uma chamada;
- reading — o telemóvel está a apresentar uma mensagem SMS;
- writing — está a ser escrita uma mensagem SMS;
- sending — o telemóvel está a enviar uma mensagem SMS.

Dois atributos adicionais são utilizados para distinguir entre marcar um número para uma chamada telefónica ou para enviar uma mensagem SMS (atributo dialflag) e entre fazer ou receber uma chamada telefónica (atributo endcallflag). Estes atributos são necessários pois o comportamento do telemóvel é diferente em cada um dos casos.

Modelação de tarefas

A área da análise de tarefas é uma área bem estudada. No âmbito deste trabalho optou-se por não adoptar uma linguagem de modelação de tarefas específica, por forma a manter a abordagem o mais genérica possível. Assim, abstraímo-nos de uma qualquer notação concreta e consideramos simplesmente o que é essencial na noção de tarefa.

Qualquer que seja a linguagem de modelação de tarefas utilizada, as tarefas serão descritas como as sequências de eventos válidas para atingir um determinado objectivo. Uma vez que estamos interessados em analisar o artefacto, só nos interessam os eventos relativos a acções que o utilizador possa efectuar. Para representar essas sequências de eventos utilizamos uma linguagem de descrição de traços de comportamento desenvolvida em [1]. A

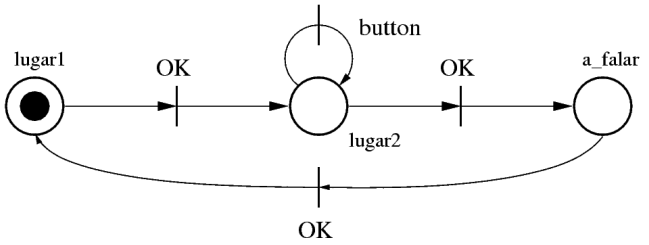

Figura 5. Rede de Petri para fazer uma chamada

linguagem possui operadores de sequência (•), iteração (*), alternativa (+) e paralelismo síncrono (||) e assíncrono (|). A semântica desta linguagem é expressa em termos de redes de Petri Condição/Evento. Outros autores propuseram também redes de Petri como notação intermédia para a representação de modelos de tarefas (cf. [16,13]).

A título de exemplo, considere-se a tarefa de efectuar uma chamada no telemóvel descrito anteriormente. Começando no estado inicial do modelo, este objectivo pode ser atingido premindo a tecla OK, marcado o número e premindo OK novamente. Para terminar a chamada, a tecla OK deve ser premida uma última vez. Este comportamento é descrito pela expressão: OK•button*•OK•OK. Para facilitar a análise vamos incluir etiquetas na definição do comportamento. O modelo para a tarefa fica então:

$$(OK \bullet button^* \bullet OK[a\_falar] \bullet OK)^*$$

Esta expressão dá origem à rede de Petri apresentada na figura 5.

As acções são modeladas por transições e são criados os lugares necessários para que as sequências válidas de acções sejam modeladas pelo *disparar* das transições. Sempre que existirem etiquetas na expressão de comportamento, estas são utilizadas para identificar os lugares apropriados da rede de Petri.

Note-se que apesar de estarmos a modelar apenas o comportamento do utilizador é também possível incluir acções do artefacto no modelo. Para decidir qual a melhor abordagem, é necessário considerar qual o objectivo da análise. Se o objectivo for analisar o modo como a estrutura das tarefas está definida, então tanto acções dos utilizadores como do artefacto devem ser incluídas no modelo de tarefas. Se o objectivo for analisar o artefacto, então ao incluir apenas acções do utilizador no modelo de tarefas, torna-se possível analisar de que modo o artefacto reage às acções do utilizador e se este conseguirá atingir os seus objectivos executando a tarefa. Esta última abordagem é menos normativa, uma vez que não coloca restrições ao comportamento do artefacto. Deste modo torna-se possível identificar situações em que o comportamento do artefacto se desvia do esperado, apesar do utilizador seguir os procedimentos pré-estabelecidos.

interactor making_a_call
attributes
lugar1, lugar2, a_falar: boolean
actions
OK button
axioms
$per(OK) \rightarrow ((lugar1 \wedge \neg lugar2) \vee (lugar2 \wedge \neg a\_falar) \vee (a\_falar \wedge \neg lugar1))$
$(lugar1 \wedge \neg lugar2) \rightarrow [OK] \neg lugar1' \wedge lugar2' \wedge keep(a\_falar)$
$(lugar2 \wedge \neg a\_falar) \rightarrow [OK] \neg lugar2' \wedge a\_falar' \wedge keep(lugar1)$
$(a\_falar \wedge \neg lugar1) \rightarrow [OK] \neg a\_falar' \wedge lugar1' \wedge keep(lugar2)$
$per(button) \rightarrow lugar2$
[button] keep(lugar1,lugar2,a_falar)

Figura 6. Interactor para a tarefa de realizar chamada

Uma vez que as tarefas podem ser apresentadas por redes de Petri, modelar as tarefas utilizando *interactors* reduz-se a modelar essas redes de Petri utilizando a lógica MAL. Isto é conseguido modelando cada lugar com uma variável boleana, representando se o lugar está ou não marcado, e cada transição com dois axiomas. Um axioma de permissão, expressando quando a transição pode disparar, e um axioma modal, expressando o efeito da transição na marcação da rede.

Por exemplo, a transição OK do lugar 1 para o lugar 2, no modelo acima, é modelada pelo axioma de permissão:

$$per(OK) \rightarrow lugar1 \wedge \neg lugar2$$

(ou seja, OK pode disparar quando o lugar 1 estiver marcado e o lugar 2 não estiver marcado) e pelo axioma modal:

$$(lugar1 \wedge \neg lugar2) \rightarrow [ok] \neg lugar1' \wedge lugar2' \wedge keep(a\_falar)$$

que significa: nas condições definidas pelo axioma de permissão (isto é necessário pois neste caso existem outras transições etiquetadas com OK) o efeito de disparar OK é deixar o lugar 1 não marcado, o lugar 2 marcado e o lugar a_falar inalterado.

Quando várias transições estão associadas ao mesmo evento, como é o caso presente, os axiomas de permissão são agregados por conjunção. O *interactor* modelando a tarefa de realizar uma chamada é apresentado na figura 6. Note-se que o axioma modal para button não necessita de guarda, uma vez que existe apenas uma transição etiquetada com essa acção.

Análise

Nesta secção mostra-se como é possível explorar o desenho proposto para o telemóvel utilizando os diferentes modelos desenvolvidos na secção anterior.

Utilizando apenas o modelo do artefacto

Utilizando apenas o modelo do telemóvel é já possível testar algumas das características do desenho proposto.

Para tal, é primeiro necessário definir um *interactor* main, onde serão definidas as propriedades a testar:

interactor main

includes

mobile **via** phone

test

...

Para testar se é possível efectuar uma chamada, a verificação da seguinte propriedade é tentada:

EF(phone.state=calling)

A resposta é de que a propriedade se verifica. Ficamos assim a saber que é possível colocar o telefone num estado em que uma chamada está a decorrer. Não sabemos, no entanto, como é possível atingir tal estado. Infelizmente não existe forma directa de descobrir quais os comportamentos que verificam a propriedade.

Um outro teste que pode ser efectuado é ver se o telefone toca sempre que uma chamada é recebida. A propriedade a verificar é:

AG(phone.action=newcall → phone.ringer=on)

Mais uma vez a resposta é que a propriedade se verifica. Note-se que questões como saber se o utilizador será capaz de se aperceber que o telefone está a tocar caem fora do âmbito deste tipo de análise. Mesmo assim, é útil ser capaz de provar que o telemóvel funciona adequadamente.

A análise não fica, no entanto, limitada a saber se o telemóvel irá tocar ou não. É uma coisa saber que o telemóvel irá tocar, é outra saber se o utilizador, caso se aperceba do toque, vai ser capaz de atender a chamada. A propriedade para testar se uma chamada pode sempre ser atendida é:

AG(phone.menu=answercall → AF(phone.state=calling))

(sempre que o menu apresenta a opção answercall é inevitável que o telemóvel ficará eventualmente no estado calling).

Esta propriedade é falsa e o contra-exemplo apresentado mostra que o utilizador pode cancelar a chamada em vez de a atender. Este comportamento é correcto, mas nós queremos considerar situações em que o utilizador quer receber a chamada. Para eliminar o comportamento anterior alteramos a propriedade para ser:

AG(phone.menu=answercall →

AF(phone.state=calling ∨ phone.action ∈ {cancel}))

Neste caso, um novo contra-exemplo é apresentado, desta vez mostrando que pode ser o telefone a deixar de tocar antes que o utilizador atenda a chamada. Este tipo de resultado levanta a questão de saber durante quanto tempo o telemóvel irá tocar antes de desistir da chamada. Por seu lado, essa análise obriga a identificar as situações em que tal pode acontecer. As duas situações típicas são:

- quem está a tentar efectuar a chamada desiste da mesma;
- a central telefónica cancela a chamada (talvez porque activou o sistema de *voice mail*).

Em qualquer dos casos, estamos a lidar com situações que fogem ao controle directo do telemóvel, pelo que, apesar de anotadas, não serão consideradas em mais detalhes na presente análise.

Quando a propriedade é novamente alterada, por forma a que o comportamento acima seja eliminado, um novo contra-exemplo é apresentado, mostrando que uma mensagem SMS pode chegar antes que o utilizador atenda a chamada. De facto, na actual especificação do sistema, a chegada de uma mensagem SMS cancela qualquer chamada que esteja pendente ou a decorrer. Este é, certamente, comportamento indesejado e a especificação deverá ser alterada de forma a resolver esta questão.

A análise apresentada mostra como este tipo de abordagem pode ser útil na detecção de potenciais problemas de usabilidade. A análise implica um processo de filtragem de comportamento indesejado através da inclusão de restrições ao comportamento nas propriedades a serem provadas. Este processo potencia a identificação das condições contextuais necessárias para que determinada propriedade do sistema se verifique. No entanto, o processo pode tornar-se demorado e difícil para sistemas complexos.

Nesse caso, as restrições necessárias podem mesmo tornar-se mais elaboradas do que o que é possível expressar directamente em CTL. Acresce a isto que, com este tipo de abordagem, não é fácil verificar se um determinado tipo de comportamento, por parte do utilizador, tem o efeito desejado no artefacto. As propriedades que podem ser escritas têm a ver principalmente com os estados em que o artefacto se vai encontrando, e não tanto com o comportamento que o sistema exibe. Raciocinar sobre como um determinado objectivo é atingido é feito de forma indirecta. O que as restrições modelam é essencialmente o que não deve acontecer para que o objectivo seja atingido.

Se pretendemos raciocinar sobre se um determinado comportamento permite atingir um determinado objectivo, torna-se mais simples recorrer ao modelo de tarefas.

Utilizando o modelo de tarefas

Para verificar o modelo do artefacto face ao modelo de tarefas, é necessário estabelecer a ligação entre os dois modelos. Para isso um novo *interactor* main é necessário:

interactor main

includes

mobile **via** device

making_a_call **via** task

axioms

task.action≠nil → task.action=device.action

task.action=nil →

device.action ∉ {button,ok,cancel,Next}

Este *interactor* estabelece a ligação entre o modelo da tarefa e o modelo do artefacto que a deverá suportar. A ligação entre os dois modelos é efectuada ao nível das acções. O primeiro axioma estabelece que sempre que uma acção ocorre no modelo de tarefas, então a mesma acção deve ocorrer no modelo do artefacto.

O segundo axioma restringe as acções que podem ocorrer livremente no modelo do artefacto (isto é, de forma independente do modelo de tarefas). Neste caso, não podem ser acções realizadas pelo utilizador. Em conjunto, os dois axiomas restringem o comportamento do sistema de forma a que as acções do utilizador possam ocorrer apenas de acordo com o modelo de tarefas e as acções do artefacto possam ocorrer livremente (respeitando a semântica do mesmo).

Utilizando esta nova versão do *interactor* main, torna-se possível efectuar uma análise mais exaustiva sobre o modo como o artefacto reage ao comportamento esperado do utilizador. Uma primeira propriedade, que é possível verificar, é saber se seguir a sequência de instruções definida pela tarefa torna possível efectuar uma chamada. A propriedade a verificar é:

EF(phone.state=calling ∧ task.a_falar)

Note-se que é importante verificar se a tarefa atingiu o lugar a_falar. Se apenas considerássemos o estado do telemóvel poderiamos obter falsos positivos. Considere-se, por exemplo, a seguinte sequência de eventos:

device.newcall•task.OK

É um comportamento que *encaixa* no comportamento definido pelo modelo da tarefa, e deixa o telemóvel no estado calling, mas sem que a tarefa seja terminada (apenas o primeiro passo é efectuado).

A propriedade definida acima é verificada pelo SMV. O próximo passo é verificar se executar a tarefa resulta sempre no estabelecimento de uma chamada. Isto pode ser feito com a propriedade:

AF(phone.state=calling ∧ task.a_falar)

Esta propriedade falha com o seguinte contra-exemplo:

task.ok•device.newcall•device.giveup•*a tarefa continua (mas o telemóvel já não está no estado esperado)...*

O que isto demonstra é que caso o telefone toque enquanto um número está a ser marcado, então esse número não pode ser chamado. De facto, no actual modelo, a chegada de uma uma nova chamada ou mensagem SMS interrompe qualquer tarefa que esteja a ser realizada. Por exemplo, verificar a propriedade:

AG((phone.state=calling ∧ task.a_falar) →

AF(task.action=ok ∧ phone.state=idle))

(é sempre possível terminar uma chamada premindo a tecla OK) mostra que se durante a realização de uma chamada uma mensagem SMS chegar, então a chamada é perdida (ou pelo menos a possibilidade de a terminar!).

Para resolver este problema uma nova versão do telemóvel deve ser desenvolvida. Por agora podemos filtrar este comportamento indesejado reescrevendo a propriedade:

AF((phone.state=calling ∧ task.a_falar) ∨

device.action ∈ {newcall,newsms})

A tentativa de verificar esta propriedade revela um problema com o modelo de tarefas: não existe limite máximo para o número de vezes que o utilizador é suposto premir botões quando está a marcar um número. Isto significa que, de acordo com o modelo da tarefa, o utilizador pode premir botões indefinidamente, actuando de acordo com o modelo, mas não conseguindo atingir o objectivo de fazer uma chamada telefónica. Para resolver isto podemos tornar o modelo da tarefa mais concreto. Por exemplo, sem perda de generalidade poderíamos considerar a tarefa de marcar números de nove dígitos.

Redesenhar o artefacto

Os dois tipos de análise realizados anteriormente coincidem na identificação de problemas relacionados com a chegada de novas chamadas telefónicas ou mensagens SMS. Em ambos os casos, o problema foi evitado através da alteração da propriedade a provar. Isso permitiu que as análises prosseguissem, mas o problema com o desenho proposto mantém-se. Para o resolver é necessário corrigir o modelo. Não é propósito deste artigo apresentar uma especificação completa de um telemóvel, no entanto, para ilustrar a abordagem, uma nova versão será apresentada.

O problema com a corrente versão do modelo é a interferência entre o comportamento iniciado pelo artefacto (a chegada de uma nova chamada ou mensagem sms) e o comportamento iniciado pelo utilizador (fazer uma chamada ou enviar uma mensagem sms). A solução passa por *isolar* os dois tipos de comportamento. Assumindo que atribuímos mais importância ao comportamento iniciado pelo utilizador, vamos deixá-lo como está e alterar o comportamento do telemóvel relativamente à chegada de novas chamadas ou mensagens SMS, por forma a que a sua resposta não seja intrusiva face às actividades do utilizador.

Relativamente a mensagens SMS a ideia base é a de que o telemóvel deverá dar indicação de que uma nova mensagem

foi recebida, mas não deverá provocar qualquer outra alteração no estado do telemóvel. Isto pode ser conseguido incorporando um indicador de novas mensagens no écran do telemóvel (um ícone que é apresentado sempre que existirem mensagens novas). Relativamente à recepção de chamadas telefónicas, a solução simplista (provavelmente demasiado simples) é rejeitar chamadas sempre que o telefone está em uso.

Estas opções de desenho causam algumas alterações no modelo. Um novo atributo boleano (sms) é introduzido para representar o estado do ícone. Uma fila de mensagens é também introduzida para guardar as mensagens ainda não lidas. Se a fila estiver cheia, novas mensagens não podem ser recebidas. A fila é representada por um atributo (queue) com valores entre zero e cinco.

As secções ``receber sms'' e ``fazer chamada'' da definição do comportamento necessitam também ser actualizadas. Os axiomas relativos à chegada de mensagens SMS ficam a ser:

$$\begin{aligned}
&per(newsms) \rightarrow queue<5\\
&[newsms]\ sms' \wedge queue'=queue+1\\
&menu=readsms \rightarrow [ok]\ queue'=queue-1 \wedge\\
&\qquad menu'=done \wedge state'=reading \wedge\\
&\qquad keep(ringer,sms,dialflag, endcallflag)\\
&menu=done \rightarrow [ok]\ sms'=(queue'>0) \wedge state'=idle\\
&\qquad \wedge\ menu'=makecall \wedge\\
&\qquad keep(ringer,queue,dialflag, endcallflag)
\end{aligned}$$

para a recepção de chamadas o axioma de permissão fica:

$$per(newcall) \rightarrow state=idle \wedge \neg ringer$$

Todos os outros axiomas são actualizados para levarem em linha de conta os novos atributos. Com o modelo resultante, torna-se agora possível provar que levar a cabo a tarefa tal como definida resulta sempre no estabelecimento de uma chamada telefónica.

O mesmo tipo de raciocínio pode ser efectuado para outras tarefas. Por exemplo, relativamente ao envio de mensagens os resultados mostram que uma tarefa similar à utilizada para as chamadas telefónicas permite ao utilizador enviar mensagens SMS, excepto se uma chamada é recebida antes de o utilizador seleccionar a opção ``send sms''.

DISCUSSÃO E CONCLUSÕES

Para garantir a qualidade dos sistemas ao mais baixo custo é necessário iniciar a análise de qualidade o mais cedo possível. A qualidade de sistemas interactivos pode ser medida em termos da sua usabilidade. Diversos métodos analíticos têm sido propostos para a análise de modelos de sistemas interactivos desde cedo no processo de desenvolvimento, incluindo métodos baseados na utilização de métodos formais de especificação e análise.

Este artigo apresentou uma abordagem formal à verificação de sistemas interactivos, que tem vindo a ser desenvolvida ao longo dos últimos sete anos. A análise tanto pode ser realizada, tendo em conta apenas o comportamento do artefacto, como permite a integração de um modelo de tarefas por forma a restringir o comportamento do artefacto a um subconjunto adequado de todos os seus possíveis comportamentos. Na versão não baseada em tarefas as restrições podem ser incluídas directamente nas propriedades a serem verificadas. À primeira vista, poderá parecer que estamos apenas a olhar para diferentes formas de atingir o mesmo objectivo. Existe, no entanto, uma diferença marcante. Numa abordagem com tarefas, o conhecimento sobre as tarefas deve ser conhecido à partida: representa o modo previsto de utilização do artefacto. O objectivo da análise é verificar se o artefacto suporta adequadamente tal utilização. Numa abordagem sem tarefas não é assumido conhecimento prévio sobre a forma de utilização do sistema (pelo menos não de forma directa). As restrições emergem das tentativas de prova efectuadas. Estas restrições representam conhecimento que é tornado explícito pela análise. Torna-se claro que as abordagens são complementares.

Outros autores propuseram abordagens similares. Paternò [17] deriva modelos do artefacto a partir da análise de tarefas. Esses modelos são especificados em LOTOS e analisados utilizando ACTL (uma lógica temporal de acções) e a ferramenta Logic Checker. A principal vantagem de utilizar CTL em vez de ACTL é que esta última lida apenas com acções, não permitindo expressar propriedades sobre o estado do artefacto de forma simples.

Tal como nós, Fields [13] utiliza modelos de artefacto e de tarefa separados. No seu caso, a análise é efectuada com a ferramenta Murϕ, a qual permite explorar o comportamento resultante da combinação dos dois modelos. Uma vantagem de utilizar SMV em relação ao Murϕ é o maior poder expressivo que permite na escrita de propriedades.

Rushby [18] utiliza SMV para analisar modelos conjuntos de artefacto e utilizador. Neste caso, no entanto, não existe uma separação clara entre os dois modelos. Apesar de Rushby se referir a modelos do utilizador, não é completamente claro se são modelos dos utilizadores ou modelos da actividade dos utilizadores (ou seja, modelos de tarefas).

Um estilo diferente de abordagem é proposto por Thimbleby [20]. Neste caso, o Mathematica é utilizado para realizar a análise da estrutura de menus de um telefone móvel. A análise, no entanto, não se refere ao efeito de acções no estado do artefacto, ou o resultado de efectuar alguma tarefa específica. É assumido que o artefacto se comporta correctamente e a sua interface é analisada relativamente à sua complexidade. A complexidade de uma interface é definida em relação ao número de acções necessárias para atingir a funcionalidade desejada em algum ponto da estrutura de menus. A análise é baseada em distribuições probabilísticas da utilização das funcionalidades do artefacto e das acções da interface.

A abordagem apresentada permite a análise de um conjunto alargado de propriedades relativas ao comportamento de artefactos interactivos e à sua interacção com os utilizadores. Relativamente a trabalho futuro, um aspecto que parece promissor é a exploração de padrões de tarefas. Até agora, a correspondência entre acções do modelo de tarefas e do modelo do artefacto tem sido efectuada ao nível dos nomes das acções. Seria útil ter um mecanismo mais genérico que permitisse ter padrões de tarefas, os quais poderiam ser instanciados como necessário. Para tornar isso possível, será necessário estender a linguagem dos *interactors* para permitir que estes possam ser parameterizados com acções.

A utilização de *model checkers* para análise de modelos é apenas uma das possibilidades, no que diz respeito à aplicação de ferramentas de raciocínio automático. A utilização de demonstradores de teoremas também tem sido explorada, embora não de forma tão aprofundada. No caso dos demonstradores de teoremas a análise realizada tem-se referido essencialmente à relação entre o estado do artefacto, a forma como esse estado é representado na interface e a forma como essa interface é percebida pelos utilizadores. Analisando a consistência entre modelos destes três níveis é possível detectar problemas na forma como a interface representa o estado do artefacto. Essa é também uma área em que se pretende continuar a investir no futuro.

AGRADECIMENTOS

O trabalho relatado neste artigo tem vindo a ser desenvolvido em colaboração com Michael D. Harrison da Universidade de York.

REFERÊNCIAS

1. J. C. Campos. *GAMA-X Geração Semi-Automática de Interfaces Sensíveis ao Contexto*. MSc. thesis, Departamento de Informática, Universidade do Minho, 1993.
2. J. C. Campos. *Automated Deduction and Usability Reasoning*. DPhil thesis, Department of Computer Science, University of York, 1999. Disponível como Technical Report YCST 2000/9.
3. J. C. Campos. Using task knowledge to guide interactor specifications analysis. In J. Jorge, N. Nunes e J. Cunha, editores, *Design Specification and Verification of Interactive Systems '03*. pp 159-174. Preliminary Proceedings.
4. J. C. Campos and M. D. Harrison. Formally verifying interactive systems: A review. In M. D. Harrison e J. C. Torres, editores, *Design, Specification and Verification of Interactive Systems '97*, Springer Computer Science, pp 109-124. Springer-Verlag/Wien, Junho 1997.
5. J. C. Campos and M. D. Harrison. The role of verification in interactive systems design. In P. Markopoulos e P. Johnson, editores, *Design, Specification and Verification of Interactive Systems '98*, Springer Computer Science, pp 155-170. Springer-Verlag/Wien, 1998.
6. J. C. Campos and M. D. Harrison. Using automated reasoning in the design of an audio-visual communication system. In D. J. Duke e A. Puerta, editores, *Design, Specification and Verification of Interactive Systems '99*, Springer Computer Science, pp 167-188. Springer-Verlag/Wien, 1999.
7. J. C. Campos and M. D. Harrison. Model checking interactor specifications. *Automated Software Engineering*, 8(3/4):275-310, Agosto 2001.
8. E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244-263, Abril 1986.
9. G. Doherty, J. C. Campos, and M. D. Harrison. Representational reasoning and verification. *Formal Aspects of Computing*, 12:260-277, 2000.
10. D. J. Duke and M. D. Harrison. Abstract interaction objects. *Computer Graphics Forum*, 12(3):25-36, 1993.
11. D. J. Duke, P. J. Barnard, D. A. Duce, and J. May. Syndetic modelling. *Human-Computer Interaction*, 13(4):337-393, 1998.
12. G. Faconti and F. Paternò. An approach to the formal specification of the components of an interaction. In C. Vandoni e D. Duce, editores, *Eurographics '90*, pp 481-494. North-Holland, 1990.
13. R. E. Fields. *Analysis of erroneous actions in the design of critical systems*. DPhil thesis, Department of Computer Science, University of York, 2001.
14. T. A. Henzinger. Some myths about formal verification. *ACM Computing Surveys*, 28(4es):119-es, Dezembro 1996.
15. K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.
16. Ph. Palanque, R. Bastide, and V. Senges. Task model - system model: towards an unifying formalism. *Proceedings of HCI International conference*, pp 489-494, Yokoohama, Japan, Julho 1995. Elsevier.
17. F. D. Paternò. *A Method for Formal Specification and Verification of Interactive Systems*. PhD thesis, Department of Computer Science, University of York, 1995. Available as Technical Report YCST 96/03.
18. J. Rushby. Using model checking to help discover mode confusions and other automation surprises. *Reliability Engineering and System Safety*, 75(2):167-177, Fevereiro 2002.
19. M. Ryan, J. Fiadeiro, and T. Maibaum. Sharing actions and attributes in modal action logic. In T. Ito e A. R. Meyer, editores, *Theoretical Aspects of Computer Software*, volume 526 of *Lecture Notes in Computer Science*, pp 569-593. Springer-Verlag, 1991.

20. H. Thimbleby. Analysis and simulation of user interfaces. In S. McDonald, Y. Waern e G. Cockton, editores, *Proc. BCS Human Computer Interaction*, volume XIV, pp 221-237, 2000.

21. R. M. Young, T. R. G. Green, and T. Simon. Programmable user models for predictive evaluation of interface designs. In K. Bice e C. Lewis, editores, *CHI'89 Proceedings*, pp 15-19. ACM Press, NY, Maio 1989.

MODELO DO TELEMÓVEL

interactor mobile

attributes

[vis] ringer: {on, off}
[vis] menu: {makecall, sendsms, dial, send, endcall, readsms, answercall, done}
state: {idle, dialling, calling, reading, writing, sending}
dialflag: {nil, call, sms}
endcallflag: {nil, make, answer}

actions

[vis] button ok cancel Next
newcall newsms giveup sent

axioms

menu
(1) menu=makecall → [Next] menu'=sendsms ∧ keep(ringer,dialflag,endcallflag,state)
(2) menu=sendsms → [Next] menu'=makecall ∧ keep(ringer,dialflag,endcallflag,state)
(3) menu ∉ {makecall,sendsms} → [Next] keep(ringer,menu,dialflag,endcallflag,state)
fazer uma chamada / enviar um SMS
(4) menu=makecall → [ok] menu'=dial ∧ dialflag'=call ∧ state'=dialling ∧ keep(ringer,endcallflag)
(5) (menu=dial ∧ dialflag=call) → [ok] menu'=endcall ∧ dialflag'=nil ∧ state'=calling ∧ endcallflag'=make ∧ keep(ringer)
(6) (menu=endcall ∧ endcallflag=make) → [ok] menu'=makecall ∧ endcallflag'=nil ∧ state'=idle ∧ keep(ringer,dialflag)
(7) [button] keep(menu,ringer,dialflag,endcallflag,state)
(8) menu=sendsms → [ok] menu'=send ∧ state'=writing ∧ keep(ringer,dialflag,endcallflag)
(9) menu=send → [ok] menu'=dial ∧ dialflag'=sms ∧ state'=dialling ∧ keep(ringer,endcallflag)
(10) (menu=dial ∧ dialflag=sms) → [ok] menu'=makecall ∧ dialflag'=nil ∧ state'=sending ∧ keep(ringer,endcallflag)
(11) per(sent) → state=sending
(12) state=sending → obl(sent)
(13) [sent] state'=idle → keep(ringer,menu,dialflag,endcallflag)
receber uma chamada/SMS
(14) per(newcall) → ringer≠on
(15) [newcall] ringer'=on → menu'=answercall ∧ keep(dialflag,endcallflag,state)
(16) menu=answercall → [ok] menu'=endcall ∧ endcallflag'=answer ∧ ringer'=off ∧ state'=calling ∧ keep(dialflag)
(17) (menu=endcall ∧ endcallflag=answer) → [ok] menu' ∈ {makecall, readsms} ∧ endcallflag'=nil ∧ state'=idle ∧ keep(ringer,dialflag)
(18) [newsms] ringer'=on ∧ menu'=readsms ∧ keep(dialflag,endcallflag,state)
(19) menu=readsms → [ok] menu'=done ∧ ringer'=off ∧ state'=reading ∧ keep(dialflag,endcallflag)
(20) menu=done → [ok] menu' ∈ {makecall, readsms} ∧ state'=idle ∧ keep(ringer, dialflag,endcallflag)
toque
(21) per(giveup) → ringer=on
(22) menu=answercall → [giveup] ringer'=off ∧ menu' ∈ {makecall, readsms} ∧ state'=idle ∧ keep(dialflag,endcallflag)
(23) menu=readsms → [giveup] ringer'=off ∧ keep(menu,dialflag,endcallflag,state)
cancelar
(24) [cancel] ringer'=off ∧ menu'=makecall ∧ dialflag'=nil ∧ endcallflag'=nil ∧ state'=idle
estadi inicial
(25) [] ringer=off ∧ menu=makecall ∧ dialflag=nil ∧ endcallflag=nil ∧ state=id1