IVY Trace Visualiser

Nuno Miguel Eira de Sousa José Creissac Campos
Departamento de Informática/CCTC
Universidade do Minho, Braga, Portugal
nunolesi@sapo.pt/josé.campos@di.uminho.pt

Sumário
No contexto do projecto IVY, tem vindo a ser desenvolvida uma ferramenta de modelação e análise de sistemas interactivos, tendo em vista a detecção de potenciais problemas de usabilidade no início do desenvolvimento de um qualquer sistema interactivo. Quando uma dada propriedade em análise não se verifica, a ferramenta procura indicar um contra-exemplo: um comportamento do modelo que demonstre a falsidade da propriedade em questão.

Estes contra-exemplos, no entanto, podem atingir tamanhos consideráveis, dependendo da complexidade do modelo, o que dificulta a sua análise. De forma a facilitar essa análise, a arquitectura da ferramenta IVY prevê um componente de suporte à análise. Este componente visa, através de representações visuais e de mecanismos de análise, facilitar a compreensão dos contra exemplos, de forma a tornar mais claro qual o problema que está a ser apontado e possíveis soluções para o mesmo.

Este artigo apresenta o componente de análise da ferramenta IVY. São apresentadas a arquitectura do componente, as representações implementadas e os mecanismos de análise disponibilizados.

Palavras-chave
Avaliação de sistemas interactivos, ferramentas de análise, *model cheking*, representações visuais.

1. INTRODUÇÃO

No desenvolvimento de sistemas interactivos cruzam-se as áreas da Interacção Humano-Computador (IHC) e da Engenharia de Software. Estudos mostram que o sucesso de tais sistemas depende, em grande medida, da sua usabilidade. A norma ISO DIS 9241-11 identifica 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 haver com a possibilidade (ou não) de os utilizadores poderem atingir os seus objectivos utilizando o sistema num dado contexto. Eficiência tem a ver com o maior ou menor esforço que os utilizadores terão de despender para atingir esses objectivos. Satisfação é uma medida subjectiva do grau de agradabilidade na utilização do sistema.

O *Software Engineering Book of Knowledge* [SWEBOK01] considera a concepção de interfaces com o utilizador como uma área relacionada (mas distinta) da Engenharia do Software, não mencionando a área da IHC. Torna-se portanto necessário recorrer a peritos na área da IHC para ter em consideração factores de usabilidade no desenvolvimento de sistemas interactivos. Na prática, as áreas da Engenharia de Software e IHC têm vivido relativamente afastadas. Torna-se assim necessário procurar estabelecer pontes de contacto entre as duas áreas.

No contexto do projecto IVY (A model based development environment – POSC/EIA/56646/2004) procura-se desenvolver técnicas e ferramentas que facilitem a incorporação de preocupações de usabilidade no desenvolvimento de software, promovendo a comunicação entre as duas comunidades. A abordagem seguida baseia-se na utilização de modelos para capturar aspectos relevantes dos sistemas interactivos e pretende possibilitar aos engenheiros de software uma maior autonomia na consideração de questões de usabilidade relacionadas com o comportamento do sistema, bem como identificar os pontos em que é necessário recorrer ao auxílio de peritos em IHC. O principal objectivo do projecto é o desenvolvimento de uma ferramenta de modelação e análise, tendo em vista a detecção de potenciais problemas de usabilidade no início do desenvolvimento de um qualquer sistema interactivo.

Os modelos são obtidos quer através de um processo de modelação baseado num editor, quer utilizando engenharia reversa no código da interface com o utilizador. A ferramenta visa permitir a verificação automática de modelos de sistemas interactivos, recorrendo ao *model-checker* SMV [McMillan93] para a análise do comportamento dos modelos.

Tal como será referido na secção 2, uma etapa importante do processo é a análise dos contra-exemplos produzidos pela ferramenta quando o teste de uma dada propriedade falha. Neste artigo apresenta-se a arquitectura e modo de funcionamento do componente responsável pelo suporte a este processo de análise. Na secção 2 apresentam-se alguns detalhes sobre a abordagem suportada pelo IVY, fornecendo deste modo contexto à discussão que se seguirá. Na secção 3 apresenta-se a arquitectura do componente de visualização. Na secção 4 apresentam-se

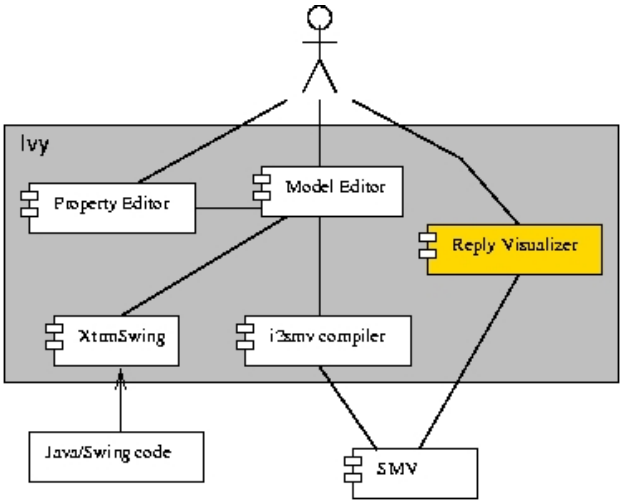

Figura 1. Arquitectura da ferramenta IVY

as diferentes representações visuais que têm vindo a ser exploradas. Na secção 5 apresentam-se os mecanismos de filtragem emarcação que permitem a exploração dos traços de comportamento. Na secção 6 apresenta-se um exemplo de aplicação da ferramenta. Finalmente, na secção 7 faz-se um resumo do trabalho realizado e lançam-se alguns apontadores para trabalho futuro.

2. IVY

A arquitectura da ferramenta é apresentada na figura 1. Tal como já descrito, o utilizador do IVY cria modelos quer através de um editor dedicado (model editor), quer através de um componente de engenharia reversa (XtrmSwing), quer ainda através de uma combinação das duas abordagens.

Os modelos são escritos recorrendo à linguagem MAL Interactors [Campos01, Campos04]. Um *interactor* [Duke93] pode ser visto como um objecto capaz de representar (parte d)o seu estado num qualquer meio de apresentação. Assim, um *interactor* possui um conjunto de atributos tipados que definem o seu estado e um conjunto de acções que definem as mensagens que o interactor pode receber e/ou enviar. No caso dos MAL interactors, o comportamento do *interactor* em reacção à mensagens recebidas é definido utilizando axiomas escritos na lógica MAL (*Modal-Action Logic*) [Ryan91]. Estes axiomas definem a semântica das acções em termos do seu efeito no estado do *interactor*.

Para a discussão que se segue, basta perceber que um modelo é construído compondo *interactors* de forma hierárquica. Deste modo, pode sempre ser representado por uma máquina de estados em que os estados são definidos pelos valores dos atributos e as transições são etiquetadas pelas acções que provocam as alterações nos atributos. A fase de verificação consiste em testar propriedades do comportamento dessa máquina de estados.

As propriedades para verificação são escritas em CTL (*Computational Tree Logic*) [Clarke86]. Para facilitar o processo de verificação, a ferramenta IVY disponibilizará um editor de propriedades (property eitor), tendo em vista

```
Trace Description: CTL Counterexample
Trace Type: Counterexample
-> State: 1.1 <-
  others.action = nil
  others.visible = 0
  others.newinfo = 0
  others.mapped = 0
  mail.action = nil
  mail.visible = 0
  mail.newinfo = 0
  mail.mapped = 0
  action = nil
-> State: 1.2 <-
  others.action = map
  others.visible = 1
  others.mapped = 1
  mail.action = update
  mail.newinfo = 1
```

Figura 2. Traço de comportamento

permitir a definição de propriedades a verificar se ser necessário conhecer a linguagem lógica subjacente.

Para mais detalhes sobre a aplicação da lógica CTL no presente contexto ver [Campos01]. Para a discussão que se segue o importante é saber que, quando uma dada propriedade não se verifica, o SMV procura fornecer um traço de comportamento que demonstre a falsidade da propriedade em questão (ver figura 2 para um exemplo). Esse traço consiste numa sequência de estados da máquina que violam a propriedade.

Para possibilitar a sua verificação, os modelos MAL interactors são compilados para a linguagem do SMV (i2smv compiler). Uma vez que a expressividade da linguagem do SMV é limitada, quando comparada com a linguagem dos MAL interactors, o processo de compilação para SMV introduz uma séria de variáveis auxiliares. Merece particular relevo a introdução de um atributo 'action', utilizado para modelar as acções, uma vez que o SMV não contempla esse conceito.

Para a discussão que se seguirá, um outro aspecto a merecer relevo tem a ver com os modelos de execução. Ao nível dos MAL interactors as acções dos diferentes interactors podem acontecer de forma assíncrona. Assim, um interactor pode executar uma acção enquanto os restantes se mantém inactivos. Ao nível do SMV, no entanto, as transições de estado ocorrem de forma síncrona. Para modelar transições de estado assíncronas, torna-se necessário introduzir uma acção especial *nil* que ao nível dos MAL interactors (nível lógico) corresponde a nada acontecer, mas ao nível do SMV (nível físico) representa uma transição de estado (para um estado com os mesmos valores dos atributos). Deste modo, o módulo SMV correspondente a um interactor pode sofrer uma transição de estado associada a uma dada acção, enquanto os res-

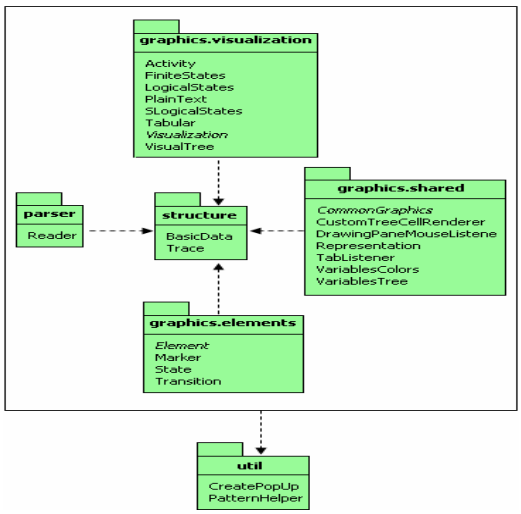

Figura 3. Arquitectura do visualizador

os restantes executam a transição associada a *nil* (ou seja, mantém o estado).

Os traços produzidos pelo processo de verificação fazem, tal como seria de esperar, referência às variáveis e estados existentes ao nível do código SMV. Assim, torna-se necessário reverter este processo para 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. De forma a facilitar a análise dos traços, a arquitectura da ferramenta IVY prevê um componente de análise de traços (reply visualizer – ver figura 1). Este componente de visualização visa, através de uma representação visual e de mecanismos de análise dos traços, facilitar a sua compreensão de forma a tornar mais claro qual o problema que está a ser apontado e possíveis soluções para o mesmo.

3. O VISUALIZADOR

Tal como já referido, o componente de visualização é responsável pela apresentação dos resultados do processo de verificação, tendo três responsabilidades principais:

- pré-processar os traços produzidos pelo SMV de forma a torná-los consistentes com a notação utilizada na escrita do modelo;
- fornecer representações visuais dos traços de forma a facilitar a sua compreensão;
- fornecer um mecanismo de análise dos traços.

A arquitectura do visualizador consiste em três componentes principais: Parser, Structure e Graphics [Sousa06] e um secundário: Util (ver figura 3).

O componente Parser é responsável por fazer o *parsing* do ficheiro do traço.

O componente Structure é responsável por guardar a informação, resultante do processo de *parsing* do ficheiro do traço, numa estrutura de dados.

O componente Graphics é responsável por visualizar a informação obtida pelos outros componentes, usando para isso diferentes representações visuais (por exemplo: diagrama de actividades, diagrama de estados finitos, etc.). Este é constituído por três sub-componentes:

- Visualization: responsável pelas representações visuais dos traços;
- Shared: responsável por fornecer funcionalidades comuns a todas as representações visuais;
- Elements: responsável pela implementação dos objectos gráficos que são usados pelas representações visuais, por exemplo: Transition (que pode ser desenhado como uma seta).

O visualizador (ver figura 4) permite ter disponíveis diversas vistas sobre o mesmo traço. Cada vista dá origem a uma *tab* e para cada *tab* pode ser seleccionada a representação visual que o utilizador desejar através de uma *combobox* com as opções possíveis.

O componente Util fornece serviços que são usados pelos componentes principais do visualizador.

Na parte superior da janela do visualizador é apresentada a fórmula em análise na notação CTL. Como o traço foi gerado, a fórmula é falsa. O objectivo mais imediato que um utilizador IVY quererá concretizar, com a ajuda da ferramenta, será o de descobrir o estado (ou estados) que violam a fórmula. Depois disso, um outro objectivo será compreender as razões dessa falha, analisando os atributos dos *interactors* e as transições entre estados. Esta análise é efectuada recorrendo às funcionalidades de filtros e marcações que serão descritas mais à frente neste artigo.

4. REPRESENTAÇÕES VISUAIS

Actualmente o visualizador possui implementadas as seguintes seis representações visuais:

- *Trace* – a representação textual original produzida pelo SMV;
- *Tree* – representação em árvore dos estados do traço;
- *Tabular* – representação tabular semelhante à existente no SMV da Cadence Labs;
- *Fisical States* – representação gráfica dos estados do traço;
- *Logical States* – representação semelhante à anterior em que os estados do traço são pré-processados para eliminar estados artificiais introduzidos pelo processo de compilação;
- *Activity Diagram* – representação centrada nas acções recorrendo a diagramas de Actividade do UML 2.0 [OMG05] (para uma introdução ao UML ver [Fowler04]).

Cada uma delas será agora descrita em mais detalhe.

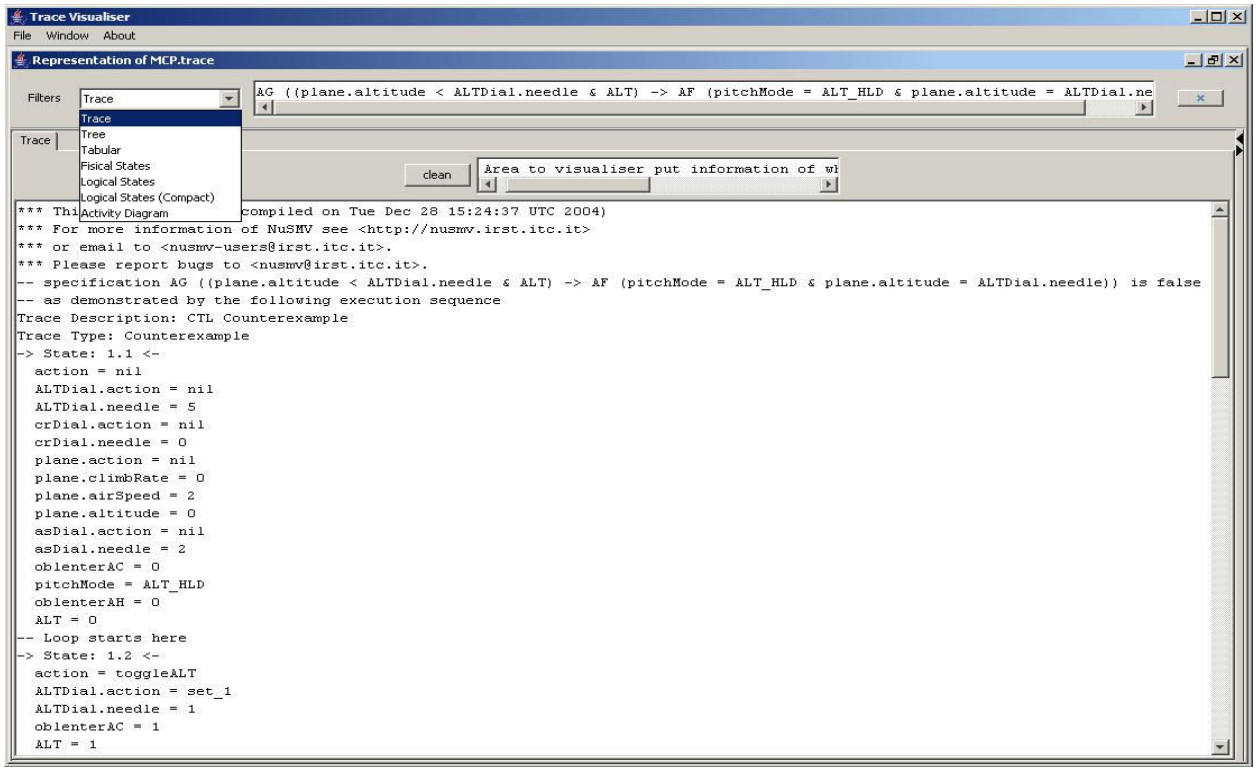

Figura 4. Interface Gráfica do Visualizador mostrando o ficheiro de trace original

4.1 Trace

A representação *Trace* (ver figura 4) corresponde ao formato textual do traço que serve de *input* ao visualizador, tal como produzido pelo SMV.

Um traço é uma sequência de estados, cada um deles definido à custa dos atributos dos *interactors* do modelo. No traço só os atributos de *interactors* que mudaram são mostrados em cada estado. Os atributos são representados na seguinte notação: <interactor>.<atributo>. Quando um atributo surge sem o prefixo do *interactor* isso significa que pertence ao interactor *main* (o interactor raiz do modelo).

Tal como referido anteriormente, o atributo 'action' modela as acções e corresponde à acção que levou o *interactor* para o estado corrente.

Antes de um estado pode existir uma indicação "Loop starts here" que indica a existência de um ciclo: chegando-se ao estado final do traço, este continua no estado em que a indicação surge.

Esta representação apresenta o traço sem qualquer tipo de pré-processamento. Assim, a sua interpretação requer alguns conhecimentos relativos ao processo de compilação. Adicionalmente, é uma representação que tende a ser extensa e difícil de analisar.

4.2 Tree

Tree é uma representação visual em forma de árvore e uma versão mais estruturada da representação anterior.

Nesta representação (ver figura 5), cada um dos nodos de topo da árvore (a seguir ao nodo root) corresponde a um estado do traço. Dentro de cada estado, os atributos e respectivos valores são agrupados, por interactor, nos nodos inferiores da árvore. Assim, o interactor *main,* implícito na representação anterior, surge aqui de uma forma explícita.

O facto de a informação dos estados ser representada numa árvore permite, comparativamente com a representação original, acrescentar novas funcionalidades, dado que o componente *JTree* (a implementação da árvore) possui uma série de características base úteis. Essas características são as seguintes:

- a possibilidade de representar relações de inclusão, agrupando folhas em nodos (neste caso as folhas são os atributos e respectivos valores e os nodos são *interactors*/estados);
- a possibilidade de expandir e colapsar nodos (neste caso, representando estados do traço ou *interactors*), permitindo ajustar em cada momento a informação que é apresentada;
- a possibilidade de fazer o *rendering* (cor do texto, cor de fundo, etc) de um nodo da árvore de forma diferenciada (neste caso os atributos com texto de cor vermelha e as acções com texto de cor azul).

A utilidade desta representação reside na possibilidade de apresentar a informação de forma agrupada e no facto de permitir seleccionar apenas a informação que se deseja, por exemplo ver só determinados estados (colapsando os outros restantes) ou ver só alguns interactors dentro de um estado.

Esta representação continua a sofrer de alguns dos problemas na representação original, pelo que poderá não

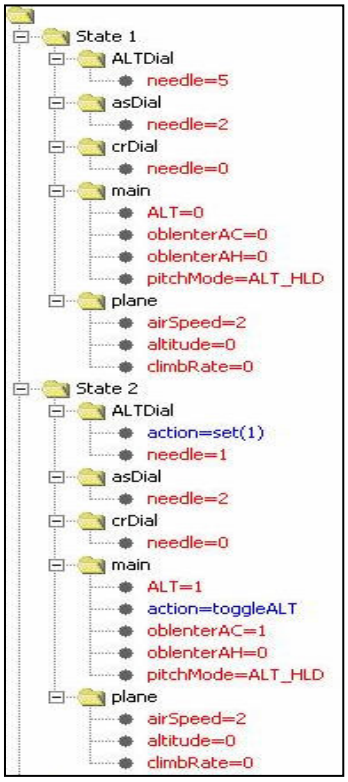

Figura 5. Representação visual Tree

ser das mais úteis na fase de análise. As principais limitações advêm do facto de ainda ser uma representação extensa e de as acções não serem representadas de um forma muito diferenciada relativamente aos atributos dos *interactors*. Por exemplo: uma acção representada por uma seta fornece uma melhor intuição de transição. No entanto, quando comparada com a representação original, a representação Tree possibilita uma visão mais estruturada do traço, pelo que poderá servir de complemento às representações visuais que serão apresentadas de seguida.

4.3 Tabular

Nesta representação (ver figura 6) a informação é apresentada numa tabela ao estilo das representações fornecidas pelo SMV da Cadence Labs ou por [Loer04]. Os cabeçalhos da coluna mostram o número dos estados. O início de um ciclo é mostrado usado um asterisco.

Uma célula com fundo a branco mostra que o valor do atributo nesse estado mudou quando comparado com o valor do atributo no estado anterior. Quando o valor se mantém de um estado para outro o fundo da célula é mostrado a cinzento. Esta ideia, adoptada de [Loer04], é muito relevante pois permite saber quando é que os *interactors* mantém o seu estado e também quando é que (ao nível lógico) estes sofrem uma transição do mesmo.

A representação tabular é uma das de mais fácil compreensão, provavelmente por ser uma das mais conhecidas e usuais. No entanto não deixa de ter um grande inconveniente: dado o elevado número de estados que os traços poderão conter, não é muito prático e eficaz estar a analisar a informação neles contida por meio de tabelas,

	1	2 *	3	4
ALTDial.action		set(1)	set(2)	
ALTDial.needle	5	1	2	2
asDial.action				
asDial.needle	2	2	2	2
crDial.action				
crDial.needle	0	0	0	0
main.ALT	0	1	1	0
main.action		toggleALT		enterAC
main.oblenterAC	0	1	1	0
main.oblenterAH	0	0	0	0
main.pitchMode	ALT_HLD	ALT_HLD	ALT_HLD	ALT_CAP
plane.action				fly
plane.airSpeed	2	2	2	5
plane.altitude	0	0	0	1
plane.climbRate	0	0	0	1

Figura 6. Representação visual Tabular

tabelas, pois rapidamente esta se torna difícil. A utilização de cor para marcar as células com alteração de informação relativamente ao estado anterior, bem como a possibilidade de utilização de filtros e marcações (ver secção 5) permitem uma expressividade adicional que ajuda a obviar este problema.

A representação tem a seu favor a compactação de informação que proporciona, permitindo uma visão global imediata de toda a informação se o número de estados não for demasiado elevado. Assim, pode ser útil na análise de traços com poucos estados, ou como complemento de outras representações no caso de traços com um número elevado de estados. Tem ainda a vantagem de ser uma representação familiar a utilizadores do *model checker*.

4.4 Fisical States

Esta representação inspira-se nos diagramas de transição de estado. Para cada *interactor* (ver figura 7) existe uma coluna mostrando os estados que o interactor atravessa ao longo do traço. O estado global, com todas as variáveis dos *interactors*, é também representado (coluna GLOBAL) para servir como índice. O índice é usado para ver em cada momento o estado global formado pelo conjunto dos estados de cada *interactor*, bem como para ver a ordem dos estados. Neste estado global também pode existir uma seta de cor verde indicando o início de um ciclo e outra seta indicando o seu fim no último estado, caso um ciclo exista no traço.

No caso das colunas relativas aos *interactors*, os atributos e os seus valores são mostrados junto ao rectângulo que representa o estado. As acções são mostradas como setas entre dois estados consecutivos com uma etiqueta com a identificação da acção. Essa seta só é mostrada se existir uma acção para o *interactor* no estado respectivo. Quando não é mostrada isso significa que a transição se ficou a dever à acção *nil* (o estado desse *interactor* não mudou, mas um outro qualquer *interactor* sofreu uma transição).

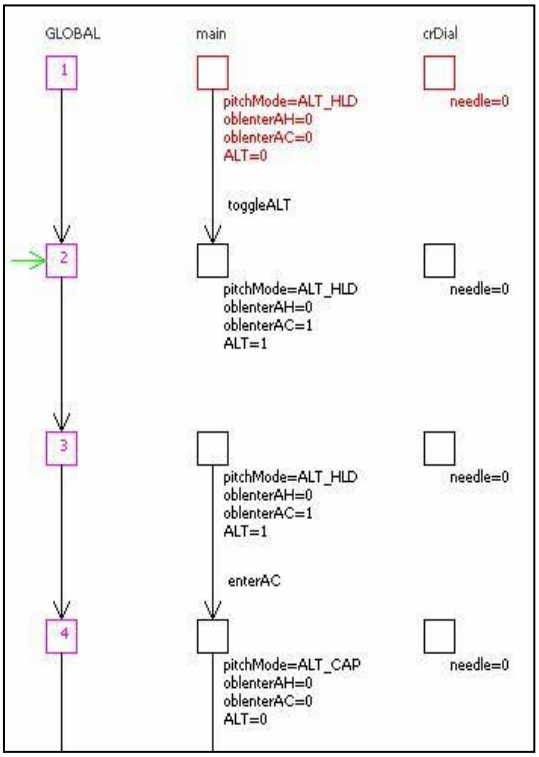

Figura 7. Representação visual Fisical States

No caso do estado global, a seta é sempre mostrada mas sem a etiqueta da acção. Os atributos não são mostrados directamente no diagrama, mas apenas quando se coloca o rato sobre um estado.

O utilizador pode decidir se pretende que a informação sobre os estados seja ou não apresentada directamente no diagrama, utilizando a funcionalidade de *popups*.

A funcionalidade dos *popups* é bastante útil porque permite seleccionar a informação que se pretende ver. Por exemplo, imaginando que se querem ver apenas as transições entre estados contidas no traço, só é necessário activar os *popups* e desta forma só são mostradas as acções. Se o utilizador quiser ver os atributos de um determinado estado basta colocar o rato dentro do rectângulo que o representa e a informação pretendida surgirá. O facto da informação não surgir toda de uma vez no ecrã (com *popups* activado) ajuda o utilizador a diminuir a quantidade da informação que é necessário analisar para descobrir o problema presente no traço.

A representação dos interactors em colunas permite separá-los uns dos outros e ver a sequência de estados de cada um de forma individual. No entanto isto levanta um problema em termos de informação redundante, dado que, em muitos casos, não existe verdadeira alteração do estado do interactor, mas sim a necessidade de transitar para outro estado por causa de uma transição ocorrida num outro interactor (ver explicação sobre estados físicos e estados lógicos na secção 2). Esta questão é eficazmente tratada noutra representação (Logical States) que irá ser descrita na próxima secção.

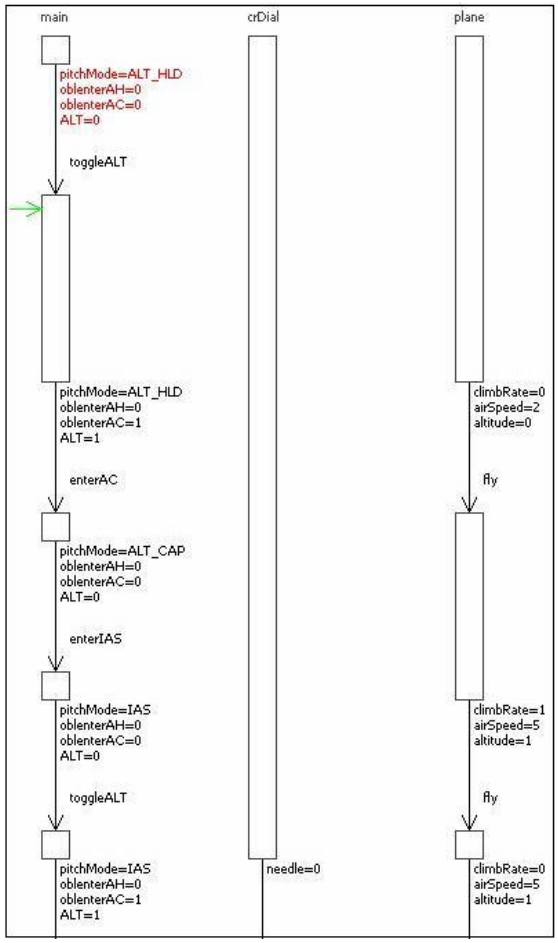

Figura 8. Representação visual Logical States

A representação Fisical States será potencialmente uma das mais úteis na análise devido à sua estruturação dos *interactors* por colunas, funcionalidades dos popups e representação das acções por setas. No entanto tem alguma redundância de informação que, tal como referido, será corrigida na representação visual Logical States.

4.5 Logical States

Esta representação (ver figura 8) é semelhante à anterior com a diferença que são representados os estados lógicos e não os estados físicos. Deste modo desaparecem sequências de estados sem transições entre eles. Em vez disso, é apresentado um único estado que "cobre" todos os estados físicos que acontecem ao nível do SMV.

Esta representação é útil para mostrar as mudanças de estado reais que acontecem no modelo, eliminando a redundância que acontecia na representação anterior. No entanto, ao fazer-se a análise do traço pode acontecer uma certa confusão quando se quer comparar *interators* ao nível de um estado. O que acontece é que o estado de um dado *interactor* pode prolongar-se por vários estados de um outro interactor. Uma vez que os valores dos atributos de um estado só são mostrados no fim do mesmo, a ideia de cada linha *do diagrama* conter o mesmo estado para todos os *interactors* fica um pouco desvirtuada em termos gráficos.

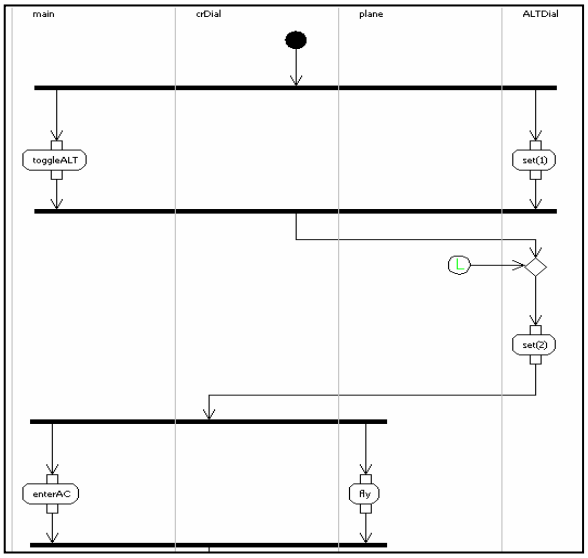

Figura 9. Representação Visual Activity Diagram

Apesar disso, graficamente esta representação tem um aspecto muito apelativo e pode fornecer inicialmente mais informação visual ao utilizador. Isto acontece devido à criação dos rectângulos grandes que eliminam a redundância da visualização do estado do *interactor*, quando na verdade este se mantém constante. Apenas se mostra o estado uma vez em vez de *n*.

Nesta representação o índice da representação anterior deixou de existir, preferindo-se mostrar o estado global como *label*.

4.6 Activity Diagram

Esta representação (ver figura 9) segue a notação da UML 2.0 para diagramas de actividade. Sendo o UML a norma para modelação orientada a objectos, é de esperar que os engenheiros de software conheçam e facilmente compreendam a notação. Adicionalmente, sendo os diagramas de actividade baseados em diagramas de fluxo de dados, é de esperar que mesmos peritos da área de IHC não tenham grande dificuldade em perceber a notação. As actividades são representadas por um rectângulo com os cantos arredondados. Os pequenos rectângulos associados às actividades representam os estados do *interactor* antes e depois das actividades ocorrerem. Os valores dos atributos podem ser consultados através de um *popup* colocando o rato sobre os rectângulos representativos dos estados.

Esta representação é bastante útil pois permite ver, de uma forma rápida, o resultado que a acção (actividade) produziu no estado de um *interactor*. Também é possível ver o estado global colocando o rato sobre a linha que liga duas actividades. A barra de sincronização de actividades permite ver as acções que são efectuadas em paralelo.

Como esta representação tem um claro foco nas acções, os atributos dos *interactors* são de certa forma escondidos, surgindo apenas como *popups,* caso essa opção opção esteja activada. Uma análise de validade de uma fórmula, com maior ênfase concentração nas transições entre estados (acções), pode utilizar esta representação pois para esse efeito é a mais aconselhada.

Esta representação pode também ser a mais aconselhada quando existem interactors com muitos atributos porque não os mostra directamente, apenas como popups, o que proporciona uma maior compactação. Pelo contrário isto também pode ser uma desvantagem porque é necessário fazer a inspecção individual dos estados dos interactors que se deseja analisar, usando o rato, o que pode tornar-se desagradável quando existem muitos estados no traço. Para minorar este problema existem as funcionalidades dos filtros e marcações que serão descritas no próximo capítulo.

Uma aproximação à representação ideal pode ser conseguida com a utilização de todas as representações descritas nesta secção. Cada uma delas tem vantagens e desvantagens, por isso é necessário usar o que cada uma tem de melhor e conjugar tudo isso para se conseguirem atingir os objectivos desejados. Neste caso, eles são: descobrir o(s) estado(s) onde a fórmula do traço não se verifica e posteriormente compreender porque é que isso acontece.

5. FILTROS E MARCAÇÕES

5.1 Filtros

De forma a facilitar a análise dos traços produzidos pelo SMV, o visualizador permite que em cada *tab* se possa aplicar um filtro que corresponde a fazer uma selecção de estados com base num determinado critério referente aos atributos dos interactors. Sempre que se aplica um filtro desse tipo é adicionado mais um *tab* com os resultados desse filtro à barra de *tabs*. Depois é possível aplicar novamente outro filtro a esse *tab* resultante (*subfiltering*) e também escolher a representação visual que se desejar. O caso limite em que o filtro já não gera mais *tabs* acontece quando o resultado do filtro associado ao *tab* só contém um estado.

Os resultados dos filtros são apresentados colorindo os elementos gráficos relevantes tais como nodos das árvores, linhas e colunas de tabelas, ou setas; em função da representação visual específica.

Na representação Tree, por exemplo, quando são usados filtros que devolvem estados, os nodos dos estados presentes no resultado são expandidos e os outros colapsados de forma a destacar só os estados pretendidos. Quando são usados filtros que devolvem variáveis de *interactors*, as células da árvore que correspondem a essas variáveis são coloridas com a cor magenta.

Adicionalmente, algumas representações visuais possuem uma funcionalidade de animação simples, por exemplo, colorindo em sequência os estados na representação visual Fisical States.

Os filtros disponíveis no visualizador são os seguintes:

1. Retornar todas as acções de um estado;

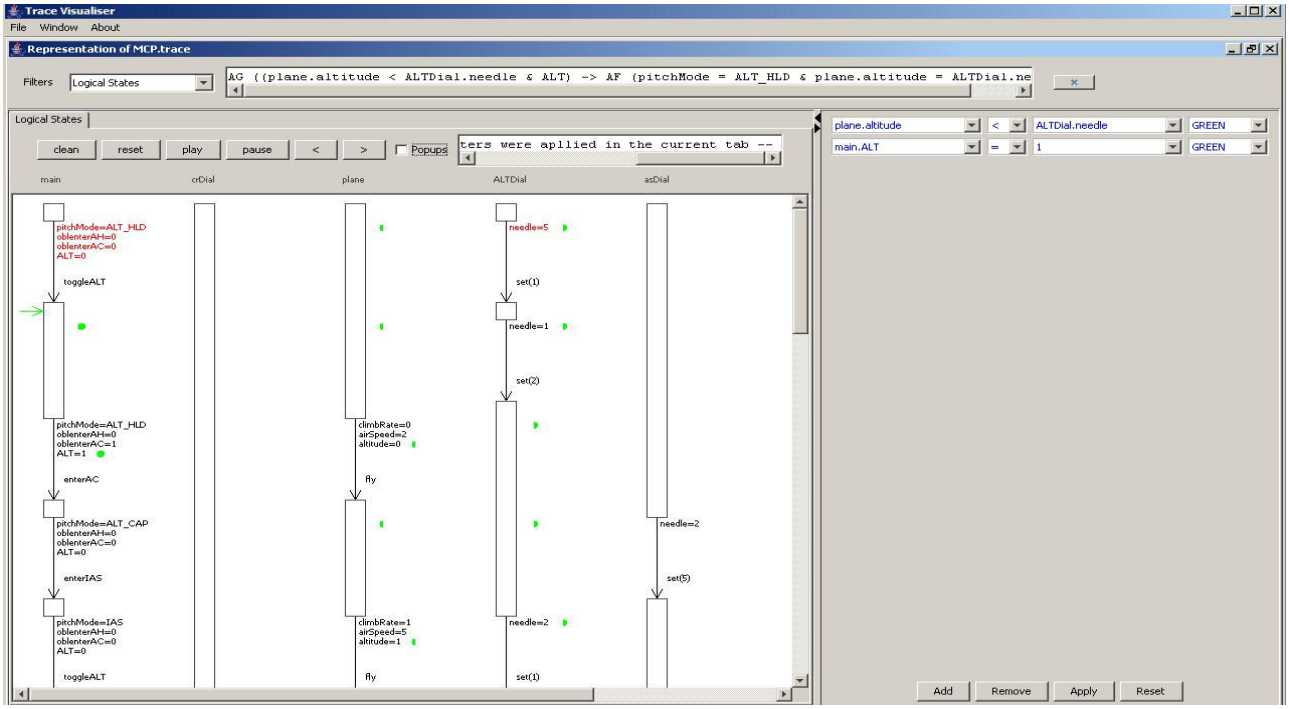

Figura 10. Análise do exemplo no diagrama de Fisical States usando marcadores

2. Retornar todos os estados onde pelo menos um atributo de uma selecção de atributos tem determinado valor;
3. Retornar todos os estados onde todos os atributos de uma selecção têm determinados valores;
4. Retornar todos os estados onde pelo menos um atributo de uma selecção de atributos mudou para um determinado valor;
5. Retornar todos os estados onde todos os atributos de uma selecção mudaram para determinados valores;
6. Retornar todos os valores dos atributos de interactors num estado;
7. Retornar todos os valores para um atributo de um interactor.

A escolha dos parâmetros para os filtros é feita através de um *Wizard*. Num primeiro painel o utilizador escolhe as variáveis dos *interactors* que pretende analisar, num segundo painel escolhe os valores que deseja para estas.

5.2 Marcadores

Existe também a possibilidade de marcar estados em função de critérios definidos sobre os atributos dos estados. Neste caso os critérios são definidos num painel presente directamente na janela principal (ver figura 10).

Os critérios são definidos estabelecendo relações (=, >,<) entre pares de atributos ou entre atributos e valores. A cada critério é associada uma cor e todos os estados que verifiquem um dado critério são anotados com a cor relativa ao mesmo.

No caso de se efectuar uma comparação entre atributos, são desenhados dois semicírculos a cheio, com a cor escolhida. Cada semicírculo é desenhado junto de cada um dos atributos, para desta forma os relacionar através da condição. No caso de comparações entre atributos e valores, são desenhados círculos a cheio com a cor escolhida. Se a opção dos *popups* estiver activada pode ser consultada a condição representada por cada marcador colocando o rato sobre o mesmo.

6. UM EXEMPLO

Para ilustrar a utilização da ferramenta iremos aplicá-la a um exemplo apresentado em [Campos01]. Trata-se da análise de um modelo do painel de controlo do piloto automático do avião MD-88.

Não sendo viável, nem relevante, apresentar aqui todo o exemplo, iremos restringir-nos à fase de análise. Após edição e compilação do modelo, uma das propriedades que se pretendeu verificar no modelo era a seguinte: sempre que o piloto automático está programado para atingir uma dada altitude, o avião acabará por voar a essa altitude em modo de manutenção da mesma. Em termos de usabilidade, esta propriedade pode ser relacionada com aspectos de previsibilidade do sistema, que influenciam o modo como os utilizadores conseguem antecipar a resposta do sistema e aprender a utilizá-lo. A propriedade pode ser expressa pela fórmula CTL:

AG(plane.altitude<AltDial.needle & ALT ->
AF(plane.altitude=AltDial.needle &
pitchmode=ALT_HLD))

(note-se que a ferramenta IVY irá disponibilizar um editor de propriedades que permitirá escreve-las sem recorrer directamente a CTL).

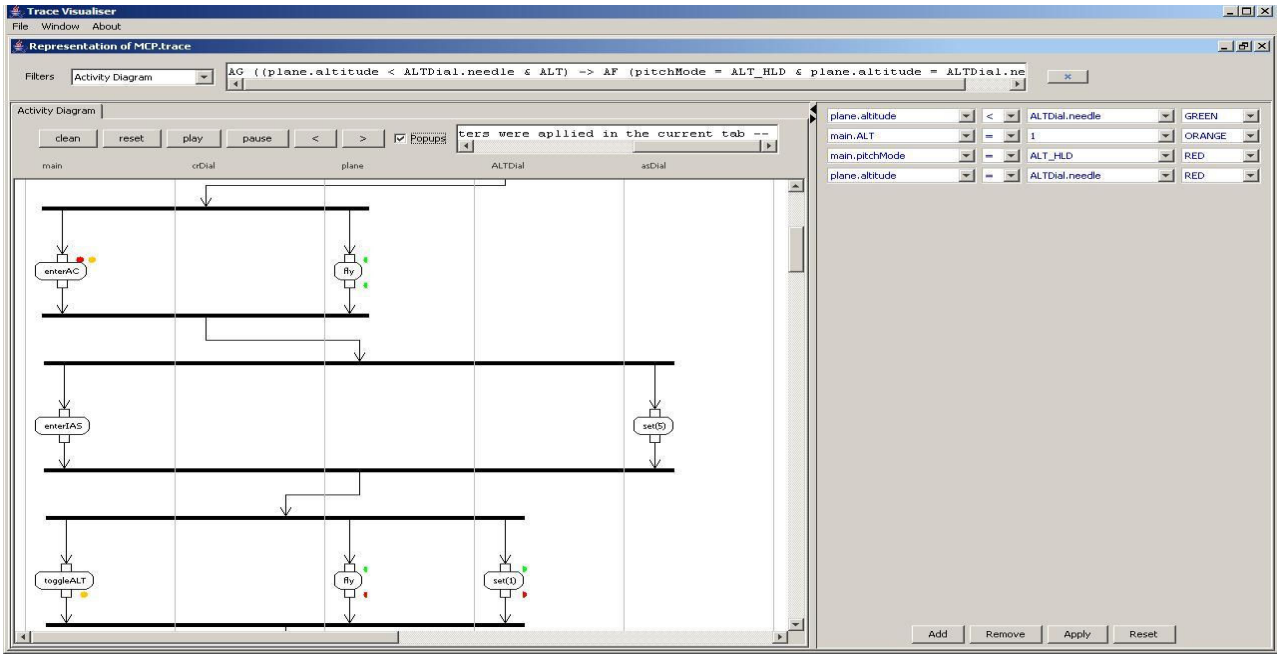

Figura 11. Análise do exemplo na representação Activity Diagrams usando marcadores

Quando se tenta a verificação da fórmula o SMV informa que ela é falsa e produz um contra exemplo (neste caso com 13 estados). Iremos agora, utilizando o visualiser, tentar perceber qual o problema que está a ser indicado pelo traço. Numa primeira fase procurou-se identificar quais os estados que verificavam o pressuposto da propriedade (avião programado para atingir uma dada altitude – plane.altitude<AltDial.needle & ALT). Optamos por criar critérios de marcação para as sub-fórmulas desta conjunção e associar-lhes a cor verde.

O resultado da aplicação da marcação mostrou que a condição passa a existir logo no segundo estado. A melhor representação para constatar este facto revelou ser a baseada em diagramas de estado. A figura 10 mostra o diagrama resultante. No estado inicial do modelo (ver topo do diagrama) apenas aparecem os semicírculos resultantes do primeiro critério (que relaciona dois atributos). Após a acção toggleALT, a marcação passa a conter não só os semicírculos do primeiro critério, como o círculo relativo ao segundo.

De seguida procurou-se confirmar que de facto um estado com o avião estabilizado à altura pretendida não acontece no traço. Criaram-se duas novas marcações para as expressões

plane.altitude=AltDial.needle

e

pitchmode=ALT_HLD

tendo-se associado a cor vermelha a ambas. Tal como seria de esperar nenhum estado ficou anotado com duas marcas vermelhas (sinal que a conjunção das duas expressões não ocorre no traço).

Finalmente, para procurar perceber a razão para o sucedido, investigou-se o atributo ALT, que modela o facto de a captura de altitude do piloto automático estar armada. Para o efeito alterou-se a cor do critério que lhe estava estava associado para laranja, de forma a distingui-lo do critério relativo à altitude.

Neste caso optou-se por um Diagrama de Actividade (ver figura 11). A análise das marcações resultantes chamou a atenção para o que se passa após o evento enterAC (evento que ocorre de modo automático e que é responsável pela activação de um modo intermédio – ALT_CAP – de aproximação final à altitude desejada). Com efeito, após esse evento, a captura de altitude (ALT) é desligada, apesar de o avião ainda estar em fase de aproximação à altitude programada (a marcação laranja desaparece, mas as marcações vermelhas não estão presentes). O que o traço monstra é que se nesse momento for alterada a velocidade vertical (evento set(5)) o piloto automático muda para modo de manutenção de velocidade vertical, perdendo-se o modo ALT_CAP e a captura de altitude. A partir desse momento é possível que o avião ultrapasse a altitude inicialmente programada no piloto automático (ver figura 11) pois o piloto automático já não está programado para parar na altitude indicada no mostrador.

7. CONCLUSÕES

Neste artigo foi apresentado componente da ferramenta IVY responsável por fornecer mecanismos de visualização e análise de traços de comportamento gerados pelo *model checker* SMV. Para além dos mecanismos de visualização, o componente fornece ainda mecanismos de análise or forma a auxiliar a identificação dos problemas encontrados pelo *model cheker*. Neste momento todas as representações visuais descritas estão funcionais.

A utilização da ferramenta tem mostrado grande potencial na facilitação da análise dos contra exemplos fornecidos pelo SMV. A funcionalidade fornecida pelos marcadores, em particular, é muito promissora quando combinada

quer com diagramas de actividade quer com as representações baseadas em estados.

Tendo em vista avaliar a utilidade de cada tipo de representação de modo mais rigoroso, está planeado um estudo com utilizadores em que se fará uma avaliação comparativa de cada uma no suporte à análise dos traços de comportamento.

O trabalho na ferramenta irá ainda continuar explorando diversos aspectos com vista a melhorar a ferramenta:

- Explorar novas representações, por exemplo para permitir animações mais elaboradas (um aspecto que aqui ficou apenas mencionado).
- Melhorar os marcadores, para que permitam condições mais complexas (por exemplo, permitindo relacionar atributos em estados diferentes).
- Melhorar a representação tabular no seguinte aspecto: na coluna dos atributos dos interactors o conceito de interactor não está claramente presente; os atributos são apresentados todos ao mesmo nível, com a estruturação em interactors indicada da forma <Interactor>.<Atributo>. Seria útil poder agrupar explicitamente os atributos por *interactor*, indicando para cada um todos os seus atributos de forma estruturada na tabela.

8. AGRADECIMENTOS

Este trabalho é suportado pela FCT (Portugal) e pelo FEDER (União Europeia) no âmbito do contrato POSC/EIA/56646/2004.

Os autores agradecem a Alexander de Ridder e Francisco Martinez Posadas o trabalho realizado em versões iniciais da ferramenta [Ridder05]. Agradecem ainda a António R. Fernandes a sugestão de criação dos marcadores.

9. REFERÊNCIAS

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

[Campos04] J. Creissac Campos, *Análise de usabilidade baseada em modelos*, Interacção 2004 – 1ª. Conferência Nacional em Interacção Pessoa-Máquina, 171-176, Grupo Português de Computação Gráfica, Julho 2004.

[Clarke86] E. M. Clarke, E. A. Emerson, 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, 1986.

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

[Fowler04] M. Fowler. *UML Distilled, third edition*. Object Technology Series, Addison-Wesley, 2004.

[Loer04] K. Loer, M. Harrison. *A Framework and Supporting Tool for the Model-based Analysis for Dependable Interactive Systems in the Context of Industrial Design*. Technical report CS-TR-873, School of Computing, University of Newcastle upon Tyne, November, 2004.

[McMillan93] K. L. McMillan. Symbolic Model Checking: An Approach to the State Explosion Problem, Kluwer Academic, 1993.

[OMG05] Object Management Group, Unified Modeling Language: Superstructure, v. 2.0. OMG specification: formal/05-07-04, Agosto, 2005.

[Ridder05] A. de Ridder, F. M. Posadas, J. C. Campos. *Technical guide for the Visualizer Component*. IVY technical report IVY-TR-5-01, Junho, 2005.

[Ryan91] M. Ryan, J. Fiadeiro, T. Maibaum. Sharing actions and attributes in modal action logic. Theoretical Aspects of Computer Science, vol. 526 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 569-593, 1991.

[Sousa06] Sousa, Nuno M.E., *IVY Trace Visualiser*, Relatório de Opção III, DI/UM, Fevereiro 2006.

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