Improving traces visualisation through layout managers

Rui Couto[†] José Creissac Campos[‡]
HASLab/INESCT TEC & Dept. of Informatics/University of Minho Portugal
{rui.couto, jose.campos}@di.uminho.pt

***Abstract*—Alloy supports reasoning about software designs in early development stages. It is composed of a modelling language and a tool that is able to find valid instances of the model. Alloy is able to produce graphical representations of analysis results, which is essential for their interpretation. In previous work we have improved the representations with the usage of layout managers. Here, we further extend that work by presenting the improvements on the approach, and by introducing a new case study to analyse the contribution of layout managers, and to support validation trough a user study.**

Alloy permite analisar especificações de software nas fases iniciais de desenvolvimento. É composto por uma linguagem de modelação e uma ferramenta capaz de encontrar instâncias de modelos. Alloy permite gerar representações gráficas com os resultados da análise, essenciais para a sua interpretação. Num trabalho anterior melhoramos as representações com o uso de gestores de layouts. Aqui, estendemos esse trabalho apresentando melhorias na abordagem, e introduzindo um novo caso de estudo, de modo a analisar as contribuições dos gestores de layouts, e também suportar a validação com um estudo.

***Keywords*—Model-driven development, Human computer interaction, Computer science**

I. Introdução

Alloy [8] é uma abordagem popular de verificação formal de software. Consiste na combinação de uma linguagem declarativa de especificação de modelos baseada em lógica de primeira ordem, e de uma ferramenta de análise dos modelos (um *model finder*). Os modelos especificados em Alloy permitem descrever *estruturas*, utilizando uma combinação de teoria de conjuntos e noções de orientação a objetos. Os estados e comportamento do modelo são definidos recorrendo a restrições (*constraints*), ou seja, expressões booleanas sobre o modelo. Embora a linguagem de modelação seja vocacionada para descrever modelos estruturais, é possível descrever comportamento, através da inclusão explícita da noção de estado nos modelos.

O Alloy tem sido utilizado com sucesso como ferramenta de verificação em diversos cenários, como, por exemplo, o nível 3 da norma ERTMS/ETCS[1] [4] ou a norma OAuth [11]. A análise é realizada recorrendo a um *SAT solver* [14], de modo a obter instâncias válidas do modelo. São suportadas duas estratégias de análise: encontrar instâncias válidas do modelo – útil, por exemplo, para a sua validação; dada uma propriedade, encontrar instâncias do modelo que a falsifiquem – útil para verificação.

Os resultados da verificação de um modelo, são cruciais na verificação formal, uma vez que suportam o processo de análise e introspeção. Consistem em instâncias concretas do modelo, descritas num formato textual. O Alloy analyzer suporta a visualização das instâncias geradas recorrendo a uma representação gráfica que se assemelha a um diagrama de instância da UML [5], onde os diferentes elementos das instâncias são representados, tal como as relações entre eles. Quando o modelo inclui uma noção de comportamento, as instâncias produzidas podem ser interpretadas como comportamentos (traços – *traces*) do modelo e a visualização ajustada para apresentar a sequência de estados, um a um.

Em [3] analisamos as limitações do visualizador tendo identificado quatro tópicos principais. Primeiro, a personalização das representações (cores, formas, etc.) é limitada a configurações pré definidas. Segundo, o algoritmo de *layout* utilizado é fixo e organiza os elementos a representar estritamente em linhas, dificultando a organização visual. Terceiro, também devido ao algoritmo de representação, não é mantida a consistência entre estados consecutivos da instância. Quarto, as transições entre estados não são animadas, o que dificulta a interpretação dos traços. Após a análise, definimos um conjunto de requisitos para colmatar estas limitações, assim como uma implementação dos mesmos recorrendo a gestores de *layouts*. O resultado foi uma abordagem concretizada na ferramenta Anima, uma ferramenta web de visualização de instâncias Alloy, com ênfase nos modelos comportamentais.

A contribuição deste artigo é uma análise de como a utilização de gestores de layouts contribui para a resolução dos problemas previamente identificados, São apresentados os resultados da avaliação empírica da solução através de um estudo com utilizadores. Para tal, é introduzido um novo caso de estudo, o conhecido problema do barqueiro. É apresentada também a arquitetura refinada da ferramenta Anima, assim como um novo mecanismo de especificação das propriedades

[†]Este trabalho é financiado por Fundos FEDER através do Programa Operacional Competitividade e Internacionalização – COMPETE 2020 no âmbito do projeto «POCI-01-0145-FEDER-006961» e por Fundos Nacionais através da FCT – Fundação para a Ciência e a Tecnologia através do projeto «UID/EEA/50014/2013».
[‡]Este trabalho é financiado por Fundos FEDER através do Programa Operacional Competitividade e Internacionalização - COMPETE 2020 e por Fundos Nacionais através da FCT - Fundação para a Ciência e a Tecnologia no âmbito do projeto POCI-01-0145-FEDER-016826. Agradecemos ao Nuno Macedo por gentilmente nos ter cedido o modelo Alloy utilizado.

[1]*European Railway Traffic Management System/European Train Control System*

visuais, através de anotação do código Alloy.

O artigo organiza-se da seguinte forma. A secção II apresenta a linguagem Alloy, o respetivo visualizador e as limitações, assim como abordagens relacionadas. A secção III apresenta a abordagem e ferramenta Anima, e a secção IV a sua validação. Por fim a secção V conclui o artigo.

II. Alloy

A. Linguagem – um exemplo ilustrativo

Os modelos escritos em Alloy, são definidos através de uma linguagem declarativa, tal como exemplificado na Listagem 1 que apresenta um excerto do conhecido puzzle do barqueiro que tem que atravessar um rio com um lobo, um ganso e uma couve, sendo que pode apenas transportar um item de cada vez. Linhas começadas por "--" são comentários e não serão consideradas por agora.

A linguagem partilha conceitos com linguagens orientadas a objetos. *Assinaturas* (`sig`) representam conjuntos de objectos (cf. classes em Java). Neste caso, na listagem são definidas assinaturas como `Margin`, `Farmer`, `Wolf`, `Goose` e `Cabbage`. As assinaturas podem ter atributos (*fields*) que as relacionam com outras assinaturas. Por exemplo, o atributo `margin` na assinatura `Object` relaciona objectos do tipo `Object` com mapeamentos de margens para estados. A palavra chave `one` na definição do mapeamento indica multiplicidade. Neste caso, a cada estado estará associada uma única margem (ou seja, `margin` é uma relação injectiva). O conceito de hierarquia entre assinaturas é expresso com a palavra reservada `extends`. Por exemplo, `Farmer` estende `Object`, logo herda o atributo `margin`.

É possível também definir predicados, asserções e factos, com *pred*, *assert* e *fact*, respetivamente. Estes são os elementos da linguagem que permitem definir as restrições ao modelo. As regras do puzzle indicam que o barqueiro pode apenas transportar um item de cada vez. Tal é definido à custa de um predicado (`move`). Por fim, existe uma hierarquia entre os itens, sendo que o lobo pode comer o ganso (se deixado sozinho com ele), e o ganso come a couve (se deixado sozinho com ela). Esta restrição é definida à custa do predicado `eat`.

Os modelos Alloy são estruturais por natureza. Como tal, a codificação de comportamento nestes modelos tem de ser feita explicitamente. Neste modelo em concreto, o comportamento (ou evolução do modelo), foi codificado recorrendo ao conceito de Estado (c.f. assinatura `State`). Como explicado acima a propriedade `margin` garante que para cada estado, a cada objecto (barqueiro, lobo, etc.) está associada uma margem.

```
1  --@Anima*(projection: State, animator: Move, spacing: 70,
2            edge.color: white, edge.direction: false,
3            edge.labels: false, background.img: river.png )
4  open util/ordering[State]
5  open util/integer
6
7  sig State {}
8  --@Anima(layout: Linear, base: root, orientation: E,
9           img: flag.png, edges: link)
10 abstract sig Margin {}
11 one sig Left, Right extends Margin {}
12
13 abstract sig Object {
14   margin : Margin one -> State
15 }
16
17 --@Anima(layout: Linear, base: margin, orientation:S,
18          img: boat.png)
19 one sig Farmer extends Object {}
20
21 abstract sig Edible extends Object {}
22
23 --@Anima(layout: Linear, base: margin, orientation:S,
24          img: wolf.png)
25 one sig Wolf extends Edible {}
26
27 --@Anima(layout: Linear, base: margin, orientation:S,
28          img: goose.png)
29 one sig Goose extends Edible {}
30
31 --@Anima(layout: Linear, base: margin, orientation:S,
32          img: cabbage.png)
33 one sig Cabbage extends Edible {}
34
35
36 pred init[p:State] {
37   all o : Object | (o.margin).p = Left
38 }
39
40 pred move[p,p':State,o:Object] {
41   (o.margin).p = (Farmer.margin).p
42
43   (Farmer.margin).p' = pass[(Farmer.margin).p]
44   (o.margin).p' = pass[(o.margin).p]
45   all o' : Edible - o | o'.margin.p = o'.margin.p'
46 }
47
48 pred alone[p,p':State] {
49   (Farmer.margin).p' = pass[(Farmer.margin).p]
50   all o' : Edible | o'.margin.p = o'.margin.p'
51 }
52
53 fun pass [m:Margin] : Margin {
54   Margin - m
55 }
56
57 fact evolution {
58   init[first]
59   all p: State - last |
60     (some o : Edible | move[p,p.next,o]) or alone[p,p.next]
61 }
62
63 pred eat[p:State] { ... }
```

Listagem 1: Excerto do puzzle do barqueiro

B. Visualizador Alloy

Através do *SAT solver* que suporta o Alloy, é possível verificar se determinadas propriedades se verificam. De modo a resolver o puzzle, podemos verificar se, começando com todos os items na margem esquerda (cf. predicado `init`), conseguimos um estado em que todos os itens se encontrem na margem direita do rio, cumprindo sempre as restrições do puzzle e, de facto, o puzzle é resolúvel, tendo a solução apresentada 8 passos (ou, *estados*).

O visualizador é um dos elementos mais relevantes da ferramenta Alloy, uma vez que permite a análise dos modelos. A representação da solução do puzzle do barqueiro (c.f. Listagem 1), tal como gerada pelo visualizador Alloy, é apresentada na Figura 1. Como é patente, a interpretação do resultado não é intuitiva. Em particular devido à forma como os estados são apresentados (toda a informação é apresentada em simultâneo).

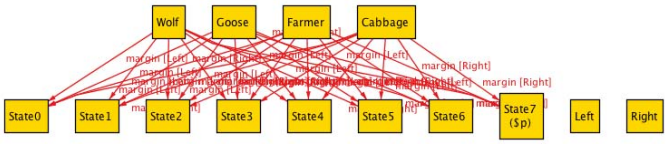
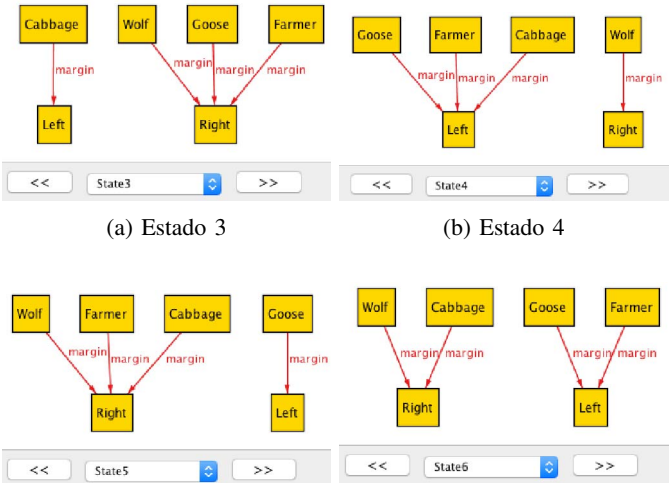


Figura 1: Solução do puzzle do barqueiro

A solução para lidar com a visualização de estados, passa pela utilização de *projeções*. Em Alloy, uma projeção permite focar a visualização numa determinada assinatura, apresentando os objetos dessa assinatura, e as suas relações, um a um. Ao projetar sobre o elemento `State`, podemos ver a evolução do modelo ao longo de diferentes estados. Na Figura 2 são apresentados quatro estados distintos do puzzle (3 ao 6). É possível ver que, por exemplo, no estado 3 o barqueiro (`Farmer`) se encontra na margem direita com o lobo (`Wolf`) e com ganso (`Goose`). No estado seguinte, o barqueiro move-se para a margem direita, "levando" consigo o ganso.



(a) Estado 3 (b) Estado 4

(c) Estado 5 (d) Estado 6

Figura 2: Projeção do solução sobre `State`

C. Limitações do visualizador

O exemplo da figura acima permite já analisar as quatro limitações mais relevantes do visualizador, neste contexto.

I) As representações geradas pelo visualizador são demasiado rígidas. A organização dos elementos é feita em linhas horizontais, de acordo com um algoritmo (fixo) pré-definido [13]. Na Figura 2 é possível ver duas linhas, sendo que uma contém quatro objetos, e a outra dois. Embora seja possível mover os objetos, isso apenas é permitido dentro da linha em que foram colocados pelo algoritmo. Não é possível que objetos sejam movidos entre diferentes linhas, muito menos organizá-los livremente na tela. Não é possível, por exemplo, colocar `Cabbage` à esquerda de `Left`, e os restantes elementos à direita de `right`, na Figura 2a, de modo a melhor representar as margens do rio.

II) Entre diferentes estados de uma projeção, não é possível fixar a posição de um determinado objeto. Recorrendo à Figura 2, não é por exemplo possível especificar que o objeto `Cabbage` deverá ser sempre o primeiro dessa linha. Este problema é agravado quando a posição de um objeto se altera de estado para estado, como por exemplo entre o estado 3 e o estado 4. É possível ver que no estado 3 a margem `Left` se encontra do lado esquerdo, mas no estado 4, a mesma margem se encontra do lado direito. A falta de consistência das representações de estado para estado dificulta a leitura e interpretação das instâncias, podendo levar até a erros de análise.

III) As diferenças entre estados consecutivos não são realçadas nem identificadas. Assim, torna-se mais difícil perceber rapidamente o que mudou entre dois estados. Por exemplo na Figura 2, entre o estado 4 e o estado 5, há uma alteração da margem à qual o barqueiro e a couve estão associados. No entanto, numa primeira análise visual não será fácil perceber essa alteração. Este problema pode ser agravado pela limitação anterior. Uma vez que a posição dos objetos não é mantida entre estados. No exemplo a margem `Left` passa a estar do lado direito no estado 5. Como tal, uma análise baseada apenas na primeira linha, pode levar a conclusão, errada, que os objetos `Wolf`, `Farmer` e `Cabbage` se encontram do lado esquerdo da margem.

IV) As capacidades de personalização das representações gráficas são algo limitadas. É possível personalizar alguns parâmetros como cor e formato dos objetos, mas apenas dentro de um conjunto pre-definido de propriedades. Por um lado, isto dificulta a interpretação do diagrama. Por exemplo, atribuir uma cor e um formato ao objeto barqueiro pode não ser suficiente para que se perceba de imediato o que o objeto representa. Por outro lado, alguns objetos podem não ter uma representação visual (e.g. margem) facilmente representável por uma imagem. Permitir uma maior costumização das propriedades poderá melhorar a legibilidade e interpretação dos diagramas. Apesar de o visualizador possuir um funcionalidade de geração automática de *layouts* (o Magic Layout), a sua aplicação teve resultados insatisfatórios, gerando uma visualização inútil para o problema, como apresentado na Figura 3

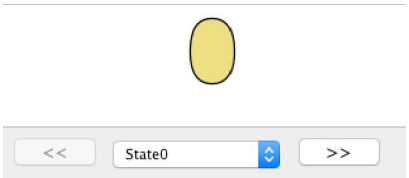


Figura 3: Aplicação do Magic Layout à solução do problema do barqueiro

Como demonstrado, o visualizador atual do Alloy apresenta algumas limitações que podem dificultar a análise de instâncias, levando, no limite, a erros de interpretação. É preciso ter em conta que estas limitações poderão, ou não, ser relevantes, dependendo do modelo e da instância sob análise.

Na próxima secção, propõe-se a aplicação de gestores de layouts como suporte à resolução destas limitações. Primeiro, no entanto, far-se-á uma breve análise de trabalho relacionado.

D. Abordagens relacionadas

A primeira abordagem para a melhoria do visualizador Alloy foi apresentada por Rayside *et al.* [12], com a implementação do *Magic Layout*, já referido acima. Esta funcionalidade (atualmente integrada na ferramenta Alloy), permite gerar temas automaticamente. Apesar das vantagens desta funcionalidade, e das melhorias propostas por Zaman *et al.* [15] de modo a colmatar alguns dos problemas conhecidos [2], esta funcionalidade continua a ser bastante limitada, como é possível ver pela Figura 3.

A utilização de representações para domínios específicos é uma das abordagens utilizadas para criar representações mais expressivas. Um exemplo é o trabalho apresentado por Gammaitoni e Kelsen [6], onde é apresentada uma metodologia para a representação de visualizações para Lightning, uma linguagem baseada em Alloy para representação de máquinas de estado. DynAlloy é outra linguagem baseada em Alloy, para a qual Bendersky e Galeotti apresentam um visualizador [1]. Electrum é uma extensão da linguagem Alloy, com suporte para representações temporais, que usa também um visualizador próprio [10].

É possível constatar, pelos trabalhos relacionados, que existem duas principais abordagem para melhorar a visualização de traços em Alloy. A primeira, consiste em melhorar o visualizador existente, de modo a conferir-lhe mais funcionalidades, e desta forma criar representações mais expressivas. A rigidez de base inerente ao visualizador continua, no entanto, presente. A segunda abordagem consiste em gerar representações para formatos específicos, o que torna as representações mais expressivas, mas limita também o seu âmbito. Apesar das contribuições oferecidas por estas abordagens, elas não resolvem os problemas previamente identificados.

III. A ferramenta Anima

A ferramenta Anima[2] foi implementada para responder às limitações identificadas acima. A ferramenta permite o *upload* de instâncias de modelos com uma componente comportamental e, com base num conjunto de especificações de *layout*, gera representações da instância, sendo possível explorar a instância, *navegando* de estado para estado. Na Figura 4 é possível ver no topo as funcionalidades de *upload* de informação, os comandos de navegação para explorar os traços, assim como a representação do estado atual no centro.

A. Representação dos estados

Na geração de representações, existe um conjunto de elementos a representar, bem como um conjunto de regras que indicam como é que os elementos devem ser dispostos. Os gestores de *layout* são um mecanismo que permite ter uma clara separação entre os elementos que estão a ser representados e a forma como a representação é feita. São, por exemplo, utilizados na programação de interfaces com o utilizador para organizar os elementos a serem exibidos [9]. Cada gestor de *layout* tem um algoritmo associado, sendo que esse algoritmo

[2]http://ise.di.uminho.pt/anima/

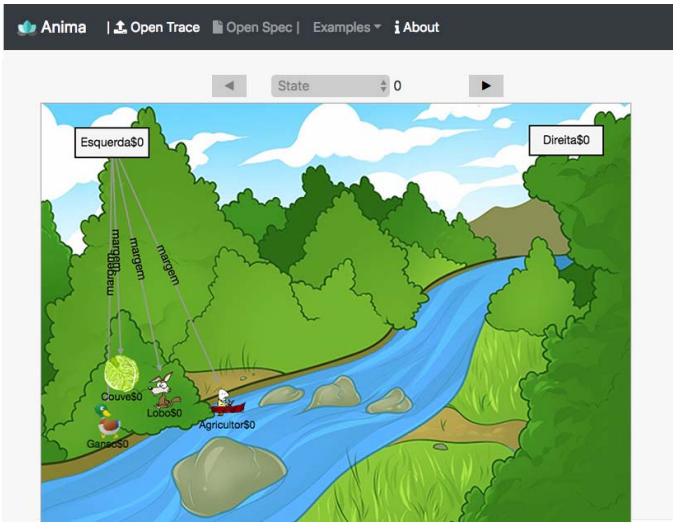
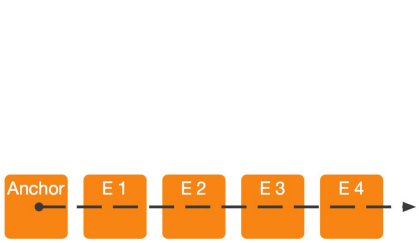


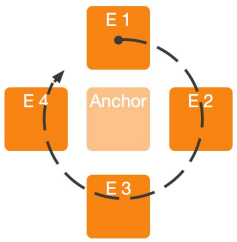
Figura 4: Ferramenta Anima

define, para esse gestor, de que forma os elementos são organizados.

Um exemplo de um gestor de *layout* suportado no Anima é apresentado na Figura 5a, o gestor de *layout* linear. Este gestor define que os elementos organizados por ele, sejam ordenados de forma sequencial, a partir de um ponto de origem (ou âncora). O algoritmo associado ao gestor indica que o primeiro elemento deverá ser colocado na âncora, e o seguinte imediatamente a seguir ao elemento anterior, e assim sucessivamente, de acordo com uma determinada orientação. Outro exemplo é o gestor de *layout* circular, apresentado na Figura 5b, onde os elementos são dispostos circularmente em volta da âncora. Neste caso, o algoritmo calcula o número de elementos, e divide o espaço disponível pelo número de elementos, dispondo um elemento em cada espaço.



(a) Gestor de layout linear



(b) Gestor de layout circular

(c) Composição de layouts

Figura 5: Gestores de layouts

É ainda possível compor os gestores de *layout* de forma hierárquica. Como tal, um elemento de um gestor pode ser um outro gestor de *layout*. Esta funcionalidade é demonstrada na Figura 5c, onde o terceiro elemento do gestor de *layout*

linear, é um gestor de *layout* circular. Esta composição dos elementos e gestores de layouts permite criar estruturas mais complexas e expressivas.

Para que a Anima possa representar uma instância, cada tipo de objetos (cada assinatura) deverá ter associado um gestor de *layout*. Neste caso específico, podem ser utilizados gestores de *layouts* lineares para representar as margens. Desta forma, elas serão sempre apresentadas na mesma ordem, solucionando um dos problemas identificados anteriormente. É possível ver na Figura 7 como a aplicação deste gestor resulta nos dois objetos estarem horizontalmente dispostos e alinhados.

Outros *layouts* lineares são utilizados para representar os elementos que se encontram em cada margem (i.e. Couve, Ganso, Lobo e Agricultor). Estes *layouts* usam como âncora a margem respetiva, tal como definida na propriedade `margin`. Assim, em cada estado, os objetos são mostrados junto à margem a que estiverem associados nesse estado.

A abordagem permite que, entre diferentes estados, sejam aplicados recorrentemente os mesmos algoritmos, o que garante a consistência da representação dos elementos. Na Figura 6 é apresentada a estrutura da solução adotada.

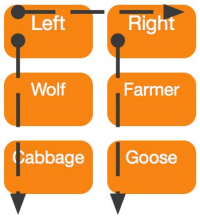


Figura 6: Organização do puzzle do barqueiro recorrendo a gestores de layouts

B. Animação

De modo a melhorar as representações gráficas, a Anima suporta animações dos elementos representados. Não só as animações tornam as transições de estado mais agradáveis visualmente, mas também permitem melhorar a interpretação dos traços. Ter a capacidade de observar visualmente as diferenças entre estados torna mais fácil o processo de análise.

Utilizando uma abordagem semelhante ao de gestores de *layout*, utilizou-se o conceito de gestor de animação. Da mesma forma que um gestor de *layout* tem um algoritmo associado, que define como os elementos são dispostos, um gestor de animações gere a forma como os elementos são tratados na transição entre estados. Um exemplo de gestor de animação, é mover o elemento entre a sua localização atual, e a sua posição no estado seguinte. No caso do exemplo apresentado, o barqueiro é movido entre as duas margens, de acordo com a sua posição num determinado estado. Outra possibilidade é apenas alterar a cor dos elementos que foram alterados numa dada transição de estado. O facto de as animações das representações serem independentes dos gestores de layouts permite que a cada gestor de layouts possam ser associados diferentes animações, ajustando a representação da evolução do modelo as necessidades de cada momento.

(a) Estado 1 (b) Estado 2

(c) Estado 3 (d) Estado 4

Figura 7: Representação dos 4 primeiros estados da solução (na ferramenta Anima)

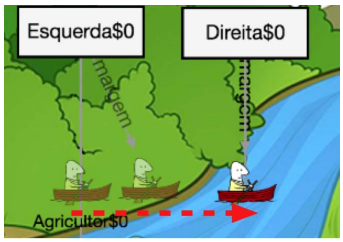


Figura 8: Animação do movimento do barqueiro

O exemplo mais básico de animação, é mover instantaneamente os elementos. Este mecanismo é similar ao implementado no visualizador padrão do Alloy. No entanto, esta abordagem dificulta a perceção das mudanças entre estados. Uma outra animação, ilustrada pela Figura 8, consiste em mover o elemento desde uma posição, até à sua posição final. Neste caso, a imagem do barqueiro é movida do lado esquerdo do ecrã, até ao lado direito (estando translucidas as posições iniciais e intermédia) num movimento contínuo. Dessa forma, acompanhando o movimento do elemento, é possível perceber a mudança que ocorreu entre dois estados.

C. Anotações

Tal como demonstrado pelo Magic Layout, determinados detalhes das representações não podem ser inferidos, é necessário que o utilizador forneça detalhes sobre os elementos a serem representados. Sem essa informação, as representações tenderiam a ser demasiado genéricas, tal como acontece no visualizador Alloy. Para tal, tornou-se necessário definir uma linguagem para descrever o layout.

De modo a evitar limitar a abordagem a uma linguagem concreta, decidiu-se criar uma linguagem de especificação intermédia. Desta forma, existe uma separação entre os elementos a representar, e os detalhes de linguagem de definição da visualização. Nesta especificação é definido, para cada elemento, de que forma ele deve ser representado. Para tal recorre-se a um ficheiro JSON que associa a cada objeto as propriedades relevantes.

O ficheiro de especificação é gerado automaticamente a partir de notações concretas de definição das representações dos objectos. A solução adoptada foi a utilização de anotações no código Alloy. Na Listagem 1, é possível ver que os elementos `Margin`, `Farmer`, `Wolf`, `Goose` e `Cabbage` se encontram anotados com um comentário na linha anterior, que inicia com `@Anima`. Isto indica que se trata de informação de visualização. De seguida, dentro de parênteses, seguem os argumentos. Por exemplo, `Farmer` está anotado com `--@Anima(layout: Linear, base: margin, orientation:S, img: boat.png)`, o que indica que os elementos desse tipo deverão ser dispostos de acordo com um layout linear, orientado a sul (`S`). Para além disso, a âncora desses elementos (`base`) é a propriedade `margin` desse objecto, ou seja, `Left` ou `Right`. Por fim, é indicado que a imagem que representa esses elementos será `boat.png`.

Existe também uma anotação global, tal como apresentado na linha 1 da listagem. Esta anotação define as propriedades globais da visualização, tais como a projeção a efetuar (c.f. `projection`), qual o gestor de animações a usar (c.f. `Move`), propriedades sobre as arestas (c.f. `edge`), ou qual a imagem de fundo a utilizar (c.f. `background`).

Através desta abordagem, é possível, recorrendo apenas ao modelo Alloy, definir de que modo a organização dos elementos será efetuada pelo novo visualizador. O Anima irá analisar as anotações e gerar a especificação intermédia em JSON. Caso não existam anotações no ficheiro, pode ser importado um ficheiro de especificação JSON independente do modelo, contendo a mesma estrutura.

D. Arquitectura/Implementação

A ferramenta foi desenvolvida com tecnologias Web, de modo a que não seja necessária a instalação de software para poder gerar visualizações. Adoptou-se uma arquitectura modular, como apresentado na Figura 9, de modo a permitir uma fácil extensão da abordagem. Os gestores de layouts e gestores de transições, `LayoutManager` e `TransitionManager` respetivamente, permitem que seja fácil acrescentar novos gestores sem necessidade de alterar o restante código.

A representação de informação e geração das visualizações também corresponde a dois componentes distintos bem definidos, nomeadamente o `Canvas` e o `Renderer`, respetivamente. Assim, é possível explorar a utilização de outras bibliotecas de visualização, actualmente é utilizada a biblioteca Cytoscape[3], sem que a representação de informação tenha de ser alterada. Da mesma forma, é possível estender as

[3]http://www.cytoscape.org, visitado em 22 de Julho de 2018.

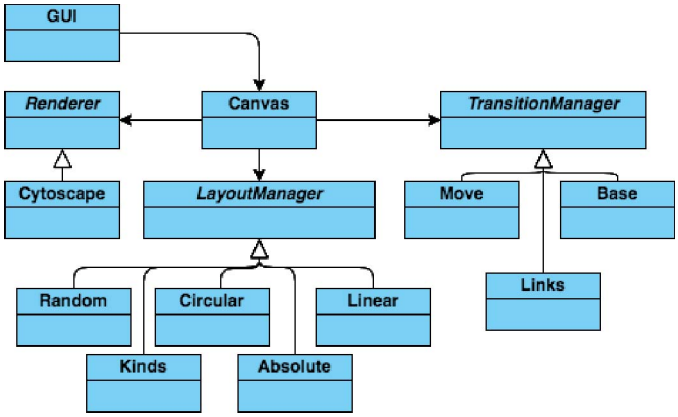


Figura 9: Arquitetura de Anima

representações e adicionar novas funcionalidades, sem que seja necessário modificar código de geração de representações já existente. Isto torna-se particularmente relevante quando considerando o suporte a especificações de representações mais antigos, à medida que a ferramenta evolui.

IV. Validação da abordagem

De um ponto de vista analítico, pode dizer-se que a abordagem implementada soluciona a limitação **I)** identificada na Secção II-C, layout pre-definido, por fornecer diversos gestores de layouts que podem ser associados com diferentes tipos de objetos. Assim, as representações são mais flexíveis. Através da utilização de gestores de layouts e dos algoritmos associados, é possível manter a consistência na posição dos objetos entre estados. Dessa forma, a limitação **II)** é resolvida. O mecanismo de gestores de animações apresentado permite implementar diferentes formas de evidenciar as alterações dos objetos entre estados. Esta abordagem resolve a limitação **III)**. Por fim, foram estendidas as capacidades de personalização das representações, com mais cores, formatos e imagens. Dessa forma, a limitação **IV)** é mitigada.

De modo a validar a abordagem apresentada de um ponto de vista empírico, foi realizado um estudo com potenciais utilizadores de Alloy. O objetivo foi perceber se as representações geradas pela ferramenta Anima permitem interpretar mais rapidamente e facilmente instâncias de modelos comportamentais. O estudo consistiu em comparar duas representações de dois traços do mesmo modelo, um deles no visualizador Alloy, outro na ferramenta Anima. O exemplo escolhido foi o puzzle do barqueiro, anteriormente descrito. Foram utilizadas as representações das Figuras 2 e 7, representação do visualizador Alloy e representação Anima, respetivamente.

A. Setup do estudo

Foi reunido um conjunto de 9 participantes, com idades compreendidas entre os 23 e os 31 anos, 8 do sexo masculino e 1 do sexo feminino. Todos os participantes tinham formação em informática. Um era detentor de um doutoramento em informática, outro de uma licenciatura em engenharia informática, e os restantes eram mestres. Dois dos participantes

indicaram ser experientes com Alloy, não tendo os restantes qualquer experiência. Nenhum dos participantes tinha contacto prévio com a nossa abordagem ou ferramenta, e todos aceitaram participar no estudo de livre vontade.

Para o estudo, criamos o modelo apresentado na Listagem 1, incluindo as anotações necessárias para o Anima (ambos já anteriormente descritos) e geramos dois traços distintos, mas de complexidade equivalente, e respetivas visualizações em Alloy e Anima. Criamos de seguida um guião onde recolhemos os dados do participante (nome, idade, habilitações literárias, género e experiência), assim como apresentamos a descrição do problema do barqueiro de forma textual. De seguida foi descrita a tarefa solicitada ao participante, que consistiu na interpretação de duas representações de traços distintos, uma gerada pelo Alloy, outra pelo Anima, e a descrição dos passos que compunham cada uma das soluções.

Cada participante fez o estudo de forma individual. Para tal, foi lhe disponibilizado um computador com as duas representações (uma para cada traço), e pedido que começasse o processo de interpretação por uma delas, sendo que metade dos participantes começaram por analisar a representação gerada pelo Alloy, e a outra metade a representação gerada pelo Anima. Não foram impostos limites temporais, mas foi medido o tempo que cada utilizador necessitou para analisar cada interpretação. Por fim, foi pedido a cada participante que respondesse ao questionário Nasa TLX [7], para cada uma das analises que fez. No final houve um processo de entrevista informal, onde foi perguntado aos participantes quais os aspetos mais positivos e negativos de cada abordagem.

B. Resultados da experiência

Durante o estudo os participantes foram independentes, não necessitando de ajuda com o processo. Foram capazes de iterar os traços de forma a analisar a solução apresentada e escrever a sua interpretação. Em relação às descrições, dois dos participantes cometeram um erro de interpretação na visualização Alloy, e descreveram erradamente uma das transições. Na ferramenta Anima, por outro lado, todas as interpretações foram corretas. Em relação ao tempo necessário para descrever os passos da solução, em Alloy os participantes necessitaram de uma média de 240 segundos, face a 117 segundos no Anima. Um Teste de Wilcoxon indicou que a diferença é significativa para $p \leq 0.05$. Os resultados parecem indicar que as interpretações em Anima demoraram cerca de 50% menos de tempo para serem interpretadas, quando comparadas com o visualizador Alloy, ao mesmo tempo que eliminaram erros de interpretação. Estes resultados dão a indicação que, não só as representações em Anima melhoram o tempo de interpretação, como também reduzem a possibilidade de erros.

Na Figura 10 são apresentados os resultados do questionário Nasa TLX. A azul, os resultados para a representação Alloy, e a vermelho para a representação Anima. É possível ver que a maior disparidade se encontra em *Mental Demand*, *Effort* e *Frustration*. Os participantes consideraram que analisar as representações Alloy é um processo mais trabalhoso, que requer mais esforço, e resulta em maior frustração. Por outro lado, *Physical Demand*, *Temporal demand* e *Performance* foram os resultados onde houve menor distância, ainda assim consideraram ter melhor resultados com a abordagem Anima.

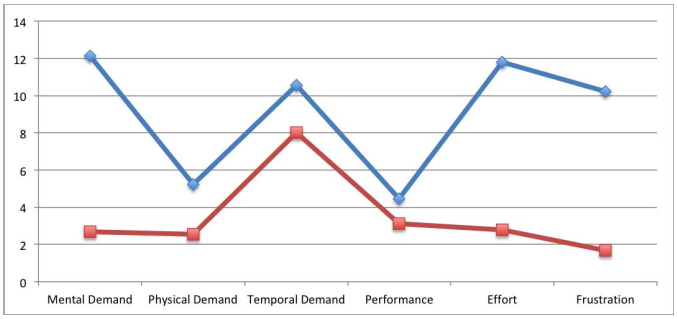


Figura 10: Resultados do questionário Nasa TLX

Na entrevista final, os participantes foram questionados sobre os aspetos positivos e negativos de cada abordagem. Em relação ao Alloy, 4 referiram como aspeto negativo a falta de consistência das posições dos objetos entre estados. De facto, a alteração da ordem das margens (c.f. Figura 2 estados 4 e 5) foi um dos pontos realçados pelos participantes. Estes comentários, por parte dos participantes, são consistentes com as observações, que indicaram ser as transições em que não se verificava consistência nas posições dos objetos, aquelas em que eles necessitaram de mais tempo para perceber a alteração ocorrida (e onde os erros de interpretação ocorreram).

Um dos participantes referiu que a visualização com caixas e setas é mais confusa, e um outro que a visualização por omissão é mais difícil de interpretar, pelas cores e formas utilizadas. Quanto à ferramenta Anima 3 participantes referiram que as representações eram mais fáceis de interpretar devido ao uso de imagens e consistência, 5 referiram que as animações facilitavam a interpretação das transições, e 2 que eram mais rápidas de interpretar. Apenas 1 participante considerou que os dois tipos de visualizações eram similares.

C. Discussão

Através deste estudo foi possível confirmar a nossa intuição acerca das representações geradas na ferramenta Anima. O estudo demonstrou que estas representações são mais fáceis de analisar, requerem menos esforço, e permitem realizar a análise de forma mais rápida. Os erros de interpretação são também reduzidos (neste caso, eliminados). Tudo isto faz com que o processo seja menos frustrante, validando a abordagem seguida na implementação da ferramenta Anima.

Relativamente ao *layout* concreto utilizado, conclui-se que facilitou a interpretação do problema, criando uma visualização com que os participantes se identificaram melhor. Fixar as margens permitiu que os participante identificassem mais rapidamente a posição dos objetos e as animações permitiram que as visualizações fossem interpretadas sem erros e sem necessidade de retroceder no traço. Algo que se revelou necessário nas visualizações geradas pelo Alloy. A utilização

de imagens mostrou ser também um fator positivo, tal como apontado pelos participantes.

Apesar dos resultados positivos, há alguns aspetos que poderão ter impacto no estudo. Em primeiro lugar, apesar de ter sido possível obter resultados estatisticamente significativos, o tamanho da amostra é ainda relativamente pequeno. Aumentar o tamanho da amostra permitirá aumentar a confiança nos resultados obtidos.

Em segundo lugar, o estudo baseia-se na interpretação de instâncias de um único modelo. Isto levanta duas questões. Por um lado, o facto de que os utilizadores estavam maioritariamente familiarizados com o puzzle em questão, pode ter feito com que em certos casos fosse mais fácil/difícil perceber o traço. Por outro lado, o puzzle do barqueiro gera soluções que são naturalmente representáveis com elementos gráficos. No entanto, para outros domínios, tal poderá não se verificar de forma tão óbvia. A análise de outros modelos é necessária para avaliar este aspeto.

Por fim, as representações visuais utilizadas podem ter tido também impacto. Por um lado, não foram utilizados temas na visualização Alloy, por outro as imagens utilizadas no Anima podem se confundir facilmente com o fundo (por exemplo, a couve e a vegetação).

Em resumo, o esforço de avaliação da ferramenta deverá continuar, não só aumentando o número de participantes do corrente estudo, mas identificando e utilizado outros casos de estudo onde para além de elementos gráficos, possam existir também a interpretação de informação (e.g. números e texto). Por fim, tanto as visualizações Alloy como as visualizações Anima utilizadas deverão ser desenhadas de modo a facilitarem a identificação dos elementos e a distinção entre tipos de elementos. No caso particular do Alloy, isso passará pela utilização de temas. Neste caso, o esforço de configuração das visualizações deverá também ser medido.

V. Conclusões

Alloy permite a descrição de sistemas através de modelos, que podem posteriormente ser analisados, resultando na geração de instâncias. Estas instâncias, podem ser graficamente visualizadas na própria ferramenta. A visualização das instâncias resultantes da análise de modelos suporta o processo de introspeção sobre os mesmos.

Em trabalho anterior identificamos quatro limitações que consideramos relevantes no processo de análise das instâncias. Em particular de instâncias de modelos com uma componente comportamental. De modo a resolver as limitações identificadas, desenvolvemos a ferramenta Anima, que integra gestores de layouts e gestores de animações para melhorar as visualizações.

Neste artigo, mostramos de que modo os problemas identificados são solucionados, e validamos a abordagem apresentada. Nove participantes interpretaram dois traços do mesmo problema recorrendo ao visualizador Alloy, e recorrendo à ferramenta Anima. Os resultados mostram que as visualizações geradas pela ferramenta Anima são interpretadas mais rapidamente quando comparadas com as geradas pelo Alloy. Para além disso, são também menos propensas a erros, sendo que em Alloy 2 participantes cometeram um erro de interpretação, face a 0 erros em Anima.

O trabalho apresentado permitiu validar a nossa intuição que os gestores de layouts permitiam melhorar as visualizações, e consecutivamente o processo de interpretação das instâncias geradas pelo Alloy.

Apesar dos resultados positivos, ou face a eles, o desenvolvimento da ferramenta Anima continua. É necessário também, não só alargar o âmbito do estudo já realizado de modo a melhor validar a qualidades das visualizações e a utilidade desta abordagem noutros contextos, como realizar novos estudos em que se avalie a dificuldade da escrita das anotações que permitem configurar as visualizações.

Referências

[1] P. Bendersky, J. P. Galeotti, and D. Garbervetsky. The dynalloy visualizer. *arXiv preprint arXiv:1401.0973*, 2014.

[2] N. Burlutskiy, M. Petridis, A. Fish, and N. Ali. Enabling the visualization for reasoning about temporal data. In *2014 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 179–180, July 2014.

[3] R. Couto, J. C. Campos, N. Macedo, and A. Cunha. Improving the visualization of alloy instances. In *Proceedings of the 4th Workshop on Formal Integrated Development Environment FIDE@FLOC 2018, Oxford, United Kingdom, 14rd July 2018.*, 2018.

[4] A. Cunha and N. Macedo. Validating the hybrid ertms/etcs level 3 concept with electrum. In M. Butler, A. Raschke, T. S. Hoang, and K. Reichl, editors, *Abstract State Machines, Alloy, B, TLA, VDM, and Z*, pages 307–321, Cham, 2018. Springer International Publishing.

[5] U. R. T. Force. Omg uml specification, 1999.

[6] L. Gammaitoni and P. Kelsen. Domain-specific visualization of alloy instances. In *International Conference on Abstract State Machines, Alloy, B, TLA, VDM, and Z*, pages 324–327. Springer, 2014.

[7] S. G. Hart and L. E. Staveland. Development of nasa-tlx (task load index): Results of empirical and theoretical research. In *Advances in psychology*, volume 52, pages 139–183. Elsevier, 1988.

[8] D. Jackson. *Software Abstractions: logic, language, and analysis*. MIT press, 2012.

[9] T. K. Kellerman and P. S. Milne. Graphical user interface layout manager, June 15 2004. US Patent 6,750,887.

[10] N. Macedo, J. Brunel, D. Chemouil, A. Cunha, and D. Kuperberg. Lightweight specification and analysis of dynamic systems with rich configurations. In *Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2016, Seattle, WA, USA, November 13-18, 2016*, pages 373–383. ACM, 2016.

[11] S. Pai, Y. Sharma, S. Kumar, R. M. Pai, and S. Singh. Formal verification of oauth 2.0 using alloy framework. In *2011 International Conference on Communication Systems and Network Technologies*, pages 655–659, June 2011.

[12] D. Rayside, F. Chang, G. Dennis, R. Seater, and D. Jackson. Automatic visualization of relational logic models. *Electronic Communications of the EASST*, 7, 2007.

[13] K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(2):109–125, Feb 1981.

[14] E. Torlak and D. Jackson. Kodkod: A relational model finder. In O. Grumberg and M. Huth, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 632–647, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[15] A. Zaman, I. Kazerani, M. Patki, B. Guntoori, and D. Rayside. Improved visualization of relational logic models. *University of Waterloo, Tech. Rep*, 2013.