CLAV - ESPECIFICAÇÃO E VERIFICAÇÃO DO MODELO FORMAL

Armando Santos and Gonçalo Duarte

University of Minho, Braga, Portugal

Resumo CLAV é uma plataforma que está a ser desenvolvida pelo Departamento de Informática da Universidade do Minho em parceria com a Direção Geral do Livro, Arquivos e Bibliotecas (DGLAB), e tem como objetivo a classificação e avaliação de todos os documentos que circulam pelas instituições públicas portuguesas. Neste momento existe um modelo do problema especificado em OWL (*Ontology Web Language*), mas tem sofrido várias alterações no decorrer do último ano e não existiu tempo para estudar o impacto dessas mesmas alterações nas pré-condições e invariantes do modelo. Neste projeto, inserido na Unidade Curricular de Laboratórios em Engenharia Informática (LEI) do MIEI/UM, pretende-se formalizar o modelo de raíz assim como os seus invariantes e garantir a consistência dos mesmos, sendo capaz de detetar erros que, até agora, não tenham sido identificados.

Keywords: Administração Pública · Métodos Formais · Engenharia Informática · Alloy Analyzer · OWL.

1 Introdução

Até agora, em Portugal, não existia nenhum sistema de informação que gerisse a classificação e a avaliação dos documentos gerados no âmbito dos processos que circulam dentro das instituições públicas portuguesas. O CLAV veio mudar isso com a elaboração de um catálogo, que se pretende que venha a ser a referência nacional de todos processos da Administração Pública (AP), tendo sido modelado numa ontologia. Esta ontologia está especificada num modelo formal que representa o conjunto de conceitos referentes aos processos de negócio e aos relacionamentos entre eles. Infelizmente, todos os dados e documentação de apoio estão espalhados, desorganizados e em diferentes formatos, o que os torna extremamente difíceis de manter num domínio tão complexo como o da AP. No entanto, embora já tenham sido feitos esforços para criar um formato neutro, como a Macro-estrutura Funcional (MEF), e vários sistemas de exploração e exportação, devido aos problemas associados com a inserção manual dos dados oriundos das diversas instituições e à complexidade e instabilidade dos invariantes e restrições associadas ao modelo não é possível garantir a coerência das relações entre os processos de negócio. Esta incoerência é extremamente crítica uma vez que a classificação e avaliação dos processos possuí legislações associadas e lida com a remoção ou conservação (digital e física) de documentos governamentais [1].

Deste modo, no contexto da unidade curricular de Laboratórios em Engenharia Informática do Mestrado Integrado em Engenharia Informática da Universidade do Minho e associado ao perfil de Métodos Formais em Engenharia Informática, apresentamos, neste artigo, a especificação e verificação, de raíz, da ontologia e o estudo da coerência dos invariantes e pré-condições que fazem parte dela. Devido à natureza puramente relacional inerente no domínio do problema em questão, iremos tirar partido de métodos algébricos e relacionais para nos ajudar a raciocinar sobre o problema em mãos e utilizar o Alloy *model checker* para nos auxiliar a encontrar falhas no desenho do modelo.

2 O Problema

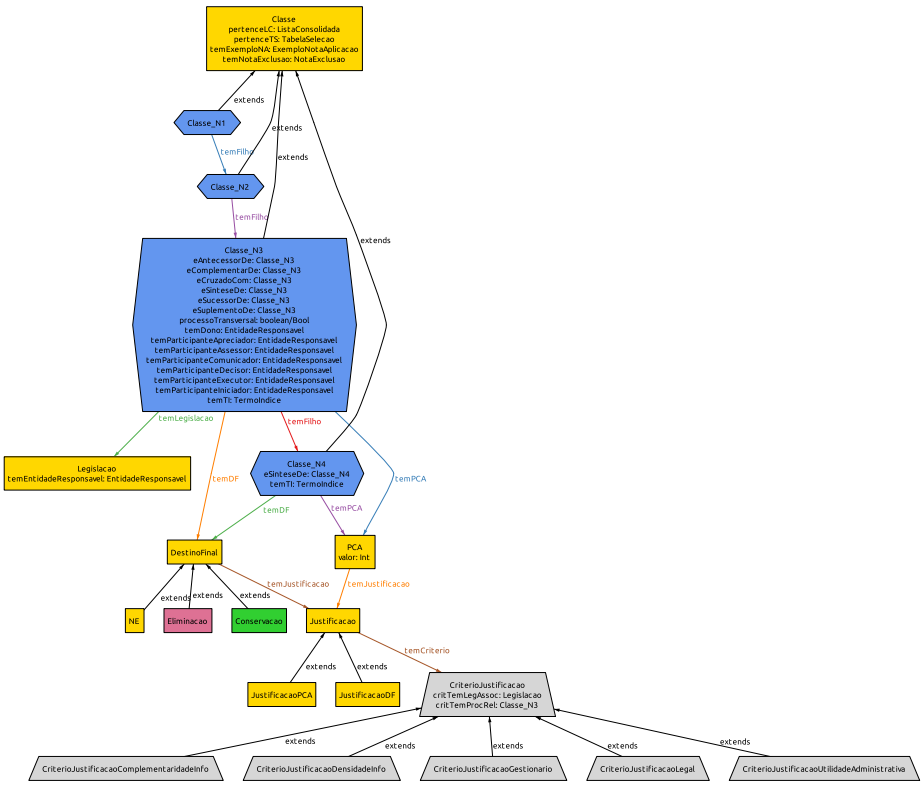


Figura 1: Meta-modelo simplificado

Cada instituição pública portuguesa desempenha uma função específica dentro da AP (p. ex. a prestação de cuidados de saúde), associado a cada função existe um conjunto de várias sub-funções (p. ex. a gestão de utentes e serviços clínicos)

e cada sub-função possui uma lista de processos de negócio que, concretamente, se materializam em documentos (p. ex. um processo de negócio pertencente à sub-função de gestão de utentes seria o registo clínico de utentes). A Lista Consolidada (LC) possui esta estrutura hierárquica de 4 níveis [4] onde os processos de negócio são passíveis de ser desdobrados para efeitos de avaliação. Cada classe da LC possui um conjunto de atributos que a descreve e a partir do 3º nível (PNs) começam a surgir relações mais complicadas entre processos no campo chamado contexto de avaliação. Este contexto de avaliação, como também iremos ver mais à frente, tem associado um conjunto de invariantes que influenciarão o campo das decisões de avaliação. Este último campo é responsável por conter a informação sobre o Prazo de Conservação Administrativa (PCA) e Destino Final (DF) de um processo que correspondem, respetivamente, ao prazo que o documento deve ser guardado e qual o seu destino uma vez que este prazo expire.

Embora as entidades principais no domínio do problema sejam as classes da LC, existem várias outras que se relacionam direta ou indiretamente com cada uma das classes e que fornecem uma maior profundidade e complexidade ao modelo como podemos observar na figura 1.

Observando o meta-modelo simplificado, onde a azul se encontram os 4 nivéis de classe, verificamos que este possui uma complexidade natural mesmo sem lhe impor alguma restrição. Sendo assim, e dado o contexto sério em que o problema está inserido, torna-se claro que deve ser feita alguma coisa no que diz respeito a dar algumas garantias acerca da coerência e consistência da LC.

2.1 Primeiro *Checkpoint*

Sendo o CLAV um projeto que já existe há 1 ano e já se encontra com alguns componentes operacionais, decidiu-se pegar em toda a documentação existente sobre o modelo e os seus requisitos e, com a ajuda do Professor José Carlos Ramalho, fazer um apanhado de todas as entidades e relações existentes. Durante esta primeira fase, bastantes das reuniões semanais serviram para apurar pequenos detalhes e dúvidas que iam surgindo.

Uma das razões que motivaram este investimento inicial, apesar de já existir uma ontologia definida pela qual nos podíamos guiar, foi a de existir muita documentação [1] [4] [5] solta e incompleta que nem sempre estava de acordo com a versão mais recente da ontologia. Foi então, elaborado um documento, atualizado, que documenta os mais recentes requisitos e invariantes e que já se tornou bastante útil no refinamento da ontologia original, nomeadamente na eliminação de entidades e relações obsoletas e no apuramento do domínio e contradomínio de várias relações.

Na Secção 3 iremos falar mais detalhadamente sobre cada entidade e relação do modelo e na secção 4 será abordada a respetiva implementação em Alloy.

2.2 Segundo *Checkpoint*

Uma das principais motivações deste projeto é estudar a forma como os mais variados invariantes interagem entre si e se, de alguma forma, se contradizem. É

neste sentido que o Alloy, sendo uma linguagem de modelação de software leve que nos permite especificar tanto o modelo como as restrições a ele associadas, ajuda a detetar erros ingénuos e subtis.

Após o investimento inicial em colecionar todo o material relevante e necessário sobre o problema, deu-se início ao ciclo de vida de verificação do problema [3]. Apesar dos invariantes serem abordados com mais detalhes na secção 3.4, é possível adiantar já que foram identificadas 38 restrições no total, sendo que 23 dessas foram acrescentadas no tal processo de verificação.

Podemos então concluir, que o Alloy teve um impacto positivo na análise das restrições do modelo e na especificação do modelo formal. É importante realçar que a utilização de um *model checker* não descarta a necessidade de prova mas é muito útil para encontrar falhas de *design* como pudemos constatar. A ausência de contra-exemplos dá uma grande confiança de que uma prova de correção está ao alcance.

2.3 Avaliação Final

Depois de validado o modelo e os seus invariantes de forma estática, isto é, apenas verificar que as restrições impostas são coerentes e que existem instâncias capazes de habitar o modelo, o percurso natural no ciclo de vida de verificação seria o de modelar a passagem do tempo e as suas ações possíveis (p. ex. a inserção de novas classes). No entanto, devido a restrições de tempo, seria mais produtivo ser capaz de observar que instâncias, presentes na base de dados, é que não iam de encontro às várias restrições. Nesse sentido, em relação à última fase do projeto foram traduzidos os invariantes em Alloy para *queries* SPARQL, motor de base de dados escolhido para armazenar toda a informação.

Devido à dimensão do modelo e quantidade de invariantes, foi colocada a hipótese de traduzir, automaticamente, as *queries* dada uma especificação.

Sendo assim, a terceira e última fase do trabalho consistiu na tradução manual dos invariantes Alloy mais específicos para *queries* SPARQL e na exploração e desenvolvimento de uma prova de conceito capaz de gerar automaticamente *query*. Estes dois últimos pontos serão explicados com maior detalhe na secção 5

3 Modelo Formal

Nesta secção será abordada a especificação formal do CLAV. Será introduzida alguma notação relativa ao cálculo relacional utilizado na formalização do problema e, posteriormente, tanto as classes presentes no domínio do modelo como as relações entre estas serão enunciadas. Por último, devido à grande quantidade de invariantes existentes, apenas serão apresentados 3, que melhor ilustram a dimensão e complexidade do CLAV. A formalização apresentada nesta secção, assim como a especificação dos invariantes em notação relacional teve o contributo e ajuda do professor José Nuno Oliveira.

3.1 Calcular com relações [3] [2]

Dentro do contexto do CLAV podemos encontrar frases do género *"A gestão de utentes é uma subfunção da prestação de cuidados de saúde"*, *"O processo de referenciação de utentes para consultas é cruzado com o processo de registo nacional de utentes"* ou *"Se um processo é complementar de outro, então o seu destino final é de conservação"*. Estas expressões podem ser interpretadas como relações tipadas entre objetos.

As relações, como as frases a cima, já existem na matemática há vários anos e possuem uma notação própria capaz de as exprimir. Em geral, a notação (infixa) $b\ R\ a$, onde a e b são os objetos e R a relação, é a que expressa mais naturalmente as relações. Esta notação aplica-se também ao uso da voz passiva, que expressa a relação inversa de R, denotada por R^o, onde $b\ R\ a$ significa o mesmo que $a\ R^o\ b$. Por exemplo, *é síntese de*^{o} = *é sintetizado por*.

Também é importante observar que relações do género R = *é o pai de*, são relações em que quando conhecido, o pai (p. ex. de uma classe) é único. Relações com esta propriedade são referidas como *simples* e satisfazem a propriedade

$$R \circ R^{\mathrm{o}} \subseteq id \tag{1}$$

onde $(\circ)$, denota a composição de relações, id é a relação identidade e $\subseteq$ é a inclusão de relações:

$$R \subseteq S \equiv \forall b; a : bRa \Rightarrow bSa. \tag{2}$$

Relações tipadas e diagramas: O uso de setas e diagramas torna possível expressar formulas relacionais mais complexas. No entanto, para ser possível representar e raciocinar sobre estes diagramas é necessário que estes estejam bem construídos.

Observando a relação (2) concluímos que apenas faz sentido se R e S forem do mesmo tipo. A notação $B \xleftarrow{\mathrm{R}} A$ declara uma relação binária que relaciona *B's* com *A's*. Por exemplo, B = *Classe nível 1* e A = *Classe nível 2* para o caso em que R = *é o pai de*.

Caminhos em diagramas são construídos a partir do encadeamento de setas, o que corresponde à composição de relações:

$$A \xleftarrow{R} B \xleftarrow{S} C \qquad\qquad b(R \circ S)c \equiv \exists a : bRa \wedge aSc \tag{3}$$

$R \circ S$

Os diagramas também advêm da comparação de caminhos, por exemplo,

$$\begin{array}{ccc} Classe\_N2 & \xleftarrow{pertenceLC2} & ListaConsolidada \\ {\scriptstyle temFilho12}\downarrow & \subseteq & \downarrow{\scriptstyle id} \\ Classe\_N1 & \xleftarrow[pertenceLC1]{} & ListaConsolidada \end{array}$$

que representa a restrição

$$temFilho12 \circ pertenceLC2 \subseteq id \circ pertenceLC1, \tag{4}$$

onde, no domínio do CLAV, as relações *pertenceLC1* e *pertenceLC2* mapeiam, respetivamente, cada classe de nível 1 na sua respetiva LC e cada classe de nível 2 na sua respetiva LC, a relação simples, *temFilho12*, relaciona uma classe de nível 1 com os seus filhos (classes de nível 2), a relação *id* é a conhecida relação identidade que relaciona cada objeto consigo próprio.

Dos diagramas à lógica [3] [2]: O que é que a expressão (4) significa em lógica proposicional?

$$temFilho12 \circ pertenceLC2 \subseteq id \circ pertenceLC1$$

$\equiv$ { inclusão de relações (2); id }

$$\forall\ c1, lc : c1\ (temFilho12 \circ pertenceLC2)\ lc \Rightarrow c1\ pertenceLC2\ lc$$

$\equiv$ { composição (3) }

$$\forall\ c1, lc : (\exists\ c2 : c1\ temFilho12\ c2\ \wedge\ c2\ pertenceLC2\ lc) \Rightarrow c1\ pertenceLC1\ lc$$

$\equiv$ { *splitting; nesting* }

$$\forall\ c1, c2, lc : c2\ pertenceLC2\ lc \wedge c1\ temFilho12\ c2\ \Rightarrow\ c1\ pertenceLC1\ lc$$

Literalmente:

Se uma c2 pertence à lista consolidada lc e c2 é filho da classe c1, então c1 também pertence à lista consolidada lc.

Ainda em menos palavras, a restrição (4), sugere:

Filho de quem pertence, também pertence.

3.2 Domínio

Com a especificação do domínio do modelo formal pretendemos descrever a essência do problema em questão que é o de garantir que, dadas as entidades existentes e as suas relações, o conjunto de invariantes impostos são coerentes, não se contradizem e permitem instâncias semanticamente corretas.

Sendo assim, nesta secção e na próxima, apesar do domínio especificado ser maior, apenas iremos apresentar as especificações que melhor capturam a essência do modelo, por questões de síntese. No entanto, será possível consultar o modelo na totalidade em anexo.

Referencial Classificativo: O Referencial Classificativo possui o conjunto de Listas Consolidadas e Tabelas de Seleção onde se encontram as diversas funções (classes de nível 1).

$$ReferencialClassificativo = ListaConsolidada + TabelaSelecao$$

Classes: Como já foi referido existem 4 níveis de classes e, como veremos na secção abaixo dá-nos jeito especificá-las da seguinte maneira.

$$Classes = Classe\_N1 + Classe\_N2 + Classe\_N3 + Classe\_N4$$

PCA: O Prazo de Conservação Administrativa é o simples tipo de dados:

$$PCA$$

Destino Final: O Destino Final é passível de ser avaliado em um de três valores: *Não Especificado (NE)*, *Eliminação* (quando o processo de negócio tem o destino de ser eliminado) e *Conservação* (quando o processo de negócio deve ser preservado). Apesar de apenas existirem estes 3 valores, foi necessário distinguilos em entidades individuais devido ao facto de cada instância de um DF estar relacionada com outras entidades. Sendo assim:

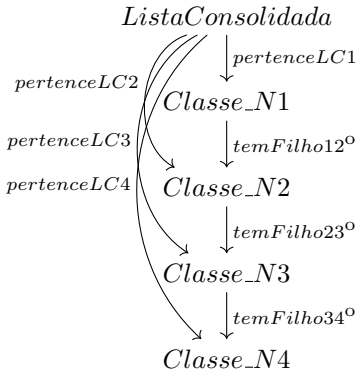
$$DestinoFinal = NE + Eliminacao + Conservacao$$

3.3 Relações envolvidas

Observando apenas as entidades definidas na secção anterior não é bem clara a existência de uma hierarquia nem é clara a existência do contexto de avaliação nas classes de nível 3. Isto deve-se ao facto de estas características apenas se evidenciarem ao analisar as relações envolvidas entre cada uma das entidades do domínio.

É de realçar que foi feito um esforço em manter as nomenclaturas da ontologia original e devido a isso, muitas das relações inversas possuem nomes diferentes. No entanto, para efeitos de simplificação e leitura utilizaremos a notação apresentada em 3.1 e as restrições de domínio/contradomínio serão esclarecidas no nome das relações.

Visão hierárquica:

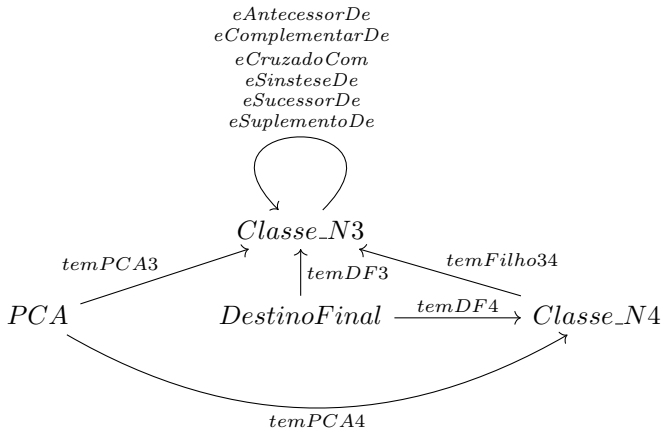


(5)

A cima, nas relações da familia *pertenceLC* o número associado a elas diz respeito ao nível da classe. Nas relações da família *temFilho*[1], o primeiro número diz respeito ao pai e o segundo ao filho. Sendo assim, com estas relações já é possível observar a cascata hierárquica das classes de processos da Administração Pública associadas a uma Lista Consolidada.

Com esta representação gráfica podemos também concluir que o sentido do invariante 4 também se aplica às relações apresentadas, dando origem a mais dois diagramas semelhantes mas para as classes filho.

Processos de negócio:



(6)

As endo-relações que dizem respeito aos processos de negócio na figura a cima são as principais relações presentes no contexto de avaliação e que irão influenciar o resultado do Destino Final. As classes de nível 4 são casos particulares em que um PN se desdobra com um certo motivo. Nesses casos as decisões de avaliação

[1] Na ontologia, a relação inversa é a *temPai*

passam para os filhos. Como podemos ver, é nos processos de negócio que se encontra uma maior densidade de relações e é também nestes que se encontra a parte mais complexa e crítica do problema e, devido a isto, a maior parte dos invariantes irá focar-se neste diagrama. É de salientar que esta figura omite entidades e relações também relevantes que lidam com legislação e critérios de justificação que poderão ser encontrados no modelo final em anexo.

3.4 Invariantes

Como já foi dado a perceber, neste projeto, apenas foi focado o domínio de relações entre cada entidade pertencente à Lista Consolidada e, dentro deste domínio reduzido, apenas foi explorado com mais profundidade a parte de classificação e avaliação de processos de negócio ignorando assim, muitas das relações de atributos e invariantes associados a estas. Como foi referido em 2.2, foram especificados, no total, 38 invariantes sendo que, 23 desses foram adicionados como resultado deste projeto.

Tentando perceber o motivo por de trás de terem sido acrescentados tantos invariantes chegamos à conclusão de que, a maior parte deles incide em invariantes que dizem respeito à taxonomia das relações, onde a injetividade é a mais comum, e os restantes provêm do trabalho realizado neste projeto onde, em conjunto com o Professor José C. Ramalho, foram discutidos e validados e dizem respeito à parte semântica/funcional do CLAV.

Injetividade: A injetividade garante que não existem duas instâncias da mesma entidade a relacionarem-se com uma outra entidade. Este invariante está presente em quase todas as relações uma vez que cada instância, principalmente ao nível dos PNs, está relacionada com legislação e critérios de justificação específicos e não faz sentido serem partilhados.

Generalizadamente, uma relação é injetiva quando:

$$ker\ R \subseteq id \tag{7}$$

Concretamente e intuitivamente, à luz de 5 e 6, uma classe não pode ter 2 pais diferentes (8) nem dois PNs (incluindo classes de nível 4) podem ter a mesma instância de DF (9):

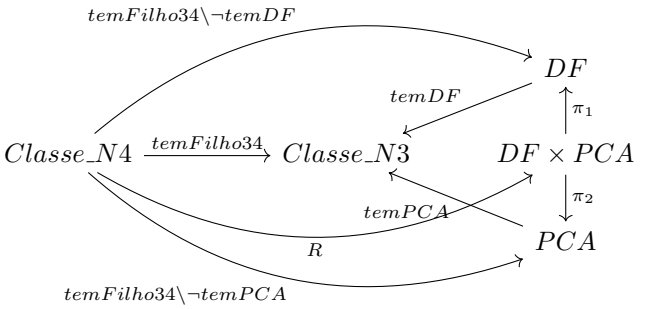
$$ker\ temFilho^{o} \subseteq id \tag{8}$$

$$ker\ temDF \subseteq id \tag{9}$$

As restrições mais complexas envolvem são as que envolvem as endo-relações dos PNs e as que envolvem desdobramento ao 4º nível. Contextualizando, apesar de não ser frequente, um processo de negócio (nível 3) é passível de ser desdobrado em dois ou mais processos filho (nível 4) sendo que, podem existir dois motivos de desdobramento: DF distinto ou PCA distinto. As classes filho resultantes de tais desdobramentos possuem critérios de avaliação e classificação

específicos que devem ser respeitados. Em baixo seguem-se 3 invariantes que caracterizam algumas das restrições impostas:

1. Se uma classe de nível 3 tem filhos, então não possui DF nem PCA (o oposto caso não tenha filhos):



(10)

$$R = \langle (temFilho34 \backslash \neg temDF), (temFilho34 \backslash \neg temPCA) \rangle$$

2. Um PN só pode ter uma (endo) relação com outro PN:

$$\begin{aligned} eAntecessorDe \subseteq \neg(&eComplementarDe \cap eCruzadoCom \\ &\cap eSinteseDe \cap eSintetizadoPor \\ &\cap eSucessorDe \cap eSuplementoDe \\ &\cap eSuplementoPara) \end{aligned} \tag{11}$$

   A disjunção desta restrição com mais 7 do género (permutações entre o lado esquerdo e direito) formam a especificação da restrição completa.
3. Se um PN é complementar ou sintese de outro, então o seu DF é de conservação; se é sintetizado por outro, então o seu DF é de eliminação; caso não seja nenhum dos especificados, o seu DF é NE (não especificado).
   Dividindo em várias cláusulas obtemos e assumindo que o PN não tem filhos:

$$eComplementarDe \backslash (temDF \circ \underline{Conservacao}) \tag{12}$$

$$(\neg eComplementarDe \cap eSinteseDe) \backslash (temDF \circ \underline{Conservacao}) \tag{13}$$

$$\begin{aligned} &(\neg eComplementarDe \cap \neg eSinteseDe \cap eSintetizadoPor) \\ &\backslash (temDF \circ \underline{Eliminacao}) \end{aligned} \tag{14}$$

$$\begin{aligned} &(\neg eComplementarDe \cap \neg eSinteseDe \cap \neg eSintetizadoPor) \\ &\backslash (temDF \circ \underline{NE}) \end{aligned} \tag{15}$$

4 Modelação em ALLOY

A partir do momento em que a formalização está completa resta-nos verificar se, dados os invariantes e a sua conjunção, existem instâncias capazes de habitarem o modelo. Uma solução seria, aleatoriamente e manualmente, popular os conjuntos de cada entidade e de forma *ad hoc* e dispendiosa, verificar se aquele conjunto poderia habitar o modelo. Outra solução bastante mais eficaz será a de recorrer a um *model checker* que, de forma automática, faz essa geração e verificação por nós. As ferramentas de *model checking* tornam o ciclo de vida de verificação muito mais produtivo.

É aqui que o Alloy Analyzer tem impacto e ajuda a perceber que tipo de refinamento deve ser dado ao modelo formalizado e de que maneira é que os invariantes têm impacto nas instâncias possíveis. Nesta secção será abordada a maneira como foi feita a tradução do modelo formal, assim como os seus invariantes para Alloy.

4.1 Especificação

Começando pelo domínio do modelo, a cada entidade corresponderá uma *sig* em Alloy. A noção de herança, capturada pelas Classes, Referencial Classificativo e Destino Final é traduzida pelas primitivas *abstract* e *extends*, inspiradas pela programação orientada a objetos. Sendo Assim:

```
abstract sig ReferencialClassificativo {}
sig ListaConsolidada extends ReferencialClassificativo {}
sig TabelaSelecao extends ReferencialClassificativo {}

/* ---- */

abstract sig Classe {}

sig Classe_N1 extends Classe {}
sig Classe_N2 extends Classe {}
sig Classe_N3 extends Classe {}
sig Classe_N4 extends Classe {}


/* ---- */

abstract sig DestinoFinal {
    temJustificacao: one Justificacao
}
sig Eliminacao, Conservacao, NE extends DestinoFinal {}
```

No que diz respeito às relações envolvidas, o Alloy permite especificar as relações entre cada entidade da seguinte forma:

```
sig Classe_N1 extends Classe {
    temFilho: set Classe_N2,
}
```

Como podemos ver, está definida uma relação $Classe\_N2 \xleftarrow{\text{temFilho}} Classe\_N1$ com uma cardinalidade *set* no contradomínio, ou seja, $temFilho$ é uma relação de 0 para muitos. Mais concretamente, a relação especificada é a relação $temFilho12$

como podemos concluir olhando para o domínio e contradomínio. Uma última nota em relação à forma como foi traduzido o modelo, é a de que apesar de ser mais intuitivo na notação relacional representar uma relação como yRx, onde $Y \xleftarrow{R} X$, em Alloy é mais intuitivo modelar da forma xRy. Mesmo a composição relacional é feita da esquerda para a direita, ao contrário da notação relacional.

4.2 Verificação

O Alloy permite especificar expressões em lógica de primeira ordem. Para isso, disponibiliza as palavras reservadas *all* e *some* para quantificar universalmente e existencialmente, respetivamente, variáveis tipadas no universo modelado.

Uma das vantagens de utilizar a ferramenta Alloy é a de tudo poder ser visto como uma relação o que ajuda na tradução do modelo em si, como vimos na secção anterior, mas também permite fazer uma tradução praticamente direta dos invariantes em notação relacional *point wise*, com o auxilio dos quantificadores, ou *point free*, com o auxilio da composição de relações.

Com isto, e pela mesma ordem, segue-se a tradução dos invariantes apresentados em 3.4:

1.
```
all c: Classe_N3 |
    some c.temFilho ⟹ no c.temDF and no c.temPCA
```

2.
```
all c1,c2: Classe_N3 |
    c1->c2 in eAntecessorDe
        ⟹ c1->c2 not in eComplementarDe + eCruzadoCom
                      + eSinteseDe + eSintetizadoPor
                      + eSucessorDe + eSuplementoDe
                      + eSuplementoPara
```

3.
```
all c: Classe_N3 | no c.temFilho ⟹ {
    some c.eComplementarDe ⟹ c.temDF in Conservacao
}
all c: Classe_N3 | no c.temFilho ⟹ {
    !(some c.eComplementarDe) and (some c.eSinteseDe)
        ⟹ c.temDF in Conservacao
}
all c: Classe_N3 | no c.temFilho ⟹ {
    !(some c.eComplementarDe) and !(some c.eSinteseDe)
    and (some c.eSintetizadoPor)
        ⟹ c.temDF in Eliminacao
}
all c: Classe_N3 | no c.temFilho ⟹ {
    !(some c.eComplementarDe) and !(some c.eSinteseDe)
    and !(some c.eSintetizadoPor)
        ⟹ c.temDF in NE
}
```

5 Tradução de *queries* SPARQL

Um modelo formal em Alloy permite raciocinar, utilizando cálculo relacionar e métodos matemáticos, sobre o problema e, tirando partido do *model checker*,

eliminar erros de conceção ingénuos ou que possam ter passado despercebidos. O modelo apresentado nas secções anteriores é apenas um modelo estático sendo que sabemos apenas, com alguma certeza, de que este é passível de ser habitado. No entanto, para que este evolua, a partir de um estado inicial, é necessária a execução de um conjunto de operações. Entre outras, estas operações correspondem a ações de inserção, atualização, remoção de instâncias no modelo e não devem permitir que as restrições impostas sejam , isto é, devem preservar os invariantes.

Um próximo passo na especificação do modelo formal do CLAV seria o de especificar tais ações, estudando assim a passagem do tempo e que tipo de propriedades, tanto de *safety* como de *liveness*, seriam verificadas. Mas, devido ao estado avaçado do projeto, um passo alternativo, e mais produtivo uma vez que já se encontra em produção, é o de a partir da especificação dos invariantes, traduzi-los em *queries* SPARQL que apanhem as instâncias da base de dados que não verificam as restrições impostas. Desta forma, será fácil identificar as anomalias no sistema e resolvê-las.

A tradução dos invariantes poderá ser feita de duas formas: manualmente ou automaticamente. Nas secções abaixo será explicada de que forma é que ambas estas formas de tradução podem ser executadas.

5.1 Tradução manual

Numa primeira vista, a sintaxe de uma *query* SPARQL, sendo este motor orientado a grafos, é bastante semelhante à de um motor de bases de dados relacional como podemos ver em baixo.

```
PREFIX P: <URI>

SELECT ?Property1 ?Property2

WHERE {
    ?v1 P:rel1 ?Property1 .
    ?v1 P:rel2 ?Property2 .
}
```

Resumidamente, e como podemos observar, o processador SPARQL funciona com *pattern matching* de triplos RDF. Baseado em lógica de primeira ordem e teoria de conjuntos, suporta operadores como a união e interseção, que permitem uma manipulação dos resultados de forma bastante simples e intuitiva.

A partir desta informação, e sabendo que o Alloy é baseado em lógica de primeira ordem e álgebra relacional, é possível inferir que estas duas linguagens possuem bastantes semelhanças. A verdade é que, para os invariantes mais simples a tradução é quase direta havendo apenas o pequeno pormenor que é o facto de ser necessário especificar a *query* na forma do invariante negado e em formato CNF (*Conjunctional Normal Form*).

Ilustrando então com um exemplo simples, a forma negada do invariante 1 será:

```
some c: Classe_N3 |
    (some c.temFilho) and ((some c.temDF) or (some c.temPCA))
```

A tradução manual deste invariante passa por analisar o tipo de variáveis a identificar onde, neste caso, serão do tipo `Classe_N3` e filtrar os resultados pelas instâncias que verificam as condições impostas no corpo do invariante. O processador SPARQL possui o operador FILTER que permite fazer esta filtragem, resultando na *query* seguinte:

```
PREFIX : <http://jcr.di.uminho.pt/m51-clav#>
PREFIX clav: <http://jcr.di.uminho.pt/m51-clav#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT * WHERE {
    ?c rdf:type :Classe_N3 .
    ?c :temFilho ?cf .

     FILTER (
          EXISTS { ?c :temDF ?df . } ||
          EXISTS { ?c :temPCA ?pca . }
     )
}
```

Queries mais complexas e que envolvam composição de relações, uma vez que o SPARQL não é capaz de tal funcionalidade, a *query* deve explicitamente unificar as variáveis de forma obter o mesmo comportamento. Veja-se o exemplo:

```
some c:Classe_N3,t:c.temTI |
    (some c.temFilho) and (t not in c.temFilho.temTI)
```

```
PREFIX : <http://jcr.di.uminho.pt/m51-clav#>
PREFIX clav: <http://jcr.di.uminho.pt/m51-clav#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT * WHERE {
     ?c rdf:type :Classe_N3 .
     ?c :temFilho ?cf .
     ?c :temTI ?ti .

     FILTER (
          NOT EXISTS { ?cf :temTI ?ti . }
          )
}
```

Esta sistematização dá uma boa confiança de que a automação deste processo está ao alcance.

5.2 Geração automática

Como vimos anteriormente, o processo de tradução da especificação de um invariante em Alloy para uma *query* SPARQL passa por várias fases. A primeira fase é a da negação do invariante em si, utilizando regras de lógica de primeira ordem conhecidas. A segunda fase, passa por identificar as variáveis quantificadas e o seu tipo sendo que, estas serão traduzidas em unificações do tipo: `?var rdf:type :<Tipo>`. A terceira fase passa por identificar, no corpo do invariante quais as condições que as variáveis unificadas devem respeitar de forma a encontrar as instâncias incorretas da base de dados. Esta última fase, tira partido do operador FILTER como mencionado na secção anterior.

Para ser possível sistematizar este processo é necessário carregar o invariante numa estrutura que permita a sua manipulação de forma prática, ou seja, é necessário efetuar o *parsing* do invariante para uma estrutura de dados. Para este

efeito, recorreu-se ao par Happy + Alex, bibliotecas em Haskell que permitem a especificação de uma gramática e de um *lexer* e respetivas ações semânticas a partir de uma DSL (*Domain Specific Language*) bastante semelhante à das ferramentas Yacc/Flex da linguagem C, para obter a árvore sintática, e à biblioteca *recursion-schemes* para a sua manipulação. É de salientar que o gerador desenvolvido tem como objetivo ser apenas uma prova de conceito logo, é apenas capaz de efetuar o *parsing* de um sub conjunto bastante limitado da sintaxe Alloy e de gerar *queries* a partir de especificações bastante simples.

Tendo o *parsing* feito, apenas é necessário transformar a estrutura obtida fazendo-a passar pelas várias fases enunciadas em cima. A codificação das várias fases em Haskell foi bastante simples devido à escolha de manter o nome das relações usadas na ontologia base no modelo Alloy pois, não foi necessário fazer nenhuma correspondência entre os nomes. Sendo assim, segue-se abaixo, o resultado da *query* gerada automaticamente a partir do invariante escolhido na secção anterior:

```
PREFIX : <http://jcr.di.uminho.pt/m51-clav#>
PREFIX clav: <http://jcr.di.uminho.pt/m51-clav#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT * WHERE {


    ?c rdf:type :Classe_N3 .

    FILTER (
        EXISTS {
            ?c :temFilho ?blG .
        }

    && (
        EXISTS {
         ?c :temDF ?76V .
        }

    ||
        EXISTS {
         ?c :temPCA ?FIn .
        }
    ))

}
```

Pode-se afirmar que as *queries* geradas são traduções corretas uma vez que ambas as ferramentas (SPARQL e Alloy) são baseadas nos mesmo princípios teóricos logo, partilham, à partida, de uma representação base.

6 Dificuldades

7 Conclusão e Trabalho Futuro

Referências

1. Alexandra Lourenço, José Carlos Ramalho, M.R.G., Penteado, P.: Plataforma m51-clav: o que há de novo? (2017)

2. Oliveira, J.: Program design by calculation (2008), draft of textbook in preparation, current version: April 2018. Informatics Department, University of Minho.
3. Oliveira, J.N., Ferreira, M.A.: Alloy meets the algebra of programming: A case study. IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, **39**(3), 309–310 (2013)
4. Ramalho, J.C.L.: Análise e modelação (2017)
5. Ramalho, J.C.L.: Requisitos funcionais (2017)