

CLAV - ESPECIFICAÇÃO E VERIFICAÇÃO DO MODELO FORMAL

Armando Santos and Gonalo Duarte

University of Minho, Braga, Portugal

Resumo CLAV   uma plataforma que est  a ser desenvolvida pela Dire o Geral do Livro, Arquivos e Bibliotecas (DGLAB) em parceria t cnica com o Departamento de Inform tica da Universidade do Minho, e tem como objetivo a classifica o e avalia o de todos os documentos que circulam pelas institui es p blicas portuguesas. Neste momento existe um modelo do problema especificado em OWL (*Ontology Web Language*), mas tem sofrido v rias altera es no decorrer do  ltimo ano e n o existiu tempo para estudar o impacto dessas mesmas altera es nas pr -condi es e invariantes do modelo. Neste projeto, inserido na Unidade Curricular de Laborat rios em Engenharia Inform tica (LEI) do MIEI/UM, pretende-se formalizar o modelo de ra z assim como os seus invariantes e garantir a consist ncia dos mesmos, sendo capaz de detetar erros que, at  agora, n o tenham sido identificados.

Keywords: Administra o P blica · M todos Formais · Engenharia Inform tica · Alloy Analyzer · OWL.

1 Introdu o

At  agora, em Portugal, n o existia nenhum sistema de informa o que gerisse a classifica o e a avalia o dos documentos gerados no  mbito dos processos que circulam dentro das institui es p blicas portuguesas. O CLAV veio mudar isso com a elabora o de um cat logo, que se pretende que venha a ser a refer ncia nacional de todos os processos da Administra o P blica (AP), tendo sido modelado numa ontologia. Esta ontologia est  especificada num modelo formal que representa o conjunto de conceitos referentes aos processos de neg cio e aos relacionamentos entre eles. Infelizmente, todos os dados e documenta o de apoio est o espalhados, desorganizados e em diferentes formatos, o que os torna extremamente dif ceis de manter num dom nio t o complexo como o da AP. No entanto, embora j  tenham sido feitos esfor os para criar um formato neutro, como a Macro-estrutura Funcional (MEF), e v rios sistemas de explora o e exporta o, devido aos problemas associados com a inser o manual dos dados oriundos das diversas institui es e   complexidade e instabilidade dos invariantes e restri es associadas ao modelo n o   poss vel garantir a coer ncia das rela es entre os processos de neg cio. Esta incoer ncia   extremamente cr tica uma vez que a classifica o e avalia o dos processos possui legisla es associadas e lida com a remo o ou conserva o (digital e f sica) de documentos governamentais [1].

Deste modo, no contexto da unidade curricular de Laborat3rios em Engenharia Inform1tica do Mestrado Integrado em Engenharia Inform1tica da Universidade do Minho e associado ao perfil de M3todos Formais em Engenharia Inform1tica, apresentamos, neste artigo, a especifica3o e verifica3o, de raiz, da ontologia e o estudo da coer3ncia dos invariantes e pr3-condi3es que fazem parte dela. Devido 3 natureza puramente relacional inerente no dom3nio do problema em quest3o, iremos tirar partido de m3todos alg3bricos e relacionais para nos ajudar a raciocinar sobre o problema em m3os e utilizar o *Alloy model checker* para nos auxiliar a encontrar falhas no desenho do modelo.

2 O Problema

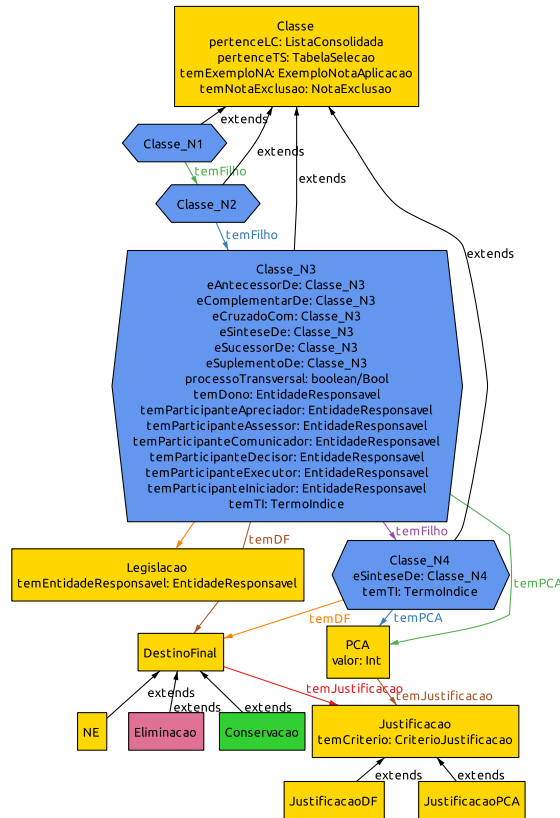


Figura 1: Meta-modelo simplificado

Cada institui3o p3blica portuguesa desempenha uma fun3o espec3fica dentro da AP (p. ex. a presta3o de cuidados de sa3de), associado a cada fun3o 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 [5] 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 níveis 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] [5] [6] 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 [4]. 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 estas *queries*. 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 [4] [3]

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 acima, 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° , onde $b R a$ significa o mesmo que $a R^\circ b$. Por exemplo, *é síntese de* $^\circ$ *é sintetizado por*.

Também é importante observar que relações do género $R = \textit{é o pai de}$, 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

$$R \circ R^\circ \subseteq id \quad (1)$$

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

$$R \subseteq S \equiv \forall b; a : bRa \Rightarrow bSa. \quad (2)$$

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 $B \xleftarrow{R} A$ declara uma relação binária que relaciona B 's com A 's. Por exemplo, $B = \textit{Classe nível 1}$ e $A = \textit{Classe nível 2}$ para o caso em que $R = \textit{é o pai de}$.

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

$$\begin{array}{ccc} A & \xleftarrow{R} & B \xleftarrow{S} C \\ & \nwarrow \quad \nearrow & \\ & R \circ S & \end{array} \quad b(R \circ S)c \equiv \exists a : bRa \wedge aSc \quad (3)$$

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

$$\begin{array}{ccc}
Classe_N2 & \xleftarrow{pertenceLC2} & ListaConsolidada \\
\downarrow temFilho12 & \subseteq & \downarrow id \\
Classe_N1 & \xleftarrow{pertenceLC1} & ListaConsolidada
\end{array}$$

que representa a restrio

$$temFilho12 \circ pertenceLC2 \subseteq id \circ pertenceLC1, \quad (4)$$

onde, no dom4nio do CLAV, as relaes *pertenceLC1* e *pertenceLC2* mapeiam, respetivamente, cada classe de n4vel 1 na sua respetiva LC e cada classe de n4vel 2 na sua respetiva LC, a relao simples, *temFilho12*, relaciona uma classe de n4vel 1 com os seus filhos (classes de n4vel 2), a relao *id* 4 a conhecida relao identidade que relaciona cada objeto consigo pr4prio.

Dos diagramas  l4gica [4] [3]: O que 4 que a expresso (4) significa em l4gica proposicional?

$$\begin{aligned}
& temFilho12 \circ pertenceLC2 \subseteq id \circ pertenceLC1 \\
& \equiv \{ \text{incluso de relaes (2); id} \} \\
& \quad \forall c1, lc : c1 (temFilho12 \circ pertenceLC2) lc \Rightarrow c1 pertenceLC2 lc \\
& \equiv \{ \text{composio (3)} \} \\
& \quad \forall c1, lc : (\exists c2 : c1 temFilho12 c2 \wedge c2 pertenceLC2 lc) \Rightarrow c1 pertenceLC1 lc \\
& \equiv \{ \text{splitting; nesting} \} \\
& \quad \forall c1, c2, lc : c2 pertenceLC2 lc \wedge c1 temFilho12 c2 \Rightarrow c1 pertenceLC1 lc
\end{aligned}$$

Literalmente:

Se uma c2 pertence  lista consolidada lc e c2 4 filho da classe c1, ento c1 tamb4m pertence  lista consolidada lc.

Ainda em menos palavras, a restrio (4), sugere:

Filho de quem pertence, tamb4m pertence.

3.2 Dom4nio

Com a especificao do dom4nio do modelo formal pretendemos descrever a ess4ncia do problema em questo que 4 o de garantir que, dadas as entidades existentes e as suas relaes, o conjunto de invariantes impostos so coerentes, no se contradizem e permitem instncias semanticamente corretas.

Sendo assim, nesta seco e na pr4xima, apesar do dom4nio especificado ser maior, apenas iremos apresentar as especificaes que melhor capturam a ess4ncia do modelo, por questes de s4ntese. No entanto, ser poss4vel consultar o modelo Alloy, na totalidade, em anexo.

Referencial Classificativo: O Referencial Classificativo possui a Lista Consolidada e o conjunto de Tabelas de Seleção onde se encontram todas as classes.

$$ReferencialClassificativo = ListaConsolidada + TabelaSelecao$$

Classes: Como já foi referido existem 4 níveis de classes e, como veremos na secção abaixo especificamo-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:

$$PCA$$

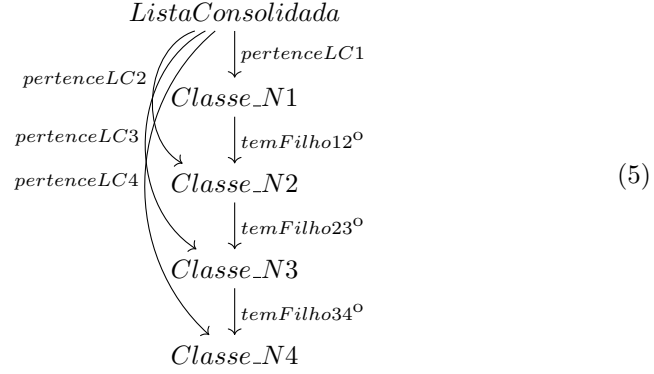
Destino Final: O Destino Final é passível de ser avaliado em um de três valores: *Não Especificado (NE)*, *Eliminação* (quando a documentação associada ao processo de negócio é eliminada, uma vez terminado o PCA) e *Conservação* (quando a documentação associada ao processo de negócio deve ser preservada, uma vez terminado o PCA). Apesar de apenas existirem estes 3 valores, foi necessário distingui-los 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

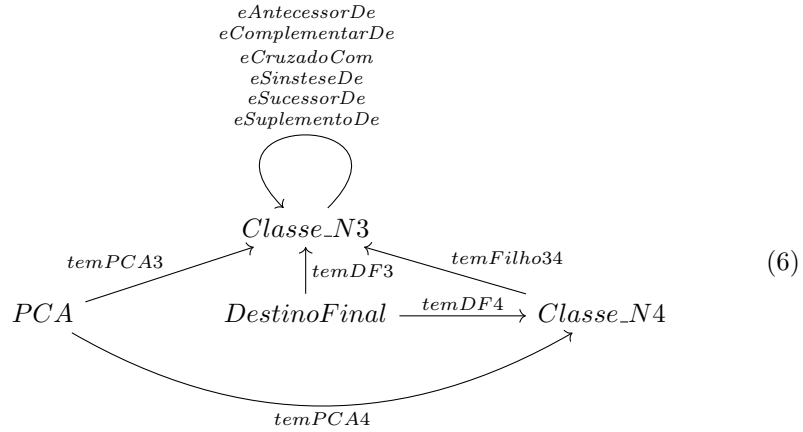
Observando apenas as entidades definidas na secção anterior não é bem clara a existência de uma hierarquia nem é clara a existência das endo-relações que fazem parte do contexto de avaliação nas de nível 3. Isto deve-se ao facto de estas características apenas se evidenciarem ao analisar as relações 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:

Acima, nas relações da família *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:

As endo-relações que dizem respeito aos processos de negócio na figura acima 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:

$$\ker R \subseteq id \quad (7)$$

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):

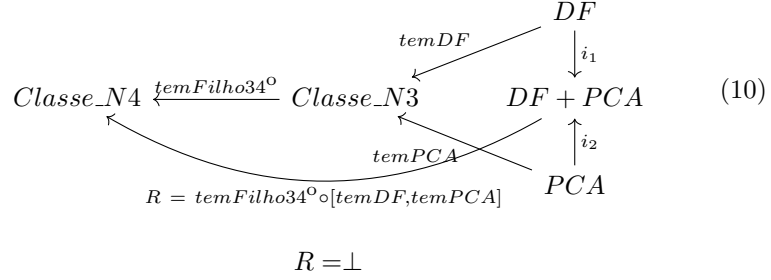
$$\ker \text{temFilho}^\circ \subseteq id \quad (8)$$

$$\ker \text{temDF} \subseteq id \quad (9)$$

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):



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

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

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 síntese 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:

$$eComplementarDe \setminus (\text{temDF} \circ \underline{Conservacao}) \quad (12)$$

$$(\neg eComplementarDe \cap eSinteseDe) \setminus (\text{temDF} \circ \underline{Conservacao}) \quad (13)$$

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

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

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 $Classe_N2 \xleftarrow{temFilho} Classe_N1$ com uma cardinalidade *set* no contradomínio, ou seja, *temFilho* é uma relação de 0 para muitos. Mais concretamente, a relação especificada é a relação *temFilho*₁₂

como podemos concluir olhando para o dom nio e contradom nio. Uma  ltima nota em rela o   forma como foi traduzido o modelo,   a de que apesar de ser mais intuitivo na notac  o relacional representar uma rela o como yRx , onde $Y \xleftarrow{R} X$, em Alloy   mais intuitivo modelar da forma xRy . Mesmo a composi o relacional   feita da esquerda para a direita, ao contr rio da notac  o relacional.

4.2 Verifica o

O Alloy permite especificar express es em l gica de primeira ordem. Para isso, disponibiliza as palavras reservadas *all* e *some* para quantificar universalmente e existencialmente, respetivamente, vari veis tipadas no universo modelado.

Uma das vantagens de utilizar a ferramenta Alloy   a de tudo poder ser visto como uma rela o o que ajuda na tradu o do modelo em si, como vimos na sec o anterior, mas tamb m permite fazer uma tradu o praticamente direta dos invariantes em notac  o relacional *point wise*, com o aux lio dos quantificadores, ou *point free*, com o aux lio da composi o de rela es.

Com isto, e pela mesma ordem, segue-se a tradu o 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 Tradu o de *queries* SPARQL

Um modelo formal em Alloy permite raciocinar, utilizando c lculo relacional e m todos matem ticos, 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 avançado 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 traduo manual deste invariante passa por analisar o tipo de variveis a identificar onde, neste caso, so do tipo `Classe_N3` e filtrar os resultados pelas instncias que verificam as condies 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 composio de relaes, uma vez que o SPARQL no  capaz de tal funcionalidade, a *query* deve explicitamente unificar as variveis 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 sistematizao d uma boa confiana de que a automao deste processo est ao alcance e, demonstra a importncia e utilidade do ciclo de vida de verificao na produtividade e qualidade do trabalho produzido onde, a formalizao dos invariantes em Alloy permite perceber o impacto destes no modelo e extrair de forma correta *queries* SPARQL que identificam instncias incorretas na aplicao em produo.

5.2 Gerao automtica

Como vimos anteriormente, o processo de traduo da especificao de um invariante em Alloy para uma *query* SPARQL passa por vrias fases. A primeira fase  a da negao do invariante em si, utilizando regras de lgica de primeira ordem conhecidas. A segunda fase, passa por identificar as variveis quantificadas e o seu tipo sendo que, estas so traduzidas em unificaes 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 ?bIG .
        }

        && (
            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

Durante o desenvolvimento do projeto foram surgindo algumas dificuldades que impactaram na produtividade e no progresso do trabalho. Apesar dos aspetos negativos associados às dificuldades encontradas, nem sempre a sua existência foi vista como algo mau.

O facto da documentação do projeto do CLAV, na fase inicial, ter estado espalhada, desorganizada e desatualizada afetou negativamente o progresso do trabalho uma vez que, foi necessário agrupar toda a informação relevante e organizá-la, de forma a facilitar as fases de desenvolvimento futuras. Alguns dos invariantes identificados eram ambíguos, característica da linguagem natural onde foram especificados, e apesar de este facto dificultar a interpretação dos mesmos, a aplicação de métodos rigorosos contribuiu para o refinamento destes.

Outra dificuldade que também contribuiu significativamente para o resultado final deste trabalho foram as restrições de tempo. Uma vez que este trabalho foi realizado em paralelo com mais quatro unidades curriculares, houve um período de estagnação (época de exames) onde os autores não tiveram tanta capacidade de trabalho.

7 Conclusão e Trabalho Futuro

Observando o trabalho no seu estado final², é possível afirmar que este foi concluído com sucesso. O trabalho conseguiu alcançar todos os objetivos propostos: a formalização do modelo e a verificação deste. A utilização de métodos formais e, em particular, o *model checker* Alloy, foram bastante eficazes em detetar erros que não tinham sido identificados.

Para além dos objetivos propostos, foi ainda apresentada uma prova de conceito que visa aumentar e melhorar a produtividade de trabalhos futuros de um tradutor de invariantes Alloy em *queries* SPARQL. Este trabalho adicional foi inspirado no facto de o CLAV ser um projeto que já se encontra em produção e não teve a capacidade de adotar os métodos apresentados ao longo deste relatório mais cedo.

O modelo formal apresentado em anexo consiste apenas na formalização da Lista Consolidada e das relações presentes na classificação e avaliação dos processos de negócio. Na realidade, o CLAV possui outros componentes complexos que irão interagir entre si e necessitam de alguma garantia de correção. Sendo assim, um trabalho futuro poderá consistir em formalizar o resto do modelo que diz respeito ao CLAV e assistir a sua implementação de forma correta antes que seja tarde demais!

Outro plano futuro, poderá ser o de melhorar a ferramenta de tradução automática de invariantes, de forma a suportar funcionalidades mais complexas. É de salientar que já existe algum trabalho, bastante incompleto, relacionado com a reescrita de expressões OWL/DL [2] em SPARQL, levando em conta a semântica

² <https://github.com/bolt12/clav>

OWL e SPARQL. Este trabalho relacionado, apresenta algumas regras de reescrita formalizadas que poderão dar jeito implementar na prova de conceito (com alguns ajustes) uma vez que, a lógica descritiva possui uma semântica bastante semelhante à do Alloy.

Concluindo, apesar deste trabalho ter contribuído significativamente para melhorar a correção do modelo da Lista Consolidada, eliminar bastantes falhas e adicionar invariantes que não tinham sido considerados, proporcionou um ambiente de desenvolvimento melhor ao anterior, baseado numa evolução incremental e não estudada do modelo, fornecendo a base para que o ciclo de verificação possa continuar e uma ferramenta (a de tradução) para que, num ambiente de produção, seja possível identificar de forma automática os erros de instância presentes

Referências

1. Alexandra Lourenço, José Carlos Ramalho, M.R.G., Penteado, P.: Plataforma m51-clav: o que há de novo? (2017)
2. Lorenz Bühmann, J.L.: Owl class expression to sparql rewriting Universität Leipzig, Institut für Informatik, AKSW, Postfach 100920, D-04009 Leipzig, Germany
3. Oliveira, J.: Program design by calculation (2008), draft of textbook in preparation, current version: April 2018. Informatics Department, University of Minho.
4. 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)
5. Ramalho, J.C.L.: Análise e modelação (2017)
6. Ramalho, J.C.L.: Requisitos funcionais (2017)

Apêndice A Invariantes

A.1 Índice

- Índice
- Invariantes
 - Índice
 - Classes de 1º Nível
 - Classes de 2º Nível
 - Classes de 3º Nível com desdobramento.
 - Invariantes sobre os PNs (Classe 3)
 - Invariantes sobre a relação Suplementar: implicações no PCA
 - Invariantes sobre a relação Síntese: implicações no DF
 - Invariantes sobre a relação Complementar: implicações no DF
 - Invariantes sobre o DF
 - Invariantes sobre o PCA
 - Invariantes sobre os Termos de Índice
 - Invariantes das relações `temRelProc`
 - Invariantes sobre as Justificações

A.2 Invariantes

A numeração dos invariantes corresponde à numeração do Alloy.

Índice

- inv1
- inv2
- inv3
- inv4
- inv5
- inv6
- inv7
- inv8
- inv9
- inv10
- inv11
- inv12
- inv13
- inv14
- inv15
- inv16
- inv17
- inv18
- inv19
- inv20

- inv21
- inv22
- inv23
- inv24
 - inv24-1
 - inv24-2
 - inv24-3
- inv25
- inv26
- inv27
- inv28
- inv29
- inv30
- inv31
- inv32
- inv33
- inv34
- inv35
- inv36
- inv37

Classes de 1º Nível

- (**inv1**) Se uma Classe_N1 pertence a uma LC/TS, consequentemente os seus filhos, netos, etc.. também têm de pertencer.

```

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 {
    ?c1 rdf:type :Classe_N1 .
    ?c1 :pertenceLC ?lc1 .

    ?c2 rdf:type :Classe_N2 .
    ?c2 :temPai ?c1 .
    ?c2 :pertenceLC ?lc2 .

    ?c3 rdf:type :Classe_N3 .
    ?c3 :temPai ?c2 .
    ?c3 :pertenceLC ?lc3 .

    ?c4 rdf:type :Classe_N4 .
    ?c4 :temPai ?c3 .
    ?c4 :pertenceLC ?lc4 .

    FILTER (
        ?c1 != ?c2
        && ?c2 != ?c3
        && ?c3 != ?c4
        && (?lc1 != ?lc2
            || ?lc2 != ?lc3
            || ?lc3 != ?lc4)
    )
}

```

- (inv2) Se uma Classe_N1 no pertence a uma LC/TS, consequentemente os seus filhos, netos, etc.. tambem no pertencem.

```

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 {
  ?c1 rdf:type :Classe_N1 .

  minus {
    ?c1 :pertenceLC ?lc1 .
  }

  ?c2 rdf:type :Classe_N2 .
  ?c2 :temPai ?c1 .

  ?c3 rdf:type :Classe_N3 .
  ?c3 :temPai ?c2 .

  ?c4 rdf:type :Classe_N4 .
  ?c4 :temPai ?c3 .

  FILTER (
    ?c1 != ?c2
    && ?c2 != ?c3
    && ?c3 != ?c4 &&
    ( EXISTS { ?c2 :pertenceLC ?lc2 . }
    || EXISTS { ?c3 :pertenceLC ?lc3 . }
    || EXISTS { ?c4 :pertenceLC ?lc4 . } )
  )
}

```

- (inv24-1) 2 Classe_N1 nao podem ter a mesma instancia NotaAplicacao

```

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 {
  ?c1 :temNotaAplicacao ?o.
  ?c2 :temNotaAplicacao ?o.
  FILTER (
    ?c1 != ?c2
  )
}

```

- (23) 2 Classe_N1 nao podem ter a mesma instancia NotaExclusao

Igual a cima

- (23) 2 Classe_N1 nao podem ter a mesma instancia ExemploNotaAplicacao

Igual a cima

Classes de 2º Nível

- (inv24-2) 2 Classe_N3 nao podem ter a mesma instancia NotaAplicacao

Igual a cima

- (24) 2 Classe_N3 nao podem ter a mesma instancia NotaExclusao

Igual a cima

- (24) Classe_N3 nao podem ter a mesma instancia ExemploNotaAplicacao

Igual a cima

Classes de 3º Nível com desdobramento.

- Classes_N3 com desdobramento:
 - (inv5) Os 4ºs niveis herdam as legislações existentes no 3 nivel (subconjunto), quer para o PCA quer para o DF
 - (inv6) Só existe desdobramento caso o PCA ou DF sejam distintos

<pre> 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 ?cf1 . ?c :temFilho ?cf2 . ?cf1 :temDF ?df1 . ?cf2 :temDF ?df2 . ?df1 :dfValor ?dfv1 . ?df2 :dfValor ?dfv2 . ?cf1 :temPCA ?pca1 . ?cf2 :temPCA ?pca2 . ?pca1 :pcaValor ?pcav1 . ?pca2 :pcaValor ?pcav2 . FILTER (?cf1 != ?cf2 && ?dfv1 = ?dfv2 && ?pcav1 = ?pcav2) }</pre>
--

- PCA distinto
 - * deve ser possível diferenciar as relações de cada uma das subdivisões ao 4º nível (por exemplo o critério de utilidade administrativa em cada um dos 4ºs níveis reflete relações suplementares com 3ºs níveis distintos; ou apenas uma subdivisão de 4º nível tem critério legal)
 - Vão ter um criterio de utilidade administrativa que referenciam classes nivel 3 distintas entre eles. (PENDENTE)**

* (inv7) O DF   avaliado tendo em conta o contexto de 3  n vel, em que:

- Se for distinto tem que haver uma rela o de s ntese entre as classes 4 filhas.
- Se for igual n o interessa

```

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 ?cf1 .
  ?c :temFilho ?cf2 .

  ?cf1 :temPCA ?pca1 .
  ?cf2 :temPCA ?pca2 .
  ?pca1 :pcaValor ?pcav1 .
  ?pca2 :pcaValor ?pcav2 .

  ?cf1 :temDF ?df1 .
  ?cf2 :temDF ?df2 .
  ?df1 :dfValor ?dfv1 .
  ?df2 :dfValor ?dfv2 .

  FILTER (
    ?cf1 != ?cf2
    && ?pcav1 != ?pcav2
    && ?dfv1 != ?dfv2
    && NOT EXISTS { ?cf1 :eSinteseDe ?cf2 . }
  )
}

```

• (inv8) DF distinto

- * Deve haver uma rela o de s ntese (de ou por) entre as classes 4 filhasa
- * O PCA   igual

```

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 ?cf1 .
  ?c :temFilho ?cf2 .

  ?cf1 :temPCA ?pca1 .
  ?cf2 :temPCA ?pca2 .
  ?pca1 :pcaValor ?pcav1 .
  ?pca2 :pcaValor ?pcav2 .

  ?cf1 :temDF ?df1 .
  ?cf2 :temDF ?df2 .
  ?df1 :dfValor ?dfv1 .
  ?df2 :dfValor ?dfv2 .

  FILTER (
    ?cf1 != ?cf2
    && ?dfv1 != ?dfv2
    && (
      NOT EXISTS { ?cf1 :eSinteseDe ?cf2 . }
      || ?pcav1 != ?pcav2
    )
  )
}

```

- (inv9) Os termos de índice vão para os 4^{os} níveis (repetidos)

```

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 . }
  )
}

```

- (inv3) As relações **temDF** e **temPCA**, não existem numa classe 3 se esta tiver filhos.

```

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 . }
  )
}

```

- (inv4) As relações **temDF** e **temPCA**, existem numa classe 3 se esta **não** tiver filhos.

```

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 (
    NOT EXISTS { ?c :temFilho ?cf . } &&
    (NOT EXISTS { ?c :temDF ?df . } ||
     NOT EXISTS { ?c :temPCA ?pca . })
  )
}

```

- Um PN tem sempre um PCA e um DF. O valor do PCA poderá ser nulo se o PCA tiver uma nota associada.
- (inv10) Se um PN (Classe 3) for **complementar** de outro que se desdobra ao 4^o nível, é necessário, com base no critério de complementaridade informacional, a relação manter-se ao 3^o nível. Pelo menos um dos 4^{os} níveis deve ser de **conservação**.

```

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 {
  ?c1 rdf:type :Classe_N3 .
  ?c2 rdf:type :Classe_N3 .

  ?c1 :eComplementarDe ?c2 .
  ?c2 :temFilho ?c2f .

  ?c2f :temDF ?df .
  ?df :dfValor ?dfv .

  FILTER (
    ?c1 != ?c2
    && ?dfv != "C"
  )
}

```

- (inv11) Um processo só tem participantes se for **transversal** (Se o campo ‘transversal’ tiver o valor sim).

```

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 :processoTransversal "N" .

  FILTER (
    EXISTS { ?c :temParticipanteComunicador ?ca . }
    ||
    EXISTS { ?c :temParticipanteIniciador ?ca . }
    ||
    EXISTS { ?c :temParticipanteApreciador ?ca . }
    ||
    EXISTS { ?c :temParticipanteDecisor ?ca . }
    ||
    EXISTS { ?c :temParticipanteAssessor ?ca . }
    ||
    EXISTS { ?c :temParticipanteExecutor ?ca . }
  )
}

```

- Uma classe N3 tem de ter sempre pelo menos um dono.
- (inv24-3) 2 Classe_N3 nao podem ter a mesma instancia NotaAplicacao
- (25) 2 Classe_N3 nao podem ter a mesma instancia NotaExclusao
- (25) 2 Classe_N3 nao podem ter a mesma instancia ExemploNotaAplicacao
- (inv25) 2 Classe_N3 nao podem ter os mesmos filhos
- (inv26) 2 Classe_N3 nao podem ter o mesmo TI

Invariantes sobre os PNs (Classe 3)

- (inv12) As relações eComplementarDe e eCruzadoCom são simétricas.

```

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 {
  ?c1 :eComplementarDe ?c2 .
  ?c3 :eCruzadoCom ?c4 .
}

```



```

FILTER (
    ?c1 != ?c2
    && ?c3 != ?c4
    && NOT EXISTS { ?c2 :eComplementarDe ?c2 . }
    && NOT EXISTS { ?c4 :eCruzadoCom ?c3 . }
)
}

```

- (inv14) As relações eSinteseDe, eSucessorDe e eSuplementoDe são anti-simétricas. (eSuplementoDe ou eSuplementoPara ?? As duas)

```

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 {
    ?c1 :eSinteseDe ?c2 .
    ?c2 :eSinteseDe ?c1 .

    ?c3 :eSintetizadoPor ?c4 .
    ?c4 :eSintetizadoPor ?c3 .

    ?c5 :eSucessorDe ?c6 .
    ?c6 :eSucessorDe ?c5 .

    ?c7 :eSuplementoDe ?c8 .
    ?c8 :eSuplementoDe ?c7 .

    ?c9 :eSuplementoPara ?c10 .
    ?c10 :eSuplementoDe ?c9 .

    FILTER (
        ?c1 != ?c2
        && ?c3 != ?c4
        && ?c5 != ?c6
        && ?c7 != ?c8
        && ?c9 != ?c10
    )
}

```

- (inv13) Na relação temRelProc um PN não se relaciona com ele próprio.

```

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 {
    ?c1 rdf:type :Classe_N3 .
    ?c2 rdf:type :Classe_N4 .

    FILTER (
        EXISTS { ?c1 :eAntecessorDe ?c1 . }
        ||
        EXISTS { ?c1 :eComplementarDe ?c1 . }
        ||
        EXISTS { ?c1 :eCruzadoCom ?c1 . }
        ||
        EXISTS { ?c1 :eSinteseDe ?c1 . }
        ||
        EXISTS { ?c1 :eSintetizadoPor ?c1 . }
        ||
        EXISTS { ?c2 :eSinteseDe ?c2 . }
    )
}

```

```

    ||
    EXISTS { ?c2 :eSintetizadoPor ?c2 . }
    ||
    EXISTS { ?c1 :eSucessorDe ?c1 . }
    ||
    EXISTS { ?c1 :eSuplementoDe ?c1 . }
    ||
    EXISTS { ?c1 :eSuplementoPara ?c1 . }
  )
}

```

- (inv15) Um PN s3 pode ter uma rela3o com outro PN (**Uma rela3o como assim? R: Um PN n3o pode ter rela33es distintas com o mesmo processo | Que rela33es? As do tipo Classe_N4 -> Classe_N3**)

```

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 {
  ?c1 rdf:type :Classe_N3 .
  ?c2 rdf:type :Classe_N3 .
  ?c1 :eAntecessorDe ?c2 .

  FILTER (
    ?c1 != ?c2
    &&
    (
      EXISTS { ?c1 :eComplementarDe ?c2 . }
      ||
      EXISTS { ?c1 :eCruzadoCom ?c2 . }
      ||
      EXISTS { ?c1 :eSinteseDe ?c2 . }
      ||
      EXISTS { ?c1 :eSintetizadoPor ?c2 . }
      ||
      EXISTS { ?c1 :eSucessorDe ?c2 . }
      ||
      EXISTS { ?c1 :eSuplementoDe ?c2 . }
      ||
      EXISTS { ?c1 :eSuplementoPara ?c2 . }
    )
  )
}

```

- (inv16) Se um PN se desdobra em 43s n3veis os Termos de 3ndice passam para os filhos.

```

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 :temTI ?ti .}
  )
}

```

- (inv17) Os termos de índice de um PN não existem em mais nenhuma classe
3

```

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 {
    ?c1 rdf:type :Classe_N3 .
    ?c2 rdf:type :Classe_N3 .

    ?c1 :temTI ?ti .

    FILTER (
        ?c1 != ?c2
        && EXISTS { ?c2 :temTI ?ti . }
    )
}

```

- (inv20) Dois PNs não podem ter a mesma instancia de DF.
- (inv21) Dois PNs não podem ter a mesma instancia de PCA.

Invariantes sobre a relação Suplementar: implicações no PCA

- (inv29) Quando o PN (sem filhos) em causa eSuplementoPara outro, deve ser acrescentado um critério de utilidade administrativa na justificação do respetivo PCA; (eSuplementoDe ou eSuplementoPara ? SuplementoPara | Que PCA, o do PN em causa ou o outro? Em causa | Não existe nenhuma relação do tipo Justificacao -> CriterioJustificacao R: temCriterio)
 - Nesse critério, critério de utilidade administrativa, devem aparecer todos os processos com os quais existe uma relação eSuplementoPara; (Que todos os processos?? Todas as que o PN em CAUSA está relacionado por a relação eSuplementoPara)
- (inv33) Quando o PN (com filhos) em causa eSuplementoPara outro, deve ser acrescentado um critério de utilidade administrativa na justificação PCA dos filhos em que os processos relacionados com o critério podem ser distintos.

Invariantes sobre a relação Síntese: implicações no DF

- Quando o PN em causa é síntese de outro, o DF deve ter o valor de “Conservação”;
- Quando o PN em causa é sintetizado por outro, o DF deve ter o valor de “Eliminação”;
- (inv34) Todos os processos relacionados (sem filhos) por uma relação de síntese deverão estar relacionados com o critério de densidade informacional da respetiva justificação. (critTemProcRel O critério é que se relaciona com os processos)
- (inv35) Todos os processos relacionados (com filhos) por uma relação de síntese, os filhos deverão estar relacionados com o critério de densidade informacional da respetiva justificação. (critTemProcRel O critério é que se relaciona com os processos)

Invariantes sobre a relao Complementar: implicaes no DF

- Uma relao de complementaridade implica a conservao dos PNs que mantm essa relao; (**implica que o DF seja de conservao? Nos dois processos? Sim e sim**)
- (**inv36**) Todos os processos relacionados (sem filhos) pela relao `eComplementarDe`, devem estar relacionados com o critrio de complementaridade informacional da respetiva justificao.
- (**inv37**) Todos os processos relacionados (com filhos) pela relao `eComplementarDe`, os filhos devem estar relacionados com o critrio de complementaridade informacional da respetiva justificao.

Invariantes sobre o DF

- (**inv30**) Um DF, na sua justificao, dever conter apenas critrios de densidade informacional, complementaridade informacional e legal
- (**inv18**) 2 DFs no podem ter a mesma instncia de Justificacao;
- (**inv27**) Um DF, na sua justificao, apenas deve ter uma instncia de JustificacaoDF;

Invariantes sobre o PCA

- (**inv31**) Um PCA, na sua justificao, dever conter apenas critrios gestionrios, utilidade administrativa e legal
- (**inv19**) 2 PCAs no podem ter a mesma instncia de Justificacao;
- (**inv28**) Um PCA, na sua justificao, apenas deve ter uma instncia de JustificacaoPCA;

Invariantes sobre os Termos de ndice

- Os termos de ndice so nicos no universo do sistema

Invariantes das relaes temRelProc S se aplica ao 3^o nvel.

- (**inv32**) Se um PN  (por ordem de prioridade):
 - `eComplementarDe` -> DF  de conservao
 - `eSinteseDe` -> DF  de conservao
 - `eSintetizadoPor` -> DF  de eliminao
 - nenhuma das acima -> DF  NE (No especificado)

```

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 :eComplementarDe ?o .
  ?c :temDF ?df .
  ?df :dfValor ?dfv .

```

```

        minus {
            ?c :temFilho ?f .
        }

    FILTER (?dfv != "C")
}

```

```

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 :eSinteseDe ?ca .
    ?c :temDF ?df.
    ?df :dfValor ?dfv.

    minus {
        ?c :temFilho ?cf .
        ?c :eComplementarDe ?cb .
    }

    FILTER (?dfv != "C")
}

```

```

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 :eSintetizadoPor ?ca .
    ?c :temDF ?df.
    ?df :dfValor ?dfv.

    minus {
        ?c :temFilho ?f .
        ?c :eComplementarDe ?cb .
        ?c :eSinteseDe ?cc .
    }

    FILTER (?dfv != "E")
}

```

```

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 :temDF ?df.
    ?df :dfValor ?dfv.
    minus {
        ?c :temFilho ?cf .
        ?c :eComplementarDe ?ca .
        ?c :eSintetizadoPor ?cb .
        ?c :eSinteseDe ?cc .
    }

    FILTER (?dfv != "NE")
}

```

Invariantes sobre as Justificações

- (inv23) 2 Justificaes nao podem ter a mesma instancia de CriterioJustificacao
- (inv22) Cada Justificacao tem no maximo 3 CriterioJustificacao diferentes
- (inv38) Cada Justificacao tem no maximo 1 Criterio de cada tipo

Ap ndice B Modelo formal Alloy

```

open util/boolean
open RelCalc

/* Atributo Composto */
abstract sig CriterioJustificacao {
    critTemLegAssoc: set Legislacao,
    critTemProcRel: set Classe_N3
}

sig CriterioJustificacaoComplementaridadeInfo extends CriterioJustificacao
    ↪ {}
sig CriterioJustificacaoDensidadeInfo extends CriterioJustificacao {}
sig CriterioJustificacaoGestionario extends CriterioJustificacao {}
sig CriterioJustificacaoLegal extends CriterioJustificacao {}
sig CriterioJustificacaoUtilidadeAdministrativa extends
    ↪ CriterioJustificacao {}

abstract sig DestinoFinal {
    temJustificacao: one Justificacao
}
sig Eliminacao, Conservacao, NE extends DestinoFinal {} /* dfValor :: DF ->
    ↪ String */

sig ExemploNotaAplicacao {}

/* Justificacao */
abstract sig Justificacao {
    temCriterio: some CriterioJustificacao
}
sig JustificacaoDF extends Justificacao {}
sig JustificacaoPCA extends Justificacao {}
/* ----- */

sig NotaAplicacao {
    naPertenceClasse: one Classe
}

sig NotaExclusao {
    nePertenceClasse: one Classe,
    usarClasse: set Classe
}

sig PCA {
    temJustificacao: one Justificacao,
    valor: one Int
}
/* ----- */

/* Classe */
abstract sig Classe {
    pertenceLC: lone ListaConsolidada,
    temNotaAplicacao: set NotaAplicacao,
    temNotaExclusao: set NotaExclusao,
    pertenceTS: lone TabelaSelecao,
    temExemploNA: set ExemploNotaAplicacao
}

```

```

sig Classe_N1 extends Classe {
    temFilho: set Classe_N2,
}
sig Classe_N2 extends Classe {
    temPai: one Classe_N1,
    temFilho: set Classe_N3,
}

sig Classe_N3 extends Classe {
    temPai: one Classe_N2,
    temDono: some EntidadeResponsavel,
    temFilho: set Classe_N4,
    temTI: set TermoIndice,
    /* temRelProc */
    eAntecessorDe: lone Classe_N3,
    eComplementarDe: set Classe_N3,
    eCruzadoCom: set Classe_N3,
    eSinteseDe: set Classe_N3,
    eSintetizadoPor: set Classe_N3,
    eSucessorDe: lone Classe_N3,
    eSuplementoDe: set Classe_N3,
    eSuplementoPara: set Classe_N3,
    /* ----- */
    /* temParticipante */
    temParticipanteComunicador: set EntidadeResponsavel,
    temParticipanteIniciador: set EntidadeResponsavel,
    temParticipanteApreciador: set EntidadeResponsavel,
    temParticipanteDecisor: set EntidadeResponsavel,
    temParticipanteAssessor: set EntidadeResponsavel,
    temParticipanteExecutor: set EntidadeResponsavel,
    /* ----- */
    temLegislacao: set Legislacao,
    temDF: lone DestinoFinal,
    temPCA: lone PCA,

    /* Relacoes de atributo */
    processoTransversal: one Bool
    /* ----- */
}

sig Classe_N4 extends Classe {
    temPai: one Classe_N3,
    temTI: set TermoIndice, // Partilha entre irmaos -- VER CAMPOS
    /* temRelProc */
    eSinteseDe: set Classe_N4,
    eSintetizadoPor: set Classe_N4,
    /* ----- */
    temDF: one DestinoFinal,
    temPCA: one PCA
}

/* Factos Classe_N4 */
fact classe_n4 {
    no Classe_N4.temNotaAplicacao
    no Classe_N4.temNotaExclusao
    no Classe_N4.temExemploNA
}
/* ----- */

abstract sig EntidadeResponsavel {
    eDonoProcesso: set Classe_N3,
    /* participaEm */
    participaEmComunicando: set Classe_N3,
    participaEmIniciando: set Classe_N3,
    participaEmApreciando: set Classe_N3,
    participaEmDecidindo: set Classe_N3,
    participaEmAssessorando: set Classe_N3,
    participaEmExecutando: set Classe_N3,

```

```

/* ----- */
}

sig Entidade extends EntidadeResponsavel {
    pertenceTipologiaEnt: set TipologiaEntidade
}

sig TipologiaEntidade extends EntidadeResponsavel {
    contemEntidade: some Entidade
}

sig Legislacao {
    temEntidadeResponsavel: set EntidadeResponsavel
}

/* ReferencialClassificativo */
abstract sig ReferencialClassificativo {
    temClasse: set Classe
}

sig ListaConsolidada extends ReferencialClassificativo {}
sig TabelaSelecao extends ReferencialClassificativo {}
/* ----- */

sig TermoIndice {
    estaAssocClasse: one Classe_N3 + Classe_N4
}

/* FACTOS */
fact {
    /* Relacoes inversas */
    -- pertenceLC
    pertenceLC in ~temClasse

    -- temFilho
    ~(Classe_N1<: temFilho) = (Classe_N2<: temPai)
    ~(Classe_N2<: temFilho) = (Classe_N3<: temPai)
    ~(Classe_N3<: temFilho) = (Classe_N4<: temPai)

    -- temNotaAplicacao
    ~temNotaAplicacao in naPertenceClasse

    -- temNotaExclusao
    ~temNotaExclusao in nePertenceClasse

    -- temDono
    ~eDonoProcesso in temDono

    -- temTI
    ~(Classe_N3<: temTI) in estaAssocClasse
    ~(Classe_N4<: temTI) in estaAssocClasse

    -- temRelProc
    -- eAntecessorDe/eSucessorDe
    ~eAntecessorDe = eSucessorDe
    -- eSinteseDe/eSintetizadoPor
    ~(Classe_N3<: eSinteseDe) = Classe_N3<: eSintetizadoPor
    ~(Classe_N4<: eSinteseDe) = Classe_N4<: eSintetizadoPor
    -- eSuplementoDe/eSuplementoPara
    ~eSuplementoDe = eSuplementoPara

    -- temParticipante
    -- temParticipanteComunicador
    ~participaEmComunicando in temParticipanteComunicador
    -- temParticipanteIniciador
    ~participaEmIniciando in temParticipanteIniciador
    -- temParticipanteApreciador
    ~participaEmApreciando in temParticipanteApreciador
    -- temParticipanteDecisor

```



```

~participaEmDecidindo in temParticipanteDecisor
-- temParticipanteAssessor
~participaEmAssessorando in temParticipanteAssessor
-- temParticipanteExecutor
~participaEmExecutando in temParticipanteExecutor

-- temLegislacao VER NO FUTURO

-- contemEntidade
~contemEntidade = pertenceTipologiaEnt
/* ----- */
}

/* INVARIANTES */

/* Se uma Classe_N1 pertence a uma LC/TS, consequentemente os seus filhos,
   ↪ netos, etc..
   tambem tem de pertencer
*/
pred inv1 {
  all c: Classe_N1, lc: ListaConsolidada | lc = c.pertenceLC ⇒
    (all cf: c.temFilho | lc = cf.pertenceLC) and
    (all cf: c.temFilho.temFilho | lc = cf.pertenceLC) and
    (all cf: c.temFilho.temFilho.temFilho | lc = cf.pertenceLC)

  all c: Classe_N1, ts: TabelaSelecao | ts = c.pertenceTS ⇒
    (all cf: c.temFilho | ts = cf.pertenceTS) and
    (all cf: c.temFilho.temFilho | ts = cf.pertenceTS) and
    (all cf: c.temFilho.temFilho.temFilho | ts = cf.pertenceTS)
}

/* Se uma Classe_N1 nao pertence a uma LC/TS, consequentemente os seus filhos
   ↪ ,netos, etc.. tambem nao pertencem */
pred inv2 {
  all c: Classe_N1 | no c.pertenceLC ⇒
    (all cf: c.temFilho | no cf.pertenceLC) and
    (all cf: c.temFilho.temFilho | no cf.pertenceLC) and
    (all cf: c.temFilho.temFilho.temFilho | no cf.pertenceLC)

  all c: Classe_N1 | no c.pertenceTS ⇒
    (all cf: c.temFilho | no cf.pertenceTS) and
    (all cf: c.temFilho.temFilho | no cf.pertenceTS) and
    (all cf: c.temFilho.temFilho.temFilho | no cf.pertenceTS)
}

/* As relacoes temDF e temPCA, nao existem numa classe 3 se esta tiver filhos
   ↪ . */
pred inv3 {
  all c: Classe_N3 | some c.temFilho ⇒ no c.temDF and no c.temPCA
}

/* As relacoes temDF e temPCA, existem numa classe 3 se esta nao tiver filhos
   ↪ . */
pred inv4 {
  all c: Classe_N3 | no c.temFilho ⇒ one c.temDF and one c.temPCA
}

/* Nas classes 3 com desdobramento */

/* Os 4os niveis herdam as legislacoes existentes no 3o nivel (subconjunto),
   ↪ quer para o PCA quer para o DF. */
pred inv5 {
  all c: Classe_N3 | some c.temFilho ⇒ {
    (c.temFilho.temPCA.temJustificacao.temCriterio.critTemLegAssoc
     + c.temFilho.temDF.temJustificacao.temCriterio.critTemLegAssoc) in c
    ↪ .temLegislacao
  }
}

```

```

/* Apenas se desdobram devido a um PCA distinto ou DF distinto */
pred inv6 {
    all c: Classe_N3 | #c.temFilho > 1  $\Rightarrow$  (#c.temFilho.temDF > 1) or
        (#c.temFilho.temPCA > 1 and (all p1,
             $\hookrightarrow$  p2: c.temFilho.temPCA | p1.
             $\hookrightarrow$  valor  $\neq$  p2.valor))
}

/* Caso o motivo de desdobramento seja PCA distinto:
   - Caso o DF seja distinto tem que haver uma relacao de sintese entre as
    $\hookrightarrow$  classes 4 filhas
*/
pred inv7 {
    all c: Classe_N3 | #c.temFilho > 1
        and (#c.temFilho.temPCA > 1)
        and (#c.temFilho.temDF > 1)
         $\Rightarrow$  (all disj c1,c2: c.temFilho |
            c1 $\rightarrow$ c2 in eSinteseDe  $\wedge$ 
            (c1.temDF in Conservacao) and (c2.temDF in Eliminacao))
}

/* Caso o motivo de desdobramento seja DF distinto:
   - Tem que haver uma relacao de sintese entre as classes 4 filhas
   - O PCA e igual
*/
pred inv8 {
    all c: Classe_N3 | #c.temFilho > 1 and
        (c.temFilho.temDF not in Eliminacao
             $\hookrightarrow$  ) and
        (c.temFilho.temDF not in
             $\hookrightarrow$  Conservacao)  $\Rightarrow$ 
        one c.temFilho.temPCA.valor and
        (all disj c1,c2: c.temFilho {
            (c1.temDF in Conservacao) and
             $\hookrightarrow$  (c2.temDF in
             $\hookrightarrow$  Eliminacao)  $\Rightarrow$ 
            c1 $\rightarrow$ c2 in eSinteseDe
        })
}

/* Os termos de indice vao para os 4os niveis */
pred inv9 {
    all c: Classe_N3,t: c.temTI | some c.temFilho  $\Rightarrow$  t in c.temFilho.temTI
}

/* Se um PN (Classe 3) for complementar de outro que se desdobra ao 4o nivel,
    $\hookrightarrow$  e necessario,
   com base no criterio de complementaridade informacional, a relacao manter-
    $\hookrightarrow$  se ao 3o nivel.
   Pelo menos um dos 4os niveis deve ser de conservacao. */
pred inv10 {
    all disj c1,c2: Classe_N3 | c1 in c2.eComplementarDe and some c2.
         $\hookrightarrow$  temFilho  $\Rightarrow$  some c3: c2.temFilho | c3.temDF in Conservacao
}

/* Um processo so tem participantes se for transversal */
pred inv11 {
    all c: Classe_N3 | False = c.processoTransversal  $\Rightarrow$  no
        (c.temParticipanteComunicador
        + c.temParticipanteIniciador
        + c.temParticipanteApreciador
        + c.temParticipanteDecisor
        + c.temParticipanteAssessor
        + c.temParticipanteExecutor)
}

```

```

/* As relacoes eComplementarDe e eCruzadoCom sao simetricas. */
pred inv12 {
    Symmetric[eComplementarDe]
    Symmetric[eCruzadoCom]
}

/* Na relacao temRelProc um PN nao se relaciona com ele proprio */
pred inv13 {
    all c: Classe_N3 | c->c not in eAntecessorDe
    all c: Classe_N3 | c->c not in eComplementarDe
    all c: Classe_N3 | c->c not in eCruzadoCom
    all c: Classe_N3 | c->c not in eSinteseDe
    all c: Classe_N3 | c->c not in eSintetizadoPor
    all c: Classe_N4 | c->c not in eSinteseDe
    all c: Classe_N4 | c->c not in eSintetizadoPor
    all c: Classe_N3 | c->c not in eSucessorDe
    all c: Classe_N3 | c->c not in eSuplementoDe
    all c: Classe_N3 | c->c not in eSuplementoPara
}

/* As relacoes eSinteseDe, eSucessorDe e eSuplementoDe sao antissimetricas. */
pred inv14 {
    Antisymmetric[Classe_N3<: eSinteseDe, Classe_N3]
    Antisymmetric[Classe_N3<: eSintetizadoPor, Classe_N3]
    Antisymmetric[Classe_N4<: eSinteseDe, Classe_N4]
    Antisymmetric[Classe_N4<: eSintetizadoPor, Classe_N4]

    Antisymmetric[eSucessorDe, Classe_N3]
    Antisymmetric[eAntecessorDe, Classe_N3]

    Antisymmetric[eSuplementoDe, Classe_N3]
    Antisymmetric[eSuplementoPara, Classe_N3]
}

/* Um PN so pode ter uma relacao com outro PN */
pred inv15 {
    all c1,c2: Classe_N3 | c1->c2 in eAntecessorDe ==> c1->c2 not in
        eComplementarDe + eCruzadoCom + eSinteseDe + eSintetizadoPor
        ↪ + eSucessorDe + eSuplementoDe + eSuplementoPara
    or
    all c1,c2: Classe_N3 | c1->c2 in eComplementarDe ==> c1->c2 not in
        eAntecessorDe + eCruzadoCom + eSinteseDe + eSintetizadoPor +
        ↪ eSucessorDe + eSuplementoDe + eSuplementoPara
    or
    all c1,c2: Classe_N3 | c1->c2 in eCruzadoCom ==> c1->c2 not in
        eAntecessorDe + eComplementarDe + eSinteseDe +
        ↪ eSintetizadoPor + eSucessorDe + eSuplementoDe +
        ↪ eSuplementoPara
    or
    all c1,c2: Classe_N3 | c1->c2 in eSinteseDe ==> c1->c2 not in
        eAntecessorDe + eComplementarDe + eCruzadoCom +
        ↪ eSintetizadoPor + eSucessorDe + eSuplementoDe +
        ↪ eSuplementoPara
    or
    all c1,c2: Classe_N3 | c1->c2 in eSintetizadoPor ==> c1->c2 not in
        eAntecessorDe + eComplementarDe + eCruzadoCom + eSinteseDe +
        ↪ eSucessorDe + eSuplementoDe + eSuplementoPara
    or
    all c1,c2: Classe_N3 | c1->c2 in eSucessorDe ==> c1->c2 not in
        eAntecessorDe + eComplementarDe + eCruzadoCom + eSinteseDe +
        ↪ eSintetizadoPor + eSuplementoDe + eSuplementoPara
    or
    all c1,c2: Classe_N3 | c1->c2 in eSuplementoDe ==> c1->c2 not in
        eAntecessorDe + eComplementarDe + eCruzadoCom + eSinteseDe +
        ↪ eSintetizadoPor + eSucessorDe + eSuplementoPara
    or
    all c1,c2: Classe_N3 | c1->c2 in eSuplementoPara ==> c1->c2 not in

```

```

        eAntecessorDe + eComplementarDe + eCruzadoCom + eSinteseDe +
        ↪ eSintetizadoPor + eSucessorDe + eSuplementoDe
    }

    /* Se um PN se desdobra em 4os niveis os Termos de Indice passam para os
       ↪ filhos */
    pred inv16 {
        all c: Classe_N3 | some c.temFilho ⇒ no c.temTI
    }

    /* Os Termos de Indice de um PN nao podem existir em mais nenhuma classe 3 */
    pred inv17 {
        all disj c1,c2: Classe_N3,ti: TermoIndice | ti in c1.temTI ⇒ ti not in
        ↪ c2.temTI
    }

    /* 2 DFs nao podem ter a mesma instancia de Justificacao */
    pred inv18 {
        all disj d1,d2: DestinoFinal | d1.temJustificacao ≠ d2.temJustificacao
    }

    /* 2 DFs nao podem ter a mesma instancia de Justificacao */
    pred inv19 {
        all disj pca1,pca2: PCA | pca1.temJustificacao ≠ pca2.temJustificacao
    }

    /* 2 PNs nao podem ter a mesma instancia de DF */
    pred inv20 {
        all disj c1,c2: Classe_N3 {
            c1.temDF ≠ c2.temDF
        }
        all disj c1,c2: Classe_N4 {
            c1.temDF ≠ c2.temDF
        }
        all disj c1: Classe_N3, c2: Classe_N4 {
            c1.temDF ≠ c2.temDF
        }
    }

    /* 2 PNs nao podem ter a mesma instancia de PCA */
    pred inv21 {
        all disj c1,c2: Classe_N3 {
            c1.temPCA ≠ c2.temPCA
        }
        all disj c1,c2: Classe_N4 {
            c1.temPCA ≠ c2.temPCA
        }
        all disj c1: Classe_N3, c2: Classe_N4 {
            c1.temPCA ≠ c2.temPCA
        }
    }

    /* Cada Justificacao tem no maximo 3 CriterioJustificacao diferentes */
    pred inv22 {
        all j: Justificacao | #j.temCriterio ≤ 3
    }

    /* 2 Justificacoes nao podem ter a mesma instancia de CriterioJustificacao */
    pred inv23 {
        all disj j1,j2: Justificacao | all c: CriterioJustificacao | c in j1.
        ↪ temCriterio ⇒ c not in j2.temCriterio
    }

    /* 2 Classes nao podem ter a mesma instancia NotaAplicacao
       2 Classes nao podem ter a mesma instancia NotaExclusao
       2 Classes nao podem ter a mesma instancia ExemploNotaAplicacao
    */
    pred inv24 {

```

```

    all disj c1,c2:Classe | all n:NotaAplicacao | n in c1.
      ↪ temNotaAplicacao ⇒ n not in c2.temNotaAplicacao
    all disj c1,c2:Classe | all n:NotaExclusao | n in c1.temNotaExclusao
      ↪ ⇒ n not in c2.temNotaExclusao
    all disj c1,c2:Classe | all n:ExemploNotaAplicacao | n in c1.
      ↪ temExemploNA ⇒ n not in c2.temExemploNA
  }

  /* 2 Classes N3 nao podem ter os mesmos filhos */
  pred inv25 {
    all disj c1,c2:Classe_N3 | all c3:Classe_N4 | c3 in c1.temFilho ⇒ c3
      ↪ not in c2.temFilho
  }

  /* 2 Classe_N3 nao podem ter o mesmo TI */
  pred inv26 {
    all disj c1,c2:Classe_N3 | all t:TermoIndice | t in c1.temTI ⇒ t not
      ↪ in c2.temTI
  }

  /* UM DF, na sua justificacao, apenas deve ter uma instancia de
     ↪ JustificacaoDF */
  pred inv27 {
    all df:DestinoFinal | df.temJustificacao in JustificacaoDF
  }

  /* UM PCA, na sua justificacao, apenas deve ter uma instancia de
     ↪ JustificacaoPCA */
  pred inv28 {
    all pca:PCA | pca.temJustificacao in JustificacaoPCA
  }

  /* Quando o PN (sem filhos) em causa 'eSuplementoPara' outro,
     deve ser acrescentado um criterio de utilidade administrativa na
     ↪ justificacao do respetivo PCA
  */
  pred inv29 {
    all c:Classe_N3 | no c.temFilho and some c.eSuplementoPara ⇒ one
      ↪ crit:CriterioJustificacaoUtilidadeAdministrativa {
        crit in c.temPCA.temJustificacao.temCriterio and crit.
          ↪ critTemProcRel = c.eSuplementoPara
      }
  }

  /* Um DF, na sua justificacao, devera conter apenas criterios de densidade
     ↪ informacional, complementaridade informacional e legal */
  pred inv30 {
    all df:DestinoFinal | all crit:df.temJustificacao.temCriterio {
      (crit in CriterioJustificacaoDensidadeInfo) or
      (crit in CriterioJustificacaoComplementaridadeInfo) or
      (crit in CriterioJustificacaoLegal)
    }
  }

  /* Um PCA, na sua justificacao, devera conter apenas criterios gestores,
     ↪ utilidade administrativa e legal */
  pred inv31 {
    all pca:PCA | all crit:pca.temJustificacao.temCriterio {
      (crit in CriterioJustificacaoGestionario) or
      (crit in CriterioJustificacaoUtilidadeAdministrativa) or
      (crit in CriterioJustificacaoLegal)
    }
  }

  /*
  Se um PN e (por ordem de prioridade):
    - eComplementarDe -> DF e de conservacao
    - eSinteseDe -> DF e de conservacao
  */

```

```

- eSintetizadoPor -> DF e de eliminacao
- nenhuma das acima -> DF e NE (Nao especificado)
*/
pred inv32 {
  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
  }
}

/* Quando o PN (com filhos) em causa 'eSuplementoPara' outro,
deve ser acrescentado um criterio de utilidade administrativa na
justificacao PCA dos filhos em que os processos relacionados com
o criterio podem ser distintos.
*/
pred inv33 {
  all c: Classe_N3 | some c.temFilho and some c.eSuplementoPara =>
  all c2: c.temFilho {
    one crit: CriterioJustificacaoUtilidadeAdministrativa
    ↪ {
      crit in c2.temPCA.temJustificacao.temCriterio
      crit.critTemProcRel in c.eSuplementoPara
    }
  }
}

/* Todos os processos relacionados (sem filhos) por uma relacao de sintese
↪ deverao
estar relacionados com o criterio de densidade informacional da
respetiva justificacao.
*/
pred inv34 {
  all c: Classe_N3 | no c.temFilho and (some c.eSinteseDe) => one crit:
  ↪ CriterioJustificacaoDensidadeInfo {
    crit in c.temDF.temJustificacao.temCriterio and crit.
    ↪ critTemProcRel = c.eSinteseDe
  }
  all c: Classe_N3 | no c.temFilho and (some c.eSintetizadoPor) => one
  ↪ crit: CriterioJustificacaoDensidadeInfo {
    crit in c.temDF.temJustificacao.temCriterio and crit.
    ↪ critTemProcRel = c.eSintetizadoPor
  }
}

/* Todos os processos relacionados (com filhos) por uma relacao de sintese
↪ deverao
estar relacionados com o criterio de densidade informacional da
respetiva justificacao.
*/
pred inv35 {
  all c: Classe_N3 | some c.temFilho and (some c.eSinteseDe) =>
  all c2: c.temFilho {
    one crit: CriterioJustificacaoDensidadeInfo {
      crit in c2.temDF.temJustificacao.temCriterio
      ↪ and crit.critTemProcRel = c.eSinteseDe
    }
  }
}

```

```

all c: Classe_N3 | some c.temFilho and (some c.eSintetizadoPor) ==>
  all c2: c.temFilho {
    one crit: CriterioJustificacaoDensidadeInfo {
      crit in c2.temDF.temJustificacao.temCriterio
      ↪ and crit.critTemProcRel = c.
      ↪ eSintetizadoPor
    }
  }
}

/*
  Todos os processos relacionados (sem filhos) pela relacao
  ↪ eComplementarDe,
  devem estar relacionados com o criterio de complementaridade
  informacional da respetiva justificacao.
*/
pred inv36 {
  all c: Classe_N3 | no c.temFilho and (some c.eComplementarDe) ==> one
  ↪ crit: CriterioJustificacaoComplementaridadeInfo {
    crit in c.temDF.temJustificacao.temCriterio and crit.
    ↪ critTemProcRel = c.eComplementarDe
  }
  all c: Classe_N3 | no c.temFilho and (some eComplementarDe.c) ==> one
  ↪ crit: CriterioJustificacaoComplementaridadeInfo {
    crit in c.temDF.temJustificacao.temCriterio and crit.
    ↪ critTemProcRel = eComplementarDe.c
  }
}

/*
  Todos os processos relacionados (com filhos) pela relacao
  ↪ eComplementarDe,
  os filhos devem estar relacionados com o criterio de
  ↪ complementaridade
  informacional da respetiva justificacao.
*/
pred inv37 {
  all c: Classe_N3 | some c.temFilho and (some c.eComplementarDe) ==>
  all c2: c.temFilho {
    one crit: CriterioJustificacaoComplementaridadeInfo {
      crit in c2.temDF.temJustificacao.temCriterio
      ↪ and crit.critTemProcRel = c.
      ↪ eComplementarDe
    }
  }
  all c: Classe_N3 | some c.temFilho and (some eComplementarDe.c) ==>
  all c2: c.temFilho {
    one crit: CriterioJustificacaoComplementaridadeInfo {
      crit in c2.temDF.temJustificacao.temCriterio
      ↪ and crit.critTemProcRel =
      ↪ eComplementarDe.c
    }
  }
}

/*
  Cada justificacao tem no maximo 1 Criterio de cada tipo.
*/
pred inv38 {
  all j: Justificacao {
    lone crit1: CriterioJustificacaoComplementaridadeInfo | crit1
    ↪ in j.temCriterio
    lone crit2: CriterioJustificacaoDensidadeInfo | crit2 in j.
    ↪ temCriterio
    lone crit3: CriterioJustificacaoGestionario | crit3 in j.
    ↪ temCriterio
    lone crit4: CriterioJustificacaoLegal | crit4 in j.temCriterio
  }
}

```

```
        lone crit5: CriterioJustificacaoUtilidadeAdministrativa |  
            ↪ crit5 in j.temCriterio  
    }
```