Silveira, Paulo Enes da

Componentes do conhecimento em estruturas de dados persistentes : aplicação em sistemas de decisão autónomos

http://hdl.handle.net/11067/5217

Metadados

Data de Publicação 2010

Resumo Os dados persistentes representam fenómenos reais ou conceptuais, arquivados de forma persistente, com as suas descrições e comportamentos em memória secundária de computadores, concretamente, em bases de dados orientadas por objectos. Neste artigo, desenvolve-se o conceito de Operador associado a Tipos Abstractos de Dados (O·TAD) que representa conhecimento sobre esses TAD (os supracitados dados persistentes). Este O·TAD permite verificar condições e agir autonomamente, sempre que o estado dos ...

Persistent data represent real or conceptual phenomena, stored in a persistent way, with their descriptions and behaviours, in secondary storage, specially in object-oriented databases. This paper presents the concept – Operator associated to Abstract Data Types (O·ADT) -, revealing knowledge about the ADT (the mentioned persistent data). This O·ADT can verify conditions and act in an autonomous way, whenever the ADT state is changed. The used ADT are those from the HBSD model (Hypergraph-Based ...

Palavras Chave Tipos abstractos de dados (Informática), Estruturas de dados (Informática)

Tipo article

Revisão de Pares Não

Coleções [ULL-FCEE] LEE, n. 10 (2010)

COMPONENTES DE CONHECIMENTO EM ESTRUTURAS DE DADOS PERSISTENTES: APLICAÇÃO EM SISTEMAS DE DECISÃO AUTÓNOMOS

Paulo Enes da Silveira
Doutor em Informática (Université Pierre et Marie Curie, Paris 6)
Doutor em Engenharia Electrotécnica e de Computadores
(Fac. Engenharia da Universidade do Porto)
Professor na Universidade Lusíada

Resumo: Os dados persistentes representam fenómenos reais ou conceptuais, arquivados de forma persistente, com as suas descrições e comportamentos em memória secundária de computadores, concretamente, em bases de dados orientadas por objectos. Neste artigo, desenvolve-se o conceito de Operador associado a Tipos Abstractos de Dados (O·TAD) que representa conhecimento sobre esses TAD (os supracitados dados persistentes). Este O·TAD permite verificar condições e agir autonomamente, sempre que o estado dos TAD seja alterado. Os TAD considerados são os do modelo HBDS (Estruturas de Dados Baseadas em Hipergrafos).

Este modelo inclui componentes formais de definição do O·TAD, a sua algoritmia e representação gráfica, os quais permitem considerar conhecimento associado aos dados persistentes, que passam a ter uma autonomia de comportamento reactivo às suas alterações de estado, dando origem a Sistemas de Decisão Autónomos. São apresentados casos de aplicação prática.

KNOWLEDGE COMPONENTS IN PERSISTENT DATA-STRUCTURES:
- APPLICATION IN AUTONOMOUS DECISION SYSTEMS

Abstract: Persistent data represent real or conceptual phenomena, stored in a persistent way, with their descriptions and behaviours, in secondary storage, specially in object-oriented databases. This paper presents the concept - Operator associated to Abstract Data Types (O·ADT) -, revealing knowledge about the ADT (the mentioned persistent data). This O·ADT can verify conditions and act in an autonomous way, whenever the ADT state is changed. The used ADT are those from the HBSD model (Hypergraph-Based Data Structure).

This model contains the formal definitions for these O·ADT, their algorithmic and graphical representation that allow reproducing knowledge associated to persistent data, which can be reactive to their state changes, creating therefore, Autonomous Decision Systems. Some case-studies are presented.

1. Introdução

O conceito de Operador associado aos Tipos Abstractos de Dados (O·TAD) foi pela primeira vez apresentado pelo autor em [1], associando componentes de conhecimento aos TAD do modelo HBDS (Estruturas de Dados Baseadas em Hipergrafos) da autoria de François Bouillé [2], com a intenção de conferir às estruturas de dados persistentes, como as bases de dados, uma autonomia de decisão em acções que se possam executar em face de alterações do estado dos dados nelas contidos.

As estruturas de dados persistentes são, no contexto deste artigo, representadas pelos TAD (Tipos Abstractos de Dados) do modelo HBDS, o qual será sucintamente apresentado, a fim de se compreender e dar contexto à acção dos O·TAD.

Estes Operadores associados a Tipos Abstractos de Dados serão detalhadamente abordados, com a sua definição, a descrição dos seus vários tipos, a sua sintaxe, grafismo e a sua algoritmia, com recurso ao EXEL [3] [4], uma linguagem algorítmica que também irá ser apresentada.

Uma vez definidos os O·TAD explica-se como estes podem ser utilizados para construir Sistemas de Decisão Autónomos (SDA) e apresentam-se casos de aplicação prática.

2. Os Tipos Abstractos de Dados do HBDS

2.1. Introdução

A acção dos Operadores que vai ser desenvolvida no próximo parágrafo, sendo associada aos TAD do HBDS, justifica a apresentação destes Tipos Abstractos de Dados.

O modelo HBDS surge como um modelo lógico e físico de base de dados, onde a questão da redundância de dados foi resolvida. Contendo uma formulação gráfica de superiores capacidades, quando comparada com os diagramas de Entidade-Relação [5] ou os Diagramas de Classes do UML [6], tem sido utilizado para modelização nos mais variados domínios, tais como os Sistemas de

Informação Geográficos [15] [16].

O conceito de Tipo Abstracto de Dados foi apresentado por Barbara Liskov[1] e Stephen Zilles [7]. François Bouillé utiliza-o para designar os tipos que constituem a base do HBDS para a modelização dos fenómenos do mundo real ou conceptual.

2.2. Origem e princípios matemáticos

O HBDS comporta um modelo lógico para estruturação dos fenómenos e um modelo físico para a sua implementação em software e em hardware. Vamos abordar a origem e os princípios matemáticos do modelo lógico.

Pode descrever-se uma coisa, através das suas propriedades, identificando-a com um nome e descobrindo as relações que ela tem com o seu ambiente. A Teoria dos Conjuntos tem esta aproximação e está na base dos seis TAD do HBDS.

Quando no processo de identificação dessas coisas, estas se agrupam por terem propriedades comuns e se observa que estes grupos podem fazer parte de um grupo mais alargado, o conceito de Hipergrafo de Claude Berge [8], visto como uma família de subconjuntos de um conjunto, aparece como o alicerce do HBDS, tal como a sua sigla aponta: Hypergraph-Based Data Structure.

Ole-Johan Dahl e Kristen Nygard conceberam a linguagem de programação SIMULA [9] nos anos 60, com a introdução do conceito de Objecto na Programação Orientada por Objectos (POO), facto este que valeu aos os seus autores a distinção do prémio "ACM A. M. Turing Award", em 2002. François Bouillé inspirou-se nos conceitos da POO do SIMULA, ao conceber o HBDS. Não será por acaso que a apresentação do HBDS é feita na tese de doutoramento de Bouillé, sendo presidente do júri da sua prova de doutoramento o próprio professor Dahl.

Referimos ainda o conceito de Grafo que está presente na formulação do HBDS, conferindo-lhe uma importante consistência matemática.

2.3. Os seis TAD do HBDS e sua representação gráfica

2.3.1. Classe, Atributo de Classe e Ligação entre Classes

Os TAD que constituem a base do modelo lógico do HBDS derivam da Teoria dos Conjuntos e da terminologia da linguagem SIMULA e da POO.

Seguidamente apresentam-se os TAD-HBDS, nomeados apenas por TAD e as suas representações gráficas (Fig. 1).

Um Conjunto que reune Elementos com Propriedades comuns é equivalente

[1] Barbara Liskov, professora e investigadora no MIT, recebeu recentemente o "2008 ACM A. M. Turing Award", "pelos seus contributos para os fundamentos práticos e teóricos das linguagens de programação e desenho de sistemas, especialmente a abstracção de dados[…]" [*in* www.acm.org]

a um TAD designado de Classe (C) representada por uma elipse e por um nó (pequena circunferência) .

Uma Propriedade de um Conjunto é equivalente ao TAD designado por Atributo de Classe (A) e representado por um pequeno quadrado ligado por um segmento de recta ao nó da Classe.

Uma Relação entre dois Conjuntos é equivalente ao TAD designado por Ligação entre Classes (L) patenteada por uma ligação em arco entre os nós das Classes.

O TAD L pode representar uma Ligação entre duas Classes, mas também uma Ligação de uma Classe sobre si própria.

Estes três TAD – C, A, L – são conceptuais (TAD-conceptuais), pois com eles pode modelizar-se uma estrutura de dados com a possibilidade de existência de fenómenos de um domínio em análise.

O significado de *estrutura* e de *domínio* utilizados neste contexto, são explicados pelo autor em [10].

2.3.2. Objecto, atributo de Objecto e ligação entre Objectos

Um Elemento de um Conjunto é equivalente a um TAD designado de Objecto (O) pertencente a uma Classe e representado por um pequeno círculo preto .

Uma propriedade de um Elemento de um Conjunto é equivalente ao TAD designado por atributo de Objecto (a), representado por um pequeno quadrado preto, ligado por um segmento de recta ao círculo do Objecto.

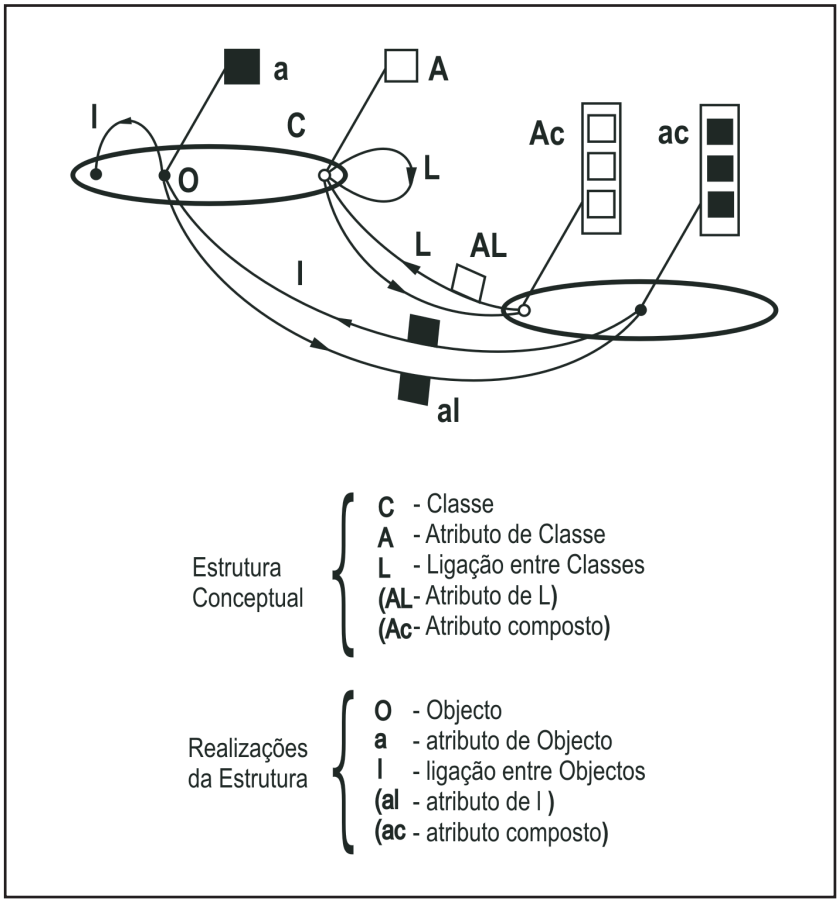


Fig. 1 – Representação gráfica dos TAD-HBDS

Uma relação entre dois Elementos de Conjuntos é equivalente ao TAD designado por ligação entre Objectos (I), representada por uma ligação em arco entre os dois círculos dos Objectos.

Uma ligação entre Objectos concretiza uma Ligação existente entre duas Classes, mas também pode materializar a existência de uma Ligação entre dois Objectos pertencentes à mesma Classe.

Objectos com atributos (a) comuns pertencem à mesma Classe.

Estes três TAD – O, a, I – são as realizações (TAD-realização) dos respectivos TAD-conceptuais – C, A, L – .

2.4. Extensões aos TAD do HBDS e sua representação gráfica

2.4.1. Hiperclasse, Hiperatributo e Hiperligação

Quando várias Classes apresentam Atributos comuns, pode considerar-se uma nova Classe, - uma Hiperclasse (HC) - com esses Atributos comuns - Hiperatributos (HA) - que inclua essas Classes (Fig. 2). Uma Hiperclasse é representada por um rectângulo com os cantos arredondados e um nó (pequena circunferência). Um Hiperatruibuto é representado de forma semelhante a um Atributo.

Conjuntos de Ligações entre dois conjuntos de Classes correspondentes a duas Hiperclasses, podem ser representados por Hiperligações (HL). Uma Hiperligação reune Ligações entre Classes incluídas em duas Hiperclasses e é reproduzida como uma Ligação entre as suas Hiperclasses.

2.4.2. Multiligação

Uma reunião de várias Ligações entre duas Classes pode dar origem a uma Multiligação (ML) representada por uma Ligação a negrito. Pode existir mais de uma Multiligação entre duas Classes, cada uma reunindo um subconjunto de Ligações entre essas Classes (Fig. 2).

2.4.3. Atributo de Ligação e Atributo composto

Quando é necessário qualificar uma Ligação, pode considerar-se um Atributo de Ligação entre Classes e os respectivos atributos de ligação entre Objectos. Um Atributo de Ligação mascara um Atributo de uma Classe que representa a Ligação em causa. A sua representação faz-se com um quadrado com um lado sobre a Ligação (Fig. 1).

É possível agrupar vários Atributos num só, designado por Atributo Composto e representado por um rectângulo envolvendo os quadrados dos Atributos (Fig. 1).

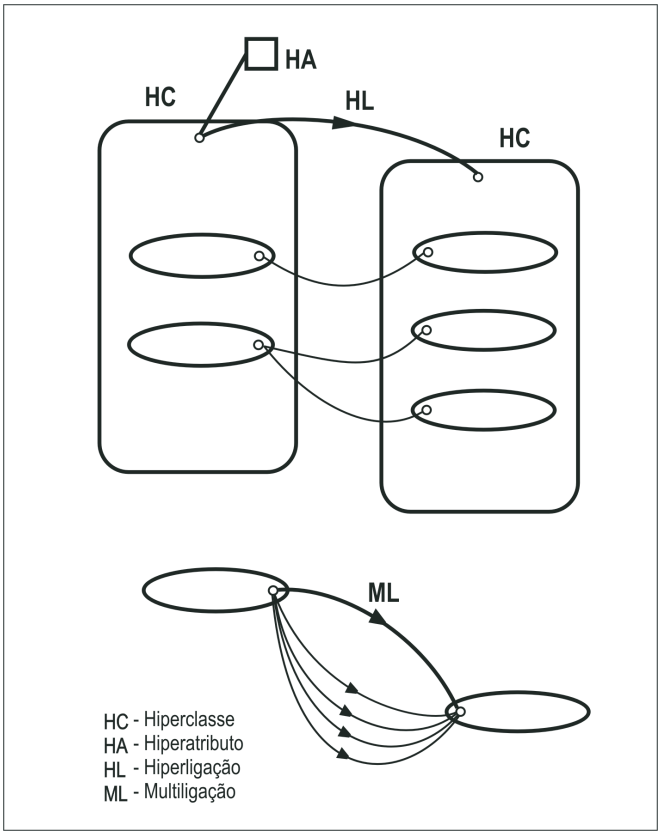


Fig. 2 – Representação gráfica das extensões dos TAD-HBDS

2.5. Dois Multigrafos e Arborescências no HBDS

É importante referir os princípios matemáticos: Multigrafos e Arborescência.

Existem dois Multigrafos, um para as Classes que tem como componentes os nós das Classes e as Ligações entre estas, e outro para os nós dos Objectos e ligações entre estes, os quais asseguram a separação clara entre os grafos das Classes e os grafos dos Objectos, não podendo haver qualquer conexão entre os dois Multigrafos. Deste modo se garante que as ligações entre Classes e ligações entre Objectos sejam disjuntas (Fig. 3) .

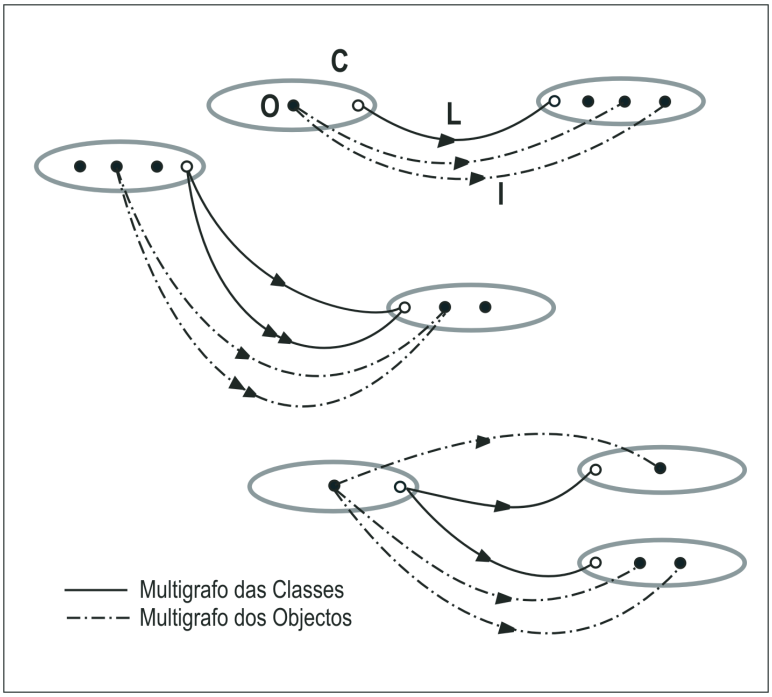


Fig. 3 – Dois Multigrafos: **C**lasses/**L**igações e **O**bjectos/**l**igações

A Arborescência contém várias árvores que simbolizam as possíveis hierarquias representativas das heranças entre Classes do HBDS. As relações hierárquicas entre Classes são representadas por segmentos de recta a negrito entre essas Classes (Fig. 4). As heranças são simples: cada Classe-filha tem apenas uma Classe-mãe. As Classes-filha acumulam implicitamente os Atributos das Classes-mãe.

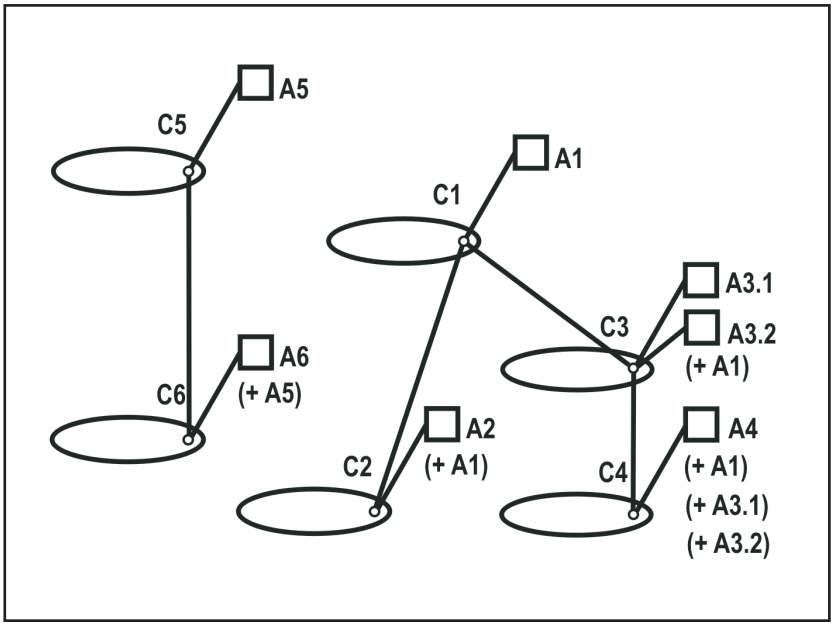


Fig. 4 – Arborescência com duas árvores representando as heranças no HBDS

2.6. Sintaxe dos TAD

Os TAD-HBDS têm uma sintaxe própria para que possam ser utilizados em algoritmia.

Segue-se, no Quadro 1, a exposição da sintaxe dos TAD com a EBNF (Extended Backus Naür Form), que utilizaremos ao longo deste artigo e um exemplo do domínio automóvel com esquema gráfico (Fig. 5).

Sintaxe C **::=** <nome C>

Exemplo: **AUTOMÓVEL**

Sintaxe A **::=** <nome C> • <nome A>

Exemplo: **AUTOMÓVEL • Marca**

Sintaxe L **::=** <nome C1> <nome L> <nome C2>

Exemplo: **AUTOMÓVEL circula_na VIA**

Sintaxe O **::=** <nome O> [<nome C>]

Exemplo: **43-PT-33 [AUTOMÓVEL]**

Sintaxe a **::=** <nome O> [<nome C>] • <nome A>

Exemplo: **43-PT-33 [AUTOMÓVEL] • Velocidade_real**

Sintaxe l **::=** <nome O1> [<nome C1>] <nome L> <nome O2> [<nome C2>]

Exemplo: **43-PT-33 [AUTOMÓVEL] circula_na IC23[VIA]**

Quadro 1 – Sintaxe EBNF dos TAD-HBDS

2.7. Operações sobre os TAD-HBDS

Os Tipos Abstractos de Dados do HBDS estão sujeitos às seguintes operações:

Criar e Eliminar; Modificar; Destruir e Recuperar; Hibernar e Ressuscitar.

As três primeiras são as mais usuais. A operação Criar constitui lógica e fisicamenete o TAD, criando a sua persistência, ou seja, a sua existência em memória secundária, internamente identificável através de um endereço. A operação Destruir executa a destruição lógica do TAD que continua a existir fisicamente e se pode Recuperar. A operação Hibernar permite ocultar o TAD que continua a existir lógica e fisicamente e que se pode Ressuscitar. As operações Recuperar e Ressuscitar devolvem o TAD ao seu estado original. A operação Eliminar, exclui lógica e fisicamente o TAD, terminando o seu estado de persistência.

	Criar	Eliminar	Modificar	Destruir	Recuperar	Hibernar	Ressuscitar
C	✓	✓		✓	✓	✓	✓
A	✓	✓		✓	✓	✓	✓
L	✓	✓		✓	✓	✓	✓
O	✓	✓		✓	✓	✓	✓
a			✓				
l			✓				

Quadro 2 – Operações válidas sobre os TAD-HBDS

No Quadro 2, indica-se para cada TAD as operações que são válidas. Os TAD **C**, **A**, **L** e **O** têm um nome, como um símbolo, sem valores, e permitem as operações Criar, Eliminar, Destruir, Recuperar, Hibernar e Ressuscitar.

Os TAD **a** (de tipo Inteiro, Real, Booleano, Cadeia de caracteres, Tabelas, ... , Algoritmo) e **l** (de tipo Booleano - a ligação entre Objectos é Verdadeira - True - quando existe entre dois Objectos e é Falsa - False - quando não existe) têm um nome e têm valores que podem variar com a operação Modificar.

No contexto deste artigo, vamos utilizar as operações Criar, Eliminar e Modificar, representadas pelos símbolos: ↑, ↓ , ←.

Para construir a estrutura HBDS da Fig. 5, utiliza-se a sintaxe indicada no Quadro 3.

↑ AUTOMÓVEL, VIA;

↑ AUTOMÓVEL • Marca, AUTOMÓVEL • Velocidade_real;

↑ AUTOMÓVEL circula_na VIA; (a sua inversa é também criada: VIA é_atravessada _por AUTOMÓVEL)

↑ 44-PT-33 [AUTOMÓVEL]; ↑ IC23[VIA];

44-PT-33 [AUTOMÓVEL] circula_na IC23[VIA] ← True; (a sua inversa é também criada: IC23[VIA] é_atravessada _por 44-PT-33 [AUTOMÓVEL];)

44-PT-33 [AUTOMÓVEL] • Marca ← "BMW";

44-PT-33 [AUTOMÓVEL] • Velocidade_real ← 80;

Quadro 3 – Criação de estrutura HBDS

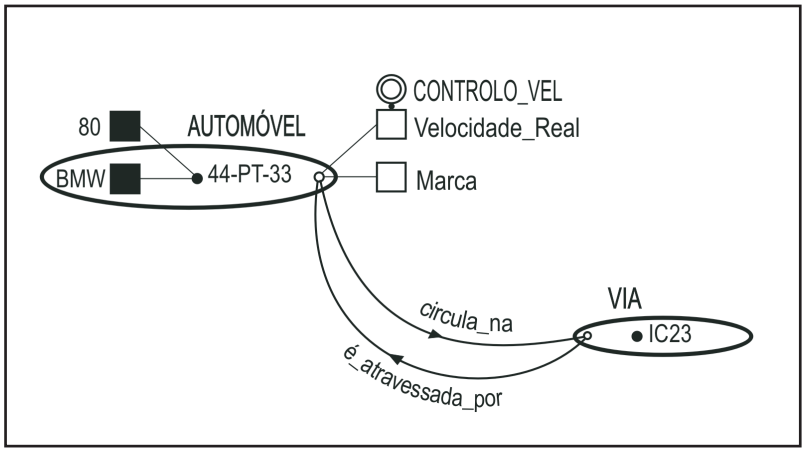


Fig. 5 – Estrutura gráfica HBDS correspondente às operações do Quadro 3 com Operador CONTROLO_VEL

2.8. Outras abordagens aos TAD-HBDS

O HBDS tem sido objecto de vários estudos que lhe têm ampliado a sua área de intervenção. A transformação automática de estruturas foi estudada por Jean Révélat [11], o que permitiu estabelecer as normas e a algoritmia da transformação dos TAD-HBDS, como por exemplo, a transformação de um Atributo em Classe ou de uma Ligação em Classe e as consequentes transformações entre os TAD associados.

O contributo de Lofti Zadeh com os Fuzzy Data [12] (Dados Vagos) fez com que Bouillé estudasse a sua aplicação ao HBDS [13] de tal modo que se pode considerar que um Objecto tem uma relação de pertença "fuzzy" (vaga) a uma Classe.

Os conceitos dos Expert Systems (Sistemas Periciais) foram também introduzidos no HBDS criando os Structured Expert Systems [14].

Os TAD-HBDS têm sido muito utilizados em modelização [15], [16] e suporte de software [17] de Sistemas de Informação Geográfica.

3. Operadores associados aos TAD

3.1. Introdução ao conceito de Operador associado ao TAD

Ao associar um conjunto de condições e de acções a um TAD, as quais podem ser executadas automaticamente quando o estado do TAD for alterado, através de uma ou mais operações permitidas sobre este, conferimos um certo

conhecimento associado ao TAD, que representamos como um Operador associado a esse TAD (*O*·TAD).

O *O*·TAD tem um algoritmo (código a ser executado) que traduz as condições e acções associadas ao TAD. Quando uma operação altera o estado do TAD, é feita automaticamente a verificação da existência de algum Operador associado a este TAD, e se as condições expressas na definição deste Operador se verificarem, são executadas as correspondentes acções.

3.2. Definição de *O*·TAD

3.2.1. Definição de *O*·TAD com EBNF

Definimos Operador (*O*) associado a um TAD (*O*·TAD) , com um nome que o identifica e com um n-pleto de cinco componentes, descritos no Quadro 4 e explicados no ponto seguinte.

O•TAD **::=** <nome *O*•TAD> ≈ <n-pleto *O*•TAD>

O símbolo terminal ≈ significa - é definido por -.

<nome *O*•TAD> **::=** <conjunto de caracteres idêntico ao do nome de uma variável>

<n-pleto *O*•TAD> **::=**
(<tipo de *O* >,<nome TAD associado>, <antes|depois>, <operações>,<ref. código *O* >)

<tipo de *O* > **::=** <*O* tipo conjunto> | <*O* tipo elemento>

<*O* tipo conjunto> **::=** *O*•CO | *O*•Aa | *O*•Ll

<*O* tipo elemento> **::=** *O*•C | *O*•A | *O*•L | *O*•O | *O*•a | *O*•l

<nome TAD associado> **::=** <nome C> | <nome A> | <nome L> | <nome O> | <nome a>
| <nome l>

<antes|depois> **::=** antes | depois

<operações> **::=** Criar | Eliminar | Modificar | { Criar, Eliminar }

<ref. código *O* > **::=** <nome *O*•TAD> ( )

Quadro 4 – Sintaxe EBNF da definição dos Operadores associados aos TAD-HBDS

3.2.2. Os cinco componentes que definem um (*O*·TAD)

3.2.2.1. O tipo dos Operadores e sua representação gráfica

O <tipo de *O*> pode ser de tipo conjunto e de tipo elemento.

O Operador é <*O* tipo conjunto> quando for associado aos TAD – C, A, L – (TAD conceptuais), dizendo respeito respectivamente aos TAD-realização – O, a, l – sendo assim três os Operadores de tipo conjunto – *O*·CO, *O*·Aa e *O*·Ll –.

Estes Operadores de tipo conjunto são executados automaticamente da seguinte forma:

O·CO – executa-se quando for realizada uma operação permitida (Criar | Eliminar | Criar, Eliminar) sobre o TAD-realização **O**, indicada na componente <operações> do n-pleto que o define;

O·Aa – executa-se quando for realizada uma operação permitida (Modificar) sobre o TAD-realização **a**, indicada na componente <operações> do n-pleto que o define;

O·Ll – executa-se quando for realizada uma operação permitida (Modificar) sobre o TAD-realização **l**, indicada na componente <operações> do n-pleto que o define.

O Operador é <*O* tipo elemento> quando individualmente for associado a um dos seis TAD – C, A, L, O, a, l – sendo assim seis os Operadores de tipo elemento – *O*·C, *O*·A, *O*·L, *O*·O, *O*·a e *O*·l.

Os Operadores do tipo elemento são executados automaticamente quando for realizada uma operação sobre o respectivo TAD associado, da forma seguinte:

- *O*·C, *O*·A, *O*·L e *O*·O – cada um destes Operadores executa-se quando for realizada a única operação permitida (Eliminar) sobre o TAD a que está respectivamente associado – **C, A, L** e **O** –, indicada na componente <operações> do n-pleto que os define. Neste contexto, não é possível considerar a operação Criar sobre os TAD **C, A, L** e **O**, uma vez que estes TAD não existindo, não poderiam ter associado Operador algum para ser executado;

- *O*·a e *O*·l - executam-se quando for realizada a única operação permitida (Modificar) sobre o TAD-realização a que está respectivamente associado – **a**, **l** –, indicada na componente <operações> do n-pleto que os define.

O *O*·TAD é representado graficamente por duas circunferências concêntricas e um ponto de associação que o fixa à representação gráfica do TAD.

Na Fig. 6, representam-se os Operadores do tipo conjunto e de tipo elemento, na parte direita da figura. À esquerda, mostram-se os Operadores do tipo conjunto – *O*·CO, *O*·Aa e *O*·Ll – como Classes de Operadores e as realizações dos seus Objectos-Operadores – *O*·co, *O*·aa e *O*·ll – graficamente semelhantes aos outros, mas com o círculo interior de cor preta.

As representações gráficas desses Objectos-Operadores das Classes de *O* tipo conjunto são mostrados, para uma melhor compreensão da actuação destes *O* tipo conjunto. Na prática, não se costumam representar, uma vez que esses

Objectos-Operadores existem implicitamente associados aos TAD-realização:

- os O·co associados aos Objectos da Classe que tem associado um O·CO;
- os O·aa associados aos atributos de Objecto, realizações do Atributo de Classe que tem associado um O·Aa;
- os O·ll associados às ligações entre Objecto, realizações da Ligação entre Classes que tem associado um O·Ll;

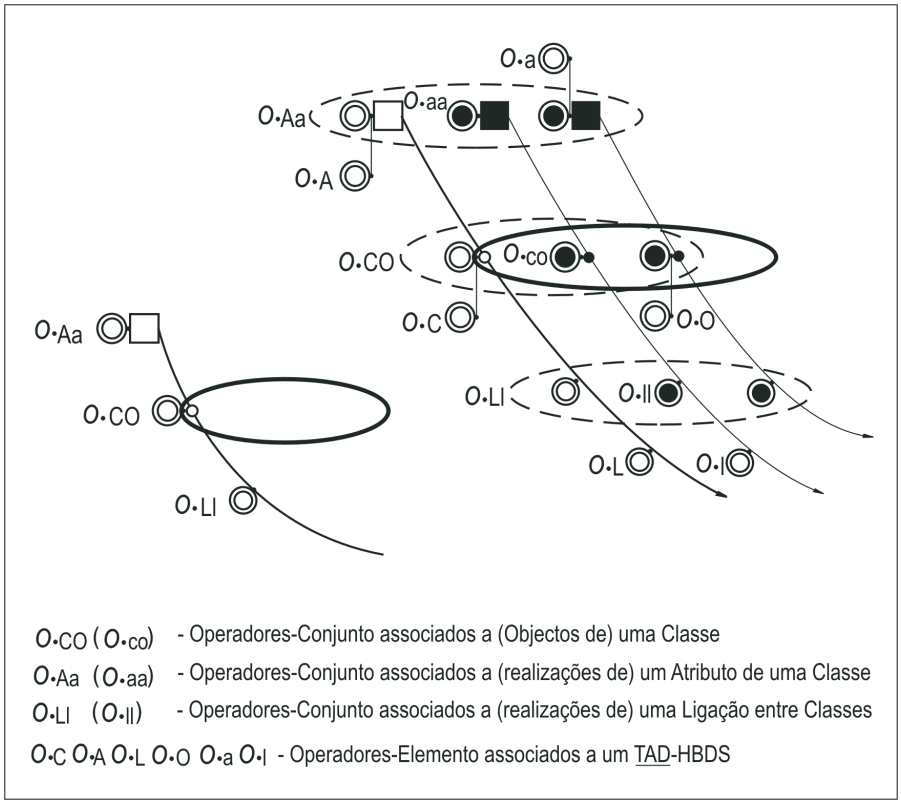


Fig. 6 – Representação gráfica dos Operadores associados aos TAD-HBDS

3.2.2.2. O nome do TAD associado

O <nome TAD associado> indica o nome do TAD sobre o qual uma operação permitida desencadeia a execução do Operador. Para os O tipo conjunto – O·CO, O·Aa e O·Ll – os nomes dos TAD associados são respectivamente os nomes de **C**, **A** e **L**. Para os O tipo elemento – O·C, O·A, O·L, O·O, O·a e O·l – os nomes dos TAD associados são respectivamente os nomes de **C**, **A**, **L**, **O**, **a** e **l**.

3.2.2.3. A execução do Operador antes ou depois da Operação

Quando uma Operação permitida sobre um TAD desencadeia um Operador a este associado, este Operador pode ser executado antes ou depois da Operação, conforme for necessário no contexto da aplicação do O·TAD em causa.

3.2.2.4. Operações sobre os TAD

O componente <operações> indica as Operações que forem realizadas sobre um TAD que desencadeiam a execução do Operador associado a este TAD. Estas Operações podem ser Criar, Eliminar ou ambas, sobre os TAD **C**, **A**, **L** e **O**, e ainda Modificar sobre os TAD **a** e **l**.

3.2.2.5. O código do Operador

A referência ao código do Operador que for executado é feita invocando um módulo com o nome do próprio Operador <nome O·TAD> ( ).

3.2.3. Exemplo de aplicação de O·TAD

Consideremos um Operador que tenha como função avisar o condutor de uma viatura a circular que ultrapassou uma velocidade determinada, tendo como base a estrutura representada na Fig. 5.

Para que todas os Objectos viaturas da Classe Viatura possam ter esse controlo de velocidade real, devemos optar por um Operador de tipo conjunto associado ao Atributo Velocidade_real (Quadro 5).

CONTROLO_VEL ≈ (O•Aa, **AUTOMÓVEL • Velocidade_real**, depois, Modificar, CONTROLO_VEL( ))

Quadro 5 – Exemplo de definição de um O·TAD

A definição deste Operador traduz-se por: O Operador de nome CONTORLO_VEL é definido como sendo do tipo conjunto, associado ao Atributo **AUTOMÓVEL · Velocidade_rea**l, e deve ser executado depois do atributo de Objecto **44-PT-33 [AUTOMÓVEL] · Velocidade_real** ser modificado, com o código contido em CONTROLO_VEL( ).

A execução, deste código acontece quando for modificada a velocidade real do Automóvel, que é actualizada em tempo-real, com o módulo CONTROLO_ VEL( ) que detecta quando a velocidade real do Automóvel ultrapassa um certo

valor dado (Fig. 5).

Salientamos que só se devem associar Operadores aos TAD dinâmicos que estiverem sujeitos a alterações de estado, ou seja, sujeitos a operações durante o seu ciclo de vida. Não faz sentido associar-se um Operador a um TAD estático que durante o seu ciclo de vida não sofra alteração alguma, pois este Operador nunca se iria executar.

3.3. Algoritmia dos O·TAD

3.3.1. Linguagem Algorítmica EXEL

O código dos Operadores vai ser descrito pelo seu algoritmo, utilizando a linguagem algorítmica EXEL, proposta por Jacques Arsac [3], posteriormente estendida por François Bouillé e pelo autor [4].

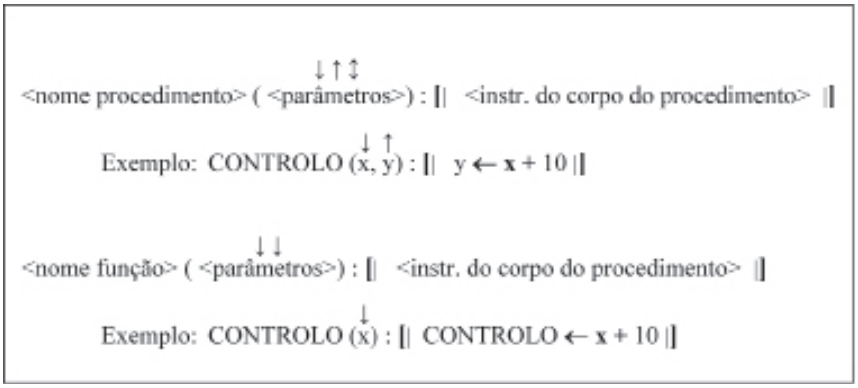
O EXEL pretende descrever um algoritmo de forma independente das linguagens de programação e comporta um pequeno conjunto de símbolos que representam as instruções fundamentais da programação – atribuição, selecção e ciclo – escritas sequencialmente da esquerda para a direita, como um texto, separadas pelo símbolo ponto e vírgula – ; –, descritas no Quadro 6.

Instruções	Significado	Exemplo
<var> ¬ <expr>	Atribuição (variável é atribuída com o valor da expressão)	X ¬120
<c> **?** <V> **|** <F> **¿**	Selecção (se a condição **c** é verdadeira executar as instruções **V**, senão executar as instruções **F**)	X **>** 100 **?** X ¬ 100 **|** X ¬ 80 **¿**
{ <c>**? ! |¿ ;** <instr> **}**	Ciclo (executa as instruções <instr> até que se verifique a condição **c**, saindo) **!** – saída do nível onde se encontra (o ciclo)	i ¬ 0; **{** i>10 **? ! |¿ ;** <instr> **;** i ¬ i+1**}**

Quadro 6 – Instruções fundamentais do EXEL

Existem outras variantes de selecção e ciclo, bem como notação para programação em paralelo, que não são apresentadas neste contexto.

Os módulos de programação são representados pelo seu nome, com os parâmetros existentes em entrada, saída ou entrada e saída (Quadro 7).



Quadro 7 – Módulos Procedimento e Função em EXEL

3.3.2. Módulos dos *O*·TAD

São três dos tipos de módulos que contêm o algoritmo dos *O*·TAD, os quais têm um cabeçalho, constituído por nome do *O*·TAD e parâmetros de entrada, com quantificação e qualificação dos mesmos e um corpo com a descrição do algoritmo respectivo (Quadro 8).

Tipo de *O*•TAD	Cabeçalho e corpo de Módulo
O•CO *O*•Aa	<nome *O*•CO> (↓x): [| <algoritmo do *O*•CO> |] $\forall x \in$ <*nome C*> <nome *O*•Aa > (↓x): [| <algoritmo do *O*•Aa> |] $\forall x \in$ <*nome C*>
O•Ll	<nome *O*•Ll> (↓x,↓y): [| <algoritmo do *O*•Ll> |] $\forall x \in$ <*nome C1*> $\wedge$ $y \in$ <*nome C2*>
O•TAD do tipo elemento	<nome *O*•TAD-elemento > : [| <algoritmo do *O*•TAD-elemento > |]

Quadro 8 – Módulos dos *O*·TAD

Os módulos dos *O*·TAD do tipo conjunto – *O*·CO ou *O*·Aa – têm um único parâmetro de entrada **x**, sendo este um qualquer Objecto da Classe C, Classe esta a que está associado o *O*·CO (**x** coincide com o Objecto **O** sujeito à operação Criar ou Eliminar que desencadear o Operador) ou *O*·Aa (**x** é o Objecto **O** de que **a** é atributo sujeito à operação Modificar que o Operador desencadear).

Os módulos dos *O*·TAD do tipo elemento – *O*·C, *O*·A, *O*·L, *O*·O, *O*·a e *O*·I – não têm parâmetros, pois o algoritmo acede directamente ao respectivo TAD a que está associado o Operador, através do seu nome.

3.3.3. Exemplo de algoritmo de um *O*·TAD e seu funcionamento

Considerando o Operador definido no ponto 3.2.3., Quadro 5, CONTROLO_VEL, para controlar a velocidade de uma viatura, emitindo um aviso ao condutor quando ultrapassa uma dada velocidade, vamos elaborar o seu algoritmo, tendo como limite de velocidade a não ultrapassar, os 120 Km/h (Quadro 9).

Definição do Operador (Quadro 5):

CONTROLO_VEL ≈
(*O*•Aa, **AUTOMÓVEL • Velocidade_real**, depois, Modificar, CONTROLO_VEL( ))

Algoritmo do CONTROLO_VEL:

CONTROLO_VEL $\overset{\downarrow}{(x)}$**: [| x [AUTOMÓVEL] • Velocidade_real > 120 ? Avisar Condutor( ) | ¿ |]**
$\forall x \in$ ***AUTOMÓVEL***

Quadro 9 – Algoritmo do Operador CONTROLO_VEL

Com a definição do Operador CONTROLO_VEL e o seu algoritmo, fica completada a sua descrição.

Tem-se em conta um sistema de engenharia e de informação, no qual cada automóvel em circulação possa ter sempre a sua velocidade real actualizada.

Sempre que a Velocidade real de um automóvel **x** se modificar, o Operador CONTROLO_VEL executa-se, verificando se essa velocidade ficou superior a 120. Em caso afirmativo, executa-se Avisar Condutor( ). Em caso negativo não há lugar a qualquer acção a executar.

A execução do Operador CONTROLO_VEL é automática quando varia a Velocidade real de um automóvel e não se executa se esta velocidade se mantiver constante, pois não é feita operação alguma de Modificação do atributo Velocidade real.

4. Sistemas de Decisão Autónomos

O conceito definido de Operador associado aos TAD, juntamente com a representação da estrutura de dados persistente de um domínio de aplicação,

permite criar estruturas de conhecimentos onde se representam conhecimentos sobre o comportamento dos TAD.

Sempre que o estado destes TAD é alterado através de operações de Criar, Eliminar e Modificar (bem como as outras não exploradas no âmbito deste artigo: Destruir e Recuperar, Hibernar e Ressuscitar), os Operadores que contêm componentes de conhecimento, executam-se automática e autonomamente, dando origem a Sistemas que denominamos de Sistemas de Decisão Autónomos (SDA).

Estes Sistemas de Decisão são Autónomos pois podem definir-se os Operadores associados aos TAD que forem necessários, para representar, de forma persistente e autónoma, os conhecimentos traduzidos pelas acções que devem ser activadas, sempre que forem verificadas condições definidas em domínios de aplicação considerados.

O funcionamento autónomo dos Operadores associados aos TAD, resulta da sua execução ser desencadeada na consequência de alterações do estado desses TAD, os quais representam os objectos reais ou conceptuais de um domínio de aplicação.

A definição de Operadores associados aos TAD, num certo domínio de aplicação, pode ser feita em tempo real, mesmo durante a vida do sistema em causa, sempre que se entenda útil, podendo assim juntar-se a um SDA novos componentes de conhecimento.

5. Caso de aplicação dos *O*·TAD: SDA no controlo de velocidades de viaturas em circulação

5.1. Controlo da Velocidade real da Viatura ao circular num Troço

Consideremos o caso do controlo de velocidades reais de viaturas em circulação em troços de vias, em relação à velocidade máxima permitida em cada um deles.

No sistema viário e nas viaturas, devem existir os componentes técnicos necessários para que um sistema de engenharia e de informação central possa ter a imagem das viaturas, suas posições e velocidades reais, bem como todas as vias e troços onde circulam, com as respectivas velocidades máximas permitidas.

Representamos na Fig. 7 a estrutura HBDS resultante das seguintes operações, já iniciadas no ponto 2.7, Quadro 3, agora actualizado no Quadro 10.

São criadas: As Classes Viatura, Via e Troço; Os Atributos da Viatura, Marca e Velocidade real; O Atributo da Via, Comprimento; Os Atributos do Troço, Comprimento e Velocidade máxima permitida; As Ligações entre Classes, Viatura circula no Troço e Troço pertence à Via; Os Objectos 44-PT-33 da Classe Viatura, Estrada Nacional n.º 9 – EN9 – da Classe Via e os Troços n.º 1 e n.º 2 – TR1 e TR2 – da Classe Troço.

São modificadas para verdadeiras as ligações entre Objectos, Viatura 44-TP-33 circula no Troço TR1 e o Troço TR1 pertence à Via EN9; São modificados os atributos de Objectos, Velocidade máxima permitida dos Troços TR1 e TR2 e Velocidade real da Viatura 44-TP-33, aos quais são atribuídos 80, 60 e 80 km/h, bem como o atributo Marca da Viatura criada, com o valor "BMW".

O atributo Velocidade máxima permitida do Troço TR1 é estático, pois após a sua atribuição de valor 80, este não varia ao longo do tempo (em circunstâncias normais), enquanto que o atributo Velocidade real da Viatura é dinâmico, já que varia conforme a aceleração que é imprimida à Viatura.

```
↑ VIATURA, VIA, TROÇO ;

↑ VIATURA • Marca, VIATURA • Velocidade_real;

↑ VIA • Comprimento;

↑ TROÇO • Comprimento, TROÇO • Velocidade_máx_permitida;

↑ VIATURA  circula_no  TROÇO; ↑ TROÇO pertence_à  VIA;

↑ 44-PT-33 [VIATURA]; ↑ EN9 [VIA]; ↑ TR1 [TROÇO], TR2 [TROÇO];

44-PT-33 [VIATURA] circula_no  TR1[TROÇO] ← True;

TR1 [TROÇO] pertence_à  EN9 [VIA]; ← True;

TR1 [TROÇO] • Velocidade_máx_permitida ← 80;

TR2 [TROÇO] • Velocidade_máx_permitida ← 60;

44-PT-33 [VIATURA] • Velocidade_real ← 80;

44-PT-33 [VIATURA] • Marca ← "BMW";
```

Quadro 10 – Criação da estrutura HBDS da Fig. 7

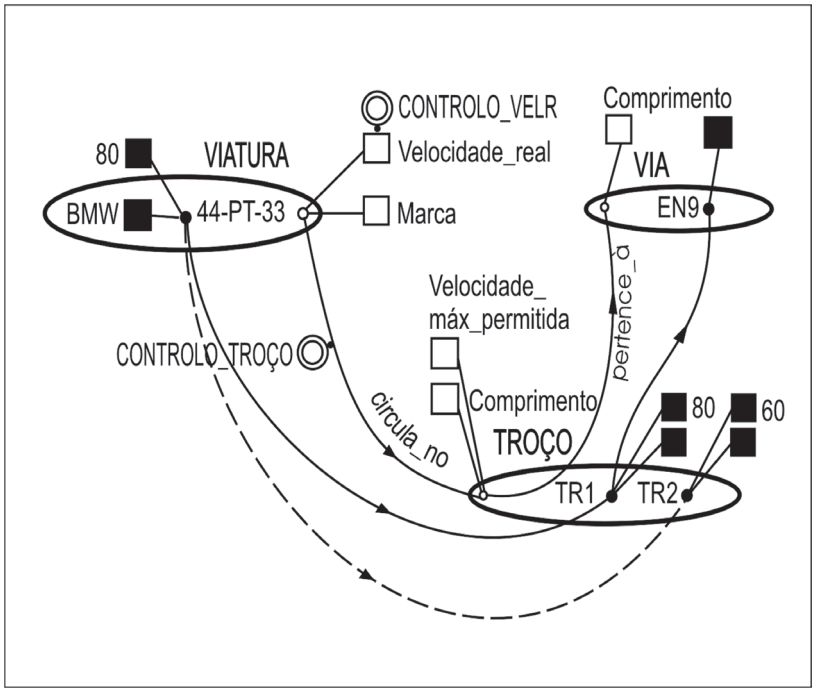


Fig. 7. – Operadores de controlo da Velocidade Real de uma Viatura em circulação com mudança de Troço

Neste contexto, criamos um *O*·TAD que permite controlar a velocidade real de viaturas em circulação em troços de vias, em relação à velocidade máxima permitida em cada um dos troços causa.

Para considerarmos todas as viaturas que circulem em todos os troços, devemos optar por um *O*·TAD do tipo conjunto, *O*·Aa, associado ao TAD Atributo dinâmico da Classe Viatura – **Velocidade real**. Apresentamos este Operador CONTROLO_VELR, a sua definição e o seu algoritmo, no Quadro 11.

Definição do Operador :

CONTROLO_VELR ≈
(*O*•Aa, **VIATURA • Velocidade_real**, depois, Modificar, CONTROLO_VELR( ))

Algoritmo do CONTROLO_VELR:

CONTROLO_VELR (↓X): [| **PROCURA (X [VIATURA] , Y[TROÇO]); X [VIATURA]**
∀X ∈ VIATURA **• Velocidade_real > Y[TROÇO] • Velocidade_máx_permitida?**
Avisar Condutor( ) | ¿ |]

Quadro 11 – Definição e Algoritmo do Operador CONTROLO_VELR

O Operador CONTROLO_VELR executa-se automaticamente quando a operação Modificar se efectivar sobre o atributo de Objecto da Classe VIATURA – **Velocidade real** – ou seja, quando for alterada a sua Velocidade real.

Analisemos a mudança de Velocidade real da Viatura 44-PT-33. Se esta se mantiver em velocidade constante de 80 km/h, como não há alteração, o O·Aa não se executa. Caso a Velocidade real se altere, o O·Aa executa-se.

Consideremos que o registo da alteração da Velocidade real se dá de 5 em 5 km/h. Quando a Viatura aumentar a sua Velocidade real de 80 para 85 km/h, o O·Aa CONTROLO_VELR dispara, executando o seu algoritmo.

Este algoritmo executa-se depois da operação de Modificar a Velocidade real para 85, para X = 44-PT-33 e utiliza um functor[2] denominado PROCURA, que tem como parâmetro de entrada o Objecto 44-PT-33[VIATURA] e como parâmetro de saída o resultado de uma busca feita por este functor, segundo a Ligação VIATURA circula no TROÇO, que resulta no Objecto Y = TR1 do TROÇO.

Em seguida, verifica se a Velocidade real (85 km/h) da Viatura 44-PT-33 ficou superior à Velocidade máxima permitida (80 km/h) no Troço TR1 e resultando verdadeiro, executa Avisar Condutor( ).

No caso desta Velocidade real descer para 75 Km/h ou inferior, o Operador CONTROLO_VELR dispara, executando o seu algoritmo, mas como a Velocidade real está dentro do limite permitido, não é executada acção alguma.

O módulo Avisar Condutor( ), pode ter a implementação que se desejar, seja um simples aviso sonoro, uma mensagem verbal, ou mesmo conceber-se um sistema mais complexo de avisos sucessivos e consequentes penalizações.

5.2. Controlo da Velocidade real de uma Viatura ao mudar de Troço

Uma vez resolvida a questão do controlo da velocidade de viaturas a circular num troço, vamos agora procurar a solução para uma viatura que circula num troço à velocidade máxima permitida e, ao sair desse troço, entre noutro com uma velocidade permitida inferior à do troço anterior.

Suponhamos que a viatura circula à velocidade real de 80 km/h, sai do troço TR1 com velocidade máxima permitida de 80 km/h e entra no troço TR2 com velocidade máxima permitida de 60 km/h (ligação a tracejado na Fig. 7).

Se a viatura reduzir a velocidade, o Operador considerado origina avisos ao condutor até à velocidade estar dentro do limite permitido; se aumentar a sua velocidade, o resultado será o mesmo.

Mas se a viatura mantiver a velocidade real de 80 km/h ao entrar no novo troço TR2, fica a circular acima do limite permitido de 60 km/h. O Operador O·Aa CONTROLO_VELR não dispara, pois não existe alteração do estado do

[2] Um *functor* é um tipo especial de mapeamento entre categorias da Teoria das Categorias. Bouillé utiliza o *functor* para obter resultados de pesquisas em estruturas HBDS.

atributo 44-PT-33 [VIATURA]· Velocidade_real a que está associado.

Para encontrar a boa solução é necessário observar o que mudou de estado, pois é esta alteração que podemos controlar. Foi a mudança de Troço que originou esta situação ilegal que o SDA implementado não solucionou.

A mudança de Troço é realizada na estrutura de dados persistente pela alteração da ligação entre os objectos 44-PT-33 [VIATURA] e TR2[TROÇO], que se modifica para verdadeira e da ligação entre os objectos 44-PT-33 [VIATURA] e TR1[TROÇO] que se modifica para falsa. Consequentemente, na Fig. 7, esta primeira ligação desaparece e a segunda, que está representada a tracejado, fica a traço contínuo.

Neste contexto, a solução é considerar um novo Operador do tipo conjunto associado à Ligação entre as Classes Viatura e Troço – O·Ll – de nome CONTROLO_TROÇO (Quadro 12 e Fig. 7).

Definição do Operador :

CONTROLO_TROÇO ≈
(O•Ll, **VIATURA circula_no TROÇO**, depois, Modificar, CONTROLO_TROÇO( ))

Algoritmo do CONTROLO_ TROÇO:

CONTROLO_TROÇO (X↓,Y↓): [| X [VIATURA] circula_no Y[TROÇO] ? ! | ¿; X [VIATURA]
∀X ∈ VIATURA ∧ Y ∈ TROÇO **• Velocidade_real > Y[TROÇO] •Velocidade_máx_permitida?**
Avisar Condutor( ) | ¿ |]

Quadro 12 – Definição e Algoritmo do Operador CONTROLO_TROÇO

Na mudança de troço da viatura, o Operador CONTROLO_TROÇO é disparado duas vezes.

A primeira, acontece quando a viatura sai do troço TR1, pois a ligação 44-PT-33 [VIATURA] circula_no TR1[TROÇO] é modificada para falso (não existente), dando origem à execução do O·Ll CONTROLO_TROÇO. Por isso, o algoritmo deste Operador começa por verificar se a ligação X [VIATURA] circula_no Y[TROÇO] é falsa e, no caso afirmativo, sai do módulo sem executar qualquer acção (por efeito do símbolo do EXEL – **!** –).

Este Operador dispara, pela segunda vez na mudança de troço, quando a viatura entra no troço TR2, uma vez que a ligação 44-PT-33[VIATURA] circula_ no TR2[TROÇO] é modificada para verdadeiro (existente). Assim, o algoritmo que verifica que esta ligação é verdadeira, não sai do módulo e com o teste já conhecido do Operador CONTROLO_VELR, que controla a velocidade real da viatura para saber se está dentro do limite permitido, avisa o condutor quando tal não acontecer.

5.3. Outros aspectos do Controlo de velocidade

Apresentámos soluções para o controlo da velocidade de viaturas circulando em troços. No desenvolvimento de uma solução mais abrangente, outros aspectos podem ser contemplados, tais como:

- ter em conta as viaturas que têm o seu próprio limite máximo de velocidade;
- desenvolver um sistema de penalizações, quando as infracções persistem, tendo em conta o estado da viatura e da carta do condutor;
- considerar o sistema viário no seu todo, com os seus tipos de vias;
- considerar a Hiperclasse das Viaturas e as outras Hiperclasses ou Classes nela contidas, como os Veículos Ligeiros, os Veículos Pesados e demais categorias.

6. Conclusão

Fizemos a apresentação de um modelo em que se consideraram componentes de conhecimentos representados pelos Operadores em forma de condições e acções associados aos Tipos Abstractos de Dados persistentes, através dos TAD-HBDS.

Para tal, introduziram-se sumariamente os princípios e os conceitos do HBDS (Hipergraph-Base Data Structure) e a linguagem algorítmica EXEL e descreveu-se detalhadamente a definição, sintaxe e algoritmia dos *O*·TAD.

Mostrámos como construir um Sistema de Decisão Autónomo (SDA), com o recurso dos *O*·TAD no seio de uma estrutura de dados persistente, com uma componente conceptual gráfica, que pode existir numa Base de Dados / Base de Conhecimentos.

Por último, construímos alguns aspectos de um SDA com *O*·TAD-HBDS, mostrando como se podem considerar componentes de conhecimento, associando-os a Tipos Abstractos de Dados que descrevem o mundo real ou conceptual de forma persistente, num modelo que tem uma autonomia de comportamento reactivo às alterações do seu estado.

Bibliografia

1. Silveira, Paulo Enes da, *Structuration Dynamique des Connaissances: PLAN/PROJET/ACTION*, Thèse de doctorat de 1'Université Pierre et Marie Curie, Paris 6, 1987, 443 p.
2. Bouillé, François, *Un Modèle Universel de Banque de Données, Simultanément Partageable et Repartie*, Thèse de doctorat d'Etat des Sciences, Université Pierre et Marie Curie, Paris 6, 1977, 570 p.
3. Arsac, Jacques, The EXEL Language, Polycopié, Fondaments de la

Programmation, Université Paris 6, 1974, 62 p.
4. Silveira, Paulo E., EXEL - Uma linguagem algorítmica de elevado nível no domínio Brainware, Universidade dos Açores, Ponta Delgada, 1984, 47 p.
5. Chen, Peter, The Entity Relationship Model – Towards a Unified View of Data, *ACM Transactions on Database Systems*, Vol. 1, N. 1, March 1976, pp. 9-36.
6. Booch, Grady ; Rumbaugh, James; Jacobson, Ivar, *The Unified Modeling Language User Guide*, Addison-Wesley, 1999, 576 p.
7. Liskov, Barbara; Zilles, Stephen, Programming with Abstract Data Types, *Procedings of ACM SIGPLAN Conference on Very High Level Languages,* Santa Monica (Cal.), 1974, pp. 50-59
8. Berge, Claude, Hypergraph generalizing bipartite graphs. *Integer and Non-linear Programming*. Edit. Abadie, North Holland Publish, 1970, pp. 507-509
9. Dahl, Ole-Johan; Nygaard, Kristen, SIMULA: an ALGOL-based simulation language, *Communications of the ACM*, V9, N9, Sept. 1966, pp. 671-678
10. Silveira, Paulo E., Estrutura de Dados Dinâmica. Aplicação à gestão e controlo em tempo real, de uma biofábrica, por computador, *Revista de Ciência e Cultura, série de matemáticas aplicadas*, N.2 Universidade Lusíada , 1997 pp. 107-140.
11. Révélat, Jean, *Transformations Automatiques de Structures de Données*, Thèse de doctorat d'Etat des Sciences, Université Pierre et Marie Curie, Paris 6, 1983, 285 p.
12. Zadeh, Lotfi A., Fuzzy sets, *Information and Control*, Vol. 8, 1965, pp. 338-353
13. Bouillé, François, Fuzzy data processing with the HBDS, *International Conference on Cybernetics and Society*, Tokyo, 3-7 november, IEEE Proceedings. Vol. 2 , 1978, pp. 1222-1227
14. Bouillé, François, A Structure Expert System for Geo-Data handling, *CODATA*, *The role of data in scientific progress*, ed. Phyllis S. Glaeser, 1985, pp. 417-420
15. Bouillé, François, Object-Oriented Methodology in Environment GIS Studies, *Intercarto Internacional Conference on GIS for Environment Studying and Mapping*, Moscouv, May 23-25, 1994, 10 p.
16. Saint-Gérant, Thierry, Comprendre pour mesurer... ou mesurer pour comprendre? HDBS : pour une approche conceptuelle de la modélisation géographique du monde réel, *in* : GUERMOND Y. (Direction), *Modélisations en géographie, Déterminisme et complexités*, Paris : Lavosier, Hermes Science publications, 2005, pp. 261-298.
17. Labarthe, Hugues; Pirot, Françoise, De la modélisation HBDS à l'implémentation sur ArcCatalog-ArcInfo© : un simple prolongement. À propos de la géodatabase obediences, *SIG2008, La conférence francophone ESRI* – Versailles (France), Palais des Congrès 1-2 Octobre 2008, Actes du Colloque, CDROM ou [on-line 22.02.2010] Esri France, available from : http://www.esrifrance.fr/sig2008/pirot_hbds.htm