PARTE II

CONTRIBUTOS

CAPÍTULO 5

ARQUÉTIPOS

(INTERACÇÃO DIRIGIDA
PELA ESTRUTURA)

ÍNDICE

Prefácio da PARTE II.

Na PARTE I desta tese, intitulada FUNDAMENTOS, procurou-se, de uma forma in-tencionalmente abrangente, como deve ser o conhecimento em qualquer área, mas necessariamente sintética e, principalmente até, sistematizadora, como o deve ser no contexto de uma tese de doutoramento, fazer uma análise dos objec-tos de estudo e do conhecimento disponível nas áreas em cuja intersecção esta tese se situa.

Nesta PARTE II, designada CONTRIBUTOS, as principais contribuições teóricas e práticas da tese serão apresentadas em capítulos separados. No capítulo que se segue apresenta-se e formaliza-se o conceito de *arquétipo*, conceito que serve de base ao modelo de interacção que posteriormente se propõe. Dada a proximi-dade representativa dos arquétipos com as noções de edição orientada pela es-trutura, estes favorecem a criação de objectos a partir da descrição, neste caso algébrica, do seu processo de manufactura ou construção.

No capítulo 6, o modelo de interacção que se designa por *Modo Assistido Sintáctico-Semântico* (MASS) é então apresentado, considerando-se, em particu-lar, as extensões a introduzir ao modelo sintáctico de interacção dos arquétipos por forma a reflectir as restrições semânticas inferíveis da especificação formal da aplicação. Um formalismo de especificação baseado na noção de arquétipo e designado *Guiões de Interacção*, é igualmente apresentado e especificado, tendo--se procurado mostrar o seu poder expressivo comparando-o com outros forma-lismos, em diferentes tipos de especificações.

No capítulo 7, um método para a construção automática de IU no âmbito do modelo de interacção apresentado é proposto, sendo apresentadas as principais regras para o desenvolvimento sistemático de Sistemas Interactivos a partir da especificação formal da camada computacional, método que se descreve, e cuja aplicabilidade se ilustra através de exemplos não triviais.

No capítulo 8, apresenta-se o sistema GAMA-X, um sistema que, tendo uma arquitectura e funcionamento baseados nos conceitos e métodos apresentados nos capítulos anteriores, serve de prova à sua viabilidade tecnológica, pois permite a geração semi-automática de controladores de diálogo e de IU para as aplicações interactivas, com características que se enquadram nos princípios da abordagem global apresentada na tese.

No capítulo 9 apresenta-se o sistema GAIA, gerador automático de interfaces adaptativas, que resulta de um estudo sobre a possibilidade de inclusão de *adaptatividade* nas IU desenvolvidas segundo o método proposto, sendo até, como se conclui, o sistema generalizável, podendo constituir uma ferramenta autónoma para fins analíticos.

5.1.- Introdução.

Neste capítulo, um dos conceitos centrais ao trabalho apresentado nesta tese, o conceito de *arquétipo*, é introduzido e formalizado. A noção de *arquétipo* está in-timamente ligada à maioria dos princípios funcionais que se irão propor quanto à forma como deve ser realizada a interacção com o utilizador (em particular com o modelo de interacção proposto) assente nos princípios de *interacção orien-tada pela estrutura* e de *interacção baseada em assistência sintáctico-semântica*, modelo designado por *Modo Assistido Sintáctico-Semântico* (MASS).

A noção de *arquétipo* surgiu primeiramente na área da computação gráfica interactiva [Martins e Oliveira 85], em particular na área dos editores gráficos, tendo sido, dadas as potencialidades reveladas, posteriormente generalizada à área do desenvolvimento de IU para aplicações não gráficas. Torna-se portanto importante mostrar em primeiro lugar as características e potencialidades repre-sentativas dos arquétipos, e, de seguida, indicar as adaptações que tiveram que ser realizadas, algumas delas simples restrições até, para que pudessem ser aplicáveis à área do desenvolvimento de IU para aplicações fundamentalmente, mas não necessariamente, não gráficas. São estas características dos *arquéti-pos*, tal como surgidas inicialmente, que se apresentam e formalizam nas várias secções deste capítulo, discutindo-se posteriormente as adaptações referidas.

5.2.- Arquétipos: Origem.

O conceito de *arquétipo* foi apresentado pela primeira vez em 1985 [Martins e Oliveira 85] no âmbito de um projecto de desenvolvimento de um sistema gráfico para a criação e manipulação de padrões têxteis, e que foi, desde o início, enca-rado como um "case-study" de interesse relativamente à aplicação de métodos formais de especificação na área dos sistemas interactivos.

O projecto encontrava-se aliás em sintonia com algumas das preocupações que, à data, começavam a surgir relativamente à necessidade de tratar a com-plexidade dos sistemas gráficos, e afins, de um modo mais rigoroso, quer recor-rendo a outros paradigmas que não o imperativo, quer mesmo pela utilização de métodos formais de especificação. Quanto à necessidade de se tentarem novos paradigmas, são de salientar os trabalhos pioneiros de Henderson [Henderson 82] e Arya [Arya 84] no contexto do paradigma funcional, e de Pereira [Pereira 85] no âmbito do paradigma lógico. Quanto à necessidade de se aplicarem méto-dos formais de especificação com o objectivo de se introduzir rigor definicional e consequente domínio de

complexidade, são de salientar, entre outros, os traba-lhos de Duce na formalização do GKS [Duce et al. 84] usando VDM, e de Goguen [Goguen 82], Gnatz [Gnatz 83] e Mallgren [Mallgren 83] na formalização de sistemas e linguagens gráficas usando métodos algébricos.

Tendo sido identificados no início do projecto os dois principais processos envolvidos numa aplicação deste tipo, ou seja, a *construção* e a *visualização* de formas, foi desde então possível estabelecer uma distinção entre os tipos de ob-jectos do sistema. Os objectos a construir e manipular são *objectos da aplicação* e devem possuir representações a este nível. Os *objectos de visualização* são re-presentações semânticas de carácter gráfico ou pictórico dos primeiros, que re-sultam de uma interpretação daqueles para efeito de visualização.

A necessidade de se encontrar uma representação formal e abstracta para estes objectos geométricos, formas e figuras, conduziu à ideia de se criarem *objectos abstractos padrão*, capazes de capturarem as propriedades essenciais relacionadas com a própria construção do objecto, permitindo-se posteriormente a criação de objectos concretos por um processo de gradual instanciação atribu-tiva. A estes objectos, que capturam a essência e não o detalhe atributivo de ob-jectos geométricos, foi dado o nome de *arquétipo*[1].

O sistema gráfico desenvolvido, designado AGSYS [Martins e Oliveira 86a], reflecte, na sua estrutura funcional, estes diferentes níveis de refinamento dos objectos, conforme se mostra na figura 5.1.

[1] Platão, na sua teoria da reminiscência, considera que o mundo é formado de "coisas" geometricamente imperfeitas, nas quais, no entanto, alguma estrutura geométrica regular é identificável por abstracção. Estas abstracções são por Platão designadas os “arquétipos das coisas”.

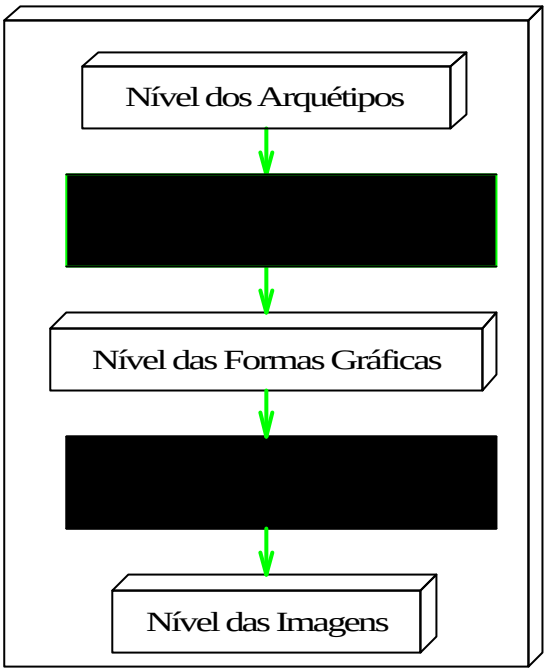

Fig. 5.1 - Arquitectura do AGSYS.

Esta "pipeline" de concretização inicia-se no nível dos *arquétipos* que repre-sentam as propriedades estruturais e topológicas dos diversos objectos que po-dem ser construídos e manipulados pelo sistema gráfico[2]. Num contexto de ori-entação aos objectos, arquétipos poderiam ser considerados *classes*, neste caso representando a estrutura comum de todas as suas instâncias[3], ainda que não o comportamento pois este não existe explicitamente.

A primeira fase do processo de instanciação consiste portanto em reificar *arquétipos* em *formas* pela introdução de valores concretos, por exemplo, para os seus atributos *dimensionais* genéricos. Esta primeira fase corresponde à *fase de construção* anteriormente referida. Tendo já *formas* estruturalmente bem cons-truídas e com atributos geométricos concretos, atributos pictóricos são-lhes adi-cionados por forma a transformá-las em *imagens abstractas*, ou seja, em objec-tos que, ao serem graficamente interpretados, darão origem a *imagens concretas* em ecrã ou noutro qualquer dispositivo de saída. Esta fase corresponde natu-ralmente à fase final de *visualização*.

O modelo posteriormente adoptado para os arquétipos teve em consideração a decisão de que toda a fase de construção fosse suportada por um editor gráfi-co parametrizado, possuindo "conhecimento" sobre o *processo*

---

[2] Fiquemo-nos de momento por esta definição genérica, dado que *arquétipos* são na secção seguinte especifi-cados e exemplos concretos apresentados.

[3] os termos *instância*, *instanciar* e *instanciação* são neologismos informáticos, com significação já estável nesta área, e que resultaram da tradução livre do termo inglês *instance* por *instância*.

de manufactura de cada um dos objectos do sistema, com o objectivo de poder auxiliar o utiliza-dor na correcta construção e instanciação de objectos de dado *tipo* (ou classe)[4]. Este editor corresponde ao verdadeiro mecanismo de interacção do sistema.

Num sistema destas características, a evidente proximidade das noções e dos processos de *edição* e *interacção* teve grande influência na ideia então sur-gida de se poder considerar *interacção baseada em edição estrutural* como uma solução a explorar. A ideia tem semelhanças com o paradigma da *edição interac-tiva dirigida pela sintaxe* ou *pela estrutura,* surgido na área dos ambientes inter-activos de construção de programas, ideias que começavam então também a ser apresentadas em sistemas tais como MENTOR, GANDALF, INTERLISP, EMACS, apre-sentados em [Barstow et al. 86], e SYNTHESIZER GENERATOR [Reps e Teitelbaum 89].

Os mais significativos sistemas de *edição orientada pela estrutura* para a construção de programas ou documentos são exaustivamente apresentados em [Minör 90], sendo as respectivas características analisadas, revelando mesmo que, ainda que sob a mesma designação, sistemas bastante diferentes podem ser encontrados. A maioria destes sistemas utiliza formalismos gramaticais para representar os mais diferentes objectos manipulados ou resultantes dessa manipulação. Em particular nos sistemas mais interessantes e completos de geração de editores estruturados para linguagens, tais como MENTOR, GANDALF ou SYNTHESIZER GENERATOR[5], os objectos a editar são construções dessa lingua-gem, sejam ou não programas, pelo que a sintaxe concreta, descrita usando gra-máticas BNF, é produzida a partir de gramáticas de sintaxe abstracta. Estas úl-timas formalizam a componente estrutural da linguagem, fundamental para a edição estrutural em independência dos detalhes sintácticos. Os objectos a edi-tar de forma interactiva são representados internamente usando árvores de sin-taxe abstracta.

A apresentação em ecrã, para efeitos de edição, de cada uma destas estrutu-ras abstractas é, em geral, igualmente representada por gramáticas que especi-ficam, por categoria sintáctica, a respectiva visualização. O facto de as apresen-tações poderem ser geradas a partir das representações abstractas, permite que diferentes visualizações possam ser, alternativamente, conseguidas. Finalmente, a passagem das estruturas abstractas à sintaxe concreta é, nestes sistemas orientados à estrutura, inverso dos sistemas orientados ao texto. Enquanto que nestes últimos a tradução é feita segundo um processo de "parsing", ou seja, inferindo

[4] na realidade "espécie" ou *sort* no contexto algébrico.

[5] Como se verá posteriormente, o Synthesizer Generator foi, de facto, usado na construção de algumas das ferramentas ligadas à realização desta tese, designadamente para a geração automática de editores estruturados para algumas das linguagens de especificação de controladores de diálogo apresentadas.

estrutura a partir de texto plano, nos primeiros prevalece a técnica de "unparsing", ou seja, o "varrimento" da estrutura que serviu à orientação, sendo produzido um texto na linguagem objecto "plana" do sistema.

No entanto, é importante que se realce que, apesar das semelhanças entre o paradigma proposto no sistema AGSYS, baseado na ideia de *interacção baseada em edição estrutural*, e o proposto nos sistemas de *edição interactiva dirigida pela estrutura*, substanciais diferenças podem ser encontradas quanto aos objectivos e, consequentemente, quanto à metodologia e implementação.

Em primeiro lugar, em sistemas como o AGSYS não se estão a construir pro-gramas ou documentos de estrutura garantidamente correcta, mas sim objectos estruturalmente correctos e com semântica gráfica. Por outro lado, no AGSYS a principal preocupação foi colocada na facilidade de interacção com o utilizador, e, apenas em consequência, na opção pela edição estruturada, enquanto que nos sistemas apresentados a primeira preocupação consiste em apresentar al- ternativas ao trabalho "batch" de "parsing" e de recuperação de erros dos compi-ladores tradicionais. Assim, nestes últimos, a interacção é apenas um meio para atingir um determinado objectivo, mas não o objectivo em si mesmo. No sistema AGSYS o principal objecto de estudo foi, portanto, a modelação e formalização da interacção. A constatação de que a edição estruturada dos objectos a construir pelo sistema poderia ser um mecanismo interessante e vantajoso, e a decisão de construir a interacção com base em tal modelo, surge neste caso como um meio e não como fim. Para além disto, os objectos editáveis neste sistema têm por fi-nalidade gerar outros objectos que são visualizações.

O mesmo tipo de considerações, bem como a verificação do potencial repre-sentativo dos arquétipos e do respectivo modelo de interacção subjacente, levou a que, mais tarde, estes tenham sido estendidos à área dos Sistemas Interacti-vos e à geração de IU.

Na secção seguinte apresenta-se uma formalização do conceito de arquétipo, e do modelo de interacção por edição baseado em arquétipos, usando não gra-máticas abstractas mas um formalismo de base algébrica. O paralelismo com os tão bem conhecidos formalismos gramaticais é, em todo o caso, estabelecido, sendo as vantagens da abordagem algébrica sublinhadas.

5.3.- Arquétipos: Apresentação.

Os arquétipos são, como se disse informalmente, objectos genéricos que preten-dem representar as propriedades estruturais de determinados objectos a cons-truir. Se essa representação estrutural coincidir com, contiver, ou permitir infe-rir, os passos do processo de construção, então os arquétipos podem servir tam-bém de orientadores do processo interactivo de manufactura dos objectos que se pretendem construir.

Considerem-se a título de exemplo os dois seguintes operadores extraídos da assinatura de uma álgebra de especificação de *círculos*:

```
mkPoint: Nat x Nat ? Point          /* coordenadas */
mkCircle: Point x Nat ? Circle      /* centro e raio */
```

O termo "mkCircle(mkPoint(0, 0), 5)" especifica o círculo com centro na ori-gem do plano cartesiano e raio igual a 5. Este termo, dado não ser parametriza-do, corresponde a um e um só possível círculo concreto. Considere-se agora que pretendíamos especificar uma classe de círculos, por exemplo, todos os que têm centro na origem. O termo correspondente deverá agora ser parametrizado rela-tivamente ao segundo argumento correspondente ao raio. Esta classe de círcu-los pode ser descrita pelo termo "mkCircle(mkPoint(0, 0), raio)".

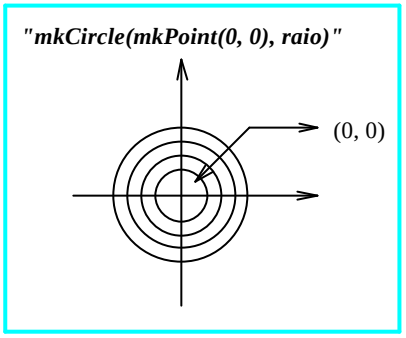

Fig. 5.2 - Arquétipo como Classe.

A figura 5.2 é uma representação pictórica da semântica do termo. Aceitan-do termos genéricos deste tipo, representando classes de objectos, poderão ago-ra considerar-se funções de mais alto nível. Considerando o exemplo em causa, uma função *mkOrigCircles* poderia ser definida da forma seguinte:

```
mkOrigCircles: Nat ? Circle                    /* raio ? círculo */
mkOrigCircles(r) ? mkCircle(mkPoint(0,0), r)
```

representando o termo "mkOrigCircles(5)" o mesmo objecto que o termo anteri-ormente apresentado (cf. "mkCircle(mkPoint(0, 0), 5)").

Levando tal grau de generalidade e abstracção ao extremo, poderíamos mes-mo considerar o termo "mkCircles" como representativo de todos os círculos que podem ser construídos, definindo-o, usando uma notação baseada no l-calcu-lus [Church 41], como:

```
mkCircles ? lp:Nat, r:Nat. mkCircle(p, r)
```

Torna-se agora importante apresentar a sua definição formal, mostrar as su-as propriedades estruturais, as características do processo da sua construção e do processo da sua manipulação, este último baseado em navegação estrutural.

A definição formal de arquétipo é feita seguindo a abordagem algébrica de especificação designada por *abordagem algébrica inicial* [Goguen et al. 77], apresentando-se no APÊNDICE B todas as definições e notação necessárias à compreensão de toda a terminologia subjacente empregue na tese.

Na abordagem algébrica, parte-se de uma assinatura ? = <S, F> (Def. B.2) que contém um conjunto S das espécies das entidades a manipular e um con-junto F dos nomes dos operadores e respectiva funcionalidade, ou seja, espécies argumento e espécie resultado.

Vamos considerar que o universo do discurso é, neste caso, um universo de *peças*, onde, genericamente, podem ser construídas peças como as que, seguin-do uma representação pictórica, se apresentam na figura 5.3[6], fundamental-mente constituídas por uma *base*, cuja forma geométrica pode ser a de um qua-drado, de um rectângulo ou de um triângulo, base na qual se distinguem duas zonas, designadas genericamente por *lado esquerdo* e *lado direito*, zonas onde se poderá inserir uma forma quadrangular, rectangular ou triangular, cada uma destas possuindo uma cor. As formas geométricas inseridas no lado direito das peças conterão um *dígito*, enquanto que as formas geométricas do lado esquerdo conterão uma letra.

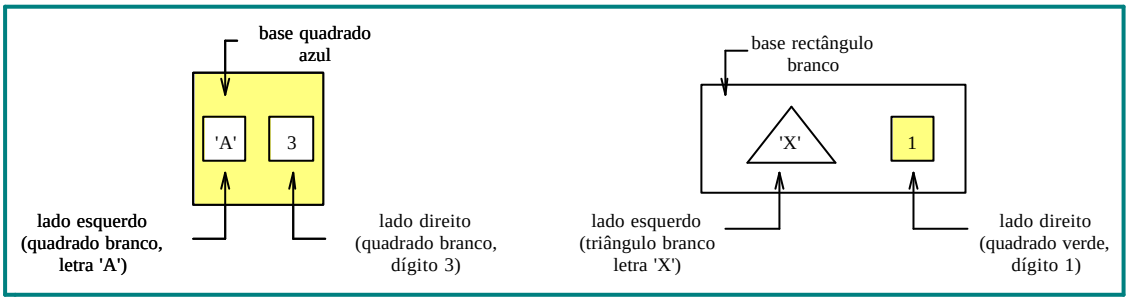

Fig. 5.3 - Exemplos de peças a construir.

Vamos assim considerar para formalizar este exemplo, a assinatura $?_{peça}$ que se apresenta a seguir,

S = { *peça, base, lado_dir, lado_esq, forma, cor, letra, dígito, ponto* }

F = { *faz_peça1 : base x lado_esq x lado_dir ? peça*

[6] O exemplo considerado, dada a sua simplicidade, não favorece a ilustração do poder representativo dos arquétipos. Porém, o objectivo principal é, neste caso, realizar a formalização dos arquétipos usando um exemplo concreto simples em vez de o fazer em abstracto.

faz_peça2 : base x forma x cor ? peça
faz_base : forma x cor ? base
faz_lddir : forma x cor x dígito ? lado_dir
faz_ldesq : forma x cor x letra ? lado_esq
quadrado : ponto x ponto ? forma
rectang : ponto x ponto ? forma
triang : ponto x ponto x ponto ? forma }

que poderia representar o conjunto de operadores e de espécies disponíveis nu- ma aplicação de desenho de tais peças a duas dimensões. Uma assinatura é, portanto, uma abstracção sintáctica. As expressões simbólicas que são geradas pela assinatura designam-se por *termos* ou *palavras*, e o conjunto de termos que se podem gerar a partir de uma assinatura designa-se por ?-linguagem, de-notando-se por $W_?$ (cf. Defs. B.3 e B.4). Se à assinatura se associar um conjun-to de variáveis X, indexado pelas espécies da assinatura, passamos a ter variá-veis de determinadas espécies, pelo que os termos passam a poder conter igual-mente variáveis. O conjunto de todas as expressões simbólicas contendo variá-veis que se podem formar a partir de ? e de X denota-se por $W_?(X)$.

Para qualquer assinatura, uma ?-álgebra particular pode ser facilmente en-contrada. Trata-se da designada *álgebra dos termos* ou *palavras*[7]. Se à assina-tura se associar o conjunto de variáveis X, então passamos a ter uma álgebra a que se dá o nome de *álgebra dos termos com variáveis*, denotando-se por $\underline{W}_?(X)$ (cf. Def. B.6).

Assim, partindo da assinatura apresentada, e considerando algumas variá-veis e constantes, os seguintes termos poderiam ser construídos:

"faz_peça1(X, Y, Z)"
"faz_peça2(faz_base(rectang(P1, P2), C1), quadrado(P3, P4), C2)"
"faz_lddir(F1, C1, 2)"

Todos estes termos são representações genéricas de objectos que podem ser criados, pois obedecem à estrutura sintáctica definida pela assinatura. Estes termos sintacticamente bem formados, genéricos e parametrizados, que repre-sentam em abstracto a estrutura e o processo de construção de objectos, desi-gnam-se por *arquétipos*, podendo ser definidos formalmente conforme a defini-ção 5.1 a seguir apresentada.

Definição 5.1: Arquétipo.

[7] *word algebra.*

> ***Arquétipos*** *são termos (ou palavras) com variáveis, pertencentes à álgebra dos termos (ou palavras) derivada da assinatura ? e que se denota por* $\underline{W}_{?}(X)$.

A álgebra dos termos com variáveis contém portanto todos os termos, com e sem variáveis, que podem ser construídos. Em [Martins e Oliveira 86b] álgebras que especificam processos de construção são, apelativamente, designadas por *álgebras de manufactura.*

Arquétipos são diagramaticamente representados por árvores abstractas em que nodos não-folha representam *acções* (i.é., *operações de construção*) e nodos folha *acções atómicas* ou *variáveis* de uma determinada espécie compatível, tal como se exemplifica na figura 5.4.

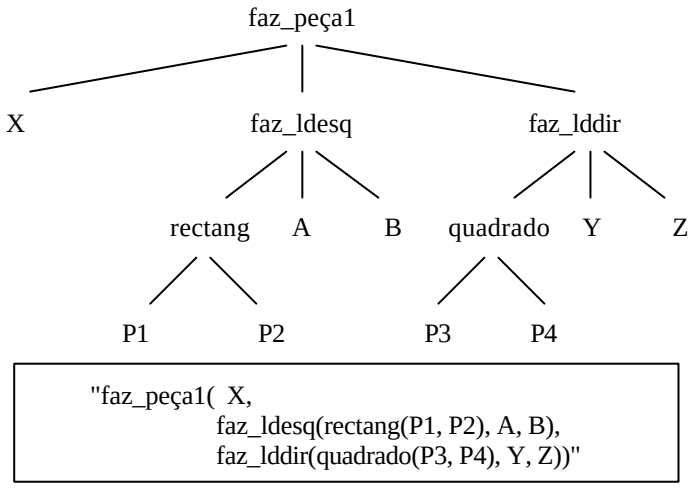

Fig. 5.4 - Arquétipo como Árvore (1).

Repare-se agora, perante o exemplo da figura 5.4, como os arquétipos repre-sentam objectos genéricos que, pela instanciação das suas variáveis, se conver-tem em objectos mais concretos.

No exemplo, a variável X poderá ser instanciada com qualquer termo (ou subárvore) da espécie *base*, que possa ser construído a partir da álgebra de ma-nufactura. A figura 5.5 exemplifica uma das possíveis substituições, que, ainda que não muito detalhada, garante a compatibilidade de espécies entre variáveis e respectivas instanciações. No exemplo, espera-se um termo da espécie *base* e o mesmo é correctamente construído, ainda que contendo variáveis.

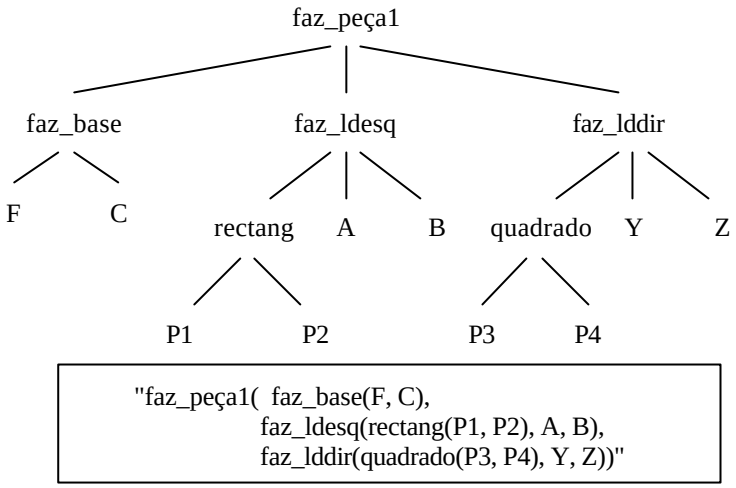

Fig. 5.5 - Arquétipo como Árvore (2).

Note-se desde já, e em função dos exemplos apresentados, a possibilidade de se estabelecer uma relação de ordem sobre arquétipos, denotável por Ù, e re-lativa ao seu grau de especialização, que nos permite afirmar que arq1 Ù arq2, sempre que arq2 se obtém de arq1 por substituição de variáveis por subtermos[8].

5.3.1.- Gramáticas versus Álgebras.

Na secção anterior arquétipos foram formalizados numa base algébrica, e de seguida apresentados quer sob a forma de termos (“strings”) de uma álgebra quer sob a forma de árvores. Torna-se importante mostrar que entre esta pers-pectiva algébrica e a abordagem gramatical existem relações que podem ser es-tabelecidas, e que, ainda que o inverso seja igualmente verdadeiro, a aborda-gem algébrica oferece flexibilidade adicional.

Consideremos pois, como exemplo concreto, a gramática de uma linguagem para a construção e desenho de formas 2D.

Gramática Abstracta

```
peça ::= base lado_dir lado_esq | base forma cor
base ::= forma cor
lado_dir ::= forma cor dígito
lado_esq ::= forma cor letra
forma ::= triang | rectang | quadrado
triang ::= ponto ponto ponto
.....
```

[8] Trata-se de uma "pré-ordem" sobre W?(X), i.é., uma ordem reflexiva e transitiva.

Uma assinatura algébrica, seja $?_{peça}$, pode de imediato ser derivada desta gramática associando uma espécie algébrica a cada símbolo não terminal e, por cada produção simples ou alternativa, criando um identificador de operador ten-do como espécies de entrada as que se associaram aos não-terminais à direita da produção, e como espécie resultado a espécie associada ao não-terminal à esquerda da produção. Produções com alternativas originam, segundo a mesma regra, tantos operadores quantas as alternativas. O conjunto das espécies seria, no exemplo,

Espécies

peça, base, lado_dir, lado_esq, forma, cor, letra,
dígito, triang, rectang, quadrado

e o conjunto de operadores,

Operadores

faz_peça1 : base x lado_esq x lado_dir ? peça
faz_peça2 : base x forma x cor ? peça
faz_base : forma x cor ? base
faz_ldesq : forma x cor x letra ? lado_esq
faz_lddir : forma x cor x dígito ? lado_dir
triang : ponto x ponto x ponto ? forma
rectang : ponto x ponto ? forma
quadrado : ponto x ponto ? forma
br, am, vrd, vm, az : ? cor
...

assinatura que pode ser mesmo apresentada diagramaticamente através de um "diagrama de aranha" ou "diagrama ADJ"[9], onde espécies se representam por elipses ou rectângulos, e linhas orientadas confluem num operador, represen-tado por um ponto com uma etiqueta identificativa do seu nome, indicando as suas espécies argumento, deste partindo uma linha indicando a sua espécie re-sultado.

A figura 5.6 apresenta o "diagrama de aranha" associado a uma das opera-ções para construção de uma *peça* do sistema.

[9] *spyder-diagram* [Jones 80] *ou ADJ-diagram* [Goguen et al. 78].

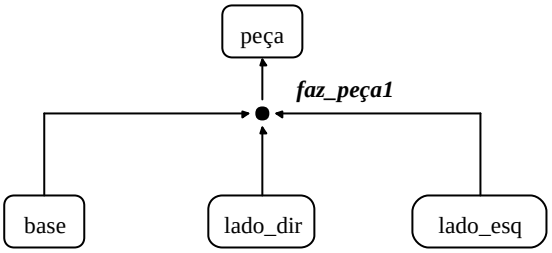

Fig. 5.6 - Diagrama de "aranha": Exemplo.

Vejamos agora, de forma sistemática, como se pode obter uma *assinatura ál-gébrica multi-espécie* (i.é. *heterogénea*) $?_G$, a partir de uma *gramática de contexto livre*[10] (ou *independente de contexto*) *G*, expressa em BNF [Goguen et al. 77].

Seja *G* = (N, T, S, P) uma gramática independente de contexto, em que N re-presenta o conjunto de símbolos *não-terminais*, T o conjunto de símbolos *termi-nais*, com N ? T = ? , S ? N é o seu símbolo *inicial* e P ? N x (N ? T)* é o conjun-to de *produções* ou *regras sintácticas* da gramática.

Para se inferir a assinatura $?_G$ implícita em *G*, usam-se as seguintes regras:

1.- Cada símbolo *não-terminal* da gramática *G* é feito corresponder a uma *espécie* em $?_G$;

2.- A cada *produção* da gramática *G* corresponde um ou mais *operadores* na assinatura, operadores construídos a partir das produções segundo as seguintes regras:

a) Dada uma produção na sua forma genérica,

$$<NT> ::= ? <NT_1> ? <NT_2> \ldots <NT_n> ?$$

onde símbolos não-terminais se representam entre <...> e ????? representam símbolos ou sequências de símbolos terminais, então, o operador ?, arbitrariamente escolhido, correspondente a essa produ-ção terá a funcionalidade,

$$? : <NT_1> x <NT_2> x \ldots x <NT_n> ? <NT>$$

Note-se que para o processo de construção da assinatura algébrica, os símbolos terminais da gramática que aparecem no meio de produ-ções genéricas são ignorados, o que mostra que a assinatura algébri-ca tem o nível representativo de uma gramática abstracta.

[10] *context-free grammar*.

b) Sendo a produção em questão uma alternativa, tal como em,

$$<NT> ::= ld_1 \mid ld_2 \mid ... \mid ld_n$$

então, ela deve ser vista como a síntese de *n* diferentes produções da forma,

$$<NT> ::= ld_1$$
$$...$$
$$<NT> ::= ld_n$$

aplicando-se a cada uma destas as regras que se apresentam.

c) Se a produção em questão contém em qualquer dos seus elementos do lado direito o símbolo de repetição $^+$ (uma ou mais ocorrências), tal como na produção que se apresenta a seguir,

$$<NT> ::= ? \ ?^? \ ?$$

onde ?, ?, e ? são sequências de símbolos, terminais ou não-terminais, e $?^?$ representa uma ou mais ocorrências de ?, então, é neces-sário considerar que $?^?$ é uma abreviatura da produção alternativa,

$$<L> ::= ? \mid ? <L>$$

ou

$$<L> ::= ? \mid <L> ?$$

d) Embora se tenha afirmado que os símbolos terminais existentes no lado direito de uma produção, quando misturados com símbolos não-terminais, são desprezáveis, situações existem em que tal não deve ser. Seja uma produção alternativa entre dois símbolos terminais,

$$<NT> ::= term_1 \mid term_2$$

como por exemplo, na produção,

$$<Cor> ::= branco \mid preto$$

Torna-se neste caso óbvio que dois operadores *0-ádicos* (ou *constantes*) da espécie <Cor> devem ser definidos na assinatura. Admitindo que se mantêm os identificadores, teríamos então os dois operadores seguintes,

branco : ? <Cor>
preto : ? <Cor>

e) Se a produção em questão contém em qualquer dos seus elementos do lado direito o símbolo de repetição * (zero ou mais ocorrências), tal como na produção que se apresenta a seguir,

$$<NT> ::= \gamma\ \gamma^{?}\ \gamma$$

onde γ, γ, e γ são sequências de símbolos, terminais ou não-terminais, e $\gamma^{?}$ representa zero ou mais ocorrências de γ, então, é neces-sário considerar que $\gamma^{?}$ é uma abreviatura da produção alternativa,

$$<L> ::= \gamma \mid \gamma\ <L>$$

ou

$$<L> ::= \gamma \mid <L>\ \gamma$$

O tratamento é semelhante ao tratamento dado à construção repetiti-va apresentada anteriormente, $^{+}$, excepto pelo facto de que, γ, que re-presenta a "string nula", enquanto símbolo terminal especial, deve ser feito corresponder a um operador 0-ádico particular da espécie <L>.

Através deste conjunto de regras fica demonstrada a possibilidade de, par-tindo de uma gramática independente de contexto, se poder, com sistematiza-ção, construir uma assinatura algébrica correspondente.

Assim, e ainda que a abordagem seguida nesta tese seja de base algébrica, com vantagens que se enunciam posteriormente, procurou-se nesta secção mostrar que esta abordagem é compatível com as mais tradicionais abordagens gramaticais. A possibilidade, demonstrada, de se representar sob a forma de es-truturas algébricas, ou termos, estruturas gramaticais, ou produções, é atracti-va, por compatibilizar duas abordagens bastante bem estabelecidas em Ciências da Computação e, de forma mais geral, relacionar a abordagem algébrica com as abordagens gramaticais empregues na área de IHC.

Esta compatibilidade e possibilidade de tradução bi-direccional entre repre-sentações algébricas e gramaticais é uma importante constatação. Consideran-do os resultados apresentados por Green em [Green 86], onde as gramáticas e os diagramas de transição de estados se provam ser sistematicamente convertí-veis no modelo de eventos, por transitividade, tal corresponde a dizer que o mo-delo formal apresentado pode ser igualmente transcrito para o modelo de even-tos, mais próximo da actual tecnologia de implementação.

5.3.2.- Arquétipos: Especificação.

Voltando ao exemplo em curso, qualquer que venha a ser a representação con-creta de peças 2D, o que se pode constatar é que, formalmente, o conjunto de-notado por $W_{?\,peça}(X)$ define todos os termos genéricos, logo com variáveis em X, que podem ser construídos a partir da assinatura $?_{peça}$. Os termos assim for-mados são os arquétipos que representam todos os objectos que se podem cons-truir a partir da linguagem definida na assinatura $?_{peça}$. Estes conjuntos, quan-do indexados, ou "tipados", pela espécie que é o resultado dos termos, constitu-em os conjuntos portadores da álgebra dos termos de $?_{peça}$, álgebra que se de-nota por $\underline{W}_{?\,peça}(X)$.

Um arquétipo é especificável, de acordo aliás com a definição que atrás foi apresentada, como sendo ou uma variável ou um par constituído por um símbo-lo de operação válido e por uma sequência de arquétipos. Recorrendo à notação CAMILA para especificação formal (ver detalhes no APÊNDICE A) definiremos como se segue a classe W?x de todos os arquétipos geráveis a partir de uma assina-tura pré-definida,

```
W? x = X | ? t
? t :: O: Op A: W? x-list ;      recursividade => árvore
```

assinatura essa especificável por,

```
? ig = Fun -> Op-set             ;    assinatura como função finita
Op = SYM;
Fun :: I: Sort-list O: Sort      ;    espécies de entrada e de saída
Sort = SYM;
```

Naturalmente, será necessário fazer depender de ? ig o domínio W? x por for-ma a garantir que os termos (arquétipos) estão correctamente construídos. Esta dependência será posteriormente registada através da escrita de um invariante.

Designa-se por *protótipo* um arquétipo completamente instanciado, i.é., um termo da álgebra dos termos sem variáveis $\underline{W}_{?\,peça}(?\,)$. Estes termos, que apenas contêm operadores, designam-se, no contexto algébrico, por *termos de base*[11]. *Protótipos* são pois especificados como sendo termos de base, conforme o mode-lo,

```
W? :: O: Op A: W? -list          ;  termos de base
```

ou seja, como estruturas recursivas contendo apenas operadores da álgebra.

Note-se igualmente que apenas os termos com variáveis representam arqué-tipos, pois apenas estes são genéricos, dado que, por um mecanismo

[11] *ground terms*.

de substi-tuição algébrica, uma variável de uma dada espécie pode ser substituída por um qualquer termo do respectivo conjunto portador dessa espécie da álgebra dos termos. Os resultados das substituições podem, naturalmente, conduzir a objec-tos concretos e finais completamente diferentes, comprovando assim a generali-dade pretendida.

Por exemplo, o arquétipo que se apresenta na figura 5.7,

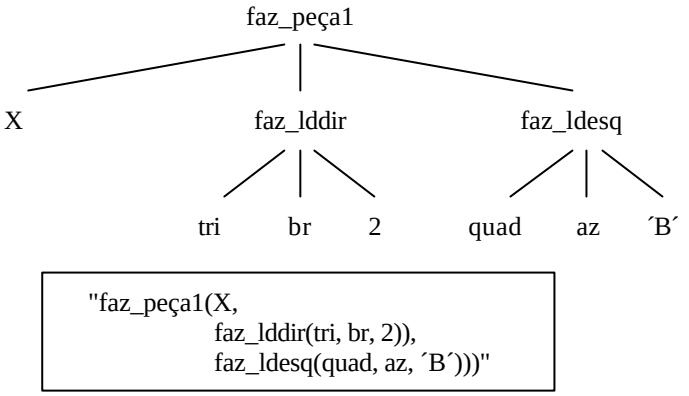

Fig. 5.7 - Instanciação de Arquétipo

no qual *tri* e *quad* são duas variáveis da espécie *forma*, pode representar em abs-tracto, ou seja apenas do ponto de vista estrutural e portanto independentemen-te de pormenores dimensionais relativos ou absolutos, uma qualquer das for-mas apresentadas na figura 5.8.

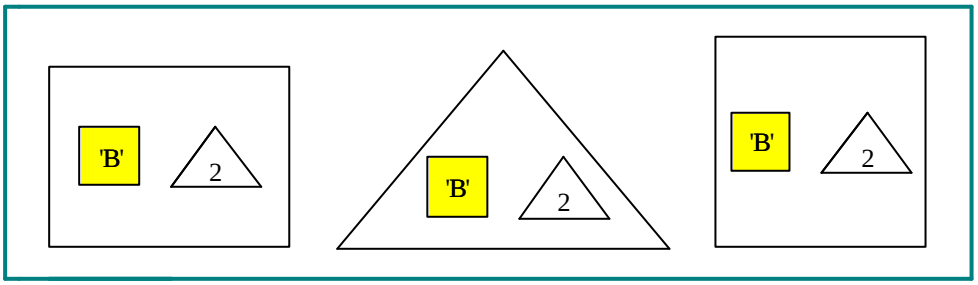

Fig. 5.8 - Formas resultantes

dependendo estes resultados da instanciação particular que fôr realizada para o objecto X. Estas diferentes formas estruturais correspondem a diferentes asso-ciações[12] de termos, todos eles da espécie *base*, à variável X, e sua posterior ins- tanciação. No exemplo, seriam aceitáveis associações tais como,

[12] *bindings.*

X = faz_base(rectang(p1, p2), c)

X = faz_base(triang(p1, p2, p3), c)

X = faz_base(quadrado(p1, p2), c)

Em [Martins e Oliveira 86b] esta generalidade e parametrização dos arquéti-pos é completamente formalizada e utilizada de forma a que arquétipos possam ser considerados *operadores* (i.é. *funções*), que, quando aplicados a um conjunto de argumentos válidos, possibilitam a criação de objectos que são gradualmente mais concretos, até se atingir o nível dos *protótipos*, ou seja, os objectos comple-tamente instanciados. Esta possibilidade, é formalmente assente no conceito de *operadores derivados*[13] que, tal como apresentado em [Goguen et al. 77], é intui-tivo, e consiste simplesmente em considerar-se que um qualquer termo da álge-bra dos termos com variáveis $\underline{W}_{?}(X)$, ainda que possa não conter valores, tal co-mo um polinómio, pode ser considerado uma *função* que associa valores às va-riáveis.

Neste sentido, e recorrendo a conceitos da área da programação funcional, arquétipos como operadores derivados podem ser considerados como *expres-sões lambda com tipos*. Por exemplo, o arquétipo representado pela árvore da fi-gura 5.9,

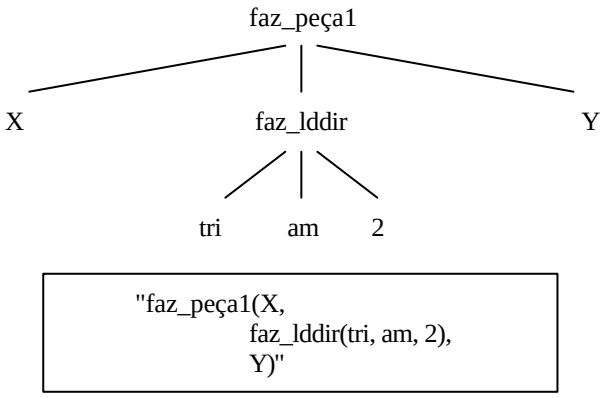

Fig. 5.9 - Arquétipo como Operador Derivado

é um operador derivado da assinatura $?_{peça}$, que poderia ser igualmente repre-sentado pela expressão-lambda com tipos,

?X:base, Y:lado_esq, tri:forma . faz_peça1(X, faz_lddir(tri, am, 2), Y)

Estes diferentes tipos de substituição são, desde logo, importantes graus de generalidade concedidos pela abordagem, que permite ainda que a instanciação de um arquétipo possa ser feita por refinamentos sucessivos até

[13] *derived operators.*

se atingir um objecto não mais refinável, ou seja, sem variáveis, e portanto um *protótipo*.

O segundo grau de liberdade, igualmente importante para uma diferencia-ção e flexibilização da abordagem relativamente a outras, inclusivé a gramatical, é que a ordem de substituição das variáveis pode, em muitos casos, ser de facto arbitrária[14]. No entanto, esta flexibilidade relativamente à ordem de substituição das variáveis traz implicações ao nível do percurso das árvores de representação dos arquétipos. Assim, podendo ser tal ordem arbitrária, devem ser fornecidos mecanismos ao utilizador para se movimentar livremente nas mesmas, ou seja, mecanismos de *navegação estrutural*[15].

Os arquétipos são, portanto, conceptualmente representados como *árvores de termos*, que são o equivalente algébrico das árvores de sintaxe abstracta. Es-tabelecido o relativo paralelismo entre esta abordagem algébrica e as aborda-gens gramaticais, e salientadas as diferenças em termos de flexibilidade, torna-se agora importante a apresentação de uma formalização destas características do modelo dos arquétipos, até agora enunciadas mas ainda não formalmente apresentadas.

5.3.3.- Um Sistema de Arquétipos.

A manipulação dos arquétipos, enquanto representantes de um processo genéri-co de construção sintacticamente (ou estruturalmente) correcto de objectos de determinados tipos e estrutura, deverá, por coerência, ser uniforme. Para que tal seja possível, então a manipulação deve ser tornada independente da álgebra de "manufactura" subjacente e que os arquétipos representam sob a forma de termos. Em resultado, será possível manipular termos da mesma maneira, quer estes tenham semântica gráfica ou outra.

A especificação das operações de construção, navegação e instanciação de arquétipos deverá portanto ser realizada tendo por contexto um sistema genéri-co de arquétipos. As experiências concretas realizadas no âmbito da especifica-ção e desenvolvimento de dois sistemas baseados em arquétipos, em particular um pequeno sistema gráfico simbólico designado por AGSYS [Martins e Oliveira 86a] e um editor gráfico parametrizado [Martins e Oliveira 87], permitiram en-contrar soluções padronizadas e genéricas não só para a modelação dos arqué-tipos mas também para a sua manipulação.

---

[14] Estas duas propriedades, intrínsecas aos arquétipos, tornam-se bastante distintivas quando, como se verá posteriormente, os mesmos servirem de base à representação dos mecanismos de interacção com aplicações, e não tanto, como aqui, onde são representações do processo de construção de determinados objectos. No entanto, e desde já, o paralelismo com a visão "construcional" poderá ficar registado.

[15] *browsing.*

Abstraindo do tipo de objectos que os arquétipos possam representar, pode-mos assim considerar que um qualquer sistema de arquétipos tem uma estrutu-ra interna especificada por,

```
Arqsys :: A: Adb      ;    Base de Dados do Sistema
          N: Nucl     ;    Ambiente de trabalho não persistente
          B: Buff     ;    Dados correntemente em manipulação
```

ou seja, o sistema é constituído por três componentes fundamentais, designada-mente, uma base de dados que associa um identificador particular a cada ambi-ente ou contexto de trabalho contendo arquétipos, protótipos e operadores,

```
Adb =  Envid -> Environ ;    Cada contexto de trabalho é identificado
```

sendo cada ambiente de trabalho representado por,

```
Environ :: S: ? ig    ;    assinatura
           M: ??      ;    função significado dos termos
           D: ?A      ;    operadores derivados
           K: Knlg    ;    directoria de termos
```

ou seja, cada contexto de trabalho é composto pela assinatura da álgebra de manufactura em questão, pela função de interpretação dos termos, pelos opera-dores derivados construídos e pela directoria de termos. A assinatura, já ante-riormente apresentada, é modelada por,

```
? ig = Fun -> Op-set          ;    assinatura como função finita.
Op = SYM;
Fun :: I: Sort-list O: Sort   ;    espécies de entrada e de saída
Sort = SYM;
```

devendo obedecer ao invariante que especifica as propriedades a satisfazer para que a mesma possa ser bem formada.

```
FUNC inv-? ig(sig: ? ig) : Bool
; numa assinatura deve existir pelo menos um operador por cada
; funcionalidade.
; o conjunto de chegada de todas as operações coincide com o conjunto das
; espécies, ou seja, é possível criar objectos de qualquer espécie.
RETURNS ~ exist(f <- dom(sig) : sig(f) == {}) ? (rs(sig) == allSorts(sig));
```

onde as funções *rs(sig)* e *allSorts(sig)*[16], determinam, respectivamente, o conjun-to das espécies resultado de todos os operadores e todas as espécies

[16] especificadas no APÊNDICE C.

da assina-tura. O invariante especifica que, para que uma assinatura seja correcta, para cada funcionalidade existe pelo menos um operador. Por outro lado, o conjunto das espécies resultado dos operadores deve coincidir com o conjunto das espé-cies da assinatura, garantindo-se assim que é possível criar objectos de todas as espécies.

Por sua vez o núcleo conterá o identificador do actual ambiente aí carregado e o próprio ambiente.

```
Nucl :: I: EnvId        ;  Identificador do Ambiente
        E: Environ      ;  Ambiente de trabalho
```

Cada ambiente de trabalho guarda igualmente os vários operadores deriva-dos entretanto criados pelo utilizador, cada um dos quais possuindo identifica-ção própria.

```
?A = Anm -> ?Arch   ;  função finita dos operadores derivados
Anm = SYM           ;  identificador
```

Esta função finita que associa identificadores a operadores derivados repre-senta uma directoria de arquétipos que está sujeita a regras de construção, i.é. invariantes, que se apresentam no APÊNDICE C.

Estes operadores derivados vão ser especificados como expressões-lambda, indicando a sequência de variáveis a instanciar e a árvore que representa a re-gra de construção.

```
?Arch :: L: X-list      ;  variáveis
         R: d? t        ;  termos sem variável na raiz
```

sendo d? t,

```
d? t ::  O: (Op | Anm)  ;  operador ou nome de arquétipo
         A: dW? x-list  ;  termos, com variáveis, compatíveis em espécie
dW? x = X | d? t
```

Torna-se evidente da especificação, que um operador derivado pode ter co-mo raiz um operador *Op* ou o identificador *Anm* de um outro operador derivado seu constituinte, mas nunca uma variável. Esta restrição tem a ver não só com alguma coerência e significado na especificação destes operadores como ainda com o facto de que a função de inferência das espécies de um termo (cf. a fun-ção *yctxt* apresentada no APÊNDICE C) não reconhece arquétipos formados por uma simples variável.

Permite-se igualmente que uma directoria de termos guarde, no contexto de trabalho, instanciações particulares de um arquétipo, associando-se a cada va-riável deste um termo de base.

```
Knlg ::  D: W? dir
         I: Anm -> Inst      ;  instanciação de um arquétipo
Inst =   X -> W?             ;  variável associada a um termo base
W? ::    O: Op               ;  termo base
         A: W? -list
```

Por outro lado, e para efeitos de edição e manipulação, os termos a editar são carregados para um *buffer* e convertidos para um formato que facilita a es-pecificação das operações de navegação sobre os mesmos, modelado por

```
Buff ::  N: [X]        ;  nome do objecto ou variável raiz
         C: Path       ;  cursor = lista de naturais
         T: [dW? x1]   ;  termo em formato buffer
```

```
Path =      Nat-list            ;  representação do caminho
dW? x1 =    X | d? t1
d? t1 ::    O: (Op | Anm)  ;  cada subárvore é representada como uma
            F: Nat -> dW? x1 ;  correspondência de Nat para dW? x1
```

Este formato corresponde a associarmos uma ordem a cada subárvore exis-tente a dado nível, e a podermos assim manter um cursor que representa o ca-minho, sob a forma de uma sequência de naturais, para a subárvore em edição. Este modelo baseia-se num processo de conversão de uma árvore de termos numa função finita de naturais para termos.

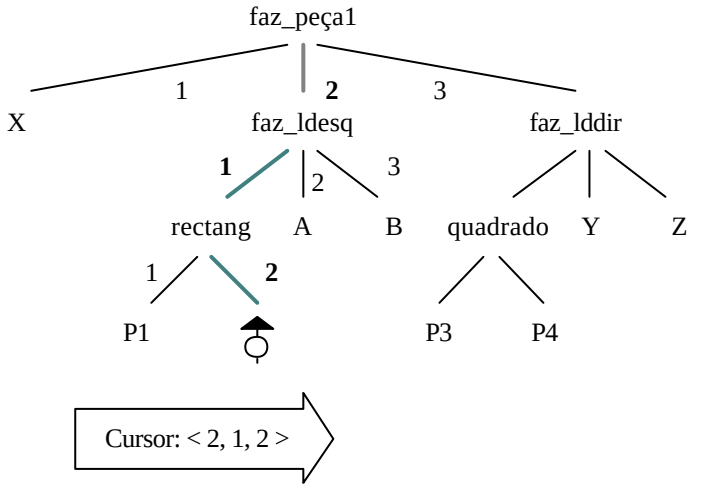

Fig. 5.10 - Esquema de Navegação.

A figura 5.10 ilustra a relação que permanentemente existe entre a localiza-ção da subárvore que se encontra em edição, e o valor do *cursor* que representa o caminho a percorrer na árvore até se atingir esse subtermo. O processo de na-vegação na árvore, que se apresenta numa das secções seguintes, vai igualmen-te basear-se neste modelo e na utilização do *cursor.*

5.3.4.- Arquétipos: Construção.

A construção de arquétipos, enquanto representantes genéricos do processo de criação de objectos, começa pela identificação interactiva da espécie do arqué-tipo pretendido. As espécies da assinatura ? de trabalho constituem, por sua vez, as possíveis espécies resultado para qualquer termo ou arquétipo a ser cria-do. Quando o utilizador selecciona, dentre os elementos deste conjunto de espé-cies, a espécie resultado, determina que apenas termos resultando na espécie indicada podem ser criados, limitando deste modo, automaticamente, o conjun-to de operadores que podem ser seleccionados para tal efeito. De facto, apenas operadores cuja espécie resultado seja coincidente com a espécie seleccionada podem e devem ser escolhidos.

As diferentes hipóteses de construção de um termo da espécie pretendida são em cada momento avaliadas segundo um processo de navegação estrutural sobre a assinatura ?, sendo os vários operadores, normais ou derivados, que re-sultam em termos da espécie pretendida apresentados, em cada momento, ao utilizador. Para cada operador seleccionado, é então necessário realizar um pro-cesso de selecção semelhante para cada uma das espécies dos seus respectivos argumentos. O processo é portanto intrinsecamente recursivo.

A operação que permite construir um objecto completamente instanciado, ou seja um termo base, é especificada pela função *buildobj* que corporiza todo o processo de interacção e edição orientada pela estrutura. A função recebe como parâmetros o estado actual do sistema e o identificador do objecto a construir, e cria um objecto da espécie a ser lida, que é guardado na directoria de objectos do sistema, identificado pelo valor da variável dada como parâmetro.

```
FUNC buildobj(sys: Arqsys, x: X) : Arqsys
; constrói um objecto
RETURNS
        let     (  env = E(N(sys)), sig = S(env), term = sdedit(sig),
                   newbuf = Buff(x, <>, ldbuf(term)) )
        in
           Arqsys(G(sys), N(sys), newbuf);
```

A função *sdedit,* invocada de *buildobj,* é a função principal do processo de construção estruturada e interactiva de termos. A função trabalha sobre uma

qualquer assinatura que lhe seja passada como parâmetro, sendo portanto ge-nérica porque estrutural, como se afirmou atrás, dando como resultado um ter-mo completamente instanciado, i.é., um termo base ou *protótipo*.

```
FUNC sdedit(sig: ? ig) : W?        ; edição dirigida pela estrutura
; constrói um termo base da assinatura
RETURNS
        let    ( sort = inpFrom(rs(sig)) ) ; lê a espécie do termo a construir
        in
           ibuildgrnt(sort, sig) ; constrói o termo base interactivamente
```

Esta função *sdedit* usa a função *inpFrom*, que é uma função genérica de in-teracção que permite que o utilizador realize uma escolha válida sobre um con-junto de identificadores, e que pode ser especificada, com base na função pré-definida da linguagem CAMILA *ppickone*, como sendo,

```
FUNC inpFrom(s: Y-set) : Y
; lê um valor obrigatoriamente no conjunto s
RETURNS
        ppickone(s, "seleccione 1 de :");
```

Especifica-se agora a função principal, *ibuildgrnt*, que constrói de forma in-teractiva um termo base a partir da respectiva espécie resultado e da assinatu-ra algébrica.

```
FUNC ibuildgrnt(s: Sort, sig: ? ig) : W?
; interactivamente constrói um termo base da assinatura
RETURNS
        let ( op = inpFrom(opsR(s, sig)) )
        in
           W? (op, < ibuildgrnt(esp, sig) | esp <- argSorts(op, sig) >);
```

Conforme a própria definição textual desta função deixava transparecer, a função é multi-recursiva construindo W?-termos pela sucessiva aceitação de um operador pertencente ao conjunto de operadores cujo resultado coincide com a espécie esperada, e, de seguida, realizando tantas invocações recursivas da função quantos os argumentos do operador anteriormente seleccionado. Cada uma destas invocações sintetizará um subtermo da espécie correspondente ao argumento esperado.

A função que permite construir interactivamente arquétipos é bastante se-melhante à anterior, devendo porém considerar-se, neste caso, um parâmetro adicional que é a família S-indexada de variáveis,

```
Var = Sort -> IdVar-set  ; função finita de espécie para nomes de variáveis
```

A função é então especificada por

```
FUNC ibuildarq(s: Sort, sig: ? ig, vars: Var) : W? x
; interactivamente constrói um arquétipo
RETURNS
        let ( tipo = inpFrom({Var, Termo}) )
        in
        if tipo == Var
        then inpFrom(vars[s])
        else
            let ( op = inpFrom(opsR(s, sig)) )
            in
                W? x(op, < ibuildarq(esp, sig) | esp <- argSorts(op, sig) >);
```

Todas estas funções devem construir objectos bem formados, ou seja, satis-fazendo os respectivos invariantes. Estes são, dada a recursividade das estrutu-ras representativas dos diversos tipos de termos, bastante complexos[17].

Ainda que relativamente incompleto, pois não são apresentadas todas as funções auxiliares usadas na sua definição, apresenta-se de seguida, no entan-to, o essencial do invariante que deve ser obedecido por um arquétipo para que seja bem formado.

```
FUNC inv-Arq(arq: W? x, sig: ? ig, v: Var) : Bool
; operadores de um arquétipo devem pertencer ao conjunto dos operadores
da
; assinatura; as variáveis devem pertencer ao conjunto de variáveis Var.
; as espécies dos argumentos de um dado operador devem coincidir com as
; espécies resultado de cada uma das subárvores do operador.
RETURNS
        (treeOps(arq) ?  allOps(sig) ? treeVars(arq)  ?  allVars(v) ?
        if is-X(arq)
            then true
            else  let (op = O(arq), ls = A(arq))
                  in
                     argSorts(op, sig) == < treeSort(x, sig, v) | x <- ls > ?
                     all(t <- elems(ls) | inv-Arq(t, sig, v) ?
                     inv1-Arq(arq, sig));

FUNC inv1-Arq(arq: W? x, sig: ? ig) : Bool
; o arquétipo é bem formado
RETURNS
```

[17] Não sendo relevantes para a especificação das operações sobre arquétipos, estes invariantes são apresentados no APÊNDICE C.

```
aridadesOk(arq, sig) ? tiposOk(arq, sig) ? varsOk(arq, sig);
```

No sistema desenvolvido, arquétipos, após a sua construção, são associados a identificadores para que posteriormente possam ser interactivamente instan-ciados, ou usados como *operadores derivados* que se podem aplicar a listas de argumentos.

5.3.5.- Arquétipos: Navegação.

A representação de termos algébricos com variáveis sob a forma de árvores, possibilita, de forma relativamente sistemática, a criação de simples e flexíveis mecanismos de *navegação* de indiscutível utilidade, quer durante a criação dos objectos quer para a sua posterior edição.

No sistema AGSYS, por questões de carácter funcional, assumiu-se que a navegação sobre arquétipos para edição deveria ser feita exclusivamente numa área do sistema designada por *buffer*, coincidindo o carregamento do termo a editar em tal área com uma transformação da sua representação, com o objecti-vo de facilitar a sua manipulação, ainda que tal não seja nem imperioso nem limitativo, mas tão só uma decisão estrutural de projecto.

Assim, e antes que se possa navegar sobre um arquétipo, este é armazenado no "buffer", momento em que a sua representação é modificada. A função *ldbuf* realiza esta conversão, transformando um W? x num W? x1, ou seja, um termo com variáveis representado por uma árvore, num termo com variáveis em for- mato interno do "buffer". Conceptualmente, o utilizador mantém desta navega-ção a imagem de que o cursor se desloca numa árvore segundo as quatro movi- mentações em geral possíveis (? ? ? ?).

```
FUNC ldbuf(t: W? x) : W? x1
; converte termo com variáveis para formato buffer
RETURNS
        if is-X(t)
            then t
            else W? x1(O(t), ff_from_list( <ldbuf(st) | st <- A(t)> ));
```

Carregado o termo no "buffer", as funções de navegação estrutural *root*, *up*, *down*, *left* e *right*, permitem a movimentação do cursor sobre o termo em edição nas quatro direcções ou colocam-no directamente na raíz da árvore.

Cada uma destas funções é de seguida especificada, sendo de notar que cada uma delas se limita a modificar o cursor (sequência dos ramos) após testar a sua posição actual para verificar se tal movimento é ou não possível.

```
FUNC root(buf: Buff) : Buff
; posiciona o cursor na raiz do termo => igual a NIL
RETURNS
```

```
Buff(N(buf), <>, T(buf));
```

A função *root* corresponde a subir à raiz da árvore pelo que o cursor passa a assumir o valor de uma sequência vazia.

```
FUNC up(buf: Buff) : Buff
; posiciona o cursor no nodo superior
RETURNS
        let ( cursor = C(buf) )
        in
            if cursor == <>  then buf
                             else Buff(N(buf), blast(cursor), T(buf));
```

A função *up* testa inicialmente se a posição actual é a raiz e, caso não seja, limita-se a remover da sequência de ramos o último (cf. *blast*), o que correspon-de a subir na árvore para o nodo "pai" (ver Fig. 5.10).

A função *down*, apresentada a seguir, testa se a raiz do subtermo onde o cursor está posicionado (cf. *term = tfpath(cursor, T(buf))* é uma variável, em cujo caso não pode ser aplicada. Se não é este o caso, e se o termo se ramifica, então junta à sequência o ramo 1, que corresponde ao trajecto descendente pela sub- árvore mais à esquerda.

```
FUNC down(buf: Buff) : Buff
; posiciona o cursor na raiz da subárvore mais à esquerda do nodo
RETURNS
        let ( cursor = C(buf), term = tfpath(cursor, T(buf)) )
        in
            if is-X(term)
                then buf
                else
                    if F(term) == []
                        then buf
                        else Buff(N(buf), cursor ? <1>, T(buf));


FUNC left(buf: Buff) : Buff
; posiciona o cursor na subárvore à esquerda do nodo
RETURNS
        let ( cursor = C(buf) )
        in
            if (cursor == <>) ? (last(cursor) == 1)
                then buf
                else
                    let ( newlast = last(cursor) - 1 )
                    in
                        Buff(N(buf), blast(cursor) ? <newlast> , T(buf));
```

A função *left* testa inicialmente se o cursor se encontra na raiz da árvore ou numa subárvore mais à esquerda de um qualquer nodo. Se não for este o caso, então actualiza o valor do último ramo da sequência decrementando-o de uma unidade, o que é equivalente ao deslocamento para o ramo imediatamente à es-querda do actual.

A função *right* testa inicialmente se o cursor se encontra na raiz da árvore. Se não for este o caso, então calcula o valor do novo último ramo da sequência, que é o valor do actual incrementado de uma unidade, e se tal subárvore existir actualiza o cursor. A função é perfeitamente simétrica relativamente à função *left* anteriormente especificada, tendo-se neste caso adoptado um critério ligei-ramente diferente para a especificação.

```
FUNC right(buf: Buff) : Buff
; posiciona o cursor na subárvore à direita do nodo
RETURNS
        let ( cursor = C(buf) )
        in
          if (cursor == <>)
            then buf
            else
              let (  newlast = last(cursor) + 1,
                     t = tfpath(blast(cursor), T(buf)) )
              in
                if ~(newlast ?  dom(F(t)))
                  then buf
                  else  Buff(N(buf),  blast(cursor)  ?  <newlast>  ,
T(buf));
```

Mecanismos de navegação mais sofisticados poderiam ser de seguida espe-cificados com base nestas cinco funções usando composição funcional.

A importância de se possuir um mecanismo de navegação sobre estas árvo-res de termos consiste na consequente flexibilidade que é oferecida ao utiliza-dor, que pode, durante o processo de instanciação de um termo, fazê-lo por uma ordem completamente ao seu livre arbítrio. Atendendo ainda ao facto de que a instanciação pode ser também realizada de forma incremental, após substituir uma variável por um termo o utilizador pode de imediato passar a navegar no novo subtermo introduzido na árvore.

No final da sessão de edição, o utilizador pode gravar o termo editado, quer na directoria geral de termos, quer na directoria de arquétipos, o que é realizado pela invocação da função *saveterm* (especificada no APÊNDICE C).

5.3.6.- Arquétipos: Instanciação.

A instanciação completa de um arquétipo é o processo que permite construir termos sem variáveis, *protótipos*, a partir dos vários arquétipos existentes, por associação de valores às variáveis do arquétipo, associações essas que podem já ter sido criadas ou que podem ser introduzidas interactivamente.

Assim, se tivermos os operadores derivados Arq1 e Arq2, que se apresentam a seguir,

Fig. 5.11 - Operadores Derivados.

onde P1, P2, P3 e P4 são variáveis de uma espécie Ponto, a operação de instan-ciação vai permitir que, usando Arq1 ou Arq2, se possam construir protótipos estruturalmente correctos, pois herdam a estrutura do seu gerador. A sessão de instanciação e gravação do termo concreto criado teria, na sua forma mais simples, o aspecto e sequência seguintes, representando as linhas iniciadas por **>** comandos do utilizador, as iniciadas por **-** diálogo gerado pelo sistema e as iniciadas por **#** linhas de apresentação de resultados.

```
> !finst(S, Arq1, quad1)
- valor para P1: ponto( 2, 5 )
- valor para P2 : origem
# PROTO: quad1 = quadrado(ponto(2, 5), origem)
> !saveterm
# SAVED: quad1
```

A operação *!finst*, que realiza esta instanciação ao nível do contexto do sistema, baseia-se na função *inst*, que dado um arquétipo e uma função finita representando as associações variável-valor estabelecidas, cria um termo com-pletamente instanciado.

```
FUNC inst(t: W? x, Inst: X -> W? ) : W?
; dado um mapa de variáveis e seus valores, Inst, esta função instancia
; completamente um arquétipo.
RETURNS
        if is-X(t)
```

```
            then Inst[t]
            else  W?(O(t), <inst(tt, Inst) | tt ?  A(t)>);
```

De notar mais uma vez a simplicidade da especificação dada a natureza re-cursiva da representação. Assim, instanciar um arquétipo consiste em determi-nar se a sua raiz é uma variável, em cujo caso a função finita *Inst* lhe é aplicada dando como resultado o termo base correspondente. Se não, um termo de base, W?, deverá ser construído tendo por raiz o operador raiz do arquétipo, e como subárvores as resultantes da instanciação das subárvores do arquétipo.

A operação *!finst*, não só vai invocar a anterior para a construção do objecto instanciado, como vai realizar a alteração no estado do sistema correspondente à criação deste novo protótipo. A função é especificada por

```
FUNC !finst(sys: Arqsys, arq: Anm, x: X) : Arqsys
RETURNS
        let (  env = E(N(sys)), buf = B(sys), Dop = D(env), sig = S(env),
               tdir = D(K(env)), vars = L(Dop[arq]),
               sorts = argSortsA(arq, sig, Dop),
               consts = < userChoice(opsRnull(s, sig)) | s ?  sorts >,
               assocs = ff_from_lists(vars, consts),
               prot = inst(R(Dop[arq], assocs)),
               newbuf = Buff(x, <>, ldbuf(prot)),
               newtdir = tdir + [x -> prot],
               nidir = I(K(env)) + [arq -> [Dop[arq] + [x -> consts]]],
               newenv = Env(sig, M(env), Dop, Knlg(newtdir, nidir)) )
        in
               Arqsys(G(sys), Nucl(I(N(sys)), newenv), newbuf);
```

ou seja, dado o estado do sistema, um nome de arquétipo e um identificador pa-ra o protótipo a criar, a operação faz a sua construção interactiva. Selecciona-dos os subcomponentes do sistema necessários à especificação, a função come-ça por encontrar a sequência de argumentos do arquétipo, associada a *vars*, a sequência das respectivas espécies, dada por *sorts* e, para cada um dos compo-nentes da lista de argumentos, pede ao utilizador uma constante dessa espécie, construindo a lista *consts*. Estabelecida a correspondência entre variáveis argu-mento e constantes, através da função que cria a função finita de variáveis para valores, *ff_from_lists(vars, consts)*, o arquétipo é de seguida instanciado por invo-cação da função *inst*. O resultado é um protótipo que vai ser colocado no *buffer*, copiado para a directoria de termos genéricos, e esta instanciação particular guardada na directoria de instanciações do arquétipo.

A última função relacionada com o modelo dos arquétipos que se apresenta, é a função que corporiza o mecanismo de inferência de espécies. Assim, dado um termo, seja ele um termo normal, uma variável ou um nome de arquétipo, esta função deduz a respectiva espécie sob a forma de uma

correspondência, ou melhor, família indexada de espécies para variáveis (cf. APÊNDICE C), ou seja, o completo *contexto* (mapa de variáveis) do termo.

```
FUNC mk_context(t: d? t, sig: ? ig, dop: ?A) : dfm(Sort, X)
RETURNS
        let (   ? = O(t),
                ss = if is-Op(?)  then argSorts(?, sig)
                                  else argSortsA(?, sig, dop),
                r = mkRelfpl(ss, A(t)),
                rx = { (s, st) | (s, st) <- r ; is-X(st) },
                ts = ran(r-rx)
            )
        in mkFamfRel(rx) ? f UNIONf({mk_context(tt, sig, dop) | tt <- ts});
```

Assim, se o símbolo raiz do termo é um operador, então as espécies dos res-pectivos argumentos podem ser encontradas através de uma consulta à assina-tura. Caso o símbolo seja um nome de arquétipo, então, usando a sua represen-tação na directoria de arquétipos e a sua espécie resultado, as espécies dos seus argumentos são inferidas. Esta lista de espécies de argumentos é então compa-rada com as correspondentes subárvores das quais apenas variáveis são consi-deradas (cf. a expressão *rx = { (s, st) | (s, st) <- r ; is-X(st) }*). Pares ordenados são então formados correspondendo ao contexto encontrado a esse nível. Todas as subárvores não-variáveis devem de seguida ser tratadas da mesma forma, o que é garantido pela invocação recursiva da função, conforme a seguinte porção da especificação,

```
let     ts = ran(r-rx)
in      mkFamfRel(rx) ? f UNIONf({mk_context(tt, sig, dop) | tt <- ts})
```

Sendo porém, apesar de tudo, os protótipos apenas representações abstrac-tas e não-parametrizadas de objectos com semântica, resta agora encontrar um mecanismo de interpretação que realize a sua tradução para o domínio semân-tico definido. Voltemos a considerar que arquétipos representam objectos com semântica pictórica e vejamos então o processo de tradução de tais objectos pa-ra imagens.

5.3.7.- Arquétipos: Visualização.

Considere-se então, a título de exemplo deste processo de tradução, o problema existente no sistema AGSYS no qual, partindo-se de arquétipos, se devem cons-truir apresentações de tais objectos, por exemplo, em ecrã.

Basicamente, o problema consiste em encontrar-se uma tradução gráfica para cada possível árvore representando o processo de manufactura de um ob-jecto. É portanto um problema de atribuição de semântica pictórica a

estruturas sintácticas necessariamente correctas, dada a assistência sintáctica fornecida, e resultantes do processo de construção de formas que são representadas por ár-vores de termos.

Para que tal seja possível, teremos que ter, antes de mais, bem definido um domínio objecto, neste caso de carácter gráfico e representado coerentemente por termos de uma outra álgebra, para o qual se possa fazer a tradução de cada espécie e de cada operador da álgebra de manufactura.

Não sendo porém tal processo relevante neste contexto, em que arquétipos irão servir de base para um modelo de interacção em sistemas cuja semântica da sua interpretação não será certamente gráfica, bastará deixar aqui registada a solução encontrada no sistema AGSYS. Neste, a função que dá significado aos termos é específica, ou seja, deveremos ter uma função de tradução particular para cada ambiente de visualização. No entanto, qualquer que seja a semântica particular em questão, o processo de tradução é genérico e consiste formalmente num homomorfismo semântico (cf. Def. B.7), segundo o qual se associa a cada termo da álgebra de manufactura, através da função significado ??, um termo da álgebra de visualização. Formalmente, este homomorfismo *h* consiste numa função que traduz termos em W? para uma ?-álgebra A que representa a sua semântica.

É portanto uma função de assinatura

$$h : \underline{W}? \; ? \; \underline{A}$$

A regra do homomorfismo representa este processo de tradução de um dado termo como sendo,

$$h(?(t1, .., tn)) \; ? \; ?^{\underline{A}}(h(t1), .. , h(tn))$$

ou seja, o significado de um termo numa álgebra A é dado pelo significado do seu operador raiz na álgebra A aplicado aos significados de cada um dos seus argumentos.

Assumindo, como se viu, que cada ambiente é representado no sistema por

```
Environ :: S: ?ig   ;     assinatura
           M: ??    ;     função significado dos termos
           D: ?A    ;     operadores derivados
           K: Knlg  ;     directoria de termos
```

então este homomorfismo semântico é especificável como,

```
??    = Sort -> Homo;
Homo = Op -> ExpSemantica
```

tratando-se, basicamente, de um processo denotacional segundo o qual cada termo de uma espécie sintáctica é feito corresponder, estruturalmente, a uma representação no domínio semântico correspondente.

A função *heval*, que se apresenta a seguir, especifica este processo de tradu-ção, sendo evidente da especificação a natureza estrutural, logo recursiva neste caso, do processo.

```
FUNC heval(t: W? , ?: ?? , sig: ? ig) : ExpSemantica
RETURNS
        let (   ? = O(t),
                opSort = oSort(? , sig),
                homo = ?[opSort],
                semExp = homo[? ] )
        in
           if A(t) == <>  then semExp
                          else <semExp> ? <heval(tt,  ?, sig) | tt <- A(t)>;
```

De notar que a expressão semântica gerada tem uma forma prefixa pura, i.é., é da forma *f x g w z* em vez de *f(x, g(w, z))*.

No sistema AGSYS uma ?-álgebra A foi definida, correspondendo ao nível se-mântico das *imagens*, sendo cada um destes termos, que representa já em abs-tracto o nível gráfico, traduzido para o conjunto de primitivas gráficas para tal efeito desenvolvidas[18] e, consequentemente, visualizado em ecrã.

5.4.- Sumário e Conclusões.

Neste capítulo introduziu-se e formalizou-se a noção de *arquétipo*. Arquétipos são entidades sintácticas parametrizadas que representam a estrutura a seguir na construção de objectos, tal como definido pela respectiva álgebra de "manu-factura".

Seguindo uma perspectiva de formalização de base algébrica, arquétipos fo-ram então definidos como *termos com variáveis* gerados a partir da assinatura da álgebra, sendo assim objectos parametrizados e genéricos, representativos de classes de objectos concretos. Mostrou-se até que esta perspectiva algébrica não é exclusiva, e que também a partir de uma descrição gramatical, e seguindo um processo sistemático, se pode encontrar a equivalente assinatura algébrica que irá gerar os arquétipos.

O processo de construção de objectos concretos a partir dos arquétipos pode fazer-se seguindo um modelo de *edição interactiva orientada pela estrutura*, permitindo-se instanciações parciais, substituição de termos por termos,

[18] O autor agradece aqui as primitivas de entrada de baixo-nível então desenvolvidas pelo Zé João, colega do Departamento de Informática da U.M. [Martins e Oliveira 86a].

nave-gação estrutural, e aplicação de arquétipos (como operadores derivados) a argu-mentos. Todas estas operações com e sobre arquétipos foram especificadas for-malmente, tendo-se considerado como contexto um sistema genérico de arqué-tipos, tendo-se ainda referido como exemplo concreto de aplicação de arquétipos um pequeno sistema gráfico simbólico, o AGSYS, cuja especificação se apresenta no APÊNDICE C.

Este sistema gráfico especificado e desenvolvido, com base num modelo de interacção por *edição orientada pela estrutura descrita por arquétipos*, pode ser visto como um sistema gráfico simbólico[19] baseado em termos algébricos. A sua aplicabilidade em sistemas de CAD e CIM, por exemplo para a construção de bases de dados de descrição de produtos e, até, na modelação de componentes, parece ser de explorar. De facto, considerando que numa base de dados de des-crição de produtos esta descrição é realizada indicando como o produto final se compõe a partir das partes, usando operadores tais como "connect", "side-by-side", "fit together", etc., e ainda propriedades geométricas e não-geométricas, parece bastante natural uma abordagem simbólico-algébrica a este problema. Tendo em atenção que, não só uma álgebra de "manufactura" permitiria definir todas as construções possíveis e, a partir da sua assinatura, todos os arquéti-pos representativos, como se teria, de imediato, um esquema de manipulação destes termos e, também, por via da axiomática especificada, um conjunto de regras bem definidas para inferência sobre, por exemplo, a dependência transi-tiva das dimensões de um produto constituído por múltiplas peças.

No que diz respeito à modelação geométrica, em particular a modelação de base construtiva, Crep[20], onde operadores lógicos, tais como ?, ? e ? , são usa-dos sobre primitivas tais como cilindros, cones, caixas, etc., a representação in-terna consiste em geral, em ambientes imperativos, num emaranhado de listas ligadas representando a sequência de operadores e primitivas que são intrinse-camente hierárquicos. Para além dos arquétipos oferecerem uma representação mais próxima do que se pretende modelar, considerar uma Crep como um termo algébrico parece simplificar não só a sua manipulação como também, com base em propriedades algébricas definidas, facilitar a aplicação de transformações que possam conduzir, por reescrita algébrica, a simplificações e a termos for-malmente equivalentes.

O autor tem plena consciência de que as anteriores considerações podem estar carregadas de algum carácter especulativo e não científico, mas apenas porque, não sendo um investigador na área da Computação Gráfica, nunca teve a possibilidade de realizar qualquer experimentação, excepto no caso particular do pequeno sistema AGSYS, relativamente à aplicação do modelo algébrico dos arquétipos em tal área. Parece no entanto importante,

---

[19] na linha de sistemas simbólicos como o AutoCAD? , baseado em *S-expressions*.

[20] *Constructive Representation cf. Constructive Solid Geometry* (CSG) [Requicha 80].

ainda que uma experi-mentação futura venha a demonstrar o contrário, deixar aqui expressa alguma intuição sobre as vantagens da aplicação do modelo algébrico dos arquétipos nestas áreas, quanto mais não seja, como complemento, em termos de estrutu-ração da representação, aos sistemas de CAD baseados em linguagens e repre-sentações funcionais.

A aplicação dos arquétipos no contexto do desenvolvimento de IU, o tema da tese, merece outro tipo de considerações. Em primeiro lugar, o *modelo algébrico dos arquétipos*, ou seja, um modelo que se baseia numa álgebra representativa do processo de construção de objectos, e na sua representação sob a forma de termos genéricos a instanciar segundo um processo de edição orientada pela estrutura, é extremamente apelatório se considerarmos um modelo de interac-ção em que os objectos a sintetizar, ou construir, devem ser comandos correctos de uma linguagem de interacção, comandos esses que são feitos corresponder à invocação de rotinas da aplicação interactiva.

Sendo comandos estruturas sintácticas, e sendo, como se mostrou, estrutu-ras sintácticas expressáveis sob a forma de termos algébricos, podemos então considerar que a linguagem de interacção pode ser associada a uma álgebra de "manufactura" de comandos, sendo os arquétipos representações dos comandos a instanciar. A edição orientada pela estrutura passa a ser vista como interac-ção para síntese de um comando orientada pela estrutura deste, o que passa a consistir, fundamentalmente, em interacção para síntese dos seus argumentos, seguindo restrições semânticas contextuais.

A associação é, portanto, evidente e simples. Uma linguagem de comandos é associada a uma álgebra de "manufactura" de comandos. Cada comando é re-presentado por um arquétipo por instanciar. A instanciação de um comando é realizada seguindo o paradigma de interacção orientada pela estrutura interna do objecto a construir, situando-se, naturalmente, os problemas de navegação correcta na navegação que deve ser realizada, neste caso, para síntese dos valo-res dos argumentos do comando.

A complexidade desta navegação fez com que, embora assumindo por base de representação o modelo dos arquétipos e o modelo de interacção por edição orientada pela estrutura subjacente, na descrição de controladores para IU, um mecanismo menos explícito de especificação fosse desenvolvido, os *Guiões de Interacção*, de base mais declarativa, mas que, apesar de tudo, dão origem a representações explícitas desta navegação para síntese de dados, sob a forma de Redes de Petri, conforme se apresenta no capítulo seguinte.

Ainda que os arquétipos e a navegação estrutural sobre os mesmos não seja explícita nos *Guiões de Interacção*, eles estão na génese dos modelos e das nota-ções para o desenvolvimento de Sistemas Interactivos que são apresentados nos capítulos seguintes.