ESPECIFICAÇÃO DE OBJECTOS COM DIAGRAMAS:
ABORDAGEM OBLOG

Amílcar Sernadas, José Félix Costa e Cristina Sernadas

Secção de Ciência da Computação
Departmento de Matemática
Instituto Superior Técnico
&
Grupo de Ciência da Computação
INESC

Prémio Descartes 1992
(Ministério das Finanças)

Sumário

Tirando partido dos mais recentes avanços nos domínios das linguagens de especificação de muito alto nível centradas em objectos, das estações de computação pessoal e dos ambientes de trabalho cooperativo, propomos uma estratégia inovadora para o desenvolvimento de sistemas de informação de grande escala. A estratégia baseia-se na ideia de adoptar a mesma linguagem (de especificação) em todas as etapas do ciclo de vida da aplicação (desde a análise de requisitos até à manutenção do sistema). O código de implementação é gerado (semi-automaticamente) a partir das especificações e nunca é manipulado directamente. Os especificadores (antigos analistas e programadores) desenvolvem o seu trabalho com recurso a uma bancada apropriada. A linguagem proposta é descrita com algum pormenor a fim de ilustrar a sua versatilidade, universalidade, expressibilidade e convivialidade. Analisamos, por fim, os resultados das experiências práticas já realizadas (desenvolvimento da própria bancada e de um pacote bancário).

Índice

1 Introdução 3
2 Estratégia proposta 4
3 Dificuldades a ultrapassar 5
4 Conceitos primitivos da abordagem 7
5 Traços gerais da linguagem 9
6 A linguagem através de exemplos 11
6.1 Universo das classes 13
6.2 Descrição de classe 13
6.3 Especialização 16
6.4 Especificação textual equivalente (GNOME) 17
6.5 Abstracção 19
6.6 Interacção 19
6.7 Reificação 20
7 Notas finais 25
8 Agradecimentos 27
9 Referências 28

1 Introdução

A indústria informática na sua vertente de produção de *software* tem estado em crise desde meados dos anos sessenta: a capacidade instalada de produção tem vindo sempre a ser deficitária face à procura constante de novos sistemas e aplicações. Os progressos espectaculares a nível da tecnologia de *hardware* não têm sido acompanhados pelos bem mais lentos avanços a nível da produção de *software*[1]. Enquanto presenciamos vertiginosos aumentos do poder de cálculo, da capacidade de memória e das facilidades de comunicação (sem acréscimo de preços, bem pelo contrário), do lado do *software* os progressos reais são discutíveis.

Assistimos, desde os anos sessenta, à implantação definitiva das *linguagens de programação* de 3ª geração em detrimento das linguagens de baixo nível anteriores, à adopção não completamente generalizada da tecnologia de *bases de dados* e, ulteriormente, ao aparecimento de linguagens de 4ª geração e à adopção mais ou menos generalizada de *métodos sistemáticos* de desenvolvimento. Mais recentemente, começamos a ver avanços nos *ambientes de trabalho* do programador e do analista, mas estes últimos foram desencadeados mais pelos progressos nas estações de trabalho (*hardware*) do que pelos objectivos a nível de produção de *software*.

Se avaliarmos o progresso real em termos de produtividade, o cenário é francamente mau[2]: sabemos que, hoje como ontem, 80% dos recursos humanos da empresa típica estão dedicados à manutenção dos sistemas existentes e apenas 20% estão livres para responder às necessidades de novas aplicações; sabemos que, hoje como ontem, desde o momento da decisão de arranque até à entrada em operação de uma nova aplicação, largos meses (se não anos, consoante a complexidade do problema) vão decorrer.

Por outras palavras, a indústria de *software*, e em especial a de aplicações, continua a mostrar-se incapaz de responder em tempo útil às solicitações. Claro que também há aspectos positivos: a complexidade e a funcionalidade das aplicações produzidas têm vindo a crescer. Pede-se hoje mais do que ontem não só em termos quantitativos como qualitativos.

Mas, na prática, o que fica patente é a incapacidade de resposta (relativa à solicitação actual cada vez mais exigente). Neste enquadramento deficitário da oferta de *software*, as consequências das leis de mercado não se fizeram esperar: hoje a percentagem do custo total de um sistema informático atribuída ao *software* é muito maior do que há umas décadas. O custo do *hardware* tem decrescido e o mesmo, claro, não se passa com o *software*. Este fenómeno é geral em todos os sectores de aplicação e não apenas no dos sistemas de informação (em que o equipamento quase se limita aos meios computacionais): por exemplo, na produção de aviões o *software* de suporte ao fabrico e o *software* instalado absorvem uma fatia cada vez mais significativa do custo total do projecto.

---

1 Cf. [45].
2 Cf. [32].

Nestas condições, é imperativo aumentar a produção de *software*, o que se pode conseguir aumentando o número de analistas e programadores, aumentando a sua produtividade e/ou promovendo a re-utilização de (componentes de) *software* já existente. Os progressos havidos neste campo atacam sobretudo a questão da produtividade (por exemplo, a adopção de linguagens de programação de nível cada vez mais alto que tornam possível a programação expedita de aplicações).

Pretendemos apresentar aqui uma nova *estratégia de desenvolvimento* de aplicações (com especial relevo para sistemas de informação) que promete significativos acréscimos globais de produtividade. A Secção 2 descreve a estratégia nas suas diferentes facetas (linguagem, método e ferramentas), bem como as vantagens aparentes da sua adopção. A Secção 3 enuncia as principais dificuldades a ultrapassar. Nas Secções 4 e 5, respectivamente, são apresentados mais pormenorizadamente os conceitos primitivos e os traços gerais da linguagem OBLOG subjacente. A Secção 6 ilustra uma parte significativa da linguagem recorrendo a exemplos elementares. Finalmente, na Secção 7, é feita uma avaliação crítica da estratégia com base na experiência já adquirida, nomeadamente no desenvolvimento da bancada OBLOG utilizando a própria abordagem OBLOG.

2 Estratégia proposta

A estratégia proposta assenta na hipótese de que se deve melhorar a produtividade em *todas* as etapas do ciclo de vida da aplicação e não apenas na programação propriamente dita. Assim, o progresso decorrente da adopção de novas linguagens e ambientes de programação será sempre limitado. Este argumento não é novo uma vez que também têm sido realizados com regularidade progressos a nível das fases de análise e desenho: adopção de métodos sistemáticos cada vez mais eficazes e, ultimamente, ferramentas CASE[3] integrando técnicas de análise e desenho. Mas vale a pena analisar em pormenor as implicações da hipótese de melhoria global (desde a análise de requisitos à manutenção).

A nível da manutenção, salta à vista que o crescimento exponencial dos seus custos ao longo da vida da aplicação se deve, em boa parte, à dificuldade crescente de fazer alterações sobre alterações. A razão parece simples: as sucessivas alterações são tipicamente realizadas sobre o código (programas) com o consequente afastamento das especificações iniciais que não vão sendo actualizadas (por falta de meios). Alterações ulteriores requerem trabalhar em exclusivo com o código sem o apoio importante das descrições de mais alto nível (especificações produzidas nas fases de análise e desenho).

A nível das fases de desenvolvimento, é evidente que um factor decisivo de abaixamento da produtividade é a existência de descontinuidades[4] entre as diversas fases: o resultado da análise

---

3 *Computer Aided Software Engineering* ou *Computer Aided Systems Engineering*.

4 *Traduttore traditore*, aforismo italiano.

tem de ser interpretado e re-escrito na fase de desenho; e o mesmo se passa entre a fase de desenho e a de programação.

A redução das tarefas de manutenção e das descontinuidades entre as etapas de desenvolvimento aponta para uma automação cada vez maior da transformação das especificações em código. No limite, não seria necessário codificar (ao nível que entendemos esta actividade hoje em dia, isto é, usando linguagens de 3ª ou 4ª gerações). Trabalharíamos apenas com as especificações (cada vez mais refinadas ao longo do processo de desenvolvimento). Em cada etapa seriam incorporados mais detalhes na especificação e revistos alguns dos produzidos anteriormente. No final, o sistema operacional seria gerado automaticamente (ou semi-automaticamente) a partir das especificações na forma final. A manutenção far-se-ia directamente sobre estas. Mais ainda, em qualquer ponto do desenvolvimento, seria possível testar o sistema especificado até esse momento[5] com as consequentes vantagens na adequação do produto aos requisitos.

Nesta solução limite ideal, todos contribuiriam para o produto final: todo o fragmento de especificação escrita por um elemento da equipa de desenvolvimento acabaria incorporado na especificação final (sem traduções!), a menos que fosse posta de parte ou revista. Sem descontinuidades semânticas entre as diversas fases, seria mais fácil garantir a adequação do produto final aos requisitos e controlar os custos e tempos de desenvolvimento e manutenção.

Assim, somos tentados a propor esta estratégia: *adopção de uma única linguagem a utilizar desde as fases iniciais de análise até ao fim da vida da aplicação*. Resta saber se as dificuldades inerentes a tal estratégia são ultrapassáveis.

3 Dificuldades a ultrapassar

A principal dificuldade está em encontrar o justo equilíbrio entre a *versatilidade* (a linguagem dever ser suficientemente versátil para ser utilizável nas fases iniciais do ciclo de vida da aplicação, nomeadamente na identificação dos requisitos de informação e na modelação do universo do discurso[6]) e a *expressibilidade* (a linguagem deve ser suficientemente poderosa para permitir detalhar a especificação tanto quanto necessário). Os progressos realizados, na última década, no campo das linguagens de especificação de muito alto nível[7] possibilitaram o desenvolvimento de uma linguagem aceitável, conforme se verá adiante.

Outro importante tipo de dificuldades decorrentes da adopção de uma única linguagem para todo o ciclo de vida está na necessidade de mudar completamente os *métodos de trabalho* de analistas e programadores: basicamente, desaparece a fronteira entre uns e outros. Todos

---

5 *Prototyping*.

6 Pelo menos tão versátil quanto as técnicas actualmente usadas para o efeito (diagramas de fluxos de dados e entidades-associações).

7 Nomeadamente, para especificação de tipos de dados abstractos [2,22] e de sistemas reactivos concorrentes [37].

devem passar a ser *especificadores*. Claro que uns, pelas suas aptidões especiais e conhecimento do negócio, darão maior contribuição (isto é, produzirão mais especificações) nas fases iniciais do desenvolvimento; outros, pelas suas capacidades específicas e conhecimento das realidades informáticas, darão maior contribuição nas fases finais do desenvolvimento. Estes novos métodos de trabalho exigirão também novas técnicas de *controlo de projecto*: por exemplo, determinar quais deverão ser as etapas e os produtos intermédios a controlar.

A linguagem em vista deverá ser *formal* pois, em última análise, serve de base à produção de código entendível por sistemas computacionais sem custos de processamento demasiado elevados[8]. No entanto, deve ser facilmente dominada pelos actuais analistas, o que sugere a adopção de uma linguagem *diagramática* (gráfica).

A terceira ordem de dificuldades, também evidente, está em garantir o nível de *desempenho*[9] do sistema gerado a partir das especificações finais. A questão é semelhante à que se pôs quando se substituiram as linguagens máquina[10] pela linguagem COBOL. Agora, como nessa altura, é óbvio que o programador excepcional que domina a linguagem de mais baixo nível poderá sempre fazer melhor. Mas o objectivo é aumentar a produtividade dos programadores não excepcionais, que são em maior número! De qualquer modo, este problema é tão crucial que influencia também os métodos de trabalho.

Para além dos especificadores já identificados, serão ainda necessários alguns (poucos) *optimizadores* cuja função será guiar a geração (semi-automática) do sistema operacional (por exemplo, por tradução para código intermédio de uma linguagem de 3ª geração). Sem intervenção do optimizador, o código será gerado por omissão de acordo com um molde prefixado. Mas o optimizador pode intervir quando necessário para optar por moldes alternativos. Esta optimização corresponde a uma generalização do conceito bem conhecido da tecnologia de bases de dados, de separar a estrutura lógica da estrutura física (só nesta última são introduzidos, por exemplo, índices para optimizar acessos). É evidente que a adequação lógica das especificações deve ser completamente alheia a tais optimizações. Estas deverão poder ser alteradas sem qualquer impacte na especificação.

Há, pois, que garantir um elevado grau de *independência física* das especificações relativamente ao código gerado. Esta é uma importante dificuldade a ser ultrapassada pela ferramenta de geração do código.

Outras dificuldades a considerar têm a ver com o facto de a estratégia proposta implicar que todos os especificadores (antigos analistas e programadores) trabalhem sobre o mesmo objecto (a especificação). Fica pois agravado o problema do *trabalho cooperativo*: mais pessoas estarão ao mesmo tempo a trabalhar sobre a especificação. Este problema tem de ser encarado ao nível da linguagem (que deverá facilitar o trabalho cooperativo, permitindo, por exemplo, a delimitação

---

8 O processamento computacional de especificações em linguagem natural ainda não é viável economicamente, e talvez nem mesmo tecnicamente, para este fim específico.

9 *Performance*.

10 *Assembly languages*.

de unidades de gestão[11]), ao nível da metodologia (quem faz o quê e quando? quem é responsável pelas diversas unidades?) e, ainda, ao nível das ferramentas.

A estratégia proposta só será viável se recorrermos a um conjunto integrado de *ferramentas CASE* para edição e teste das especificações, geração de protótipos e do sistema final com optimização opcional, etc. Tais ferramentas, para suporte adequado à utilização cooperativa de uma linguagem diagramática, só são possíveis há relativamente pouco tempo, tirando partido dos grandes avanços em estações de trabalho individuais com grande capacidade (de cálculo e memória) e redes locais.

4 Conceitos primitivos da abordagem

Esboçada a estratégia, põe-se a questão da escolha das primitivas semânticas adequadas a uma linguagem utilizável desde as fases iniciais do desenvolvimento da aplicação, mas suficientemente poderosa para permitir a especificação exaustiva do sistema. Há ainda que garantir à partida a satisfação de outros requisitos, de que se destaca a adopção de construtos que promovam a re-utilização das especificações e dos sistemas especificados.

O muito alto nível de abstracção pretendido sugere a adopção da primitiva *objecto* (presente em tantas abordagens à modelação conceptual[12]). Aqui entende-se por *objecto*[13]:

> algo que existe (no mundo real ou no sistema informático), que é criado e eventualmente destruído; entretanto, enquanto existe, pode sofrer alterações no seu *estado* por interacção com outras entidades; o seu estado reflecte-se nos valores dos *atributos*; muda de estado quando algum *evento* relevante acontece.

A título de ilustração, considere-se uma conta bancária como um objecto. Um dos seus atributos é o saldo. Entre os eventos relevantes encontram-se débito e crédito.

Assim, especificar um sistema corresponde a definir um conjunto de objectos. Em geral, surgem muitos objectos similares, do mesmo tipo: são agrupados em *classes*. De facto, temos de especificar as classes e não os objectos que surgem como instâncias daquelas. Por exemplo, especificamos a classe conta bancária admitindo que possa ter muitas instâncias.

O sistema assim construído é uma comunidade de objectos (instâncias de classes) que interactuam uns com os outros (por partilha ou por chamada de eventos): um débito feito pelo leitor na sua conta bancária é um evento que surge tanto na conta como em si próprio (também considerado objecto). Sublinhe-se que os objectos da comunidade evoluem independentemente uns dos outros excepto nos momentos de interacção: visão do sistema como conjunto de objectos

---

11 *Management units*.
12 Tais como [1,13,14,29,33,34,35,36,38,39].
13 Cf. [19,20,21,39,41,43,48].

concorrentes que podem interactuar. A mesma visão é válida para o mundo real em que o sistema se insere.

É óbvio que alguns objectos são *passivos* (nada lhes acontece a menos que outros objectos interactuem com eles: caso da conta bancária), enquanto outros são *activos* (evoluem por iniciativa própria: caso dos clientes do banco). No sistema informático são exemplos de objectos passivos os registos na base de dados e são exemplos de objectos activos os programas em execução (aos quais é atribuído CPU).

A linguagem pretendida deve, pois, permitir a descrição de classes de objectos (passivos e activos, de maneira tão uniforme quanto possível) e das suas interacções. Por omissão, admite-se que os objectos evolvem em concorrência.

A noção de objecto integra dados (atributos) e processamentos (eventos). Permite assim estruturar de maneira eficaz a especificação e o sistema (ou universo do discurso) descrito. Esta estruturação vai ao encontro da maneira como o analista encara a realidade. É uma faceta essencial das abordagens ditas *centradas em objectos*[14].

Outra faceta característica destas abordagens diz respeito a construtos que promovem a re-utilização (neste caso de especificações e dos sistemas assim especificados). A forma mais conhecida de tais construtos é a *especialização*. Para ilustração, voltemos ao exemplo das contas bancárias. Quer ainda na fase de especificação, quer já na fase de manutenção, pode surgir a necessidade de introduzir contas de poupança "que são contas mas têm algumas peculiaridades[15]".

A linguagem pretendida deve possuir um mecanismo de especialização que permita fazer isto mesmo: especificar a classe conta de poupança reaproveitando tudo o que se queira da especificação de conta. Note-se que já na fase de operação do sistema deve ser possível enriquecê--lo com a nova classe sem que isso afecte os objectos já existentes. Mais ainda, outros objectos que interactuem com contas também poderão passar a interactuar com contas de poupança, encarando-as como contas (sem dar pelas referidas peculiaridades).

A estruturação eficaz (dados+processamentos) permitida pela noção de objecto e os níveis de re-utilização promovidos pelo mecanismo de especialização conduzem a grandes ganhos a nível da verificação e manutenção do sistema produzido[16].

A abordagem proposta surge, pois, como pertencendo à família da abordagens centradas em objectos (com as vantagens daí decorrentes e acabadas de enunciar). Mas contém alguns elementos menos vulgares nessas abordagens, isoladamente ou em conjunto. Já referimos a visão dos objectos concorrentes.

---

14 *Object-oriented*.
15 Tais como: mais atributos, mais eventos, comportamento mais constrangido.
16 Cf. [11,12,30,54,55].

Outro aspecto fulcral é a possibilidade de refinar progressivamente a especificação. Por exemplo, de início podemos considerar um débito como uma operação atómica (evento). Contudo, ulteriormente, iremos com certeza reconhecer que, afinal, um débito é uma operação complexa com vários passos, mas que, no entanto, deve manter a atomicidade: isto é, um débito é uma transacção[17].

A linguagem deve, pois, permitir a *reificação* de especificações com objectos de mais baixo nível já especificados, suportando a primitiva *transacção* a nível lógico. Esta dimensão do processo de especificação tem sido ignorada nas abordagens centradas em objectos (a solução usual é uma utilização abusiva da especialização e, portanto, sem a noção de transacção).

Em oposição à especialização surge a noção de *abstracção* em que se extrai por capsulamento um objecto mais simples de outro já dado: mecanismo semelhante ao das perspectivas em bases de dados, mas aqui estendido a objectos. Este mecanismo permite fazer crescer o reticulado de herança em direcção oposta à especialização. Também tem sido ignorado nas abordagens centradas em objectos, apesar das inúmeras vantagens a nível da *independência lógica* entre objectos.

Todos estes construtos são ilustrados adiante (ainda que com exemplos elementares) na Secção 6. Outras facilidades da linguagem proposta (não ilustradas neste artigo) dizem respeito às *unidades de gestão* para promoção do trabalho cooperativo de uma equipa com vários (ou muitos) elementos a trabalhar simultaneamente sobre a mesma especificação. Estas são directamente ligadas aos conceitos básicos da abordagem (objecto e reificação), ao contrário das linguagens actuais em que as unidades de gestão (ficheiro em que se encontra o programa ou especificação) nada têm a ver com os construtos das linguagens.

5 Traços gerais da linguagem

A linguagem da abordagem OBLOG começou a ser desenvolvida no INESC em 1986, inicialmente como linguagem textual de modelação conceptual[18]: chegou a ser produzido um processador incipiente com financiamento de um projecto europeu.

Só em 1989 surgiu interesse comercial na abordagem[19]. A dinâmica decorrente veio a afectar decisivamente o desenvolvimento ulterior da abordagem e da sua linguagem: (i) a opção pela linguagem de tipo diagramático foi feita nessa altura com base no perfil de alguns dos seus utilizadores potenciais (analistas e, até, especialistas de organização e métodos); (ii) também só nessa altura se esboçou a estratégia de utilização da mesma linguagem em todas as etapas do ciclo de vida da aplicação; (iii) e, finalmente, foi só por solicitação dos interesses comerciais que se

17 Cf [43,49,52].
18 Cf. [3,4,5,41].
19 Por parte do Grupo Espírito Santo (GES).

encarou a hipótese de gerar código operacional (automatica ou semi-automaticamente) a partir das especificações OBLOG.

A partir de 90, o esforço de implementação da linguagem na sua versão gráfica (no contexto de uma bancada CASE integrada) saiu do ambiente universitário do INESC: da fase do protótipo passava-se à do produto comercial[20].

Entretanto, continuámos no INESC a definição rigorosa da semântica da linguagem (nas suas versões textual e diagramática): o problema principal a ultrapassar foi a chocante ausência de fundamentação das noções subjacentes às abordagens ditas centradas em objectos. Apesar da popularidade crescente deste paradigma computacional, os seus fundamentos encontravam-se nos finais dos anos 80 ainda muito pouco desenvolvidos. Os trabalhos sobre a semântica da linguagem OBLOG têm contribuído para o esclarecimento de muitas das questões em causa[21].

A linguagem OBLOG e os métodos associados têm sido, entretanto, documentados progressivamente em diversas publicações[22]. Fazemos aqui uma síntese da situação actual, incluindo os últimos desenvolvimentos.

As características fundamentais da linguagem, já afloradas nas secções anteriores, resumem-se rapidamente:

- alto nível de abstracção, permitindo a sua utilização nas fases iniciais do desenvolvimento de aplicações (enunciado de requisitos de informação e modelação conceptual);

- fácil utilização mesmo pelos agentes sem formação informática, para o que contribui decisivamente a sua natureza diagramática e as primitivas semânticas de alto nível;

- largo espectro de aplicação graças à universalidade destas primitivas;

- grande poder de expressão, permitindo a sua utilização nas fases ulteriores de desenvolvimento da aplicação até à especificação exaustiva do sistema pretendido e durante a sua manutenção;

- possibilidade de geração (automática ou semi-automática) das aplicações a partir das especificações;

- adequação ao desenvolvimento de grandes aplicações (milhões de linhas de código tradicional de 3ª geração).

Numa perspectiva mais técnica, devemos salientar os seguintes aspectos:

---

20 O desenvolvimento da bancada OBLOG arrancou no início de 1990 na ESDI, empresa do GES.
21 Cf. [6,7,8,9,10,15,16,17,18,27,40,42].
22 Cf. [10,28,44,48,51,52,53].

- linguagem de tipo "centrada em objectos", com alguns enriquecimentos a seguir referidos;

- linguagem formal com semântica matemática rigorosa e para a qual é possível desenvolver um cálculo lógico para verificação da coerência das especificações[23];

- inclusão de construtos de estruturação dos dados[24], incorporando na abordagem centrada em objectos a experiência adquirida na área do desenho lógico de bases de dados;

- adopção do paradigma dos objectos concorrentes, particularmente adequado ao desenho lógico de sistemas a implementar em arquitecturas abertas, multi-processador e do tipo cliente-servidor;

- descrição uniforme (com os mesmos construtos) de objectos passivos e activos, removendo do desenho lógico a tradicional barreira de Von Neuman entre memória e processadores;

- disponibilidade de construtos para a reificação progressiva das especificações, adoptando como primitiva lógica a noção de transacção;

- possibilidade de constituição de bibliotecas de especificações, não só de objectos como também de comunidades destes (incluindo interacções);

- inclusão do mecanismo dual da especialização (tradicional em linguagens centradas em objectos): a abstracção que permite definir perspectivas (como é usual no contexto da tecnologia de bases de dados);

- integração da noção de unidade de gestão (do trabalho cooperativo) nas primitivas da linguagem.

Na secção seguinte ilustramos a maioria dos conceitos básicos da abordagem tal como se reflectem nos construtos da linguagem na sua versão diagramática mais actual. Exemplificamos também um dos aspectos mais avançados (reificação).

6 A linguagem através de exemplos

Tratando-se de uma linguagem diagramática, faz sentido ilustrar a sua utilização com base nos diversos tipos de diagrama disponíveis:

---

23 Cf. [23,24,25,26,31,46,47,49,50].

24 Mecanismos de *data modeling*, tais como dependências funcionais e chaves.

- *diagrama de universo de classes*: cada partição[25] da especificação contém um diagrama deste tipo onde se indicam as suas classes, respectivas chaves, e relacionamentos entre classes[26];

- *diagrama matricial de classe* (um por cada classe): em que se declaram os respectivos atributos e eventos[27];

- *diagrama de comportamento de classe* (um por cada classe): em que se define o comportamento permitido a cada instância da classe (que eventos podem acontecer, por que ordem e em que situações);

- *diagrama de efeitos de evento sobre atributos* (um por cada tipo de evento de cada classe): diagrama que estabelece os efeitos da ocorrência do evento nos valores dos atributos;

- *diagrama de interacção entre classes* (um por cada conjunto de classes cujas instâncias interactuam por partilha ou chamada de eventos): diagrama em que se indicam que eventos de uma classe correspondem a eventos noutra classe;

- *diagrama de atribuição de valores* (sempre que necessário): diagrama que serve para impor condições nos valores dos atributos e/ou dos parâmetros dos eventos.

Perante esta proliferação de diagramas, o leitor deverá interrogar-se sobre se as especificações OBLOG serão pouco concisas. Discutiremos esta importante questão[28] na última secção. Veremos que, apesar da proliferação de diagramas, a linguagem é de facto muito concisa quando comparada com as linguagens tradicionais.

De seguida, ilustramos a utilização destes diagramas básicos a propósito de um exemplo elementar (escolhido por dispensar explicações na comunidade dos informáticos): constituição de uma biblioteca de pilhas e outras estruturas de informação. Damos ênfase à utilização da especialização por ser tão importante para a reutilização (bibliotecas e aplicações). Incluímos ainda a especificação textual equivalente[29] dos diagramas apresentados. Finalmente, ilustramos os mecanismos de abstracção (capsulação) e reificação (implementação).

Trata-se de uma exposição muito sintética: o leitor interessado numa descrição pormenorizada da linguagem OBLOG e da sua semântica matemática deverá consultar a documentação disponível[30].

---

25 A partição é uma unidade de gestão central na subdivisão do trabalho e também importante no processo de reificação.

26 Os relacionamentos entre classes são tipicamente reflectidos por atributos entre elas.

27 Alguns dos atributos já poderão ter sido definidos no diagrama do universo em causa, caso em que são copiados automaticamente pelo editor para o diagrama matricial.

28 Imediatamente identificada pelos primeiros analistas que utilizaram a linguagem em aplicações reais.

29 Na linguagem GNOME: versão textual da linguagem OBLOG usada actualmente no INESC para experimentação de novas primitivas.

30 E em especial [10].

6.1 Universo das classes

No diagrama da Fig. 1, relativo ao universo das classes (*data model*, na terminologia tradicional de bases de dados), introduzimos as várias classes relevantes, estabelecemos alguns relacionamentos (por exemplo, o relacionamento duplo entre as bombas e as pilhas reflectido nos atributos `origem` e `destino` da classe `BOMBA` e o relacionamento *is-a* entre a classe `PILHA-DIMENSIONADA` e a classe `PILHA`[31]) e dotamos a classe `PILHA` do atributo chave `nome` com valores no tipo de dados *`string`*.

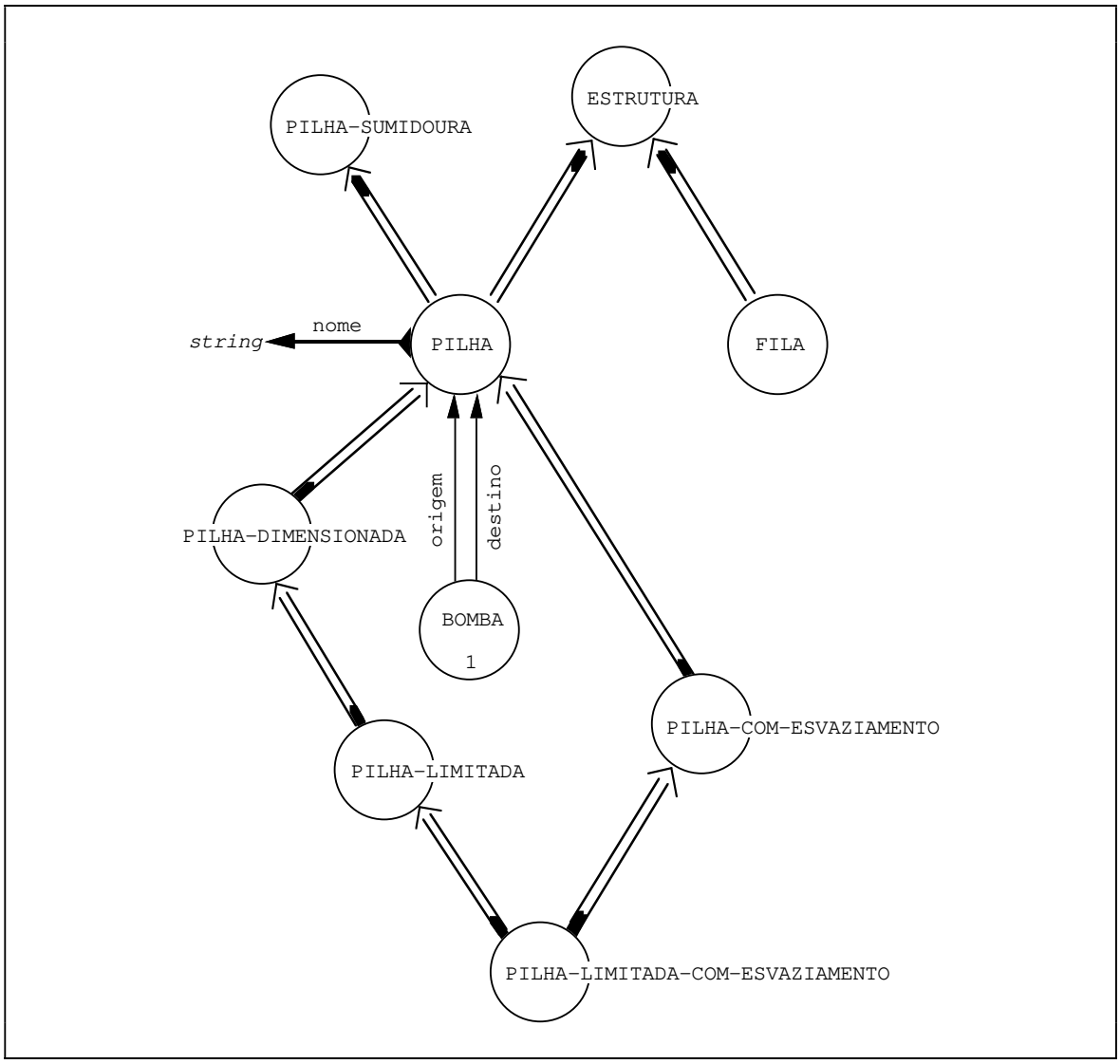

Fig. 1: diagrama do universo das classes

6.2 Descrição de classe

No diagrama da Fig. 2 apresentamos os atributos da classe `PILHA`:

[31] Relacionamento de especialização: todas as instâncias da classe `PILHA-DIMENSIONADA` são também instâncias da classe `PILHA`.

- o atributo `vazia` de codomínio `bool` (tipo de dados: Booleanos);
- o atributo `topo` de codomínio `int` (tipo de dados: inteiros);
- e ainda o atributo `nome` automaticamente importado do diagrama do universo das classes.

Também introduzimos na Fig. 2 os eventos da classe `PILHA`:

- o evento de nascimento[32] paramétrico `abre` com parâmetro `id` de domínio `string`;
- o evento de morte[33] `fecha`; o evento paramétrico `sobrepõe` com parâmetro `valor` de domínio `int`;
- e o evento `retira`.

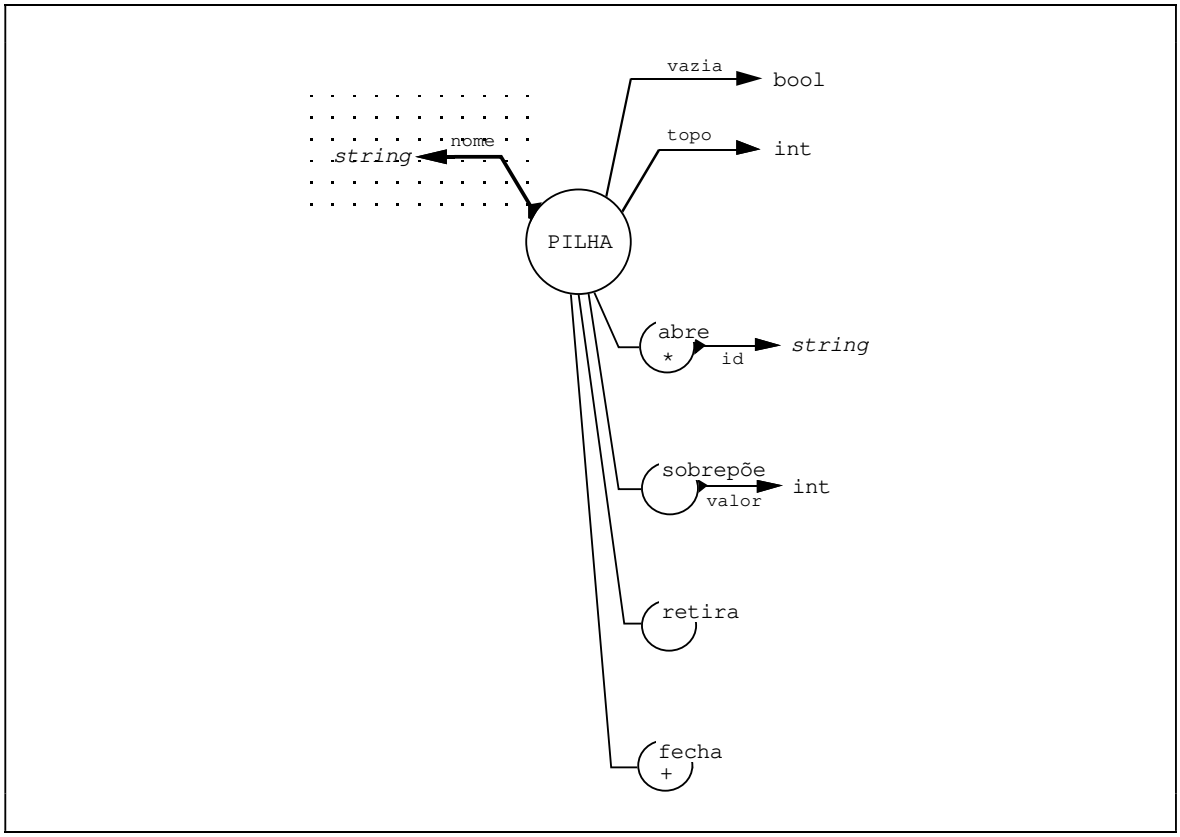

Fig. 2: diagrama matricial da classe `PILHA`

No diagrama da Fig. 3 definimos o comportamento das instâncias da classe `PILHA`. Após o nascimento (evento `abre`) cada pilha fica numa situação em que:

- pode ser destruída (evento `fecha`);
- ou pode sofrer uma ocorrência do evento `retira` (mas apenas na condição adicional do valor do atributo Booleano `vazia` ser falso) voltando à mesma situação;
- ou pode sofrer uma ocorrência do evento `sobrepõe` voltando também à mesma situação.

---

32 Os eventos de nascimento são assinalados com *.

33 Os eventos de morte são assinalados com +.

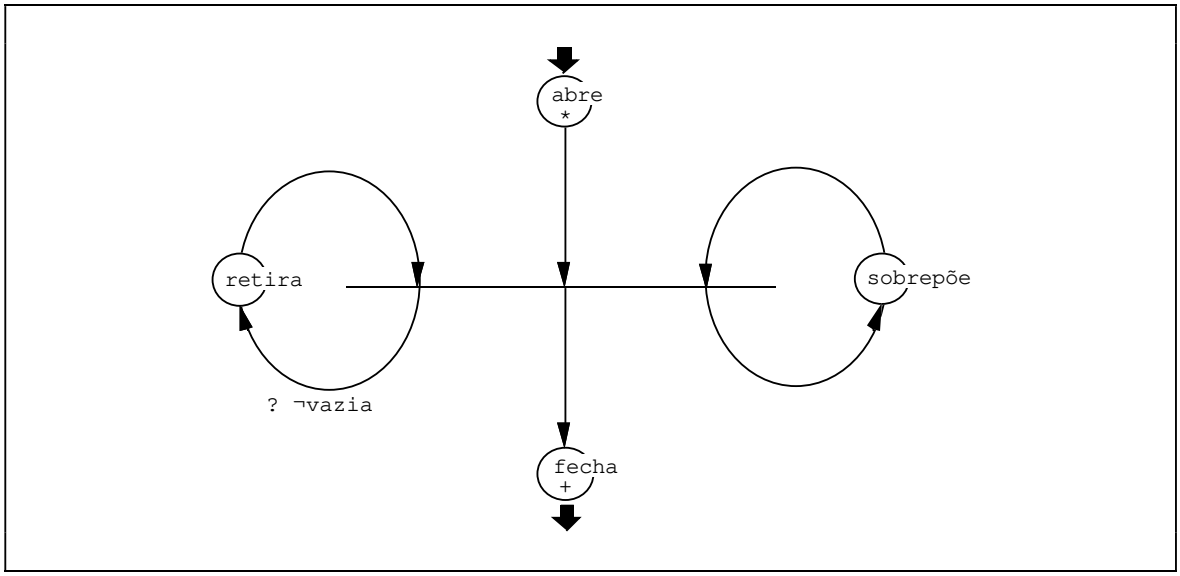

Fig. 3: diagrama de comportamento da classe `PILHA`

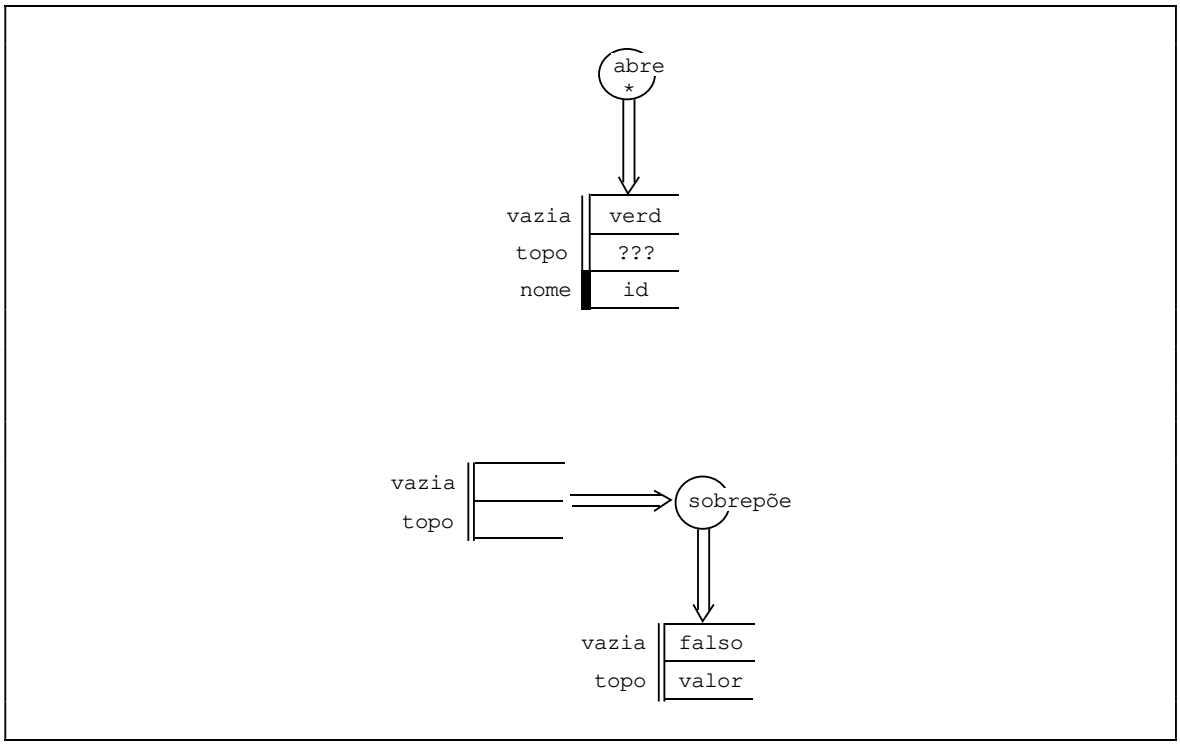

Fig. 4: diagramas dos efeitos dos eventos `abre` e `sobrepõe`

A Fig. 4 inclui os diagramas dos efeitos dos eventos `abre` e `sobrepõe`:

- Após o nascimento (ocorrência de `abre`) os valores dos atributos ficam como se segue: `vazia` com o valor verdadeiro; `nome` com o valor do parâmetro `id` da instância de `abre` em causa; e `topo` com valor não especificado.

- Após uma sobreposição (ocorrência de `sobrepõe`) os valores ficam do seguinte modo: `vazia` com o valor falso; e `topo` igual ao valor do parâmetro (`valor`) da instância de `sobrepõe` em causa.

6.3 Especialização

Vejamos agora como é possível introduzir uma especialização da classe pilha. Suponhamos que pretendemos detalhar a especificação da classe `PILHA-DIMENSIONADA` herdando a especificação da classe `PILHA`, mas adicionando mais um atributo (`dimensão`) para contar em cada momento o número de elementos na pilha.

A Fig. 5 contém o diagrama matricial da nova classe (`PILHA-DIMENSIONADA`) onde se encontra declarado o atributo adicional `dimensão` com codomínio `nat` (tipo de dados: naturais) e se indica que se trata de uma especialização da classe `PILHA` (indicação importada automaticamente do diagrama do universo das classes).

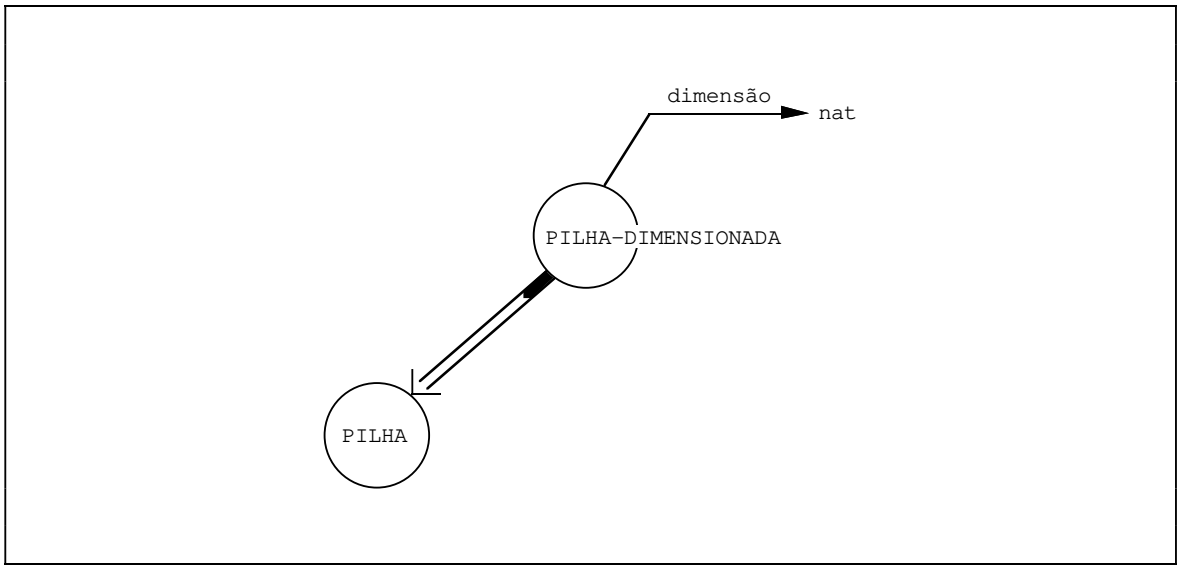

Fig. 5: diagrama matricial de `PILHA-DIMENSIONADA`

A Fig. 6 contém os diagramas dos efeitos dos eventos no novo atributo (os efeitos sobre os atributos herdados são por omissão os especificados para a classe pilha).

Não é necessário especificar o comportamento das instâncias da nova classe uma vez que se pretende que seja o mesmo das pilhas.

Trata-se de um exemplo muito simples de especialização. O mecanismo disponível na linguagem OBLOG é muito poderoso pois permite também:

- introduzir novos eventos (caso em que é necessário especificar os efeitos dos novos eventos também sobre os atributos herdados[34]);
- constranger o comportamento herdado, levando em linha de conta os novos atributos;
- enriquecer a especificação do comportamento, levando em linha de conta os novos eventos;
- redefinir parcial ou totalmente componentes da especificação herdada[35]: por exemplo, redefinindo os efeitos de um evento.

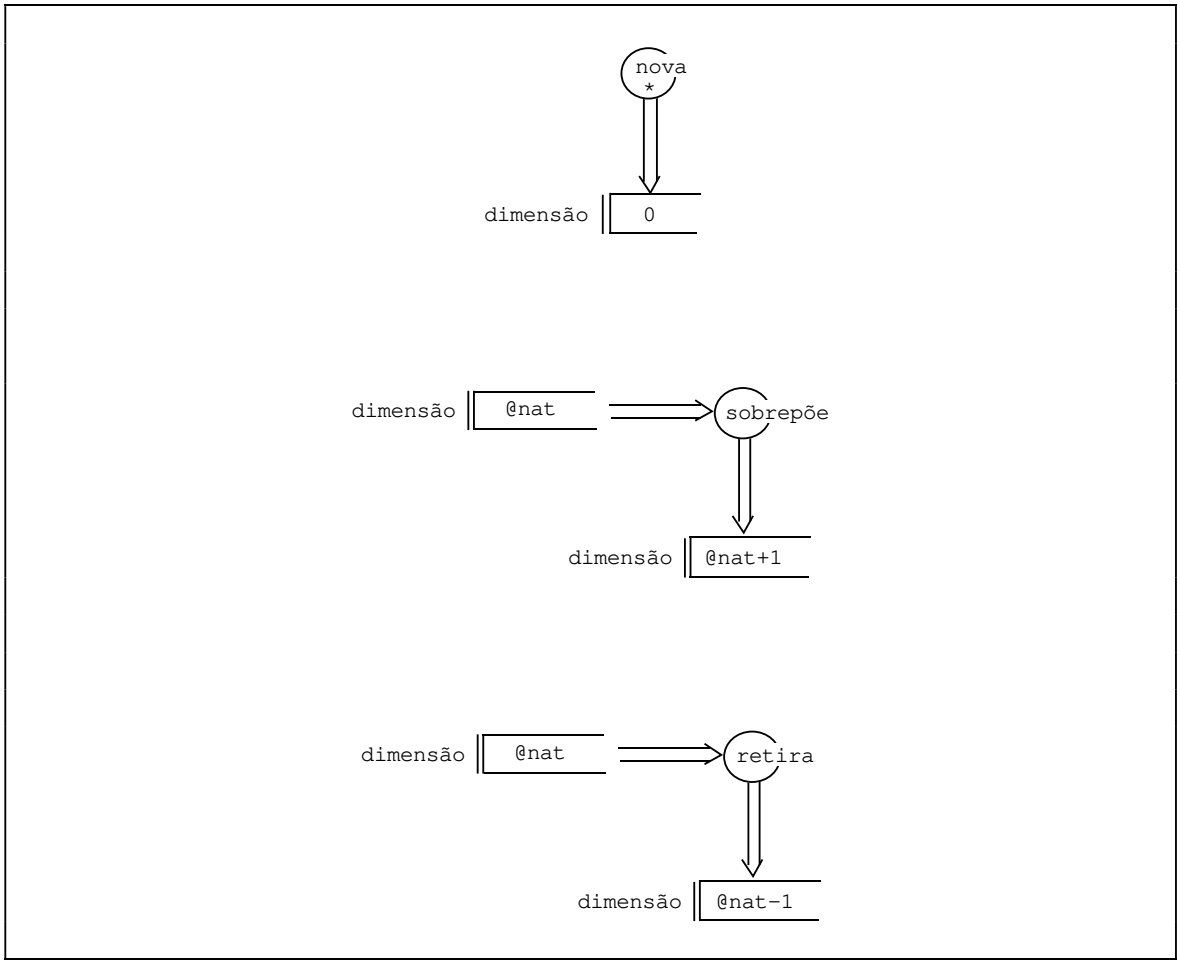

Fig. 6: diagramas dos efeitos dos eventos da classe PILHA−DIMENSIONADA

6.4 Especificação textual equivalente (GNOME)

Para fins de comparação, e em especial para avaliar os custos da adopção de uma linguagem diagramática, segue-se um extracto da especificação textual correspondente. As especificações textuais são claramente mais concisas, mas, por outro lado, apresentam desvantagens óbvias ao nível da aceitação por parte de agentes não especialistas em informática. Mais ainda, não são utilizáveis, nem parcialmente, no diálogo com os utilizadores, embora seja discutível se as especificações diagramáticas constituem progresso neste aspecto.

34 Efeitos colaterais (*side effects*).
35 Herança parcial (*overriding*).

```
let PILHA be instance of CLASSE
  template
    attributes
      topo: integer;
      vazia: boolean;
      inv nome: string
    actions
      birth abre;
      sobrepõe(integer);
      retira;
      death fecha
    axioms
      enabling
        {not(vazia)} retira
      valuation
        [abre(S)]vazia = true;
        [abre(S)]nome = S;
        [sobrepõe(N)]vazia = false;
        [sobrepõe(N)]topo = N
      state
        [sobrepõe(N);retira] = []
      key
        <nome>
  end
end

let PILHA-DIMENSIONADA be
  instance of CLASSE;
  specialization of PILHA
    template enriched by
      attributes
        comprimento: natural
      axioms
        valuation
          [nova]comprimento = 0;
          [sobrepõe(N)]comprimento = comprimento + 1;
          [retira]comprimento = comprimento - 1
    end
end

let PILHA-COM-ESVAZIAMENTO be
  instance of CLASSE;
  specialization of PILHA
    template enriched by
      actions
        esvazia
      axioms
        state
          [esvazia] = [abre(nome)]
    end
end

let PILHA-LIMITADA be
  instance of CLASSE;
  specialization of PILHA-DIMENSIONADA
    template enriched by
      axioms
        enabling
          {comprimento<50} sobrepõe(N)
    end
end

let PILHA-COM-ESVAZIAMENTO-LIMITADA be
  instance of CLASSE;
  multiple_specialization_of PILHA-LIMITADA, PILHA-COM-ESVAZIAMENTO
end
```

Esta especificação textual inclui um exemplo de especialização múltipla (a classe PILHA-COM--ESVAZIAMENTO-LIMITADA), mecanismo não ilustrado nos diagramas anteriores, mas que faz parte integrante da linguagem também na sua versão diagramática.

6.5 Abstracção

Ilustramos o construto dual da especialização com a especificação de uma perspectiva[36] da classe PILHA: a classe PILHA-SUMIDOURA. Esta é obtida por abstracção de PILHA escondendo alguns eventos e atributos.

Para especificar a nova classe basta introduzir o respectivo diagrama matricial (ver Fig. 7): limitamo-nos a listar os atributos e os eventos que são preservados na abstracção[37].

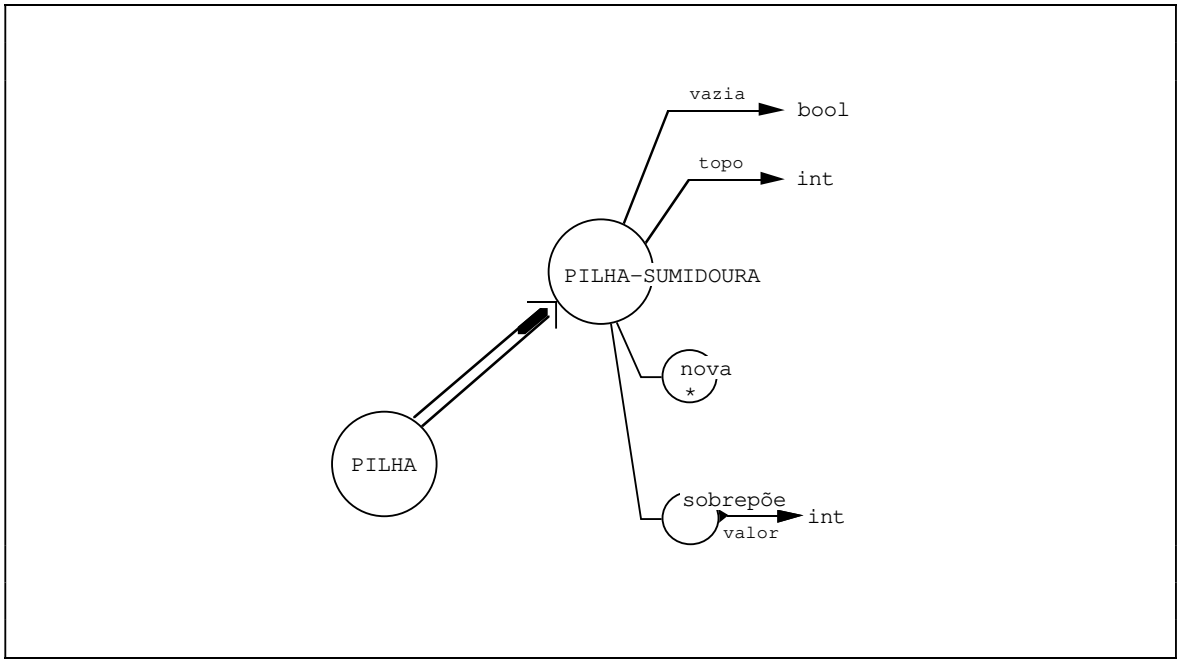

Fig. 7: diagrama matricial da classe PILHA-SUMIDOURA

O comportamento e os efeitos dos eventos sobre os atributos são derivados canonicamente do que foi especificado para a classe PILHA: não é preciso, pois, voltar a especificá-los.

6.6 Interacção

Já vimos como especificar cada classe. Dos mecanismos básicos da linguagem OBLOG falta descrever o construto que permite impor que as instâncias das classes interactuem umas com as outras.

---

36 *View*.

37 Operação muito simples com o editor por selecção dos atributos e eventos a manter no diagrama matricial de origem.

No contexto do exemplo considerado, é interessante examinar a especificação da interacção entre as instâncias da classe `BOMBA` e as da classe `PILHA`. E é possível proceder a essa especificação com um mínimo de informação sobre cada uma dessas classes.

Com efeito, suponhamos que da classe `BOMBA` só sabemos que dispõe (para além dos atributos `origem` e `destino` já introduzidos no diagrama do universo) de um evento `transfere` sem parâmetros.

A ideia é sincronizar cada evento `transfere` de uma bomba com um evento `sobrepõe` da pilha destino e um evento `retira` da pilha origem. Mais, o parâmetro deste evento `sobrepõe` deve ser o valor do atributo `topo` da pilha origem (imediatamente antes dos três eventos se darem). Esta interacção está especificada na Fig. 8.

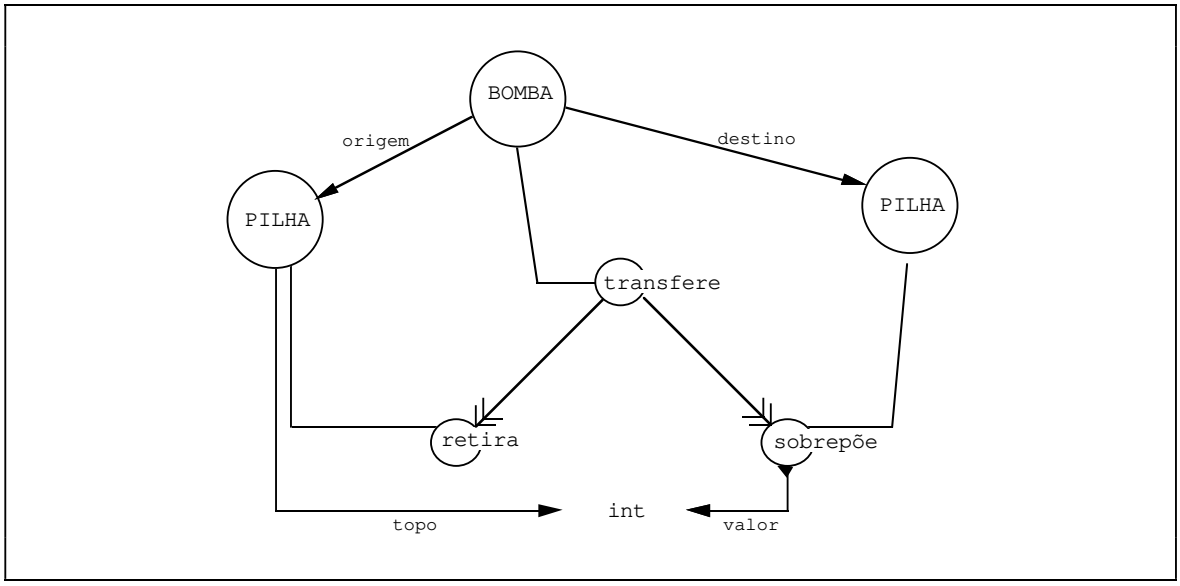

Fig. 8: diagrama de interacção (com atribuição de valores) entre `PILHA` e `BOMBA`

6.7 Reificação

Terminada a apresentação sumária dos construtos básicos da linguagem OBLOG, voltamos a nossa atenção para o mecanismo de reificação. Recorde-se que por reificação se entende uma relação especial entre duas comunidades de objectos: um objecto da primeira é implementado com recurso a alguns objectos da segunda.

Para ilustração, e ainda no mundo das pilhas, considere-se o problema de especificar a reificação de uma pilha recorrendo a um vector e a um apontador (problema clássico em informática): ver Fig. 9.

A cada evento da pilha (por exemplo, `sobrepõe`) deve corresponder uma transacção envolvendo os dois objectos do universo seguinte (alteração do vector e alteração do apontador). E cada atributo da pilha (por exemplo, `topo`) deve ser derivável a partir dos atributos dos dois objectos

com que construímos a implementação (conteúdo da célula do vector indicada pelo valor do apontador).

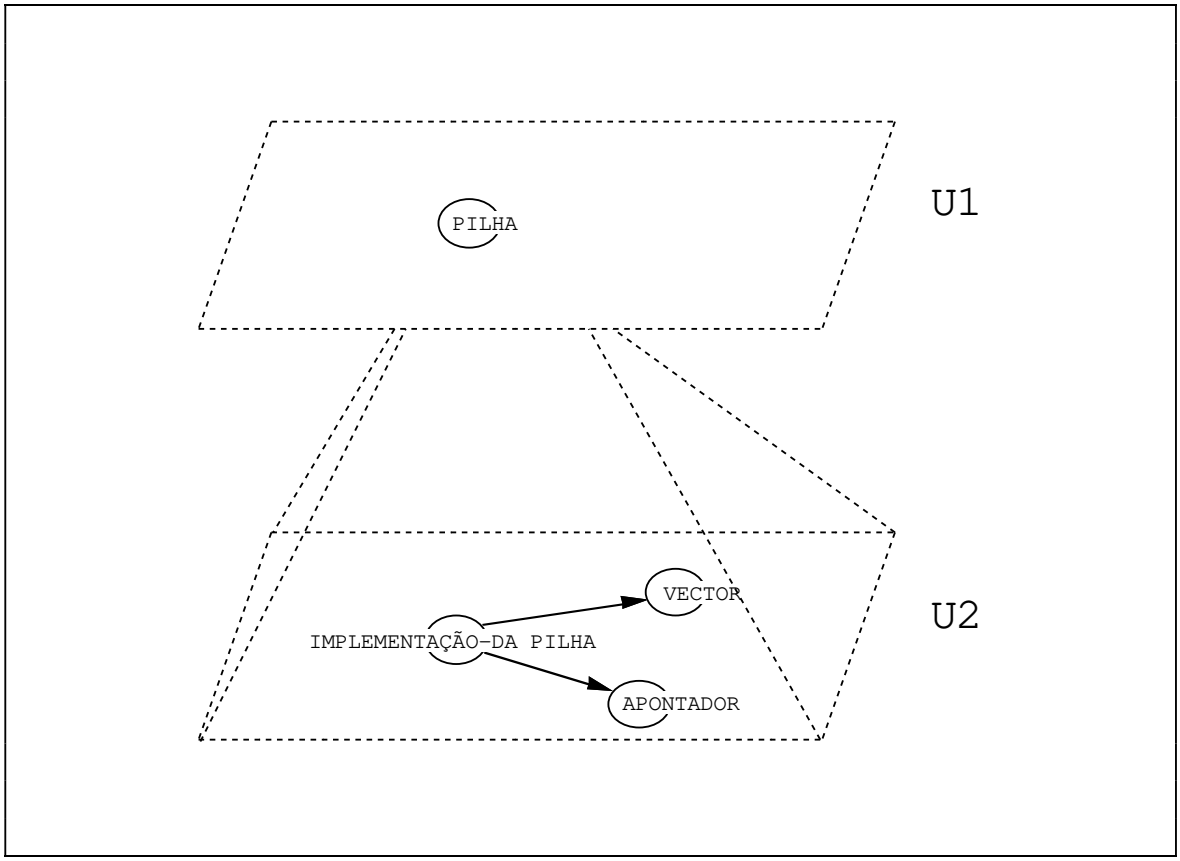

Fig. 9: O universo U1 reificado no universo U2

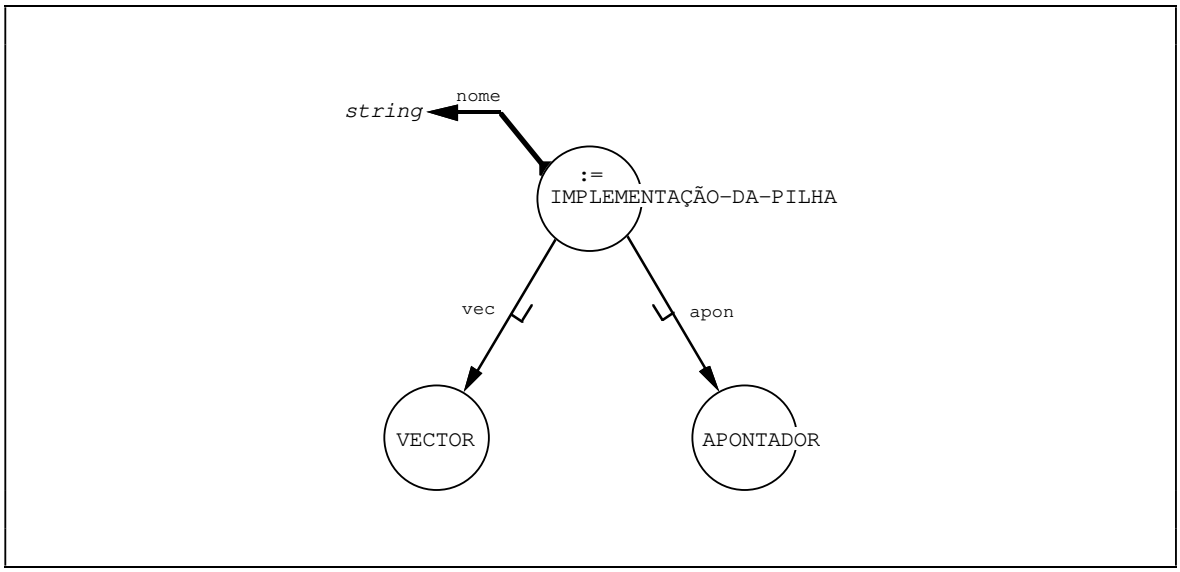

Fig. 10: diagrama do universo U2

Para estabelecer a reificação pretendida temos de começar por introduzir no diagrama do universo U2 (ver Fig. 10) as classes relevantes: IMPLEMENTAÇÃO-DA-PILHA, APONTADOR e

VECTOR. Os relacionamentos entre estas classes são explicados adiante, a propósito do diagrama matricial da classe IMPLEMENTAÇÃO-DA-PILHA.

Temos ainda de especificar cada uma destas classes, o que é feito nas Figs. 11 e 12 para as classes VECTOR e APONTADOR, respectivamente.

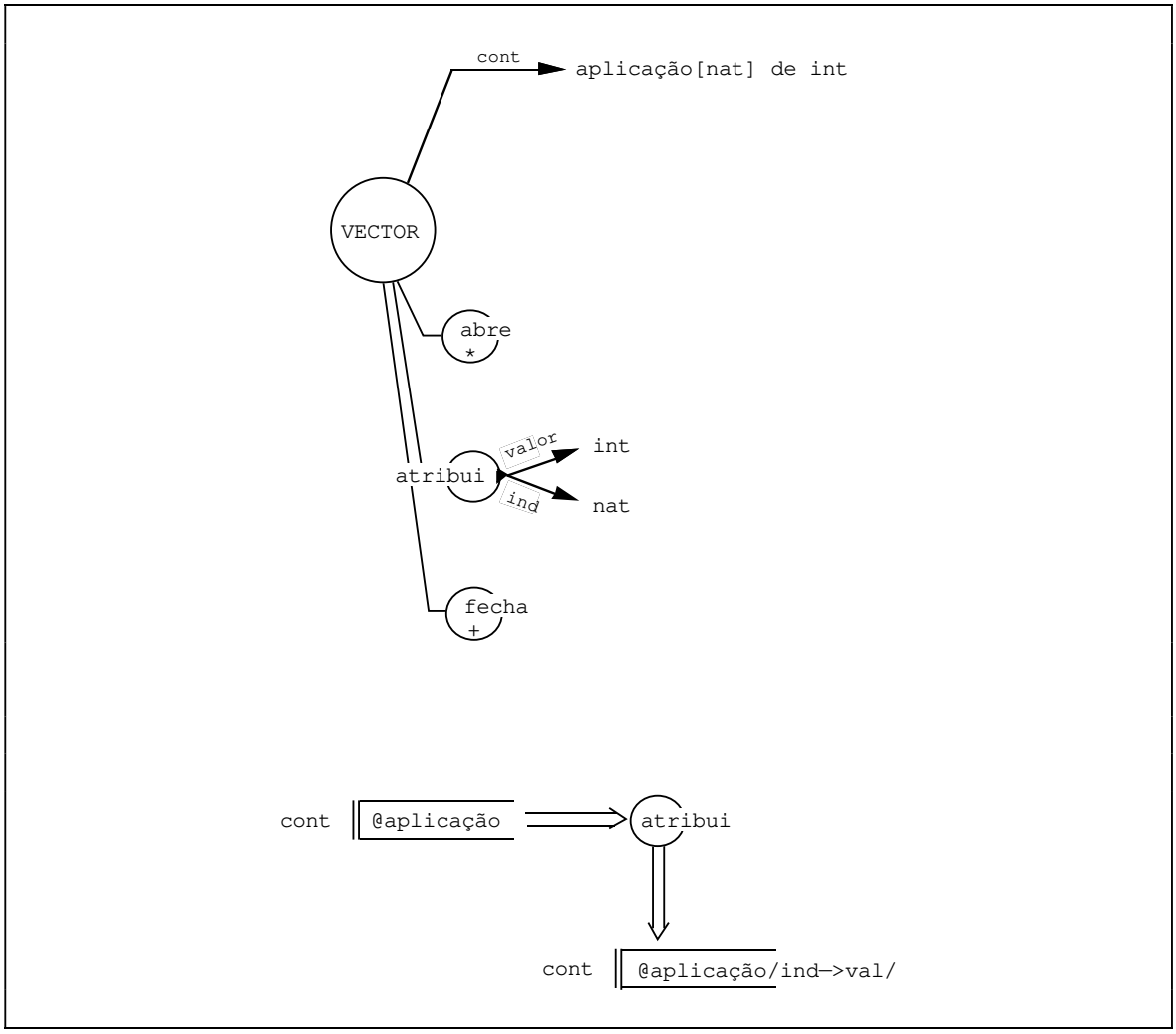

Fig. 11: diagramas da especificação da classe VECTOR

Estes diagramas não trazem qualquer novidade pelo que dispensam comentários, excepto no que se refere ao tipo de dados aplicação [nat] de int.

Cada elemento deste tipo é uma tabela que retorna um inteiro dado um natural[38]. Admitimos como disponível neste tipo de dados a operação de atribuição x/y->z/ que atribui o valor z à posição de índice y na tabela x.

[38] Pressupomos que só a um número finito de naturais se associa um inteiro não nulo.

Note-se que omitimos os respectivos diagramas de comportamento, pois, por omissão, isso significa que todos os eventos são sempre possíveis, sem quaisquer constrangimentos para além dos relativos ao nascimento e à morte.

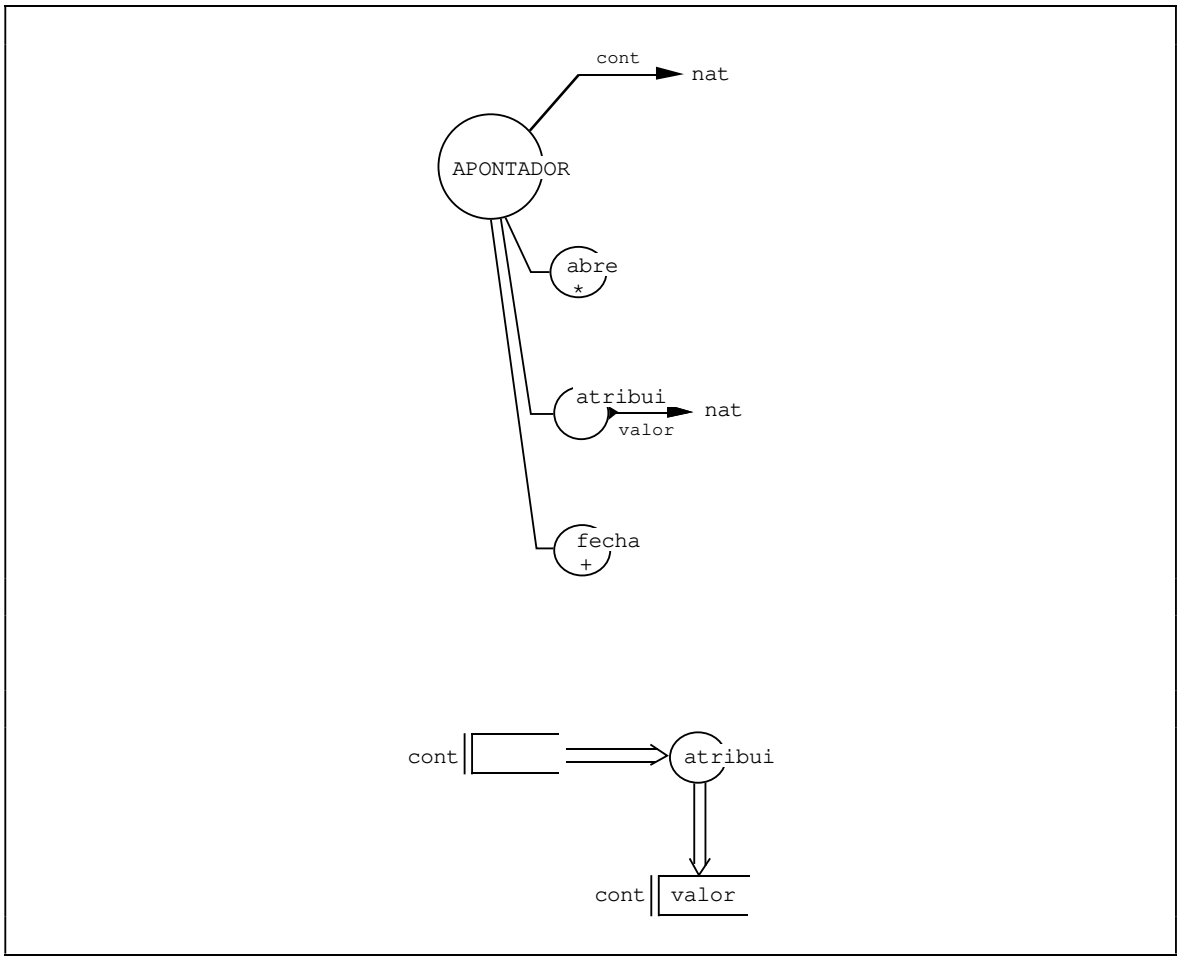

Fig. 12: diagramas da especificação da classe `APONTADOR`

Na Fig. 13 apresentamos o diagrama matricial da classe `IMPLEMENTAÇÃO-DA-PILHA`. Surgem aqui ilustrados alguns construtos da linguagem ainda não discutidos.

Nomeadamente, o diagrama introduz:

- Transacções (representadas por elipses): acções complexas (a descrever em diagramas semelhantes aos de comportamento: ver Fig. 14) que mantêm o carácter atómico essencial aos eventos (pois correspondem a eventos do universo de abstracção superior).

- Atributos derivados, com as expressões de derivação indicadas pelo símbolo :=[39] (por exemplo o atributo `vazia` calculável com a expressão `apont.cont=0`).

[39] Em alternativa às expressões (textuais) podem ser utilizados diagramas de atribuição.

- E, finalmente, incorporações (por exemplo, a incorporação do vector apontado pelo atributo `vec` na instância em questão da classe `IMPLEMENTAÇÃO-DA-PILHA`).

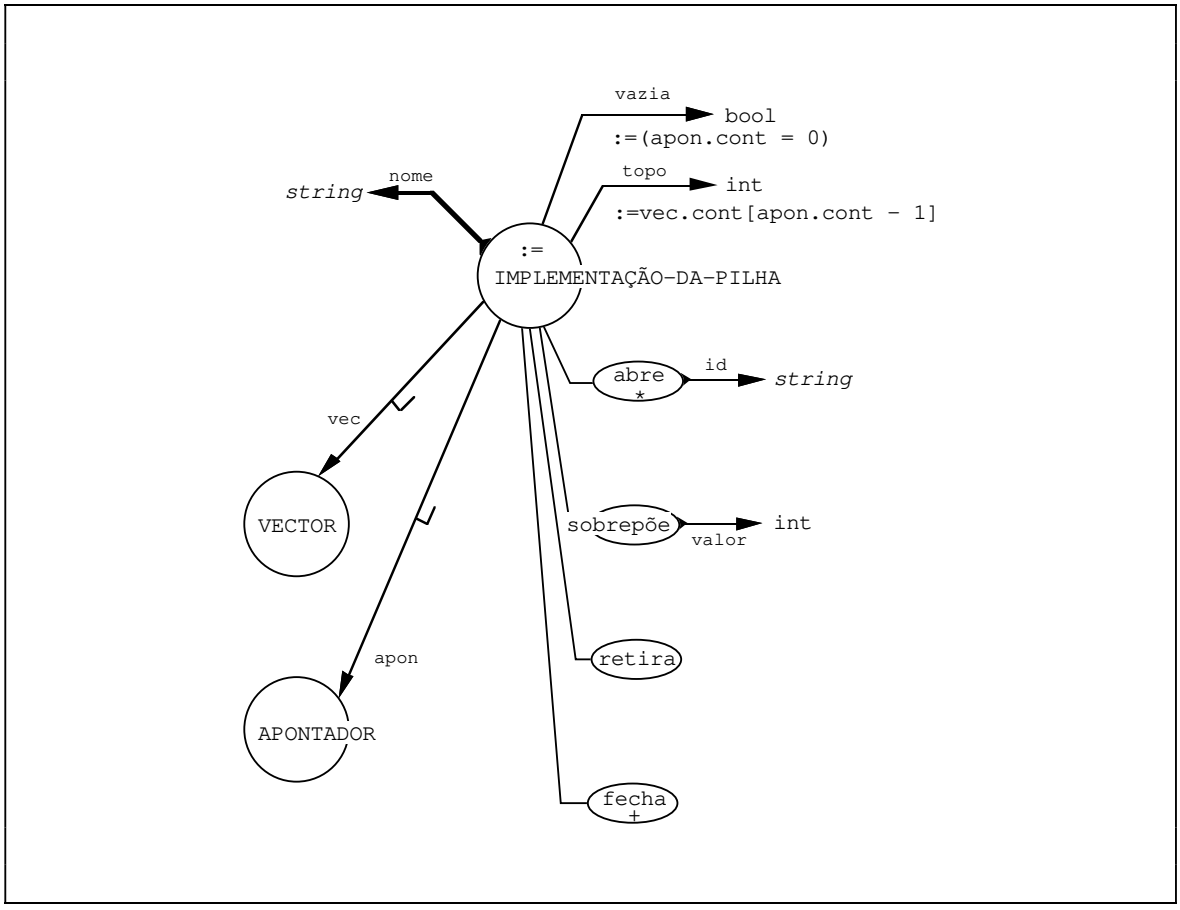

Fig. 13: diagrama matricial da classe `IMPLEMENTAÇÃO-DA-PILHA`

Resta ilustrar a especificação de uma transacção, o que é feito na Fig. 14 para o caso da transação `sobrepõe`.

Como seria de esperar, a execução de um evento `sobrepõe` numa pilha corresponde à execução transaccional de uma sequência de eventos de atribuição no vector e no apontador da reficação da pilha em causa. Claro que é necessário especificar os valores dos parâmetros de todos estes eventos, os quais estão intimamente relacionados uns com os outros. Tal é especificado no diagrama de atribuição de valores (também incluído na Fig. 14) associado ao diagrama transaccional em questão.

Este diagrama ilustra uma utilização já não trivial das capacidades da linguagem no campo da fixação de valores, tipicamente usadas para impor condições aos valores de atributos e dos parâmetros de eventos, como é o caso aqui. Note-se que os diagramas de atribuição de valores são suficientemente poderosos para se constituirem como uma autêntica linguagem diagramática de consulta[40].

[40] *Query language*, o que já era bem conhecido: uma boa linguagem de especificação tem de incluir construtos para definição de condições suficientes para uma linguagem de consulta.

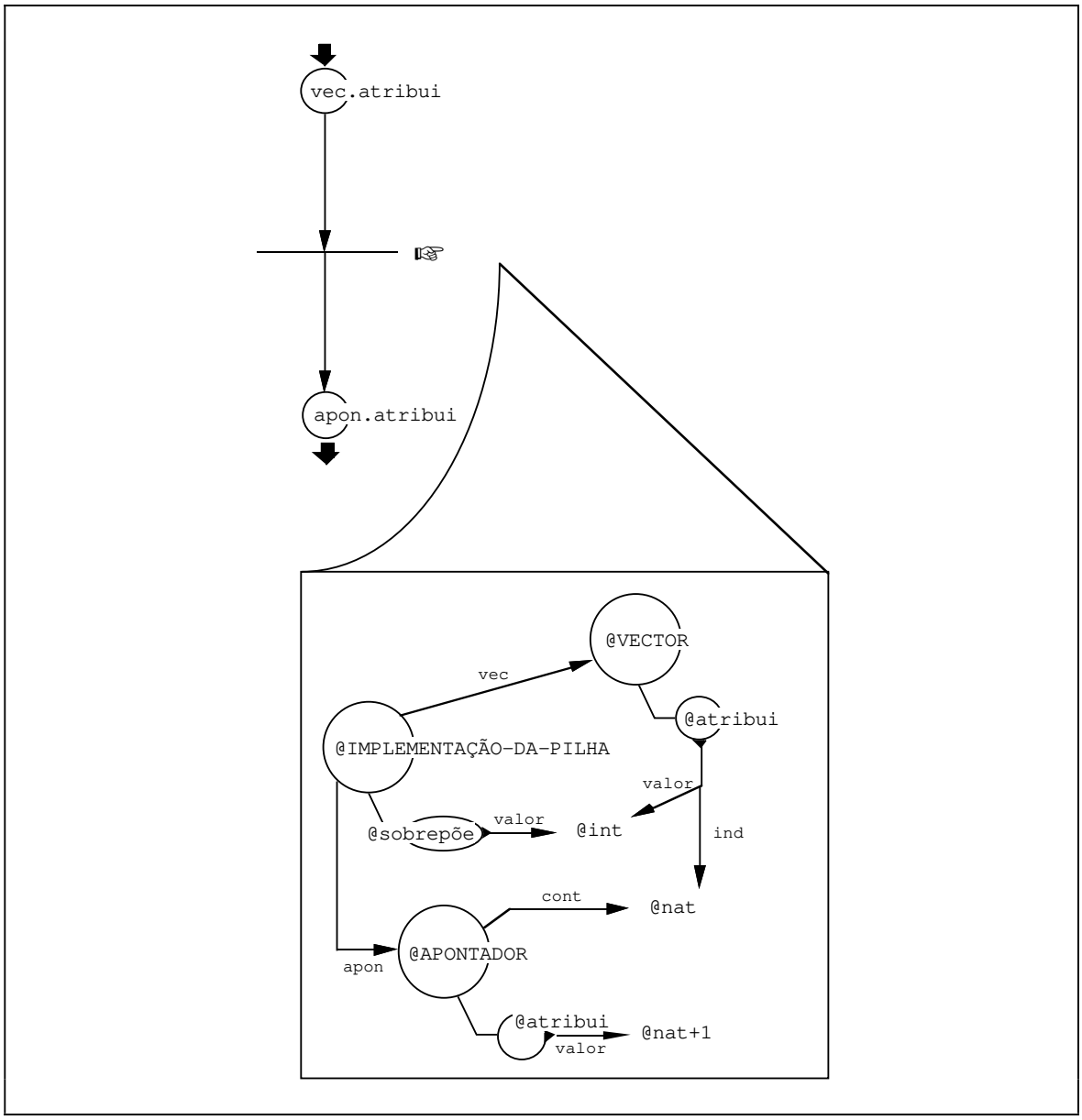

Fig. 14: diagrama transaccional de `sobrepõe` da classe `IMPLEMENTAÇÃO-DA-PILHA` com diagrama associado de atribuição de valores

7 Notas finais

Após esboçarmos uma estratégia para o desenvolvimento de aplicações informáticas baseada na ideia de utilizar uma linguagem de especificação diagramática e de muito alto nível em todas as fases do ciclo de vida do sistema, com geração semi-automática do código operacional, apresentámos com algum detalhe a referida linguagem, através de exemplos elementares que, no entanto, foram suficientes para ilustrar a maioria dos construtos. Pretendemos, assim, mostrar que foi de facto possível encontrar o justo equilíbrio entre a versatilidade e a expressiblidade indispensáveis a tal linguagem.

Claro que ficam muitas questões em aberto sobre a validade da estratégia (e da linguagem subjacente) que só a prática poderá vir a responder definitivamente:

(i) Será a linguagem de facto utilizável por analistas?

(ii) Até que ponto o aumento de produtividade e a diminuição de custos de manutenção poderão compensar o custo de adaptação a métodos de trabalho completamente novos?

(iii) Será possível gerar aplicações com desempenho aceitável?

(iv) Qual será o grau de abertura[41] das aplicações assim geradas?

Com a experiência prática adquirida até ao momento é possível, desde já, extrair algumas indicações úteis. Passamos, pois, a descrever essa experiência e as conclusões preliminares que pudémos retirar da mesma.

A abordagem OBLOG tem vindo a ser posta à prova em dois grandes projectos reais. O primeiro tem por objectivo o desenvolvimento da própria bancada OBLOG cuja funcionalidade inclui edição, teste, geração e memorização em repositório, com suporte ao trabalho cooperativo de grandes equipas[42] em grandes projectos[43]. O segundo visa o desenvolvimento de um pacote integrado de aplicações bancárias. Ambos estão hoje em dia em progresso[44], embora o segundo, por razões óbvias, esteja atrasado em relação ao primeiro.

O projecto da bancada OBLOG implicou a afectação de cerca de 30 homem-ano no período 90-92. O projecto do pacote bancário é bem mais pesado: estima-se um custo total de cerca de 200 homem-ano.

No desenvolvimento da bancada adoptou-se a estratégia proposta: é inteiramente especificada em OBLOG, nas suas sucessivas versões, e o código de 3ª geração[45] é gerado manualmente mas por aplicação cega de um algoritmo de conversão até que o *bootstrapping* seja concluído. Neste projecto têm trabalhado jovens informáticos com forte formação universitária nas tecnologias relevantes[46]. Logo, nada podemos concluir sobre a questão (i). Também não são óbvias quaisquer indicações sobre a questão (ii). Por outro lado, a *performance* da bancada assim gerada (manualmente mas já seguindo o algoritmo a automatizar) dá indicações de ser muito boa e facilmente controlável. Quanto à questão (iv), torna-se evidente que tudo depende das plataformas escolhidas como alvo da geração (e, portanto, independente da abordagem em si). Nem sempre é fácil a escolha face às constantes evoluções do mercado e das normas em vigor ou que se prevêem vir a vigorar no período futuro relevante. São críticas as escolhas relativas ao

41 No sentido de *open system*.
42 Dezenas de especificadores.
43 Conduzindo a milhões de linhas de código tradicional de 3ª geração.
44 Na ESDI, *Software House* do Grupo Espírito Santo, com consultoria científica do INESC.
45 Linguagem C.
46 Tipicamente recém-licenciados em informática ou em matemática.

sistema operativo, gestor de janelas, linguagem de codificação, sistema de bases de dados e sistema gestor de transacções. Em resumo, desta experiência podemos concluir da validade da abordagem quanto ao desempenho de aplicações interactivas (mesmo extremamente pesadas como é o caso de uma bancada). Resta saber se a extrapolação para sistemas transaccionais não levantará problemas inesperados. Também é interessante notar que o desenvolvimento da bancada demonstrou que é possível "casar" facilmente peças de *software* geradas a partir de OBLOG com outras (desenvolvidas nas mesmas plataformas alvo, claro).

Finalmente, este projecto respondeu completamente à questão que pusemos nas secções anteriores sobre se a linguagem seria suficientemente concisa. As estatísticas actuais definem o seguinte coeficiente: em média, *1 página de especificação OBLOG (um diagrama) conduz à geração de 10 páginas de código de 3ª geração (500 linhas)*.

O projecto de desenvolvimento do pacote integrado de aplicações bancárias está ainda numa fase inicial em que, no entanto, já se utiliza intensivamente a linguagem OBLOG como técnica de análise de requisitos de informação e de modelação conceptual. Infelizmente, dado que a bancada ainda não está disponível numa versão suficientemente estável, esta experiência de utilização da linguagem tem sido levada a cabo sobretudo com "lápis e papel" e o seu substituto "editor gráfico universal". Ficou demonstrado que uma linguagem formal, mesmo diagramática, obriga a recurso ao suporte computacional. Apesar disso, com uma equipa de uma dezena de analistas foi possível testar a abordagem nas etapas iniciais do ciclo de vida da aplicação. Dado o perfil dos analistas envolvidos[47], podemos retirar a ilação de que a resposta à pergunta (i) será positiva. Quanto à pergunta (ii) ainda não há elementos suficientes. Mas, será de sublinhar que o período de formação dos analistas é de pelo menos seis meses.

O trabalho de refinamento da abordagem continua em várias frentes, com base na experiência que vai sendo adquirida, nas lições retiradas do esforço de implementação da bancada, no acompanhamento de projectos similares e, obviamente, nos desenvolvimentos teóricos ainda em curso. A maior parte destes trabalhos teóricos é de momento dedicada ao afinamento da metodologia, ao estabelecimento de um cálculo lógico para verificação das especificações[48] e à consolidação da semântica operacional da linguagem.

8 Agradecimentos

Os autores expressam a sua gratidão a todos os colegas da aventura OBLOG e em especial a: Adelaide Marques, Amílcar Vicente, Ana Barros, Antónia Lopes, António Rito Silva, Filipe Santos, Jean-Marc Zeippen, João Gouveia, João Pedro Sousa, Joaquim Jorge Costa, José Caetano

47 Na sua maioria profissionais muito experientes de organização e análise de diversos bancos que participam no projecto.

48 De futuro, espera-se que os avanços no domínio da lógica de objectos permitam enriquecer a abordagem de modo a permitir o desenvolvimento de aplicações não algorítmicas, baseadas no conhecimento (*knowledge-based*).

Pereira, José Granado, José Luiz Fiadeiro, Margarida Gonçalves, Miguel Caldas, Miguel Torres Baptista, Paula Gouveia, Pedro Inácio, Pedro Resende, Peter Hartel e Philippe Dols.

9 Referências

[1] P.Butterworth, A.Otis e J.Stein, "The GemStone Object Database Management System". In: *Comm. ACM 34*, 1991, 64-77.

[2] R.Burstall e J.Goguen, "The Semantics of Clear, A Specification Language. In: *Proc. 1979 Copenhagen Winter School on Abstract Software Specification*, D.Bjorner (ed), Springer-Verlag, 1980, 292-332

[3] J.F.Costa, A.Sernadas e C.Sernadas, *OBL-87: Manual do Utilizador*, INESC, 1988.

[4] J.F.Costa, A.Sernadas e C.Sernadas, *OBL-89: Manual do Utilizador*, INESC, 1989.

[5] J.F.Costa, A.Sernadas e C.Sernadas, *OBLOG Users Manual*, INESC, 1989.

[6] J.F.Costa e A.Sernadas, "Categorial Framework for Process Interconnection". In: *The INESC Journal of Research and Development*, *em publicação*.

[7] J.F.Costa, A.Sernadas, C.Sernadas e H.-D.Ehrich, "Object Interaction". In: *Proceedings Mathematical Foundations of Computer Science*, Springer Verlag, *em publicação*.

[8] J.F.Costa, A.Sernadas e C.Sernadas, "Object Inheritance Beyond Subtyping", *INESC/DMIST*, May 1992, 27 páginas, *submetido para publicação*.

[9] J.F.Costa, A.Sernadas e C.Sernadas, "Inductive Objects", INESC/DMIST, June 1992, 27 páginas, *submetido para publicação*.

[10] J.F.Costa, A.Sernadas e C.Sernadas, *Formal Specifications*, INESC-DMIST, June 1992, 150 páginas.

[11] B.Cox, "There is a Silver Bullet", *Byte*, 1990, 209-218

[12] O.-J.Dahl, B.Myhrhaug e K.Nygaard, "SIMULA 67, Common Base Language", *Norwegian Computer Center*, 1967.

[13] O.Deux *et al*, "The O_2 System". In: *Comm. ACM 34*, 1991, 35-48.

[14] H.-D.Ehrich, A.Sernadas e C.Sernadas, "Abstract Object Types for Databases". In: *Advances in Object-Oriented Database Systems*, K.Dittrich (ed), Springer Verlag, 1988, 144-149.

[15] H.-D.Ehrich, A.Sernadas e C.Sernadas, "Objects, Object Types and Object Identity". In: *Categorical Methods in Computer Science with Aspects from Topology*, Springer Verlag, 1989, 142-156.

[16] H.-D.Ehrich, A.Sernadas e C.Sernadas, "From Data Types to Object Types". In: *Journal of Information Processing and Cybernetics*, EIK 26(1/2), 1990, 33-48.

[17] H.-D.Ehrich e A.Sernadas, "Algebraic Implementation of Objects over Objects". In: *Stepwise Refinement of Distributed Systems: Models, Formalisms, Correctness*, J.deBakker, W.-P.deRoever e G.Rozenberg (eds), Springer Verlag, 1990, 239-266.

[18] H.-D.Ehrich, J.Goguen e A.Sernadas, "A Categorial Theory of Objects as Observed Processes". In: *Foundations of Object-oriented Languages*, J.deBakker, W.-P.deRoever and G.Rozenberg (eds), Springer Verlag, 1991, 203-228.

[19] H.-D.Ehrich e A.Sernadas, "Object Concepts and Constructions". In: *Proc. of the IS-CORE Workshop'91*, G.Saake e A.Sernadas (eds), Informatik-Berichte 91-03, Tech. Univ. Braunschweig, 1991, 1-24.

[20] H.-D.Ehrich, M.Gogolla e A.Sernadas, "Objects and their Specification". In: *Recent Trends in Data Type Specification: 8th Workshop on Specification of Abstract Data Types - Selected Papers*, M.Bidoit e C.Choppy (eds), Springer Verlag, *em publicação*.

[21] H.-D.Ehrich, G.Saake e A.Sernadas, "Concepts of Object Orientation". In: *Proceedings Workshop Informationssysteme und Künstliche Intelligenz*, R.Studer (ed), IFB 303, Springer Verlag 1992, 1-19.

[22] H.Ehrig e B.Mahr, *Fundamentals of Algebraic Specification 1*, Springer-Verlag, 1985.

[23] J.Fiadeiro e A.Sernadas, "Logics of Modal Terms for Systems Specification". In: *Journal of Logic and Computation*, 1(2), 1990, 187-227.

[24] J.Fiadeiro e T.Maibaum, "Describing, Structuring and Implementing Objects". In: *Foundations of Object-oriented Languages*, J.deBakker, W.-P.deRoever e G.Rozenberg (eds), Springer Verlag, 1991, 274-310.

[25] J.Fiadeiro, C.Sernadas, T.Maibaum e G.Saake, "Proof-theoretic Semantics of Object-oriented Specification Constructs". In: R.Meersman, W.Kent e S.Khosla (eds), *Object Oriented Databases: Analysis, Design and Construction*, North Holland, 1991, 243-284.

[26] J.Fiadeiro e T.Maibaum, "Temporal Theories as Modularisation Units for Concurrent System Specification", *Formal Aspects of Computing*, 4(3), 1992, 239-272.

[27] J.Fiadeiro, J.F.Costa, A.Sernadas e T.Maibaum, "Process Semantics of Temporal Logic Specification". In: *Recent Trends in Data Type Specification: 8th Workshop on Specification of Abstract Data Types - Selected Papers*, M.Bidoit e C.Choppy (eds), Springer Verlag, *em publicação*.

[28] J.Fiadeiro, C.Sernadas, T.Maibaum e A.Sernadas, "Describing and Structuring Objects for Conceptual Schema Development". In: *Conceptual Modelling, Databases and CASE: An Integrated View of Information Systems Development*, P.Loucopoulos e R.Zicari (eds), John Wiley, *em publicação*.

[29] D.Fishman *et al*, "IRIS: An Object-Oriented Database Management System". In: *ACM Trans. Off. Inf. Sys. 5*, 1987.

[30] A.Goldberg e D.Robson, *Smalltalk 80: The Language and its Implementation*, Addison-Wesley, New York, 1983.

[31] P.Gouveia, *Tableaux Para Raciocínio Local Sobre Objectos*, Tese de Mestrado, Setembro IST, 1992.

[32] J.Granado,"Financial Business and Information Systems Evolution", In: *Proc. of INDC'92*, Helsínquia, Março de 1992.

[33] M.Heitz, "HOOD: A Hierarchical Object-Oriented Design Method". In: *Proc. of the 3rd German Ada Users Congress*, Munich, 1988, 12-1—12-9.

[34] N.Kerth, "MOOD: A Methodology for Structured Object-Oriented Design". In: *OOPSALA'88*, San Diego, 1988.

[35] W.Kim *et al*, "Features of the ORION Object-Oriented DBMS". In W.Kim e E.H.Lochovsky (eds), *Object-Oriented Concepts, Databases, and Applications*, Addison-Wesley, 1988.

[36] B.Meyer, *Object-Oriented Software Construction*, Prentice-Hall, Englewood Cliffs, 1988.

[37] Z.Manna e A.Pnueli, "Verification of Concurrent Programs: The Temporal Framework". In: *The Correctness Problem in Computer Science*, R.Boyer e J.Moore (eds), Academic Press, London, 215-273.

[38] E.Seidewitz e M.Stark, *General Object-Oriented Software Development*, Document No. SEL-86-002, NASA Goddard Space Flight Center, Greenbelt, Maryland, 1986

[39] A.Sernadas, C.Sernadas e H.-D. Ehrich, "Object-oriented Specification of Databases: An Algebraic Approach", *Proc. 13th Int. Conf. on Very Large Databases VLDB'87*, P.Hammerslay (ed), Morgan-Kaufmann, 1987, 207-116.

[40] A.Sernadas, J.Fiadeiro, C.Sernadas e H.-D.Ehrich, "Abstract Object Types: A Temporal Perspective". In: *Temporal Logic in Specification*, B.Banieqbal, H.Barringer e A.Pnueli (eds), Springer Verlag, 1989, 324-350.

[41] A.Sernadas, J.Fiadeiro, C.Sernadas e H.-D. Ehrich, "The Basic Building Blocks of Information Systems", *Information System Concepts: An In-depth Analysis*, E.Falkenberg e P.Lindgreen (eds), North Holland, 1989, 225-246.

[42] A.Sernadas, H.-D.Ehrich e J.F.Costa, "From Processes to Objects". In: *The INESC Journal of Research and Development*, 1(1), 1990, 7-27.

[43] A.Sernadas e H.-D.Ehrich, "What is an Object, After All?". In: *Object Oriented Databases: Analysis, Design and Construction*, R.Meersman, W.Kent e S.Khosla (eds), North Holland, 1991, 39-69.

[44] A.Sernadas, C.Sernadas, P.Gouveia, P.Resende e J.Gouveia, *OBLOG — An Informal Introduction*, INESC, Janeiro de 1991.

[45] A.Sernadas, "Tendências da Engenharia Informática". In: *TÉCNICA*, Número Especial de 1991 — 80 Anos do IST e da AEIST, Março de 1992, 77-83.

[46] A.Sernadas, C.Sernadas e J.F.Costa, "Object Specification Logic", INESC/DMIST, June 1992, 25 páginas, *submetido para publicação*.

[47] C.Sernadas, P.Gouveia, L.Silva e A.Lopes, "Objects as Structuring Units for Incorporating Dynamics in Deductive Conceptual Modeling". In: *The Deductive Approach to Information Systems and Databases*, Universitat Politècnica de Catalunya, 1990, 93-110.

[48] C.Sernadas e J.Fiadeiro, "Towards Object-Oriented Conceptual Modelling". In: *Data and Knowledge Engineering*, 6(6), 1991, 479-508.

[49] C.Sernadas, P.Gouveia e A.Sernadas, "Refinement: Layered Definition of Conceptual Schemata". In: *Information System Concepts*, E.Falkenberg, C.Rolland e E.El-Sayed (ed), North-Holland, 1992, 19-51.

[50] C.Sernadas, P.Gouveia e A.Lopes, "Gentzen-type System for Verification in Conceptual Modeling". In: *Foundations of Models and Languages for Data and Objects*, J.Goers, A.Hever e G.Saake (eds), 1991, 215-228.

[51] C.Sernadas, P.Resende, P.Gouveia e A.Sernadas, "In-the-large Object-oriented Design of Information Systems". In: *The Object-Oriented Approach in Information Systems*, F.Van Assche, B.Moulin e C.Rolland (eds), North Holland, 1991, 209-232.

[52] C.Sernadas, P.Gouveia, J.Gouveia, A.Sernadas e P.Resende, "The Reification Dimension in Object-oriented Data Base Design". In: *Specification of Data Base Systems*, D.Harper e M.Norrie (eds), Springer Verlag, 1992, 275-299.

[53] J.P.Sousa, C.Sernadas e A.Sernadas, "An Object-oriented Specification Tool for Graphical Interfaces". In: *Computers and Graphics*, 14(1), 1990, 29-40.

[54] B.Stroustrup, *The C++ Programming Language*, Addison-Wesley, Reading, Mass., 1986.

[55] P.Wegner, "Learning the language". In: *Byte* 14, 1989, 245-253.