Programação Dirigida por Contratos Adaptativos — Especificação de Propriedades Comportamentais Dinâmicas

Italo S. Vega

***Abstract*—The design of adaptive automata involves the development of rules that, when applied, can transform the topology of states and transitions during a computation. Adaptive specifications were introduced by Vega as a way to represent the different transformations in the automaton state space. In this article, we present a technique that helps in designing adaptive transitions that interconnect the states in the adaptive state space of an adaptive automaton.**

***Index Terms*—adaptive specifications, adaptive contracts, object-oriented modeling, software design, state machine, UML.**

I. Introdução

A LÓGICA proposta por Hoare [1] introduziu uma forma de raciocínio sobre o correto funcionamento de programas de computador na forma de triplas $\{P\}C\{Q\}$. Caso a pré-condição P seja verdadeira, estabelece-se a pós-condição Q pela execução do comando P. Posteriormente, a lógica de Hoare foi estendida para lidar com ponteiros para estruturas de dados por Reynolds [2]. A *lógica da separação* lida com porções de memória apenas, ao invés do espaço global de estados do sistema. Mas, como seria isso útil no processo de desenvolvimento de software? Meyer propôs uma técnica de projeto de software baseada na noção de contrato — os componentes de um sistema respeitam uma relação de obrigação-benefício durante uma particular colaboração [3], [4]. Componentes de software, tipicamente implementados como objetos no paradigma de objetos, executam comportamentos computacionais [5]chamados de métodos. Estes são concebidos, portanto, a partir do que se deseja que o componente faça em um determinado contexto computacional. *Contratos* formalizam as propriedades que devem ser exibidas por comportamentos ao serem executados: quando violados, acusam problemas de execução comportamental.

Por outro lado, Bock propõe uma representação das ocorrências de estados de um comportamento usando *estados-objeto*. Esta técnica trata estados como se fossem classes dinamicamente instanciadas e destruídas na forma de objetos. Sob tal perspectiva de modelagem, estados-objeto possuem atributos estruturais e podem ser ligados com outros estados-objeto. Como consequência, obtém-se uma representação de comportamentos mais expressiva durante a etapa de modelagem baseada em estados.

Especificações adaptativas foram introduzidas por Vega com a intenção de organizar a descrição de modelos comportamentais de autômatos adaptativos [6]. Representações parciais de máquinas de estados foram sugeridas com base em autômatos adaptativos. Entretanto, a relação entre as diferentes configurações de um particular autômato adaptativo ficaram implícitas na especificação. Este artigo propõe uma maneira de se lidar com esta questão, combinando as técnicas de programação por contratos e representação de estados por meio de estados-objeto. *Contratos adaptativos*, introduzidos neste artigo, formalizam as propriedades dinâmicas das operações modeladas por autômatos adaptativos vistos como objetos. Elas formalizam a conexão entre as diferentes configurações de um particular autômato adaptativo ao longo da sua existência na dimensão temporal.

Na Seção II apresentam-se os conceitos de estado-objeto e de contratos de operação. Em seguida, na Seção III conecta-se a noção de adaptatividade com objeto e estado-objeto. A Seção IV ilustra a utilização da técnica de programação baseada em contratos adaptativos. Os efeitos da programação dirigida por contratos adaptativos são discutidos também naquela Seção. Finalmente, na Seção V apresentam-se projetos que pretendem utilizar os contratos adaptativos durante a sua realização.

II. Estado-Objeto e Contrato de Operação

Estados-objeto e contratos de operações, juntamente com a noção de especificação adaptativa, formam a base das ideias que originaram os contratos adaptativos.

A. Estado-Objeto

Em uma série de artigos a respeito de modelagem de comportamentos no contexto da orientação a objetos, Bock discute três maneiras de se modelar estados: (i) estado-comportamento, (ii) estado-condição e (iii) estado-objeto [7]. Utiliza-se a primeiras delas em conjunção com transições de estado para especificar reação a eventos. Uma máquina pode apresentar diversas reações a um determinado tipo de evento, dependendo do seu estado corrente. Esta é a visão tradicional na ciência da computação e, em particular, no estudo de autômatos finitos. A

Italo S. Vega trabalha no Departamento de Computação da Faculdade de Ciências Exatas e Tecnologia da Pontifícia Universidade Católica de São Paulo, São Paulo, capital, Brasil e-mail: `italo pucsp.br`.

Manuscrito recebido no dia xx/xx/2016.

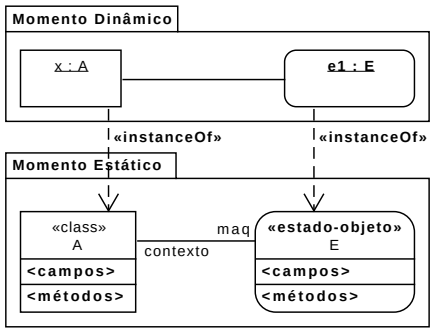

Figure 1: Diagrama UML de um estado-objeto de acordo com Bock

segunda maneira interpreta um estado como restrições que determinados valores devem obedecer em um instante de tempo. Estados-condição serviram de base, por exemplo, para a linguagem TLA+ de Lamport, que suporta a especificação de comportamentos com fórmulas da *Temporal Logic of Actions* [8]. Diferentes condições situacionais ao longo da execução de um comportamento originam diferentes estados computacionais.

Na técnica de estado visto como objeto, Bock sugere que estados sejam percebidos como instâncias de classes — construídos e destruídos ao longo do momento dinâmico de execução do comportamento [9]. Objetos tanto podem ser usados para capturar a perspectiva da primeira forma (estado-comportamento), quanto da forma estado-condição. Isto porque eles são especificados por classes constituídas por campos e métodos [5]. Na Fig. 1 ilustra-se a ideia por meio de um diagrama UML de estrutura. No momento estático, existe a classe A associada a um estado-classe E contendo eventuais campos e métodos que especificam estados-objeto desta classe. Em um momento dinâmico, instâncias da classe A (como o objeto x) encontram-se ligadas a objetos do estado-classe M (como o objeto e1). Condições e ocorrências comportamentais podem ser modelados por valores de campos e ocorrências de execução de métodos invocadas pelo estado-objeto e1.

Ainda na Fig. 1, os papéis de objetos maq e contexto suportam a combinação de campos e métodos que se referem a variáveis de estado com campos que se referem a outras estruturas de informação. Se x for um campo de A, então ele pode ser manipulado no contexto E por meio da expressão contexto.x. De maneira similar, se y for um campo de E, ele pode ser acessado por meio da expressão maq.y no contexto A. Estas expressões serão posteriormente utilizadas nos contratos adaptativos. Na próxima seção destacam-se os elementos centrais da programação por contratos de Meyer.

B. Contrato de Operação

A noção de comportamento em sistemas de software remete a procedimentos e métodos, relacionados ao ciclo chamada-invocação-execução conforme apresentado na

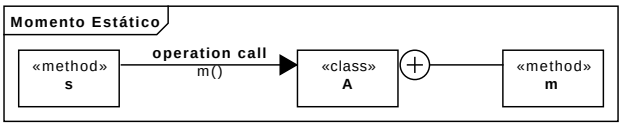

Figure 2: Chamada síncrona da operação **m**

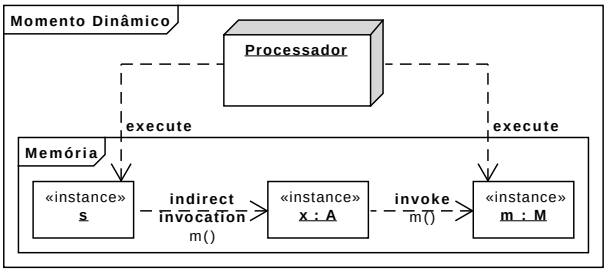

Figure 3: Invocação indireta e execução do comportamento m

especificação UML [5]. No momento estático, programa-se uma chamada de comportamento que, no caso do paradigma de objetos, traduz-se como uma *chamada de operação*. Na Fig. 2, no contexto do método s programou-se uma chamada síncrona de operação m() por meio da classe A. Esta, por sua vez, contém (por aninhamento) o método m.

No momento dinâmico, chamadas de operações originam *mensagens* (síncronas) e *eventos* os quais produzem *invocações indiretas* de métodos. Indiretas, pois o comportamento a ser executado depende do objeto que recebeu a mensagem. Considere-se um objeto s programado para enviar a mensagem chamando a operação m sobre o objeto x (Fig. 3). Esta invocação indireta de comportamento é resolvida pelo objeto x — invocação do comportamento m. Como resultado, o processador passa a *executar* este comportamento. A interação entre os objetos s e x é conhecida por *colaboração*.

Na utilização da técnica de programação por contratos explicita-se a condição a ser estabelecida para que um comportamento seja invocado; a [pré-condição]. Também deve-se explicitar a condição que deverá ser estabelecida pela execução do comportamento; a [pós-condição]. Na proposta de Meyer, o processador verifica a pré-condição no contexto de envio da mensagem, enquanto a pós-condição é verificada no final da execução do comportamento. No caso da Fig. 3, a pré-condição deve ser verdadeira no contexto de envio da mensagem m de s para x. A pós-condição deve ser verificada pelo processador no final da execução do comportamento m.

Na Fig. 4, representam-se os elementos centrais de um contrato comportamental de acordo com os estados-objeto de Bock. No momento estático, especifica-se o contrato de execução de comportamentos com base no estado-classe E como segue. Na ocorrência de um evento no qual se verifica

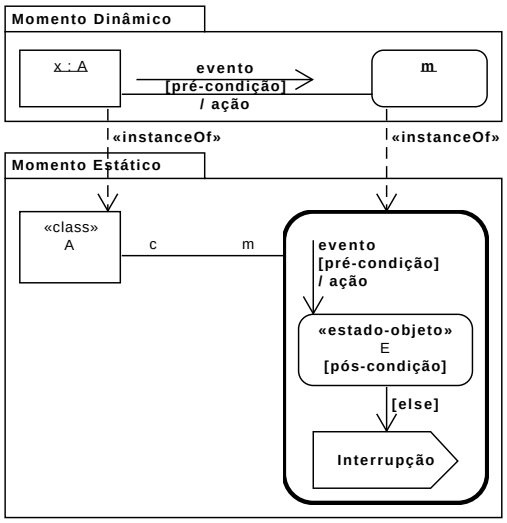

Figure 4: Representação de contratos em estados-objeto

a [pré-condição], a invocação indireta da **ação** provoca a sua execução. (Uma ação é uma espécie de método cuja execução não pode ser interrompida uma vez iniciada). Ao término da execução da ação, a [pós-condição] deve estar satisfeita. Caso contrário, interrompe-se o processo computacional devido à violação do contrato de execução da ação em um estado-objeto da classe E.

A ideia de estados-objeto suporta análises mais expressivas para a programação de um sistema computacional, enquanto a programação por contratos introduz a possibilidade de se observar violações de contratos — indícios de falhas computacionais. A combinação destas duas técnicas será explorado na concepção de contratos adaptativos a seguir.

III. Adaptatividade e Contratos Adaptativos

No caso da tecnologia adaptativa, a camada adaptativa proporciona um mecanismo para modificar a topologia de um dispositivo subjacente. Caso este seja um autômato, as modificações topológicas afetam os movimentos por ele realizados. Sabe-se que as ações adaptativas antes e depois, se realizadas, contribuem para alterar a topologia de um autômato adaptativo em razão de algum propósito de modelagem. Modelados como execuções comportamentais, os movimentos de um autômato podem ser projetados com o auxílio dos contratos adaptativos e estados-objeto como será mostrado a seguir.

A. Autômatos e Mecanismo Adaptativo

Seguindo a formulação de Neto e Bravo [10], um autômato de estados finito é composto por um conjunto de estados Q, um alfabeto finito e não vazio de símbolos Σ, uma função de transição δ, um estado inicial $q_0 \in Q$ e um conjunto de estados finais $F \subseteq Q$. Em particular, a função δ mapeia pares ordenados que especificam o estado corrente $q \in Q$ e a entrada corrente $\alpha \in \Sigma$ em um novo estado $q \in Q$: $(p, \alpha) \mapsto q$.

Um autômato de pilha estruturado possui uma estrutura semelhante àquela do autômato de estados finitos, além de um alfabeto de pilha Γ e elementos que suportam transições externas, responsáveis pelo esquema de chamada e de retorno. As transições de chamada possuem a forma $(p, \epsilon) \mapsto (\downarrow\ q_b, q_x)$. Interpreta-se p como o estado de chamada de uma submáquina com estado inicial q_x. No final da sua execução, a submáquina retorna o controle para o estado q_b. Cabe ao autômato de pilha preservar o estado q_b na sua memória durante a ocorrência da transição. As transições de retorno possuem a forma $(p, \epsilon) \mapsto (\uparrow\ q_r, q_r)$, para algum p no conjunto de estados finais da submáquina corrente. O estado q_r representa qualquer um dos estados anteriormente empilhados na memória do autômato pela submáquina chamada a partir daquela corrente. Esse estado é removido da memória e a submáquina chamadora continua a sua execução a partir de q_r.

Finalmente, Neto e Bravo referem-se ao mecanismo adaptativo. Ações adaptativas modificam o comportamento do autômato adaptativo, alterando o conjunto de regras que o definem. Define-se o mecanismo adaptativo pela vinculação de ações adaptativas anteriores e posteriores às regras não-adaptativas que definem as transições do autômato adaptativo. As transições adaptativas assumem a forma $(p, \alpha) \mapsto q[\mathcal{B} \bullet \mathcal{A}]$. Na ocorrência de α, estando o autômato no estado p, executa-se a ação adaptativa $\mathcal{B}$, realiza-se a transição para q e executa-se a ação adaptativa $\mathcal{A}$, nesta ordem.

Na técnica de programação proposta neste artigo, estados-objeto, chamadas de operações e contratos adaptativos são arquétipos dos elementos formais utilizados para a definição de autômatos finitos, autômatos de pilha e autômatos adaptativos. Assim, estados-objeto possuem alguma relação com os estados em Q. De modo semelhante relacionam-se chamadas de operações com chamadas e retornos de submáquinas e as ações adaptativas $\mathcal{B}$ e $\mathcal{A}$ com os métodos dos contratos adaptativos. A próxima seção define o elemento central da técnica: os contratos adaptativos.

B. Contratos Adaptativos

Partindo-se do modelo de contratos representado na Fig. 4, se o estado-objeto m for interpretado como um mecanismo adaptativo, então ele assumiria a responsabilidade de modificar o comportamento de x: ao longo do seu ciclo de vida, x teria o seu comportamento dinamicamente modificado por m. Em um contrato adaptativo, operações pré-adaptativas e pós-adaptativas (inspiradas nas funções adaptativas do mecanismo adaptativo) são vinculadas às pré- e pós-condições de um contrato: ao serem chamadas, provocam a modificação do comportamento dos objetos da classe A.

Os elementos estruturais de um contrato adaptativo são especificados conforme ilustrado na Fig. 5. Na ocorrência

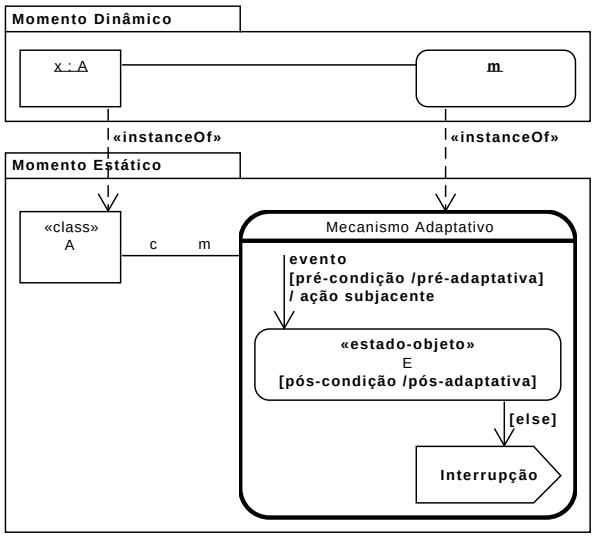

Figure 5: Elementos estruturais de um contrato adaptativo

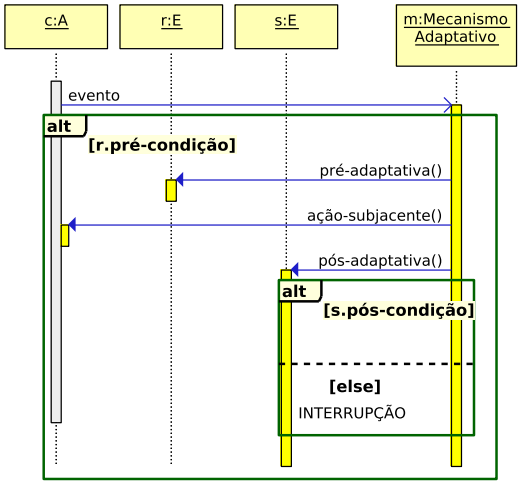

Figure 6: Vista do momento dinâmico de um contrato adaptativo

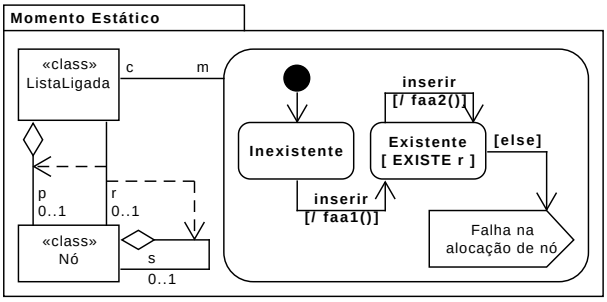

Figure 7: Contrato adaptativo da operação **inserir** em uma lista-ligada

de um evento do tipo **evento**, o estado-objeto corrente do mecanismo adaptativo m verifica a [pré-condição]. Se ela for verdadeira, ele dispara a transição, provocando a chamada da operação **pré-adaptativa**. Em seguida, ocorre a chamada da operação ação-subjacente envolvendo o contexto-objeto c. Finalmente, ocorre a chamada da operação **pós-adaptativa** e o estado-objeto seguinte do mecanismo adaptativo verifica a [pós-condição]. Se ela for falsa, ele emite um sinal de interrupção do comportamento computacional do objeto c.

Em uma vista de diagrama de sequências de mensagens, o momento dinâmico de um contrato adaptativo apresenta-se como na Fig. 6. O tratamento do **evento** é encaminhado para o mecanismo m. Este objeto, em colaboração com o estado-objeto corrente r, verifica a [pré-condição]. Sendo verdadeira, inicia-se o comportamento de reação ao evento propriamente dita. O mecanismo m envia uma mensagem para r chamando a operação **pré-adaptativa**. Em seguida, ocorre a chamada da operação **ação-subjacente**. Por último, m chama a operação **pós-adaptativa** e, juntamente com o estado-objeto seguinte s, verifica a [pós-condição]. Ocorrerá interrupção do comportamento de c se tal condição for falsa.

Contratos adaptativos especificam propriedades relacionadas com adaptatividade estrutural e comportamental. Uma *adaptatividade estrutural* é uma mudança nos campos de um objeto. Uma *adaptatividade comportamental* é uma mudança nos métodos de uma classe. Em ambos os casos, a adaptatividade decorre da chamada das operações **pré-adaptativa** e **pós-adaptativa**. Esta observação sugere os seguintes modelos de computação:

- computação inspirada em procedimento — o comportamento computacional é estabelecido pela ordem dos passos de um procedimento;
- computação inspirada em estrutura adaptativa — o comportamento computacional depende da evolução dinâmica dos campos de um objeto;
- computação inspirada em comportamento adaptativo — o comportamento computacional depende da evolução dinâmica dos campos e dos métodos de um objeto.

Computações inspiradas em procedimento são tradicionalmente estudadas na literatura ([11], [12]). Computações inspiradas em comportamento adaptativo são mais facilmente suportados pelos mecanismos de programação funcional ([13], [14]). O exemplo da próxima seção ilustra a noção de contratos adaptativos e de um modelo inspirado em estrutura adaptativa.

IV. Exemplo Ilustrativo

Uma estrutura de dados tradicional para a implementação do tipo de dado *lista* é a *lista ligada simples* [12]. Trata-se de um modelo de implementação cuja estrutura se constrói via alocação dinâmica de memória. Neste exemplo, enfatiza-se apenas a evolução da estrutura ao longo de uma série de eventos de inserção: ignora-se a perspectiva lógica de armazenagem dos valores.

A especificação estática (Fig. 7) introduz a classe ListaLigada composta por um objeto opcional p da classe Nó. O

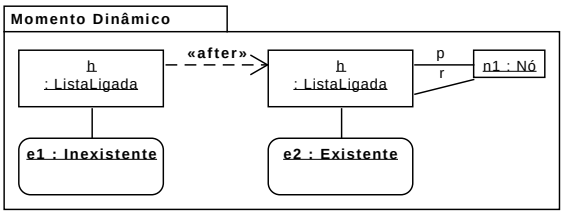

Figure 8: Estrutura de objetos depois da primeira transição de estado (exemplo-20)

campo r é uma referência para o objeto denominado p (no contexto de um objeto da classe ListaLigada) ou para o objeto s (no contexto de um objeto da classe Nó). A linha tracejada indica uma dependência existencial no momento dinâmico: a existência de r depende da existência de p ou de s. Na especificação da máquina de estados m observa-se que o estado inicial de um objeto ListaLigada é **Inexistente**, indicando que não existe o objeto p. De acordo com o contrato adaptativo, diante de uma ocorrência de evento do tipo inserir realiza-se uma chamada da operação pré-adaptativa **faa1**:

Method IV.1: faa1

```
1  input: ListaLigada c
2  begin
3     // modificação da estrutura de "c":
4     c.p ← Nó.new()
5     c.r ← c.p
6     // modificação da máquina de "c":
7     c.m ← Existente.new()
8     c.m.pósCondição()
9  end
```

No diagrama de instâncias mostrado na Fig. 8, observa-se o objeto h da classe ListaLigada em um momento de recém instanciação. Aí, ele encontra-se ligado ao estado-objeto e1. Quando h trata um evento do tipo inserir, a sua máquina de estados m realiza uma transição, chamando a operação **faa1**. Por conseguinte, o objeto h passa de uma configuração na qual ele se encontrava ligado ao estado-objeto e1 para outro, agora ligado ao estado-objeto e2. Além disso, devido ao atendimento à chamada da **faa1**, o objeto h conecta-se ao objeto p dinamicamente instanciado, ajustando, também, a referência para o nó corrente r.

Em uma vista dinâmica com destaque da transição do estado **Inexistente** para **Existente**, o diagrama de sequências de mensagens apresentado na Fig. 9 observa-se que um dos efeitos da operação pré-adaptativa é a instanciação de um novo nó n1 da lista ligada h. Também ocorre a construção do estado-objeto seguinte e2, responsável por verificar a [pós-condição] de existência de n1.

Retornando ao momento estático (Fig. 7), uma nova ocorrência de evento do tipo inserir, mas a partir do estado **Existente**, provoca uma chamada da operação pré-adaptativa **faa2** do contrato:

Method IV.2: faa2

```
1  input: ListaLigada c
2  begin
```

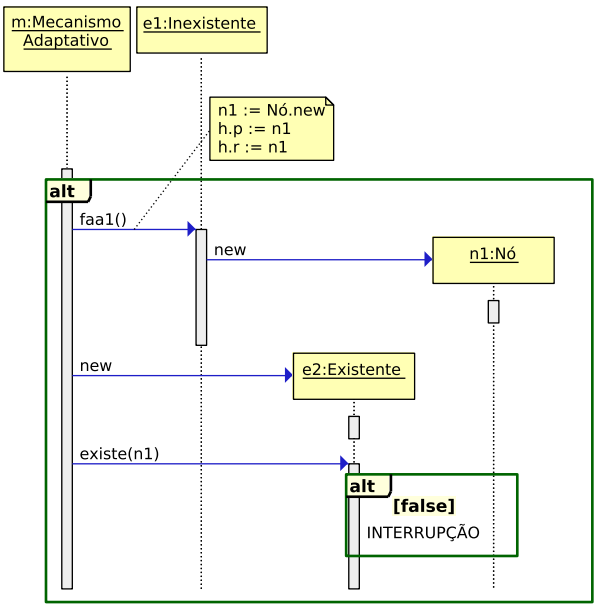

Figure 9: Ordem temporal de construção dos objetos devido à adaptação **faa1**

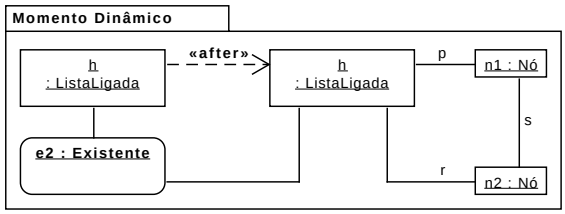

Figure 10: Estrutura de objetos depois da segunda transição de estado

```
3     // modificação da estrutura de "c":
4     c.r.s ← Nó.new()
5     c.r ← c.r.s
6     // modificação da máquina de "c":
7     c.m.pósCondição()
8  end
```

Em uma vista da estrutura dinâmica (Fig. 10), observa-se que a lista ligada h continuou conectada ao estado e2 depois de atender ao evento inserir pela segunda vez. Complementarmente, a sua estrutura de suporte à armazenagem dos elementos da lista foi modificada: existem dois nós encadeados.

A ordem temporal de modificação da estrutura de h revela que é durante o tratamento da chamada da operação pré-adaptativa **faa2** que se constrói o segundo nó da lista (Fig. 11). Como mostrado na Fig. 9, cabe ao estado-objeto e2 confirmar a pós-condição de existência do novo da estrutura ligada.

V. Consideração Final

A noção de contratos para formalizar propriedades de operações com pré-, pós-condições e invariantes ajuda na identificação de problemas de execução [4]. No entanto,

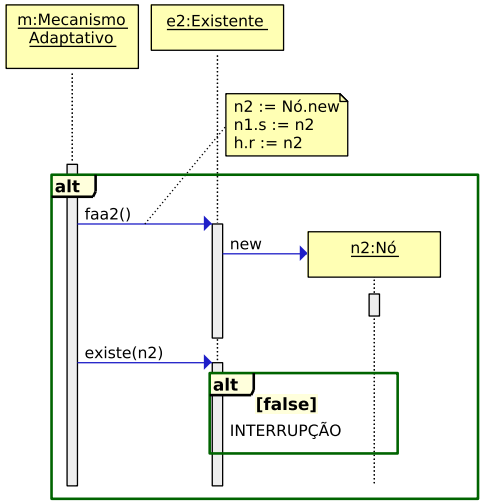

Figure 11: Ordem temporal de construção dos objetos devido à adaptação **faa2**

outras preocupações escapam do alcance de uma análise baseada em contratos. Por exemplo, em determinadas situações, espera-se que a execução de um comportamento estabeleça a pós-condição antes do seu término, mas e se a execução não terminar? Tais contratos meramente especificam correção parcial decorrente do tratamento das chamadas das operações. O requisito para que uma operação eventualmente também termine conduz a uma situação de correção total, o que pode ser feito incorporando-se uma convenção de término a todos os contratos [15].

A programação por contratos adaptativos oferece uma alternativa à convenção. Em um contrato adaptativo, a existência de operações **pré-adaptativa** e **pós-adaptativa** cria a possibilidade de se modificar o comportamento de um objeto de software em um momento dinâmico de funcionamento. Ajustes realizados nos seus campos e métodos são verificados sistematicamente pelas pré- e pós-condições, emitindo-se um sinal de interrupção caso alguma falha contratual tenha ocorrido.

Como se pode observar no exemplo ilustrativo, a preocupação com o projeto das modificações de uma estrutura de dados dinâmica foi separada de uma outra, referente à interface de funcionamento desejado em termos do tipo de dado abstrato. Neste caso, o contrato adaptativo foi utilizado para lidar com o modelo de evolução da estrutura em separado ao suporte da lógica de armazenagem dos dados do tipo abstrato.

Em uma situação mais concreta, espera-se empregar os contratos adaptativos para projetar um protótipo de ferramenta que sugira configurações de ambulâncias de emergência para atendimento de chamadas de gestantes [16]. Aqui, contratos adaptativos podem considerar que o diálogo entre o atendente e a gestante gradativamente estabelece a estrutura do modelo da gestante até que seja possível calcular a sugestão de configuração de uma ambulância de atendimento.

Há, também, um projeto de pesquisa em andamento cujo objetivo principal é analisar determinadas propriedades de encadeamento de cenas em narrativas de ambientes de aprendizagem. Após a narração de uma cena, a transição para a cena seguinte pode modificar a continuidade da história, considerando eventos de interação proporcionado pelos aprendizes. Contratos adaptativos podem ser empregados para administrar o espaço das cenas seguintes em função das interações dos aprendizes de modo que sejam evitadas configurações com ciclos ou ilhas de narração, por exemplo.

Este artigo também introduziu uma extensão da UML para a representação de contratos adaptativos, bem como dos conceitos de suporte (chamada, invocação indireta e execução). Espera-se que ela auxilie na interpretação dos modelos estáticos e dinâmicos de software projetado com o paradigma de objetos.

Agradecimentos

O autor agradece aos membros do GEMS-PUCSP pelas diversas discussões a respeito do refinamento das ideias referentes aos contratos adaptativos: Carla, Karina, Daniel, Eduardo, Francisco, Giorno e Mário.

Referências

[1] C. Hoare, “An axiomatic basis for computer programming,” *Communications of the ACM*, 1969.

[2] J.C. Reynolds, “Separation logic: A logic for shared mutable data structures,” *Proceedings of the 17th annual ieee symposium on logic in computer science*, Washington, DC, USA: IEEE Computer Society, 2002, pp. 55–74.

[3] B. Meyer, “Design by contract,” *IEEE*pp. 40–51.

[4] B. Meyer, *Object-oriented software construction*, Paperback; Prentice Hall PTR, 2000.

[5] *OMG Unified Modeling Language*, Object Management Group, 2015.

[6] I.S. Vega, “Especificações adaptativas e objetos, uma técnica de design de software a partir de statecharts com métodos adaptativos,” 2012.

[7] C. Bock, “A more object-oriented state machine,” *Journal Of Object-Oriented Programming*, vol. 12, January. 2000.

[8] L. Lamport, *Specifying systems: The tla+ language and tools for hardware and software engineers*, Addison, 2002.

[9] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Hardcover; Addison-Wesley Professional, 1994.

[10] J.J. Neto and C. Bravo, “Adaptive automata - a reduced complexity proposal,” 2002, pp. 161–170.

[11] B.W. Kernighan and R. Pike, *The practice of programming*, 1999.

[12] A.V. Aho, J.D. Ullman, and J.E. Hopcroft, *Data structures and algorithms*, Addison-Wesley, Reading, Mass., 1983.

[13] D.S. Touretzky, *COMMON lisp: A gentle introduction to symbolic computation*, The Benjamin/Cummings

Publishing Company, Inc., 1990.

[14] B.C. Pierce, *Types and programming languages*, The MIT Press, 2002.

[15] C.A. Szyperski, *Component software: Beyond object-oriented programming*, AddisonWesley, 1999.

[16] D.F. Lourenço, "A inteligência computacional no auxílio de atendimento a emergências médicas: Atendimentos emergenciais as mulheres gestantes," Dissertação de Mestrado, Pontifícia Universidade Católica de São Paulo, 2016.

Italo S. Vega recebeu os títulos de Mestre e Doutor em Engenharia de Software na Escola Politécnica da Universidade de São Paulo, Brasil, em 1993 e 1998, respectivamente. Professor Associado do Departamento de Computação da Pontifícia Universidade Católica de São Paulo e líder do Grupo de Estudos em Modelagem de Software (GEMS) do programa de pós-graduação em Tecnologia da Inteligência e *Design* Digital (TIDD).