FABRÍCIO DOI

OBJETOS ADAPTATIVOS: APLICAÇÃO DA TECNOLOGIA ADAPTATIVA À ORIENTAÇÃO A OBJETOS

São Paulo
2007

FABRÍCIO DOI

OBJETOS ADAPTATIVOS: APLICAÇÃO DA TECNOLOGIA ADAPTATIVA À ORIENTAÇÃO A OBJETOS

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Mestre em Engenharia.

São Paulo
2007

FABRÍCIO DOI

OBJETOS ADAPTATIVOS: APLICAÇÃO DA TECNOLOGIA ADAPTATIVA À ORIENTAÇÃO A OBJETOS

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Mestre em Engenharia.

Área de concentração:
Sistemas Digitais

Orientador:
Prof. Dr. Paulo Sérgio Muniz Silva

São Paulo
2007

Este exemplar foi revisado e alterado em relação à versão original, sob responsabilidade única do autor e com a anuência de seu orientador.

São Paulo, de julho de 2007.

Assinatura do autor _____

Assinatura do orientador _____

FICHA CATALOGRÁFICA

Doi, Fabricio
Objetos adaptativos : aplicação da tecnologia adaptativa à orientação a objetos / F. Doi. -- ed.rev. -- São Paulo, 2007.
87 p.

Dissertação (Mestrado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais.

1. Modelo adaptativo 2. Programação orientada a objetos 3. Engenharia de programação I. Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Computação e Sistemas Digitais II. t.

DEDICATÓRIA

Dedico este trabalho a meus pais.

AGRADECIMENTOS

Ao professor Dr. Paulo Sérgio Muniz Silva pela orientação, dedicação, compreensão e incentivo para a realização do trabalho.

Aos professores que participaram da banca do exame de qualificação, Prof. Dr. João José Neto e Prof. Dr. Ricardo Luis de Azevedo da Rocha que muito contribuíram com suas sugestões na avaliação da proposta inicial desta dissertação.

Aos meus pais, Pantaleão Doi e Maria de Lourdes Doi, que me deram carinho e condições para realizar este trabalho.

A toda a minha família que estiveram presente em todos os momentos.

À Vanessa, que me apoiou e esteve presente durante todo este trabalho.

A todos que direta ou indiretamente contribuíram para a realização deste trabalho.

RESUMO

Este trabalho estuda o problema da construção de sistemas orientados a objetos com características adaptativas, tendo como principal objetivo simplificar o processo de construção. Para isso o trabalho utiliza como base teórica a Tecnologia Adaptativa e sua aplicação em diversos formalismos. O Modelo Adaptativo de Objetos foi utilizado como base de comparação de soluções para a construção de sistemas adaptativos. Nesta pesquisa são apresentadas aplicações e uma proposição para a construção e modelagem de sistemas adaptativos, através da extensão do conceito de objetos com características da tecnologia adaptativa. Através deste estudo avaliou-se o impacto da aplicação do dispositivo adaptativo em um formalismo com tipo. Os resultados obtidos no presente trabalho demonstram que a tecnologia adaptativa é propícia para linguagens orientadas a objetos e que os diagramas UML são capazes, com pequenas extensões, de representar o comportamento adaptativo adequadamente.

Palavras-chave: Tecnologia Adaptativa, Meta-modelos, Modelo Adaptativo de Objetos,

ABSTRACT

This study addresses the issue of implementing object-oriented software with adaptive characteristics, having as primary purpose simplify the implementing process. The key theoretical basis consisted in adaptive technology and its application in various formalisms. Adaptive Object Model has been taken as comparison basis to solutions to implement adaptive systems. This study describes applications and a proposition to implement and model adaptive systems, through the extension of object concept with adaptive technology characteristics. It also evaluates the impact of applying adaptive devices in formalism with types. The results obtained demonstrate that adaptive technology is suitable for object-oriented languages and that UML diagrams are capable of presenting adaptive behavior appropriately with a small number of extensions.

Keywords: Adaptive Technology, Meta models, Adaptive Object Model

SUMÁRIO

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

UML	*Unified Modeling Language* (Linguagem de Modelagem Unificada)
OMG	*Object Management Group* (Grupo de Gerenciamento de Objetos)
API	*Application Program Interface* (Interface de Programação para Aplicação)
AOM	*Adaptive Object-Model* (Modelo Adaptativo de Objetos)
CLI	*Common Language Infraestructure* (Infra-estrutura de Linguagem Comum)
OO	Orientação a Objetos

1 INTRODUÇÃO

A modelagem de objetos adaptativos (YODER; JOHNSON, 2002) é um estilo arquitetônico que permite a criação de sistemas orientados a objetos flexíveis, que permitem a realização de customizações pelo usuário do sistema, de acordo com o surgimento de novas necessidades. Entretanto a modelagem de objetos adaptativos possui problemas, sendo que este trabalho trata da complexidade de sua representação, que mistura a solução à descrição das regras de negócio.

Em outro aspecto, a tecnologia adaptativa (NETO, 2001) é uma generalização de vários trabalhos derivados dos autômatos adaptativos (NETO, 1993) e tem por objetivo aumentar a expressividade de formalismos simples com pequenas modificações, sem que modelos feitos de acordo com o formalismo predecessor às mudanças sejam perturbados.

Este trabalho aplica algumas das características da tecnologia adaptativa, na orientação a objetos, mais especificamente na modelagem de objetos adaptativos, para aumentar a expressividade dos modelos realizados sob este estilo arquitetônico, diminuindo a sua complexidade. Entretanto, para resolver esta questão, um problema ainda pouco explorado pela tecnologia adaptativa terá de ser explorada: a aplicação do dispositivo adaptativo em um formalismo com tipo e a conseqüente alteração que tal aplicação requer na modelagem UML (*Unified Modeling Language)* (OMG, 2005) de sistemas orientados a objetos.

1.1 MOTIVAÇÃO

A principal motivação do presente trabalho é buscar uma forma simplificada para a construção de sistemas adaptativos, ou seja, sistemas que necessitam de ajustes freqüentes ou de uma maior flexibilidade durante sua execução, que por vezes é conseguida através de criações arquitetônicas que permitem a interpretação de requisitos, com meta-arquiteturas ou arquiteturas reflexivas. Uma forma de arquitetura reflexiva é o modelo adaptativo de objetos (AOM), que modela, em sua maioria,

sistemas que gerenciam certo tipo de produto e que podem adicionar outros produtos com as regras apropriadas (YODER; JOHNSON, 2002). Nesta linha de sistemas, pode-se incluir sistemas de gerenciamento de laboratório clínico, também conhecidos como LIS (*Laboratory Information System*), em que o laboratório está condicionado a novas regras impostas pelos seus principais clientes, os planos de saúde, ou a novos exames ou métodos de análises que requeiram diferentes informações para a criação de laudos apropriados.

Além do AOM, outras soluções podem ser utilizadas para a criação deste tipo de sistemas, como a geração automática de código e o uso de técnicas de reflexão. Além destes, alguns padrões de projeto prevêem esse tipo de variação permitindo uma solução para esses tipos de sistemas. A geração automática de código se diferencia por desacoplar o meta-modelo do sistema, entretanto necessita de geração de código e nova compilação para que as mudanças tenham efeito, enquanto no AOM as mudanças são imediatas. O uso de técnicas de reflexão induz à utilização de mecanismos prontos, já existentes no sistema, que devem ser incluídos em um processo dinamicamente, tornando o processo adaptativo. Novamente, a grande diferença entre esta solução e o AOM é a necessidade de um código compilado, pronto para ser executado, enquanto o AOM interpreta uma informação de acordo com o meta-modelo desenhado para desenvolver um comportamento apropriado ao sistema. Portanto, o AOM é útil à construção de sistemas que possibilitam uma adequação do mesmo em tempo de execução, sem que haja a necessidade de compilação ou geração de código.

Embora o AOM permita a construção de sistemas flexíveis, ele aumenta a complexidade de sistemas, pois mistura à descrição do escopo do problema o meta-modelo gerado para permitir tal ganho, o que dificulta a compreensão do sistema.

Com o estudo da tecnologia adaptativa (NETO, 2001; NETO, 1993), uma alternativa parece surgir para resolver a dificuldade do aumento da complexidade do sistema enfrentado pelo AOM, com a introdução do dispositivo adaptativo. Pela simplicidade com que ela é tratada, espera-se que a inclusão da tecnologia adaptativa na teoria dos

objetos permita que a adaptatividade possa ser aplicada sem apresentar os problemas citados e a custos aceitáveis.

Apesar de tal simplicidade, outra motivação para o presente trabalho é a representação do dinamismo que a tecnologia adaptativa provê. Ainda não se encontrou uma maneira suficientemente simples e clara para representar o comportamento adaptativo dos modelos. Isso é um problema para a pesquisa por novas soluções com a Tecnologia Adaptativa, pois, sem uma boa representação, a criação, compreensão e depuração de soluções adaptativas tornam-se mais difíceis.

1.2 OBJETIVOS

O principal objetivo do presente trabalho é aplicar alguns conceitos da tecnologia adaptativa à orientação a objetos, mais especificamente ao AOM, para criar uma camada em que se definem os objetos adaptativos e realizar uma análise preliminar de alguns dos benefícios e dificuldades trazidas pela tecnologia adaptativa à orientação a objetos. Para tanto, serão estudados alguns aspectos elementares da teoria dos objetos (ABADI; CARDELLI, 1996), procurando utilizá-los pragmaticamente para que os resultados do presente trabalho possam ser utilizados nas linguagens orientadas a objetos disponíveis comercialmente, destacando-se alguns problemas que possam ocorrer nessa integração, como a questão da alteração dinâmica do tipo do objeto ao longo de seu ciclo de vida e a influência dessa alteração no sistema de tipos. No presente trabalho, este último aspecto é apenas formulado, pretendendo-se que sua análise seja retomada formalmente em um trabalho futuro que deverá estender a teoria de objetos de Abadi e Cardelli (1996) para abranger o conceito de dispositivo adaptativo (NETO, 2001).

Com a aplicação de alguns conceitos da Tecnologia Adaptativa, espera-se poder simplificar o entendimento, a documentação e a manutenção de sistemas construídos sobre o estilo arquitetônico do Modelo Adaptativo de Objetos, podendo separar claramente um meta-modelo construído segundo esse estilo da solução de um problema qualquer com o uso de um arcabouço.

Uma questão fundamental da aplicação da Tecnologia Adaptativa à Orientação a

Objetos é a modelagem das abstrações computacionais. Assim, o presente trabalho analisa alguns dos diagramas mais usados da modelagem UML (*Unified Modeling Language)* definido pela OMG (*Object Management Group*)[1]. Os diagramas de classe e de seqüência serão revistos e preparados para representarem os objetos adaptativos e seu comportamento, apresentando-se uma proposta para a representação do dinamismo da tecnologia adaptativa aplicada à orientação a objetos.

1.3 ESTRUTURA DO TRABALHO

O trabalho está dividido em três partes principais: a apresentação dos fundamentos teóricos envolvidos, da solução proposta e, finalizando, das conclusões e trabalhos futuros vislumbrados.

Os capítulos 2, 3 e 4 apresentam os fundamentos teóricos necessários, descrevendo a Orientação a Objetos (OO), a Tecnologia Adaptativa e o Modelo Adaptativo de Objetos (AOM).

No capítulo 2 é apresentada a Teoria dos Objetos, bem como alguns aspectos de sua modelagem e alguns problemas que devem ser enfrentados na sua extensão. No final do capítulo apresenta-se uma síntese do formalismo do Cálculo de Objetos sem tipo.

No capítulo 3, o Modelo Adaptativo de Objetos é apresentado como uma solução correlata para a criação de sistemas adaptáveis. Sua arquitetura e seus principais aspectos são estudados, com a apresentação dos principais padrões de projeto (GAMMA; 1995) utilizados nesta arquitetura.

O capítulo 4 apresenta a Tecnologia Adaptativa como ferramenta para a extensão de formalismos. Apresenta um breve histórico da tecnologia, mostrando os formalismos e situações em que foi aplicada. Em seguida, apresenta o Dispositivo Adaptativo e sua aplicação nos autômatos. Por fim, apresentam-se um exemplo com autômatos adaptativos e algumas observações sobre a tecnologia.

---

[1] OMG (*Object Management Group)* é um consórcio formado por grandes empresas de computação para produzir e manter as especificações da indústria da computação para a interoperabilidade das aplicações das empresas, como definido em seu site http://www.omg.org.

O capítulo 5 apresenta a solução proposta para a aplicação da Tecnologia Adaptativa à Orientação a Objetos. Os principais problemas na integração são analisados e uma solução será proposta. Além disso, o capítulo apresenta algumas proposições para a alteração da UML 2.0 para utilizá-la na descrição de sistemas adaptativos.

Por fim, no capítulo 6 o trabalho é concluído sendo apresentadas situações em que os objetos adaptativos podem ser aplicados, e as vantagens e desvantagens desse tipo de solução. Também são sugeridos temas para o desenvolvimento de trabalhos futuros.

2 CONCEITOS DA ORIENTAÇÃO A OBJETOS

Neste capítulo alguns conceitos da orientação a objetos são apresentados, de forma a embasar a sua posterior extensão. Assim, apresentam-se algumas características gerais desse paradigma de programação, alguns aspectos da representação de modelos orientados a objetos e um formalismo criado para sustentar a orientação a objetos.

2.1 CARACTERÍSTICAS GERAIS DA ORIENTAÇÃO A OBJETOS

O texto desta seção baseia-se integralmente em (ABADI; CARDELLI, 1996), procurando sintetizar o significado dos conceitos de objetos dentre as formas como este é tratado em diferentes linguagens de programação.

Os objetos começam a ser utilizados na construção de sistemas de software nos anos 60, a partir da analogia da construção de um sistema mecânico onde objetos concretos são utilizados. Esta visão se encaixa nas três fases de construção de um software: a análise é melhorada com a analogia em si, permitindo maior integração do software com o que ele representa; durante a fase de projeto são criados modelos aderentes e com maior persistência, ou seja, tem mais resistência às mudanças provocadas durante a vida do sistema; para a construção, o reuso de código é facilitado, criando objetos reusáveis e presentes de maneira mais eficaz.

Apesar das diferenças apresentadas entre os paradigmas de programação, os chamarizes da orientação a objetos não são exclusivos dela. A resistência do modelo pode ser realizada com a organização do programa em volta de tipos de dados abstratos (LISKOV; GUTTAG, 1986). Reuso de código pode ser feito através de modularização (PARNAS, 1972; WIRTH, 1983) e parametrização (MILNER, 1978; MILNER; et al, 1989). Uma vantagem do paradigma dos objetos é o reuso de código, quando é permitida não só a reutilização de código através do mecanismo de herança, como a sobreposição de objetos ou métodos, sem a necessidade de concordância total de tipo ou interface. Os objetos são derivados de outros criados anteriormente, combinando reuso (pelo mecanismo de herança) e variação (com a sobreposição). Os métodos podem ser

sobrescritos, sempre mantendo a interface do método antigo, mas podendo utilizar atributos existentes no objeto derivado.

O paradigma da Orientação a Objetos vem se desenvolvendo com a criação de Padrões de Projeto (GAMMA; et al, 1995). Com seu crescente uso os padrões de projeto vêm se afirmando como uma forma apropriada de organização de sistemas.

Foram muitas as linguagens que antecederam a linguagem Java no paradigma da orientação a objetos, sendo a mais conhecida a linguagem denominada Smalltalk (NCITS, 1998). As linguagens orientadas a objeto, apesar de pertencerem ao mesmo paradigma, podem ser divididas em dois grandes grupos: as linguagens baseadas em classes e as linguagens baseadas em objetos.

2.1.1 LINGUAGENS BASEADAS EM CLASSES

Como exemplos de linguagens baseadas em classes, temos Java, C++ e o próprio Smalltalk. Estas linguagens têm como característica o conceito de classe o qual fornece a descrição de objetos gerados por ela.

Os objetos das linguagens baseadas em classes podem possuir métodos embutidos no próprio objeto, ou podem acessá-los em um conjunto de métodos da classe através da delegação. Na primeira opção, mais fácil de entender, os métodos são incluídos juntos aos seus dados no momento da criação do objeto, como sugere a **Erro! Fonte de referência não encontrada.**.

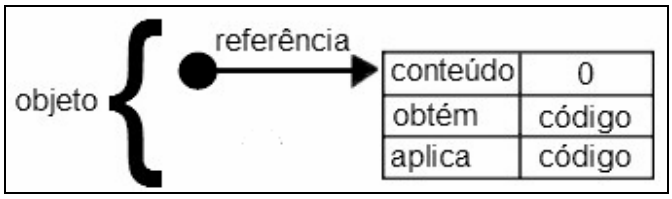

Figura 1 – Métodos embutidos nos objetos – Adaptado de (Abadi; Cardelli, 1996)

Na segunda opção, os códigos dos métodos ficam agrupados em conjunto(s) de método(s), como mostrado na **Erro! Fonte de referência não encontrada.**. Desta forma, os objetos delegam a execução de um método do conjunto de métodos ocupando menos espaço quando coexiste em um maior número de objetos de uma classe.

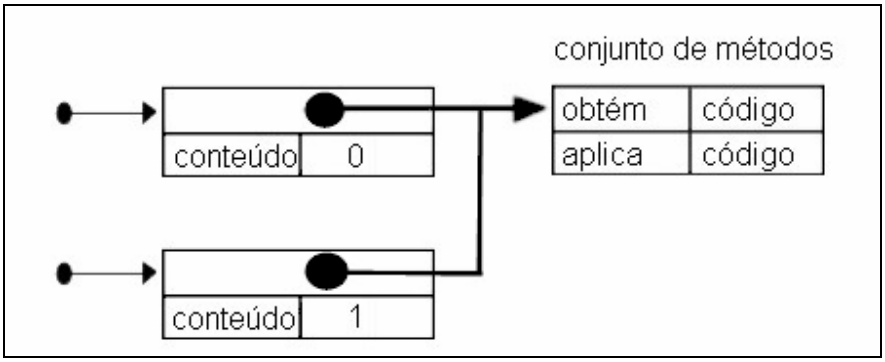

Figura 2 – Métodos pertencendo a conjuntos separados dos objetos – Adaptado de (Abadi; Cardelli, 1996)

Outro conceito importante nas linguagens baseadas em classes é o conceito de subclasse e herança. Uma subclasse descreve uma classe a partir da extensão e mudanças da superclasse, com a replicação implícita dos campos e métodos, que podem ou não ser sobrescritos. A herança é o compartilhamento dos atributos entre uma classe e suas subclasses. Assim, a subclasse pode utilizar seus próprios métodos, como os métodos herdados da classe, utilizando um identificador especial para acessá-los (*super* nas linguagens Java e C#). Da mesma forma, é possível especificar que se está utilizando o método do objeto na subclasse e não da superclasse utilizando outro identificador especial (*this* em Java e em C#).

As linguagens baseadas em classe possuem, ainda, a subsunção, ou seja, se um objeto é uma instância de uma classe (isto é, ela foi criada a partir desta classe), então ele também é instância da superclasse. Essa propriedade traz duas possibilidades na invocação de um método. Quando um método deve receber um objeto da superclasse, ele pode receber um objeto da subclasse ou da superclasse, e a chamada de um método sobrescrito torna-se um problema. Para isso têm-se duas alternativas: disparo estático do método, que sempre chamará o método da superclasse, ou o disparo dinâmico, que verificará a classe específica a que pertence o objeto e chamará o método correspondente.

Como conseqüência da possibilidade de se passar por parâmetros objetos de subclasses em um método que recebe uma instância de superclasse, as linguagens provêem uma forma de se verificar a qual classe um objeto pertence. Isso é discutido por puristas da

OO, com argumentos tais como a violação da abstração dos objetos, com uma conseqüente fragilidade dos programas quando uma nova subclasse é criada.

Teóricos contrários à verificação da classe do objeto argumentam que a Orientação a Objetos permite a especialização dos métodos da superclasse, isto é, a especialização de um método sobrescrito com a alteração dos argumentos e do tipo do resultado. Com isso, um método pode ser sobrescrito para ser específico em cada subclasse, de forma que a verificação da classe não precisa ser utilizada.

Outra ferramenta para evitar o uso da verificação de tipo é a especialização Auto Tipo (*Self Type Specialization*). Esta especialização não tem nenhuma semelhança com o *this* presente nas linguagens C# e Java, que não possuem tal especialização. De forma resumida, a especialização Auto Tipo permite que um desenvolvedor especifique como entrada ou saída de um método, o tipo do objeto em questão, e não da classe em que o método foi definido. Em outras palavras, se existe uma classe “Pedido” que apresenta um método “Novo” o qual devolve um *Self Type*, e outra classe “Pedido Médico” que herda as características da classe “Pedido”, então o método “Novo” de um objeto da classe “Pedido Médico” devolve um objeto *Self Type*, ou seja, devolve um objeto da classe “Pedido Médico” e não um objeto genérico da classe “Pedido”, como normalmente ocorre nas linguagens comerciais.

Há a necessidade de se distinguir os conceitos de tipo de um objeto e da classe a que ele pertence. O tipo de um objeto indica somente a lista de propriedades (campos e métodos) que ele possui, sem sua construção. Dado que um objeto é de um determinado tipo, é possível saber tudo o que ele contém. Já saber que um objeto é pertencente a uma classe não é suficiente para conhecer seus atributos. Isso porque, devido à subsunção, o objeto pode, na verdade, ser instância de uma subclasse da classe, possuindo diferenças na implementação e podendo conter outros atributos.

Assim, o subtipo pode ser definido como possuindo todos os componentes do tipo e, possivelmente, outros adicionais. Isso faz com que, nem sempre, uma subclasse pertença a um subtipo, já que a subclasse pode, simplesmente, sobrescrever um ou mais métodos da classe (sem especializá-los) e pertencer ao mesmo tipo da classe.

As linguagens baseadas em classes podem ainda conter o conceito de protocolo de objeto. Um protocolo de objeto define um conjunto de atributos que um objeto deve conter para pertencer a este protocolo. Desta forma, um método pode ser parametrizado com este protocolo e diferentes tipos de objetos, independente de serem relacionados por herança, podem satisfazer como parâmetro do método.

2.1.2 LINGUAGENS BASEADAS EM OBJETOS

As linguagens baseadas em objetos tiveram menos atenção e por isso foram menos desenvolvidas. Como exemplos têm-se as linguagens Simula, Emerald, Cecil e Omega. São mais simples que as linguagens baseadas em classes e, devido à ausência de classes, os objetos são criados através de outros objetos, denominados criadores de objetos. Por essa ausência de classe, a programação possui como grande atrativo uma maior flexibilidade na criação de cada objeto. Entretanto, a organização do sistema tem maior complexidade, o que faz com que pseudo-classes sejam criadas para facilitar o desenvolvimento de sistemas gerados a partir desse tipo de linguagem.

Para que objetos parecidos sejam construídos, as linguagens baseadas em objetos possuem os conceitos de protótipos e de clonagem. Diferente das classes, os protótipos são partes do sistema como instâncias e servem como base para que outros objetos sejam criados a partir da sua clonagem, a criação de cópias idênticas, mas com estados independentes. Para se dispor da clonagem com vantagem, é possível alterar os atributos e os métodos dos objetos criados.

A herança em linguagens baseadas em objetos pode aparecer de duas formas: embutida nos objetos ou por delegação. A primeira ocorre quando os objetos clonados contêm cópias dos atributos do objeto doador. Já a herança por delegação é feita através do uso de referências para o objeto doador, de forma que, ao chamar um método não sobrescrito o código executado será o do doador. A grande questão da delegação é que os doadores são partes ativas do sistema e quaisquer mudanças nestes objetos afetam seus clones. Isso pode tanto fragilizar os sistemas (DONY; et. al., 1992; TAIVALSAARI, 1993), como pode ser usado como vantagem. Por outro lado, a delegação permite um consumo mais eficiente do espaço utilizado pela aplicação.

Os tipos nas linguagens baseadas em objetos como Cecil e Omega restringem o dinamismo apresentado por linguagens deste tipo, justamente por serem parecidas com os tipos utilizados nas linguagens baseadas em classes. Nestas linguagens, os tipos têm duas características distintas: a mudança de modo (*mode-switching*) pode receber mais facilmente o tipo apropriado; e o tratamento de especialização de método e o auto tipo devem ser revistos, já que os métodos podem ser atualizados, o que confunde o conceito de tipo. A mudança de modo é um caso especial de herança dinâmica, em que uma coleção de atributos herdados é trocada por uma coleção de atributos similares.

2.2 MODELAGEM ORIENTADA A OBJETOS

A modelagem orientada a objetos está ancorada na abstração de dados, a qual deve representar no âmbito computacional uma visão do problema a ser modelado centrado na analogia a coisas e conceitos do mundo real. Assim, em vez de tratar simplesmente com algoritmos e fluxo da informação, a modelagem trata conjuntamente a informação que será tratada e as operações sobre ela, pretendendo ser de entendimento mais simples por pessoas que não participem da construção de sistemas.

Para a especificação de projetos orientados a objetos surgiu a UML – *Unified Modeling Language* (Linguagem de Modelagem Unificada), atualmente na versão 2.0 (OMG, 2005). A UML é uma linguagem de modelagem bastante adequada à orientação a objetos. A UML define uma série de artefatos, e especifica a representação de seus elementos formadores. Os artefatos, em especial os diagramas, têm diferentes intuitos. Aqui serão expostos alguns dos artefatos que serão influenciados pela adição do dispositivo adaptativo na teoria dos objetos: o diagrama de caso de uso, de estado, de classe, de seqüência e de comunicação.

2.2.1 DIAGRAMA DE CASOS DE USO

O Diagrama de Casos de Uso (OMG, 2005) tem o objetivo de melhorar a comunicação entre o cliente do sistema e os desenvolvedores, com a documentação e esclarecimento dos requisitos os quais o cliente espera que sejam atendidos pronta ou futuramente pelo sistema. Além disso, os casos de uso são importantes para esclarecer situações não

atendidas diretamente pelo sistema, mas que fazem parte do contexto a que o sistema será aplicado.

Além do diagrama, é parte essencial nesse contexto a especificação dos casos de uso (não definidos explicitamente na UML), que permitem o melhor entendimento entre as partes. O diagrama é apenas uma visão resumida e pictórica dos casos de uso. A especificação dos casos de uso representa toda a interação entre o ator e o sistema, as atividades que têm valor para o ator e detalhes de como as regras devem ser realizadas. Por outro lado, o diagrama de casos de uso representa apenas o ator e seu relacionamento com o sistema, dando uma visão mais genérica do sistema sem entrar em detalhes de especificação das atividades.

Em princípio, esse diagrama não deve ser alterado com a adição da tecnologia adaptativa, mas pode ser interessante evidenciar a presença de elementos dinâmicos no momento da extração de requisitos, podendo fazer com que a descrição e o diagrama tenham novos componentes.

Os casos de uso foram propostos inicialmente em (JACOBSON, 1992), em que é definida uma metodologia de desenvolvimento orientado a objetos. Os casos de uso são essencialmente formados por atores, um conjunto de eventos que se realizam e condições de pré e pós-realização. Como ator subentende-se quaisquer entidades que interagem com o sistema em questão, seja um usuário ou um outro sistema que se comunica com o sistema modelado. Atores são entidades que nunca pertencem ao sistema. O caso de uso é a forma como é especificado o comportamento que o sistema deve prover, incluindo o comportamento normal e excepcional e o tratamento de erro.

Os casos de uso podem relacionar-se através da extensão e da inclusão. Na extensão, o comportamento de um caso de uso pode ser estendido a partir do comportamento de um outro caso de uso. A extensão pode conter uma condição para a sua ocorrência, bem como um ou mais pontos de extensão (característica do caso de uso que indica o lugar no comportamento do caso de uso em que ele pode ser estendido) em que ela pode ocorrer. Já a inclusão ocorre quando o comportamento de um caso de uso é incluído no comportamento de outro caso de uso.

Note que, assim como o diagrama não fornece detalhes dos casos de uso, o mecanismo adaptativo não precisa estar presente neste diagrama, o que fortalece a hipótese de que o diagrama não deva ser alterado.

2.2.2 DIAGRAMA DE ESTADOS

A UML divide as máquinas de estados (OMG, 2005) em dois tipos: comportamental e protocolar. A máquina de estado comportamental (*Behavioral state machines*) pode ser utilizada para mostrar o comportamento de vários elementos do modelo, como uma instância de uma classe. A máquina de estado protocolar (*Protocol state machines*) é usada para expressar protocolo de uso, sendo conveniente na definição do ciclo de vida de objetos, ou da ordem de invocação de suas operações.

Pelo fato de o diagrama mostrar o comportamento de elementos do sistema, uma alteração com a inclusão da tecnologia adaptativa é necessária, já que as ações adaptativas devem alterar o comportamento dos elementos representados. Além disso, esse diagrama tem uma grande semelhança com o que representa os autômatos e se originou a partir dos *statecharts*, ambos já estendidos com o dispositivo adaptativo, como visto no capítulo 4.

2.2.3 DIAGRAMA DE CLASSES

O diagrama de classes (OMG, 2005) é o mais importante diagrama da UML, pois ele representa a informação que o sistema trata, sendo, desta forma, estático. Mesmo sendo estático, este diagrama deve sofrer alterações com a inclusão dos objetos adaptativos. Em primeiro lugar, ele retrata objetos através de classes estáticas e não objetos adaptativos com classes maleáveis (ou seja, classes que permitam os objetos se adaptarem), sendo importante esta diferenciação e representação. Além disso, o diagrama fixará os tipos a que um objeto de uma classe poderá pertencer durante seu ciclo de vida.

O diagrama de classes é formado por classes, com a representação de suas denominações, atributos e métodos, bem como as associações e dependências que têm entre si, generalizações realizadas e abstrações existentes. As associações representam a

relação semântica que ocorre entre duas instâncias. Dependência é uma relação indicando que um elemento necessita de outro para sua especificação ou implementação. A generalização é uma relação hierárquica entre uma classe mais genérica e outra mais específica.

Seguindo a linha de especulação sobre mudanças no diagramas de classes devidas à introdução do dispositivo adaptativo, o diagrama deve se manter estrutural e, a menos que esta estrutura não seja mudada, este diagrama não sofrerá mudanças com a ação do dispositivo adaptativo. Como apresentado no item 5.1.2, a mudança na estrutura só ocorre em uma forma de adaptatividade, a adaptatividade baseada em classes, que não será tratada neste trabalho. Desta forma, este diagrama não terá comportamento dinâmico com a introdução do dispositivo adaptativo e apenas representará a maleabilidade dos objetos, como apresentado no item 5.5.1.

2.2.4 DIAGRAMA DE SEQÜÊNCIA

O diagrama de seqüência (OMG, 2005) mostra como cada caso de uso é realizado através dos objetos do sistema. O diagrama representa o comportamento do sistema que se tornará dinâmico com a tecnologia adaptativa, trazendo a necessidade da alteração do diagrama. O diagrama de seqüência tem aspecto temporal, demonstrando a ordem com que as interações entre os objetos ocorrem.

O diagrama de seqüência representa instâncias de classes em suas linhas de vida (*lifelines*), e como, e em que ordem, ocorrem as interações entre elas e o recebimento de eventos externos para a realização de um comportamento (ou caso de uso) do sistema. Essas interações são representadas através das mensagens trocadas entre os objetos e da representação da execução dos métodos na linha de vida do objeto.

O diagrama de seqüência, diferente do diagrama de classes, não deve ter um comportamento específico, mas diversos comportamentos, de acordo com o contexto e o histórico do objeto. Portanto, o diagrama terá comportamento dinâmico, ou, em outras palavras, terá diferentes formas de acordo com o contexto e seu histórico. Uma solução

para representar o comportamento adaptativo neste diagrama é apresentada no item 5.5.2.

2.2.5 DIAGRAMA DE COMUNICAÇÃO

Complementar ao diagrama de seqüência, existe o diagrama de comunicação (OMG, 2005). O que distingue os dois diagramas é o ponto de vista de como os casos de uso são resolvidos, ou as formas como as interações são realizadas. Enquanto o diagrama de seqüência mostra a ordem em que ocorrem as interações, o diagrama de colaboração destaca os relacionamentos entre os objetos, mostrando a topologia destes relacionamentos. Assim como o diagrama de seqüência, este diagrama deve ser revisto ao aplicar a tecnologia adaptativa, pois com as ações adaptativas há a tendência da topologia e seus relacionamentos serem alterados em tempo de execução do sistema.

Este diagrama apresenta as instâncias e as trocas de mensagens, mas sem representar a linha de vida graficamente (ele apresenta algumas seqüências de mensagens através de simples numeração). Além disso, não leva em conta a sobreposição de mensagens, assumindo-as como irrelevantes.

2.3 TEORIA DOS OBJETOS

Em (ABADI; CARDELLI, 1996) é apresentada a primeira formulação matemática completa de uma teoria de objetos. No presente trabalho, do formalismo será apresentado apenas o Cálculo de Objetos sem tipo, o qual servirá para mostrar que um objeto é uma coleção de atributos nomeados, todos eles sendo considerados como métodos (ABADI; CARDELLI, 1996). Além disso, a teoria de objetos completa – incluindo o cálculo de objetos com tipos – fornecerá as bases teóricas para sua extensão com o dispositivo adaptativo.

O objetivo desta seção é fornecer uma idéia geral do estilo de argumentação do cálculo. Para tanto, o texto apresenta, a título ilustrativo, alguns fragmentos elementares do cálculo de objetos sem tipo, que tratam da semântica primitiva de operações de objetos, os quais agem sobre métodos no sentido dado pela teoria de objetos. A hipótese a ser

refinada e analisada em um trabalho futuro é que tais operações de objetos podem ser apropriadamente estendidas para embutir formalmente ações adaptativas.

O Cálculo de Objetos ou Cálculo-ς é formado por um pequeno conjunto de primitivas: a definição de objetos e as reduções (primitivas de semântica). Resumidamente, cada método tem uma variável vinculada representando o eu (*self*) e um corpo que produz um resultado. As únicas operações nos objetos são: invocação de métodos e atualização de métodos. A invocação executa um determinado método, retornando um resultado. A atualização, sob a ótica da semântica funcional, produz uma cópia do objeto na qual o método é substituído por um outro método. O conceito de campo, presente em linguagens orientadas a objetos, formalmente não faz parte do cálculo de objetos. Os autores justificam que um campo pode ser tratado como um tipo especial de método, em que a invocação do método devolve o valor do campo e a atualização do método corresponde à atribuição de um valor ao campo[2].

Um objeto $[l_1=\varsigma(x_1)b_1, \ldots, l_n=\varsigma(x_n)b_n]$ é definido como uma coleção de componentes $l_i=\varsigma(x_i)b_i$, de nomes l_i distintos e métodos associados $\varsigma(x_i)b_i$, para $i \in 1\ldots n$; a ordem dos componentes não é importante. O método $\varsigma(x)b$ possui parâmetro próprio x e corpo b.

A invocação de um método é escrita $o.l$, em que l é o nome do método invocado, pertencente ao objeto. A atualização $o.l \Leftarrow \varsigma(x)b$ corresponde à criação de uma cópia do objeto com a substituição do método nomeado l com $\varsigma(x)b$. É importante frisar que, com a inclusão do dispositivo adaptativo, o cálculo passa a ter três novas operações, a exclusão, a atualização e a inclusão de métodos, como pode ser visto no item 5.1.1.

Os campos podem ser retratados como métodos que não usam a variável própria, os chamados métodos próprios. Assim, uma invocação de método passa a ser uma seleção de campo e uma atualização de método passa a figurar como atualização de campo.

---

[2] As linguagens apresentam diferentes definições para objetos. Nenhuma das linguagens comerciais orientadas a objetos mais conhecidas como Java, C# e C++, por exemplo, apresentam métodos que possam ser atualizados como sugere o modelo de objetos da Teoria de Objetos de (Abadi, Cardelli, 1996). Além disso, a denominação dos conceitos do modelo de objetos realizado por tais linguagens é distinta. Por exemplo, método é chamado de função membro em C++. Mais diversa ainda é a denominação de campo. C# apresenta-o como campo, Java como campo ou variável de instância e C++ como membro de dados.

Apesar desta diferenciação, um método pode, a qualquer momento, se transformar num campo e vice-versa.

O texto citado define a redução, passos de redução e define as equações de simetria, unicidade, transitividade e prova a completude do cálculo. Para o presente trabalho, isto tudo serve de base para a utilização do conceito de que um objeto é formado por um conjunto de apenas uma forma de entidade, os métodos, possibilitando a visão de que métodos e atributos de um objeto possam ser vistos da mesma maneira.

3 MODELO ADAPTATIVO DE OBJETOS

Segundo (YODER; JOHNSON, 2002) o Modelo Adaptativo de Objetos, ou *Adaptive Object-Model* (AOM), é um sistema que representa classes, atributos, relacionamentos e comportamento como meta-dados. Desta forma, o modelo é baseado em instâncias e não em classes, e a execução do sistema é realizada com a interpretação dos meta-dados.

Também conhecido como Modelo Ativo de Objetos ou Modelo Dinâmico de Objetos, o Modelo Adaptativo de Objetos é uma estrutura reflexiva particular, que permite ao usuário definir novos comportamentos para o sistema de acordo com a criação de novas necessidades.

Com isso é possível construir sistemas que atendam negócios em que as regras variam com freqüência seguindo um determinado padrão. Assim, o modelo pode ser feito para atender a este padrão de alteração, permitindo ao sistema atender diferentes regras que surgirem durante a sua vida. Desta forma, o modelo pode ser preparado para que estas mudanças sejam incluídas no sistema por um usuário do sistema, sem a necessidade de intervenção de um desenvolvedor. Um exemplo de variação é a inclusão de novos produtos em uma empresa. Cada produto de uma empresa pode ter caracterizações diferentes, fazendo com que o comportamento dos negócios tenha de ser individualizado por produto.

Enquanto os modelos de objetos representam a informação tratada pelo sistema, o modelo adaptativo de objetos representa a forma como a informação é interpretada. Ou seja, um modelo adaptativo pode ser visto como um meta-modelo da dinâmica do processamento da informação no sistema, apresentando a informação de forma implícita[3].

O AOM utiliza uma série de padrões de projetos como o Compósito (*Composite*), o Interpretador (*Interpreter*), o Construtor (*Builder*), a Estratégia (*Strategy*) (GAMMA; et al, 1995), o Objeto Tipo (*Type Object*) (JOHNSON; WOLF, 1998), a Propriedade (*Property*) (FOOTE; YODER, 1998) e o Objeto Regra (*Rule Objects*) (ARSANJANI,

[3] Pretende-se elaborar esta interpretação na dissertação e em um artigo futuro.

2000). A prática usual é aplicar os padrões Objeto Tipo e Propriedade conjuntamente para formar a arquitetura Quadrado de Tipos (*Type Square*) (YODER; et al, 2001). Com estes padrões de projeto, o AOM pode representar os atributos e as regras do negócio como meta-dados que são interpretados em tempo de execução. Como tudo é interpretado, o modelo pode ser alterado com uma simples alteração nos meta-dados.

Neste aspecto, o AOM permite uma flexibilidade bastante destacada por todos os arquitetos que experimentam este tipo de solução (YODER;JOHNSON, 2002). Entretanto, os modelos produzidos segundo o AOM são normalmente complexos, esbarrando na dificuldade de se encontrar mão de obra com experiência neste tipo de modelagem (YODER;JOHNSON, 2002).

O grande problema do AOM é sua utilização como solução para as freqüentes alterações nas regras de negócio. Daí o modelo passa a misturar a informação tratada pelo sistema com a solução, fazendo com que o modelo se enfraqueça, pois ele perde a abstração da regra modelada, prejudicando o desenvolvimento. Isso faz também que seu custo de manutenção aumente muito, inviabilizando, por muitas vezes, a utilização deste tipo de arquitetura. Outra visão é que as alterações freqüentes nas regras de negócio fazem parte do negócio, e por isso o modelo deve refletir isto de maneira adequada, levando ao modelo adaptativo, mas, do mesmo modo, o modelo é complexo e encarece a solução.

Apesar de o AOM ser uma solução interessante para modelar sistemas para negócios em que as regras têm grande dinamismo, todos os autores de AOM indicam que é necessário um bom estudo antes de se utilizar este recurso. Isso acontece pois o AOM apresenta um problema em desempenho, devido à necessidade de interpretação dos meta-dados, além do já comentado problema de custo de manutenção. O desempenho tem uma queda significativa em relação a outros sistemas pelo fato de o código feito no AOM ser duplamente interpretado: a interpretação da máquina virtual orientada a objetos e a interpretação que o programa possui internamente para criar o comportamento adaptativo. Essa queda de desempenho unida ao alto custo de construção e manutenção deve, então, ser considerada para se verificar a viabilidade da construção de um sistema com o AOM.

Uma alternativa para esse problema do AOM é apresentada em (DANTAS; et al, 2004) com a união do AOM à programação orientada a aspectos (KICZALES, 1996). A orientação a aspectos prevê uma separação do problema em diversos grupos, tornando-se assim mais fácil de compreender as soluções modeladas e facilitando o reuso de código. A partir desta visão, o autor procura separar o modelo adaptativo da informação do negócio.

Além da união com a programação orientada a aspectos, existe a possibilidade de implementar a adaptatividade na própria máquina virtual orientada a objetos, como a *Java Virtual Machine* ou o *.NET Microsoft Framework*. Uma máquina virtual é um programa que simula uma máquina computacional, realizando operações definidas para ela. Uma máquina virtual orientada a objetos realiza operações sobre objetos, com a interpretação do código em tempo de execução. Então, uma proposta é a extensão de uma destas máquinas para ela se tornar uma máquina virtual orientada a objetos adaptativos.

Um exemplo de extensão de máquina virtual é feito em (ORTIN; et al, 2005). Neste trabalho a Infra-estrutura de Linguagem Comum de Código Compartilhado – *Shared Source Common Language Infrastructure* (MICROSOFT - ROTOR, 2006), também conhecida como Rotor, é estendida para que ela apresente a reflexão estrutural. O Rotor é uma versão de código aberto da *Microsoft* que implementa a Infra-estrutura de Linguagem Comum – *Common Language Infraestructure* (ECMA, 2006), um padrão criado pela *ECMA International*[4] para permitir que aplicações sejam escritas em diferentes linguagens de alto nível e possam ser executadas em diferentes ambientes, sem a necessidade de reescrever as aplicações para considerarem as características de cada ambiente.

A reflexão estrutural é uma das distinções para a reflexão feitas em (ORTIN; et al, 2005). Levando em conta o que pode ser refletido, o trabalho faz a distinção das formas

[4] A *ECMA International* é uma associação de indústrias fundada em 1961 e dedicada à padronização da Tecnologia de Comunicação de Informação (ICT) e da eletrônica de consumo (CE). Sítio http://www.ecma-international.org

de reflexão em Introspecção, Reflexão Estrutural e Reflexão Computacional (Comportamental). A Introspecção é a simples consulta da estrutura de um sistema, na qual o programador pode consultar informações das classes, objetos, métodos e campos em tempo de execução, como ocorre no pacote *java.lang.reflect* (Java) e no espaço de nome *System.Reflection* (.NET). A Reflexão Estrutural permite a alteração da estrutura de um sistema, como ocorre na linguagem de programação Python em que campos e métodos podem ser adicionados tanto a classes como a objetos. A Reflexão Computacional permite a mudança da semântica do sistema, o que provoca a mudança do comportamento do sistema, como ocorre no MetaJava (GOLM; KLEINÖDER, 1997), uma extensão Java que permite a alteração no mecanismo de disparo de métodos.

3.1 ARQUITETURA DO AOM

A arquitetura do Modelo Adaptativo de Objetos apresenta a forma como a informação deve ser interpretada. Desta maneira, o modelo deve representar a estrutura de como a informação é armazenada, para que o interpretador trabalhe com ela.

O AOM, em geral, apresenta-se, principalmente, como solução para quatro casos: utilização de objetos compostos, objetos formados por outros objetos; existência de diferentes algoritmos para um mesmo problema; representação de regras instáveis, que necessitam de constante manutenção; representação de tipos de dados variáveis.

Para resolver cada um destes casos, o AOM cria objetos num meta-modelo em vez de criar uma classe para cada objeto. Para isso lança mão de diversos padrões de projetos (GAMMA; et al, 1995), sendo que cada caso utiliza um grupo mais específico de padrões.

Os objetos compostos são resolvidos por padrões como Compósito, Interpretador e Construtor (GAMMA; et al, 1995). Com esses padrões é possível criar objetos complexos a partir da composição de objetos mais simples. Da mesma forma, isso permite que, em tempo de execução, seja definido como e quais objetos formam um objeto composto. Isto dá uma característica adaptativa a esta solução, permitindo uma flexibilidade maior aos sistemas modelados desta forma. Um exemplo poderia ser uma

montadora de automóveis, como mostrado na **Erro! Fonte de referência não encontrada.**. No exemplo, um automóvel é composto por peças, que podem ser motor, rodas, entre outros, ou seja, é uma composição de peças. A montadora junta as peças, bastando trocar ou reposicionar as peças para construir um modelo diferente. No exemplo, as peças formam com o automóvel o Compósito e o Montador representa o padrão Construtor. O padrão Interpretador não foi utilizado.

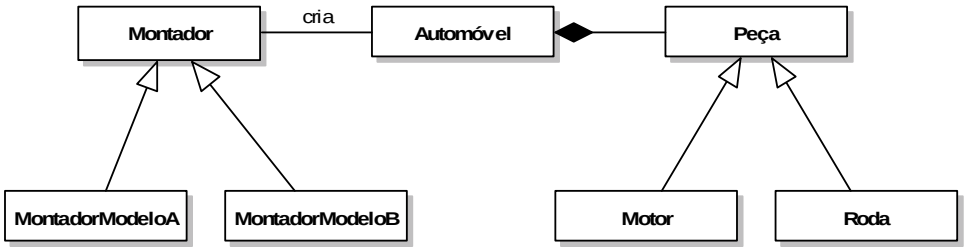

Figura 3 – Exemplo de Montadora de Automóveis

Outro caso ocorre quando mais de um algoritmo pode servir como solução para um problema. Isso pode acontecer em engenharia, quando uma fórmula é válida em condições específicas, existindo outra para outras condições. O AOM resolve este problema normalmente através do padrão Estratégia (GAMMA; et al, 1995), permitindo tratar cada algoritmo como um objeto e fazendo com que o algoritmo correto possa ser aplicado no momento oportuno. O exemplo da **Erro! Fonte de referência não encontrada.** mostra algoritmos de ordenação. Com poucos elementos a serem ordenados um algoritmo mais simples pode ser utilizado, como o *Bubble Sort*, e com um número grande de elementos já passa a ser necessário a utilização de algoritmos mais sofisticados como o *Quick Sort*.

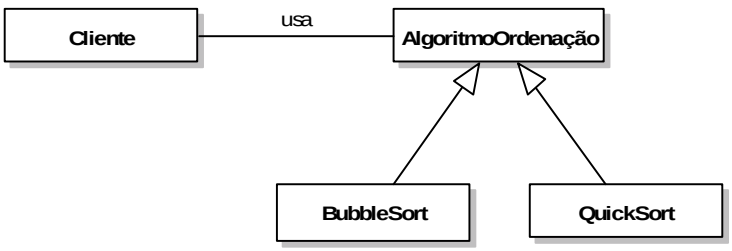

Figura 4 – Algoritmos de ordenação

Além destes problemas, juntam-se os casos de representação de regras e, de forma mais geral, da informação tratada pelo sistema modelado. Ao tratar as regras como objetos do sistema, as quais podem ser definidos através de uma meta-estrutura, o AOM permite uma flexibilidade nas regras do negócio que o sistema gerencia. Assim, o fluxo de informações pode ser alterado conforme novas necessidades do negócio. Da mesma maneira, a informação tratada pode ser alterada dentro de uma meta-estrutura que a defina. No casos das regras, o padrão mais utilizado é o Objeto Regra (ARSANJANI, 2000), enquanto as informações são meta-modeladas com os padrões Propriedade (FOOTE; YODER, 1998), Objeto Tipo (JOHNSON; WOOLF, 1998) e sua composição, o Quadrado de Tipos (YODER; et al, 2001). O exemplo da **Erro! Fonte de referência não encontrada.** apresenta apenas a representação da informação, sendo adaptado de (YODER; JOHNSON, 2002). Na **Erro! Fonte de referência não encontrada.**a, o automóvel é caracterizado da maneira natural, em que uma classe abstrata é generalizada para a criação de outros modelos, sendo os atributos definidos em suas classes. Na **Erro! Fonte de referência não encontrada.**b, o automóvel passa a ser formado por atributos determinados dinamicamente através do relacionamento com a classe Propriedade, o que permite que novas propriedades sejam inseridas a qualquer momento, sem necessidade de revisão do modelo. Ainda na **Erro! Fonte de referência não encontrada.**b, os tipos de automóveis são obtidos através de um relacionamento com a classe TipoAutomóvel, não sendo mais necessária a criação de generalizações para a classe Automóvel. Neste exemplo, o padrão Quadrado de Tipos não foi utilizado e a representação de regras foi deixada para ser mostrada mais adiante.

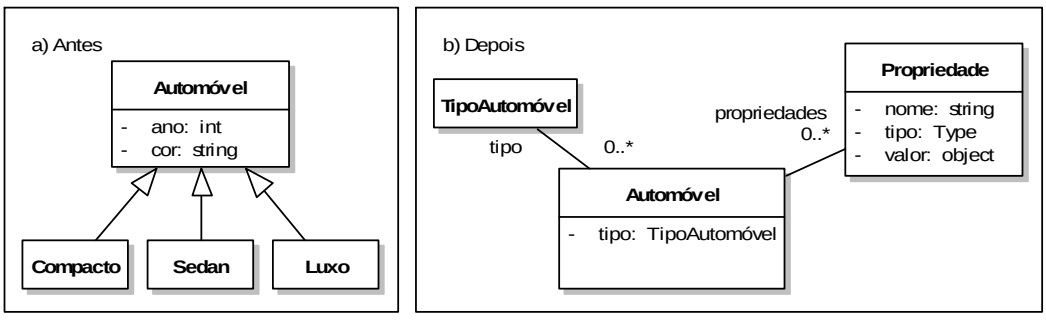

Figura 5 – a) Carro representado num modelo não adaptativo b) Carro num modelo adaptativo

Desta forma, uma solução AOM pode servir para um dos casos apresentados, ou um conjunto deles, o que faz com que não exista uma regra para sua aplicação. Desta forma, o AOM não é uma regra de como modelar, mas sim a utilização de meta-modelos para permitir aos sistemas possuírem características de adaptatividade. Assim, o AOM nem ao menos força a utilização dos padrões de projeto sugeridos, mas a sua utilização é recomendada já que eles já foram implantados com sucesso durante outras experiências. Por isso, é importante para um analista entender os padrões que pode utilizar e como utilizá-los para criar um modelo adaptativo de objetos mais adequado aos seus problemas. Por isso, cada padrão citado será apresentado e, em seguida, o seu papel no AOM será discutido.

Os primeiros padrões a serem apresentados (Compósito, Interpretador, Construtor e Estratégia) são comuns também em sistemas que não seguem a arquitetura do AOM. Este texto irá seguir (GAMMA; et al, 1995), em que estes padrões são discutidos.

Em seguida serão apresentados os padrões mais importantes do AOM (Objeto Tipo, Propriedade, Quadrado de Tipos e Objeto Regra). A importância destes padrões é tal que todo modelo AOM normalmente possui ao menos um deles presente.

3.1.1 COMPÓSITO

O Compósito (GAMMA; et al, 1995) é um padrão muito utilizado para construir árvores que representam hierarquias parte-todo. Esse padrão permite ao cliente tratar o objeto todo, bem como partes dele da mesma forma.

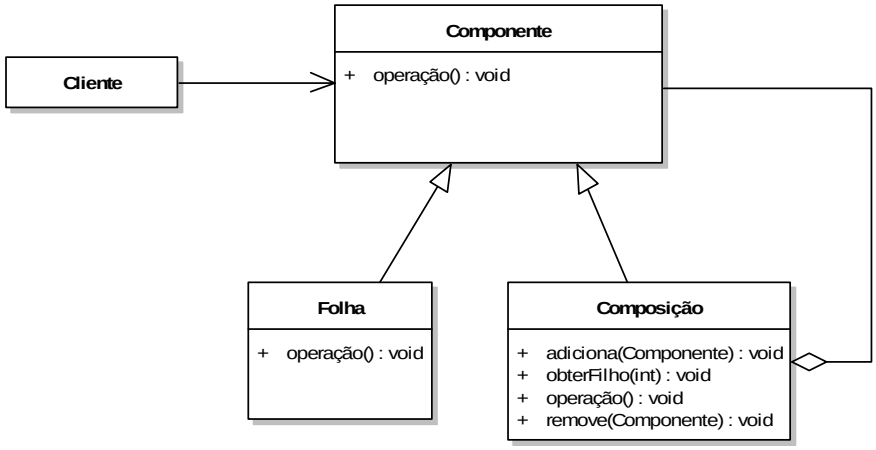

Figura 6 - Compósito

Como se vê na **Erro! Fonte de referência não encontrada.**, o padrão é formado por três classes: o componente, e seus filhos, a folha e a composição. Note que a composição é, além de filho do componente, formado por componentes, que é o que define o padrão.

No AOM, o Compósito tem o papel de permitir a construção de estruturas dinâmicas de tipos e regras.

3.1.2 INTERPRETADOR

O Interpretador (GAMMA, 1995, pp. 243-255) é um padrão em que define a representação de uma linguagem e um interpretador para esta representação, de forma que sentenças desta linguagem possam ser interpretadas.

Este padrão, diferente de muitos outros existentes, não possui uma estrutura básica, pois cada linguagem que é representada pode possuir um modelo diferente. Mesmo dentro de uma linguagem, diferentes estruturas podem ser montadas. Um exemplo de interpretador para uma expressão regular é mostrado na **Erro! Fonte de referência não encontrada.**. A figura apresenta como centro a Expressão Regular, que pode ser do tipo Literal, Seqüencial, Alternativa ou Repetitiva, sendo que a expressão seqüencial e a alternativa são compostas por duas expressões regulares (seqüenciais ou alternadas) e a repetitiva pela expressão regular que deve ser repetida.

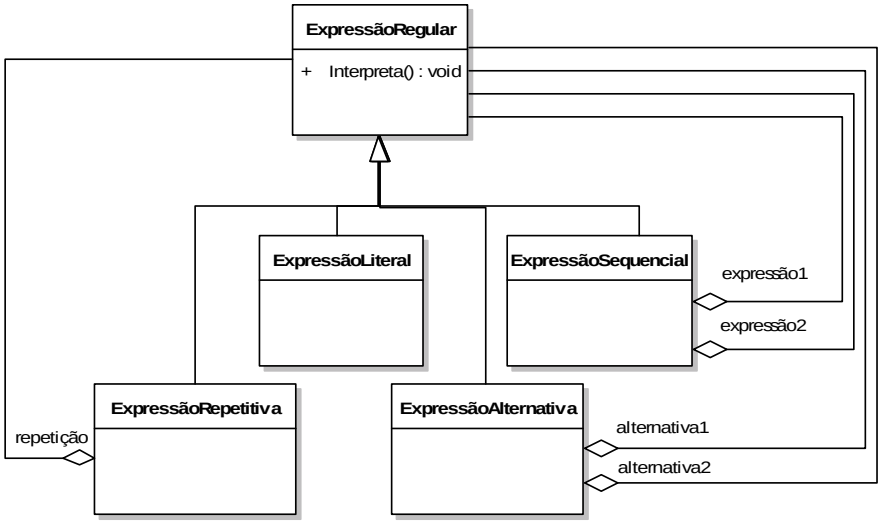

Figura 7 – Interpretador (exemplo para Expressão Regular)

No AOM este padrão é importantíssimo, pois ele permite a personalização do comportamento dos objetos. Com a possibilidade de fazer a interpretação de uma linguagem em tempo de execução, é possível definir expressões em tempo de execução, o que permite características adaptativas serem realizadas.

3.1.3 CONSTRUTOR

Esse padrão de projeto tem a intenção de "separar a construção de um objeto complexo de sua representação, de forma que o processo de construção possa criar diferentes representações" (GAMMA; et al, 1995, pp. 97-106).

O Construtor consiste numa classe que serve como padrão para construção de objetos ou composições que formam um objeto, de forma a simplificar e padronizar a sua construção.

Na **Erro! Fonte de referência não encontrada.**, o construtor é mostrado. É possível verificar a existência de uma classe Diretor, que delega a construção de partes de um produto a cada construtor necessário para sua realização. Desta forma o Produto é criado a partir de diversos Construtores, que são organizados ou dirigidos pelo Diretor.

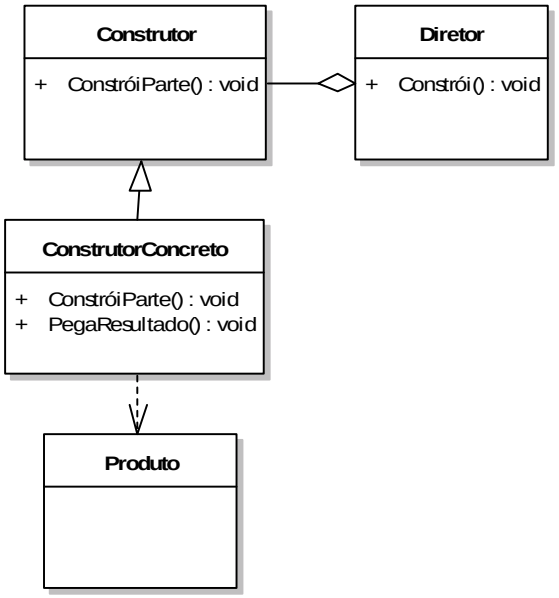

Figura 8 – Construtor

No AOM, este padrão costuma ser utilizado para construir as estruturas a partir dos dados extraídos do meta-modelo.

3.1.4 ESTRATÉGIA

Estratégia é um padrão que "define uma família de algoritmos, encapsula cada um, e faz seu intercâmbio" (GAMMA; et al, 1995, pp. 315-323). Com isso, esse padrão permite que o algoritmo a ser utilizado possa ser definido em tempo de execução, de acordo, por exemplo, com o contexto em que está inserido.

O padrão Estratégia consiste em criar uma interface, ou uma classe abstrata, de como deve ser a chamada do algoritmo, e cada algoritmo diferente é encapsulado em uma classe diferente. Assim, o padrão Estratégia permite que o contexto defina qual o melhor algoritmo a ser executado em tempo de execução. A estrutura básica do padrão Estratégia é mostrada na **Erro! Fonte de referência não encontrada.**. O Cliente pode escolher qual Estratégia adotar, sendo a sua interface definida na classe Estratégia. O desenvolvedor pode definir quantas estratégias de execução de algoritmos forem necessárias, bastando criar as classes filhas (representado por EstratégiaConcreta).

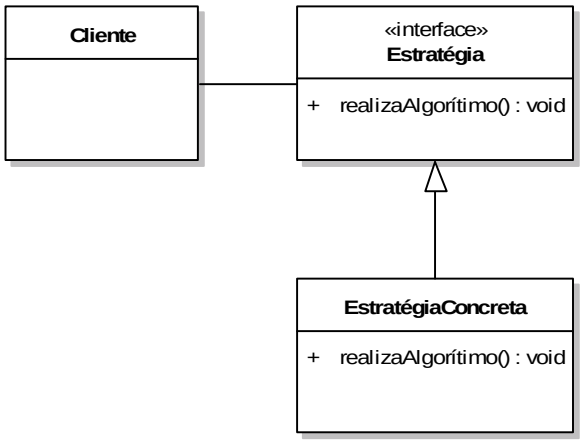

Figura 9 – Estratégia

Para o AOM, este padrão tem a mesma importância do Interpretador: ele permite a personalização do comportamento do objeto. É, porém, um pouco diferente na questão da construção, visto que os algoritmos, em princípio, são definidos em tempo de projeto e não de execução.

3.1.5 OBJETO TIPO

Um dos padrões mais importantes da arquitetura do AOM é o Objeto Tipo, também conhecido como *Power Type*, ou *Item Descriptor* ou *Instance of an Instance*. Ele tem a função de "desmembrar as instâncias de suas classes, de modo que essas classes possam ser construídas como instâncias de outra classe" (JOHNSON; WOOLF, 1998 pp. 47). Desta forma, novas classes podem ser criadas dinamicamente, permitindo ao sistema ter sua própria regra de checagem de tipo.

O objeto tipo é utilizado para permitir que o tipo dos objetos adaptativos possa ser determinado e até mesmo definido em tempo de execução. Com o objeto tipo é possível que um sistema crie novos tipos de objetos conforme a necessidade criada por seus usuários. Assim, se em desenvolvimento uma informação pode não ser bem definida, pode-se definir um meta-modelo em que o usuário defina como são as informações que o sistema deve tratar.

Como mostrado na **Erro! Fonte de referência não encontrada.**, o Objeto Tipo é formado por duas classes: uma que representa o objeto (Classe); e outra que representa o tipo deste objeto (TipoClasse).

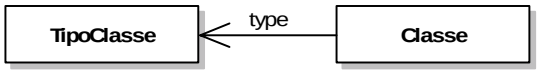

Figura 10 – Objeto Tipo

Desta forma, o tipo de um objeto passa a ser uma variável, atribuída com uma instância de "TipoClasse". Assim uma instância de "TipoClasse" representa o tipo de uma ou mais intâncias de "Classe", e estes tipos do objeto podem ser controlados pelo programador, que tem um grau de liberdade maior para criar Tipos dinamicamente, como representado na **Erro! Fonte de referência não encontrada.**, em que instâncias de tipos são representadas pelas entidades "ObjetoTipo1" e "ObjetoTipo2" e se apresentam duas instâncias de objetos "Objeto1" e "Objeto2" ambas do tipo 1, sendo o tipo 2 apresentado somente a título de exemplo.

No AOM, esse padrão é essencial, pois como os dados são interpretados como meta-dados, o tipo de informação deve ser inferido em tempo de execução. Utilizando o Objeto Tipo é possível descobrir, em tempo de execução, o tipo do objeto e aplicar, desta forma, o tratamento correto no contexto em que estiver, ou mesmo criar o objeto com o tipo definido em tempo de execução.

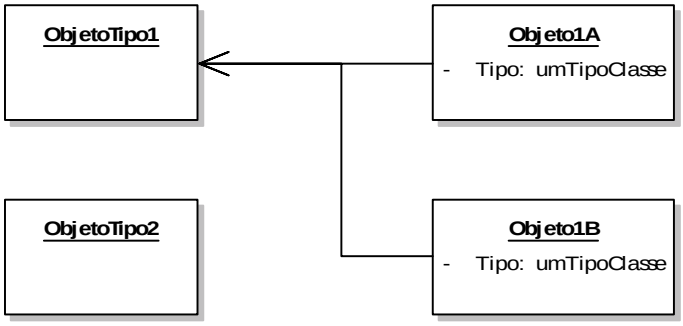

Figura 11 – Objeto Tipo (exemplo)

3.1.6 PROPRIEDADE

Também conhecido como *Attributes*, *Annotations*, *Dynamic Attributes*, *Dynamic Variables*, *Variable State*, *Dynamic Slots* e *Property List*, o padrão Propriedade (FOOTE; YODER, 1998) é outro de grande importância para o AOM.

A Propriedade é um padrão similar ao Objeto Tipo, mas que, em vez de tratar classes como instâncias, tratam as propriedades que pertencem aos objetos. Ele faz com que as propriedades de um objeto saiam da definição da classe do objeto e passem a instâncias de uma classe genérica denominada Propriedade, como mostrado na **Erro! Fonte de referência não encontrada.**.

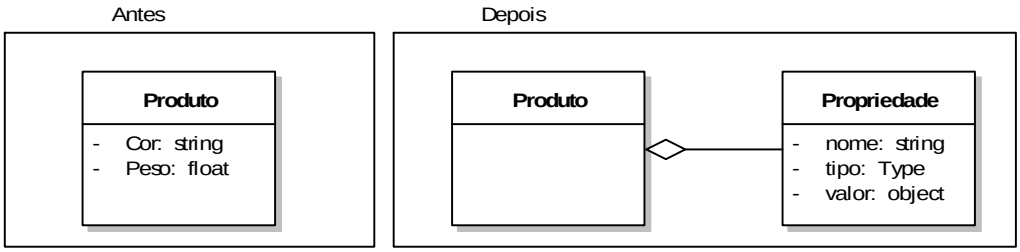

Figura 12 – Propriedade

Desta forma um produto não precisa, necessariamente, possuir as mesmas propriedades ou atributos que outro, apesar de pertencerem a uma mesma classe. Note que, apesar de perfeitamente possível, o tipo do objeto (no caso do produto), não é garantido no modelo.

Este padrão é essencial no AOM quando é necessário tratar propriedades como meta-dados. Propriedades como meta-dados permitem que o usuário de um sistema possa definir quais os campos ou atributos que uma entidade necessita, fazendo com que o sistema se torne mais aderente às suas necessidades.

3.1.7 QUADRADO DE TIPOS

Com a utilização dos padrões Objeto Tipo e Propriedade, a junção dos dois padrões em torno de uma classe, uma Entidade, é comumente utilizada. Assim, uma Entidade pertence a um Tipo de Entidade, e possui uma ou mais propriedades.

Este padrão de projeto, denominado Quadrado de Tipos (YODER; et al, 2001), vai um pouco além, definindo ainda que as propriedades tenham um tipo designado pelo tipo das Entidades que a contêm, conforme a **Erro! Fonte de referência não encontrada.**.

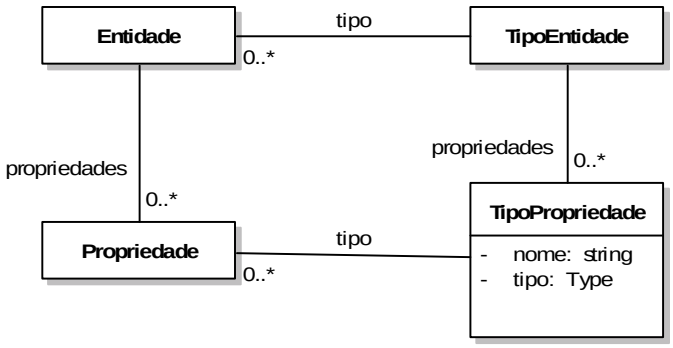

Figura 13 – Quadrado de Tipos

Esta estrutura é muito utilizada no AOM. Ela faz com que o tipo se relacione às propriedades de forma que todo objeto de um tipo tem um conjunto de propriedades definidos nesta estrutura. Com isso, o padrão Propriedade junta-se ao padrão Objeto Tipo dando mais consistência a este padrão, o qual passa a ser mais completo por ligar as propriedades de um objeto ao tipo a que ele pertence.

3.1.8 OBJETO REGRA

O objetivo deste padrão é "Fazer o projeto e construção de processos de negócios informatizados extensíveis e adaptáveis, sem fragilizá-las a mudanças indesejáveis, fazendo com que as regras sejam a eles aplicáveis." (ARSANJANI, 2000, pp. 3).

Isso é feito criando um arcabouço em que nos permite criar regras, inserindo condições e ações a serem executadas, de acordo com a necessidade do negócio.

A estrutura desse padrão é um pouco mais complexa, já que ele serve como um Mediador (GAMMA; et al, 1995) entre dois outros padrões: Ação (*Action* ou *Command*) (GAMMA; et al, 1995) e Assessor (ARSANJANI, 2000). Respectivamente, estes padrões fazem com que ações e assertivas sejam tratadas como objetos, fazendo com que novas ações ou assertivas possam ser criadas dinamicamente. Uma possível estrutura para estes padrões é mostrada nas figuras **Erro! Fonte de referência não**

encontrada. e **Erro! Fonte de referência não encontrada.**, como proposto em (ARSANJANI, 2000).

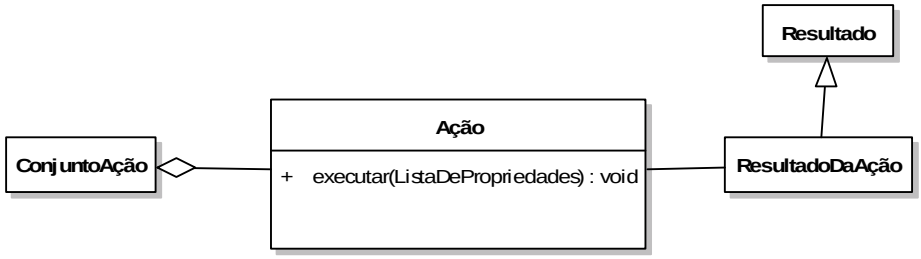

Figura 14 – Ação

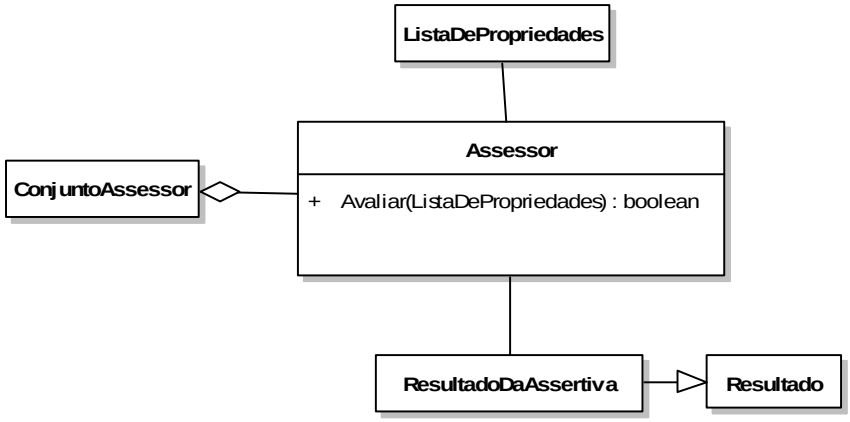

Figura 15 – Assessor

O padrão Objeto Regra é, então, formado por assertivas e ações realizadas através deste padrão. Caso as assertivas se confirmem, o Objeto Regra encarrega-se de executar cada uma das ações a que está relacionado. A estrutura, apesar de simples se vista como na **Erro! Fonte de referência não encontrada.**, não demonstra como as assertivas são feitas, nem como as ações são executadas, ficando a cargo do objeto e da construção do padrão.

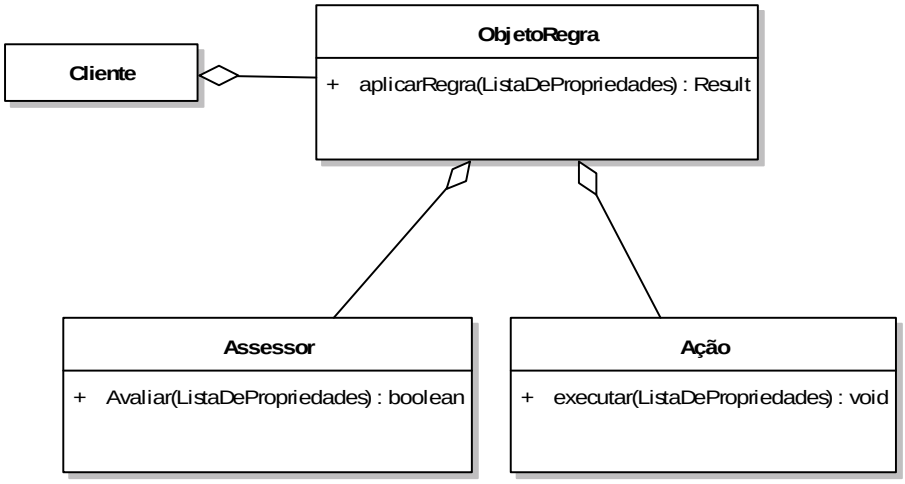

Figura 16 – Objeto Regra

O padrão é importante para o AOM, pois é a partir dele que as regras são personalizadas, e o comportamento do sistema pode ser adaptado. É interessante notar que este padrão depende, mesmo que informalmente, de outro padrão importante do AOM: a Propriedade. Isso porque as ações irão incidir sobre elas, provocando a alteração de seus valores. Se o conjunto de regras não puder alterar o estado dos objetos em um sistema, estes perdem a sua funcionalidade.

3.2 TRABALHOS CORRELATOS

Alguns trabalhos podem aumentar o poder dos modelos adaptativos, ora permitindo um maior desempenho, ora aumentando as possibilidades de construção. Este é o caso do espaço de nome (*namespace – OMG, 2005*) *Reflection.Emit*, no Microsoft .NET Framework 2.0, e o Javassist (CHIBA, 1998), uma Interface de Programação de Aplicativos - API (*Application Programming Interface*) - para Java.

O espaço de nome *Reflection.Emit* é uma das novidades da segunda versão do Microsoft .NET Framework. Este espaço de nome apresenta a possibilidade de criar, em tempo de execução, os chamados métodos dinâmicos, permitindo estender o comportamento de um objeto de acordo com o meio em que ele estiver atuando. A grande vantagem desta solução é o desempenho, pois o código descrito já deve estar na linguagem intermediária da máquina virtual da Microsoft, o MSIL (*Microsoft Intermediate Language*), não necessitando do processo de compilação. Como desvantagem, pode-se dizer na

necessidade de se tratar diretamente com o MSIL, o que, por vezes, pode não ser amigável. Entretanto, esta solução traz uma quantidade enorme de possibilidades, já que a geração automática de código pode ser aperfeiçoada.

O Java, diferentemente do .NET, apresenta o código intermediário de maneira mais acessível ao desenvolvedor. O Javassist é uma API que procura facilitar o trabalho de acesso ao código intermediário, que é armazenado nos arquivos de extensão *class*, permitindo inclusive que uma classe inteira seja criada ou reescrita em tempo de execução. Em relação ao Reflection.Emit ele possui a desvantagem de não poder tratar objetos separadamente. Mas não deixa de ter seu valor, pois não só permite a criação de novos métodos, como permite a reestruturação de um sistema em tempo de execução.

4 TECNOLOGIA ADAPTATIVA

Este capítulo apresenta a tecnologia adaptativa desenvolvida no Laboratório de Tecnologia Adaptativa da Universidade de São Paulo, fazendo um resumo do seu histórico, apresentado inicialmente em (PISTORI; 2002), e atualizando-o com os acontecimentos mais recentes. Logo a seguir, o texto apresenta o dispositivo adaptativo, responsável por tornar um formalismo não adaptativo em adaptativo e exemplifica com a aplicação do dispositivo nos autômatos de pilha estruturados. Por fim, algumas considerações sobre a tecnologia são realizadas para discutir possíveis vantagens e desvantagens da sua aplicação.

4.1 HISTÓRICO

A Tecnologia Adaptativa tem origem em (NETO, 1994) quando surgem os Autômatos Adaptativos através de uma extensão do formalismo dos autômatos de pilha estruturados (NETO; MAGALHÃES, 1981a; NETO; PARIENTE; LEONARDI, 1999; NETO, 1987). O texto que se segue tem forte influência de (PISTORI; 2002), primeiro trabalho sobre o histórico da tecnologia adaptativa, e atualiza o histórico com os últimos trabalhos realizados até a data do presente trabalho.

A tecnologia adaptativa tem por objetivo dar maior poder de expressividade a formalismos com poucas modificações, sem que modelos feitos de acordo com o formalismo subjacente às mudanças sejam perturbados (NETO, 2001).

Após o trabalho dos autômatos adaptativos, muitos outros o seguiram, como a criação dos *statecharts* adaptativos (JUNIOR, 1995; SANTOS, 1997), das gramáticas adaptativas (IWAI, 2000; IWAI; NETO, 2000), das redes de Markov adaptativas (BASSETO; NETO, 1999), das árvores de decisões adaptativas (PISTORI; NETO, 2002), das redes Neurais adaptativas (ROCHA, 2001) e das tabelas de decisões adaptativas (NETO, 2001). Nestes trabalhos e em outros surgiram aplicações em diferentes contextos como em compiladores (LUZ; NETO, 2003), robótica (SOUZA; HIRAKAWA; NETO, 2004), reconhecimento de imagens (PISTORI; NETO, 2004;

PISTORI; et al, 2006), síntese de voz (ZUFFO; PISTORI, 2004), e na geração de música por computação (BASSETO;NETO, 1999).

O segundo trabalho surgiu (JUNIOR, 1995), tratando da especificação e projeto de sistemas reativos complexos. Neste trabalho o autor define os *Statecharts* Adaptativos e cria a primeira ferramenta de edição e simulação de dispositivos adaptativos denominada STAD (*STatecharts* ADaptativos). Utilizando redes de Petri, deu-se seqüência a este trabalho (SANTOS, 1997) para incorporar a idéia de sincronização de processos e criar os *Statecharts* Adaptativos Sincronizados (SAS). Neste trabalho, o STAD foi substituído pelo SAS (ou STAD-S), que trabalhava sobre o formalismo com a sincronização de processos.

Logo apareceu a primeira experiência prática com a tecnologia adaptativa (BASSETO; NETO; 1999). Utilizando redes de Markov Adaptativas, foi criado um compacto gerador de músicas denominado LASSUS. Baseado em poucas regras de criação, cada execução do gerador traz uma nova composição sonora que apresenta uma grande qualidade estética.

Visto que os primeiros estudos basearam-se no reconhecimento de linguagem, não demorou a surgir o primeiro formalismo gerador de linguagem. As Gramáticas Adaptativas (IWAI, 2000; IWAI; NETO, 2000) são apresentadas junto ao teorema da equivalência expressiva entre as gramáticas e os autômatos adaptativos.

Em 2000 é verificado o poder computacional dos autômatos adaptativos, comparado ao da máquina de Turing (ROCHA; NETO, 2000). No ano seguinte os conceitos fundamentais que atingem as tecnologias adaptativas existentes são consolidados e generalizados (NETO, 2001), com a introdução dos dispositivos adaptativos baseados em regras. Neste trabalho são apresentadas também as tabelas de decisões adaptativas.

Outro trabalho (ROCHA, 2001) utiliza um modelo de autômatos adaptativos como uma rede neural, para a modelagem de aprendizado dentro de um dispositivo de busca de soluções por máquina adaptável.

As árvores de decisões adaptativas são introduzidas em (PISTORI; NETO, 2002) como um forte aliado para a tomada de decisões por computação. Neste trabalho o autor faz uma comparação da solução adaptativa com outras soluções feitas para tomada de decisão, obtendo resultado satisfatório.

Em seguida, foi criado um programa para o projeto, construção e teste de autômatos adaptativos, denominado ADAPTOOLS (PISTORI, 2003; PISTORI; NETO, 2003). Apesar de outros formalismos adaptativos já terem apresentado ferramentas para seu desenvolvimento, o formalismo precursor só foi ter uma ferramenta quase uma década após seu trabalho precursor (1994).

Em outro trabalho (SOUZA; HIRAKAWA; NETO, 2004), a adaptatividade é estudada no mapeamento de área realizado por um robô. O robô é governado por um autômato adaptativo, e conforme alterações em seus sensores, este autômato é atualizado.

Num trabalho recente na área, os autômatos adaptativos foram utilizados para gerar sintetizadores de voz (ZUFFO; PISTORI, 2004). O que o autor espera é que diferentes sons possam ser gerados para uma mesma sílaba com o auxílio dos autômatos adaptativos. Assim, o autômato se especializa para cada contexto, e as sílabas podem ser apresentadas de maneira adequada ao contexto.

A última publicação feita até a data do presente trabalho (PISTORI; et al, 2006), utiliza a tecnologia adaptativa para o reconhecimento de imagens para auxilio a deficientes físicos. O trabalho é uma continuação de (PISTORI; NETO; 2004), em que a tecnologia adaptativa é testada para o reconhecimento de sinais de surdos-mudos.

O último acontecimento importante para a tecnologia adaptativa foi o I Workshop de Tecnologia Adaptativa, realizado em São Paulo no mês de janeiro de 2007. Contando com a presença da maioria dos contribuidores da tecnologia adaptativa, o evento permitiu a discussão de alguns dos principais aspectos da tecnologia adaptativa, como trabalhos em andamento, aplicações da tecnologia adaptativa e rumos que os seus estudos devem tomar.

4.2 DISPOSITIVO ADAPTATIVO

O texto que segue baseia-se integralmente em (NETO, 2001), em que o dispositivo adaptativo foi generalizado a partir de diversas experiências da aplicação da tecnologia adaptativa a formalismos não adaptativos.

O dispositivo adaptativo AD é definido como sendo um dispositivo que segue um comportamento de um dispositivo subjacente até que alguma ação adaptativa não nula seja realizada, alterando o conjunto de regras e, dessa forma, o dispositivo. A partir de então, o dispositivo se comportará de uma nova maneira, de acordo com as novas regras, até que outra ação adaptativa não nula seja executada.

Desta forma, o AD pode ser visto na forma $AD_k=(C_k, AR_k, S, c_k, A, NA, BA, AA)$, em que k corresponde a um passo de computação iniciando em 0, correspondente ao início do dispositivo, e incrementa de 1 quando uma ação adaptativa não nula é executada, sendo que:

- $AD=(ND_0, AM)$ é um dispositivo adaptativo sobre um dispositivo subjacente inicial ND_0 com um mecanismo adaptativo AM.
- ND_k é o dispositivo não adaptativo subjacente em algum passo de operação k, definido por um conjunto de regras não adaptativas NR_k.
- C_k é o conjunto de todas as configurações possíveis para ND num passo k, sendo c_k a configuração inicial no passo k.
- ε (cadeia vazia) é o elemento neutro das operações união e intersecção em um conjunto qualquer. O uso de ε significa que, naquela circustância, nenhum elemento válido do conjuntoestá sendo usado.
- S é o conjunto finito e fixo de todos os eventos que são estímulos de entrada válidos para o AD ($\varepsilon \in S$).
- $A \subseteq C$ é o subconjunto das configurações de aceitação. Seu complemento é o subconjunto das configurações de falha F.

- *BA* e *AA* são conjuntos de funções adaptativas, ambas contendo a ação nula ($\varepsilon \in BA \cap AA$). BA corresponde ao conjunto das ações a serem executadas antes da aplicação da regra, e AA ao conjunto das ações a serem executadas após a aplicação da regra.
- $w=w_1w_2...w_n$, $i=1,...,n$ é uma cadeia de estímulos de entrada, em que $w_i \in S - \{\varepsilon\}$.
- *NA* é o conjunto finito e fixo de todos os símbolos de saída permitidos pelo *AD*.
- AR_k é o conjunto de regras adaptativas, onde $AR_k \subseteq BA \times C \times S \times C \times NA \times AA$. Uma regra adaptativa ar_k, do conjunto AR_k tem a forma $ar_k=(ba,c_i,s,c_j,z,aa)$, significando que a ação adaptativa $ba \in BA$ é executada em resposta a um estímulo *s*. No caso de essa ação apagar a regra que está sendo executada, a execução é abortada. Caso contrário, a ação não adaptativa $nr=(c_i,s,c_j,z)$ é executada e, por fim, a ação adaptativa $aa \in AA$ é executada.
- *AR* é o conjunto de todas as regras adaptativas possíveis para o *AD*.
- *NR* é o conjunto de todas as regras não-adaptativas subjacentes possíveis para o *AD*.
- $AM \subseteq BA \times NR \times AA$, definido para um *AD*, é o mecanismo adaptativo a ser aplicado em qualquer passo de operação *k* para cada regra em $NR_k \subseteq NR$. *AM* deve ser tal que ele opere como uma função quando aplicado a qualquer subdomínio $NR_k \subseteq NR$. Isso determinará um par de ações adaptativas que poderão ser anexadas em qualquer regra não adaptativa.

Um dispositivo adaptativo inicia-se uma configuração inicial, seguida pela aplicação da cadeia de estímulos de entrada. Em seguida cada estímulo é analisado, sendo aplicada a regra apropriada para cada estímulo. Neste momento, três situações distintas podem ocorrer:

- Não existe regra para responder ao estímulo e a cadeia deve ser rejeitada.
- Existe uma única regra a ser aplicada para o estímulo e o próximo contexto é bem definido.

- Existe mais de uma regra para o estímulo e todas as regras devem ser aplicadas paralelamente, como de modo usual em um dispositivo não determinístico. Em ambientes sem paralelismo isso deve ser adequadamente simulado.

Na aplicação de uma regra, primeiramente é executada a ação adaptativa anterior *ba* e, se esta implicar em exclusão da regra que está sendo imposta, a aplicação da regra é abortada e volta-se ao passo de verificação das regras que devem ser aplicadas. Com a execução da ação adaptativa o *AD* passa de uma configuração inicial para uma intermediária.

Em seguida é aplicada a regra não adaptativa, fazendo com que o *AD* passe para uma segunda configuração intermediária. Por fim, a ação adaptativa posterior *aa* é executada, chegando ao autômato final, ou seja, o autômato após a aplicação da regra adaptativa. Esses passos são repetidos até o final dos estímulos da cadeia de entrada, até que se chegue a uma configuração de aceitação ou a uma falha, com a parada do dispositivo.

4.3 EXEMPLO DE DISPOSITIVO ADAPTATIVO

Uma maneira de exemplificar o dispositivo adaptativo é analisar como ele é utilizado no trabalho original, ou seja, como o dispositivo é utilizado nos autômatos de pilha estruturados para criar autômatos adaptativos.

Os autômatos de pilha estruturados são vistos como conjuntos de autômatos de estados finitos interligados através da pilha. Dessa forma, o autômato se simplifica em relação aos autômatos de pilha convencionais, pois a pilha tem função única bem definida: armazenar o endereço de retorno da submáquina. Além disso, as submáquinas que formam o autômato são mais simples, tornando este tipo de construção de melhor entendimento.

O autômato de pilha estruturado possui dois tipos de transições: as internas, em que há o consumo da cadeia de entrada; e as externas que são responsáveis pela chamada e retorno das submáquinas.

Em (NETO; 1993), os autômatos de pilha estruturados são definidos como uma 8-tupla $M=(Q, A, \Sigma, \Gamma, P Z_0, q_0, F),$ em que: Q é um conjunto de estados de M, Σ é o alfabeto de

entrada, Γ é o alfabeto de pilha que contém Z_0, o marcador de pilha vazia. O estado inicial q_0 pertence ao conjunto Q, e o conjunto F é subconjunto de Q, denotando os estados finais do autômato. O conjunto A é o conjunto das submáquinas a_i, $i=1,...,n$ implementando M da forma:

$$a_i = (Q_i, \Sigma, P_i, q_{0,i}, F_i)$$

em que $\{ Q_1, ..., Q_n\}$ e $\{ P_1, ..., P_n\}$ são partições de Q e P. F_i é um subconjunto de Q_i, representando os estados de retorno da submáquina a_i, e $q_{0,i}$ é o estado de entrada da submáquina.

Cada transição p do conjunto P é da forma:

$$(\gamma g, e, s\alpha) : \rightarrow (\gamma g', e', s'\alpha)$$

À esquerda é representada a situação da máquina antes da aplicação de p e à direita é representada a situação após a aplicação de p. g e g' representam o conteúdo no topo da pilha, e e e' são estados da máquina, s e s' são átomos do conjunto Σ. Os meta-símbolos γ e α representam, respectivamente, as partes da pilha e da cadeia que não são alteradas pela produção.

Além disto, algumas regras são definidas:

- Quando e e e' pertencem à mesma submáquina, a transição p representa uma transição interna da submáquina e g e g' devem ser vazios.
- Quando g' não é vazio, g, s e s' devem ser vazios, e a transição p representa uma chamada de submáquina. Neste caso, e' deve ser um estado de entrada de submáquina e g' o estado de retorno da submáquina.
- Quando g não é vazio, g', s e s' devem ser vazios e a transição p representa um retorno de submáquina, e o estado de retorno deve ser o estado armazenado na pilha na chamada da submáquina.

No mesmo artigo (NETO; 1993), ao aplicar o dispositivo adaptativo, tem-se algumas pequenas alterações no formalismo acima.

Uma transição passa a ser da forma

$(\gamma p, e, s\alpha) : A, \rightarrow (\gamma p', e', s'\alpha), B$

Em que *A* e *B* são listas de chamadas a funções adaptativas executadas, respectivamente, antes e depois da aplicação da regra.

Uma Função Adaptativa é definida neste artigo como uma 9-tupla: *FA* = (*F*, *P*, *V*, *G*, *C*, *E*, *I*, *A*, *B*) na qual:

- *F* é o nome da função adaptativa
- *P* é a lista de parâmetros formais
- *V* é o conjunto de variáveis da função
- *G* é o conjunto de geradores da função
- *C* é o conjunto de padrão de produção de procura e inspeção
- *E* é o conjunto de padrão de produção de procura e remoção
- *I* é o conjunto de padrão de produção de procura e inserção
- *A* é uma ação adaptativa a ser executada antes de *F*
- *B* é uma ação adaptativa a ser executada após *F*

As variáveis de ***V*** são nomes simbólicos usados para guardar valores resultantes de alguma ação adaptativa que efetua consulta. Os geradores em ***G*** são nomes simbólicos que referenciam novos valores a cada uso, podendo ser preenchidos por um valor apenas durante a execução de uma função adaptativa.

Uma ação de consulta de ***C*** é uma ação que realiza uma pesquisa no conjunto de regras por um determinado padrão sem realizar qualquer alteração no conjunto. Uma ação de remoção de ***R*** remove as regras do conjunto de regras que seguem um dado padrão. Já a ação de inserção de ***I*** adiciona uma regra com um padrão especificado no conjunto de regras.

É importante frisar novamente que, por executar uma função adaptativa antes da aplicação da regra, a regra pode ser excluída por ela. Neste caso, a aplicação da regra é

cancelada, e outra regra deverá ser executada para a transição em que se encontra o autômato.

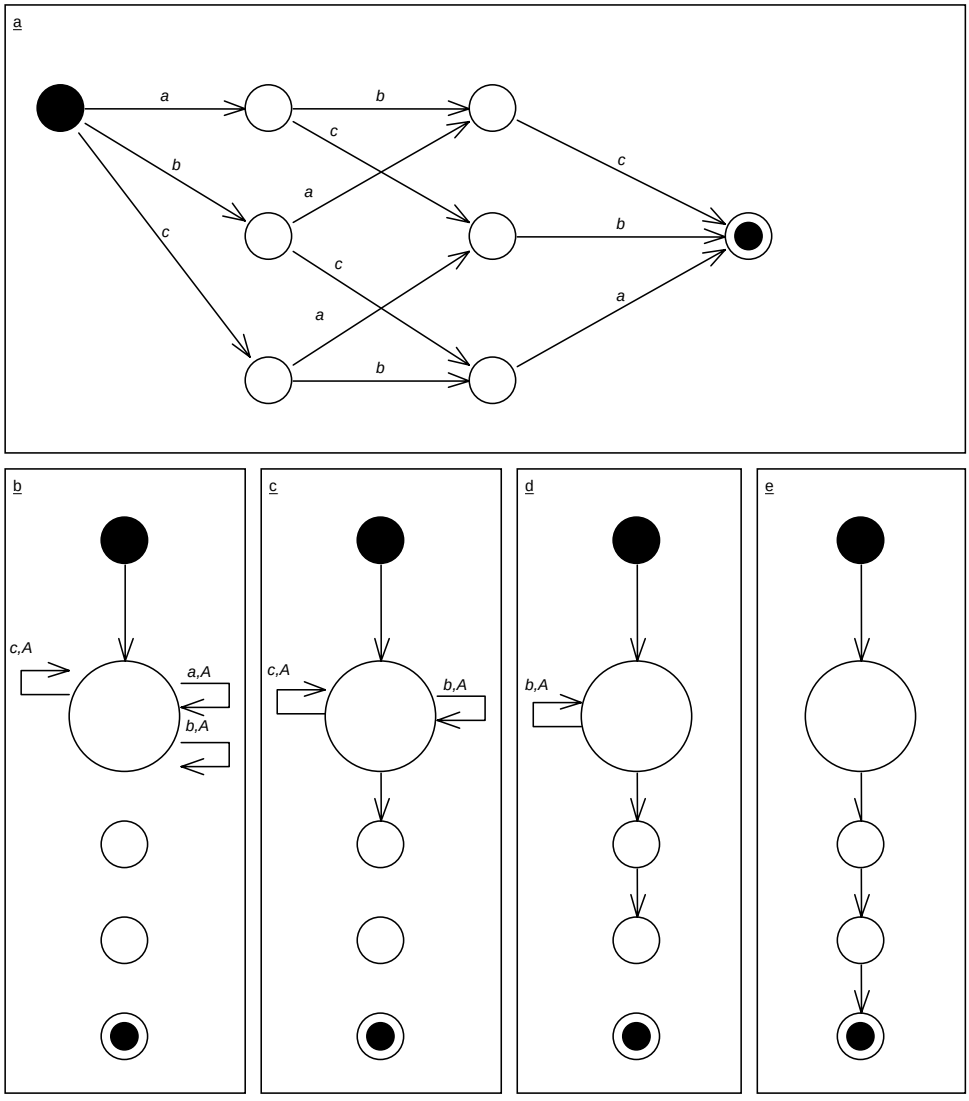

Figura 17 – a) Autômato Finito reconhecedor de quaisquer combinações formadas por um *a*, um *b* e um *c*; b) Autômato Adaptativo (AA) reconhecedor de um *a*, um *b* e um *c*, com ação adaptativa A, que retira a transição utilizada e cria uma transição em vazio em direção ao estado final; c) AA após reconhecer *a*; d) AA após reconhecer *a* e *c*; e) AA após reconhecer *a*, *b* e *c*.

Com isso, o modelo de autômato de pilha estruturado torna-se dinâmico, ou adaptativo, e a cada ação adaptativa executada ele se altera. Um bom exemplo do poder do autômato adaptativo é o reconhecedor de cadeias formadas por uma letra *a*, uma *b* e uma *c* em qualquer ordem. Esse tipo de reconhecimento é importante, por exemplo, para

linguagens naturais, em que a ordem dos elementos nem sempre é fixa. Com autômatos de estado finito, uma possível solução pode ser como a mostrada na Figura 17a. Com autômatos adaptativos, a solução pode ser simplificada como na Figura 17b, adaptada de (NETO, 2000). O autômato, também conhecido como Margarida, devido a seu formato, apresenta uma transição. Ou pétala, para cada letra que precisa ser reconhecida possuindo uma ação adaptativa *A*. Conforme as letras vão sendo consumidas, as ações adaptativas fazem com que a transição utilizada seja retirada, ou seja, uma pétala cai, e uma nova transição em direção ao estado final é adicionada, com a formação do caule. Desta forma, o autômato só chega ao estado final se todas as pétalas caírem. É clara a simplicidade trazida pelo dispositivo adaptativo neste exemplo, com a drástica redução no número de estados e transições.

4.4 ALGUMAS DIFICULDADES DA TECNOLOGIA ADAPTATIVA

Um problema do dispositivo adaptativo, tanto nesta aplicação como em outras em que foi aplicado, é como mostrar o comportamento dinâmico em um diagrama. Em geral, a execução do modelo é necessária para que seu entendimento seja claro, como feito na Figura 17c, Figura 17d e Figura 17e, em que alguns passos de execução do reconhecedor citado são mostrados.

A Tecnologia Adaptativa permite a criação de novas soluções para alguns problemas complexos com o aumento da expressividade dos formalismos em que é aplicada, sem aumento significativo da sua complexidade.

Apesar do ganho de expressividade, a tecnologia adaptativa tem alguns problemas que devem ser destacados, para que não haja empecilhos à sua aplicação na teoria dos objetos. A primeira grande questão é a representação do dinamismo que a tecnologia adaptativa provê. Decorrente desta dificuldade na representação, que parece ainda não ter sido resolvida, a criação de uma solução adaptativa é dificultada, já que o desenvolvedor tem mais dificuldades em vislumbrar a forma como o modelo funciona.

Uma necessidade que a aplicação do dispositivo adaptativo gera é a necessidade da alteração na modelagem do formalismo subjacente para que ela represente as

características dinâmicas. Entretanto, a representação do dinamismo é dificultada pela natureza da forma como os modelos dos formalismos normalmente são exibidos: diagramas gráficos, de natureza estática. A natureza da exibição dos modelos dificulta a representação de características dinâmicas mesmo com a inserção de novas características nos diagramas. Desta forma, na maioria dos exemplos de aplicação do dispositivo adaptativo recorre-se a uma seqüência temporal de diagramas para evidenciar o comportamento dinâmico do modelo, como realizado no exemplo do reconhecedor de um caractere *a*, um caractere *b* e um caractere *c*.

Este problema da representação traz embutido um outro problema: a criação de uma solução adaptativa pode tornar-se menos intuitiva conforme a representação do dinamismo se torne menos expressiva. Em outras palavras, a representação do dinamismo tem de ser clara e objetiva para que o desenvolvedor possa vislumbrar e até aprimorar o comportamento adaptativo sugerido pelo modelo.

Por isso a representação do dinamismo deve ter um estudo cuidadoso, sendo que a simples adição de novas características nos diagramas existentes para o formalismo precursor do formalismo adaptativo pode não ser suficiente para que eles fiquem adequados ao novo formalismo, sendo necessária uma análise cuidadosa da qualidade da representação sugerida.

O presente trabalho sugere uma linha de representação do dinamismo para a orientação a objetos, e espera, com isso, ampliar o estudo de novas soluções adaptativas. Enquanto os trabalhos existentes focalizam a representação das ações adaptativas, este, além de representá-las, mostrará a evolução do formalismo conforme a execução de ações adaptativas.

5 OBJETOS ADAPTATIVOS

Este capítulo apresenta uma aplicação prática de alguns conceitos da tecnologia adaptativa ao conceito de objetos, formando os objetos adaptativos sob um ponto de vista pragmático. Desta forma, este capítulo define o que vem a ser um objeto adaptativo e outros conceitos necessários para a aplicação dos conceitos da tecnologia adaptativa em objetos. Após estas definições a questão dos tipos é discutida, em questões como a mudança dinâmica de tipos e as conseqüentes alterações necessárias no sistema de tipos. Por fim, a construção e decisões relacionadas ao projeto são apresentadas.

5.1 CONCEITOS

Objeto adaptativo é um elemento que representa um objeto ou conceito do mundo real possuindo propriedades que variam conforme o tempo. Desta forma, um objeto pode alterar seu conjunto de propriedades e com isso especializar seu comportamento conforme o meio em que atua. Isso permite que a analogia com objetos físicos se fortaleça, já que alguns objetos reais podem ser alterados para se adaptarem a diferentes contextos.

Na Orientação a Objetos em voga, que segue as linguagens baseadas em classe, um objeto segue um padrão fixo de acordo com a classe a que pertence. Esta classe determina o comportamento e as características presentes em quaisquer objetos criados no sistema, de acordo com um conjunto de regras bem definido. A idéia do objeto adaptativo é que uma classe permita aos objetos a possibilidade de serem adaptados de acordo com a necessidade do contexto em que é utilizado. Isso pode ser realizado com a possibilidade de inclusão e exclusão de métodos, atributos e relacionamentos entre objetos. Assim, dois objetos da mesma classe podem ter um conjunto de propriedades bem diferente, o que faz com que tenham comportamentos distintos. Outra maneira de se apresentar esta característica da adaptatividade, é a visão de que uma classe poderá potencialmente mapear infinitos tipos através da alteração do conjunto original de suas propriedades. Esses tipos muitas vezes são desconhecidos em tempo de análise e projeto

do sistema, e surgem ou desaparecem com a inclusão ou a exclusão das propriedades do objeto.

5.1.1 NOTAS PARA A EXTENSÃO DO CÁLCULO DE OBJETOS SEM TIPOS

Em relação à teoria dos objetos estudada no capítulo 3, há a necessidade de uma alteração na definição básica das operações dos objetos. Em vez de o objeto possuir apenas dois tipos de operação (invocação e atualização do método), ele deverá possuir quatro: invocação, inclusão, alteração e exclusão do método, sendo a atualização realizável através da exclusão seguida de uma inclusão de um método.

O problema dessa alteração é que o Cálculo-ς sem tipo está essencialmente baseado na seguinte característica das linguagens orientadas a objetos: um objeto pode emular outro objeto que tenha menos métodos, pois o primeiro tem o protocolo completo do último (ABADI; CARDELLI, 1996, p.93). De modo informal, a principal dificuldade é que o cálculo não apóia de modo simples a inclusão e a exclusão.

Uma questão fundamental a ser tratada é a de como permitir que os objetos tenham seus tipos alterados sem que isso viole o sistema de tipos definido para uma solução computacional. A extensão da teoria de objetos para lidar com objetos adaptativos servirá de base para a formalização da verificação de tipos. Em outros palavras, o cálculo de objetos adaptativos permitirá a inferência de tipos e suas variações permitidas pelo sistema de tipos definidos pelo programados.

5.1.2 FORMAS DE ADAPTATIVIDADE EM OBJETOS

A adaptatividade pode ser realizada de duas formas: baseada em objetos, em que a classe será sempre a mesma, mas permitirá que cada objeto se altere de acordo com as necessidades impostas pelo contexto em que é inserido; e baseada em classes, em que a classe pode ser alterada de acordo com novas necessidades do sistema.

É importante ressaltar que as formas de adaptatividade não estão relacionadas aos tipos de linguagens estudados no capítulo 2. As linguagens são divididas segundo a entidade sobre a qual se programa. Já a adaptatividade é dividida de acordo com a entidade que sofrerá ações adaptativas.

Desta forma, esta divisão só faz sentido em linguagens baseadas em classes, pois o conceito de classe inexiste em linguagens baseadas em objetos. Além do mais, as linguagens baseadas em objetos já fornecem um grau de liberdade maior em relação às linguagens baseadas em classes, que é o ganho que é esperado com a inclusão da adaptatividade nas linguagens baseadas em classes.

5.1.2.1 ADAPTATIVIDADE BASEADA EM OBJETOS

A adaptatividade baseada em objetos é denominada desta forma porque as ações adaptativas agem sobre as instâncias das classes (objetos), mantendo as classes dos sistemas inalteradas. Desta forma, uma instância de uma classe vai possuir diferenças em relação à sua classe após a execução de uma ou mais ações adaptativas, fazendo com que o seu comportamento dependa do seu contexto e seu histórico, e não apenas da classe que o define.

Assim como os objetos começaram a ser utilizados em sistemas como analogias do mundo físico, os objetos adaptativos podem ser introduzidos da mesma maneira. Um exemplo da vida real em que acontece essa alteração dos objetos, enquanto a classe permanece intacta, é em relação a corridas de carros de passeio. Carros de passeio são todos fabricados em uma linha de produção e, tirando um ou outro problema que possa ocorrer durante a montagem, saem exatamente iguais. No momento da compra do carro, algumas alterações já podem ocorrer, com a inclusão de acessórios (ar condicionado, direção hidráulica, banco de couro, entre outros). Mais que isso, um carro de passeio pode ser alterado para participar de corridas. As corridas de carro de passeio permitem em suas regras algumas modificações no carro e proíbem outras. Estas modificações, ou adaptações, que ocorrem no carro fazem com que o carro tenha características bem distintas dos outros originados pela fábrica, porém ele continua tendo sido fabricado lá, e a fábrica não precisou ser reformada para que suas características fossem alteradas.

Essa forma de adaptatividade tem características bem interessantes como a manutenção da estrutura da informação durante a vida do sistema e a facilidade da inclusão de pequenas, porém importantes, variações.

A manutenção da estrutura da informação facilita o projeto de sistemas adaptativos, visto que a análise desta estrutura (como ela se desenvolve ao longo do atendimento dos requisitos do sistema) é idêntica à feita em projetos de sistemas orientados a objetos.

Pequenas variações podem ser necessárias em contextos não previstos durante a criação de algumas classes. Uma variação pode ser a adição de um atributo no objeto ou uma pequena extensão em um método. Um caso em que isso poderia ser utilizado ocorre em sistemas de laboratórios clínicos. Cada plano de saúde atendido por um laboratório exige diferentes informações para o faturamento de um exame realizado por um de seus afiliados. Desta forma, cada plano de saúde exibe um contexto diferente, fazendo com que o sistema tenha de ser capaz de gerenciar as informações possíveis em cada caso. Por se tratar de um agente externo, a necessidade de uma nova informação não só é possível, como acontece, e a inclusão dela no sistema pode ser facilitada com uma ação adaptativa que inclua um novo atributo.

O conjunto de problemas que a adaptatividade baseada em objetos pretende resolver pode ser resolvido pelos Modelos Adaptativos de Objetos, tratado no capítulo 3. Porém, espera-se que a adaptatividade baseada em objetos, junto a uma extensão apropriada e mínima dos diagramas UML, seja capaz de superar os problemas existentes nos modelos adaptativos de objetos vistos no capítulo 3, facilitando a interpretação dos modelos em sistemas adaptativos com a introdução da representação da característica adaptativa na UML.

5.1.2.2 ADAPTATITIVADE BASEADA EM CLASSES

A adaptatividade baseada em classes, denominada assim por atuar sobre as classes, é importante principalmente na ocorrência de alteração de requisitos do sistema. Essa importância ocorre em dois âmbitos: estrutural e desempenho da aplicação.

Continuando com a analogia com o mundo real feito no item 5.1.2.1 e com a fábrica de carros, supõe-se que um projetista da fábrica tenha tido excelentes idéias para o novo modelo de um carro e a fábrica precisará passa por um processo de modernização para

poder fabricar este novo modelo. Estas idéias correspondem a novos requisitos e a fábrica (a classe) tem de ser remodelada para poder atendê-los.

No plano estrutural, a alteração de requisitos pode requerer uma alteração da estrutura do sistema. A adaptatividade baseada em classes permite isso de modo diferente da baseada em objetos, a qual faz com que instâncias possuam estruturas diferentes da sua classe geradora, sem modificá-la. A manutenção da classe na adaptatividade baseada em objetos faz com que a estrutura da informação (representada na UML pelo diagrama de classes) fique intacta. Já a adaptatividade baseada em classes altera as classes, implicando na atualização dessa estrutura, a qual passa a ser compatível com novos requisitos.

No âmbito do desempenho, a adaptatividade baseada em classes tende a ser melhor, já que todas as instâncias da classe serão criadas para atender aos novos requisitos do sistema. No caso da adaptatividade baseada em objetos, a criação das instâncias continuaria a ser feita pelo modelo original, o qual poderia não atender aos novos requisitos, forçando a execução de ações adaptativas antes de se começar a utilizar uma nova instância.

Uma possibilidade para a construção da adaptatividade baseada em classes é a utilização do Javassist, descrito no item **Erro! Fonte de referência não encontrada.**. Entretanto, devido ao fato de ela tratar a alteração da estrutura da informação do sistema, a complexidade de seu estudo aumenta muito e, por isso, esse tipo de adaptatividade deverá ser tratado de forma adequada em um trabalho futuro.

Para o presente trabalho, serão detalhados o projeto e a construção de um sistema com adaptatividade baseada em objetos, servindo também como um teste para que haja a continuidade dos estudos da tecnologia adaptativa na teoria de objetos. Além disso, a construção de um sistema com adaptatividade baseada em classes é mais complicada, pois requer a alteração de código compilado, ou a utilização de recursos de geração automática de código e reflexão para poder ser realizada.

5.2 TIPOS EM OBJETOS ADAPTATIVOS

A complexidade do trabalho com tipos dos objetos adaptativos é devido a sua natureza dinâmica, pois existe a necessidade de diferentes tipos de objeto de acordo com as necessidades impostas por cada contexto em que ele é utilizado. Assim, conforme as ações adaptativas são executados, uma instância de objeto adquire um novo tipo e passa a se comportar de maneira diferente do tipo anterior a que pertencia.

Deve ficar claro que, para cada ocorrência de uma ação adaptativa de inclusão ou exclusão de propriedade, o tipo do objeto será alterado. Isto porque o que define o tipo de um objeto é o conjunto de propriedades que o compõe e, ao alterá-lo, o objeto terá seu tipo alterado. Se não existirem chamadas para as ações adaptativas de inclusão ou exclusão em qualquer ponto da classe geradora do objeto, ele sempre terá o mesmo tipo e se comportará da mesma forma que na teoria de objetos não adaptativos, obedecendo assim a um dos princípios da tecnologia adaptativa que prevê a completa separação entre o dispositivo adaptativo e o formalismo subjacente.

Outro ponto importante é a diferenciação entre tipo e a classe a que uma instância pertence. Diferente do que acontece nas linguagens comerciais mais importantes em voga (como o Java, o C++ e o C#), em que os conceitos de classe e tipo se misturam devido à relação entre tipo e classe ser unária, na adaptatividade uma classe pode ser relacionada a diversos tipos, que muitas vezes são desconhecidos em tempo de desenvolvimento do sistema. Como discutido anteriormente, sem chamadas para ações adaptativas, os objetos possuirão o mesmo tipo, e a relação volta a ser de uma classe para um tipo, o que reforça a separação entre o dispositivo adaptativo e o formalismo adaptativo defendido pela tecnologia adaptativa.

Além deste problema de nomenclatura, existe um problema de semântica a ser resolvido. Um objeto pode ser esperado como sendo de um tipo em um contexto e chegar a esse com outro tipo, devido a ações adaptativas que o modificaram em seu passado. O que esta solução propõe é uma limitação imposta às ações adaptativas, utilizando o conceito de interfaces e fazendo com que as ações adaptativas devam atendê-las para evitar que este tipo de problema aconteça. Para resolver este problema de semântica criou-se o

mecanismo adaptativo, responsável por aplicar ações adaptativas e por evitar que o sistema de tipos seja violado.

5.2.1 MECANISMOS ADAPTATIVOS

Um mecanismo adaptativo é o mecanismo através do qual é alterado o conjunto de propriedades de um objeto, e, desta forma, o tipo ao qual o objeto pertence. Ele é definido como um tipo de propriedade capaz de modificar o conjunto de propriedades do objeto a que pertence. Para isso deve-se especificar como ele deve realizar essa alteração de forma que, inclusive, seja possível prever os tipos que um objeto possa receber durante o seu ciclo de vida.

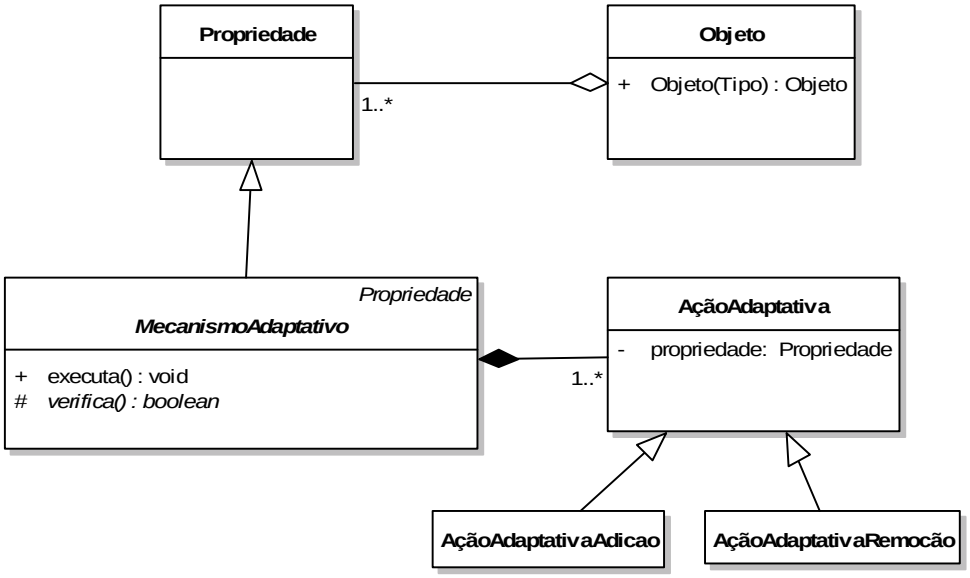

Figura 18 – Mecanismo Adaptativo

Como mostrado na Figura 18 o mecanismo adaptativo é um tipo de propriedade que é composto por ações adaptativas, que são mecanismos capazes de adicionar ou remover propriedades de um objeto. Desta forma, as ações adaptativas ficam centralizadas no mecanismo adaptativo, possibilitando um melhor controle do ciclo de vida do objeto e uma melhor rastreabilidade dos tipos deste objeto. Por ser um tipo de propriedade, um mecanismo adaptativo pode ser incluído ou excluído por uma ação adaptativa (o que é considerada uma adaptatividade de segunda ordem), sendo importante para se ter um comportamento adaptável, em que características dinâmicas são necessárias. Porém, a

questão da compreensão do ciclo de vida de um objeto torna-se mais complexa, dificultando a compreensão deste tipo de sistema.

Outra característica do mecanismo adaptativo é que, além de especificar como a alteração de propriedades deve ser realizada, ele deve conter um verificador para verificar se o tipo gerado após a sua execução é compatível com o contexto em que o objeto está inserido. Para tanto, o tipo criado deve conter um mínimo de propriedades utilizadas no contexto, ou seja, deve implementar a interface necessária pelo contexto em que é inserido.

Uma decisão realizada para simplificar o uso de mecanismos adaptativos é a divisão em três tipos: transformador de tipo, extensor de tipo e redutor de tipo. Existe uma grande semelhança destes tipos com as ações adaptativas existentes, porém enquanto as ações estão diretamente relacionadas a uma propriedade, estes estão relacionados ao tipo do objeto em questão. Um maior detalhamento de cada um dos tipos de mecanismos é fornecida adiante.

O mecanismo adaptativo deve ser associado a um método, para ser executado antes ou depois do método, assim como ocorre com uma função adaptativa em uma transição de um autômato adaptativo.

5.2.1.1 Transformador de Tipo

O Transformador de Tipo é um mecanismo que, como sugere o nome, transforma o objeto do seu tipo atual para outro tipo (ou interface), pré-existente no sistema. Com este tipo de mecanismo é possível, por exemplo, que um objeto se especialize em outro, sem a necessidade de criação de duas instâncias de objeto distintas.

Um exemplo de aplicação que requer a mudança de tipo ocorre na segurança, em que um algoritmo inteligente possa ser capaz de se auto-modificar para criar dificuldades para invasores de sistemas. Com o transformador de tipo, pode haver duas ou mais classes de algoritmos e uma inteligência para modificá-los de forma que o invasor tenha mais dificuldade para identificar o esquema de segurança que protege o sistema.

Para definir esse tipo de mecanismo são necessárias algumas informações: o tipo (ou interface) de destino do objeto e as ações de exclusão, inclusão e alteração necessárias. Além disso, esse mecanismo deve ter, como todo mecanismo adaptativo, uma verificação a ser realizada após a sua execução para aferir se todas as propriedades do tipo (ou interface) de destino são consistentes com as propriedades presentes no objeto.

A Figura 19 representa o transformador de tipo. É importante destacar que ele possui uma propriedade indicando qual o tipo final do objeto após a aplicação do mecanismo. Assim como ocorre em outros mecanismos, há uma especialização do método de verificação.

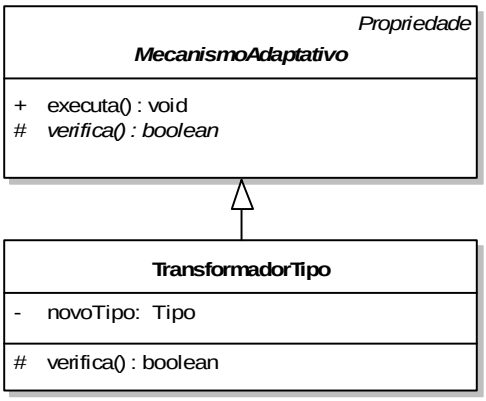

Figura 19 – Transformador de Tipo

O transformador de tipo pode ser utilizado ainda para fazer alterações de método de um objeto, provocando assim mudança de comportamento sem alterar a interface do objeto, o que permitiria utilizar o objeto em um certo contexto com um comportamento especializado. Haveria neste caso uma transformação de tipo sem a alteração da interface, o que pode ser útil em determinados casos.

5.2.1.2 Extensor de Tipo

O Extensor de Tipo é um mecanismo que estende o tipo de um objeto com novas propriedades. Desta forma, ele possibilita a criação de extensões de comportamento, possibilitando a criação de novos comportamentos no objeto. Além disso, ele permite a alteração de propriedades internamente, de forma a manter a interface anterior à aplicação do mecanismo, com a possibilidade de utilizar a extensão.

Este mecanismo poderia ser deixado em uma ou mais classes para permitir a realização de pequenos ajustes durante a execução de um sistema. Desta forma o sistema não precisaria de nova fase de desenvolvimento, diminuindo os seus custos de manutenção.

O extensor de tipo é, então, definido somente por ações de atualização e inclusão de propriedades. O extensor pode ser visto como um mecanismo adaptativo limitado, pois é apenas uma especialização deste último.

A Figura 20 representa o extensor de tipo. Diferente do transformador de tipo, este mecanismo não apresenta o tipo final do objeto, pois este pode ter que ser determinado em tempo de execução do sistema.

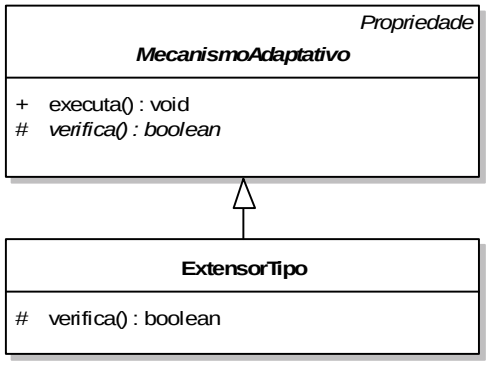

Figura 20 – Extensor de Tipo

Uma diferença do Extensor para o Transformador é que o Tipo em questão nem sempre existe no sistema. Desta forma, a verificação do tipo do objeto é diferenciada, sendo que o ambiente de execução deve pesquisar no sistema se existe uma interface ou tipo correspondente ao conjunto de propriedades final do objeto para evitar duplicidade de tipos, ou criar um novo tipo (ou interface) para os casos em que não exista um tipo ou interface similar.

5.2.1.3 Redutor de Tipo

O redutor de tipo é um mecanismo que altera o tipo de um objeto através da exclusão de propriedades de um objeto. Com isso ele se simplifica ou apresenta um comportamento deteriorado. Isto pode ser útil, por exemplo, para representar este tipo de comportamento típico requerido em realidade virtual ou em jogos eletrônicos.

Este mecanismo, representado na Figura 21, é complementar ao extensor de tipo, sendo formado por ações de alteração e exclusão de propriedades. Embora similar ao extensor, a verificação no redutor é mais complexa, visto que não se pode sempre garantir a interface necessária ao contexto do objeto, como ocorre no extensor. Desta forma, é necessário fazer a verificação após o final da execução para garantir o funcionamento do sistema.

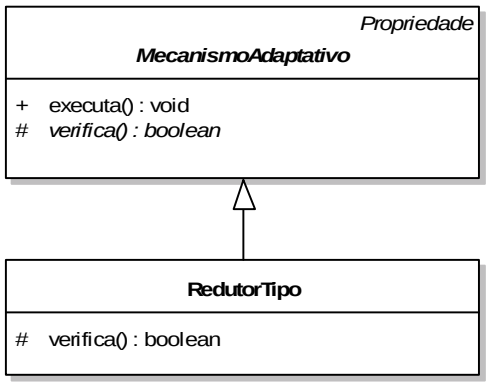

Figura 21 – Redutor de Tipo

Da mesma forma que o Extensor de Tipo, o Tipo final nem sempre existe no sistema. Assim, deve ser executado o mesmo procedimento: procurar, inicialmente, por um tipo ou interface correspondente ao conjunto de propriedades final do objeto; em seguida, ao encontrá-lo, ele deve ser aplicado ao objeto e, em caso contrário, criar um novo tipo e aplicá-lo ao objeto.

5.2.2 INFERÊNCIA DE TIPOS PELA CLASSE

Com a centralização das ações adaptativas nos mecanismos adaptativos e a visão de que um mecanismo altera o tipo do objeto a cada execução, é possível prever o comportamento do ciclo de vida do objeto de acordo com os mecanismos que ele provê. Entretanto, a complexidade da inferência de tipos aumenta com o número de mecanismos adaptativos de um objeto e, ainda mais, se um mecanismo adaptativo cria ou altera mecanismos adaptativos.

Desse modo, é possível criar uma árvore de tipos, em que o tipo inicial é o tipo criado ao se instanciar um objeto da classe. Em seguida, é possível criar ramos para cada

mecanismo adaptativo existente, de forma que cada alteração possa ser visualizada e, conseqüentemente, os tipos possíveis possam ser mais bem analisados para que o sistema funcione da maneira esperada.

A criação desta árvore de tipos pode ser resultado de um algoritmo, em que os mecanismos adaptativos e suas chamadas são recuperados do código fonte da classe. De maneira sintética, uma vez recuperados os mecanismos e suas chamadas é possível inferir quais os tipos possíveis para o objeto a partir do tipo inicial, decorrente de uma simples construção de uma instância da classe. Esse algoritmo é essencial para um compilador de objetos adaptativos, pois através dele é possível verificar se o programa viola, em algum momento, o sistema de tipos.

O algoritmo deve partir de um tipo inicial, que é o tipo gerado pela criação de uma instância da classe. A partir de então, enumeram-se os mecanismos adaptativos da classe. Em um primeiro nível, aplica-se cada mecanismo para o tipo inicial, de modo que sejam descobertos os tipos gerados com a aplicação de um mecanismo adaptativo. Em seguida, aplica-se cada mecanismo em cada tipo gerado no passo anterior, de forma que sejam descobertos os tipos gerados em um segundo nível, e assim sucessivamente. Deve-se ressaltar que cada tipo gerado possui um conjunto próprio de mecanismos adaptativos, de forma que cada aplicação do mecanismo adaptativo em um tipo atua de maneira distinta.

O algoritmo tem a tendência de não terminar, a menos que sejam colocadas algumas restrições. Em primeiro lugar, tipos iguais ou subtipos, não precisam ser analisados mais de uma vez. Para o primeiro caso (tipos iguais) a execução do algoritmo gerará duas árvores de tipos idênticas, sendo desnecessária para a verificação de tipos. O caso de subtipos é mais complexo, mas, em resumo, pode-se dizer que o algoritmo irá criar uma árvore derivada do tipo menos especializado, de forma que todos os seus nós terão um correspondente na árvore mais genérica. Logo, se o sistema de tipos não for violado para um tipo, não o será para o seu subtipo e o sistema estará seguro.

Mesmo com tais restrições, o algoritmo pode não terminar, com a criação de famílias de tipos não relacionadas. Conseqüentemente, sua utilização torna-se inviável. Uma

possibilidade para resolver este problema é definir um número máximo de níveis que um objeto evolui com respeito à verificação de tipo (idealmente um sistema não deveria ter este tipo de limitação). Com isso, se a utilização de um mecanismo adaptativo sempre gerar novos tipos em um certo nó da árvore de tipos, restringir a sua execução a um número limitado de vezes para possibilitar o seu término. Como conseqüência, não se pode garantir a segurança do sistema de tipos criado, uma vez que a verificação de tipos não será completa, apenas aproximada. O tratamento formal desta questão deverá fazer parte de um trabalho futuro.

No item 5.5, retorna-se ao resultado das execuções dos mecanismos adaptativos no ciclo de vida de objetos.

5.3 CONSTRUÇÃO

Existem algumas soluções possíveis para produzir objetos adaptativos. A primeira delas seria utilizar o Modelo Adaptativo de Objetos (capítulo 3) para criar uma camada de abstração para os objetos adaptativos. Assim, utilizando o AOM é possível criar um objeto adaptativo através desta estrutura e fornecer um arcabouço para que o desenvolvedor utilize objetos adaptativos de forma transparente, em que teria, basicamente, interfaces para incluir e excluir elementos de um objeto.

Outra possibilidade de construção de objetos adaptativos é alterar uma máquina virtual orientada a objetos, como a *Java Virtual Machine* ou a máquina virtual do *Microsoft* .NET, para representar objetos adaptativos, o que possibilitaria não só a construção, como um melhor aproveitamento dos recursos da máquina virtual para obter um melhor desempenho. Neste caso, uma possibilidade de se realizar a pesquisa de método, mostrada no capítulo 3, poderia ser feita segundo a Figura 22. Nela, uma tabela denominada métodos adaptados é adicionada, podendo ser referenciada pela tabela de métodos da classe. Deste modo, o acesso aos métodos adaptativos é uniforme. Da mesma maneira, deve haver métodos de inclusão, exclusão e invocação dos métodos com acesso à tabela de métodos adaptados, não incluídos na figura para simplificá-la.

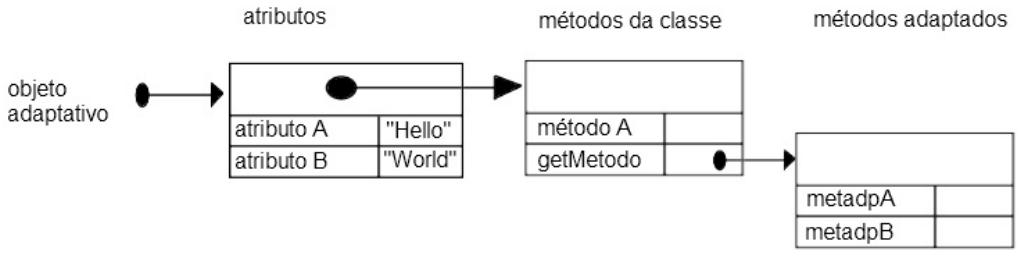

Figura 22 – Métodos Adaptativos

Apesar da vantagem de melhor utilização dos recursos, a utilização do Modelo Adaptativo de Objetos foi escolhida principalmente pelo fato de requerer um menor tempo de desenvolvimento e servir para demonstrar o ganho de expressividade que a adaptatividade traz aos objetos, um dos objetivos do presente trabalho.

5.3.1 REFLEXÕES E DECISÕES DE PROJETO

Para a construção realizada neste trabalho, algumas decisões tiveram que ser feitas, para reduzir o tempo de desenvolvimento e ter maior congruência com a Tecnologia Adaptativa e com a Teoria de Objetos. Deve-se lembrar que, mesmo tendo uma visão mais pragmática, a solução do presente trabalho baseia-se nos conceitos da tecnologia adaptativa para procurar criar uma solução teoricamente bem fundamentada, de modo a tornar possível a continuidade da pesquisa em diversos aspectos, inclusive a criação de um formalismo para os objetos adaptativos. O texto que segue apresenta algumas reflexões sobre as alternativas de decisões de projetos e as principais decisões.

Em primeiro lugar, escolheu-se o uso do AOM para a criação de um arcabouço que tenha a capacidade de gerar objetos adaptativos. O arcabouço pode ser feito sem grandes dificuldades, atendendo aos requisitos que a tecnologia adaptativa exige. No arcabouço, o objeto adaptativo é criado em um meta-modelo, o qual permite a criação de todos os conceitos estudados para o objeto adaptativo.

Com o arcabouço, preservam-se as características da orientação a objetos e um desenvolvedor só precisa entender os objetos adaptativos caso necessite e queira utilizá-los. Esta separação é uma das características necessárias da tecnologia adaptativa, a qual

prevê que modelos projetados no formalismo original, no caso modelos orientados a objetos, continuem sendo válidos no formalismo estendido com o dispositivo adaptativo. Deve-se reforçar que a tecnologia adaptativa não está sendo aplicada a um formalismo, mas algumas idéias desta tecnologia estão sendo aplicadas a um caso particular da teoria dos objetos, aos contextos particulares de uma linguagem orientada a objetos.

Uma das necessidades que surgem com a adaptatividade é a separação dos conceitos de tipo e classe nos objetos adaptativos. Por isso, um objeto adaptativo deve ter, além de um método *ObterTipo(),* já presente nas linguagens de programação C# e Java (método *GetType()*), um método *InstânciaDe()* ou *ObterClasse()*. Para isso, os objetos adaptativos utilizarão o padrão Objeto Tipo (JOHNSON; WOOLF, 1998), estudados no capítulo 3. O método *ObterTipo()* será sobrescrito para retornar o tipo determinado dinamicamente pelo meta-modelo. Já o método *ObterClasse()* retornará a classe que gerou a instância em questão.

Diferente da teoria dos objetos formulada em (ABADI; CARDELLI, 1996), em que um objeto é um conjunto de métodos e os campos são tipos de métodos específicos, como visto no item **Erro! Fonte de referência não encontrada.**, os atributos e os métodos serão tratados separadamente, como dois tipos diferentes de propriedade, para simplificar desenvolvimento e para aproximar da visão dada por linguagens orientadas a objetos usuais, como o Java, o C++ e o C#. Com essa separação, espera-se facilitar o trabalho do desenvolvedor de objetos adaptativos, o qual já está acostumado com esta divisão.

As propriedades serão feitas através do padrão Propriedade (FOOTE; YODER, 1998) estudado no capítulo 3. Métodos e campos estendem essa classe, caracterizando os tipos diferentes de propriedades, sendo que os métodos poderão ser realizados de diferentes formas, através da herança de uma estrutura básica contendo um método de disparo, um nome, um conjunto ordenado de parâmetros e um tipo de retorno do método. Em princípio, uma das formas será através do padrão Objeto Regra (ARSANJANI, 2000 – capítulo 3) e outra através do espaço de nome *System.Reflection.Emit*, o qual permite a emissão de métodos dinamicamente.

Diferente do que prevê a tecnologia adaptativa, uma ação adaptativa irá ser aplicada a uma entidade apenas, e não a um conjunto de entidades paralelamente. Assim, as propriedades e métodos que precisem ser modificados, serão recuperados através de sua assinatura, única no sistema, e farão com que as ações adaptativas atuem sobre uma única propriedade por vez, simplificando o entendimento de sua aplicação.

Uma outra restrição necessária, é que as ações adaptativas só poderão ser chamadas através dos mecanismos adaptativos, um tipo de propriedade que possui as regras para aplicação das ações adaptativas. Estes mecanismos se utilizarão das ações adaptativas para alterar o tipo do objeto e, depois, irão verificar se o sistema de tipos continua consistente.

O padrão de consulta dos objetos adaptativos deverá estar de acordo com o nome das propriedades e dos métodos, e um conjunto de tipos de parâmetros, sempre vazio para o caso das propriedades. Isto quer dizer que o padrão de consulta seguirá a assinatura de métodos, propriedades e mecanismos adaptativos, de forma que o resultado da consulta será um subconjunto unitário do conjunto de propriedades do objeto adaptativo, sendo vazio quando o padrão não for encontrado.

As ações adaptativas são métodos embutidos nos objetos adaptativos, podendo ser utilizados pelo desenvolvedor através da especificação de um ou mais mecanismos adaptativos. Serão ações específicas, ou seja, agem sobre uma propriedade. Assim serão criados três métodos para a realização das ações adaptativas:

- *ObterPropriedade(padrão de consulta)*, que devolve um subconjunto de propriedades que aderem ao padrão de consulta passado como argumento.
- *RemoverPropriedade(propriedade)*, que remove uma propriedade do objeto.
- *AdicionarPropriedade(propriedade)*, que adiciona uma propriedade no objeto.

Outra necessidade da tecnologia adaptativa é a aplicação da ação adaptativa antes e/ou depois da aplicação da regra em questão. No caso dos objetos, a aplicação da regra corresponde ao uso de uma propriedade, seja a execução de um método, a atribuição ou recuperação de um valor de uma propriedade, ou a execução de um mecanismo

adaptativo. Para incluir o mecanismo adaptativo antes ou depois de uma propriedade, serão utilizados dois novos atributos (um para executar o mecanismo antes e outro para executar depois do uso da propriedade) para a propriedade, especificando qual mecanismo deve ser executado. Desta forma, na utilização de uma propriedade, deve-se verificar se tais atributos existem, para a aplicação do mecanismo adaptativo apropriado.

Outra importante decisão foi o relacionamento entre o objeto adaptativo e o objeto convencional. Este relacionamento poderia ser uma especialização, em que o objeto adaptativo especializa alguns métodos, como o método ObterTipo, e incrementa outros, as ações adaptativas. Outra possibilidade seria representá-los como uma composição, em que o objeto adaptativo contém um objeto convencional, de forma que o objeto adaptativo só exporia as propriedades necessárias.

A especialização restringe mais o objeto adaptativo, visto que a dependência do objeto não adaptativo é grande. Entretanto ela se adequa à tecnologia adaptativa, na medida em que a tecnologia preconiza que a camada subjacente deve ser mantida para ser utilizada normalmente. A composição provê uma maior flexibilidade, simplificando a construção do mecanismo adaptativo. Pode-se dizer que ela também atende à tecnologia adaptativa, na medida em que o objeto adaptativo representa um conjunto maior que o objeto não adaptativo, incluindo ele próprio. Entretanto, a camada subjacente (o objeto não adaptativo) passa, nesta visão, a ser dependente da camada adaptativa (o objeto adaptativo), pois ela tem controle total sobre a outra camada, impossibilitando o uso da camada subjacente de maneira transparente. Por esta razão, a especialização foi escolhida para implementar o objeto adaptativo.

Além disso, um ponto ainda não trabalhado foi o não determinismo que a tecnologia adaptativa prevê (item 4.2). Este item deverá ser tratado em um trabalho futuro, em que o dispositivo adaptativo seja incluído no formalismo do objeto. Como este trabalho é pragmático, este item não será levado em consideração, por questões de simplificação do escopo da pesquisa.

5.3.2 CONSTRUÇÃO DA ADAPTATIVIDADE BASEADA EM CLASSES

Apesar de não ser o foco deste trabalho, vale a pena destacar algumas possibilidades para a construção desta outra forma de adaptatividade, tanto como referência para trabalhos futuros, como para esclarecer um pouco mais as diferenças entre as duas formas de adaptatividade tratadas no texto.

Para construir a adaptatividade baseada em classes, pode-se pesquisar a possibilidade de uso da Geração Automática de Código, munida por ferramentas de reflexão, assim como a possibilidade de uso da reflexão estrutural.

Reflexão, como foi visto, é a possibilidade, em tempo de execução, de se descobrir características intrínsecas ao objeto que está sendo utilizado. É possível instanciar objetos de uma classe desconhecida em tempo de desenvolvimento, bem como aplicar valores a seus atributos ou executar métodos. Com isso, seria possível, sem muitas dificuldades, gerar e compilar automaticamente o código de uma classe em tempo de execução para utilização desta nova classe. Esta classe poderia substituir uma outra vigente até então, ou mesmo ser uma nova classe no sistema, passando a atender novos requisitos do sistema.

A reflexão estrutural permite a alteração das classes e objetos em tempo de execução. Infelizmente este tipo de reflexão não é implementado nas bibliotecas das máquinas virtuais vigentes, como a JVM e o .NET. Apesar disto, pode-se implementar esta forma de reflexão como realizado em (ORTIN; et. al; 2005), em que é possível ter um ganho de desempenho em relação ao código gerado automaticamente, já que não existem os passos de geração, compilação e acesso do código gerado através da reflexão comportamental.

5.3.3 ARCABOUÇO DE OBJETOS ADAPTATIVOS

A partir das decisões tomadas, pôde-se esboçar o projeto do arcabouço de objetos adaptativos como um meta-modelo de objetos. Deve-se inicialmente representar os objetos comuns, ou não adaptativos, em um meta-modelo, para depois estendê-lo com os mecanismos adaptativos.

A Figura 23 representa uma parte de um meta-modelo de objetos, representando apenas algumas características essenciais, como o Tipo do Objeto, com um relacionamento em que um objeto tem um tipo, e um tipo pode ser aplicado a um ou mais objetos. Além do tipo, o meta-modelo representa as propriedades e suas assinaturas, que as identificam unicamente e através da qual será realizada a ação adaptativa de consulta. Ainda são representados os tipos de propriedades: Campo, que possui um valor implícito e métodos para recuperá-lo e alterá-lo, e Método, que possui um corpo que representa a sua implementação e um método para executá-lo.

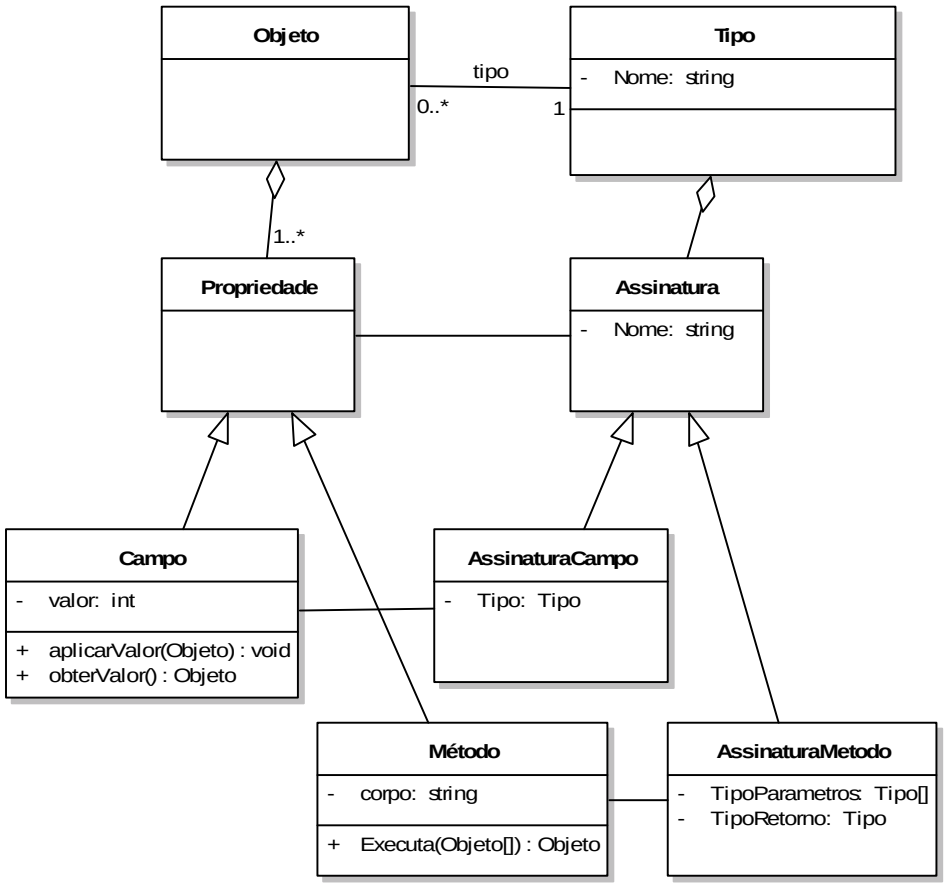

Figura 23 – Meta-modelo de objeto

A inclusão do dispositivo adaptativo no meta-modelo introduz um novo tipo de propriedade: o mecanismo adaptativo o qual estende o objeto não adaptativo, tornando-o adaptativo através dos métodos de inclusão, exclusão e alteração de propriedades.

A Figura 24 mostra o meta-modelo evidenciando as mudanças ocorridas no meta-modelo anterior com a introdução da camada adaptativa. Pode-se verificar na figura que um objeto adaptativo é uma especialização de um objeto não adaptativo, bem como o mecanismo adaptativo é um tipo de propriedade com três subtipos: o transformador de tipo, o extensor de tipo e o redutor de tipo.

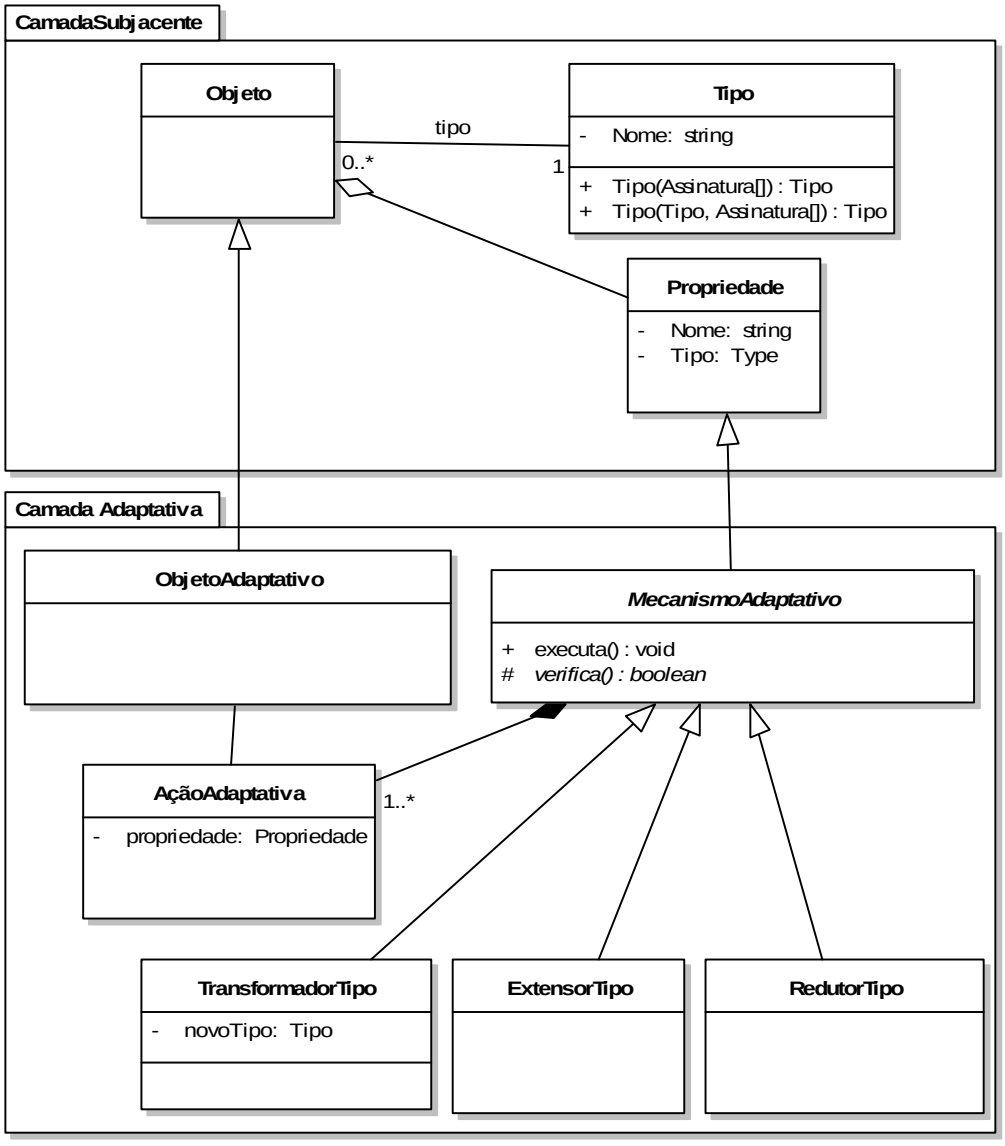

Figura 24 – Meta-modelo de objeto adaptativo

A Figura 24 não apresenta, por simplificação, o atributo do método que indica qual mecanismo deve ser aplicado antes e/ou depois da sua execução. Além disso, a figura mostra as ações adaptativas, pertencentes a um mecanismo adaptativo e atuando sobre o objeto adaptativo.

Com este modelo e as decisões descritas no item 5.3.1 é então possível construir o arcabouço e, a partir dele, construir um simples protótipo funcional demonstrando o seu funcionamento.

5.4 PROTÓTIPO

Com a solução do Modelo Adaptativo de Objetos descrita no item 5.3, projetou-se um arcabouço adaptativo e desenvolveu-se um protótipo para demonstrar o funcionamento da funcionalidade adaptativa nos objetos adaptativos. O protótipo retrata o exemplo mostrado no capítulo de tecnologia adaptativa (item 4.3), mas realizado através dos objetos adaptativos. O exemplo foi escolhido devido à sua simplicidade e porque permite mostrar todas as ações adaptativas em ação.

Foram realizadas duas formas de se executar o protótipo: reconhecimento de uma cadeia formada por um caractere *a*, um caractere *b* e um caractere *c*; e reconhecimento de uma cadeia de um número qualquer de letras, formada pelas letras iniciais do alfabeto, sem repetição de letras (ou seja, uma cadeia de uma letra deve ser o *a*, uma cadeia de duas letras deve ser formada por um *a* e um *b*, e assim por diante).

O protótipo permite ainda que se execute o reconhecimento passo-a-passo, ou seja, uma letra por vez, ou o reconhecimento completo. O primeiro foi feito para acompanhar as alterações no objeto reconhecedor após o reconhecimento de cada letra. O reconhecimento completo é útil para a verificação do funcionamento do algoritmo, pois permite a entrada de diversas cadeias e seu reconhecimento num tempo menor.

O protótipo se inicia com a entrada da cadeia pelo usuário e seleção do tipo de reconhecimento (um *a*, um *b*, e um *c* ou n letras).

A classe base para qualquer reconhecedor é a classe ReconhecedoraDeCadeia. Ao se selecionar um reconhecimento de um *a*, um *b* e um *c*, a classe gera uma instância

reconhecedora de vazio, e o sistema o altera para ser do tipo reconhecedor de um *a*, um *b* e um *c*. Para isso adicionam-se três métodos, cada um para consumir uma das letras, e três propriedades responsáveis por indicar a necessidade de se consumir cada letra. Ao executar um consumo de uma letra, o método de consumo é removido e a propriedade de necessidade de reconhecimento da letra é marcada. Assim ocorre até o final da cadeia, quando todos os métodos de consumo são excluídos e as propriedades marcadas, indicando reconhecimento da cadeia. Quando a cadeia for inválida, ou alguma das propriedades irá permanecer desmarcada (faltam letras na cadeia), ou não existe o método de consumo para a letra que está sendo reconhecida. No primeiro caso, o reconhecimento acaba sem haver a confirmação da cadeia pelo reconhecedor. No segundo, uma exceção é lançada indicando que uma letra não pode ser reconhecida. O acesso ao objeto é feito através de dois métodos não adaptáveis: um indicando se o reconhecimento foi feito (cadeia consumida e todas as propriedades marcadas) e outro fazendo o reconhecimento da próxima letra na cadeia, passando por todos os métodos existentes. No caso de não haver o consumo, gera-se o erro de letra não reconhecida.

O reconhecimento de um número qualquer de letras é feito de maneira análoga ao reconhecimento de uma cadeia formada por um caractere *a*, um caractere *b* e um caractere *c*. A diferença é o número de métodos e propriedades criadas, sendo variável neste caso. Com tantas semelhanças, o reuso de código foi utilizado através do mecanismo de herança da solução mais simples.

A seqüência de passos da execução do reconhecimento de uma cadeia "*bac*" pode ser resumida como segue. O reconhecedor é iniciado e seu tipo é alterado para um reconhecedor de um *a*, um *b* e um *c*. Ao reconhecer o primeiro caractere, o consumo de *b* é executado e excluído, e o reconhecedor passa a ser do tipo reconhecedor de um *a* e um *c*. Ao reconhecer o caractere *a*, o consumo de *a* é executado e excluído, e o reconhecedor passa a ser do tipo reconhecedor de um *c*. Por fim, o reconhecedor consome o caractere *c* e se transforma num reconhecedor de cadeia vazia, aceitando a cadeia de entrada "*bac*".

No caso do reconhecedor de um *a*, um *b* e um *c*, o número de tipos de reconhecedores possíveis é limitado e conhecido, podendo ser representado pela Figura 25, na qual se representa a árvore de tipos desse reconhecedor. O exemplo passo a passo é demarcado em negrito, demonstrando que apenas alguns tipos da árvore são utilizados no ciclo de vida deste objeto.

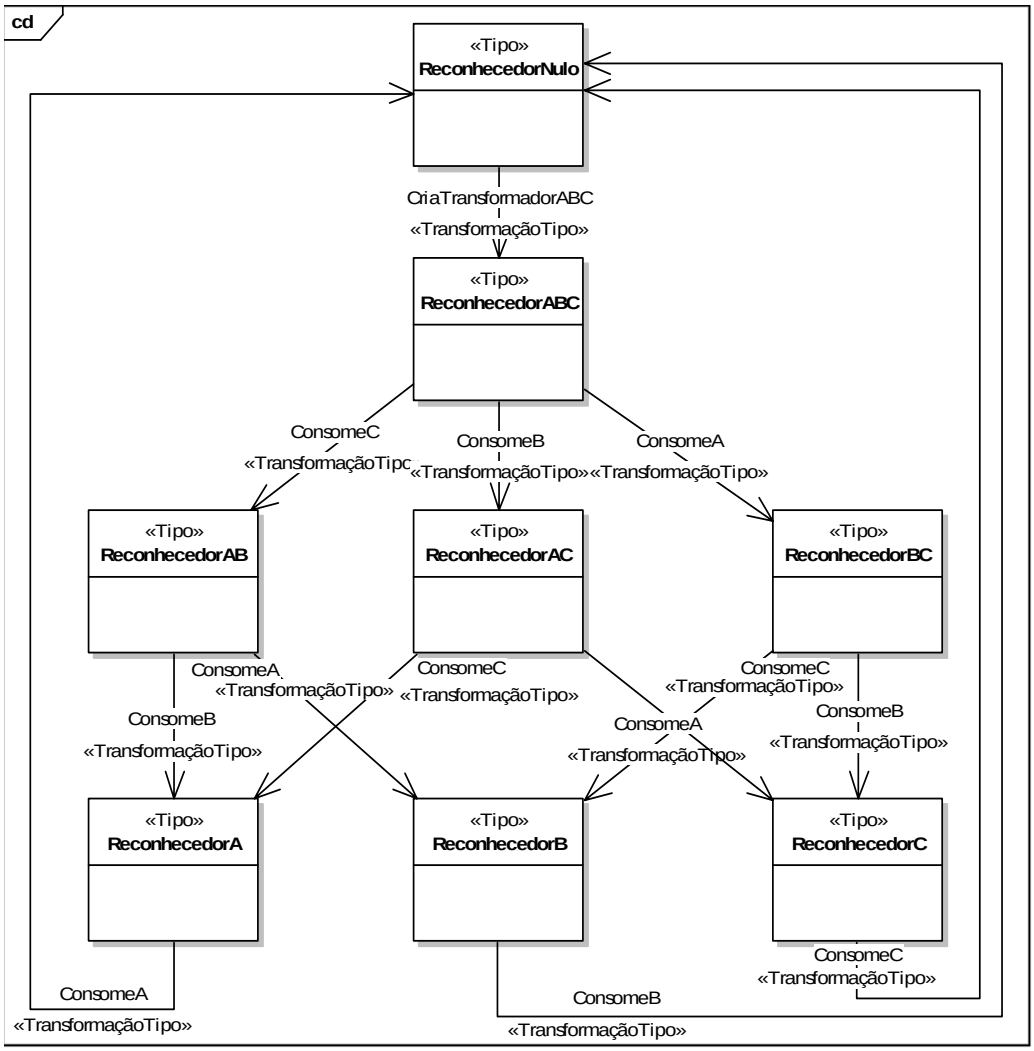

Figura 25 – Árvore de tipos do reconhecedor de um *a*, um *b* e um *c* em qualquer ordem(em destaque o reconhecimento da cadeia "*bac*")

5.5 MODELAGEM DE OBJETOS ADAPTATIVOS

Apesar dos problemas que o objeto adaptativo se propõe a resolver poderem ser resolvidos com outras soluções, como o Modelo Adaptativo de Objetos do capítulo 3 e o

uso de reflexão, todas as soluções esbarram no problema da modelagem. A intenção deste capítulo é tanto mostrar os requisitos para a representação do comportamento adaptativo em diagramas de uma extensão da UML, como preservar os aspectos positivos fornecidos pela modelagem orientada a objetos da UML. Em outras palavras, o critério de decisão primordial é, sempre que possível, não alterar a UML convencional.

Seguindo a linha de pesquisa da tecnologia adaptativa, em que novos diagramas foram propostos, este estudo irá procurar trazer para os objetos adaptativos uma forma de representar seu dinamismo, estendendo os diagramas vigentes da UML. Pretende-se que os elementos dos modelos constantes nos diagramas originais, relativos à solução não adaptativa, continuem a ser válidos nos novos diagramas, como preconiza a tecnologia adaptativa.

Para tanto, o presente trabalho pretende dar um primeiro passo, analisando apenas alguns diagramas essenciais à modelagem orientada a objetos, deixando os restantes para um trabalho futuro. Desta forma, serão analisadas apenas as extensões para os diagramas de classes e de seqüência, além da inclusão de um novo diagrama, responsável pela visualização dos tipos possíveis que um objeto pode assumir durante seu ciclo de vida.

A escolha dos dois diagramas deve-se à importância deles (ou os diagramas mais utilizados pelos desenvolvedores) e à característica de cada um. O diagrama de classes tem aspecto estrutural, sendo o mais utilizado diagrama da modelagem orientada a objetos, pois serve como base para toda a modelagem, inclusive na criação de outros diagramas. O diagrama de seqüência tem aspecto dinâmico, apresentando o comportamento inter-objetos do sistema. A utilização de dois diagramas com aspectos diferentes deve nortear os estudos para a extensão dos outros diagramas, de modo que todos estejam aptos a representar o comportamento adaptativo de um sistema de maneira adequada.

A extensão será realizada sobre a UML 2.0 (OMG, 2005), com a criação de perfis UML para a representação das características adaptativas incluídas nos objetos adaptativos.

5.5.1 DIAGRAMA DE CLASSES

O modelo de classes tradicional apresenta a estrutura dos objetos do sistema. Como preconizado pela Tecnologia Adaptativa, o formalismo que recebe a camada adaptativa deve continuar funcionando após a sua inserção. Em outras palavras, as novidades trazidas pela tecnologia adaptativa devem ser alternativas, ou seja, o usuário deve poder escolher entre utilizá-la ou não. Se optar por não utilizá-las, a sua existência não deve incorrer em mudanças na forma como trabalha, mantendo sua representação nos diagramas intacta. Assim, a substituição de um enfoque pelo outro é facilitada, uma vez que o uso das características do objeto adaptativo é opcional.

O modelo de classes adaptativo deve apresentar uma nova propriedade a ser aplicada em suas propriedades existentes: a propriedade da adaptatividade. Esta propriedade deve indicar que uma propriedade original pode ser modificada em tempo de execução, podendo até mesmo ser excluída. Com isso, propriedades não adaptativas definiriam a estrutura tradicional da classe aos objetos. Com a extensão das propriedades adaptativas, os objetos passariam a ter características próprias, mesmo pertencentes às classes, já que poderiam, a qualquer momento, incluir ou excluir uma propriedade, método ou mecanismo adaptativo.

O modelo de classes da UML é um dos que menos deve sofrer alterações para ser transformado em um modelo adaptativo. O que deve ser alterado neste modelo é a apresentação da visão do que pode ou não ser alterado por ações adaptativas, sem que o objeto deixe de pertencer à classe original. Além disso, o modelo deverá evidenciar os mecanismos adaptativos pertencentes à classe, mostrando como os objetos podem ter seu tipo alterado em tempo de execução.

Como visto no item **Erro! Fonte de referência não encontrada.**, isso só acontece porque a forma de adaptatividade em objetos estudada não é a baseada em classes, que implicaria em mudanças estruturais e, deste modo, iria implicar em diferentes modelos de classe durante a execução de um sistema que aplica este tipo de adaptatividade.

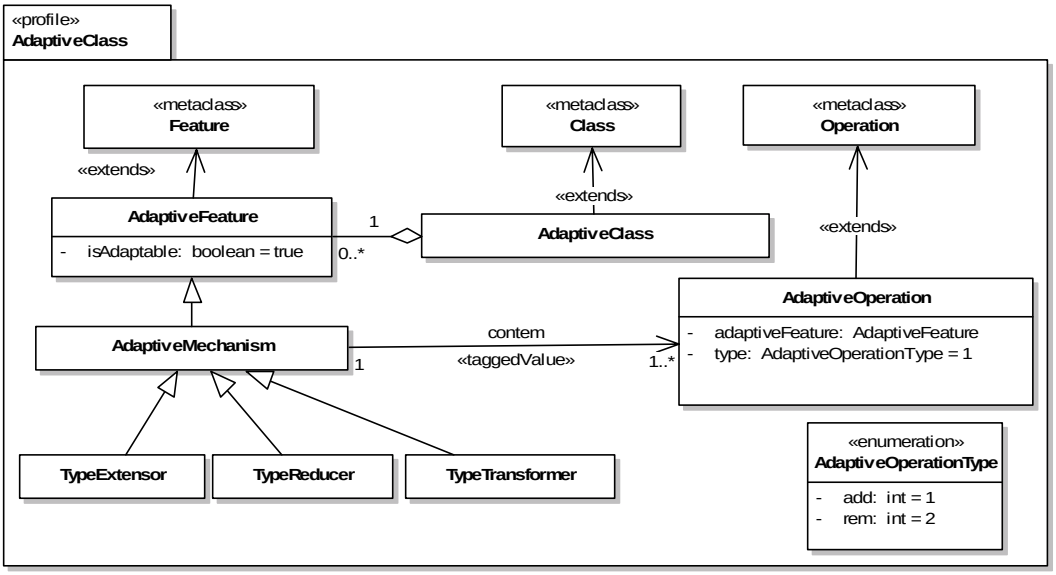

Figura 26 – Perfil UML AdaptiveClass

Assim, para representar as características da adaptatividade, criou-se um perfil UML, conforme mostrado na Figura 26 [5]. A figura representa uma classe adaptativa ("*AdaptiveClass*"), que indica se uma classe cria ou não objetos adaptativos. Uma classe adaptativa possui propriedades adaptativas ("*AdaptiveFeature*"), sendo propriedades que podem ser adicionadas e removidas do objeto em tempo de execução. Além disso, apresentam-se os mecanismos adaptativos ("*AdaptiveMechanism*"), responsáveis por aplicar um conjunto de operações ou ações adaptativas ("*AdaptiveOperation*"), de forma a alterar o tipo de um objeto, podendo ser do tipo Transformador ("*TypeTransformer*"), Redutor ("*TypeReducer*") ou Extensor ("*TypeExtensor*"). Uma ação adaptativa pode ser de adição ("*AddFeature*") ou de remoção ("*RemFeature*"), e é aplicada a uma instância de uma classe com a adição de uma nova propriedade ou remoção de uma propriedade existente. As classes *AdaptiveClass, AdaptiveFeature* e *AdaptativeOperation* são definidas como extensões das meta-classes *Class*, *Feature* e *Operation*, respectivamente, do meta-modelo da UML (OMG,2005).

[5] A ferramenta utilizada para criar os perfis não oferece suporte à notação introduzida na última revisão da UML 2.0 para a extensão de meta-classes (o símbolo de herança em que o triângulo tem preenchimento em preto). No entanto, o relacionamento <<extendes>> disponível, tem a mesma semântica de extensão de meta-classe definida na UML 2.0.

O modelo de classes, diferente da maioria dos modelos estudados pela Tecnologia Adaptativa, não apresenta comportamento dinâmico, ou seja, em qualquer momento do sistema, independente das ações adaptativas que venham a ocorrer, o modelo é sempre o mesmo e deve representar o sistema da mesma maneira. A justificativa para isso é que as classes do sistema não mudam, e sim suas instâncias, fazendo com que esse diagrama seja estático, não sendo alterado pelos mecanismos adaptativos.

Apesar de o diagrama de classes não representar o comportamento dinâmico, ele representa a configuração inicial dos objetos, representando as instâncias no momento em que são criadas. Isso delimita os tipos que as instâncias podem receber com a execução dos mecanismos adaptativos pertencentes a ela.

Com a definição deste perfil UML, é possível criar um diagrama para o exemplo do reconhecedor de um *a*, um *b* e um *c*. Como representado na Figura 27, uma classe leitora de cadeia, utiliza o reconhecedor adaptativo para verificar se a cadeia tem o formato esperado. É importante ressaltar que o diagrama não representa explicitamente todos os tipos a que o reconhecedor pode pertencer, mas é possível inferi-los pelos mecanismos adaptativos existentes (do tipo *TypeReducer*).

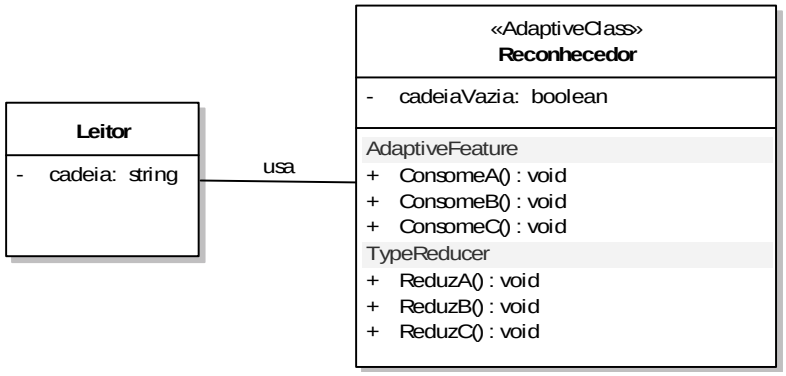

Figura 27 – Um exemplo de diagrama de classes adaptativo para o reconhecedor de um a, um b e um c.

A visão de quais tipos uma instância pode receber e do ciclo de vida do objeto, com a identificação dos tipos por qual ele passa, serão responsabilidade, respectivamente, dos diagramas de tipos adaptativos e de seqüência, apresentados adiante.

5.5.2 DIAGRAMA DE SEQÜÊNCIA

Outra classe de diagramas importante na modelagem orientada a objetos é a classe dos diagramas dinâmicos, a qual representa o comportamento de instâncias do objeto durante seu ciclo de vida, Esta classe é formada por três diagramas, cada um representando o comportamento do objeto sob um determinado ponto de vista. O diagrama de seqüência apresenta o comportamento do ponto de vista temporal, ou seqüencial. Além dele, existem o diagrama de estados e o diagrama de comunicação. O primeiro, apresenta o ponto de vista de máquinas de estados finitas e o segundo focaliza as interações entre as linhas de vida dos objetos, relevando a arquitetura da estrutura interna dos objetos e de suas comunicações através de mensagens.

Para representar o comportamento adaptativo, escolheu-se um dos diagramas, de forma que o tratamento a ser adotado para os demais diagramas deva derivar deste. O diagrama de estados é similar aos diagramas já estudados na tecnologia adaptativa, como os autômatos (Neto, 1994) e os *statecharts* adaptativos (Junior, 1995) e, por isso, não será visto neste trabalho. O diagrama de comunicação e o diagrama de seqüência são equivalentes semanticamente (OMG, 2005), e aparentemente a solução para que eles representem as características adaptativas deve ser similar.

A representação de sistemas com tipos alteráveis pode ser analisada sob o ponto de vista da classificação dinâmica (Odell et al, 2003). De modo informal, a classificação dinâmica corresponde à aplicação de regras ou classificações dinamicamente a entidades, fazendo com que estas possuam diferentes comportamentos e possam ser utilizadas em diferentes contextos.

Desta maneira, em um exemplo prático do dia-a-dia (Odell et al, 2003), pessoas podem ser empregadas ou não, e se empregadas podem ocupar diferentes cargos. Neste caso, a pessoa corresponde a uma entidade e o status de empregabilidade e cargo, a uma classificação. Mais que isso, o status pode mudar, ao se despedir, promover ou rebaixar um funcionário. Isso altera a classificação da entidade, ou, em outras palavras, o tipo do objeto, considerando-se que a entidade é um objeto e a classificação é um tipo. Com isso o tipo do objeto adaptativo passa a ser o tipo formado pelo conjunto das suas partes.

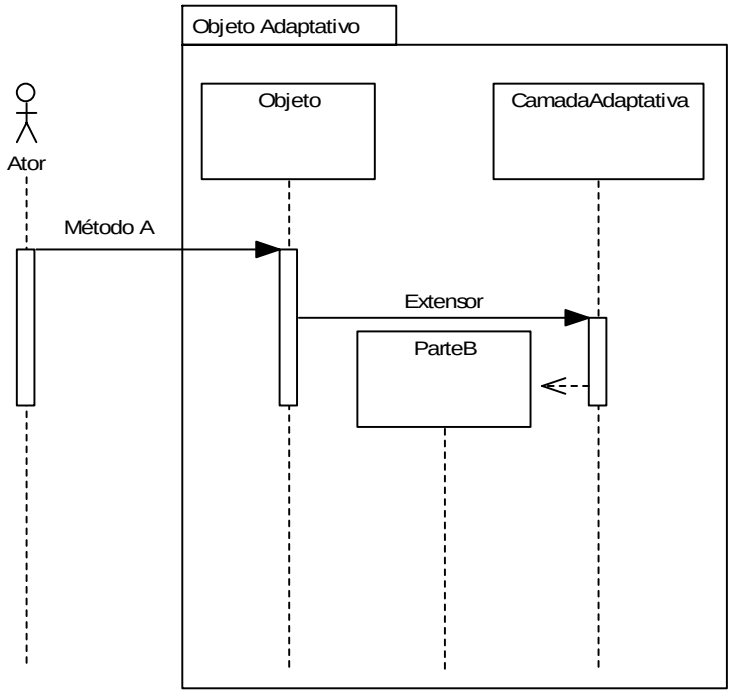

Figura 28 – Representação do mecanismo de extensão de tipo - adaptada de (Odell, 2003)

Assim, a mesma notação utilizada para a classificação dinâmica, pode ser usada para representar os tipos dinâmicos decorrentes da tecnologia adaptativa. Desta forma, a representação de um mecanismo de extensão de tipo pode ser comparada à classificação (por exemplo, a promoção do funcionário), e representado como mostrado na Figura 28. Diferente da representação de classificação, a extensão de tipo se inicia no dispositivo adaptativo e não em um elemento qualquer do diagrama. O objeto adaptativo é considerado como um conjunto formado pelo objeto original, o dispositivo adaptativo e as variações que nele ocorram.

De modo análogo à comparação entre o mecanismo de extensão e a classificação, pode-se aplicar a representação de uma desclassificação para o mecanismo de redução de tipo, conforme mostrado na Figura 29. Da mesma forma que o extensor de tipo, a figura mostra que o dispositivo adaptativo causa a eliminação de uma parte do objeto, de modo que o objeto adaptativo passa a pertencer ao tipo das partes restantes.

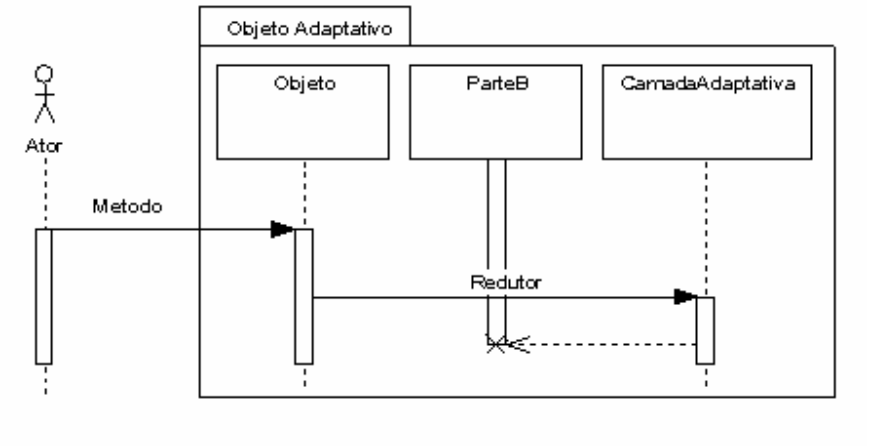

Figura 29 – Representação do mecanismo de redução de tipo – adaptada de (Odell, 2003)

Por fim, faz-se a analogia entre o mecanismo de transformação de tipo com a reclassificação para representá-lo conforme a Figura 30. Neste caso, o Transformador aplicará duas ações: eliminará o tipo original e criará o tipo de destino. Desta maneira, para o usuário, o objeto adaptativo está tendo o seu tipo trocado.

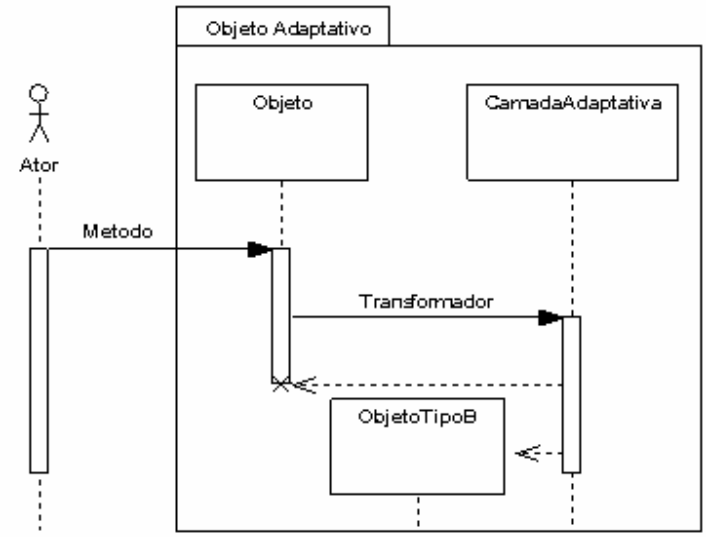

Figura 30 – Representação do mecanismo de transformação de tipo – adaptada de (Odell, 2003)

Com esta representação é possível ressaltar a variação de tipos proveniente da aplicação dos mecanismos adaptativos. Entretanto ela não é suficiente, por si só, para indicar a existência de uma única entidade, pois esta representação sugere que, ao se trocar um tipo, é criada uma nova instância, diferente da instância original, quando na realidade a

aplicação da tecnologia adaptativa faz com que a mesma instância tenha seu tipo alterado.

Por isso, utiliza-se o conceito de pacote da UML (OMG, 2005) para agrupar as diferentes partes que formam o objeto adaptativo. Desta forma, um pacote representa uma instância de um objeto adaptativo, formado por uma parte representando o dispositivo adaptativo, e outra representando a camada subjacente.

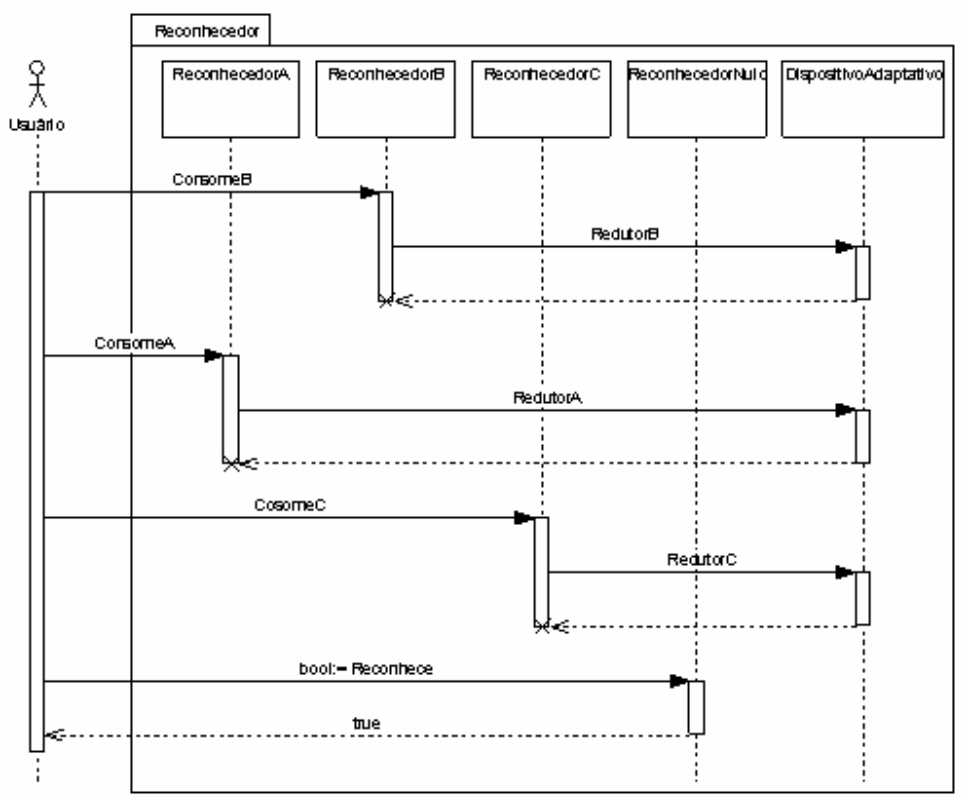

Figura 31 – Diagrama de seqüência do reconhecedor de um *a*, um *b* e um *c*.

Assim, um exemplo de um diagrama de seqüências para um objeto adaptativo pode ser mostrado na Figura 31. A figura representa o diagrama de seqüências para um reconhecimento de uma cadeia válida ("*bac*"), como no exemplo do protótipo. É importante ressaltar a ação de três mecanismos adaptativos adaptativos (*RedutorB*, *RedutorA* e *RedutorC*) no dispositivo adaptativo, excluindo os reconhecedores de *a*, *b* e *c* na medida que um caractere é reconhecido e consumido. Assim, num primeiro momento, o reconhecedor adaptativo é do tipo *ReconhecedorABC*, para depois de

consumir um *b* se tornar num *ReconhecedorAC*, e assim por diante, até se tornar um reconhecedor de cadeia nula e reconhecer a entrada.

Com isso o diagrama representa o que cada objeto terá que fazer para realizar uma função do sistema, incluindo possíveis ações adaptativas. Este modelo, diferente do de classes, é dinâmico (no sentido adaptativo), sendo alterado por ações adaptativas. Assim, de acordo com a ordem da aplicação dos mecanismos adaptativos, um novo comportamento ocorre, fazendo com que o diagrama de seqüência tenha várias possibilidades de descrição do comportamento de um objeto num certo contexto. Note que, diferente dos outros modelos estudados na Tecnologia Adaptativa que apresentam todo tipo de alteração no modelo, este modelo apenas representa alterações no comportamento do sistema.

É importante ressaltar que este diagrama representa um possível caso para um objeto adaptativo, que está em um determinado contexto e tem um determinado histórico. Diferentes objetos em diferentes contextos poderiam se comportar de maneira diferente da apresentada, e precisariam de outro diagrama de seqüências para a representação de seu comportamento. De qualquer modo, a representação de um comportamento com o uso de mecanismos adaptativos é simples, apresentando o comportamento adaptativo de maneira adequada.

5.5.3 DIAGRAMA DE TIPOS ADAPTATIVOS

O diagrama de tipos adaptativos tem por objetivo mostrar como as ações adaptativas agem sobre o objeto em que atuam, representando as possibilidades de tipos que um objeto pode assumir durante seu ciclo de vida. Desta maneira, é possível ter uma noção de como as mudanças de tipo causadas pelos mecanismos adaptativos ocorrem no ciclo de vida de um objeto.

Para a representação dos tipos no ciclo de vida de um objeto serão utilizadas duas entidades da UML 2.0:

- Tipo, como a principal entidade no diagrama. Apesar de o Tipo ser definido na UML 2.0, não existe uma representação específica para esta entidade, aparentemente pelo

fato de a UML representar linguagens baseadas em classes que possuem relacionamento unário com tipos e, portanto, a representação de tipos e classes pode ser a mesma. Para o presente trabalho não será proposto nenhuma nova representação, sendo utilizada a representação clássica de classe com o estereótipo <<*type*>> correspondente.

- Associação, como base para a criação de um estereótipo para representar uma troca de tipo, indicando qual mecanismo adaptativo do tipo original é responsável pela mudança para o tipo de destino.

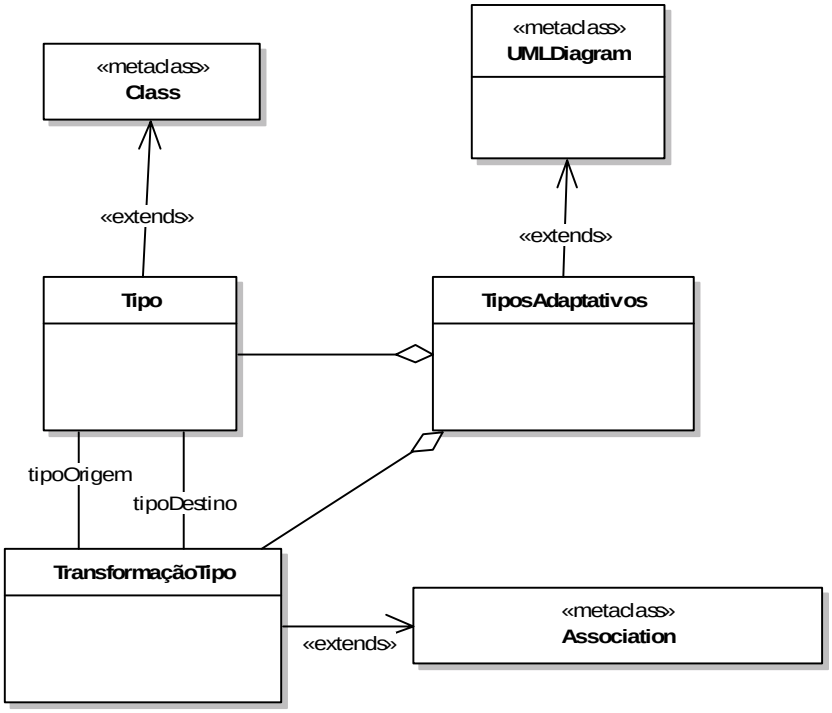

Figura 32 – Perfil UML do Diagrama de Tipos Adaptativos

Assim, o perfil UML para o diagrama de tipos pode ser visualizado na Figura 32. Nela pode-se perceber a criação de um novo diagrama como extensão da meta-classe *UMLDiagram*, denominado TiposAdaptativos, um estereótipo para classe representando os tipos de um objeto (Tipo) e um estereótipo de transformação de tipo (TransformaçãoTipo). Tipo é uma extensão da meta-classe *Class* e TransformaçãoTipo é uma extensão da meta-classe UML *Association*.

Deste modo pode-se criar um diagrama como o apresentado no item 5.4, na Figura 25, para representar os tipos do ciclo de vida de um reconhecedor de uma cadeia com um *a*, um *b* e um *c*.

6 CONCLUSÕES E TRABALHOS FUTUROS

A aplicação da tecnologia adaptativa não foi realizada em sua completude, pois não foi aplicada em um formalismo. Algumas de suas idéias foram utilizadas para criar um arcabouço com funções simulando as características do dispositivo adaptativo. Mesmo assim a tecnologia adaptativa mostrou-se muito útil na criação deste arcabouço, simplificando em alguns aspectos a construção de sistemas com características adaptativas em relação, por exemplo, ao Modelo Adaptativo de Objetos.

Por ter sido uma aplicação prática de parte dos conceitos da tecnologia adaptativa, não é possível tirar conclusões mais sólidas sobre a aplicação do dispositivo adaptativo em um formalismo com tipos, como ocorrido. Entretanto, a idéia mostrou-se viável, à medida que o arcabouço construído apresentou-se funcional para a construção de um exemplo em que as ações adaptativas são utilizadas.

Deve-se ressaltar que a presente solução não é completa, sendo válida apenas no escopo do exemplo apresentado e poderá ser funcional para a construção de alguns outros sistemas citados no texto. Isso ocorre porque a solução baseou-se em um estudo prático e as limitações impostas pela solução, como a criação de mecanismos adaptativos, devem ser melhor analisadas quando do estudo da extensão do formalismo de objetos com o dispositivo adaptativo.

Desta forma, um importante trabalho futuro a ser feito é a extensão de um formalismo de objetos com o dispositivo adaptativo, com o tratamento apropriado da questão dos tipos e procurando uma solução completa para a extensão do conceito de objetos com o dispositivo adaptativo. Para isso, vale lembrar que alguns aspectos de como isso pode ser feito está destacado no item **Erro! Fonte de referência não encontrada.**, e um estudo aprofundado da teoria de objetos apresentada em (ABADI; CARDELLI, 1996) e outros formalismos de objetos é necessário para tal trabalho. Além disto, alguns aspectos da tecnologia adaptativa não foram tratados com sua devida importância, como a questão do indeterminismo sugerido por ela, sendo necessário um estudo bem detalhado das características do dispositivo adaptativo (NETO, 2001) para que ele seja aplicado corretamente ao formalismo de objetos que for estendido.

Em relação ao Modelo de Objetos Adaptativos (AOM), é possível dizer que os objetos adaptativos apresentam ganhos significativos em relação ao seu uso. A encapsulação dos seus principais padrões de projeto em um arcabouço, por exemplo, simplifica a tarefa de construção de um sistema com características adaptativas, já que boa parte da programação necessitada pelo AOM está pronta para ser utilizada.

Outra vantagem dos objetos adaptativos em relação ao AOM é a simplificação da abstração. Enquanto no AOM, um objeto adaptativo é simulado através de um conjunto de instâncias de diferentes classes, o arcabouço torna isso transparente ao programador, que passa a se preocupar como uma única entidade.

Além do encapsulamento de boa parte dos padrões de projeto e da simplificação da abstração, este trabalho apresentou uma simplificação para a representação de características adaptativas em alguns diagramas UML. Como o AOM está fundamentado em padrões de projeto, que normalmente são soluções estruturais em diagramas de classes, a representação se torna complexa conforme aumenta o número de características adaptativas de um sistema e o de padrões de projeto da solução. Com o uso de perfis UML e da noção de mecanismos adaptativos foi possível simplificar a representação de características adaptativas em diagramas de classe. Desta forma, uma das vantagens da tecnologia adaptativa foi obtida neste caso: os modelos tornaram-se mais expressivos. Para representar o mesmo comportamento adaptativo, um diagrama de classes do AOM precisa de um conjunto maior de classes do que na representação criada, em que a uma classe pode apresentar as características adaptativas necessárias.

A representação do comportamento foi outro ganho em relação ao AOM. Enquanto os trabalhos do Modelo Adaptativo de Objetos não fazem menção à representação do comportamento, este trabalho procurou evidenciar o comportamento através do uso dos diagramas de seqüência que, com a utilização de alguns conceitos trazidos da classificação dinâmica, foi suficiente para representar o comportamento adaptativo de maneira adequada.

Outro aspecto interessante foi a criação de um novo diagrama, para representar o aspecto dinâmico dos tipos. Como a UML separa a modelagem orientada a objetos em diferentes

visões, com diferentes diagramas para cada aspecto do sistema, foi necessária a criação de um novo diagrama para representar a questão das alterações do tipo de um objeto em seu ciclo de vida.

Como dito, apenas alguns diagramas da UML foram revistos para a representação do comportamento adaptativo. Assim, outro trabalho futuro interessante é o estudo de como esse comportamento deve ser representado nos diagramas não estudados, de forma que a modelagem de um sistema com características adaptativas possa ser feita inteiramente e de maneira adequada, com os diagramas da UML.

Outra limitação da presente solução é a questão do desempenho do dispositivo adaptativo. Apesar de não ter sido medido o desempenho do dispositivo, é possível se afirmar que o seu desempenho não é o melhor possível pois se trata de uma solução baseada em meta-modelos necessitando de interpretadores, que causam uma sobrecarga desnecessária de processamento. Assim, um outro trabalho a ser realizado, visando obter melhor desempenho dos objetos adaptativos poderia ser a criação de uma máquina virtual orientada a objetos adaptativos, ou a extensão de uma máquina virtual orientada a objetos existente, como a Máquina Virtual Java ou a Máquina Virtual .NET, da Microsoft.

Outra possibilidade para um melhor aproveitamento de recursos é a construção de um sistema com adaptatividade baseada em classes, como sugerido nos itens 5.1.2.2 e 5.3.2. Porém, a representação aqui tratada terá que ser revista, uma vez que este tipo de adaptatividade traz mudanças estruturais, não tratadas pelas extensões da UML descritas neste texto. Esta solução deve ser mais adequada para um conjunto de problemas que a apresentada por este trabalho sendo um estudo de grande importância.

REFERÊNCIAS BIBLIOGRÁFICAS

Abadi, M., Cardelli, L., **A Theory of Objects (monographs in computer science).** Springer-Verlag New York, Inc., 1996.

Arsanjani, A. **Rule Object Pattern Language**. Proceedings of PLoP2000. Technical Report #wucs-00-29, Dept. of Computer Science, Washington University Department of Computer Science, October 2000.

Chiba, S. **Javassist --- A reflection-based programming wizard for Java.** In *Proceedings of the ACM OOPSLA'98 Workshop On Reflective Programming in C++ and Java*. October, 1998.

Dantas, A., Yoder, J.W., Borba, P., Johnson, R. **Using Aspects to Make Adaptive Object-Models Adaptable.** Workshop Position Paper; ECOOP '04. Oslo Norway. 2004.

Dony, C., Malenfant, J., Cointe, P. **Prototype-based languages: From a new taxonomy to constructive proposals and their validation.** Procedings of the ACM Conference on Object-Oriented Programming Systems, Languages, and Applications; pp. 201-217. 1992.

ECMA International. **Standard ECMA-335. Common Language Infraestructure (CLI) Particions I to IV.** June, 2006.

Forman, I. R., Forman, N. **JAVA Reflection IN ACTION**. Manning Publication Co., Greenwich, CT, 2005.

Foote, B., Yoder, J.W.. **Metadata and Active Object Models**. Proceedings of Plop98. Technical Report #wucs-98-25, Dept. of Computer Science, Washington University Department of Computer Science, October 1998. URL: http://jerry.cs.uiuc.edu/~plop/plop98.

Gamma, E., Helm, R., Johnson, R., Vlissides, J. **Design patterns: elements of reusable object-oriented software.** Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 1995.

Golm, M. e Kleinöder J. **MetaJava-A Platform for Adaptable Operating-System Mechanisms**, Tech. Report TR-I4-97-12, Universität Erlangen-Nürnberg: IMMD IV, Aug. 1997; Presented at the ECOOP 97 Workshop on Object-Orientation and Operating Systems, June 10, 1997, Jyväskylä, Finland.

Herrington, J. **Code Generation In Action.** Manning Publications. July, 2003.

Iwai, M. K. **Um formalismo gramatical adaptativo para linguagens dependentes de contexto.** Tese de Doutorado, USP, São Paulo, 2000.

Iwai, M. K. e Neto, J. J. **Introdução às Gramáticas Adaptativas.** Boletim Técnico, PCS-POLI-USP, São Paulo, Brasil, 2000.

Jacobson I. **Object-Oriented Sotware Engineering.** Addison Wesley Publishing Company, 1992.

Jacobson I., Booch G., and Rumbaugh J. **The Unified Software Development Process.** Addison Wesley Publishing Company, 1999.

Johnson, R., Woolf, B. "**Type object". Pattern languages of program design 3.** Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 1998.

Junior, J. R. A. **Stad: uma ferramenta para representação e simulação de**

sistemas através de statecharts adaptativos. Tese (Doutorado). Escola Politécnica, Universidade de São Paulo, São Paulo, Brasil, 1995.

Kiczales, G. **Aspect-Oriented Programming.** ACM SIGPLAN Notices, Vol. 28, n. 4, December, 1996.

Liskov, B. H., Guttag, J. **Abstraction and specification in program development.** MIT Press. 1987.

Milner, R. **A theory of type polymorphism in programming.** Journal of Computer and System Sciences 17(3); pp. 348-375. 1978.

Milner, R., Tofte, M., Harper, R. **The definition of Standard ML.** MIT Press. 1989.

NCITS. ANSI/INCITS 319-1998: Programming Language Smalltalk. InterNational Committee for Information Technology Standards (formerly NCITS). 01-jan-1998.

Neto, J. J. **Adaptive Rule-Driven Devices - General Formulation and Case Study.** Lecture Notes in Computer Science. Watson, B.W. and Wood, D. (Eds.): Implementation and Application of Automata 6th International Conference, CIAA 2001, Vol.2494, Pretoria, South Africa, July 23-25, Springer-Verlag, 2001, pp. 234-250.

Neto, J. J. **Solving Complex Problems With Adaptive Automata.** Lecture Notes in Computer Science. S. Yu, A. Paun (Eds.): Implementation and Application of Automata 5th International Conference, CIAA 2000, Vol.2088, London, Canada, Springer-Verlag, 2000, pp.340.

Neto, J. J**. Adaptive Automata for Context -Sensitive Languages.** SIGPLAN NOTICES, Vol. 29, n. 9, pp. 115-124, September, 1994.

Neto, J. J. e Basseto, B. A. **A Stochastic Musical Composer Based on Adaptative Algorithms.** Proceedings of the 6th Brazilian Symposium on Computer Music - SBC&M99, Rio de Janeiro, 1999.

Neto, J. J.; Junior, J. R. A. **Modeling adaptive reactive systems.** In: International Conference on Applied Modelling and Simulation. Cairns, Australia: [s.n.], 1999.

Neto, J. J.; Junior, J. R. A. **Using adaptive models for systems descripton**. In: *International Conference on Applied Modelling and Simulation*. Cairns, Australia: [s.n.], 1999.

Neto, J. J.; Junior, J. R. A.; Santos, J. M. N. dos. **Synchronized statecharts for reactive systems**. In: *Proceedings of the IASTED International Conference on Applied Modelling and Simulation*. Honolulu, Hawaii: [s.n.], 1998. p. 246.251.

Neto, J. J. e Magalhães, M. E. S. **Reconhecedores Sintáticos - Uma Alternativa Didática para Uso em Cursos de Engenharia.** XIV Congresso Nacional de Informática, pp. 171-181, São Paulo, 1981.

Neto, J. J., Moraes, M. **Using Adaptive Formalisms to Describe Context-Dependencies in Natural Language.** Lecture Notices in Artificial Intelligence. N.J. Mamede, J. Baptista, I. Trancoso, M. das Graças, V. Nunes (Eds.): Computational Processing of the Portuguese Language 6th International Workshop, PROPOR 2003, Volume 2721, Faro, Portugal, June 26-27, Springer-Verlag, 2003, pp 94-97.

Neto, J. J., Pariente, C. B. and Leonardi, F. **Compiler Construction - a Pedagogical Approach.** Proceedings of the V International Congress on Informatic Engineering - ICIE 99, Buenos Aires, Argentina, 1999.

Odell J., Parunak H., Brueckner S., Sauter J. **Changing Roles: Dynamic Role Assignment.** In *Journal of Object Technology*, vol 2, no. 5, September-October 2003, pp. 77-86.

OMG - Object Management Group. **Unified Modeling Language: Superstructure.** August, 2005

Opperman, R.; Rashev, R.; Kinshunk. **Adaptability and Adaptivity in Learning Systems**, Knowledge Transfer, V. 2, 1997.

Ortin, F., Redondo, J.M. **Designing an Adaptable Heterogeneous Abstract Machine by means of Reflection.** Elsevier *Information and Software Technology,* Volume 47, Issye 2, pp. 81-94. February 2005.

Ortin, F., Redondo, J. M., Vinuesa, L., Cueva, J. M. **Adding Structural Reflection to the SSCLI.** *Journal of .Net Technologies,* Volume 3, Numer 1-3, pp. 151-162. May 2005.

Parnas, D. L. **On the criteria to be user in decomposing systems into modules.** Communications of the ACM 15(12); pp. 1053-1058. 1972.

Pistori, H. **Tecnologia Adaptativa em Engenharia de Computação: Estado da arte e aplicações.** Tese (Doutorado) Edição Revisada. – Escola Politécnica, Universidade de São Paulo, São Paulo, Brasil, 2003.

Pistori, H.; Martins, P. S.; Pereira, M. C.; Castro JR., A.A.; Neto, J.J. **SIGUS – Plataforma de Desenvolvimento de Sistemas para Inclusão Digital de Pessoas com Necessidades Especiais.** IV Congresso Iberdiscap: Tecnologias de Apoio a Portadores de Deficiência, Vitória, Fevereiro 20-22, 2006.

Pistori, H., Neto, J. J. **An Experiment on Handshape Sign Recognition using Adaptive Technology: Preliminary Results.** XVII Brazilian Symposium on Artificial Intelligence - SBIA'04. São Luis, September 29 - October 1, 2004 [SBIA Website] **(to appear in Springer-Verlag Lecture Notes)**

Pistori, H.; Neto, J. J. **AdapTree - Proposta de um Algoritmo para Indução de Árvores de Decisão Baseado em Técnicas Adaptativas.** Anais Conferência Latino Americana de Informática - CLEI 2002. Montevideo, Uruguai, Novembro, 2002.

Rocha, R. L. A. **Uma proposta de uso de tecnologia adaptativa para simulação de redes neurais em um dispositivo computacional.** In: IX Encuentro Chileno de Computación 2001, Punta Arenas. Proceedings of the Encuentro Chileno de Computación. Punta Arenas: Universidad de Magallanes, v. CD-ROM, p. 1-9, 2001.

Rocha, R. L. A. e Neto, J. J. **Autômato adaptativo, limites e complexidade em comparação com máquina de Turing.** In: Proceedings of the second Congress of Logic Applied to Technology - LAPTEC'2000. São Paulo: Faculdade SENAC de Ciências Exatas e Tecnologia, p. 33-48, 2001.

Santos, J. M. N. d. **Um formalismo adaptativo com mecanismo de sincronização para aplicações concorrentes**. Dissertação (Mestrado). Escola Politécnica, Universidade de São Paulo, São Paulo, Brasil, 1997.

Souza, M. A. A., Hirakawa, A. R. **Robotic mapping and navigation in unknown environments using adaptive automata.**Proceedings of Internacional Conference on Adaptive and Natural Computing Algorithms – ICANNGA 2005, Coimbra, march 21-23, 2005.

Souza, M. A. A., Hirakawa, A. R., Neto, J. J. **Adaptive Automata for Mobile Robotic Mapping.** Proceedings of VIII Brazilian Symposium on Neural Networks - SBRN'04. São Luís/MA - Brazil. September 29 - October 1, 2004.

Taivalsaari, A. **Kevo, a prototype-based object-oriented language based on concatenation and module operations.** Report LACIR 92-02. University of Victoria. 1992.

Wirth, N. **Programming in Modula-2.** Springer-Verlag. 1983.

Yoder, J.W., Balaguer, F., Johnson, R. **Architecture and design of adaptive object-model.** ACM SIGPLAN Notices, Vol. 36, n. 12, pp. 50-60, December, 2001.

Yoder, J.W., Johnson, R. **The adaptive object-model architectural style.** Proceedings of the IFIP 17th World Computer Congress – TC2 Stream / 3rd IEEE/IFIP Conference Of Software Architecture: System Design, Development and Maintenance, p.3-27, August 25-30,2002.

Zuffo, F., Pistori, H. **Tecnologia Adaptativa e Síntese de Voz: Primeiros Experimentos.** Anais do V Workshop de Software Livre - WSL. Porto Alegre, , 2-5 de Junho, 2004.

PÁGINAS DE INTERNET

Microsoft – Rotor. http://msdn.microsoft.com/netframework/programming/clr/ Último acesso em junho de 2006