Condicionamento de Ambiente

Carlos da Silva dos Santos
Hammurabi das Chagas Mendes

1 Nome

Condicionamento de Ambiente

2 Contexto

O presente padrão tem como objetivo regularizar as circunstâncias de funcionamento de um determinado sistema de software, prevenindo falhas causadas por premissas não atendidas relacionadas ao seu ambiente operacional.

O que é sugerido, de uma forma mais precisa, é a interceptação das interações entre o sistema "condicionado" e o sistema no qual ele está contextualizado. Estas interações podem se dar, por exemplo, através do fluxo de dados trocados entre os dois.

A necessidade de condicionamento das interações entre os sistemas deve-se ao fato de que um deles adota particularidades de funcionamento que não são necessariamente assumidas corretamente pelo outro.

3 Problema

Todo sistema físico precisa que certas condições ambientais sejam preenchidas para seu funcionamento correto. O corpo humano necessita que sua temperatura seja mantida em torno de 36 graus Celsius. Aparelhos elétricos requerem alimentação por uma fonte de energia de características determinadas (p. ex. corrente alternada com 110 V de tensão). É comum também que os sistemas físicos sejam dotados de subsistemas independentes que garantam o atendimento dessas condições ambientais. Caso tenha sua temperatura muito elevada, o corpo humano passa a secretar suor para promover a perda de calor. Os sistemas elétricos apresentam toda a sorte de proteções contra surtos de tensão ou corrente excessivas, incluindo fusíveis e disjuntores.

Assim como os sistemas físicos, sistemas de software somente podem operar corretamente quando certas condições de ambiente são preenchidas: formato e tamanho esperados de entrada, quantidade de memória disponível, latência de rede, etc [3].

Uma estratégia para se lidar com tais restrições é tornar sistema em questão responsável por controlar a aplicação das restrições. Por exemplo, uma função de biblioteca pode verificar os parâmetros de entrada antes da execução, retornando um código de erro caso haja algum parâmetro inválido.

O problema com essa abordagem é que nem todas as restrições podem ser previstas no momento em que o módulo é codificado. Pode ser que as restrições sejam variáveis com o tempo ou dependam do contexto de execução: um módulo A pode estar sujeito a um conjunto de restrições quando coopera com o módulo B e a outro conjunto diferente

quando interage com o módulo *C*. Essa situação é mais grave quando lidamos com sistemas legados ou adquiridos de terceiros, em que não há possibilidade de modificar o código do sistema.

O problema consiste então em garantir as condições necessárias para o funcionamento de um módulo de software, de forma flexível e que possa aceitar componentes "caixa-preta".

4 Forças

O Condicionamento de Ambiente supõe ser utilizado em situações onde é necessário controlar e contextualizar o comportamento de um fragmento de software frente a outro. Porém existem diversas condições, vantagens e desvantagens relacionadas a seu uso. Nesta seção, estes aspectos serão apresentados.

Podemos citar várias forças que favorecem a adoção do padrão:

- Necessidade de uso de sistemas ou componentes desenvolvidos por terceiros, que não permitam modificação direta caso haja a necessidade.
- Necessidade de controlar a entrada ou validar a saída de um sistema ou componente baseado em regras que variam com o tempo.
- Necessidade de uso de um conjunto de sistemas ou componentes onde cada um necessita de um condicionamento particular em relação a suas premissas.
- Necessidade de permitir que sistemas legados continuem em operação em um novo, e possivelmente hostil, ambiente operacional.
- Necessidade de favorecer comportamentos do tipo "falha-prematura" ("*fail-early*", do inglês).
- Necessidade aumentar a previsibilidade quanto aos possíveis estados do sistema.

Podemos citar também algumas forças que desfavorecem a adoção do padrão:

- Algumas vezes o condicionamento ou validação do fluxo de dados entre os atores podem ser computacionalmente caros, por exemplo, quando o fluxo supõe dados ordenados segundo algum parâmetro.
- Mesmo quando o último caso não se aplica, há perdas de eficiência inerentes à existência do mecanismo de condicionamento externo, principalmente se há um grande número de condições envolvidas.

5 Solução

A solução para o problema apresentado consiste em desacoplar do sistema ou componente controlado as responsabilidades pela consistência dos dados trocados entre ele e o ambiente que o envolve. Além disso, deve-se atuar explicitamente sobre este ambiente, de forma a torná-lo condizente com o modo de operação do sistema controlado.

Mais especificamente, uma série de medidas podem ser adotadas em conjunto ou separadamente. São elas:

1. Criar *filtros de entrada* que validam ou mesmo modificam a entrada direcionada ao sistema controlado.
2. Criar *guardas de saída* que validam ou mesmo modificam a saída oriunda do sistema controlado.
3. Criar entidades que validam ou mesmo modificam o ambiente de operação do sistema controlado de forma a atender suas premissas de funcionamento.

6 Estrutura

Na sua versão mais genérica, o condicionamento de ambiente envolve os seguintes participantes (ver figura 1):

- Condicionado: Unidade de software que realiza uma função de interesse, cujo comportamento queremos controlar. Primordialmente, esse controle é feito por meio de ações sobre o ambiente. De maneira opcional, o *Condicionado* pode apresentar interfaces próprias de medição e controle que possibilitem um controle mais direto.
- Ambiente: Elemento que interage com o *Condicionado*, seja por meio de trocas de mensagens ou através da disponibilização de alguma característica utilizada pelo último.
- Condicionante: Módulo que atua sobre o Ambiente, de modo que este mantenha-se compatível com as *RegrasDeOperação*, visando o bom funcionamento do *Condicionado*.
- RegraDeOperação: Parâmetro que representa uma premissa de funcionamento do *Condicionado*.

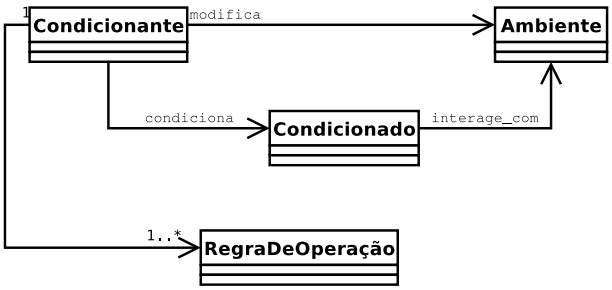

Figura 1: Estrutura básica do padrão

Em muitos contextos, as interações mais importantes entre *Ambiente* e *Condicionado* dizem respeito a entradas direcionadas ao *Condicionado* e saídas fornecidas por esse módulo a outros subseqüentes. Nesses casos se torna vantajoso conceber o Condicionamento de Ambiente como aplicação de *filtros de entrada* e *guardas de saída*.

A figura 2 representa o uso de um filtro de entrada. Com relação à figura 1, o papel de *Ambiente* é representado pelo *Cliente* que submete entradas a serem processadas pelo *Condicionado*. O *Condicionante* toma a forma de um *Filtro de Entrada*, que intercepta as requisições do *Cliente* e eventualmente modifica os parâmetros recebidos,

garantindo que esses se limitem à faixa de valores aceitos pelo *Condicionado*, conforme descrito pela *Regra de Operação*.

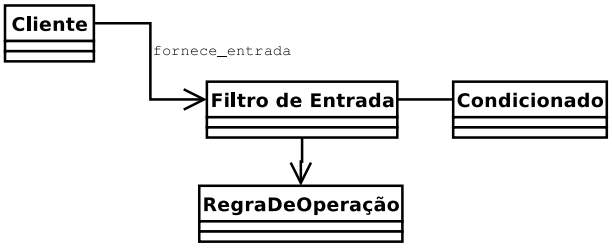

Figura 2: Especialização da estrutura para filtragem de entrada

A figura 3 representa o uso de um guarda de saída. Com relação à figura 1, o papel de *Ambiente* é representado pelo *Cliente* que recebe resultados de computações realizadas pelo *Condicionado*. O *Condicionante* toma a forma de um *Guarda de Saída*, que recebe as saídas processadas pelo *Condicionado*. O Guarda de Saída verifica se as saídas obedecem à *Regra de Operação*, evitando a propagação de valores que a violem.

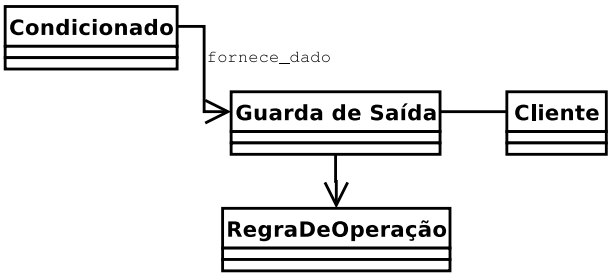

Figura 3: Especialização da estrutura para implementar guardas de saída

Embora tenhamos apresentado a estrutura do padrão por meio de diagramas de classes, a forma exata de implementação depende da natureza do módulo condicionado. Num nível mais baixo, o padrão proposto pode ser utilizado para proteger o acesso a uma estrutura de dados ou um método. Num nível mais alto, serviços ou aplicações inteiras podem fazer o papel de módulo condicionado.

Como exemplo de implementação não baseada em classes, a proteção de uma chamada de método pode ser realizada com o auxílio de Programação Orientada a Aspectos (AOP). Nesse caso as regras para o condicionamento seriam confinadas em aspectos a serem injetados no módulo condicionado.

7 Usos conhecidos

7.1 Prevenção de ataques baseados em injeção em SQL

Um ataque baseado em injeção em SQL (*SQL injection attack*) é uma técnica que que se aproveita da falta de filtragem das informações de entrada em sistemas que geram comandos SQL para manipular conjuntos de dados. Tipicamente, estas informações de entrada são oriundas dos próprios usuários finais do sistema. Usuários mali-

ciosos podem então se aproveitar da falta de tratamento de entradas e inserir strings que geram, por sua vez, comandos SQL de caráter nocivo à camada de dados da aplicação.

Aplicações web que implementam busca em uma determinada base de dados são muito comuns, e as consultas correspondentes devem ser, por definição, geradas a partir de entrada dos usuários. Consequentemente, várias abordagens para o tratameto deste problema já foram apresentadas. Um exemplo concreto desta abordagem é o Perl DBI [1].

7.2 Programação ignorante quanto a Falhas

A fim de minimizar o efeito de erros de acesso a memória em programas escritos na linguagem C, Rinard *et al* [5] propuseram um compilador especial que insere pontos de verificação em meio ao código. O código inserido mantém um registro das estruturas alocadas em memória, sendo capaz de antever a ocorrência de acessos de memória ilegais. Em tais casos, o acesso ilegal é bloqueado, sendo substituído por uma das seguintes medidas:

- Acessos para escrita (*write*) são simplesmente desprezados.
- Leituras (*reads*) ilegais são trocadas pela fabricação de um valor, computado com base em heurísticas.

O compilador descrito acima foi aplicado [5] a alguns programas populares da plataforma Unix (Apache HTTP Server, Sendmail, Mutt, etc), gerando os seguintes efeitos:

- Os servidores se tornam imunes a erros de memória.
- Garante-se operação contínua, requisições legítimas não são afetadas pelas de conteúdo malicioso.

A abordagem adotada foi batizada de "Ignorante quanto a Falhas" (*failure oblivious*) porque converte situações de erro não-previstas em casos previsíveis e corretamente gerenciados pelo programa; desse modo a operação normal é garantida mesmo frente à ocorrência de falhas.

8 Conseqüências

- O Condicionamento de Ambiente evita a reescrita de software, substituindo-a (ao menos parcialmente) pela manipulação do ambiente de execução. A estratégia defendida pelo padrão é possivelmente mais rápida e de menor custo que a abordagem em contrapartida.
- Aumento da disponibilidade dos sistemas afetados [3].
- Maior previsibilidade quanto ao comportamento do software, pois o número de estados possíveis dele é diminuído [3].
- Perda de desempenho imposta pela verificação de entradas/saídas e pela manipulação do ambiente de execução.

9 Padrões relacionados

Um padrão bastante relacionado ao "Condicionamento de Ambiente" é o "Interceptador" [6]. Basicamente, as diferenças são os motivos pelos quais ambos são empregados.

Os filtros de entrada e os guardas de saída podem ser implementados por meio do padrão "Pipes and Filters" [2].

O padrão "Estratégia" [4] pode ser utilizado para se implementar regras de operação variáveis em termos de tempo ou contexto.

Referências

[1] http://dbi.perl.org/, acessado em outubro/2005.

[2] F. Buschman, R. Meunier, H. Rohnert, P. Sommerland, and M. Stal. *Pattern-Oriented Software Architecture: A system of Patterns*. John Wiley & Sons, 1996.

[3] George Candea. Predictable software – a shortcut to dependable computing. Technical report, Stanford University, março 2004. arXiv.org:cs.OS/0403013.

[4] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[5] Martin Rinard, Cristian Cadar, Daniel Dumitran, Daniel M. Roy, Tudor Leu, , and William S. Beebee Jr. Enhancing server availability and security through failure-oblivious computing. In *Proceedings of the 6th Symposium on Operating Systems Design and Implementation*, dezembro 2004. http://www.cag.lcs.mit.edu/ rinard/paper/, acessado em outubro/2005.

[6] D. Schmidt, M. Stal, H. Rohnert, and F. Buschmann. *Pattern-Oriented Software Architecture: Patterns for Concurrent and Distributed Objects*. John Wiley & Sons, 2000.