ArchPattern: reconhecimento de Padrões Arquiteturais em Sistemas Java

Aline Pires[*]
Fernando Carvalho[**]
Leonardo Barroso[***]
Vinícius Seufitele[****]

Palavras-chave: Arquitetura de *software*. Padrões arquiteturais. Recuperação de arquitetura. Análise estática. Análise dinâmica. Sistemas Java.

1 Introdução

O processo de recuperação de arquitetura vem sendo estudado, com o objetivo de apoiar a manutenção de sistemas existentes, visto que esta etapa é a mais custosa no ciclo de vida de um software, consumindo até 60% do custo total do orçamento da empresa desenvolvedora (PRESSMAN, 2005). Tal processo é uma atividade de engenharia reversa cuja finalidade é obter uma arquitetura documentada para um sistema existente (DEURSEN *et al.*, 2004). Sua motivação provém da necessidade de compreender esses sistemas para fins de manutenção, reengenharia, reutilização, etc. Sistemas existentes, usualmente chamados de sistemas legados, normalmente não possuem uma documentação arquitetural atualizada para apoiar estas atividades.

Um processo de recuperação de arquitetura se constitui de 3 fases (VASCONCELOS, 2007), a saber: extração de informações das fontes disponíveis do sistema, tais como código fonte, documentação, rastros de execução, executável e especialistas; abstração destas informações para o nível arquitetural; e, posteriormente, a apresentação dos resultados obtidos. Sistemas legados costumam apresentar um grande volume de informação e, por este motivo, a fase de abstração envolve alto esforço e custo se técnicas e/ou regras apropriadas não forem empregadas.

O processo de recuperação de arquitetura é semi-automatizado e depende, principalmente, das regras ou heurísticas da fase de abstração. Em grande parte das abordagens encontradas na literatura, estas são específicas para um domínio (KAZMAN; CARRIÈRE, 1997), linguagem de programação (SCHMERL *et al.*, 2006) ou então falham em serem bem definidas (RIVA; RODRIGUEZ, 2002). Vasconcelos (2007) propõe um processo de recuperação de arquitetura em que as regras para agrupamento de classes em elementos arquiteturais são genéricas em relação a domínio e linguagem de programação, porém o ferramental desenvolvido é específico para sistemas orientados a objetos escritos em Java. A visão arquitetural recuperada é funcional, e cada elemento arquitetural representa um subsistema do domínio. Entretanto, abordagens de engenharia reversa genéricas em relação a

[*] Doutora em Engenharia de Sistemas e Computação/COPPE-UFRJ. Professora do CEFET Campos.
[**] Graduado no curso de Tecnologia em Desenvolvimento de *Software* pelo CEFET Campos.
[***] Graduado no curso de Tecnologia em Desenvolvimento de *Software* pelo CEFET Campos.
[****] Graduado no curso de Tecnologia em Desenvolvimento de *Software* pelo CEFET Campos.

domínio e linguagem de programação são difíceis de serem obtidas (VASCONCELOS, 2007). Há trabalhos da literatura que detectam padrões em sistemas legados, mas focam em padrões de baixa granularidade, como os de projeto (HEUZEROTH *et al*., 2003) (CORREA *et al*., 2000), ou detectam padrões específicos de uma linguagem de programação (YEH *et al*., 1997).

Este trabalho propõe uma abordagem de recuperação de arquitetura de sistemas legados escritos em Java, por meio da detecção de padrões em nível de granularidade arquitetural reconhecidos na literatura, como o MVC (*Model-View-Controller*) (BUSCHMANN, 1996) e Camadas (*layers*). A recuperação se baseia na definição de regras genéricas no que diz respeito a domínio, no entanto específicas para sistemas escritos em Java, baseando-se em estruturas sintáticas da linguagem e padrões de codificação. A abordagem proposta combina as análises estática e dinâmica, utilizando-se da primeira para coletar as evidências para a identificação do padrão, que serão, posteriormente, validadas pela segunda, a partir de rastros de execução (*execution traces*) coletados durante a execução do sistema.

O objetivo deste trabalho é indicar que classes ou pacotes da aplicação representam que elementos de um padrão arquitetural, facilitando, assim, a compreensão do sistema. A fim de apoiar a realização da abordagem, desenvolveu-se uma ferramenta de apoio à execução: *ArchPattern*, apresentando resultados em nível de granularidade de classes ou pacote, facilitando a análise e compreensão de sistemas de grande escala.

Partindo desta Introdução, o restante do artigo está organizado da seguinte forma: a Seção 2 apresenta a abordagem proposta para a identificação de padrões arquiteturais, isto é, *ArchPattern*; a Seção 3 apresenta seu ferramental de apoio com um exemplo de uso da abordagem; e a Seção 4 apresenta as conclusões e trabalhos futuros.

2 Detecção de Padrões Arquiteturais em Sistemas Java

Inicialmente, foram desenvolvidas regras para a detecção do padrão MVC (BUSCHMANN, 1996) na abordagem *ArchPattern*. O padrão MVC se compõe de 3 tipos de elementos arquiteturais, a saber: Modelo, que representa o modelo do negócio; Controlador, que trata os eventos disparados pelos usuários na Visão e é notificado sobre mudanças no Modelo; e Visão, que representa a interface com o usuário e deve refletir o estado atual do Modelo, sendo notificada por quaisquer mudanças. No padrão MVC, o Controlador e a Visão são observadores de um Modelo, conforme descrito por Buschmann (1996).

A capacidade de detecção do padrão está relacionada à correta identificação das características de cada um dos seus elementos. A Tabela 1 estabelece como as características do padrão MVC são mapeadas, para as regras da ferramenta ArchPattern, a fim de permitir a sua identificação nos sistemas

alvo. Convém ressaltar que a Tabela 1 apresenta um resumo das regras tratadas na abordagem proposta, sendo que um detalhamento completo destas regras pode ser obtido em Carvalho *et al*. (2008).

Como pode ser observado, as regras apresentadas na Tabela 1 são baseadas nas características do padrão MVC extraídas de (BUSCHMANN, 1996) e no seu mapeamento para estruturas sintáticas e padrões de codificação da linguagem Java. Todas as regras possuem suporte por meio da análise estática, mas somente algumas, indicadas na segunda coluna da Tabela 1, possuem o suporte da análise dinâmica. A coluna M representa o elemento Modelo do padrão MVC, a V o elemento Visão, e a C o elemento Controlador que compõe o padrão. Cada regra está relacionada aos elementos que contribuem para reconhecimento. As regras da análise estática verificam a existência de estruturas (classificadores ou pacotes) e seus relacionamentos no código fonte do sistema alvo. As regras da análise dinâmica verificam o comportamento dos objetos em tempo de execução, confirmando as evidências obtidas pela análise estática.

Existe uma inter-relação entre as regras apresentadas. Para identificar um elemento arquitetural, a ferramenta leva em consideração o conhecimento de outros elementos. Uma Visão, por exemplo, precisa receber, em sua construção, um objeto Modelo. Para tanto, é necessário avaliar regras que identifiquem objetos candidatos a Modelo. Os resultados das regras podem evoluir a cada iteração, e por isso a execução destas é iterativa, e se retro-alimentam. Dessa forma, o processamento ocorre até que o conjunto de respostas demonstre resultados inalterados, ou seja, até que o estado dos resultados das regras não se modifique, ou até um número máximo de iterações.

Tabela 1
Regras de detecção dos Elementos Arquiteturais do Padrão MVC

Regras	Dinâmica	M	V	C
Notifica Observadores	*x*	x		
Não notifica Observadores	x		x	x
Usa coleção de Observadores		x		
Estende um Objeto da GUI			x	
Não estende componentes da GUI		x		x
Manipula objetos da GUI			x	
Não manipula componentes da GUI		x		x
Implementa uma Interface do tipo *Listener*				x
Não implementa interface *Listener*		x	x	
Adiciona Observadores	x	x		
Não adiciona Observadores	x		x	x
Registra-se como Observador	x		x	x
Não se registra como Observador	x	x		
Referencia Objeto Modelo			x	x
Solicita outros serviços do Modelo	x		x	x
Recebe um objeto Modelo como parâmetro			x	x
Cria um Objeto Controlador	x		x	
Associa um Controlador como ouvinte de uma Visão			x	
Referencia um objeto Visão	x			x

Na prática, entretanto, mesmo sistemas baseados em padrões arquiteturais usualmente ferem algumas de suas especificações. Por isso, a identificação do padrão não alcança exatidão. Esta

característica é acentuada em sistemas legados, nos quais a estrutura do sistema encontra-se degradada em função das inúmeras manutenções ao longo do tempo. Por este motivo, a abordagem proposta calcula a aderência (porcentagem) das regras atendidas, por classe do sistema, e indica a probabilidade (em %) que a classe tem de representar um elemento do tipo Modelo, um Controlador, uma Visão, etc. Além disso, também é possível visualizar os resultados em nível de granularidade de pacotes. O objetivo é compreender a arquitetura do sistema por meio da identificação destes elementos arquiteturais e tornar a abordagem escalável a sistemas com um grande número de classes, em que a análise em nível de classes poderia exigir maior esforço para a compreensão do sistema.

3 Descrição da ferramenta e exemplo de uso

O ferramental que dá suporte à abordagem *ArchPattern* implementa as análises estática e dinâmica. A análise estática detecta os elementos do sistema e seus relacionamentos por meio do uso de um *parser* (interpretador) para a linguagem de programação alvo. Desenvolvida, originalmente, como extensão da ferramenta Ares (VERONESE; NETTO, 2001), esta funcionalidade foi estudada e adaptada neste trabalho, recuperando dados arquiteturais da aplicação e alimentando um metamodelo que se baseia no metamodelo da UML, contendo classes, pacotes, dependências, heranças, etc. O metamodelo do Ares foi estendido, implementando a detecção de novas estruturas como Iterações e Invocações de serviço, não tratadas anteriormente.

Neste trabalho, são adicionadas ao metamodelo informações de rastros de execução (*execution traces*) provenientes da análise dinâmica, que representam as chamadas de métodos da aplicação em um cenário de uso ou funcionalidade executada, sendo coletados durante a execução monitorada do sistema pela ferramenta Tracer (VASCONCELOS, 2007).

No ferramental proposto neste trabalho, foram utilizadas Fábricas para a construção e manipulação dos objetos deste metamodelo, como, por exemplo, a Fábrica de Arquivos, a de Classificadores, de Rastros, dentre outras. Os objetos são construídos pelas Fábricas com as informações provenientes da análise estática e da análise dinâmica e percorridos durante o processamento das regras para detecção dos elementos arquiteturais.

Utilizamos o exemplo TesteMVC para apresentação da ferramenta de apoio à ArchPattern. Ele foi desenvolvido para testes das regras do padrão MVC. O objetivo era obter resultados da ferramenta correspondentes a uma aderência máxima de cada classe a um elemento do padrão, uma vez que o sistema implementa totalmente as características do padrão MVC. A Figura 1 apresenta uma tela com os resultados aferidos para a classe TesteView, escolhida aleatoriamente entre as classes deste exemplo. Ao selecionar a classe TesteView, a ferramenta apresentou os resultados das regras com aderência de 100% às especificações do elemento Visão do padrão MVC. Ela satisfaz também às regras de outros elementos, como não implementar um escutador de eventos (*listener*) para o elemento

Modelo. No entanto, estes casos representam regras coincidentes entre diferentes tipos de elementos. O percentual de aderência a cada elemento do padrão é mostrado no final da tela.

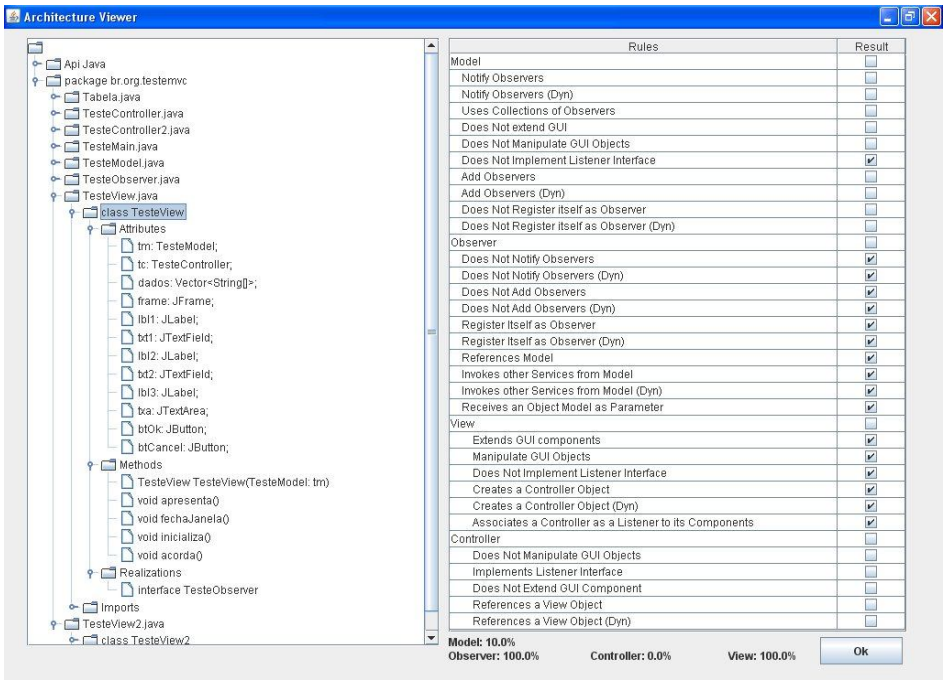

Figura 1: Sistema de Teste das Regras para a Detecção do padrão MVC por classes

A abordagem, também, pode apresentar, alternativamente, os resultados em nível de pacote, e é possível mostrar a quantidade de classificadores que atendem a cada regra para um elemento sobre o total de classificadores do pacote selecionado. O objetivo deste tipo de detecção é atender a sistemas de grande escala, com um grande número de classes, permitindo uma análise em nível de pacote. Vale ressaltar que a abordagem proposta já foi testada com outros exemplos, como a ferramenta Ares (VERONESE; NETTO, 2001), obtendo resultados satisfatórios (CARVALHO *et al.*, 2008).

4 Conclusões

Como contribuições, a *ArchPattern* permite a identificação dos elementos de um padrão arquitetural, permitindo a compreensão dos sistemas existentes escritos em Java.

Inicialmente, este trabalho propõe regras para o reconhecimento do padrão MVC, embora a abordagem proposta possa ser estendida para a detecção de outros padrões. Para isso, é necessário ampliar o conjunto de regras e implementá-las no ferramental de suporte à *ArchPattern*, uma vez que o metamodelo, juntamente com os serviços que podem ser requeridos às Fábricas, oferecem o conjunto de informações necessárias à detecção de outros padrões. Neste momento, está sendo implementada e avaliada a detecção do padrão Camadas (BUSCHMANN, 1996). Como trabalhos

futuros, *ArchPattern* deverá ser avaliada por meio de mais estudos de caso em sistemas reais, sendo utilizada por outros desenvolvedores, com o intuito de melhor avaliar e aperfeiçoar a sua capacidade de detecção, o conjunto de regras e a sua usabilidade. Devem ser avaliados também sistemas com arquiteturas híbridas e que estejam, há algum tempo, em funcionamento (isto é, sistemas legados), além de sistemas reais que não implementem um determinado padrão arquitetural investigado.

5 Referências

BUSCHMANN, F. *et al*. *Pattern-Oriented Software Architecture*: a System of Patterns, John Wiley & Sons. 1996.

CARVALHO, F. *et al.* Reconhecimento de Padrões Arquiteturais em Sistemas Java. *V Workshop de Manutenção de Software Moderna*. Florianópolis, jun. 2008.

CORREA, A.; WERNER, C.; ZAVERUCHA, G. Object Oriented Design Expertise Reuse: an Approach based on Heuristics, Design Patterns and Anti-Patterns *In*: Sixth International Conference on Software Reuse, Viena, Austria, jun. 2000, p. 336-352.

DEURSEN, A.V. *et al.* Symphony: view-Driven Software Architecture Reconstruction, Fourth Working IEEE/IFIP Conference on Software Architecture, Oslo, Norway, jun., 2004, p. 122-132.

HEUZEROTH, D. *et al*. Automatic Design Pattern Detection. *11th IEEE Internatinal Workshop on Program Comprehension*. Portland, Oregon, USA, May, 2003, p. 94-103.

KAZMAN, R.; CARRIÈRE, S. J. Playing Detective: Reconstructing Software Architecture from Available Evidence. Relatório Técnico CMU/SEI-97-TR-010, Carnegie Mellon University, out. 1997.

PRESSMAN, R. S. Software Engineering: a Practitioner's Approach. 6. ed. McGraw-Hill, 2005.

RIVA, C., RODRIGUEZ, J.V. Combining Static and Dynamic Views for Architecture Reconstruction, Sixth European Conference on Software Maintenance and Reengineering (CSMR'027), Budapeste, Hungria, Mar. 2002, p. 47-56.

SCHMERL, B. *et al*. Discovering Architectures from Running Systems. *In*: IEEE Transactions on Software Engineering, v. 32, n. 7, 2006, p. 454-466.

VASCONCELOS, A. Uma Abordagem de Apoio à Criação de Arquiteturas de Referência de Domínio Baseada na Análise de Sistemas Legados. *Tese de Doutorado*. COPPE/UFRJ, Rio de Janeiro, 2007.

VERONESE, G. O.; NETTO, F. J. *ARES*: uma Ferramenta de Auxílio à Recuperação de Modelos UML de Projeto a partir de Código Java. Projeto Final de Graduação. IM, UFRJ, Rio de Janeiro, 2001.

YEH, A.; HARRIS, D.; CHASE, M. Manipulating Recovered Software Architecture Views. *In Proc. of International Conference on Software Engineering*. Boston, MA, USA, May, 1997, p. 184-194.