ARCHJAVA: RECONHECIMENTO DE PADRÕES ARQUITETURAIS EM SISTEMAS JAVA

Fernando Carvalho, Leonardo Barroso, Vinícius Seufitele, Aline Vasconcelos

CEFET Campos (Centro Federal de Educação Tecnológica de Campos)
Dr. Siqueira, 273 – Pq. Dom Bosco – CEP: 28030-130 - Campos dos Goytacazes- RJ

fernando.carvalho@cefetcampos.br, leo.barroso@yahoo.com.br,
vinicius.seufitele@gmail.com, apires@cefetcampos.br

Abstract. *Software architecture recovery aims at supporting software comprehension for maintenance and reuse purposes. In order to comprehend a software, it is necessary to know its structure, data flow and behavior, what uses to be determined by the adopted architectural patterns. In this paper, a set of rules to architecture recovery that are domain independent is proposed. They are based on static and dynamic analysis for the recognition of architectural patterns in Java systems. The former analyses the classes structure and the latter continues the evaluation by verifying objects behavior. In order to evaluate the proposed heuristics, case studies are presented.*

Resumo. *A recuperação da arquitetura de software visa apoiar a sua compreensão para fins de manutenção e reutilização. Para compreender um software, é necessário conhecer a sua estrutura, seu fluxo de dados e seu comportamento, o que costuma ser determinado pelos padrões arquiteturais adotados. Neste artigo, é proposto um conjunto de regras para a recuperação da arquitetura, que são independentes de domínio. Elas se baseiam nas análises estática e dinâmica para o reconhecimento de padrões arquiteturais em sistemas Java. A primeira analisa a estrutura das classes e a segunda prossegue verificando o comportamento dos objetos. A fim de avaliar as regras propostas, estudos de casos são apresentados.*

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 cujo objetivo 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 há algum tempo nas empresas, normalmente chamados de sistemas legados, usualmente não possuem uma documentação arquitetural atualizada para apoiar estas atividades.

Uma das formas de se compreender a arquitetura de um software é através da identificação dos padrões arquiteturais adotados. Segundo Buschmann (1996), os padrões arquiteturais provêem um conjunto de subsistemas ou componentes pré-definidos, especificando as suas responsabilidades, restrições topológicas, restrições

sobre tipos de componentes e conectores, organizando, dessa forma, os relacionamentos e comunicação entre eles. Em relação aos sistemas legados, há uma grande dificuldade na detecção de padrões em função da degradação de sua estrutura ao longo de várias manutenções e da combinação de padrões, pois, em geral, os sistemas são híbridos.

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 são empregadas. A fase de apresentação também é complexa, pois exige a escolha de uma visão arquitetural entre vários modelos possíveis, e o volume de informações a ser exibido costuma ser grande.

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 e Carrière 1997], linguagem de programação [Schmerl *et al.* 2006] ou então falham em ser bem definidas [Riva e 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 seu ferramental é específico para sistemas orientados a objetos escritos em Java. A visão arquitetural recuperada é funcional, onde cada elemento arquitetural representa um subsistema do domínio. Entretanto, abordagens de engenharia reversa genéricas em relação a domínio e linguagem de programação são difíceis de ser obtidas [Vasconcelos 2007]. Há trabalhos da literatura que detectam padrões em sistemas legados, mas focam em padrões de baixa granularidade, como padrões de projeto [Heuzeroth *et al.* 2003] [Correa *et al.* 2000], ou detectam padrões específicos de uma linguagem de programação [Yeh *et al.* 1997].

Neste contexto, este trabalho propõe uma abordagem de recuperação de arquitetura de sistemas legados escritos em Java, através da detecção de padrões em nível de granularidade arquitetural reconhecidos na literatura, como o MVC (*Model-View-Controller*) 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, se baseando em estruturas sintáticas da linguagem e padrões de codificação. Diferente da abordagem de [Schmerl *et al.* 2006], que realiza a detecção de padrões com base na análise dinâmica, ou seja, do programa em execução, a abordagem proposta combina análise estática e dinâmica. Em relação à visão arquitetural recuperada, esta é ortogonal à visão recuperada em [Vasconcelos 2007] no sentido de que uma classe que compõe um subsistema do negócio pode representar um elemento de visão, i.e. interface gráfica, ou controlador na visão baseada em MVC.

O objetivo deste trabalho é indicar quais classes da aplicação representam quais elementos de um padrão arquitetural. Para atingir tal objetivo, as informações são provindas da análise estática e análise dinâmica, sendo utilizadas conjuntamente para identificar a presença de elementos de um determinado padrão. A estrutura de cada padrão é analisada pela análise estática, que coleta os tipos dos objetos, ou seja, classes e interfaces, os seus relacionamentos, métodos e atributos, sendo o comportamento do

padrão verificado pela análise dinâmica, a partir de rastros de execução (*execution traces*), coletados durante a execução do sistema. O trabalho ainda não adota um modelo de visão arquitetural gráfico, apresentando apenas a informação de classes *versus* padrões identificados, com as regras que apontaram a presença do padrão.

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, i.e. ArchJava, destacando as regras propostas para a identificação do padrão MVC; a seção 3 apresenta a implementação da abordagem de reconhecimento de padrões arquiteturais, descrevendo como as análises estática e dinâmica se complementam; a seção 4 apresenta dois estudos de caso, que demonstram a aplicação da abordagem; finalmente, a seção 5 apresenta conclusões e trabalhos futuros.

2. Detecção de Padrões Arquiteturais na Abordagem ArchJava

Inicialmente, foram desenvolvidas regras para a detecção do padrão MVC. O padrão MVC se compõe de 3 tipos de elementos arquiteturais, a saber [Buschmann 1996]: Modelo, que representa o modelo do negócio; Controlador, que trata os eventos disparados pelos usuários na Visão e é notificado sobre mudanças no Modelo, pois seu comportamento depende do estado do Modelo; e Visão, que representa a interface com o usuário e deve refletir o estado atual do Modelo, sendo notificada também sobre mudanças no Modelo. No MVC, o Controlador e a Visão são observadores de um Modelo (padrão de projeto *Observer* [Buschmann 1996]).

A capacidade de detecção do padrão está relacionada com a correta identificação das características de cada um dos seus elementos. As tabelas 1, 2 e 3 estabelecem como uma característica do padrão MVC é tratada através de regras para permitir a sua identificação em sistemas Java.

Tabela 1. Regras para a Detecção do Elemento Arquitetural Visão no Padrão MVC

Características do padrão	Regras baseadas na análise estática	Regras baseadas na análise dinâmica
Representa uma interface gráfica da aplicação.	* A classe pode possuir cláusula *extends* de uma classe GUI (Interface Gráfica de Usuário) do Java, como o *JFrame, JPanel, JWindow etc.* * Outra possibilidade é a referência a objetos de interface gráfica do Java para interação com o usuário, como elementos dos pacotes Swing e Awt.	Não se aplica.
Não controla eventos.	* A classe não possui o *implements* de uma classe *Listener*.	Não se aplica.
Relaciona-se com o Modelo.	* Deve referenciar o Modelo. * O Modelo deve ser obtido durante sua criação ou através de algum serviço. * Deverá se registrar junto ao Modelo. * Chama um método do modelo, sem ser do de registro, para obter dados.	* Chama o método de registro em tempo de execução. * Verifica se outro(s) método(s) que não o de registro, que pode ser para atualização de dados, é(são) chamado(s) em tempo de execução.
Relaciona-se com o Controlador.	* Cria o objeto Controlador, passando o Modelo como parâmetro. * Associa o Controlador aos seus objetos de Interface para tratar seus eventos.	* Corrobora a associação da Visão com o Controlador verificando a construção do Controlador em tempo de execução.

Como pode ser observado, essas regras são baseadas em estruturas sintáticas da linguagem e padrões de codificação. Elas foram determinadas a partir do estudo do padrão arquitetural e da sua implementação em diferentes sistemas Java. A Tabela 1 apresenta as regras baseadas na análise estática e dinâmica para a identificação do elemento Visão. A tabela 2 apresenta regras para identificação do Controlador.

Tabela 2. Regras para a Detecção do Elemento Controlador no Padrão MVC.

Características do Padrão	Regras baseadas na análise estática	Regras baseadas na análise dinâmica
Trata eventos.	* A classe implementa, através de uma clausula *implements*, um *Listener*. Ex. *ActionListener*.	Não se aplica.
Não possui interface com o usuário.	* A classe não deve possuir cláusula *extends* de uma classe GUI. * Não deve manipular objetos de interface gráfica, solicitando estes serviços à visão.	Não se aplica.
Relaciona-se com o Modelo.	* Deve referenciar o Modelo. * O Modelo deve ser obtido durante sua criação ou através de algum serviço. * Deve se registrar junto ao Modelo. * Chama um método do modelo para obter dados que não seja o de registro.	* Chama o método de registro em tempo de execução. * Verifica se outro(s) método(s) que não o de registro, ou seja, para atualização dos dados, é(são) chamado(s) em tempo de execução.
Relaciona-se com a Visão.	* Possui um atributo (ou coleção) do tipo de uma classe Visão. * Possui um método cujo parâmetro seja uma classe Visão.	* Corrobora a associação do Controlador com a Visão verificando a chamada de um método que tenha a Visão como parâmetro em tempo de execução.

Existe uma inter-relação entre as regras apresentadas em cada uma das Tabelas, ou seja, para identificar uma Visão é necessário conhecer candidatos a Controlador e candidatos a Modelo, sendo o mesmo verdadeiro para os outros elementos do padrão. Por este motivo, as duas primeiras regras da Tabela 1 são suficientes para atribuir a característica de Visão a uma classe, sendo esta posteriormente validada pelas duas últimas regras. Vale ressaltar que as regras da análise dinâmica não se restringem somente ao escopo do construtor da classe, mas também ao método onde o construtor é chamado. Cabe observar que as classes referentes à interface de usuário, tais como elementos de tela ou escutadores de eventos compõem uma lista, não exaustiva, conforme os sistemas estudados. Esta lista deve ser ampliada para acompanhar as mudanças tecnológicas e casos novos que venham a ser estudados. Como pode ser visto pelas Tabelas 1 e 2, os elementos Visão e Controlador possuem muitos aspectos em comum pelo fato de ambos serem Observadores do Modelo. A tabela 3 apresenta as regras para a detecção do Modelo.

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, onde a estrutura do sistema costuma se encontrar degradada em função das inúmeras manutenções ao longo do tempo. Por este motivo, a abordagem proposta calcula a porcentagem das regras atendidas por cada classe do sistema e indica uma probabilidade de X% que a classe tem ser um Modelo, um Controlador, uma Visão etc.

Tabela 3. Regras para a Detecção do Elemento Arquitetural Modelo no Padrão *MVC*.

Características do Padrão	Regras baseadas na análise estática	Regras baseadas na análise dinâmica
Possui uma coleção de Observadores.	* Possui um atributo cujo tipo seja uma coleção.	Não se aplica.
Notifica alterações aos Observadores.	* Possui um método que itera sobre o atributo cujo tipo seja uma coleção invocando um método sobre cada elemento Observador.	* Em tempo de execução, neste método é verificado se são disparadas mensagens para os objetos Controlador e Visão.
Adiciona um Observador.	* Um método do Modelo possui como parâmetro um objeto, que representa um Controlador ou uma Visão, e o adiciona no atributo coleção.	* É verificado em tempo de execução se o Modelo recebe uma chamada de método onde um Observador é passado como parâmetro.
Remove um Observador.	* Um método do Modelo possui como parâmetro um objeto, que representa um Controlador ou uma Visão, e o remove do atributo coleção.	Não se aplica.
Não possui interface com o usuário.	* A classe não deve possuir cláusula *extends* de uma classe GUI. * Não deve manipular objetos de interface gráfica, solicitando estes serviços à Visão.	Não se aplica.

3. Ferramenta de Suporte à Abordagem ArchJava

Nesta seção é descrito o ferramental construído para a detecção dos padrões arquiteturais com base na abordagem ArchJava. A fim de obter as informações para a detecção, foi necessário buscar e estender ferramentas de análise estática e dinâmica para sistemas Java. Neste sentido, foram adotadas algumas ferramentas utilizadas em [Vasconcelos 2007], como o Tracer para análise dinâmica e a Ares para análise estática.

A ferramenta Ares [Veronese e Netto 2001] realiza a leitura e interpretação de código fonte Java para recuperação do modelo de classes e pacotes de um sistema com base no metamodelo da UML. Durante este trabalho, foi necessário estender a ferramenta Ares para ampliar a sua capacidade de inferência. Para tanto, estudou-se o Ares e seu *parser,* a fim de acrescentar as seguintes capacidades: identificação de Invocação, Atribuição, Comentário, Iteração, entre outras. As Iterações permitem observar o comportamento de notificação dos observadores pelo objeto observado e as Invocações permitem inferir sobre demais comportamentos dos objetos do padrão. Ares implementa uma estrutura de fábricas, notadamente as de Arquivo, Classificadores e Métodos. A fábrica de arquivo, seguindo o padrão Java, cria um objeto arquivo que pertence a um Pacote e possui uma área de importações e classificadores. Estes Classificadores podem ser Classes, Interfaces ou Enumerações. A fábrica de Classificadores é responsável hierarquicamente pelas Interfaces implementadas, Atributos e Métodos. A Fábrica de Métodos controla a criação de métodos. A fim de unificar as informações provindas da análise estática e dinâmica, uma Fábrica de Rastros foi incorporada ao modelo original da ferramenta Ares, lendo o resultado da análise dinâmica e gerando um rastro de execução que liga um método chamador a um método chamado, sendo possível se determinar a classe a qual o método pertence.

A ferramenta Tracer [Vasconcelos 2007] contempla o monitoramento das chamadas de métodos que são realizadas durante a execução do programa alvo. O

resultado é persistido na forma de um arquivo XML. Sua estrutura é composta de uma hierarquia de chamadas de métodos, seus parâmetros e o tempo da execução. Este arquivo é analisado e seus elementos alimentam uma fábrica de rastros que são utilizados na detecção de regras baseadas na análise dinâmica. Quanto maior o número de caminhos executados na aplicação durante o monitoramento, mais preciso tende a ser o resultado da análise dinâmica. As regras são implementadas recuperando dados das fábricas e navegando sobre as listas de seus objetos. A API de serviços das fábricas permite obter todas as informações necessárias para execução das regras. Cabe esclarecer que as regras possuem uma forte interdependência, isto é, o resultado de uma regra pode impactar o resultado de outras, principalmente quando há interdependência entre a detecção dos elementos do padrão. Por exemplo, o Controlador faz referência à Visão, e vice-versa. A solução é manter a re-execução das regras enquanto houver alterações no estado de solução das mesmas, ou até que se atinja um número considerável de execuções. Para obtermos uma avaliação inicial da abordagem, foram conduzidos dois estudos de caso, que são descritos a seguir.

3.1. Estudos de Caso

Primeiro foi feito um estudo de caso sobre um sistema exemplo MVC especificamente desenvolvido para testes das regras. Este sistema, de pequeno porte, possui 8 classes e implementa as características deste padrão com aderência máxima. A Figura 1, a seguir, apresenta uma tela da ferramenta com os resultados aferidos para a classe Modelo da arquitetura MVC. Como pode ser observado, ao selecionar a classe TesteModel, a ferramenta apresenta os resultados das regras voltados para a classe. A classe TesteModel é 100% aderente às especificações do elemento Modelo no padrão MVC. Ela satisfaz também às regras de outros elementos, como, por exemplo, não implementar um escutador de eventos (*listener*). No entanto, estes casos representam regras coincidentes entre diferentes tipos de elementos, que futuramente serão ressaltadas na ferramenta. O percentual de aderência a cada elemento do padrão é apresentado no final da tela.

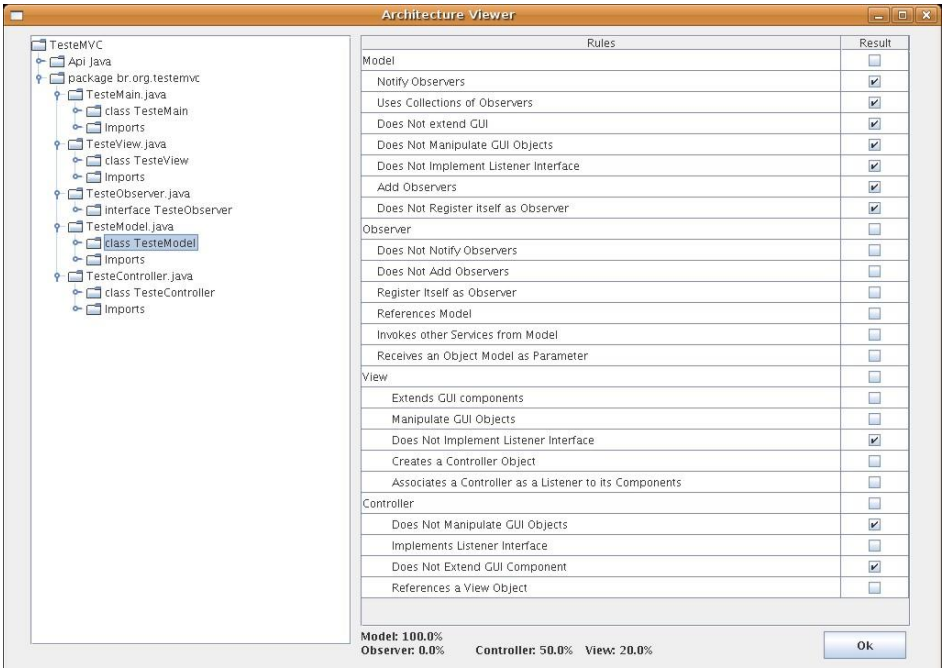

Figura 1. Sistema de Teste das Regras para a Detecção do MVC

Sistemas reais não costumam seguir precisamente um padrão arquitetural. Como mostra a Figura 2, que representa a análise da própria ferramenta Ares, um sistema de porte médio, com aproximadamente 34 classes, que segue a arquitetura MVC, a classe ControleEngenhariaReversa, dita Controladora, se comporta como tal, com as regras voltadas para os elementos *Observer* e Controlador com índices mais altos, embora satisfazendo algumas das demais regras. Outras classes do Ares apresentam resultados mais confusos, como GerenteAres, que apresenta 42% de características de Modelo, 33,3% Observer, 40 % View e 25% Controller.

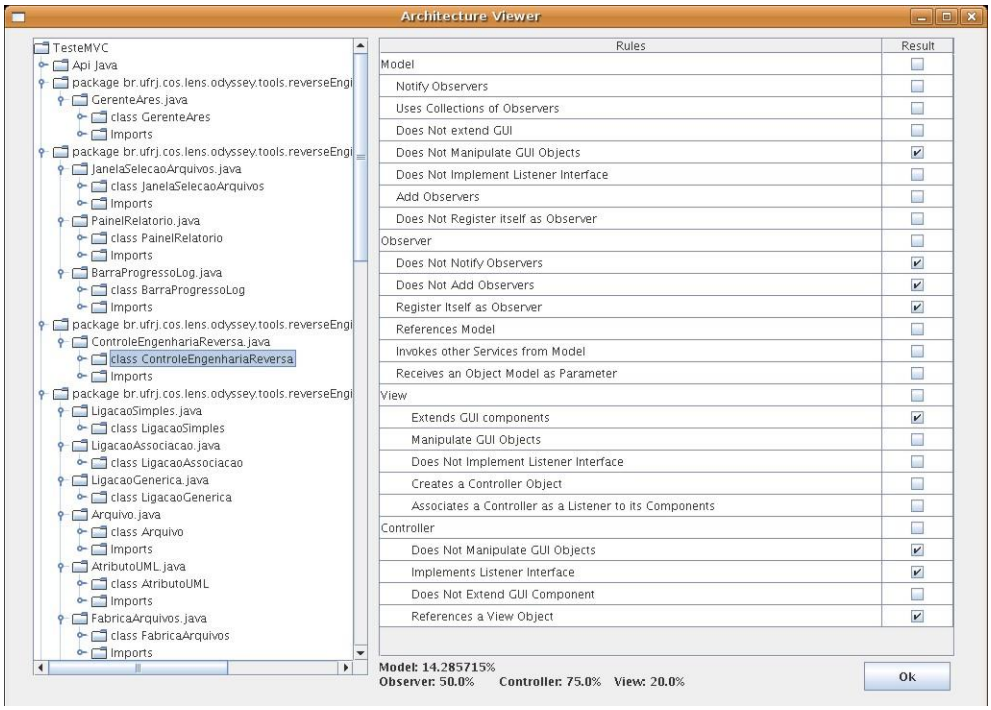

Figura 2. Detecção do MVC para a Classe ControleEngenhariaReversa do Ares

4. Conclusões

Como contribuições, ArchJava permite não só a identificação dos elementos de um padrão, mas também auxilia na identificação dos pontos onde as classes infringem a definição do padrão, através dos percentuais de proximidade a um tipo de elemento do padrão calculados para as classes. Inicialmente, este trabalho propõe regras para o reconhecimento do padrão MVC [Buschmann 1996], embora a abordagem proposta possa ser estendida para a detecção de outros padrões arquiteturais, como o padrão Camadas (*layers*), através da ampliação do conjunto de regras e da sua implementação na ferramenta de suporte a ArchJava. Para detectar novos padrões, as fábricas podem ser estendidas com novos serviços, seguindo uma evolução do metamodelo, quando necessário, para capturarem novos dados provenientes das análises estática e dinâmica. Dessa forma, seria possível atender à novas regras de padrões que necessitem de dados que as fábricas não possuam até então.

Como trabalhos futuros, ArchJava deverá ser avaliada através de estudos de caso reais, sendo utilizada por outros desenvolvedores, com o intuito de melhor avaliar e aperfeiçoar a detecção das regras. Esses sistemas reais devem possuir características de sistemas legados, conforme explicado no início do artigo, a fim de verificar a eficiência

de ArchJava neste contexto. Os constantes estudos de caso permitirão adicionar novas regras ao conjunto pré-existente. ArchJava deve ser aplicada também sobre sistemas que não sigam um determinado padrão, a fim de verificar o seu comportamento neste caso. Além disso, devem ser ressaltadas na ferramenta regras coincidentes entre diferentes elementos de um padrão. Também está sendo estudado um critério de atribuição de pesos para as regras, baseados no caráter de importância e capacidade que cada regra tem de tornar o seu classificador mais próximo de um determinado elemento arquitetural. Finalmente, a detecção de novos padrões deve ser incorporada à ArchJava.

5. Referências Bibliográficas

Alexander Yeh, David Harris e M. Chase (1997) "Manipulating Recovered Software Architecture Views". In Proc. of International Conference on Software Engineering, Boston, MA, USA, Maio, pp. 184-194.

Alexandre Correa, Claudia Werner, Gerson Zaverucha (2000) "Object Oriented Design Expertise Reuse: an Approach based on Heuristics, Design Patterns and Anti-Patterns", In: Sixth International Conference on Software Reuse, Viena, Austria, Junho, pp.336-352.

Aline Vasconcelos (2007) Uma Abordagem de Apoio à Criação de Arquiteturas de Referência de Domínio Baseada na Análise de Sistemas Legados. Tese de D.Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

A.V. Deursen., C. Hofmeister, R., Koschke, L. Moonen e C. Riva (2004) “Symphony: View-Driven Software Architecture Reconstruction”, Fourth Working IEEE/IFIP Conference on Software Architecture, Oslo, Norway, Junho, pp. 122-132.

B. Schmerl, J. Aldrich, J., D. Garlan, R. Kazman e H. Yan (2006) "Discovering Architectures from Running Systems". In IEEE Transactions on Software Engineering, vol 32, no. 7, pp. 454-466.

Claudio Riva e J.V. Rodriguez (2002) “Combining Static and Dynamic Views for Architecture Reconstruction”, Sixth European Conference on Software Maintenance and Reengineering (CSMR’027), Budapeste, Hungria, Março, pp. 47-56.

Dirk Heuzeroth, Thomas Holl, Gustav Högström e Welf Löwe (2003) "Automatic Design Pattern Detection", 11th IEEE Internatinal Workshop on Program Comprehension, Portland, Oregon, USA, Maio, pp. 94-103.

Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad e Michael Stal. (1996) "Pattern-Oriented Software Architecture: A System of Patterns", 1st ed., John Wiley & Sons.

R. Kazman e S.J. Carrière (1997) “Playing Detective: Reconstructing Software Architecture from Available Evidence”, Relatório Técnico CMU/SEI-97-TR-010, Carnegie Mellon University, Outubro.

R. S. Pressman (2005) Software Engineering: A Practitioner's Approach, 6a ed. McGraw-Hill. 2005.

Veronese, G.O., Netto, F.J. (2001) “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, Brasil.