Arquitetura de Aplicações Spring MVC: Uma Análise Baseada no Acoplamento Lógico

Eduardo P. de Sousa[1], Eduardo M. Guerra[1], Gustavo A. Oliva[2], Maurício F. Aniche[3]

[1] Instituto Nacional de Pesquisas Espaciais (INPE)

[2]Queen's University at Kingston

[3]Delft University of Technology

Abstract. *Frameworks are fundamental pieces in the development of Web applications, providing them with a set of features and a reference architecture. However, the multilingual nature of those applications makes it difficult to investigate the relationships among their components using structural coupling techniques. To overcome this difficulty, this paper proposes a study based on logical coupling techniques to investigate the relationships between architectural components of those applications. We applied this technique to 114 projects that use the Spring MVC framework in order to understand the sensitivity of the components to changes in other architectural layers, evidencing coupling points that could hinder the evolution of those applications. As a result, we found a strong coupling between Web pages and Controllers, as well as an asymmetry in the coupling between Controllers and Repositories, where Controllers are more prone to change due to changes in Repositories than the other way around.*

Resumo. *Frameworks são peças fundamentais no desenvolvimento de aplicações Web, provendo a elas um conjunto de funcionalidades e uma arquitetura de referência. No entanto, a natureza multi-linguagem dessas aplicações dificulta a investigação das relações entre seus componentes utilizando conceitos de acoplamento estrutural. Para superar essa dificuldade, este trabalho propõe um estudo baseado em técnicas de acoplamento lógico para investigar a relação entre os componentes arquiteturais dessas aplicações. Essa técnica foi aplicada a 114 projetos que utilizam o framework Spring MVC com intuito de investigar a sensibilidade dos componentes à mudanças em outras camadas arquiteturais, evidenciando pontos de acoplamento que possam dificultar a evolução das aplicações. Como resultado, verificou-se um forte acoplamento entre páginas Web e Controllers, assim como uma assimetria no acoplamento entre Controllers e Repositories, onde Controllers estão muito mais sujeitos às mudanças em Repositories do que o inverso.*

1. Introdução

O desenvolvimento e a utilização de *frameworks* para aplicações tem tido grande destaque nos anos recentes. Em algumas áreas do desenvolvimento de software, como no desenvolvimento de aplicações Web, o uso de *frameworks* é um padrão *de-facto* na indústria. Os *frameworks* têm por característica fundamental permitir o reúso de componentes e padrões bem estabelecidos [Johnson 1997]. Assim, eles se diferenciam das tradicionais bibliotecas de software por não apenas disponibilizar um conjunto de funcionalidades

ao desenvolvedor, mas também uma arquitetura e um conjunto de abstrações de alto nível [Fayad and Schmidt 1997] que servem de base para o processo de desenvolvimento.

Apesar da grande importância dos *frameworks* para a indústria de desenvolvimento Web, poucos estudos dedicam-se a explorar o impacto destes *frameworks* sobre a arquitetura das aplicações que os utilizam. Investigar a implementação dessas arquiteturas em aplicações desenvolvidas com *frameworks*, permitiria compreender quais os componentes ou camadas dessas aplicações estão mais sujeitos a altos níveis de acoplamento, o que pode tornar a evolução da aplicação mais custosa e sujeitá-la a uma obsolescência precoce [Stevens et al. 1974, Oman and Hagemeister 1992].

A investigação da arquitetura de aplicações Web, no entanto, envolve alguns desafios peculiares. Essas aplicações são naturalmente desenvolvidas com um amplo conjunto de linguagens, cada qual específica a um domínio da aplicação. Em uma única aplicação Web é possível encontrar desde linguagens de programação como *Java* e *JavaScript*, até linguagens de marcação como *XML* ou *HTML*, entre outras. Essa pluralidade de linguagens dificulta o uso de técnicas de investigação convencionais, como a análise estática de código-fonte, uma vez que as relações entre artefatos de diferentes linguagens nem sempre podem ser facilmente estabelecidas [Kontogiannis et al. 2006].

A investigação do acoplamento lógico por meio da mineração de repositórios de software é uma técnica que permite estabelecer relacionamentos entre artefatos de software com base no histórico de modificações destes artefatos [Oliva and Gerosa 2015]. Essa técnica revela a existência de relacionamentos entre artefatos que são frequentemente alterados em conjunto, sem que seja necessário conhecer o seu formato ou conteúdo [Gall et al. 1998]. A dissociação do acoplamento lógico em relação ao conteúdo torna-o bastante adequado para investigação do acoplamento em aplicações que envolvem diversas linguagens e formatos de arquivo, como as aplicações Web.

Neste trabalho, utilizamos a mineração de repositórios de software para investigar a arquitetura de 114 aplicações Web desenvolvidas com o *framework Spring MVC*. Este *framework* é voltado para o desenvolvimento de aplicações Web na linguagem *Java* e preconiza o uso do modelo de camadas MVC. A análise de um grande número de projetos desenvolvidos com este *framework* permite visualizar a existência de padrões arquiteturais entre essa aplicações, a fim de compreender quais componentes são mais sensíveis à mudança e a níveis mais altos de acoplamento.

2. Contexto

O estudo apresentado neste trabalho baseou-se em técnicas de mineração de repositórios de software para a investigação do acoplamento lógico entre componentes de diferentes camadas arquiteturais em aplicações desenvolvidas com o *framework Spring MVC*. Nesta seção, apresentamos estes componentes e a técnica utilizada para a investigação do seu acoplamento.

2.1. Componentes Arquiteturais

O *framework Spring MVC* define uma série de componentes que podem ser estendidos pelas aplicações, esses componentes tem papel fundamental na arquitetura de cada aplicação pois representam conceitos pertencentes a diferentes camadas, como interface com o usuário, tratamento de requisições, regras de negócio e persistência de dados.

Em uma aplicação Web do *framework Spring MVC*, a camada *View* do modelo MVC é geralmente implementada por meio da linguagem HTML, ou com uso de linguagens que são interpretadas de maneira dinâmica para a geração de código HTML, como JSP (*Java Server Pages*). Já os componentes das camadas *Model* e *Controller* são implementados na linguagem Java.

Na camada *Controller*, o *framework Spring MVC* define um único componente a ser estendido pelas aplicações, o próprio *Controller*, que é responsável por tratar as requisições recebidas pela aplicação. Já na camada *Model* são definidos os componentes *Service*, *Repository*, *Entity* e *Component*, responsáveis pela persistência dos dados da aplicação e implementação das regras de negócio.

Apesar dos componentes definidos pelo *Spring MVC*, o desenvolvedor possui flexibilidade na definição de sua arquitetura, uma vez que somente o componente *Controller* deve ser obrigatoriamente estendido pela aplicação.

2.2. Acoplamento Lógico

As técnicas convencionais para investigação do acoplamento entre artefatos de software são geralmente baseadas na análise estática do código-fonte em busca de referências a outros artefatos. Essas técnicas, no entanto, são incapazes de detectar alguns tipos de acoplamento, como o acoplamento dinâmico estabelecido em tempo de execução, ou o acoplamento entre código-fonte e documentação [Gall et al. 1998].

As técnicas de investigação do acoplamento lógico buscam superar essas dificuldades, utilizando como premissa o fato de que artefatos relacionados são geralmente modificados em conjunto [Yau et al. 1978]. Assim, usando como base o histórico de modificações dos artefatos, pode-se utilizar técnicas de mineração de repositórios de software em busca de associações entre esses artefatos [Zimmermann et al. 2005].

O método para identificação do acoplamento lógico baseia-se na extração do histórico de alterações dos artefatos de software, geralmente realizada a partir de sistemas de controle de versão. Essas alterações são agrupadas temporalmente, e então um algoritmo de busca por regras de associação, como Apriori [Agrawal et al. 1993], é utilizado para estabelecer um conjunto de regras na forma $X_1 \Rightarrow X_2$, onde X_1 e X_2 são cada qual um subconjunto exclusivo do conjunto de artefatos do software [Ying et al. 2004].

Uma regra $X_1 \Rightarrow X_2$ denota a existência de um ou mais conjuntos de alterações contendo simultaneamente os artefatos de X_1 e X_2, onde X_1 é chamado de antecedente e X_2 é chamado de consequente. Assim, uma regra de associação no contexto do acoplamento lógico expressa o quão frequente são alterações simultâneas em X_1 e X_2 por meio de um valor absoluto (número de alterações) ou relativo (razão do número de alterações simultâneas sobre o total de alterações), essa medida de frequência é chamada de *suporte*. Além do suporte, as regras de associação possuem um valor de *confiança*, que expressa o número de alterações no consequente dado que o antecedente foi alterado. Esse valor é expresso pela razão do número de alterações simultâneas no antecedente e consequente sobre o total de alterações no antecedente.

As métricas de suporte e confiança são utilizadas na investigação do acoplamento lógico para estabelecer, respectivamente, a frequência de uma determinada associação e a força da associação entre o antecedente e o consequente [Zimmermann et al. 2005]. Uma

regra de associação com valores de suporte e confiança suficientemente altos denotam que a ocorrência de X_1 implica com frequência na ocorrência de X_2, dessa forma, no contexto do acoplamento lógico, essa regra de associação denotaria que quando X_1 é alterado, X_2 tende a ser alterado também.

3. Análise das Aplicações Desenvolvidas com Spring MVC

O nosso estudo da arquitetura dos projetos desenvolvidos com o *framework Spring MVC* busca identificar relações entre os componentes das camadas arquiteturais da aplicação, reconhecendo possíveis padrões na implementação da arquitetura dessas aplicações. Com o intuito de identificar e explorar esses padrões arquiteturais estabeleceu-se algumas questões de pesquisa:

(QP1) Quais componentes arquiteturais do framework Spring MVC são mais sensíveis a mudanças realizadas nas páginas da aplicação?

(QP2) O acoplamento entre páginas e componentes é uniforme entre os projetos que utilizam o framework Spring MVC?

(QP3) Como se dá o acoplamento entre componentes dos projetos que utilizam o framework Spring MVC?

Com a *QP1*, pretendemos quantificar o acoplamento entre as páginas Web, que são o principal componente da camada *View*, e os componentes das demais camadas da aplicação, identificando àqueles que possuam níveis mais altos de acoplamento.

O modelo *MVC* recomenda que *Views* possam depender de *Model*, mas jamais o contrário [Fowler 2002]. Dessa forma cria-se a possibilidade de que a camada *Model* seja acessada pela *View* com ou sem o intermédio da camada *Controller*. Enquanto a *QP1* apenas procura identificar o componente mais sensível, a *QP2* explora as relações entre *Views* e demais camadas em busca de padrões e possíveis *outliers*. A *QP3*, por sua vez, volta-se para as relações formadas entre os componentes da aplicação pertencentes às camadas *Model* e *Controller*, com intuito de explorar as relações formadas entre eles na busca por relações comuns aos projetos.

3.1. Metodologia

Foram analisados 114 projetos Java desenvolvidos com o *framework Spring MVC*. Esse conjunto de projetos foi anteriormente utilizado na literatura [Aniche et al. 2016b, Aniche et al. 2016a], tendo sido selecionado com base nas seguintes condições: *(i)* possuir um histórico de desenvolvimento com ao menos 500 *commits*, *(ii)* possuir ao menos 10 classes que implementem componentes arquiteturais definidos pelo *framework*. O conjunto de projetos original era composto de 120 projetos, no entanto, alguns deles não se encontram mais disponíveis, o que reduziu o conjunto inicial a 114 projetos.

Neste estudo, foram analisadas as regras de associação formadas entre os diferentes componentes arquiteturais prescritos pelo framework Spring MVC. Os componentes arquiteturais utilizados foram *Controller, Service, Entity, Repository* e *Component*, e além destes, as páginas HTML e JSP que compões a apresentação da aplicação. Para identificação dos componentes arquiteturais, utilizou-se como critério a presença das anotações relacionadas a cada componente [Aniche et al. 2016b, Aniche et al. 2016a]. Já na identificação de componentes da apresentação, foi utilizada a extensão de nome de arquivo

como critério, sendo consideradas as extensões htm, html, jsp e jspx. A tabela 1 exibe o número de projetos contendo cada uma das classes de componentes consideradas nessa análise.

Tabela 1. Ocorrência das classes de componentes nos projetos analisados.

Item	Projetos	%
@Controller	114	100.0
JSP	95	83.3
@Component	89	78.1
HTML	88	77.2
@Service	81	71.1
@Entity	65	57.0
@Repository	55	48.3
HTM	9	7.9
JSPX	5	4.4

O histórico de *commits* dos projetos foi pré-processado, tendo sido efetuadas as seguintes ações: *(i)* classificação dos arquivos Java segundo o componente arquitetural implementado; *(ii)* classificação dos demais arquivos segundo sua extensão de nome de arquivo; *(iii)* eliminação das duplicidades entre os itens de cada *commit*. Após o pré-processamento, um total de 117.551 *commits* foram utilizadas como entrada para o algoritmo *Apriori* [Borgelt 2003].

A Tabela 2 mostra a ocorrência dos itens de interesse entre os *commits* analisados. Cada item é identificado por sua extensão de nome de arquivo, seguido da anotação do componente arquitetural definido pelo *framework Spring MVC*. Os itens identificados como *java:other* são arquivos de código-fonte Java cujo papel arquitetural não pôde ser identificado. Dentre os componentes arquiteturais, destacam-se *controllers* e *services*, presentes em 17,6% e 14,5% das *commits*, respectivamente.

Tabela 2. Número de *commits* que contém itens de interesse do estudo.

Item	Commits	
java:other	65.851	56,0%
java:controller	20.699	17.6%
java:service	17.056	14,5%
jsp:other, jspx:other	16.182	13,8%
htm:other, html:other	9.356	8,0%
java:component	9.017	7,7%
java:repository	6.342	5,4%
java:entity	5.736	4,9%

4. Resultados

Nesta seção, são apresentados os resultados obtidos na investigação da arquitetura das aplicações por meio do acoplamento lógico.

4.1. QP1: Quais componentes arquiteturais do *framework Spring MVC* são mais sensíveis a mudanças realizadas nas páginas da aplicação?

Dentre os 114 projetos analisados, ocorre o acoplamento lógico entre páginas da Web e componentes do tipo *Controller* em 92% deles, enquanto *Components* e *Services* estão acoplados às páginas em aproximadamente 60% dos projetos.

Enquanto as páginas Web estão presentes em média em 21,7% das alterações realizadas nos projetos, a frequência do acoplamento (suporte) que envolve páginas Web e *Controllers* é de aproximadamente 8,2%, com desvio padrão de 12,1%, enquanto o segundo componente mais frequentemente acoplado às páginas Web são os *Services*, com uma média de 5,5% e desvio padrão de 9,4%.

Já a força do acoplamento (confiança) entre páginas Web e *Controllers* é de em média 37%, com desvio padrão de 24,8%, enquanto o acoplamento com *Services* apresenta uma média de 21,4% e desvio padrão de 20,6%, os demais componentes possuem confiança média inferior a 15%. No sentido inverso, ou seja, quando as páginas Web fazem parte do consequente, os valores médios de confiança variam entre 22% e 29%, sugerindo que as alterações nos componentes acarretam de forma similar em alterações nas páginas Web.

Estes resultados demonstram que os *Controllers* apresentam maior sensibilidade às mudanças em páginas da aplicação, estando mais sujeitos a alterações decorrentes de mudanças nas páginas do que os demais componentes.

4.2. QP2: O acoplamento entre páginas e componentes é uniforme entre os projetos que utilizam o *framework Spring MVC*?

A Tabela 3 exibe a distribuição em quartis do suporte e confiança do acoplamento entre páginas Web e os outros componentes arquiteturais. A partir dos dados apresentados pode-se verificar uma distribuição compacta do suporte e confiança entre os componentes, com intervalos pouco amplos entre o primeiro e terceiro quartil. Apenas os *Controllers* se sobressaem por seus valores de suporte e confiança distribuídos em um intervalo mais amplo e com valores mais altos do que os demais, corroborando sua sensibilidade à mudanças nas páginas Web..

Apesar dos resultados mostrarem distribuições compactas para suporte e confiança, o que permite supor um acoplamento uniforme entre a maioria dos projetos, é importante salientar o grande intervalo entre o terceiro quartil e o valor máximo do suporte e confiança, sugerindo a existência de diversos projetos que fogem a essa distribuição uniforme.

4.3. QP3: Como se dá o acoplamento entre componentes dos projetos que utilizam o *framework Spring MVC*?

A associação mais frequente entre componentes arquiteturais é formada por *Controllers* e *Services*, com um suporte médio de 9,5% e desvio padrão de 10,6%. Alterações em componentes do tipo *Service* também estão frequentemente associadas com alterações em *Repositories*, com 7,25% de suporte médio, e desvio padrão de 7,4%.

Os valores de confiança indicam que em média 56,6%, com desvio padrão de 26,3%, das mudanças em *Repositories* acarretam alterações em *Services*. A associação

Tabela 3. Associação entre páginas web e componentes arquiteturais.

		Suporte				Confiança			
	Consequente	Q1	Q2	Q3	Max	Q1	Q2	Q3	Max
1	java:component	0,3	0,9	2,9	27,9	2,6	8,0	17,8	100,0
2	java:controller	0,4	2,9	11,3	66,1	16,2	38,5	51,0	100,0
3	java:entity	0,3	0,9	3,0	21,4	4,3	8,7	18,1	100,0
4	java:repository	0,3	1,7	4,2	31,4	4,5	10,4	21,1	58,9
5	java:service	0,5	1,6	5,4	55,4	5,5	16,1	31,4	100,0

inversa, no entanto, tem um valor de confiança de 31,5%, e desvio padrão de 23,0%. Alterações nos *Repositories* também acarretam em alterações nos *Controllers* em 48,2% do tempo, com desvio padrão de 26,5%, enquanto o inverso ocorre em apenas 19,1% dos casos, com desvio padrão de 16,7%.

Estes resultados demonstram a aplicação dos preceitos de desacoplamento entre *Model* e *View* do modelo MVC, com o uso da camada *Controller* como intermediária entre essas camadas. Enquanto componentes do tipo *Controller* mostram-se sensíveis à mudanças ocorridas em páginas Web e em componentes do tipo *Repository*, a associação inversa não sobressai aos demais componentes, ou seja, mudanças nos *Controllers* não afetam páginas Web ou componentes do tipo *Repository* com mais frequência do que os demais componentes.

5. Conclusão e Trabalhos Futuros

O estudo apresentado, demonstra que um amplo conjunto de projetos compartilha características comuns em termos de sua arquitetura. A arquitetura de camadas prescrita pelo *framework Spring MVC* é seguida com poucas variações por grande parte dos projetos, no entanto, um número considerável de *outliers* sugere que uma investigação direta destes projetos possa revelar variações significativas do modelo MVC. Uma extensão do estudo apresentado poderá incluir novas dimensões para a análise dos projetos, como o número de arquivos, de componentes arquiteturais ou mesmo seu tempo de vida. O que permitiria uma melhor caracterização destes *outliers*.

Enquanto os componentes arquiteturais do *framework Spring MVC* foram facilmente identificados por meio de suas anotações, essa abordagem não permitiu a identificação de um grande número de classes da aplicação, pois elas não utilizam as anotações do *framework*. A criação de uma heurística mais abrangente para a identificação de componentes arquiteturais permitirá um entendimento mais completo das aplicações, ou mesmo a extensão desta análise a outros *frameworks* e arquiteturas.

Agradecimentos: Este trabalho tem apoio do CNPq (445562/2014-5) e da FAPESP (2015/16487-1).

Referências

Agrawal, R., Imieliński, T., and Swami, A. (1993). Mining association rules between sets of items in large databases. In *Acm sigmod record*, volume 22, pages 207–216. ACM.

Aniche, M., Bavota, G., Treude, C., Van Deursen, A., and Gerosa, M. A. (2016a). A validated set of smells in model-view-controller architectures. In *Software Mainte-*

nance and Evolution (ICSME), 2016 IEEE International Conference on, pages 233–243. IEEE.

Aniche, M., Treude, C., Zaidman, A., van Deursen, A., and Gerosa, M. A. (2016b). Satt: Tailoring code metric thresholds for different software architectures. In *Source Code Analysis and Manipulation (SCAM), 2016 IEEE 16th International Working Conference on*, pages 41–50. IEEE.

Borgelt, C. (2003). Efficient implementations of apriori and eclat. In *FIMI'03: Proceedings of the IEEE ICDM workshop on frequent itemset mining implementations*.

Fayad, M. and Schmidt, D. C. (1997). Object-oriented application frameworks. *Communications of the ACM*, 40(10):32–38.

Fowler, M. (2002). *Patterns of enterprise application architecture*. Addison-Wesley Longman Publishing Co., Inc.

Gall, H., Hajek, K., and Jazayeri, M. (1998). Detection of logical coupling based on product release history. In *Software Maintenance, 1998. Proceedings., International Conference on*, pages 190–198. IEEE.

Johnson, R. E. (1997). Frameworks=(components+ patterns). *Communications of the ACM*, 40(10):39–42.

Kontogiannis, K., Linos, P., and Wong, K. (2006). Comprehension and maintenance of large-scale multi-language software applications. In *Software Maintenance, 2006. ICSM'06. 22nd IEEE International Conference on*, pages 497–500. IEEE.

Oliva, G. A. and Gerosa, M. A. (2015). Change coupling between software artifacts: Learning from past changes. In Christian Bird, T. M. and Zimmermann, T., editors, *The Art and Science of Analyzing Software Data*, chapter 11, pages 286–319. Morgan Kaufmann.

Oman, P. and Hagemeister, J. (1992). Metrics for assessing a software system's maintainability. In *Software Maintenance, 1992. Proceerdings., Conference on*, pages 337–344. IEEE.

Stevens, W. P., Myers, G. J., and Constantine, L. L. (1974). Structured design. *IBM Systems Journal*, 13(2):115–139.

Yau, S. S., Collofello, J. S., and MacGregor, T. (1978). Ripple effect analysis of software maintenance. In *Computer Software and Applications Conference, 1978. COMPSAC'78. The IEEE Computer Society's Second International*, pages 60–65. IEEE.

Ying, A. T., Murphy, G. C., Ng, R., and Chu-Carroll, M. C. (2004). Predicting source code changes by mining change history. *IEEE transactions on Software Engineering*, 30(9):574–586.

Zimmermann, T., Zeller, A., Weissgerber, P., and Diehl, S. (2005). Mining version histories to guide software changes. *IEEE Transactions on Software Engineering*, 31(6):429–445.