Verificação de aderência entre arquiteturas conceituais e emergentes utilizando a abordagem PREViA

Marcelo Schots[1], Marlon Silva[1], Leonardo Murta[2], Cláudia Werner[1]

[1]Programa de Engenharia de Sistemas e Computação – COPPE/UFRJ
Caixa Postal 68.511, Rio de Janeiro, RJ, 21945-970

{schots, marlon, werner}@cos.ufrj.br

[2]Instituto de Computação – Universidade Federal Fluminense
Rua Passo da Pátria 156, Niterói, RJ, 24210-240

leomurta@ic.uff.br

Abstract. *Software architectures evolve over time due to several reasons. It is important to keep the consistency between the developed applications (represented by emerging architectures) and their conceptual architecture anytime in the software development lifecycle. The PREViA approach (Paradigm for Representing the Evolution through Visualization of Architectures) aims at improving perception and comprehension of the evolution of software architecture models through a method for visualizing such evolution; this visualization represents the result of the comparison between versions of these models under different comparison perspectives and view focus.*

Keywords. *Software Evolution, Software Architecture, Software Visualization, Traceability*

Resumo. *Arquiteturas de software evoluem no decorrer do tempo devido a diversas razões. É importante que a consistência entre as aplicações desenvolvidas (representadas por arquiteturas emergentes) e sua arquitetura conceitual seja mantida em qualquer instante do tempo no ciclo de vida do desenvolvimento do software. A abordagem PREViA (Procedimento para Representar a Evolução por meio da Visualização de Arquiteturas) visa facilitar a percepção e a compreensão da evolução de modelos arquiteturais de software a partir de um método de visualização desta evolução; esta visualização representa o resultado da comparação entre versões destes modelos sob diferentes perspectivas de comparação e focos de visualização.*

Palavras-chave. *Evolução de Software, Arquitetura de Software, Visualização de Software, Rastreabilidade*

1. Introdução

A arquitetura de um software envolve a descrição dos elementos a partir dos quais os sistemas são construídos, as interações entre esses elementos, padrões que direcionam sua composição e restrições sobre estes padrões [Shaw & Garlan 1996]. As diferentes representações de um sistema, cada uma sob uma diferente perspectiva, são denominadas visões arquiteturais. Existem diversas abordagens para definir, descrever e identificar estas visões (e.g. [Kruchten 1995] e [Razavizadeh *et al.* 2009]), dentre elas a

notação UML (*Unified Modeling Language*), voltada para sistemas orientados a objetos. Por meio desta notação, é possível descrever o projeto arquitetural em diferentes níveis de detalhamento. Por se tratar de um modelo extensível, é possível incluir elementos adicionais aos modelos por meio da criação de perfis e estereótipos.

A arquitetura é comumente modificada no decorrer do tempo, devido a mudanças que ocorrem no ambiente e demandas por inclusão ou alteração dos requisitos do software. Adicionalmente, os programadores podem vir a seguir caminhos distintos dos que foram considerados pelos projetistas de software. Situações como estas fazem com que a arquitetura conceitual, ou seja, a arquitetura planejada na fase de projeto do software, comece a divergir da arquitetura emergente, que é a arquitetura efetivamente implementada em código fonte – obtida, por exemplo, por engenharia reversa. Essa divergência pode representar, por exemplo, elementos da arquitetura conceitual que ainda não foram implementados, ou mesmo elementos implementados – presentes na arquitetura emergente – que não foram descritos/planejados na arquitetura conceitual.

É importante manter a consistência entre estas arquiteturas em qualquer instante do tempo [Murphy *et al.* 2001; Völter 2007]. Neste sentido, a verificação da aderência entre as arquiteturas pode evitar posteriores impactos negativos e auxiliar no processo de tomada de decisões em nível de projeto e implementação, especialmente no que tange a processos de manutenção e reengenharia. Nestes dois processos, visto que a arquitetura conceitual possui uma descrição em alto nível do sistema implementado, esta pode ser utilizada para reconstruir o software (em outra linguagem de programação, por exemplo) ou mesmo para compreender melhor o software passível de manutenção.

As diferenças arquiteturais obtidas por mecanismos de *diff* não são muito intuitivas (algumas vezes restringindo-se a representações textuais) e podem demandar muito tempo e esforço para que sejam compreendidas [Lintern *et al.* 2003]. Para contornar este problema, técnicas de visualização de software podem fornecer uma melhor representação dessas informações. Tais técnicas proporcionam uma forma mais simples e intuitiva de compreensão dos dados representados [Ball & Eick 1996], tornando "visíveis" os artefatos e o comportamento do software por meio de representações visuais. Segundo Mukherjea & Foley (1995), a visualização destas informações é particularmente importante por permitir que as pessoas utilizem o raciocínio perceptivo (em vez do raciocínio cognitivo) na realização das tarefas, o que facilita a detecção de padrões que revelam a estrutura implícita nos dados.

Este artigo apresenta a abordagem PREViA, que permite a visualização da evolução de modelos arquiteturais de software, a partir de suas versões sob diferentes perspectivas de comparação. A partir do uso desta abordagem, é esperado que o engenheiro de software obtenha uma melhor compreensão e percepção da aderência da arquitetura emergente com relação à arquitetura conceitual.

O presente trabalho está organizado em outras cinco sessões. A abordagem PREViA é apresentada na Seção 2. Na Seção 3, um exemplo ilustra a aplicação da abordagem. A Seção 4 apresenta alguns trabalhos relacionados. A Seção 5 inclui considerações finais, algumas limitações e os próximos passos.

2. A abordagem PREViA

A abordagem PREViA (Procedimento para Representar a Evolução por meio da Visualização de Arquiteturas) fornece a visualização da evolução de modelos arquiteturais de

software a partir da comparação de versões da(s) arquitetura(s) armazenadas em fontes de dados versionados (e.g., repositórios de gerência de configuração). Para isto, são utilizados conceitos e técnicas de visualização de software a fim de facilitar a compreensão e a percepção desta evolução.

Os focos de visualização são a aderência da arquitetura emergente à arquitetura conceitual no decorrer do tempo e as diferenças encontradas entre duas versões distintas no processo de evolução arquitetural. Este trabalho é voltado para o primeiro foco.

A Figura 1 mostra uma simplificação do fluxo de atividades da abordagem PREViA. Para dar início ao processo, é necessário selecionar a fonte de dados, a partir da qual as arquiteturas emergentes serão extraídas. Em seguida, é feita uma comparação da arquitetura conceitual com cada versão da arquitetura emergente, de forma a detectar os elementos comuns a ambas e as diferenças existentes (i.e., possíveis divergências). Com base nestas diferenças, é gerado um modelo resultante que contém todos os elementos analisados de ambos os modelos, de forma que possam ser exibidos de acordo com o resultado das comparações efetuadas.

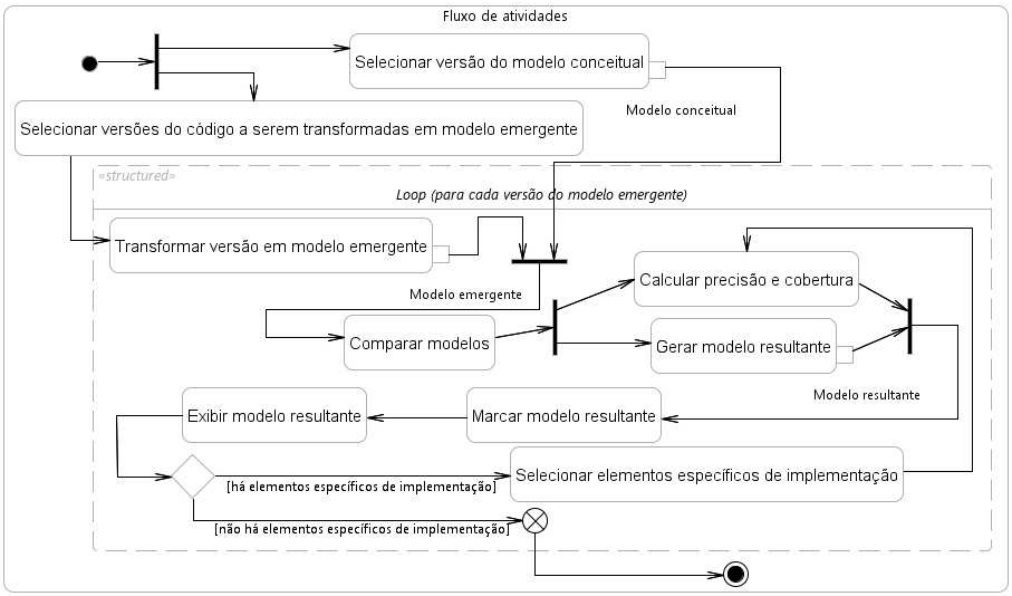

Figura 1. Fluxo de atividades da abordagem PREViA

É esperado que a arquitetura emergente – extraída do código fonte – contenha elementos que são específicos da implementação – e.g., classes provedoras de acesso e persistência dos dados da aplicação (ou *Data Access Objects*), ou elementos relacionados a tecnologias específicas. Parte-se, então, para uma identificação semi-automática destes elementos, por meio da automatização do processo de obtenção da similaridade entre os elementos de cada arquitetura, permitindo que um engenheiro de software inserido no contexto do desenvolvimento estabeleça manualmente associações entre elementos conceituais e emergentes manualmente (e.g., se elementos de implementação da arquitetura emergente forem incorretamente identificados como conceituais).

Com o intuito de quantificar a adequação entre as arquiteturas conceitual e emergente, são utilizados os conceitos de **precisão** (*precision*) e **cobertura** (*recall*). No contexto deste trabalho, estas medidas podem indicar, respectivamente, a exatidão e a completeza de um determinado instante do desenvolvimento (ou seja, uma dada versão

do software) com relação ao que foi planejado na arquitetura conceitual. Os cálculos são efetuados da seguinte forma:

$$\text{Precisão} = \frac{n_{ci}}{n_i - n_{ii}} \quad \text{(i)} \qquad\qquad \text{Cobertura} = \frac{n_{ci}}{n_c} \quad \text{(ii)}$$

Nas fórmulas (i) e (ii), n_{ci} é o número de elementos conceituais implementados, n_i é o número total de elementos implementados, n_{ii} é o número de elementos implementados identificados como específicos de implementação, que devem ser subtraídos de forma a não interferir no cálculo da precisão, e n_c é o número total de elementos conceituais. Essas medidas são aplicáveis a elementos em diferentes níveis de abstração, desde que o modelo utilizado para representar a arquitetura esteja hierarquicamente definido/configurado. Após o cálculo das métricas para cada elemento, o modelo resultante é marcado com os valores de precisão e cobertura obtidos para cada elemento.

Parte-se, então, para a etapa de visualização do modelo resultante. O objetivo desta visualização é transmitir a ideia de "sobreposição" da arquitetura emergente à arquitetura conceitual no decorrer do tempo. Desta forma, à medida que é identificado um grau de similaridade entre elementos das arquiteturas emergente e conceitual, esta última é "preenchida", realçando possíveis divergências encontradas com relação à primeira. Durante a visualização, é possível especificar manualmente os elementos que são específicos de implementação; como tais elementos impactam nos valores obtidos para a precisão, retorna-se ao cálculo desta métrica de forma a atualizar as marcações do modelo resultante. Uma vez que um elemento foi identificado desta forma, não é necessário identificá-lo novamente nas demais versões que o contenham.

2.1. Implementação

No que tange à implementação da abordagem PREViA, optou-se por estender a ferramenta EvolTrack [Cepeda *et al.* 2008], um mecanismo de extração e visualização do ciclo de evolução de projetos de software que possui uma arquitetura baseada em *plugins*, o que o torna extensível. Como prova de conceito, a implementação da PREViA utiliza a UML para representar as arquiteturas; para a marcação dos elementos (em termos de métricas obtidas e seleção de elementos específicos de implementação), são utilizados perfis UML específicos, aplicados à arquitetura. A Figura 2 mostra o mapeamento entre os módulos do mecanismo EvolTrack e da abordagem PREViA (nesta figura, os módulos são divididos horizontalmente e os diferentes conectores estão distribuídos verticalmente). O transformador de modelos, por exemplo, é um conector composto de três módulos que desempenham as atividades representadas na Figura 1.

Figura 2. Mapeamento entre módulos EvolTrack (acima) e PREViA (abaixo)

O conector de fonte de dados provido nativamente pelo EvolTrack é voltado para repositórios de controle de versão do Subversion. Isto motivou a construção de um novo conector que provê as funcionalidades do conector anterior e não é acoplado a um

sistema de controle de versões específico. Para isto, foi utilizada a API denominada Maven SCM, um software livre que provê mecanismos para o uso de ferramentas de gerência de configuração por meio de interfaces genéricas. O conector foi projetado de forma a ser extensível, isto é, existe a possibilidade de se estender esta interface para outros sistemas de controle de versão. A Figura 3 (a) mostra a tela de configuração do conector, incluindo uma tela de seleção das versões do código fonte a serem transformadas, via engenharia reversa, em modelos emergentes. Caso as diferentes versões de um mesmo artefato não estejam em um repositório (i.e., estejam armazenadas em uma pasta local), um segundo conector permite o acesso a estas versões.

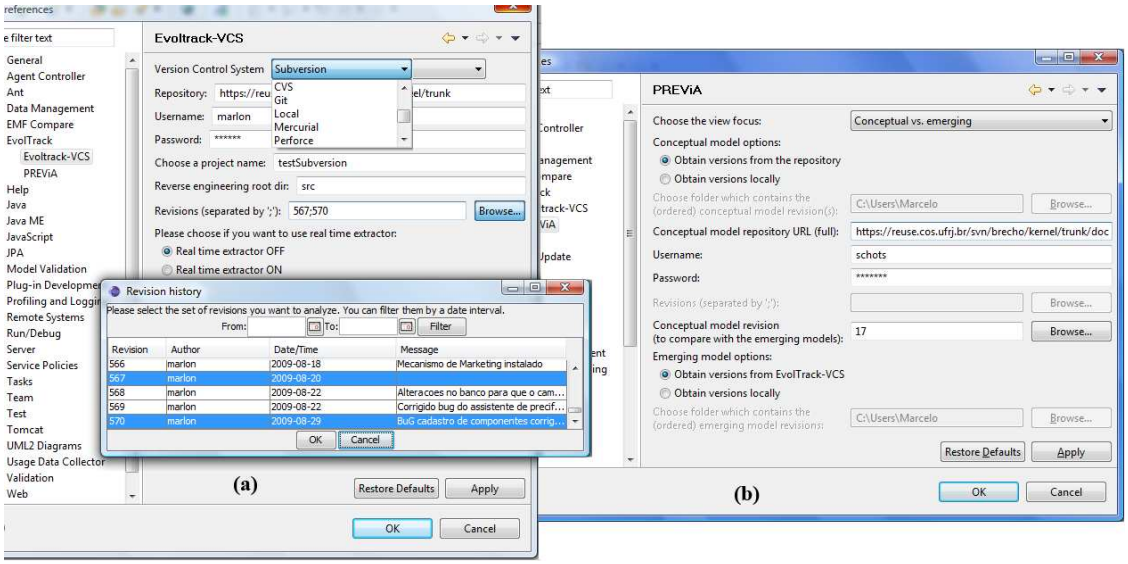

Figura 3. Configuração do conector e perspectiva de comparação/visualização

No que diz respeito à visualização, cada conector representa um foco de visualização selecionado. Para o foco de aderência entre as arquiteturas (representado neste trabalho), a arquitetura conceitual é representada como marca d'água, e cada elemento desta que tiver sido identificado na arquitetura emergente "preenche" esta marca d'água, como pode ser visto na seção a seguir. A Figura 3 (b) exibe a tela na qual os dados sobre as arquiteturas e o foco de visualização são preenchidos.

3. Exemplo de Utilização

Nesta seção, é apresentado um exemplo de aplicação da abordagem PREViA. O exemplo faz uso de um projeto fictício, denominado *HotelSystem*, adaptado de [Prudêncio 2008]. A equipe do projeto é formada por um engenheiro de software (responsável por gerenciar todo o processo de desenvolvimento), um arquiteto de software e três desenvolvedores. Toda a equipe utiliza um sistema de controle de versões para os artefatos gerados em suas atividades no processo.

Com base nos documentos de especificação dos requisitos, o arquiteto de software elabora um modelo conceitual (que será inspecionado por um engenheiro de software) e, em sendo aprovado, tal modelo é repassado para os desenvolvedores, que iniciam a etapa de implementação. Num determinado momento do desenvolvimento, o engenheiro de software deseja examinar o status do processo de desenvolvimento. Seguindo o fluxo de atividades da Figura 1, ele seleciona a versão mais recente do código fonte (a partir de uma fonte de dados versionados) e a arquitetura conceitual

(que, até o momento, não foi modificada). A partir desta versão do código fonte, é então extraída a arquitetura emergente.

Após a geração do modelo resultante, as diferenças entre as arquiteturas conceitual e emergente são marcadas e o cálculo das métricas de precisão e cobertura é efetuado e também marcado. O modelo resultante é exibido na Figura 4.

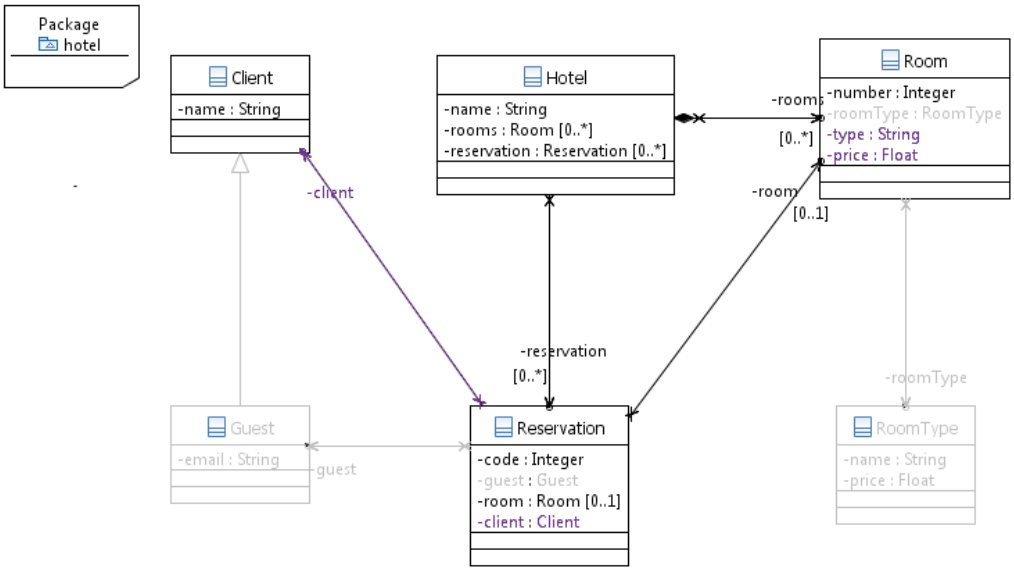

Figura 4. Sobreposição dos modelos conceitual e emergente

A partir das características desta visualização, é possível observar que a classe *Guest* não foi implementada (e foi criado um atributo *client*, do tipo *Client*, na classe *Reservation*). Além disso, os atributos da classe *RoomType* (que também não foi implementada) parecem ter sido implementados diretamente na classe *Room*. Percebe-se, nesta situação, que a arquitetura conceitual já não representa mais o que está sendo implementado no sistema. Cabe ao engenheiro de software decidir se (1) a arquitetura conceitual deverá ser atualizada com as informações modificadas pelos desenvolvedores, ou (2) a equipe de desenvolvimento deverá ser alertada para a detecção do desvio do planejamento inicial, corrigindo o erro nas próximas versões.

No exemplo, os valores de precisão e cobertura do pacote *hotel* são, respectivamente, 11 ÷ (14 – 0) = 0,786 e 11 ÷ 19 = 0,579 (supondo que não há nenhum elemento específico de implementação). Os valores destas métricas são propagados dos níveis mais baixos de abstração aos mais altos (e.g., o valor das métricas das classes é baseado em suas propriedades e operações). Quanto maior o valor da precisão, mais elementos implementados estão no modelo conceitual, mas isto não implica em mais elementos conceituais terem sido implementados, o que é retratado pela cobertura.

4. Trabalhos Relacionados

Lanza & Ducasse (2002) apresentam a visualização da evolução de software por meio de uma matriz denominada matriz de evolução, onde cada coluna representa uma versão do software, enquanto cada linha representa uma classe em suas diferentes versões no sistema. A partir de duas métricas extraídas do código fonte (número de métodos e número de atributos), é possível visualizar informações acerca do comportamento da evolução das classes. Algumas deficiências da abordagem são: (i) a utilização de um

vocabulário característico do domínio da astronomia para categorizar o comportamento das classes, e (ii) a pouca exploração de recursos e técnicas de visualização de software para a representação dos conceitos (e. g., diferentes formas e cores). Comparada à abordagem PREViA, as informações contidas na matriz de evolução se concentram no aspecto da refatoração, enquanto a PREViA é voltada principalmente para métricas de evolução de modelos (podendo contemplar também métricas de código fonte no momento da extração das arquiteturas emergentes).

O trabalho desenvolvido por Ohst *et al.* (2003) visa a detecção e visualização de diferenças existentes entre versões de diagramas UML. A abordagem proposta por esses autores para a visualização das diferenças entre dois diagramas se assemelha à abordagem PREViA, pois também é utilizado um único documento que contém elementos comuns a ambas as versões e elementos específicos de cada versão, sendo estes últimos destacados por uma determinada coloração. Os autores, porém, não trataram a comparação entre arquiteturas conceitual e emergente. Além disto, a abordagem não prevê a exibição da sequência temporal das versões com suas diferenças capturadas. Tal funcionalidade é provida por meio da integração entre o EvolTrack e a PREViA.

A ferramenta SAVE (*Static Architecture Visualization and Evaluation*) [Knodel *et al.* 2006] tem como propósito a avaliação e visualização de arquiteturas estáticas, com foco em verificações de conformidade entre os relacionamentos arquiteturais a partir da comparação efetuada entre a arquitetura conceitual e a emergente (sendo esta obtida através de engenharia reversa, utilizando-se de estratégias de transformação do código para modelo). Um aspecto não tratado pela ferramenta é a representação visual da variação da similaridade entre as arquiteturas, o que é possível com a abordagem PREViA graças às métricas de precisão e cobertura. SAVE também não permite a navegação por entre versões dos modelos obtidos e suas diferenças.

5. Considerações Finais

Possíveis divergências e inconsistências entre as aplicações desenvolvidas e sua arquitetura conceitual correspondente podem vir a causar diversos impactos negativos no processo de desenvolvimento. A abordagem PREViA, apresentada neste artigo, visa facilitar a percepção e a compreensão da evolução de arquiteturas de software e da aderência da arquitetura emergente à arquitetura conceitual, a partir de um método de visualização da evolução de modelos. Além da descrição da abordagem, um protótipo foi construído (utilizando a UML como prova de conceito) e um exemplo de utilização da PREViA também foram apresentados.

Algumas limitações da abordagem são: (i) sua utilização em projetos que seguem a metodologia MDE (*Model Driven Engineering*), uma vez que a abordagem não identifica se a implementação do modelo está completa e correta em termos funcionais, e (ii) a subjetividade da atividade "Selecionar elementos específicos de implementação", fortemente dependente do conhecimento do engenheiro de software.

A próxima etapa do trabalho é a avaliação da abordagem por meio de um estudo de observação, com o objetivo de verificar se o aspecto da visualização foi diferencial para a identificação de diferenças e divergências entre os modelos. Para isto, duas equipes deverão receber projetos com e sem a aplicação da abordagem PREViA. Serão considerados aspectos quantitativos (e.g., o tempo gasto na identificação) e qualitativos (por meio de questionário *follow-up* – e.g., facilidade de uso da abordagem).

Agradecimentos

Os autores agradecem à CAPES, ao CNPq e à FAPERJ pelo apoio financeiro para a realização deste trabalho.

Referências

Ball, T. and Eick, S. (1996). "Software Visualization in the Large", *IEEE Computer*, v. 29, n. 4 (April), pp. 33-43.

Cepeda, R. S. V., Murta, L. G. P., and Werner, C. M. L. (2008). "Visualizando a Evolução de Software no Desenvolvimento Distribuído". *INFOCOMP* (UFLA), v. S. Ed., pp. 81-90.

Knodel, J., Muthig, D., Naab, M. and Lindvall, M. (2006). "Static Evaluation of Software Architectures", In: *Proceedings of Conference on Software Maintenance and Reeng. (CSMR)*, pp. 279-294, IEEE Computer Society, Washington, DC, March.

Kruchten, P. (1995). "Architectural Blueprints - The 4+1 View Model of Architecture", *IEEE Software.* 12, 6, pp. 42-50, November.

Lanza, M. and Ducasse, S. (2002). "Understanding software evolution using a combination of software visualization and software metrics", In: *Proceedings of Langages et Modèles à Objets (LMO 2002)*, pp. 135-149, Paris, Lavoisier, August.

Lintern, R., Michaud, J., Storey, M., and Wu, X. (2003), "Plugging-in Visualization: Experiences Integrating a Visualization Tool with Eclipse". In: *ACM Symposium on Software Visualization (SoftVis)*, pp. 47-ff, San Diego, California, June.

Mukherjea, S. and Foley, J. D. (1995). "Requirements and Architecture of an Information Visualization Tool", In: *Proceedings of the IEEE Visualization '95 Workshop on Database Issues For Data Visualization*, pp. 57-75, Eds. Lecture Notes In Computer Science, vol. 1183, Springer-Verlag, London.

Murphy, G. C., Notkin, D., and Sullivan, K. (2001). "Software Reflexion Models: Bridging the Gap between Design and Implementation. *IEEE Trans. Software Eng.*, v. 27, n. 4, pp. 364-380, April.

Ohst, D., Welle, M., and Kelter, U. (2003). "Differences between versions of UML diagrams". *SIGSOFT Software Engineering Notes*, 28, 5, pp. 227-236, September.

Prudêncio, J. G. G. (2008). *Orion: Uma Abordagem para Seleção de Políticas de Controle de Concorrência*. Dissertação de M.Sc., COPPE/UFRJ, Rio de Janeiro, Brasil.

Razavizadeh, A., Verjus, H., Cimpan, S., and Ducasse, S. (2009). "Multiple Viewpoints Architecture Extraction", In: *Proceedings of the 2009 Joint Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architectures*, pp. 14-19, Cambridge, UK, September.

Shaw, M. and Garlan, D. (1996), *Software Architecture - Perspectives on an Emerging Discipline*, Prentice-Hall, Inc. Upper Saddle River, NJ, USA.

Völter, M. (2007). "Software Architecture Documentation in the Real World". Tutorial. In: *22nd Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2007)*, ACM, Montreal, Quebec, Canada, October.