Um Framework para Recuperação Arquitetural Independente de Plataforma

Luis Paulo T. de Oliveira e Sandro S. Andrade

Grupo de Pesquisa em Sistemas Distribuídos, Otimização, Redes e Tempo-Real (GSORT)
Instituto Federal de Educação, Ciência e Tecnologia da Bahia (IFBa)
Av. Araújo Pinho, nº 39 - Canela - Salvador - BA - CEP: 40.110-150

{luisoliveira, sandroandrade}@ifba.edu.br

Abstract. *Architecture recovery techniques constitute effective mechanisms for getting knowledge about the different artifacts which comprise a software system. Nevertheless, since a large number of recovery approaches exist nowadays, providing simple and flexible platforms to develop, evaluate, and compare different architecture recovery techniques is of paramount importance. This paper presents an architecture recovery framework aiming at achieving flexibility regarding: i) the development technologies adopted in the software whose architecture is expected to be recovered; ii) the used architecture recovery algorithm; and iii) the modeling notation which will describe the recovered models. We instantiate such a framework in order to recovery the architecture of systems developed in C++/Qt, using the ACDC recovery algorithm, and describing the recovered models in the UML modeling language.*

Resumo. *Técnicas para recuperação arquitetural representam mecanismos eficientes para obtenção de conhecimento sobre os artefatos que compõem um sistema. Entretanto, em meio a um grande número de abordagens existentes, a prospecção de plataformas simples e flexíveis para desenvolvimento, avaliação e comparação de diferentes técnicas de recuperação arquitetural se torna uma importante atividade. Este trabalho apresenta um framework para recuperação arquitetural caracterizado pela flexibilidade em relação a: i) plataforma de desenvolvimento utilizada no software cuja arquitetura será recuperada; ii) algoritmo de recuperação utilizado; e iii) notação de modelagem aplicada na representação da arquitetura recuperada. Um exemplo de instanciação do framework para recuperação arquitetural de sistemas desenvolvidos em C++/Qt, com utilização do algoritmo de recuperação ACDC e representação de arquiteturas em UML é apresentado ao final do artigo.*

1. Introdução

Compreender as decisões e táticas utilizadas no projeto da arquitetura de um *software* é fundamental para um melhor suporte às atividades de manutenção, evolução e configuração de aplicações [Taylor et al. 2009]. Decisões arquiteturais impactam diretamente o grau de atendimento dos requisitos não-funcionais desejados, a manutenção da integridade conceitual à medida em que a aplicação evolui e a adoção de estratégias de reuso tais como as linhas de produto. Portanto, recuperar arquiteturas de aplicações já existentes – para fins de documentação ou suporte às atividades subsequentes – é uma tarefa de fundamental importância.

Neste contexto, uma série de iniciativas na academia e na indústria têm como foco o desenvolvimento de métodos flexíveis e sofisticados para recuperação de arquiteturas a partir do conjunto disponível de artefatos de *software* [Maqbool and Babri 2007]. Tais iniciativas diferem entre si em relação aos elementos arquiteturais (componentes, conectores, interfaces e estilos) considerados na recuperação, público-alvo dos modelos recuperados, artefatos utilizados como entrada do processo de recuperação (código-fonte, *traces* de execução e modelos prévios), notação utilizada na descrição dos modelos recuperados, dentre outros aspectos.

Uma abordagem para recuperação arquitetural [Garcia et al. 2013a] define os aspectos arquiteturais a serem recuperados e as técnicas a serem utilizadas na obtenção destas informações. Aspectos comumente recuperados incluem: informações estruturais (módulos/componentes, *interfaces* e conectores); informações comportamentais (fluxos de dados e de controle, frequência de interações, etc); granularidade da representação (sistema-subsistema, módulos principais, dependência entre bibliotecas, etc); e visões arquiteturais (estrutural, concorrência, implantação, dados, etc). Os modelos recuperados são descritos em alguma notação de modelagem arquitetural, que varia desde linguagens informais – tais como textos semi-estruturados e gráficos *ad-hoc* – até aquelas semi-formais e formais tais como a UML (*Unified Modeling Language*) [Medvidovic et al. 2002] e as ADLs (*Architecture Description Languages*) [Medvidovic and Taylor 2000], respectivamente.

Arquitetos frequentemente encontram restrições na escolha das abordagens para recuperação. A maioria das soluções são dependentes da linguagem de programação e/ou plataforma de desenvolvimento utilizados. Adicionalmente, mesmo suportando o ambiente de desenvolvimento adotado, a técnica de recuperação pode não considerar as informações desejadas ou evidenciar aspectos arquiteturais não relevantes ao cenário em questão. Ainda, a notação de modelagem utilizada pela abordagem na descrição dos modelos recuperados pode não ter a expressividade e formalidade necessárias para conferir a utilidade desejada aos modelos resultantes. Por fim, a capacidade de identificação de conectores como elementos arquiteturais de primeira-classe é também fator importante na escolha por uma determinada técnica de recuperação arquitetural.

Sob esta perspectiva, este artigo apresenta o projeto e implementação de um *application framework* para recuperação arquitetural independente da plataforma de desenvolvimento, da notação de modelagem arquitetural utilizada e do algoritmo de recuperação adotado. Assim, tais *hot-spots* podem ser facilmente configurados, de modo a atender diferentes necessidades de usuários. Esta característica concede flexibilidade e extensibilidade ao permitir que desenvolvedores implementem novas formas de recuperação, suportem novas plataformas de desenvolvimento (combinações de tecnologias) e utilizem novas notações de modelagem. Adicionalmente, um *hot-spot* para definição de *code snippets* permite a especificação de mecanismos para recuperação de conectores.

O *framework* foi implementado como um componente da ferramenta DuSE-MT[1] [Andrade and Macêdo 2013] – *software* livre para modelagem e análise de arquiteturas, desenvolvido em C++ e Qt. O DuSE-MT oferece um ambiente flexível para criação e manipulação de modelos de *software*. Um núcleo independente de metamodelo é dispo-

[1] `http://duse.sf.net`

nibilizado e implementações das linguagens UML e MOF (*Meta-Object Facility*) estão atualmente disponíveis. Tanto o DuSE-MT quanto o *framework* apresentado neste trabalho podem ser utilizados em diversas plataformas, tais como Linux, Windows e MacOS.

Uma instanciação do *framework* para recuperação arquitetural de sistemas desenvolvidos em C++/Qt foi realizada. O GCC-XML [King 2014] foi utilizado como *backend* para obtenção de informações do código-fonte, enquanto a recuperação arquitetural foi realizada pelo algoritmo de clusterização ACDC (***A**lgorithm for **C**ompreension-**D**riven **C**lustering*) [Tzerpos 2001]. Os modelos recuperados são descritos em UML e gerados através do módulo *QtModeling*, parte integrante do DuSE-MT.

O restante deste artigo está organizado como segue. A seção 2 apresenta os fundamentos sobre recuperação arquitetural, enquanto a seção 3 discute os requisitos funcionais e não-funcionais do *framework* proposto, bem como a arquitetura, *hot-spots* e inversões de controle que caracterizam a solução. A seção 4 apresenta como o *framework* foi instanciado de modo a suportar a recuperação arquitetural de sistemas implementados em C++/Qt. A seção 5 apresenta como o *framework* foi validado. Por fim, a seção 6 discute os trabalhos correlatos e a seção 7 apresenta as conclusões e trabalhos futuros.

2. Recuperação Arquitetural

Compreender a arquitetura de um *software* e garantir a sua consistência à medida em que os artefatos de implementação evoluem são fatores críticos para a manutenção de sistemas. Entretanto, modificações arbitrariamente realizadas e que adicionam, removem ou modificam decisões arquiteturais previamente tomadas, frequentemente provocam desvios e/ou erosões arquiteturais [Taylor et al. 2009]. Em algum momento, é necessário recuperar a arquitetura descritiva (reificada nos artefatos de implementação) para fins de análise ou comparação com a arquitetura prescrita pelo arquiteto.

A recuperação arquitetural [Garcia et al. 2013a] tem como objetivo inferir decisões tomadas sobre a arquitetura do *software* a partir de artefatos de implementação disponíveis e/ou investigação de características comportamentais do sistema em execução. Dentre as principais técnicas para recuperação arquitetural, destacam-se: clusterização hierárquica e *graph pattern matching*.

A clusterização hierárquica [Maqbool and Babri 2007] é utilizada para recuperação de visões arquiteturais estruturais a partir do código-fonte disponível. O objetivo é formar grupos de itens ou entidades, de modo que entidades dentro de um grupo sejam similares entre si e diferentes daquelas presentes em outros grupos. Cada entidade é caracterizada por um vetor de *features*, que é utilizado como entrada para alguma métrica de avaliação de similaridade. Uma matriz $n \times n$ contendo as similaridades entre cada par de entidades é então utilizada pelo algoritmo de clusterização para definição dos *clusters*. Um amplo conjunto de diferentes métricas para avaliação de similaridade está atualmente disponível na literatura [Maqbool and Babri 2007].

As técnicas baseadas em *graph pattern matching* [Sartipi 2003, Conte et al. 2004] analisam o código-fonte da aplicação para derivar grafos representando dependências entre entidades. Técnicas de *matching* são então empregadas para verificação de similaridades com arquiteturas de referência previamente definidas. Tanto algoritmos para *matching* exato (ex: *tree search*) quanto aproximado (ex: otimização contínua e métodos espectrais) estão disponíveis atualmente.

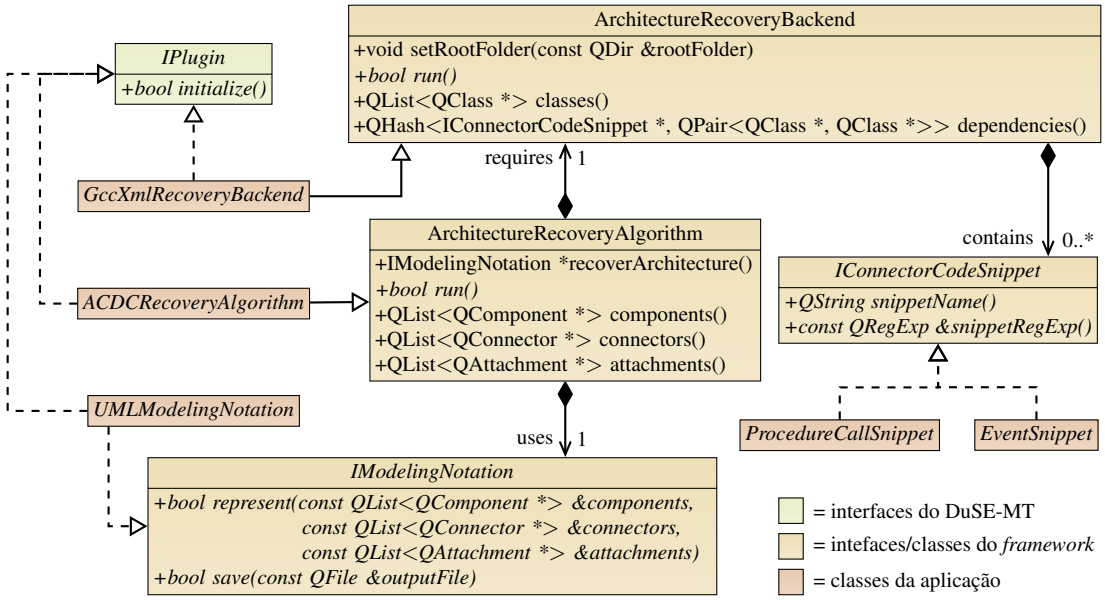

Figura 1. Interfaces do DuSE-MT, interfaces/classes definidas pelo *framework* e exemplo de classes implementando os *hot-spots* requeridos.

Um dos grandes problemas na área de recuperação arquitetural é a inexistência de teorias e *"first-principles"* que sedimentem uma avaliação comparativa mais precisa entre métodos de recuperação arquitetural. Trabalhos recentes [Garcia et al. 2013b, Garcia et al. 2012] tentar minimizar este problema através da definição de métodos parcialmente automatizados de recuperação.

3. O Framework Proposto

O *framework* foi projetado de modo a apresentar um alto grau de configurabilidade em relação a três aspectos principais (*hot-spots*): plataforma de desenvolvimento utilizada no *software* cuja arquitetura será recuperada; algoritmo de recuperação arquitetural adotado; e notação de modelagem arquitetural utilizada na representação dos modelos recuperados.

Conforme apresentado na Figura 1, o *framework* define uma *engine* genérica para recuperação arquitetural através da definição de quatro classes/interfaces que viabilizam a implementação dos *hot-spots* indicados. Tais classes/interfaces permitem a instanciação do *framework* de modo a recuperar arquiteturas de *softwares* desenvolvidos em uma linguagem de programação específica (ex: C++), utilizando um algoritmo particular de recuperação (ex: ACDC ou ARC [Maqbool and Babri 2007]) e descrevendo as arquiteturas recuperadas através de uma notação de modelagem específica (ex: UML, ADLs ou gráficos informais [Medvidovic and Taylor 2000]).

A classe `ArchitectureRecoveryAlgorithm` desempenha um papel central no *framework* pois define a API a ser implementada pelos diferentes algoritmos de recuperação arquitetural. O método *recoverArchitecture()* é um *template method* que delega a execução do método abstrato *run()* a subclasses e, em seguida, executa o método *represent()* do objeto que implementa a notação de modelagem sendo utilizada no momento (instância de `IModelingNotation`). Um algoritmo particular de recuperação deve implementar o método *run()* – utilizando as

informações de artefatos providas pelo *backend* de recuperação sendo utilizado (instância de `ArchitectureRecoveryBackend`) – e popular as estruturas de dados que representam os componentes, conectores e ligações (*attachments*) encontrados na recuperação (entregues pelos métodos *components()*, *connectors()* e *attachments()*, respectivamente). O *framework* define *bridges* de `ArchitectureRecoveryAlgorithm` com o *backend* de recuperação e com a notação de modelagem utilizados.

`ArchitectureRecoveryBackend` é a interface responsável por desacoplar, dos algoritmos de recuperação, os mecanismos utilizados para analisar artefatos descritos em diferentes plataformas de desenvolvimento. Linguagens de programação, tais como C++, Java ou PHP, requerem o uso de diferentes *parsers* para obtenção das informações elementares e de baixa granularidade que subsidiam a execução dos algoritmos de recuperação arquitetural. Dessa forma, o uso do *framework* aqui proposto na recuperação de arquiteturas de *softwares* descritos em uma determinada plataforma de desenvolvimento requer a implementação da classe `ArchitectureRecoveryBackend` para aquela plataforma, informando como o método *run()* analisa o código-fonte e obtém informações básicas sobre classes e dependências entre classes.

Conectores de *software* [Mehta et al. 2000] desempenham um papel crucial no projeto e análise de arquiteturas. Eles mediam a interação entre componentes e são fundamentais para a satisfação de certos requisitos não-funcionais. Dentre os conectores mais utilizados, destacam-se o *Procedure Call* – caracterizado por uma comunicação acoplada, síncrona e binária entre componentes – e o *Event* – geralmente utilizado para comunicação desacoplada, assíncrona e do tipo um-para-muitos. `IConnectorCodeSnippet` permite a definição de expressões regulares que analisam o código-fonte em busca de trechos que descrevem a utilização de um conector específico. O objetivo é flexibilizar a detecção de conectores não previstos e viabilizar a sua representação como elementos arquiteturais de primeira-classe.

`IModelingNotation` define a API a ser implementada de modo a suportar diferentes formas de representação dos artefatos arquiteturais recuperados. As notações utilizadas variam desde as mais informais, tais como linguagem natural e gráficos *ad-hoc*, até as notações mais formais e rigorosas, tais como as ADLs. Um determinada notação deve implementar o método *represent()* informando como os componentes, conectores e ligações – encontrados pelo algoritmo de recuperação em uso – deverão ser representados na notação de modelagem em questão. Por exemplo, para a notação UML, o método *represent()* implementa a instanciação e configuração das metaclasses que representam os elementos arquiteturais recuperados. Adicionalmente, o método *save()* pode ser implementado para realizar a serialização do modelo em algum formato previamente definido, como por exemplo a serialização de modelos UML no formato XMI (*XML Metadata Interchange*).

Todas as implementações concretas de *hot-spots* foram desenvolvidas como *plugins* do DuSE-MT e, portanto, implementam a interface `IPlugin` definida pela ferramenta. Este aspecto é importante para permitir a fácil instalação de novos algoritmos, *backends* e notações, bem como a configuração *on-the-fly* das implementações a serem utilizadas em uma determinada recuperação.

4. Instanciando o Framework

O *framework* proposto neste trabalho foi instanciado em uma aplicação que realiza a recuperação de arquiteturas de sistemas implementados em C++/Qt. Para isto, um *backend* de recuperação baseado na ferramenta GCC-XML [King 2014] foi desenvolvido, em conjunto com *code snippets* para detecção de conectores dos tipos *Procedure Call* e *Event* [Mehta et al. 2000]. O algoritmo de recuperação ACDC [Tzerpos 2001] foi também implementado, motivado pelo seu melhor desempenho e precisão de recuperação, conforme relatado em [Garcia et al. 2013a]. Uma implementação de notação de modelagem para geração de modelos UML representando a arquitetura recuperada foi também desenvolvida como parte da aplicação.

4.1. O backend de recuperação baseado no GCC-XML

O GCC-XML [King 2014] é uma extensão do `g++` que produz uma descrição XML de um programa C++/Qt a partir das representações utilizadas internamente pelo compilador. Tais representações em XML são úteis pois são fáceis de serem lidas e manipuladas, se comparadas ao desenvolvimento completo de *parsers* para a linguagem C++.

Conforme apresentado na Figura 1, a classe `GccXmlRecoveryBackend` utiliza o GCC-XML para exportar representações XML de todos os arquivos de código-fonte encontrados a partir do diretório *rootFolder*. Os mecanismos para manipulação de XML, presentes no *toolkit* Qt, são então utilizados para a obtenção das classes que compõem o sistema e os *code snippets* são investigados para identificar a forma de dependência e os conectores utilizados na comunicação entre instâncias das classes da aplicação. Esta operações são realizadas na implementação do método abstrato *run()* e populam as estruturas de dados retornadas pelos métodos *classes()* e *dependencies()*.

Duas instâncias de `IConnectorCodeSnippet` trabalham em conjunto com a classe `GccXmlRecoveryBackend`: `ProcedureCallSnippet` e `EventSnippet`. Tais *snippets* são responsáveis pela definição de expressões regulares que identificam invocações síncronas convencionais de métodos e utilização do mecanismo de *signals/slots* do Qt (implementação do conector *Event*), respectivamente.

4.2. O algoritmo de recuperação arquitetural ACDC

O ACDC [Tzerpos 2001] é um algoritmo de recuperação arquitetural que integra detecção de padrões comumente adotados em recuperações manuais às operações básicas de *clusterização* de entidades. O objetivo é produzir *clusters* que não abstraiam demasiadamente a estrutura interna do *software* e que utilizem rótulos significativos.

O algoritmo é composto por dois estágios principais: *i)* aplicação dos padrões de detecção para geração da decomposição que representa a arquitetura; e *ii)* alocação de entidades órfãs a *clusters* já existentes. O algoritmo define critérios para detecção dos seguintes padrões: *source file clusters* (aglomeração de entidades presentes em um mesmo arquivo de código-fonte), *body-header conglomeration* (aglomeração de declarações e definições de entidades), *leaf collection and support library identification* (entidades com número de dependentes maior que 20) e *ordered and limited subgraph domination* (uso de conjuntos dominantes de grafos para detecção de sub-sistemas).

A adoção de órfãos é uma técnica de clusterização incremental, utilizada no ACDC para alocar entidades não clusterizadas na primeira fase a algum *cluster* já exis-

tente. Quando mais de um *cluster* pré-existente são igualmente propensos à adoção de uma determinada entidade, um novo *cluster* é criado.

O ACDC apresenta três características interessantes. Primeiro, a cada novo subsistema (*cluster*) criado pelo ACDC, um rótulo significativo é a ele atribuído, por exemplo, o nome do nó dominador do conjunto dominado encontrado. Segundo, a cardinalidade dos *clusters* encontrados é sempre limitada (não mais que 20 entidades), de modo a não abstrair demasiadamente a estrutura da aplicação. Por fim, a utilização dos padrões produz recuperações mais facilmente compreensíveis pois refletem as operações comuns de recuperação executadas manualmente por arquitetos.

4.3. Suportando a notação de modelagem UML

Após a execução de um determinado algoritmo de recuperação, os componentes, conectores e a topologia da arquitetura recuperada estão disponíveis através dos métodos *components()*, *connectors()* e *attachments()*. Entretanto, é necessário representar a arquitetura através do uso de alguma notação de modelagem.

Para viabilizar a geração de modelos arquiteturais descritos em UML, a classe `UmlModelingNotation` (apresentada na Figura 1) implementa a interface `IModelingNotation` e utiliza os recursos do módulo *QtModeling* – parte integrante do DuSE-MT – para criar e exportar um diagrama de componentes que represente a arquitetura recuperada. O *QtModeling* implementa a versão 2.4.1 dos metamodelos das linguagens UML e MOF, bem como as operações de serialização de modelos no padrão XMI.

Para isso, cada componente resultante do algoritmo de recuperação é instanciado como um componente UML. Dependências UML, anotadas com um estereótipo que descreve um tipo particular de conector, são utilizadas para representar interações entre componentes. O modelo UML é serializado no formato XMI e pode então ser visualizado no DuSE-MT ou em qualquer outra ferramenta em conformidade com a especificação XMI.

5. Avaliação

Visto que este trabalho não tem como objetivo a proposta de novas técnicas para recuperação arquitetural, a avaliação do *framework* proposto busca analisar dois aspectos fundamentais: *i)* adequação dos *hot-spots* às variações atualmente existentes; e *ii)* facilidade de compreensão e de instanciação do *framework*, por desenvolvedores.

Em relação ao primeiro aspecto, realizou-se um estudo para verificação da adequação da API definida no *framework* à implementação dos principais algoritmos de recuperação arquitetural, *backends* para obtenção de artefatos e notações de modelagem. O objetivo é verificar se a API não é restrita demais a ponto de inviabilizar a utilização de certos algoritmos de recuperação, ou ampla demais a ponto de não trazer benefícios à produtividade de desenvolvimento.

Resultados indicam que os seis algoritmos de recuperação mais utilizados atualmente (ACDC, WCA, LIMBO, Bunch, ZBR e ARC) são facilmente suportados, embora nenhuma facilitação na implementação destes algoritmos seja fornecida pelo *framework*. Uma possível melhoria é a definição de subcategorias de algoritmos (ex: baseados em clusterização hierárquica, *graph pattern matching*, etc), onde *template methods* disponibi-

lizariam uma infrastrutura que melhoraria a produtividade na implementação de técnicas particulares (*hot-spots* de menor granularidade).

Experimentos controlados estão sendo atualmente realizados para aceitar/rejeitar a hipótese de que o *framework* aqui proposto traz melhorias na produtividade e pode ser facilmente utilizado, quando comparado à implementação de técnicas de recuperação arquitetural sem o uso de *frameworks* ou com *frameworks* correlatos.

6. Trabalhos Correlatos

Diversos esforços para recuperação arquitetural podem ser encontrados na literatura. [Risi et al. 2012] apresentam uma abordagem para recuperação arquitetural baseada na utilização de *Latent Semantic Indexing* (LSI) e clusterização via *k-means*. Adicionalmente, eles definem mecanismos para adição e remoção de componentes na representação LSI (*fold-in/fold-out*), reduzindo o custo computacional necessário para a recuperação.

[Wu et al. 2007] propõem uma abordagem para recuperação arquitetural que é utilizada como um guia para processos subsequentes de re-engenharia de sistemas legados. Técnicas para análise de dependência entre módulos são utilizadas para a obtenção de recuperações que apresentem componentes com alta coesão, baixo acoplamento e interfaces de comunicação mínimas.

[Platenius et al. 2012] apresentam uma abordagem para recuperação arquitetural que considera a presença das deficiências de projeto mais relevantes. O método proposto apresenta os benefícios alcançados caso as deficiências detectadas sejam removidas, como base para a execução de um processo de re-engenharia. Os componentes mais relevantes e as deficiências de projeto são detectados através do uso de métricas que avaliam incompatibilidade de interfaces e organização lógica de classes em pacotes.

[von Detten and Becker 2011] apresentam uma abordagem híbrida que combina técnicas de clusterização e detecção de padrões para realização de recuperação arquitetural. A abordagem utiliza ainda a detecção e remoção de *bad smells* para melhorar os resultados das operações de clusterização.

Um *plugin* para o Eclipse – denominado MARPLE – suportando a detecção de *design patterns* e recuperação de arquiteturas é apresentado em [Fontana and Zanoni 2011]. O *plugin* baseia-se na utilização de métricas e elementos básicos extraídos da árvore sintática abstrata que representa o código-fonte. A detecção de *design patterns* é realizada através da identificação de subcomponentes que indicam a presença de um *design pattern* particular. Experimentos para detecção de *Abstract Factories* e geração de visões arquiteturais são também apresentados no artigo.

[Sartipi 2003] apresenta um modelo e uma ferramenta para recuperação arquitetural, baseados na utilização de padrões arquiteturais definidos pelo usuário e de técnicas de *graph pattern matching*. Experimentos avaliando a complexidade de tempo e espaço da recuperação, bem como a precisão dos artefatos recuperados são citados no artigo.

[Huang et al. 2006] apresentam uma técnica para recuperação arquitetural baseada na obtenção de dados comportamentais de um *software* em execução. Adicionalmente, modificações em *run-time* podem ser efetuadas no sistema através da manipulação direta do modelo recuperado. Os autores utilizam uma plataforma de *middleware* com capacidades de reflexão computacional para implementar a recuperação e a adaptação

dinâmica. Experimentos com o PKUAS – um servidor de aplicação baseado no JEE – são apresentados ao final.

O *framework* apresentado neste artigo difere dos trabalhos acima citados em uma série de aspectos. Primeiro, a infraestrutura aqui proposta permite a recuperação de arquiteturas sem uma dependência direta com uma determinada plataforma de desenvolvimento, algoritmo de recuperação ou notação de modelagem. Segundo, a solução dá ênfase à recuperação e representação de conectores de *software* como entidades arquiteturais de primeira-classe. Tal característica é importante em sistemas cujo atendimento de requisitos de escalabilidade, tolerância a falhas e segurança é merito majoritariamente do uso de conectores sofisticados. Por fim, o *framework* aqui proposto pode constituir plataforma eficiente para comparação de técnicas de recuperação e/ou definição de *benchmarks* a serem utilizados em futuras pesquisas.

7. Conclusões e Trabalhos Futuros

Este artigo apresentou o projeto e implementação de um *framework* para recuperação arquitetural independente de plataforma, de algoritmo de recuperação e de notação de modelagem das arquiteturas recuperadas. Um exemplo de instanciação do *framework* para recuperação de arquiteturas de sistemas implementados em C++ foi também apresentado.

Possíveis trabalhos futuros incluem a definição de mecanismos mais sofisticados para detecção de conectores *composite*, projeto de *hot-spots* de menor granularidade para facilitar a implementação de algoritmos específicos de recuperação e melhor suporte a algoritmos de clusterização através da disponibilização prévia de métricas de similaridade.

Referências

[Andrade and Macêdo 2013] Andrade, S. S. and Macêdo, R. J. d. A. (2013). A search-based approach for architectural design of feedback control concerns in self-adaptive systems. In *Proceedings of the 7th IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, SASO 2013, Philadelphia, PA, USA. IEEE.

[Conte et al. 2004] Conte, D., Foggia, P., Sansone, C., and Vento, M. (2004). Thirty years of graph matching in pattern recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, 18(3):265–298.

[Fontana and Zanoni 2011] Fontana, F. A. and Zanoni, M. (2011). A tool for design pattern detection and software architecture reconstruction. *Information Sciences*, 181(7):1306–1324.

[Garcia et al. 2013a] Garcia, J., Ivkovic, I., and Medvidovic, N. (2013a). A comparative analysis of software architecture recovery techniques. In *IEEE/ACM 28th International Conference on Automated Software Engineering (ASE)*, pages 486–496. IEEE.

[Garcia et al. 2013b] Garcia, J., Krka, I., Mattmann, C., and Medvidovic, N. (2013b). Obtaining ground-truth software architectures. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 901–910. IEEE Press.

[Garcia et al. 2012] Garcia, J., Krka, I., Medvidovic, N., and Douglas, C. (2012). A framework for obtaining the ground-truth in architectural recovery. In *Joint Working IEEE/IFIP Conference on Software Architecture (WICSA) and European Conference on Software Architecture (ECSA)*, pages 292–296. IEEE.

[Huang et al. 2006] Huang, G., Mei, H., and Yang, F.-Q. (2006). Runtime recovery and manipulation of software architecture of component-based systems. *Automated Software Engineering*, 13(2):257–281.

[King 2014] King, B. K. (2014). GCC-XML. http://gccxml.github.io/. Acesso: 29/04/2014.

[Maqbool and Babri 2007] Maqbool, O. and Babri, H. (2007). Hierarchical clustering for software architecture recovery. *IEEE Transactions on Software Engineering*, 33(11):759–780.

[Medvidovic et al. 2002] Medvidovic, N., Rosenblum, D. S., Redmiles, D. F., and Robbins, J. E. (2002). Modeling software architectures in the unified modeling language. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 11(1):2–57.

[Medvidovic and Taylor 2000] Medvidovic, N. and Taylor, R. N. (2000). A classification and comparison framework for software architecture description languages. *IEEE Transactions on Software Engineering*, 26(1):70–93.

[Mehta et al. 2000] Mehta, N. R., Medvidovic, N., and Phadke, S. (2000). Towards a taxonomy of software connectors. In *Proceedings of the 22nd International Conference on Software Engineering*, ICSE '00, pages 178–187, New York, NY, USA. ACM.

[Platenius et al. 2012] Platenius, M. C., von Detten, M., and Becker, S. (2012). Archimetrix: Improved software architecture recovery in the presence of design deficiencies. *2011 15th European Conference on Software Maintenance and Reengineering*, 0:255–264.

[Risi et al. 2012] Risi, M., Scanniello, G., and Tortora, G. (2012). Using fold-in and fold-out in the architecture recovery of software systems. *Formal Aspects of Computing*, 24(3):307–330.

[Sartipi 2003] Sartipi, K. (2003). Software architecture recovery based on pattern matching. In *Proceedings of the International Conference on Software Maintenance (ICSM)*, pages 293–296. IEEE.

[Taylor et al. 2009] Taylor, R. N., Medvidovic, N., and Dashofy, E. M. (2009). *Software Architecture: Foundations, Theory, and Practice*. Wiley Publishing.

[Tzerpos 2001] Tzerpos, V. (2001). *Comprehension-Driven Software Clustering*. PhD thesis, University of Toronto.

[von Detten and Becker 2011] von Detten, M. and Becker, S. (2011). Combining clustering and pattern detection for the reengineering of component-based software systems. In *Proceedings of the Joint ACM SIGSOFT Conference – QoSA and ACM SIGSOFT Symposium – ISARCS on Quality of Software Architectures – ISARCS*, QoSA-ISARCS '11, pages 23–32, New York, NY, USA. ACM.

[Wu et al. 2007] Wu, L., Feng, Y., and Yan, H. (2007). Software reengineering with architecture decomposition. In *Proceedings of the 2007 ACM Symposium on Applied Computing*, SAC '07, pages 1489–1493, New York, NY, USA. ACM.