USP - UNIVERSIDADE DE SÃO PAULO
IME - INSTITUTO DE MATEMÁTICA E ESTATÍSTICA
DCC - DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

Arquitetura de Software

MAC5701 - Tópicos em Ciência da Computação

- Relatório de Estudos -

Ney Barros Luz

São Paulo, junho de 2002

Resumo

Este documento refere-se ao relatório de estudos, requisito para obtenção de créditos na disciplina MAC5701, Tópicos em Ciência da Computação.

O relatório consolida o material estudado sobre Arquitetura de Software e apresenta uma análise da relação entre o material estudado e o tema de pesquisa do mestrado.

Sumário

Capítulo 1

Introdução

Nos últimos anos a área de Arquitetura de Software cresceu em importância dentre as áreas de pesquisa voltadas ao desenvolvimento de Software. A idéia de criar um mecanismo de comunicação entre os participantes no desenvolvimento de um software, a possibilidade de antecipar um conjunto de decisões de projeto sobre um dado sistema e ainda, permitir que o modelo de abstração contruído seja passível de utilização num outro contexto constituem razões pelas quais essa área vem recebendo uma especial atenção.

Uma arquitetura de software consiste na estrutura dos componentes de um sistema, seus interrelacionamentos, princípios e linhas mestras que dão a sustentação ao projeto e a evolução no decorrer do tempo [CN96].

Por um outro lado, um grupo que atua na área de Processamento Digital de Imagens vem desenvolvendo, ao longo dos últimos anos, um software que implementa um procedimento de projeto automático de operadores de imagens a partir de exemplos de treinamento [Tom96]. Este processo é fundamentado na teoria de Morfologia Matemática [Ser82, Hei94] e Aprendizado Computacional [AB92, Hau92].

Esse sistema está implementado na plataforma KHOROS [KR94], no ambiente UNIX, e constitui uma *toolbox* denominada PAC. Os programas desse sistema estão divididos em três grupos: módulo de treinamento (que implementa o procedimento de projeto automático), módulo de aplicação (que possibilita a aplicação dos operadores projetados) e módulo de ferramentas auxiliares. O sistema PAC pode ser utilizado para projetar operadores para diversos tipos de processamentos de imagens (filtragem, reconhecimento de padrões, extração de textura, etc).

Recentemente percebeu-se que a parte central do procedimento de projeto de operadores de imagens do sistema implementado pode ser essencialmente reduzida ao procedimento de projetar classificador de padrões. Classificadores de padrões podem ser utilizados para a análise de dados em geral [CM98, Vap98]. Por exemplo, uma classe de dados cuja análise vem demandando especial atenção dos pesquisadores recentemente são os dados gênicos [AFV94, BO98]. Estas constatações motivaram no grupo o desejo de expandir o sistema, de modo que ele seja útil não somente para a análise de imagens, mas também para a análise de outros tipos de dados. Acreditamos que isto será possível desde que sua arquitetura permita a expansão.

O nosso objetivo foi efetuar o estudo de um conjunto de artigos relacionados a Arquitetura de Software. A finalidade destes estudos foi familiarizar-se com esta área para, posteriormente, aplicar os conhecimentos adquiridos para modelar a arquitetura de um software voltado para projeto de classificadores, expandindo o sistema PAC atual.

Capítulo 2

O que é arquitetura de software

A arquitetura de software de um programa ou sistema de computação é a estrutura ou estruturas do sistema, que compreende os componentes de software, as propriedades visíveis externamente destes componentes, e as relações entre eles. [BCK98]. A expressão "propriedades visíveis externamente" refere-se ao que outros componentes podem se utilizar de um dado componente, tais como seus serviços.

A arquitetura é uma abstração de sistema, definindo componentes e como eles interagem. Ela desconsidera detalhes internos sobre os componentes. Tais informações não fazem parte da arquitetura.

Os sistemas podem ter muitas visões. Nenhuma visão isoladamente pode ser considerada como a arquitetura e o conjunto de visões de uma arquitetura não é fixado a priori, ou seja, será aquilo que for útil para a análise, comunicação ou compreensão. Cada visão ressalta os detalhes que lhe competem, ignorando os outros detalhes que serão ressaltados em outras visões [Var02].

A idéia por trás da definição é a de abstração. Uma arquitetura de software deve considerar isoladamente um aspecto do sistema de maneira que esse aspecto possa se constituir num meio de análise e estudo.

A definição de arquitetura de software acima gera algumas implicações. Dentre elas temos:

- uma arquitetura define componentes e informações sobre como os componentes interagem um com o outro. Isso quer dizer que uma arquitetura omite informações sobre componentes que não fazem parte da interação;

- nenhuma estrutura do sistema pode ser dita como sendo a arquitetura.

A omissão na definição do que são os componentes e os relacionamentos é intencional, de maneira que tais componentes possam ser definidos de acordo com um contexto.

- todo software possui uma arquitetura. Isso ocorre porque todo sistema pode, de alguma maneira, ser exibido como um composição de componentes e relações entre esses componentes;

- os comportamentos dos componentes fazem parte da arquitetura, pois são a partir dele que são feitas a interações.

Exemplos de possíveis componentes são objetos, processos ou bibliotecas; clientes, servidores, filtros, camadas e bancos de dados. Podem ser programas, módulos ou até mesmo sistemas.

Denominaremos a interação entre os componentes como conectores. Exemplos de conectores são chamadas a procedimentos e variáveis de acesso compartilhado. Outros exemplos são protocolos cliente-servidor, protocolos de acesso a banco de dados, eventos broadcast e *piped streams*.

Capítulo 3

Conceitos preliminares

Arquitetura de software, assim como outras áreas de estudo, possui uma terminologia abordando diversos de seus conceitos. Alguns deles estão descritos a seguir:

Componente computacional: uma porção de uma partição do sistema, compreendendo um contexto, seus casos de uso, modelos de análise, modelos de projeto, especificações de implementação, especificações de testes e uma forma de interação com os outros componentes computacionais.

Componente software: uma porção de uma partição do sistema software, sendo representado como módulos, funções, pacotes ou classes.

Disponibilidade: a proporção entre quanto tempo o sistema está operando e entregando seus serviços, e o tempo total.

Elementos de um sistema baseado em computador: Pessoas, Procedimentos, Documentação, Hardware, Software e Banco de dados.

Estilo Arquitetônico: é um esquema de organização estrutural, estabelecido como um padrão de utilização, contemplando um conjunto de componentes, suas responsabilidades e a forma de interação entre eles.

Performance: capacidade de um software alocar seus recursos computacionais para atender solicitações de serviço satisfazendo a requisitos de tempo.

Produto software: Programas de computadores e documentação associada.

Relacionamento: denota uma conexão entre componentes. Um relacionamento pode ser estático ou dinâmico. Relacionamentos estáticos podem ser apontados diretamente no código fonte. Relacionamentos dinâmicos são obtidos a partir de conexões temporais e interações dinâmicas entre os componentes, não podendo ser visíveis a partir da estrutura estática do código fonte. Exemplos de relacionamentos estáticos são a agregação e herança. Já exemplos de relacionamentos geralmente dinâmicos são a criação de objetos, comunicação entre objetos e transferência de dados.

Sistema legado: Qualquer sistema já existente.

Sistema software: um número de programas separados, arquivos de configuração, documentação de software e documentação de usuário.

Visão arquitetônica : de maneira abstrata, corresponde a observação do software sob uma perspectiva. Concretamente, corresponde a um diagrama descrevendo essa perspectiva. No diagrama são representados os componentes e as relações entre esses componentes, detectados a partir da perspectiva e somente por esta. Um conjunto dessas visões arquitetônicas determinam então uma arquitetura para o software. Uma visão ou um conjunto de visões arquitetônicas são determinados a partir dos papéis desempenhados por cada um dos vários participantes do processo software. Uma arquitetura de software completa seria, portanto, aquela formada pelo conjunto de visões arquitetônicas que contemplam as visões de todos os participantes.

Capítulo 4

Tópicos em Arquitetura de Software

4.1 Estilos arquitetônicos

Estilos arquitetônicos são padrões de organização de sistema. Uma catalogação de estilos mais comuns é dada em [SG96].

Um estilo arquitetônico consiste de uma coleção de tipos de componentes com uma descrição do padrão de interação entre eles.

4.1.1 Pipes/Filters

No estilo arquitetônico Pipes/Filters os componentes são os *filters* e os conectores são os *pipes*. Os componentes possuem um conjunto de entradas e um conjunto de saídas e lê *streams* de dados na entrada e produz *strems* de dados na saída. Os conectores *pipes*, por sua vez, transmitem saídas de um filtro para a entrada de um outro.

O ponto chave do etilo pipes/filters consiste nos filtres serem entidades independentes, ou seja, não compartilham estados com outros filtros.

Exemplos de *pipers/filters* são os programas escritos em Unix Shell. Algumas de suas propriedades são:

- o comportamento do sistema é dado pela composição dos comportamentos de todos os filtros individuais;
- o estilo suporta bem o aspecto de reuso, pois permite que quaisquer

dois filtros sejam utilizados, uma vez que haja compatibilidade entre a saída do primeiro com a entrada do segundo;

- são fáceis de manter e atualizar;
- permitem determinados tipos de análise especializada como *throughput* e *deadlock*.

4.1.2 Abstração de dados e Organização Orientada a Objetos

Nesse estilo, os componentes são objetos e a iteração entre eles é feita a partir da invocação de funções e procedimentos As propriedades desse estilo são:

- o objeto é responsável pela integridade de sua representação;
- a representação é oculta de outros objetos

A vantagem desse estilo está na possibilidade de troca de implementação sem afetar os clientes. Mas um outro aspecto relevante, e que pode caracterizar como desvantagem, para um objeto interagir com outro ele precisa saber a identidade do segundo.

4.1.3 Invocação implícita

Na invocação implícita um componente oferece um conjunto de eventos como interface. Em vez de invocar um procedimento diretamente, um componente anuncia um evento. Os outros componentes podem ter interesse nesse evento ou não. Uma propriedade peculiar desse estilo é que não se sabe que componentes serão afetados pela ocorrência de um evento.

4.1.4 Sistemas em camadas

No estilo em camadas, a organização é hierárquica e cada camada inferior oferece serviços para a camada superior que lhe é adjacente. Exemplos de utilização desse estilo são os protocolos de comunicação. São propriedades desse estilo:

- cada camada interage com no máximo duas outras camadas;

- permite o reuso, uma vez que a possibilidade de padronizar a interface admite implementações por diversos implementadores

4.2 Projeto de software em nível de arquitetura

Organizamos nossa forma de raciocionar a partir de abstrações. Podemos entender o conceito de abstração como o processo de isolarmos coisas, separando elementos de um todo para efeitos de uma análise. É assim que percebemos o mundo. Isso implica que outros elementos são desconsiderados, ou considerados irrelevantes para o aspecto em estudo. A abstração também envolve considerar um conjunto de elementos como sendo uma unidade. Um elemento do conjunto não possui identidade suficiente para ser considerado relevante unicamente e faz parte de um conjunto maior, esse sim, importante para a questão em análise.

Em computação, organizamos nossa forma de percepção de maneira idêntica. Nos utilizamos de elementos concretos e/ou abstratos em diferentes níveis, dependendo do aspecto com que estamos nos preocupando. Por exemplo, quando nos atemos a aspectos a nível de circuito num computador, observamos elementos tais como flip-flops, contadores, codificadores, decodificadores, etc. Em termos de software, no nível de programação, os elementos são abstratos e consistem em algoritmos, estruturas de dados, programas e subprogramas.

Contudo, conforme aumenta o tamanho e complexidade de um sistema software, o projeto e a especificação da estrutura do sistema torna-se um aspecto mais significativo do que somente a escolha de algoritmos e estruturas de dados. Um novo nível de abstração parece se tornar necessário, sob o risco de nos perdermos olhando as árvores e não percebendo, de fato, a floresta.

A idéia de solução consiste em mudar o enfoque. Em geral, não vemos um programa como um conjunto de instruções em assembly mas sim como um conjunto de algoritmos e estruturas de dados. A mudança de enfoque então seria visualizar o software sob um outro nível de abstração, que consiste em perceber o software como um conjunto de componentes, onde esses componentes podem variar em complexidade, desde unidades como classes até mesmo a sistemas gerenciadores de bancos de dados inteiros. A escolha de quais componentes representar e a relação a ser descrita entre eles dependerá de qual aspecto de arquitetura se pretende abordar.

Um projeto de software a nível de arquitetura consistiria então em visualizar o software como um conjunto de abstrações envolvendo componentes e relações entre esses componentes.

Os aspectos estruturais envolvem a organização de um sistema como uma composição de componentes, estruturas de controle globais, protocolos para comunicação, sincronização e acesso a dados.

4.3 Avaliação de arquiteturas

Como garantir que a arquitetura escolhida para o software está correta? Esta questão não é fácil de ser resolvida. Modificabilidade, performance, segurança, disponibilidade e confiança de um sistema software são atributos de qualidade determinados a partir da arquitetura. É esse raciocínio uma das principais motivações para visualizar a arquitetura de software como uma área [CKK02]. Acredita-se que decisões arquitetônicas tem um profundo impacto nos atributos de qualidade. Na avaliação de um arquitetura, portanto, deve ser feita uma avaliação das decisões de projeto para verificar se elas atendem aos requisitos dos atributos de qualidade.

Os pré-requisitos para uma avaliação são um conjunto de requisitos dos atributos de qualidade e uma especificação da arquitetura que inclua claramente as decisões de projeto arquitetônico. Tais pré-requisitos não são freqüentemente encontrados na prática, de maneira que deve-se dar uma atenção para elicitá-los e refiná-los.

4.4 Arquiteturas de software específicas a domínio

Arquiteturas de software específica a domínio consiste num conjunto de abstrações particulares a um dado domínio, tais como hospitais, escolas, empresas, fábricas, etc. Estes domínios podem compartilhar algumas abstrações, tais como funções para obter dados, monitorar, controlar e oferecer suporte a decisão. Mas, dependendo das características de seus usuários típicos, seu ambiente, requisitos de qualidade (segurança de informação, por exemplo), esses domínios poderão ter abstrações diferenciadas.

4.5 Arquitetura reversa

O conceito de arquitetura reversa está descrito em [OS99].

A arquitetura reversa está intimamente relacionada com o processo de manutenção de software e é voltada para os sistemas que não possuem uma descrição explícita de sua arquitetura, servindo então como uma forma de se obter a sua representação arquitetônica.

A arquitetura reversa pode ser vista como sendo o aproveitamento do conceito de engenharia reversa para o nível de arquitetura, consistindo no processo de analisar um sistema legado com o objetivo de identificar os principais componentes e os relacionamentos entre eles.

O objetivo da arquitetura reversa é promover uma melhor compreensão do sistema, num nível mais alto, para posteriores manutenções e a representação arquitetônica é obtida a partir de análises de códigos-fontes, documentação, histórico do sistema, analistas, programadores, usuários e do próprio sistema no ambiente operacional, dentre outros.

4.6 Rearquitetura

Outro conceito descrito em [OS99] é o de rearquitetura.

A rearquitetura consiste numa análise e alteração de arquiteturas de sistemas, de maneira a se obter uma nova arquitetura e, a partir daí, um novo sistema.

Esse conceito também pode ser visto como a adaptação do entendimento de reengenharia, um outro conceito já conhecido no vocabulário da engenharia de software, consistindo na reestruturação de uma arquitetura com o objetivo de satisfazer a um conjunto de requisitos.

A rearquitetura se apóia na arquitetura reversa para o caso dos sistemas legados e seu uso típico atualmente envolve a troca de aplicações monolíticas para arquiteturas distribuídas.

Capítulo 5

Relação entre o material estudado e o tema de pesquisa

O nosso objetivo foi efetuar o estudo de um conjunto de artigos relacionados a Arquitetura de Software. A finalidade destes estudos foi primariamente familiarizar-se com esta área.

O trabalho previsto para a dissertação envolve modelar a arquitetura de um software voltado para projeto de classificadores.

O material estudado se relaciona com o tema de pesquisa na medida em que ele ofereceu a aquisição de conhecimentos técnicos fundamentais da área de Arquitetura de Software. Tais conhecimentos solidificaram os aspectos teóricos de maneira que, a partir daí, possam ser desenvolvidos os aspectos práticos do tema de pesquisa.

Referências Bibliográficas

[AB92] M. Anthony and N. Biggs. *Computational Learning Theory - An Introduction*. Cambridge University Press, 1992.

[AFV94] M. D. Adams, C. Fields, and J. C. Venter, editors. *Automated DNA Sequencing and Analysis*. Acdemic Press, 1994.

[BBC+00] F. Bachmann, L. Bass, J. Carriere, P. Clements, D. Garlan, J. Ivers, R. Nord, and R. Little. Software Architecture Documentation in Practice: Documenting Architectural Layers. Technical Report CMU/SEI-2000-SR-004, The Software Engineering Institute, March 2000.

[BCK98] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Addison-Wesley, 1998.

[BKW97] M. Barbacci, M. Klein, and C. Weinstock. Principles for Evaluating the Quality Attributes of a Software Architecture. Technical Report CMU/SEI-96-TR-036, The Software Engineering Institute, Carnegie Mellon University, March 1997.

[BO98] A. D. Baxevanis and B. F. F. Ouellette, editors. *Bioinformatics - A Practical Guide to the Analysis of Genes and Proteins*. John Wiley, 1998.

[CKK02] P. Clements, R. Kazman, and M. Klein. *Evaluating Software Architectures: Methods and Case Studies*. Addison-Wesley, 2002.

[CM98] V. Cherkassky and F. Mulier. *Learning from Data – Concepts, Theory and Methods*. John Wiley & Sons, 1998.

[CN96] P. Clements and L. Northrop. Software Architecture: An Executive Overview. Technical Report CMU/SEI-96-TR-003, The Software Engineering Institute, Carnegie Mellon University, February 1996.

[FOL02] FOLDOC. Free On-Line Dictionary Of Computing. [online] Disponível na Internet via WWW. URL: http://wombat.doc.ic.ac.uk/foldoc/index.html, Junho 2002.

[GHJV95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[GS94] D. Garlan and M. Shaw. An Introduction to Software Architecture. Technical Report CMU-CS-94-166, School of Computer Science, Carnegie Mellon University, January 1994.

[Hau92] D. Haussler. Decision Theoretic Generalizations of the PAC Model for Neural Net and Other Learning Applications. *Information and Computation*, 100:78–150, 1992.

[Hei94] H. J. A. M. Heijmans. *Morphological Image Operators*. Academic Press, Boston, 1994.

[JRvdL00] M. Jazayeri, A. Ran, and F. van der Linden. *Software Architecture for Product Families: Principles and Practice*. Addison Wesley, 2000.

[KABC96] R. Kazman, G. Abowd, L. Bass, and P. Clements. Scenario-based analysis of software architecture. *IEEE Software*, pages 47–55, November 1996.

[KR94] K. Konstantinides and J. Rasure. The KHOROS Software Development Environment for Image and Signal Processing. *IEEE Transactions on Image Processing*, 3(3):243–252, 1994.

[OS99] W. Onyino and I. Sommerville. Architecture Centric Software Engineering: Taxonomy Of Terms. Technical Report CSEG/7/1999, Computing Department, Lancaster University, 1999.

[Pre96] R. S. Pressman. *Software Engineering: A Practitioner's Approach.* MacGraw-Hill, 1996.

[PW92] D. E. Perry and A. L. Wolf. Foundations for the study of software architecture. *ACM SIGSOFT Software Engineering Notes*, 17(4):40–52, October 1992.

[Ser82] J. Serra. *Image Analysis and Mathematical Morphology.* Academic Press, 1982.

[SG96] M. Shaw and D. Garlan. *Software Architecture: Perspectives on an Emerging Discipline.* Prentice-Hall, 1996.

[Som01] I. Sommervile. *Software Engineering.* Addison Wesley, 6th edition, 2001.

[Tom96] Nina Sumiko Tomita. Programação Automática de Máquinas Morfológicas Binárias baseada em Aprendizado PAC. Master's thesis, Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, SP - Brasil, 1996.

[Vap98] V. N. Vapnik. *Statistical Learning Theory.* John Wiley and Sons, 1998.

[Var02] Ane Cristina Varoto. Visões em arquitetura de software. Master's thesis, Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, SP - Brasil, Março 2002.

[WS92] W. Waite and A. Sloane. Software Synthesis via Domain-Specific Software Architectures. Technical Report CU-CS-611-92, University of Colorado, September 1992.

[You02] Yourdictionary. The Global Language Resource. [online] Disponível na Internet via WWW. URL: http://www.yourdictionary.com, Junho 2002.