INF016 – Arquitetura de Software

08 - Implementação

Sandro Santos Andrade
sandroandrade@ifba.edu.br

Instituto Federal de Educação, Ciência e Tecnologia da Bahia
Departamento de Tecnologia Eletro-Eletrônica
Graduação Tecnológica em Análise e Desenvolvimento de Sistemas

Introdução

- A arquitetura de um *software* é projetada através de princípios e conhecimentos de engenharia, aumentando a confiança de que as qualidades do sistema atendam às exigências dos usuários
- Portanto, a implementação deve ser derivada da arquitetura
- A arquitetura é:
  - Prescritiva: como estruturar módulos, suas inter-conexões e comportamentos
  - Restritiva: formas de comunicação proibidas, tipos de comportamentos e estados não permitidos, etc

Introdução

- Mapeamento:
  - Relacionamento da arquitetura em artefatos de implementação
- Não necessariamente é um-para-um:
  - Geralmente um componente ou conector é implementado por muitos artefatos
  - Uma única biblioteca pode ser compartilhada por muitos componentes
- Quando este mapeamento não é mantido, potencializa-se a ocorrência de degradações arquiteturais

Introdução

- Como conceitos arquiteturais podem ser mapeados em linguagens de programação, ambientes de desenvolvimento, bibliotecas e componentes reutilizáveis, *frameworks*, *middleware* etc ?
- *Middleware* e modelos de componentes possuem decisões próprias que podem influenciar ou conflitar com as decisões arquiteturais já tomadas

Conceitos

- O problema do mapeamento
  - *Traceability*: qualquer mecanismo para conectar diferentes artefatos de *software*
    - Em particular, para conectar a arquitetura à implementação
  - Mapeamento de diferentes tipos de decisões arquiteturais:
    - Componentes e Conectores:
      - Exemplos de partições durante a implementação: *packages*, bibliotecas e classes
      - O problema então é mapear componentes e conectores do nível arquitetural em partições do nível de implementação
      - Se a implementação não é particionada de acordo com as fronteiras de componentes e conectores descritas na arquitetura, potencializa-se a ocorrência de degradação

Conceitos

- O problema do mapeamento
  - Mapeamento de diferentes tipos de decisões arquiteturais:
    - Interfaces:
      - Se especificadas em termos de assinaturas de métodos o mapeamento se resume na tradução de assinaturas em código
      - Se forem mais complexas, tais como especificação de um protocolo ou um conjunto de transições de estado o esforço de mapeamento será maior
    - Configurações:
      - As mesmas interações e topologias da configuração devem estar preservadas na implementação
      - Muitas linguagens de programação permitem que um módulo se refira a outro somente através da sua interface
      - Reflexão computacional e descoberta dinâmica de serviços

Conceitos

- O problema do mapeamento
  - Mapeamento de diferentes tipos de decisões arquiteturais:
    - *Rationale*:
      - Geralmente não é especificamente mapeado na implementação
      - Geralmente é capturado através de comentários em código-fonte ou documentação externa
    - Propriedades dinâmicas (comportamentais):
      - Algumas especificações comportamentais podem ser traduzidas diretamente em arcabouços de código ou até implementações completas
      - Exceção: especificações comportamentais formais (mais adequadas para análises e testes do que implementação)

Conceitos

- O problema do mapeamento
  - Mapeamento de diferentes tipos de decisões arquiteturais:
    - Propriedades não-funcionais:
      - Uma das tarefas mais difíceis
      - Isso faz com que o refinamento de propriedades não-funcionais em decisões funcionais se torne muito importante
      - Geralmente obtida por uma combinação de técnicas:
        - Documentação do *rationale*
        - Inspeções
        - Testes
        - Estudos de caso

Conceitos

- O problema do mapeamento
  - Mapeamento *one-way* e *round-trip*
    - Arquiteturas e implementações evoluem em conjunto
    - Manter a arquitetura e a implementação sincronizados é um problema desafiador
    - Aspectos deste mapeamento que não possuem *traceability* são frequentemente os primeiros a divergir
    - Opção 1: exigir que a arquitetura seja modificada primeiro (mapeamento *one-way*)
    - Opção 2: permitir que mudanças sejam iniciadas tanto na arquitetura quanto na implementação (mapeamento *two-way*)
      - Útil para detectar e resolver degradações porém é mais complexo e mais caro

Conceitos

- *Architecture Implementation Frameworks* (AIFs)
  - O cenário ideal é projetar a arquitetura primeiro e então selecionar tecnologias de implementação que melhor atendem às suas necessidades
    - É difícil, pois as linguagens de programação raramente suportam explicitamente construtores do nível arquitetural
    - Tecnologias são geralmente ditadas por fatores extrínsecos ou acidentais tais como custo, maturidade, suporte disponível, cultura organizacional e requisitios externamente impostos
  - Uma abordagem interessante é usar ou desenvolver um *Architecture Implementation Framework*

Conceitos

- *Architecture Implementation Frameworks* (AIFs)

> ***Architecture Implementation Framework*:** *software* que atua como uma ponte entre um estilo arquitetural particular e um conjunto de tecnologias de implementação. Tal solução disponibiliza, em código, elementos chave do estilo arquitetural de uma forma que ajuda os desenvolvedores na implementação de sistemas em conformidade com as prescrições e restrições do estilo

- Exemplo:
  - *UNIX Standard I/O Library*
    - Mapeia o estilo *pipe-and-filter* em linguagens procedurais e não-concorrentes tais como C
- Entretanto, o *AIF* pode não impedir que o desenvolvedor viole as restrições do estilo

Conceitos

- *Architecture Implementation Frameworks* (AIFs)
  - Como estes *AIFs* são representados nos modelos arquiteturais ?
    - São considerados como um substrato subjacente aos componentes e conectores
    - Portanto não são modelados explicitamente como um componente ou conector da arquitetura
    - Entretanto, frequentemente disponibilizam implementações dos componentes e conectores mais comuns do estilo
  - O mesmo estilo pode ser suportado por diferentes *AIFs*
    - Podem variar em diversas dimensões de qualidade

Conceitos

- Avaliando *Architectural Implementation Frameworks*
  - Suporte a Plataforma
    - Um *AIF* liga três elementos: um estilo arquitetural, uma linguagem de programação e um sistema operacional
    - A disponibilidade de um *AIF* para o cenário em questão é fator primordial da implementação
  - Fidelidade:
    - Um *AIF* não precisa implementar **todos** os aspectos do estilo. Ex: pode implementar aspectos de comunicação mas deixar aspectos de concorrência para os desenvolvedores
    - Geralmente disponibilizam suporte mas não garantia a restrições estilísticas
    - Quais mais fiel menor a degração e maior o custo

Conceitos

- Avaliando *Architectural Implementation Frameworks*
  - Hipóteses Compatíveis
    - Em um cenário ideal as decisões e restrições induzidas pelo *AIF* são as mesmas do estilo arquitetural
    - Entretanto, *AIFs* geralmente introduzem restrições adicionais:
      - O sistema será instanciado e configurado somente pelo *AIF*
      - Componentes e conectores individuais não possuem *thread* própria
      - Cada componente da arquitetura deve ser associado a um módulo da linguagem de programação
    - Ex: tanto o *AIF* quando um *toolkit* para *GUIs* exigem que os elementos sejam derivados de uma classe base
      - Se a linguagem não suporta herança múltipla então existe uma incompatilidade entre os *frameworks* mesmo sem constituir uma incompatibilidade arquitetural

Conceitos

- Avaliando *Architectural Implementation Frameworks*
  - Eficiência
    - *AIFs* geralmente adicionam uma camada de funcionalidade entre a aplicação e o *hardware*
    - Um principal perigo é uma diminuição da eficiência, pois geralmente o *AIF* permeia toda a aplicação e media a comunicação, por exemplo ditando a política de concorrência do sistema
    - Como avaliar ?
      - *Benchmarks* executados com parâmetros representativos da aplicação alvo
      - Ex: se um *AIF* troca no máximo 10.000 mensagens por minuto em uma aplicação *dummy* somente para este propósito não faz sentido utilizar este *AIF* para uma aplicação real que troca 20.000 mensagens por minuto

Conceitos

- Avaliando *Architectural Implementation Frameworks*
  - Outras Considerações
    - Tamanho
    - Custo
    - Facilidade de Uso
    - Disponibilidade de Código-Fonte
    - Confiabilidade
    - Robustez
    - Portabilidade

Conceitos

- *Middleware*, Modelos de Componentes e *Application Frameworks*
  - Tecnologias para integração de *software* e disponibilização de serviços acima e além daqueles oferecidos pela linguagem de programação / sistema operacional
  - Exemplos: CORBA, JavaBeans, COM/DCOM/COM+, .NET, JMS, tecnologias de *web services*, etc
  - *Architecture Implementation Frameworks* são uma forma de *middleware*:
    - *AIFs* têm foco no estilo arquitetural
    - Soluções de *middleware* têm foco nos serviços

Conceitos

- *Middleware*, Modelos de Componentes e *Application Frameworks*
  - Soluções de *middleware* frequentemente adicionam restrições aos sistemas implementados
    - Exemplo: CORBA/COM/RMI:
      - Aplicação dividida em objetos executados em diferentes máquinas
      - Objetos expõem seus serviços utilizando IDL (*Interface Definition Language*)
      - Objetos descobrem outros objetos a partir de um *name/trade service*
      - Objetos se comunicam no estilo *request-response* passando somente parâmetros serializáveis
  - É preciso ter cuidado para que o *middleware* não influencie demasiadamente o projeto arquitetural

Conceitos

- *Middleware*, Modelos de Componentes e *Application Frameworks*
  - Principais conflitos entre estilos arquiteturais e soluções de *middleware*:
    - O estilo arquitetural não é compatível com o estilo induzido pelo *middleware* escolhido
    - O *middleware* é escolhido primeiro, baseado nos serviços disponibilizados, e permite-se que isto influencie sobremaneira o estilo arquitetural da aplicação

Conceitos

- *Middleware*, Modelos de Componentes e *Application Frameworks*
  - Resolvendo estas incompatibilidades:
    - Mudando o estilo: somente deve ser feito quando os benefícios de uso do *middleware* superam o custo de adaptação do *middleware* para acomodar o estilo
    - Mudando o *middleware*: pode ser difícil porque soluções de *middleware* geralmente são grandes, complexas e proprietárias
    - Desenvolvendo um *glue code*: cria-se um *AIF* sobre o *middleware*, alavancando as compatibilidades e resolvendo as incompatibilidades

Conceitos

- *Middleware*, Modelos de Componentes e *Application Frameworks*
  - Resolvendo estas incompatibilidades:
    - Ignorando os serviços do *middleware* não utilizados
    - Ocultando o *middleware*: se o *middleware* não é extensivamente utilizado na aplicação pode-se encapsulá-lo em um componente ou conector
  - Soluções de *middleware* são amplamente utilizadas na implementação de conectores

Conceitos

- Criando um novo *Architecture Implementation Framework*:
  - Motivação:
    - O estilo arquitetural em uso é inédito
    - O estilo arquitetural não é inédito mas está sendo implementado em uma plataforma para a qual um *AIF* não existe
    - O estilo arquitetural não é inédito, um *AIF* já existe para a plataforma em questão, porém é inadequado

Conceitos

- Criando um novo *Architecture Implementation Framework*:
  - Desenvolver um novo *AIF* é uma tarefa complexa que requer:
    - Amplo conhecimento do estilo arquitetural
    - Limitação do trabalho do *AIF* a problemas relacionados ao estilo arquitetural
    - Escolha do escopo do *AIF*
    - Cuidado com *over-engineering*
    - Diminuição do *overhead* gerado ao desenvolvedor
    - Desenvolvimento de uma estratégia para integrar recursos legados e *COTS*

Conceitos

- Concorrência:
  - Geralmente implementada com diversas políticas
  - Muitos estilos arquiteturais possuem prescrições específicas para concorrência
  - Muitos *AIF* e soluções de *middleware* realizam gerenciamento de concorrência
  - Entretanto, ainda é difícil desenvolvedor sistemas concorrentes
  - Encapsular a implementação de aspectos de concorrência em um *middleware* ou *framework* de qualidade pode ajudar a reduzir os riscos de *deadlocks* e condições de corrida

Conceitos

- Tecnologias Generativas:
  - É uma estratégia atrativa para manter o mapeamento entre a arquitetura e a implementação
  - Abordagens:
    - Geração de implementações completas do sistema ou de elementos
      - Extremamente difícil na prática
    - Geração de arcabouços ou interfaces
      - Geralmente interfaces e comportamento baseado em diagramas de transição de estados
    - Geração de composições *(assemblies)*
      - Sistemas construídos a partir de bibliotecas e conectores reutilizáveis
      - Típico em engenharia de *software* de domínio específico

Conceitos

- Garantindo consistência arquitetura-implementação
  - Mesmo utilizando um *AIF* a implementação pode não estar em conformidade com a arquitetura prescritiva
  - Estratégias para detecção de consistência:
    - Criação e manutenção de *Traceability Links*
    - Inclusão do Modelo Arquitetural como um artefato de implementação
    - Geração da implementação a partir da arquitetura

INF016 – Arquitetura de Software

08 - Implementação

Sandro Santos Andrade
sandroandrade@ifba.edu.br

Instituto Federal de Educação, Ciência e Tecnologia da Bahia
Departamento de Tecnologia Eletro-Eletrônica
Graduação Tecnológica em Análise e Desenvolvimento de Sistemas