Modelagem de Arquitetura com UML

Arquitetura de *software*

- "Uma arquitetura de *software* deve conter: a definição dos **elementos de projeto** que compõem o *software*; a descrição das **interações** entre estes elementos; os **padrões de composição** dos elementos; e um **conjunto de restrições** sobre estes padrões." [Shaw 96]

Arquitetura de *software*

- Modelo de Shaw, 1996
  - componentes
  - conectores
  - configuração

Arquitetura de *software*

- Componente
  - modela a computação e/ou armazenamento de informações
  - desenvolvido independentemente
  - exemplos de componentes
    - ➔cliente
    - ➔servidor
    - ➔banco de dados
    - ➔aplicação inteira

Arquitetura de *software*

- Conector
  - modela as interações entre os componentes
  - desenvolvido independentemente
  - exemplos de conectores
    - ➔protocolos de comunicação
    - ➔Middleware (ex: CORBA, RMI)

Arquitetura de *software*

- Configuração
  - topologia
  - conjunto de componentes combinados usando-se os conectores
  - grafo de componentes e conectores ligados, descrevendo uma estrutura arquitetural

Arquitetura de *software*

- Principais motivações para definição de uma Arquitetura de Software:
  - reuso de elementos de projeto permitindo maior rapidez na construção do *software;*
  - facilita a comunicação entre os *stakeholders* do sistema.

Uso de UML como uma "ADL" (Architecture Description Language)

- Mesmo havendo uma "perda" em capturar todos os aspectos de uma arquitetura, o uso de UML tem sido crescente
- Vantagens:
  - Uso de uma notação amplamente conhecida em "oposição" às ADLs
  - Notação acessível para arquitetos, desenvolvedores e gerentes
  - Permite ilustrar múltiplas visões

Múltiplas Visões

- Vários stakeholders têm diferentes conceitos e interesses em aspectos distintos da arquitetura.
- Assim como na construção civil, diferentes tipos de “plantas” são usadas para representar diferentes aspectos da arquitetura.

Múltiplas Visões

- De forma similar, na arquitetura de *software* pode-se usar várias “plantas” para diversos propósitos:
  - Para organizar a funcionalidade do sistema em termos de elementos do domínio;
  - Para tratar da decomposição lógica do sistema;
  - Para tratar aspectos de concorrência (em tempo de execução);
  - Para tratar do mapeamento de módulos e interfaces para arquivos fontes.

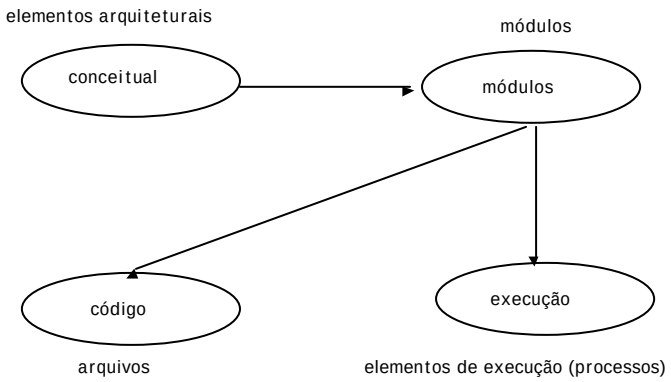
Múltiplas Visões

- Diferentes visões tratam de aspectos distintos durante o desenvolvimento de um sistema. Exemplo de abordagens:
  - Kructhen - O Modelo 4 + 1: Lógica, Processo, Física, Implantação e Casos de Uso.

Múltiplas Visões

- Soni, Nord e Hofmeister: Consideram a representação da arquitetura de software em 4 visões:
  - ➔visão conceitual
  - ➔visão de módulo
  - ➔visão de execução
  - ➔visão de código

Múltiplas Visões: Relação entre as visões



Visão Conceitual

- Captura aspectos funcionais do sistema
- Stakeholder interessado: Usuário final
- Descrição da arquitetura em termos de elementos arquiteturais
- Elementos: componentes, portas e conectores

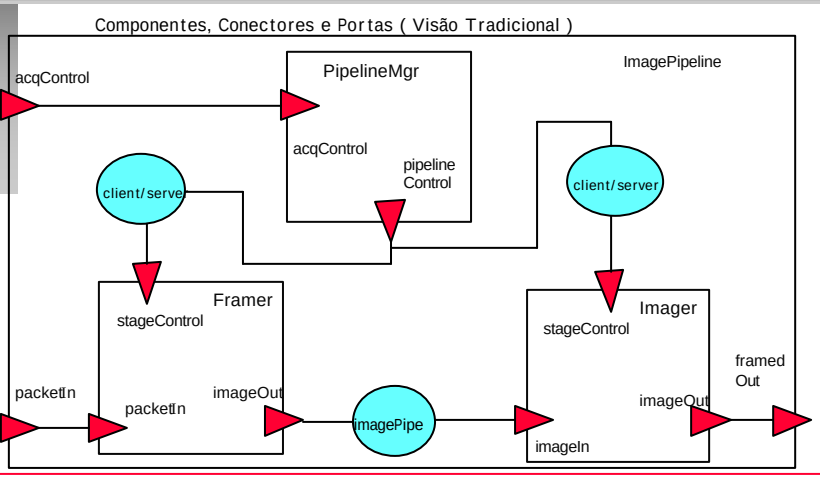
Visão Conceitual

- Componentes, portas e conectores são modelados através do diagrama de classes de UML (“*stereotyped class*”)
- Interação entre componentes através de Diagrama de Seqüência

Visão Conceitual - Exemplo

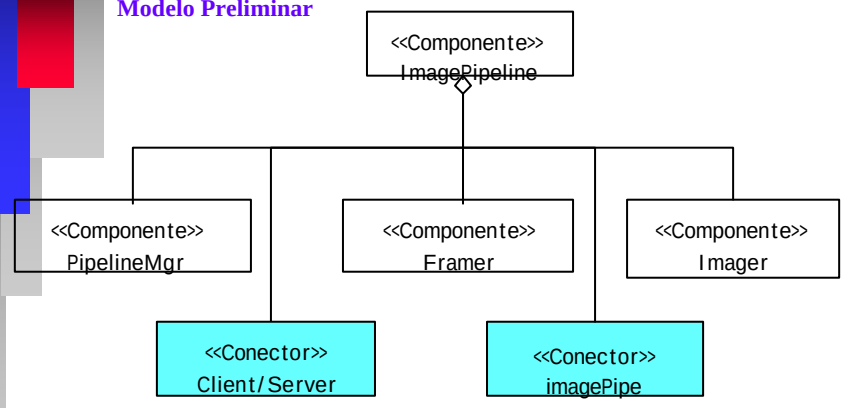
- O **sistema de captura de imagem** capta um conjunto de imagens digitalizadas. O **usuário controla** a captura pela seleção de um procedimento a partir de um **conjunto de procedimentos pré-definidos**. Então, inicia tal procedimento e provavelmente ajusta-o durante a captura. O dado bruto para as imagens é capturado por um **dispositivo de hardware**, uma “câmera”, sendo então enviado para o processador de imagem (***image pipeline***) onde é convertido para imagens. O *image pipeline* faz esta conversão, **primeiro compondo o dado bruto em imagens discretas**, e então executa um ou mais **padrões de transformações** de imagem para melhorar a resolução das imagens resultantes.

Visão Conceitual – modelo informal

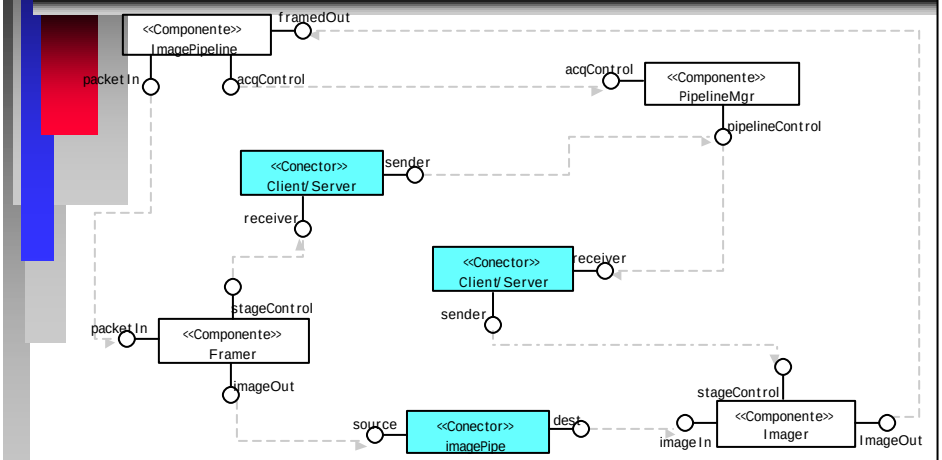


Visão Conceitual - Diagrama de Classes UML

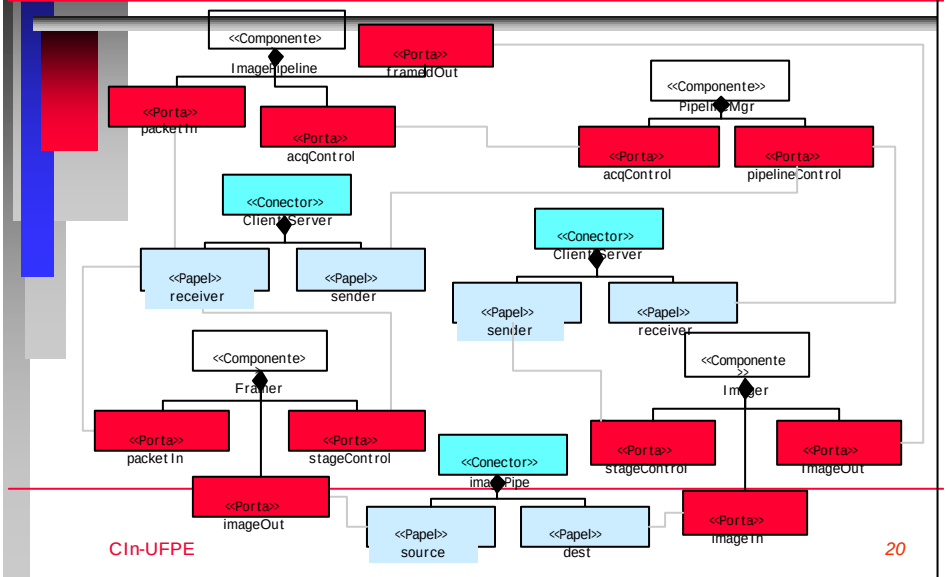
Modelo Preliminar



Visão Conceitual - Diagrama de Classes UML



Visão Conceitual - Diagrama de Classes UML



Visão de Módulo

- Subsistemas são decompostos em módulos os quais são organizados em camadas
- Stakeholders interessados: Analistas e Programadores
- Elementos: módulos, subsistemas e camadas
- Elementos conceituais são mapeados em módulos

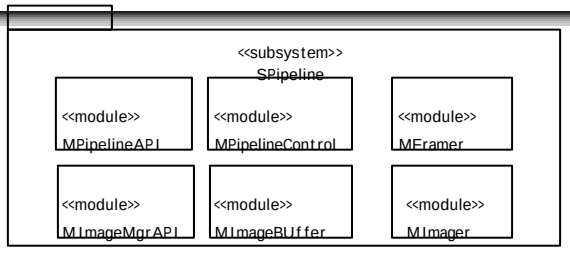
Visão de Módulo

- Componentes são “implementados” por módulos e subsistemas
- Portas, conectores e componentes podem ser combinados em um único módulo
- Dependências de uso entre módulos são derivadas das associações entre os elementos conceituais

Visão de Módulo

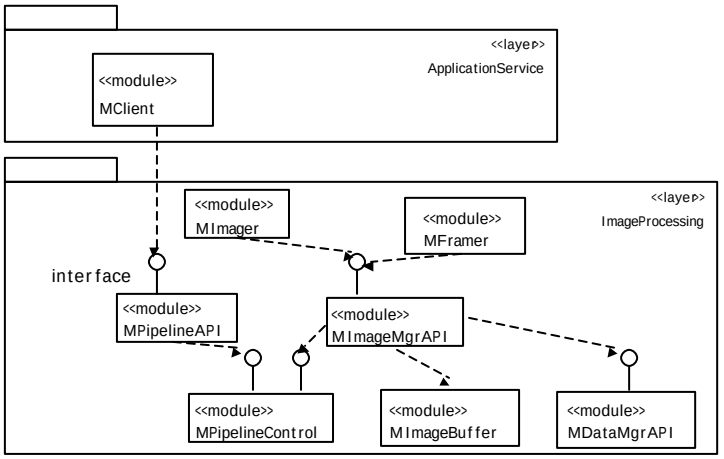
- Módulos são representados por classes estereotipadas
- Subsistemas e camadas são representados por pacotes estereotipados
- Dependências de uso são representadas por dependências UML

Visão de Módulo



Elemento Conceitual	Subsistema ou Módulo
ImagePipeline (cp)	SPipeline
acqControl(pt),pipelineControl(pt)	MpipelineAPI
pipelineMgr(cp), ImagePipe(cn),Client/ Server (cn)	MpipelineControl, MImageBuffer
stageControl(pt), imageIn(pt), imageOut(pt)	MImageMgrAPI
Framer (cp)	MFramer
Imager(cp)	MImager

Visão de Módulo - Dependências de Uso



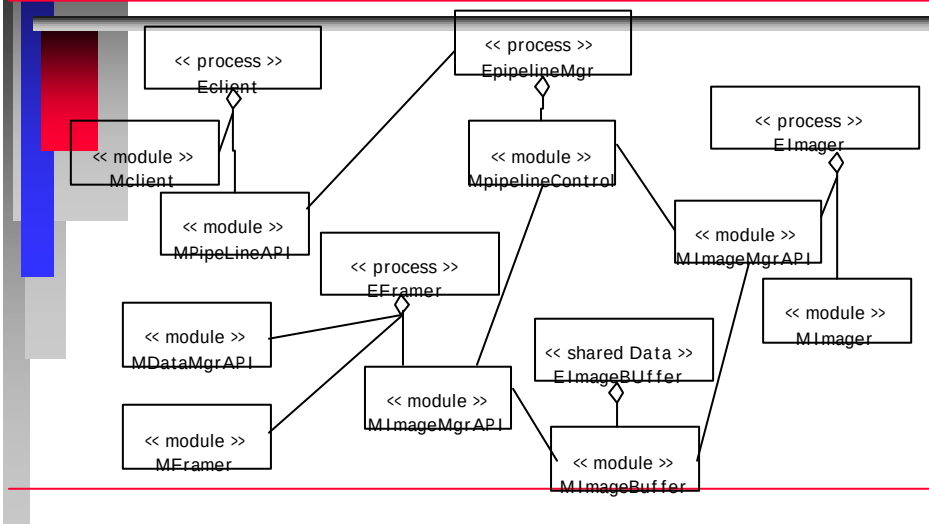
Visão de Execução

- Visão do sistema em tempo de execução (TE)
- Stakeholder interessado: Integradores do sistema
- Define o mapeamento dos módulos em elementos de TE (processos, *threads*, etc)
- Define a comunicação entre os elementos de TE, e os atribui a recursos físicos

Visão de Execução

- Elementos: cenário de tempo de execução e caminho de execução
- Conceitos principais: Uso de recursos e performance
- Classes UML estereotipadas são usadas para representar elementos de TE
- Estereótipos de acordo com o nome do elementos da plataforma: <<process>> <<shared data>>

Visão de Execução



Visão de Código

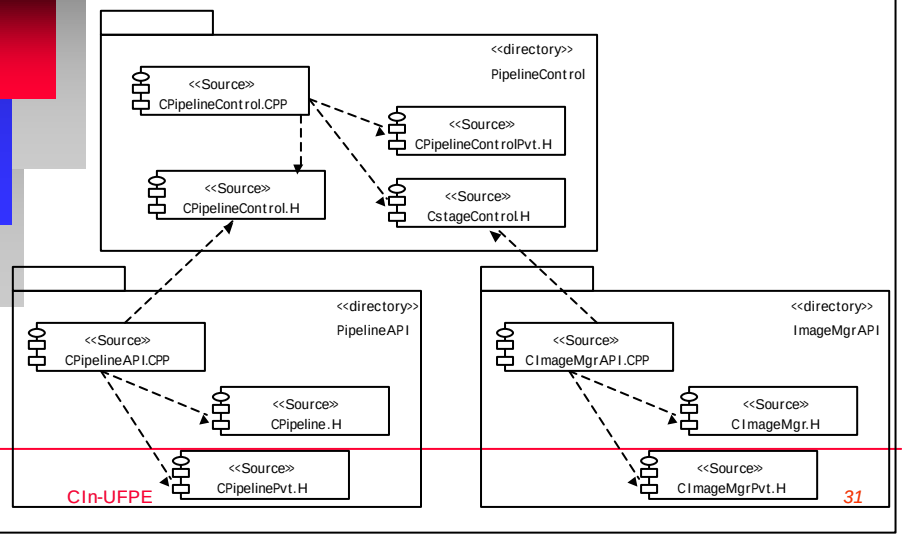
- Contém arquivos e diretórios
- Stakeholder interessado: Engenheiros de Sistema (topologia, entrega, instalação, etc)
- Mapeamento dos módulos e interfaces da visão de módulo em arquivos fonte
- Elementos de TE da visão de execução são mapeados em arquivos executáveis

Visão de Código

- Elementos: código fonte, código intermediário, código executável, diretório
- Arquivos são representados por componentes UML estereotipados
- Diretórios são representados por pacotes UML estereotipados

Visão de Código

Dependência entre arquivos fonte



Conclusão

- Arquitetura de Software promove o reuso em alta escala
- A UML pode ser usada como uma “ADL” (apresenta algumas deficiências). Existem propostas para sua extensão.
- A grande motivação do uso da UML é que ela é um padrão largamente aceito e difundido