

# **ArcADe: Um Modelo de Processo para Análise e Projeto Baseado em Arquitetura de Software**

*Marco Antônio Fagundes de Moraes*  
*Secretaria de Informática do Tribunal Regional*  
*Eleitoral do Pará - TRE/Pa*  
*mfagunde@tre-pa.gov.br*

*Alexandre Marcos Lins de Vasconcelos*  
*Centro de Informática da Universidade Federal*  
*de Pernambuco – CIn/UFPE*  
*amlv@cin.ufpe.br*

## **Resumo**

*Reuso de software tem recebido uma crescente atenção como alternativa para o aumento de produtividade e redução do custo de desenvolvimento nas organizações produtoras de software. Nesse contexto, a arquitetura de Software (AS) merece ser destacada, pois permite reuso em níveis mais abstratos. Entretanto, o tratamento explícito de AS não tem sido o foco dos processos de software largamente utilizados, devido a algumas razões: terminologia específica (componentes, conectores e configuração); AS é uma disciplina emergente; pouco suporte de ferramentas disponíveis. Neste artigo, apresentamos o processo ArcADe (Software Architecture-based Analysis and Design) que integra conceitos e padrões largamente difundidos com a AS. Este processo teve uma influência do RUP (Rational Unified Process) e trata da relação entre os requisitos e as abstrações arquiteturais, elaboração, representação e concretização da arquitetura.*

## **1. Introdução.**

O reuso de software [9] tem recebido uma crescente atenção como uma alternativa para as organizações reduzirem tempo e custos envolvidos no processo de construção de software. Nesse contexto, diversas abordagens que tratam do reuso têm sido definidas (e.g., Arquitetura de Software - AS [16], *Middleware* [4], Padrões de Projeto [6], Desenvolvimento Baseado em Componentes [1], etc.). Dentre essas abordagens a AS merece destaque, pois permite reuso em níveis bem abstratos, ao mesmo tempo, que fornece um potencial para a aplicação das demais abordagens. Por exemplo, considerando que a AS fornece uma descrição do software em termos de componentes, conectores e configuração, podem-se utilizar Padrões de Projeto e *Middleware* na concretização de componentes e conectores respectivamente.

Processos de software largamente utilizados (e.g., *Rational Unified Software Process* – RUP [10]) não favorecem o projeto arquitetural segundo a disciplina de AS [16], pois tratam da arquitetura seguindo uma abordagem particular. Ou seja, não existe uma integração natural entre a disciplina de AS e os processos amplamente utilizados. Essa ausência de integração pode ser entendida por algumas razões: AS é uma disciplina recente; AS não é apoiada pela maioria das ferramentas disponíveis; AS enfatiza o projeto da “computação” e da “comunicação”, e não somente da “computação” como geralmente acontece; AS é descrita através de componentes, conectores e configuração, e não apenas através de diagrama de classes como geralmente é feito em vários projetos orientados a objetos.

Para tratar do problema da ausência de integração da AS com processos largamente utilizados, algumas abordagens têm sido propostas, como por exemplo: utilizar UML (*Unified Modeling Language*) como ADL (*Architecture Description Language*) [7], *Catalysis* [3] e MDA (*Model-Driven Architecture*) [11]. A primeira abordagem trata da representação da arquitetura através de UML, em substituição às ADLs tradicionais que, em sua maioria, apresentam um certo rigor formal e sintaxes complexas. O processo *Catalysis* envolve todo o ciclo de desenvolvimento, trata explicitamente de componentes e conectores, e utiliza a UML para construção de seus modelos. O MDA utiliza UML e enfatiza a construção de modelos independentes de plataforma e tecnologia de implementação, com posterior transformação desses modelos em arquiteturas de software específicas para determinadas plataformas.

Considerando as abordagens mencionadas acima, dois pontos devem ser levados em consideração. Primeiro, as abordagens que utilizam UML para descrever a não envolvem todos os aspectos de projeto da arquitetura (e.g., Desenvolver ou Selecionar uma Arquitetura), pois tratam somente da atividade de representação da arquitetura (ver Subseção 2.2.2 – as atividades do desenvolvimento baseado em arquitetura). Segundo, *Catalysis* e MDA não são, por enquanto, amplamente difundidos em ambientes comerciais de desenvolvimento de software.

Com base no contexto descrito, selecionamos um processo de software largamente utilizado, no caso o RUP, e propomos um modelo de processo<sup>1</sup> para análise e projeto que integra os elementos do RUP (conceitos, características e padrões) com a AS. Esse modelo, chamado *ArcADe* [13] (*software Architecture-based Analysis and Design*), adota os conceitos e princípios da AS como base, e utiliza os elementos do RUP para organizar e representar o seu fluxo de trabalho. Devido ao nosso modelo adotar padrões largamente utilizados (e.g., UML), acreditamos que sua aceitação será facilitada.

Além desta seção introdutória, este artigo está organizado da seguinte forma: a Seção 2 introduz os conceitos básicos da nossa proposta. A Seção 3 apresenta a estrutura do *ArcADe*. A Seção 4 apresenta um estudo de caso, onde o modelo é aplicado no projeto de um sistema de software. Finalmente, a Seção 5 apresenta uma análise do modelo, conclusões e algumas direções para trabalhos futuros.

## 2. Escopo e Definições.

Os conceitos e princípios básicos de AS têm um papel importante no modelo proposto. Elementos arquiteturais, tais como, componentes, conectores e configurações descrevem a arquitetura do software no *ArcADe*. As subseções seguintes apresentam o escopo e os conceitos básicos usados em nossa proposta.

### 2.1. Escopo e Uso do Modelo

O modelo proposto é aplicável a qualquer organização de desenvolvimento de software, que deseje alcançar níveis bem abstratos de reuso e melhorar suas capacidades de desenvolvimento, evolução, operação e suporte do software. O *ArcADe* não presume estrutura organizacional particular ou filosofias de gerenciamento. Por outro lado, ele considera um modelo de desenvolvimento iterativo e incremental, organizando as atividades de análise e projeto, de modo a auxiliar os profissionais envolvidos no entendimento e utilização da disciplina de AS durante o desenvolvimento de software. Em relação ao uso do modelo destacamos os aspectos listados na Tabela 1, apresentada a seguir.

**Tabela 1** Uso do modelo

Quem?	Analistas, Projetistas e Arquitetos de Software
Por que?	Para se obter os benefícios inerentes a AS (e.g., clara estruturação do software, reuso em níveis bem abstratos, facilidade do gerenciamento da complexidade pela decomposição do problema).
Como?	Considerando um cenário de utilização (ver Seção 4), seguindo o fluxo de trabalho definido no modelo e executando suas atividades conforme seus guias passo-a-passo.
Quando?	Durante o mapeamento dos requisitos para a arquitetura, em conjunto com a realização das abstrações arquiteturais para obtenção de um projeto detalhado a ser submetido à implementação.

O *ArcADe* pode ser utilizado em projetos de software grandes e complexos (e.g., sistemas com diferentes plataformas, comunicação com legados, múltiplas linguagens de programação). Podendo também ser adaptável a projetos de tamanho e complexidade reduzidos. Adicionalmente, o modelo possibilita sua integração em um processo já existente na organização, desde que os artefatos (o conceito de artefato é apresentado na Subseção 2.2.3) de entrada e saída do modelo sejam contemplados pelo processo considerado.

<sup>1</sup> Segundo a ISO 15504 [8] Modelo define os objetivos em alto nível de abstração; Processo é a definição operacional de um conjunto de atividades para alcançar um propósito específico

## 2.2. Termos e definições

Nesta subseção, apresentamos os conceitos básicos de AS, o desenvolvimento baseado em arquitetura, uma visão geral do RUP e analisamos como esse processo trata da arquitetura.

**2.2.1. Arquitetura de software.** Existem diversas definições de AS (e.g.[2,15]), dentre as quais a de Shaw & Garlan [16] é a mais tradicionalmente adotada pela comunidade da área de AS. Essa definição foi utilizada como base do modelo proposto e considera que AS fornece a descrição mais abstrata do software, envolvendo sua estrutura, comportamento e propriedades chave (e.g., portabilidade). Em um processo de software, a AS desempenha um papel importante como ponte entre os requisitos e a implementação (ver Figura 1.a).

Arquiteturas de software são geralmente descritas em termos de três abstrações básicas: componentes, conectores e configuração. Componentes modelam os elementos que realizam processamento ou armazenam informações (e.g., clientes e servidores), comunicando-se com o ambiente externo através de uma interface (conjunto de portas). Conectores modelam as interações e as regras de comunicação entre os componentes. Configuração representa a interligação de componentes e conectores em uma topologia específica. A Figura 1.b apresenta uma visão geral da AS.

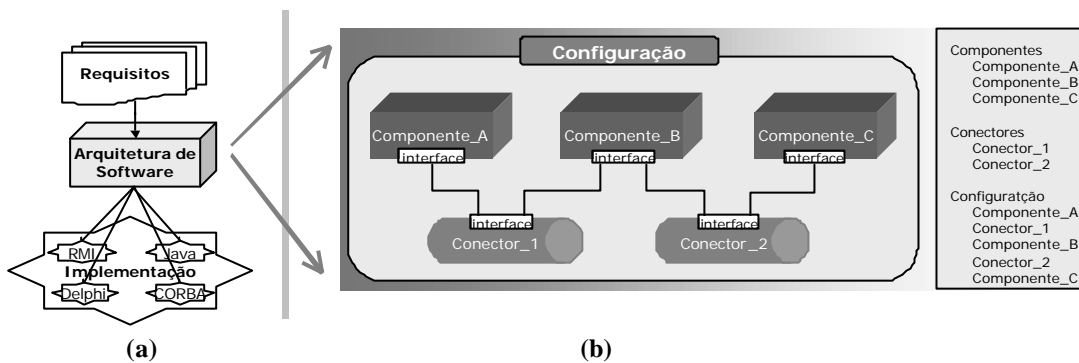


Figura 1 (a) AS como ponte entre requisitos e implementação (b) Visão geral da AS

Essencialmente, a AS apresenta uma descrição do software onde a “computação” (incluída em componentes) e a “comunicação” (embutida na noção de conectores) são claramente separadas [15]. Essa separação reduz o acoplamento entre as partes de um sistema, aumentando as oportunidades de reuso de componentes e conectores em outros contextos.

**2.2.2. Desenvolvimento Baseado em Arquitetura (DBA).** A adoção dos princípios da AS no desenvolvimento de software implica na realização de cinco atividades principais, como proposto por Bass et al. [2]:

1. Entender o domínio dos requisitos - recomenda-se efetuar uma análise do domínio, na qual similaridades e variações são antecipadas, enumeradas e registradas.
2. Desenvolver ou selecionar a arquitetura - após a análise dos requisitos, um estilo arquitetural particular pode ser adotado (seleção) ou a arquitetura do software é definida a partir do zero (desenvolvimento).
3. Representar a arquitetura - nessa atividade utiliza-se uma notação que precisamente descreva a arquitetura do software. Essa notação pode ter uma base formal que permita simular, verificar as propriedades ou ainda construir um protótipo do software.
4. Analisar ou avaliar a arquitetura - de posse da representação da arquitetura, é possível verificar se a mesma exibe as propriedades desejadas (e.g., reusabilidade).
5. Implementar o sistema baseado na arquitetura - é o último passo para a realização da arquitetura do software em elementos concretos de implementação (nível de código).

**2.2.3. Rational Unified Process (RUP).** O RUP [10] é um processo genérico para desenvolvimento de software desenvolvido pela *Rational Software Corporation*, cujas principais características são: orientado a objetos (utilizando UML para construção de seus modelos), dirigido por casos de uso, centrado na arquitetura (o RUP segue uma visão particular de arquitetura – ver Subseção 2.3) e desenvolvimento iterativo e incremental.

Esse processo possui cinco conceitos básicos: responsável, artefato, atividade, fluxo e subfluxo. Responsável é um papel que pode ser atribuído a uma pessoa ou grupo. Artefato é uma informação produzida, modificada ou usada. Atividade é uma unidade de trabalho que pode ser executada por um responsável. Fluxo é uma seqüência de atividades que são agrupadas em etapas do desenvolvimento (e.g., fluxo de requisitos). Subfluxo é usado para estruturar um fluxo em termos de atividades, responsáveis, artefatos de entrada e saída.

O RUP fornece modelos para cada artefato e guias (práticas e técnicas) para executar as atividades. A seguir, analisamos como o RUP trata da arquitetura em relação à definição de AS (ver Subseção 2.2.1).

### 2.3. Análise do tratamento da arquitetura no RUP.

Para analisar como o RUP trata da arquitetura, inicialmente efetuamos um paralelo entre os fluxos de processo do RUP e o DBA (ver Figura 2.a) para identificarmos as semelhanças e diferenças no tratamento da arquitetura nesses dois processos. Em seguida, analisamos e verificamos o nível de integração da AS com o RUP. Nessa verificação, observamos que o RUP adota o Modelo de Visão 4+1 [10] (ver Figura 2.b) para tratar a arquitetura.

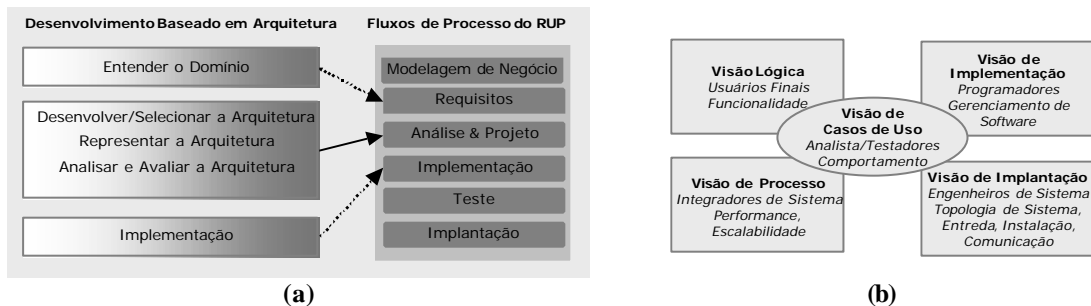


Figura 2 (a) Relação entre o DBA e fluxos de processo do RUP (b) O modelo de visão 4+1

Como resultado da análise do tratamento da arquitetura pelo RUP, verificamos que o mesmo não está em conformidade com a definição apresentada na subseção 2.2.1, no que diz respeito à terminologia usada na AS (componentes, conectores e configuração) e na pouca ênfase à redução do acoplamento. Tal aspecto dificulta uma adaptação/extensão do RUP para tratar a AS, haja visto que a diferença está no conceito base, no caso “arquitetura”. Assim, uma provável adaptação do RUP acarretaria na redefinição/reorganização de todas as atividades envolvidas no tratamento da arquitetura. Entretanto, os benefícios do RUP, tais como, difusão de uso e suporte de ferramentas não devem ser descartados. Com base nesse contexto, formulamos um modelo de processo que integra a AS com elementos do RUP, conforme apresentamos na próxima seção.

### 3. A abordagem ArcADe.

O *ArcADe* (*software Architecture-based Analysis and Design*) é um modelo de processo para análise e projeto que integra a AS com elementos do RUP (conceitos, métodos e técnicas). O modelo adota a AS como base, para definição das etapas do desenvolvimento e utiliza os elementos do RUP, para organizar e representar o processo em um fluxo de trabalho. Neste ponto, destacamos que o nosso modelo não é uma adaptação/extensão do RUP, mas sim, uma integração dos elementos desse processo com a AS.

Vale ressaltar que a integração deve envolver tanto os requisitos funcionais como os não-funcionais (RNFs). Contudo, em nossa proposta, consideramos somente os requisitos funcionais, pois o RUP é dirigido por casos de uso, os quais são empregados somente na descrição dos requisitos funcionais. A adequação do *ArcADe* para tratar os RNFs, pode ser efetuada futuramente usando técnicas propostas em [15], por exemplo.

### 3.1. Níveis de abstração no ArcADe.

O ArcADe segue o modelo de desenvolvimento iterativo e incremental, adotado pelo RUP. Nesse modelo, a idéia básica é compor uma arquitetura abstrata a partir dos requisitos, fornecendo uma descrição de componentes e suas interações em um nível elevado de abstração (ver Figura 3 – Nível 1. Projeto da Arquitetura). Essa arquitetura terá um subconjunto das abstrações arquiteturais (ver componente A e conector 1 na Figura 3) mapeadas para uma arquitetura concreta, fornecendo uma representação arquitetural mais orientada à implementação (ver concretização do componente A e conector 1 na Figura 3 – Nível 2. Projeto Detalhado).

No nível de projeto detalhado, as abstrações arquiteturais (componentes e conectores) serão concretizadas através do uso de estratégias de projeto (e.g., orientação a objetos – Padrões de Projeto) ou da incorporação de produtos existentes (e.g., *Commercial Off-The-Shelf* - COTS [1]). Este processo repete-se (iterativamente) até que a arquitetura global (incremental) tenha sido concretizada em termos de elementos de projeto.

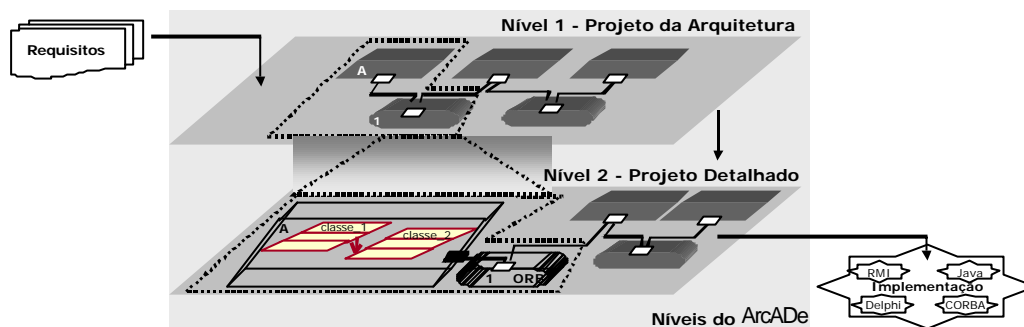


Figura 3 Níveis de abstração do ArcADe

Na próxima subseção apresentamos a organização do nosso modelo conforme os níveis de abstração apresentados na Figura 3.

### 3.2. A organização do modelo.

O RUP especifica atividades, artefatos e guias para tratar a arquitetura (de acordo com o Modelo de Visão 4+1 - ver Subseção 2.3) no fluxo de Análise e Projeto. A partir desse fluxo e dos níveis de abstração apresentados na subseção 3.1, estruturamos o ArcADe em oito subfluxos (ver Figura 4.b): Selecionar uma Arquitetura Candidata; Definir a Arquitetura Abstrata; Refinar a Arquitetura e Analisar Comportamento; Projetar Componentes, Selecionar Componentes, Projetar Conectores e Selecionar Conectores. Neste ponto, destacamos que componentes e conectores são projetados/selecionados explicita e distintamente, devendo ser integrados para compor a arquitetura global do sistema. Essa integração deve acontecer em estágios posteriores do processo (etapa de implementação do sistema). Contudo, devido ao ArcADe tratar da etapa de análise e projeto, o estágio dos requisitos, modelagem de negócio e implementação serão tratados futuramente, conforme mencionado na seção 5.

Conforme mencionamos no início da Seção 3, o RUP forneceu elementos para construção do modelo proposto, um dos quais foi a organização do trabalho em um fluxo de atividades (ver Figura 4.a). As “atividades” apresentadas nos digramas da Figura 4 representam “subfluxos” (o conceito de subfluxo foi apresentado na Subseção 2.2.3). Em certos casos, esses subfluxos possuem o mesmo nome, mas é importante salientar que, devido à diferença no tratamento da arquitetura (ver Subseção 2.3.), a maioria das atividades dos subfluxos do ArcADe (Figura 4.b) foram elaboradas a partir de abordagens baseadas em AS, e somente algumas importadas do RUP, conforme apresentado na próxima subseção.

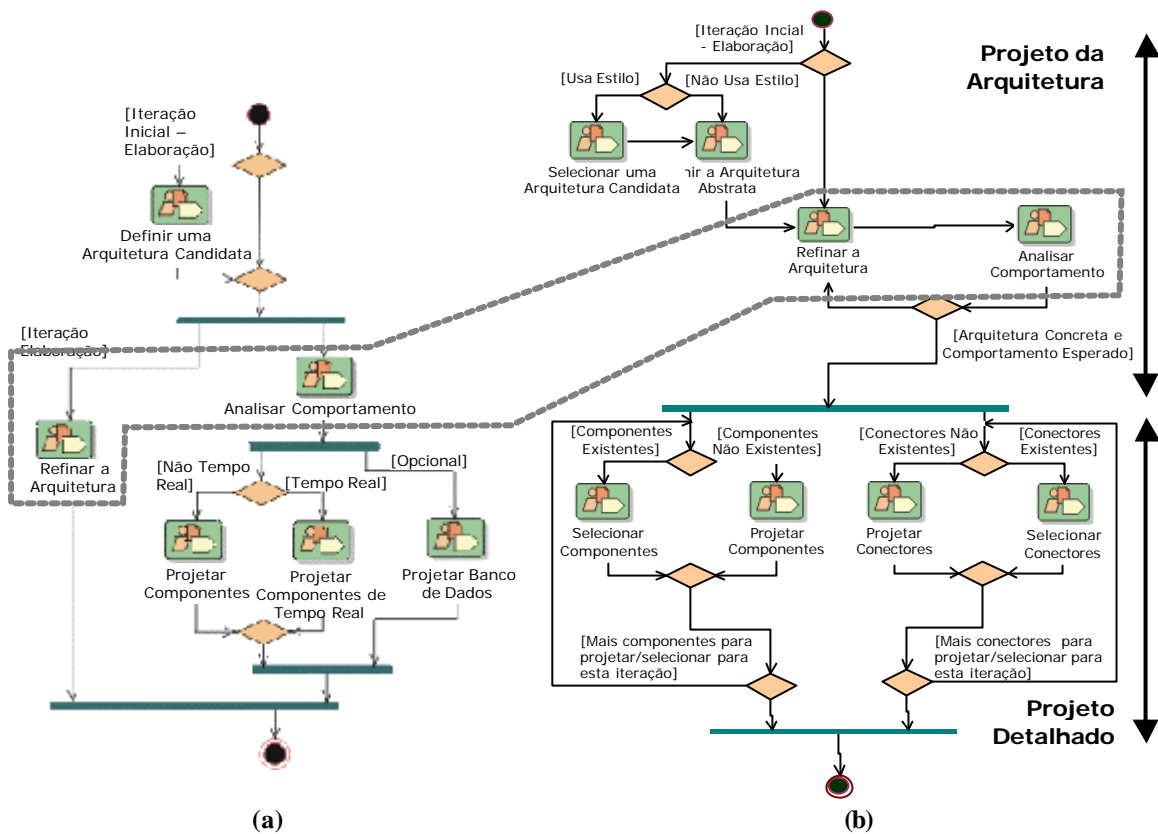


Figura 4 (a) Fluxo de análise e projeto do RUP (b) Modelo ArcADe

Em relação às diferenças na organização dos dois diagramas, destacamos como aspecto principal a execução dos subfluxos Refinar a Arquitetura e Analisar Comportamento (ver destaque da Figura 4). No RUP essa execução é paralela, mas no ArcADe ela é seqüencial. Optamos por essa organização, devido ao refinamento provocar mudanças estruturais na arquitetura, haja vista que elementos arquiteturais (componentes e conectores) podem ser decompostos, agregados ou removidos. Por causa dessa mudança estrutural, deve-se efetuar uma análise de comportamento para verificar se as propriedades estruturais são preservadas após o refinamento. Por exemplo, se dois componentes não se comunicavam na arquitetura abstrata, eles também não devem estar ligados na arquitetura refinada.

### 3.3. O modelo ArcADe.

O ArcADe fornece uma compilação de diversas abordagens baseadas em AS, tais como, [4,5,7], que são organizadas em oito subfluxos de trabalho (ver Figura 4.b). Cada subfluxo é descrito em termos de atividades que, em sua maioria, foram elaboradas a partir de abordagens baseadas em AS. As demais atividades, que não se enquadram nesse contexto, foram importadas do RUP e estão indicadas com (\*) na descrição apresentada nas subseções seguintes.

**3.3.1. Subfluxo “Selecionar uma arquitetura candidata”.** Utiliza a abordagem CBSP [5] para relacionar os elementos do domínio com as abstrações arquiteturais (componentes e conectores), a fim de elaborar um modelo intermediário da arquitetura, que será relacionado com estilos arquiteturais [16] para seleção de uma arquitetura candidata. As atividades desse subfluxo são: Relacionar o Domínio com a Arquitetura e Identificar Oportunidades de Reuso em Nível Arquitetural.

**3.3.2. Subfluxo “Definir a arquitetura abstrata”.** Componentes e conectores do modelo intermediário (produzido no subfluxo anterior) são mapeados no vocabulário da arquitetura candidata e dispostos seguindo as regras de configuração do estilo. Fazendo isto, produz-se um modelo preliminar da arquitetura, que mostra a relação de serviços entre os componentes. Esse modelo deve ser convertido para a representação definitiva da arquitetura do software usando UML. As atividades desse subfluxo são: Mapear Vocabulário do Domínio sobre a Arquitetura Candidata, Modelar Elementos Arquiteturais, Representar a Arquitetura e Analisar a Arquitetura.

**3.3.3. Subfluxo “Refinar a arquitetura”.** A arquitetura abstrata é submetida às regras de refinamento [14] para a obtenção de uma arquitetura em um nível menos abstrato, que servirá de base para a etapa de concretização das abstrações arquiteturais. As atividades desse subfluxo são: Aplicar Regras de Refinamento e Revisar a Arquitetura.

**3.3.4. Subfluxo “Analisar comportamento”.** Devido ao fato de UML não permitir um exame de comportamento rigoroso como o que existe em *Wright* [12], realizamos uma análise limitada através da construção de diagramas de estados para os componentes (abstratos e concretos) envolvidos no refinamento. A partir de uma análise na lista dos estados possíveis nos diagramas construídos, podemos verificar se o comportamento da arquitetura global ainda é preservado após o refinamento. As atividades desse subfluxo são: Modelar Comportamento e Analisar Modelo.

**3.3.5. Subfluxo “Projetar componentes”.** Em um nível mais concreto de projeto, componentes podem ser projetados através da aplicação de Padrões de Projeto, seguindo a abordagem [6] selecionam-se os padrões que forneçam a solução para o problema tratado. Em seguida, detalham-se as propriedades das classes (e.g., atributos) e dos casos de uso envolvidos para definir a estrutura e funcionalidade do componente. Finalmente, utiliza-se a OMG IDL [17] (*Interface Description Language*) para especificar interface do componente. As atividades desse subfluxo são: Efetuar Projeto Detalhado do Componente, Projetar Classes, Projetar Casos de Uso (\*), Especificar a Interface do Componente e Projetar Componente de Dados (\*).

**3.3.6. Subfluxo “Selecionar componentes”.** Uma vantagem do *ArcADe* em relação aos processos utilizados (e.g., RUP) é que ele tanto guia a construção (projeto) quanto apoia a incorporação de produtos existentes (e.g., COTS). Para a incorporação, adotamos o método CRE [1] que identifica os produtos candidatos que atendam um critério de seleção (e.g., Interface do Componente). Esse método indica, através de um *ranking*, qual produto integrará o sistema final. Caso nenhum produto seja selecionado, sugerimos a execução do subfluxo Projetar Componentes. As atividades desse subfluxo são: Identificar, Descrever, Avaliar e Aceitar Produtos Candidatos.

**3.3.7. Subfluxo “Projetar conectores”.** Em alguns casos, os tipos básicos de mecanismos de comunicação existentes (e.g., Remote Procedure Call) não são suficientes para atender as necessidades de conexão entre os componentes, isto é, não podem ser utilizados para implementar os conectores. Nesse contexto, sugere-se uma extensão dos tipos básicos através da abordagem [18] para a produção de um novo conector. As atividades desse subfluxo são: Especificar Partes Concretas do Conector e Estender Tipo Básico.

**3.3.8. Subfluxo “Selecionar conectores”.** Similarmente à reutilização de componentes, conectores podem ser mapeados em tecnologias *middleware* (e.g., ORB). A fim de guiar o processo de seleção de um *middleware* adequado, sugerimos novamente o método CRE [1]. A partir do critério de seleção (e.g., suporte a múltiplas plataformas) identificam-se as tecnologias candidatas. Dentre essas tecnologias o método CRE indica qual será integrada no sistema final. Caso nenhuma tecnologia seja selecionada, sugerimos a execução do subfluxo Projetar Conectores. As atividades desse subfluxo são: Identificar, Descrever, Avaliar e Aceitar Tecnologias Candidatas.

Vale salientar que, cada atividade do modelo proposto é detalhada em passos e artefatos de entrada/saída, conforme apresentado em [13]. Na seção seguinte, ilustramos a execução dos subfluxos do *ArcADe*, considerando um determinado cenário de utilização.

## 4. Um cenário de utilização do *ArcADe*.

Esta seção ilustra a utilização do *ArcADe* em um estudo de caso real, onde as atividades agrupadas nos subfluxos são executadas através de seus guias passo-a-passo. Nesse estudo, usamos uma aplicação desenvolvida em colaboração com Núcleo de Tecnologia da Informação (NTI) da UFPE (Universidade Federal de Pernambuco). Nas próximas subseções, apresentamos a execução do *ArcADe* considerando um determinado cenário e verificamos os benefícios alcançados com sua utilização.

### 4.1. A aplicação SIG@UFPE.

A aplicação SIG@UFPE – Sistema de Informação e Gestão Acadêmica da UFPE possui como uma de suas atribuições a automatização do processo de controle acadêmico em todos os níveis de ensino: graduação, pós-graduação e extensão. Trata de serviços como, matrícula, acompanhamento e encerramento de período. Todas essas operações são realizadas de forma descentralizada e segura através da Web.

Em face à existência de diversos cenários possíveis de execução do *ArcADe* e do tamanho da aplicação SIG@UFPE, ilustramos a utilização do modelo proposto considerando dois aspectos. Primeiro, selecionamos um cenário de execução representativo para a ilustração, envolvendo três elementos: utilização de estilos arquiteturais; uso de componentes inexistentes; e emprego de *middleware* para realizar conectores. Segundo, limitamos o escopo do SIG@UFPE para algumas de suas funcionalidades (ver Subseção 4.2). Contudo, é válido ressaltar que, outros cenários devem ser experimentados para verificação completa do modelo, conforme mencionado na seção 5.

Considerando o cenário acima, seis subfluxos devem ser executados: Selecionar Arquitetura Candidata, Definir Arquitetura Abstrata, Refinar a Arquitetura, Projetar Componentes e Selecionar Conectores. As subseções seguintes apresentam a execução de cada um desses subfluxos.

### 4.2. Artefatos base para execução dos subfluxos.

O *ArcADe* considera como uma das suas principais entradas o documento de requisitos. No sistema SIG@UFPE, o módulo em construção trata do planejamento de matrícula e matrícula, o qual foi utilizado no estudo de caso. Entre os principais requisitos funcionais desse módulo destacam-se: [R01] Definir Atividades Acadêmicas a Ofertar; [R02] Realizar Pré-matrícula; [R03] Definir Atividades Acadêmicas Ofertadas; [R04] Criar Turmas e Subturmas; [R05] Controlar Espaço Físico e Recursos Humanos; [R06] Realizar Matrícula em Atividade Acadêmica; e [R07] Matricular em Turma. Dentre esses requisitos, limitamos o escopo do estudo ao requisito [R04], pois o julgamos como um dos mais representativos devido ao grau de complexidade, número de elementos envolvidos durante sua realização e necessidade de comunicação com outros componentes para cumprir suas responsabilidades.

### 4.3. Execução dos subfluxos do *ArcADe*.

**4.3.1. Subfluxo “Selecionar uma Arquitetura Candidata”.** Inicialmente efetuamos o relacionamento dos requisitos com a arquitetura de software através da abordagem CBSP (*Component-Bus-System-Property*) [5], onde os requisitos ([R01]...[R07]) foram considerados de relevância arquitetural e classificados como componente de processamento (Cp). As dependências entre esses componentes foram obtidas pela necessidade de comunicação entre eles. Por exemplo, na Figura 5.a, o componente Gerenciador de Turmas e Subturmas depende da informação de Espaço Físico (componente de dados – Cd).

Componentes e dependências auxiliam o arquiteto a encontrar um estilo arquitetural adequado. Considerando o domínio da aplicação (Web) o estilo cliente-servidor [16] organizado em três níveis foi selecionado (Figura 5.b). A partir do modelo CBSP e do estilo selecionado, define-se a arquitetura abstrata do sistema específico, conforme apresentado na próxima subseção.

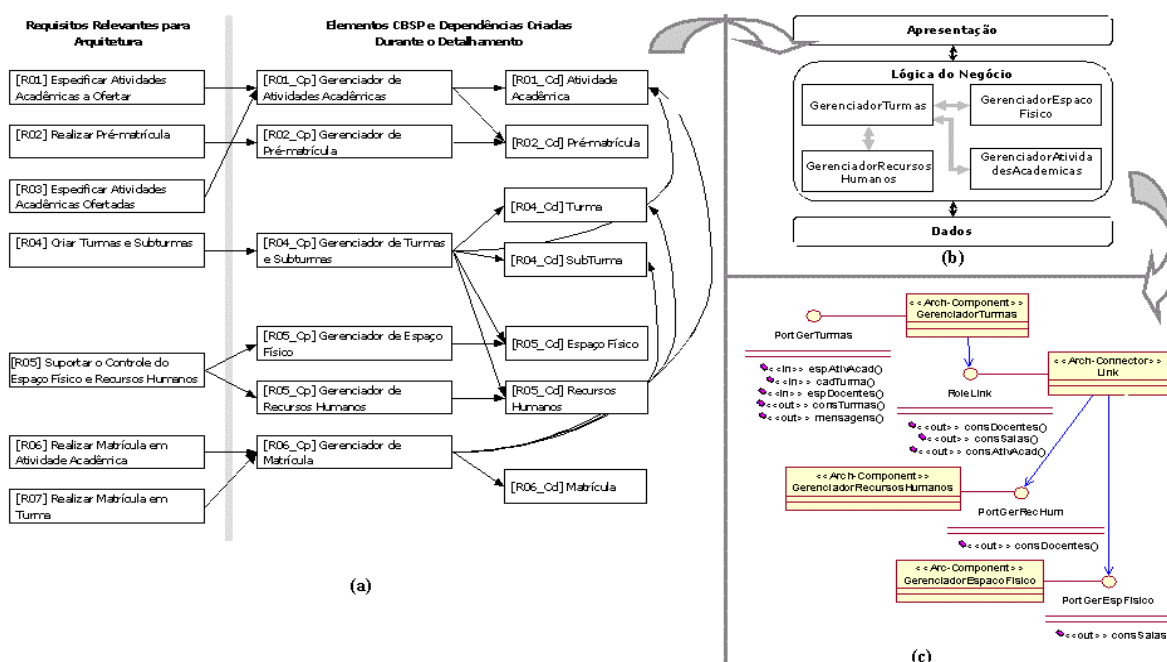
**4.3.2. Subfluxo “Definir a Arquitetura Abstrata”.** Conforme mencionamos no início da Subseção 4.2, limitamos o escopo do estudo de caso ao requisito [R04]. A partir desse requisito e considerando a organização da arquitetura em três níveis, a estrutura do sistema fica da seguinte forma: o componente GUI é dedicado ao nível de apresentação; os componentes Gerenciadores são alocados para o nível de lógica do negócio; e os componentes de dados são dedicados ao nível de dados. Esse modelo preliminar, que mostra a relação de serviços, está esquematizado na Figura 5.b.

Em seguida, documentam-se as abstrações arquiteturais em um formulário elaborado com base em [12], contendo os seguintes itens: Interface (Serviços Oferecidos - portas de entrada e de saída, Serviços Requeridos), Tipo, e Semântica do componente. A Tabela 2 apresenta um resumo da especificação do componente GerenciadorTurmas utilizando esse formulário.

**Tabela 2 Especificação (Parcial) do Componente GerenciadorTurmas**

<b>Interface</b>
<b>Serviços Oferecidos</b>
<b>Portas de Entrada</b>
espAtivAcad: Oferece o serviço responsável pela atribuição de uma atividade acadêmica para uma turma
....
<b>Portas de Saída</b>
consTurma: Fornece dados referente a uma determinada turma.
...
<b>Serviços Requeridos</b>
consSalas: Solicita o serviço responsável pela obtenção das salas disponíveis...
<b>Tipo</b>
GerenciadorTurmas: Este tipo de componente encapsula as funcionalidades para criar turmas ...
<b>Semântica:</b> Para a criação de uma turma, o GerenciadorTurmas deve solicitar as atividades acadêmicas ...

De posse da especificação dos elementos arquiteturais (ver Tabela 2) e do modelo de relação de serviços (ver Figura 5.b), representamos a arquitetura em UML usando a abordagem [7] (ver Figura 5.c), onde definimos os estereótipos `<<Arch-Component>>` e `<<Arch-Connector>>` para modelar componentes e conectores respectivamente. As interfaces dos elementos arquiteturais foram modeladas por classes de interfaces. As portas e papéis foram representadas pelos estereótipos `<<in>>` e `<<out>>` indicando o sentido da comunicação. O modelo da Figura 5.c será submetido a regras de refinamento na próxima subseção.

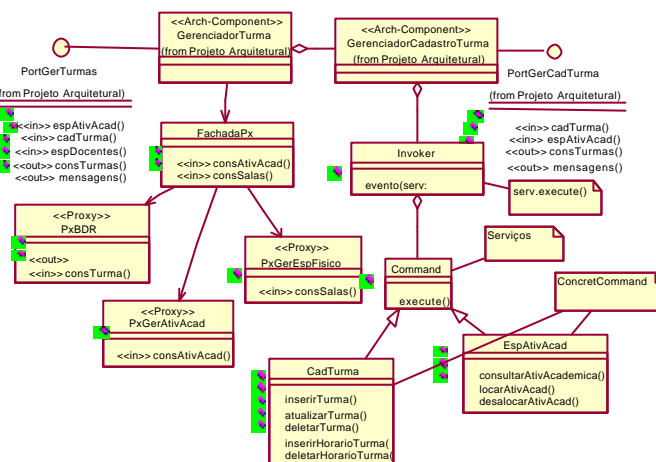


**Figura 5 (a) Modelo CBSP (b) Relação de serviços (c) Representação UML da arquitetura**

**4.3.3. Subfluxo “Refinar a Arquitetura”.** Com a arquitetura abstrata definida, segue-se com sua transposição para um nível mais concreto. Para efeito ilustrativo, refinamos o componente GerenciadorTurmas por decomposição em dois: GerenciadorCadastroTurmas oferece o serviço de criação da turma (atribuição da atividade acadêmica) e gerenciamento de detalhes da turma (e.g., salas, horários, etc); e GerenciadorAlocDocentes oferece os serviços para alocação de um ou mais docentes a uma turma criada. Na próxima seção verificamos se a arquitetura global preserva suas propriedades após o refinamento.

**4.3.4. Subfluxo “Analisar Comportamento”.** Devido ao refinamento provocar alterações estruturais, devemos verificar se a arquitetura refinada preserva o comportamento da arquitetura abstrata. Para isso, construímos um diagrama de estados UML para cada componente envolvido no refinamento. Conforme mencionamos na subseção 3.3, UML permite uma análise de comportamento limitada, onde analisamos os diagramas de estados e observamos que a lista de eventos (estados possíveis) dos componentes resultantes do refinamento está contida na lista de eventos do componente abstrato. Assim, verificamos que o comportamento da arquitetura abstrata foi preservado.

**4.3.5. Subfluxo “Projetar Componentes”.** Cada componente da arquitetura refinada deve ser concretizado através de estratégias de projeto. Observando a especificação dos componentes arquiteturais (ver Tabela 2) e a representação da AS (ver Figura 5.c), selecionamos três padrões de projeto de acordo com seus propósitos: *Facade*, *Proxy* e *Command*. Esses padrões foram aplicados no projeto dos componentes GerenciadorTurmas e GerenciadorCadastroTurmas. Nessa aplicação, os padrões foram detalhados e instanciados no contexto dos componentes (ver Figura 6).



**Figura 6 Projeto detalhado do componente GerenciadorTurmas**

Após o projeto da estrutura interna e da funcionalidade do componente (ver Figura 6), deve-se especificar sua interface através do uso da IDL (*Interface Description Language*) OMG [17].

**4.3.6. Subfluxo “Selecionar Conectores”.** Inicialmente, os *middleware* que sejam potenciais candidatas são localizados de acordo com o critério de seleção ( ver Figura 7.a). Em seguida, selecionamos os *middleware* que atendiam ao critério (ver Figura 7.b).

Critério de Seleção		
■ Suporte a Comunicação Intra e Inter-processo		
■ Características do Conector		
■ Suporte a Plataformas		
■ Método de Comunicação		

Tecnologia	Desenvolvido Por	Fonte
ILU	Xerox PARC	<a href="ftp://ftp.parc.xerox.com/pub/ilu/ilu.html">ftp://ftp.parc.xerox.com/pub/ilu/ilu.html</a>
ORB(Visroker)	Inprise	<a href="http://www.inprise.com">http://www.inprise.com</a>
RMI	Sun Microsystems	<a href="http://java.sun.com/jdk/rmi">http://java.sun.com/jdk/rmi</a>
EJB	Sun Microsystems	<a href="http://java.sun.com/products/ejb">http://java.sun.com/products/ejb</a>

**(a) Critério para seleção de middleware**      **(b) Lista de middleware candidatas**

Em seguida, coletamos as informações detalhadas sobre as tecnologias, a fim de expor os pontos positivos e negativos de cada tecnologia. Analisando essas informações, eliminamos duas tecnologias: RMI, por não promover a interoperabilidade; e ILU, por não ser tão difundida, tendo pouca documentação disponível e suporte. Finalmente, verificamos qual tecnologia melhor atende aos requisitos de comunicação. Nesse momento, efetuamos alguns testes para verificar os aspectos legais e de comunicação. Considerando o uso da técnica de tomada de decisão e a conformidade dos testes, o *middleware* ORB (Visibroker) foi selecionado.

#### 4.4. Benefícios Alcançados.

Os benefícios alcançados com a utilização do *ArcADe* são os mesmos fornecidos pela AS, tais como, reuso em níveis abstratos, utilização de estilos arquiteturais e tratamento de conectores como entidade de primeira-classe. Em contraste com o que é utilizado no NTI (ver Figura 8), podemos destacar o seguinte:

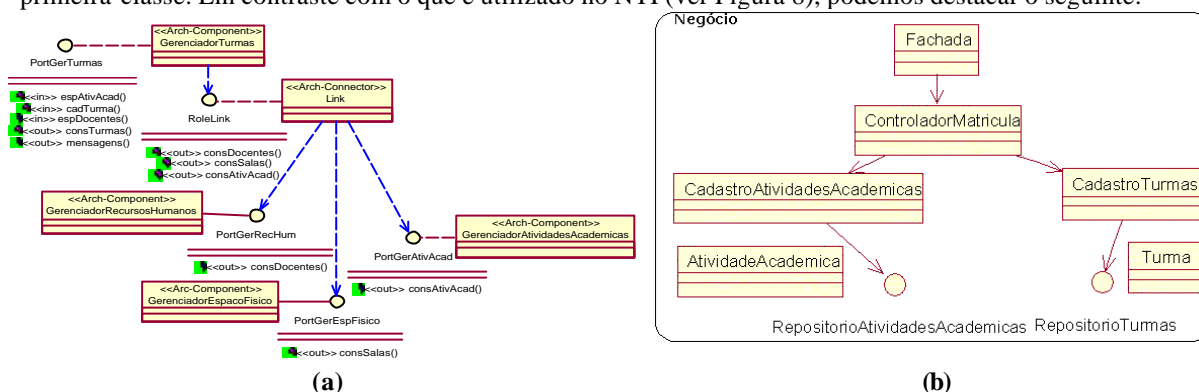


Figura 8 (a) Arquitetura produzida pelo *ArcADe* (b) Arquitetura obtida pelo processo usado no NTI

Primeiro, no *ArcADe* os componentes são diferenciados entre si através de um nome indicando seu tipo. Segundo, o modelo proposto trata elementos arquiteturais em níveis bem abstratos, por exemplo, na interface dos componentes os serviços representam uma série de operações a serem disponibilizadas, em particular, o serviço “cadastrar turmas” fornece as operações de inclusão, exclusão e alteração de uma turma. Finalmente, no *ArcADe* conectores possuem *status* de primeira-classe, ou seja, torna explícito o projeto de conectores (ver a classe *Link* na Figura 8.a), com isto promove-se a redução do acoplamento entre os componentes do sistema, aumentando-se as oportunidades de reuso de componentes e conectores em outros contextos.

No estudo de caso realizado, ilustramos a concretização de um componente (*GerenciadorTurmas*) e a seleção de um *middleware* para realizar um conector. Vale salientar que o *ArcADe* considera o modelo iterativo e incremental. Sendo assim, os demais componentes e conectores podem ser concretizados em iterações futuras até que o nível concreto da arquitetura global tenha sido alcançado.

#### 5. Conclusões e Trabalhos Futuros.

Este artigo apresentou o modelo *ArcADe* (*Software Architecture-Based Analysis and Design*) que define como a arquitetura deve ser tratada explicitamente em um processo de desenvolvimento iterativo e incremental. Esse modelo possui oito subfluxos que são organizados em dois níveis de abstração distintos. No nível de Projeto da Arquitetura podem-se executar os seguintes subfluxos: Selecionar uma arquitetura candidata; Definir a arquitetura abstrata; Refinar a arquitetura; e Analisar comportamento. No nível de Projeto Detalhado podem-se executar os seguintes subfluxos: Projetar componentes; Projetar conectores; Selecionar componentes; e Selecionar conectores. Todos esses elementos são reunidos para tratar explicitamente a arquitetura e os elementos arquiteturais (componentes e conectores).

Os subfluxos do modelo proposto foram testados, de forma experimental, no projeto do sistema SIG@UFPE (desenvolvido pelo Núcleo de Tecnologia da Informação da UFPE) para ilustrar sua utilização, detectar inconsistências e verificar a coerência das atividades e passos definidos em cada subfluxo.

O ArcADe não envolveu todo o processo de desenvolvimento do software, pois foi concentrado na etapa de análise e projeto. No entanto, outras etapas do processo também podem ser afetadas e conseqüentemente precisam ser adaptadas. Por exemplo, a etapa de implementação deve ser redefinida, pois caso seja selecionado um elemento arquitetural já existente, o esforço será de adaptação e não de implementação.

Apenas um cenário foi utilizado para ilustrar a execução do modelo proposto, o que limitou a verificação do ArcADe em apenas seis subfluxos. Sendo assim, a criação de novos cenários para exercitar e analisar os outros subfluxos, bem como a definição das outras etapas do processo de desenvolvimento de software são trabalhos de pesquisa a serem analisados futuramente, de modo a que venhamos a ter um processo completo de desenvolvimento baseado em arquitetura de software.

## 6. Referências.

- [1] C. Alves, “Seleção de Produtos de Software Utilizando uma Abordagem Baseada em Engenharia de Requisitos”, Dissertação de Mestrado. Universidade Federal de Pernambuco, Brasil, Mar, 2001.
- [2] L. Bass, P. Clements and R. Kazman. *Software Architecture in Practice*, Addison-Wesley, 1998.
- [3] Catalysis. Website: <http://www.catalysis.org>, All contents copyrighted © 1998. Last access in Jan/2004.
- [4] E. Dashofy, N. Medvidovic and R. Taylor, “Using Off-The-Shelf Middleware to Implement Connectors in Distributed Software Architectures”, 1998.
- [5] A. Egyed, P. Grunbacher, and N. Medvidovic, “Refinement and Evolution Issues in Bridging Requirements and Architecture – The CBSP Approach”, In *Proc. of the From Software Requirements to Architectures Workshop (STRAW 2001)*, Toronto, Canada, May 14, 2001.
- [6] E. Gamma, R. Helm, R. Johnson and J. Vlissides. *Padrões de Projeto – Soluções Reutilizáveis de Software Orientado a Objetos*, Porto Alegre: Bookman, 2000.
- [7] D. Garlan, A. Kompanek, and S. Cheng, “Reconciling the Needs of Architectural Description with Object-Modeling Notations”, <http://www-2.cs.cmu.edu/~able/publications/uml01/>. Last access in Jan/2004.
- [8] ISO/IEC TR 15504. Information technology – Software process assessment. Canada, 1998.
- [9] I. Jacobson, M. Griss and P. Jonsson. *Software Reuse – Architecture, Process and Organization for Business Success*, Addison-Wesley, 1997.
- [10] P. Krutchen. *The Unified Software Development Process, An Introduction*, Addison Wesley, 1999.
- [11] MDA Website: <http://www.omg.org/mda>, Last Updated Tuesday, March 12, 2002. Last access in Jan/2004.
- [12] N. Medvidovic and R. Taylor, “A Classification and Comparison Framework for Architecture Description Languages”, *IEEE Transactions on Software Engineering*, Vol 26. N° 1, Jan, 2000.
- [13] M. Moraes, “Um Framework de Análise e Projeto Baseado em Arquitetura de Software”, Dissertação de Mestrado. Universidade Federal de Pernambuco, Brasil, Jul, 2002.
- [14] M. Moriconi, X. Qian, and R. Riemenschneider, “Correct Architecture Refinement”, *Appeared in IEEE Transactions on Software Engineering*, April, 1995, Vol.21, N° 4, pp. 356-372.
- [15] N. Rosa, G. Justo, and P. Cunha, “A Framework for Building Non-Functional Software Architectures”, In *16th ACM Symposium on Applied Computing*, Las Vegas, USA, 2001. 16th ACM SAC. , 2001. p.141 – 147
- [16] M. Shaw and D. Garlan. *Software Architecture: Perspectives on an Emerging Discipline*, Prentice Hall, New Jersey, 1996.
- [17] J. Siegel. *CORBA Fundamental end Programming*, John Wiley & Sons, Inc.1996.
- [18] B. Spitznagel and D. Garlan, “A Compositional Approach for Constructing Connectors”, In *Proc. of the Working IEEE/IFIP Conference on Software Architecture (WICSA’01)*, 2001.