Utilização do *design pattern Architecture Configurator* em um ambiente de suporte para Configuração de Arquiteturas

Jonivan Coutinho Lisbôa, Orlando Gomes Loques Filho

Instituto de Computação – Universidade Federal Fluminense (UFF)
Rua Passo da Pátria 156 – Bloco E – 3º. Andar – 24210-240 – Niterói – RJ – Brasil

{jlisboa,loques}@ic.uff.br

Resumo. *O objetivo deste artigo é mostrar que o padrão Architecture Configurator provê uma base para a concepção e implementação de ambientes de suporte à configuração de arquiteturas de software. O padrão foi proposto porque observam-se certos fatores recorrentes no processo de implantação de uma arquitetura, aplicado à configuração de sistemas baseados em componentes que interagem entre si através de requisição e fornecimento de serviços. Os conceitos propostos no padrão são aplicados na prática através de um ambiente de suporte, que permite a configuração e execução de uma arquitetura de modo transparente e com mínimo impacto no processo de implementação de seus componentes.*

Abstract. *This paper aims to show that design pattern Architecture Configurator provides a sound basis for conception and implementation of support environments for the configuration of software architectures. Architecture Configurator was proposed because certain recurrent facts are observed in the software architecture configuration process, when applied to systems based on components that interact with each other requiring and supplying services. The concepts proposed in the pattern have a practical application by using a support environment that allows configuration and execution of a software architecture, in a transparent fashion and with minimum impact on the basic implementation process of its components.*

1. Introdução

Cada vez mais a concepção e o desenvolvimento de sistemas de software vêm se pautando em aspectos como a abstração, a separação entre interesses funcionais e não funcionais, a modularização e a reutilização de componentes em diferentes contextos de software, entre outros. Com isso, a abordagem arquitetural no projeto de software vem ganhando destaque. Aplicam-se conceitos como a descrição de arquiteturas de software por meio de linguagens de descrição arquitetural (ADL – *Architecture Description Language*), e a utilização de padrões para representar modelos recorrentes de software, e também para modelar o processo de implantação da configuração de um sistema [Schmidt *et al.*2000].

Um sistema de software pode ser definido a partir de seus componentes e das interações entre eles [Shaw, Garlan 1996]. Os componentes funcionais do sistema são chamados comumente de *módulos*. Os módulos do sistema carregam em si a funcionalidade do sistema, e podem ser representados pelas classes de aplicação,

procedimentos, funções e programas executáveis em si. Num sistema, os módulos podem desempenhar o papel de *servidores*, quando executam alguma ação concreta no sistema (serviço), e de *clientes*, quando coordenam a ação dos servidores, requisitando apropriadamente os serviços oferecidos. Em alguns casos, um módulo pode ao mesmo tempo desempenhar os dois papéis.

Os módulos de um sistema podem ser interligados diretamente entre si, ou através de *conectores*, que encapsulam os requisitos não-específicos do sistema, e permitem com isso uma maior flexibilidade na execução do mesmo. Os conectores mediam os contratos de interação entre os módulos, e possuem similaridade com os módulos, pois podem ser escritos em alguma linguagem de programação, sendo representados por objetos, classes, ou mesmo por alguma facilidade oferecida por um *middleware* ou *framework* operacional [Loques *et al.* 2000].

As ligações entre módulos são definidas em função de pontos de interação específicos – as *portas*. Estas são responsáveis pela ligação real entre componentes e/ou conectores. Existem dois tipos de porta: portas de saída, que são representadas pela requisição de um serviço, e portas de entrada, que são as assinaturas dos métodos disponíveis em um módulo servidor, e que podem ser encontradas na definição da interface de tal módulo.

Uma arquitetura Cliente-Servidor simples, com a interação entre o Cliente e o Servidor mediada por um conector, pode ser representada como na figura 1. Notem-se a porta de saída (*request*) e a porta de entrada (*provide*).

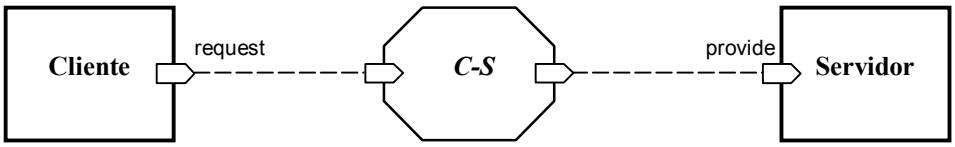


Figura 1. Uma arquitetura Cliente-Servidor simples

Pode-se dizer que a configuração de um sistema consiste em duas fases: a instanciação dos módulos funcionais (os componentes do sistema) e eventuais conectores que possam ser utilizados, e a realização das ligações entre os módulos e/ou conectores, através da conexão de portas de entrada e saída apropriadamente, de acordo com a descrição da topologia do sistema. Esse processo de configuração pode ser entendido como a implantação de uma arquitetura de software específica, segundo a qual o sistema foi modelado.

2. O padrão de projeto *Architecture Configurator*

O padrão *Architecture Configurator* foi proposto porque observam-se certos fatores recorrentes no processo de implantação de uma arquitetura de software. Tal padrão fornece um suporte para a implementação de configurações arquiteturais, fundamentando-se nos mecanismos de interceptação, encaminhamento e manipulação de requisições realizadas entre módulos de uma aplicação, e também na interligação entre módulos e conectores. Esses mecanismos são aplicados de forma transparente em relação aos elementos básicos da arquitetura (componentes, conectores e portas), e suportam propriedades como reutilização, abstração e separação de interesses. Deste

modo, os módulos e conectores participantes de uma arquitetura podem ser implementados de forma autônoma e integrados de acordo com a configuração arquitetural.

Uma descrição detalhada do padrão pode ser encontrada em [Lisbôa, Carvalho, Loques 2002]. De modo geral, o padrão pode ser aplicado a arquiteturas genéricas que apresentem componentes que fornecem e requisitam serviços. Os requisitos não-funcionais são encapsulados através de conectores configurados na arquitetura. Como ilustração da utilização do padrão, o diagrama da figura 2 mostra a solução apresentada para a implementação de uma arquitetura Cliente-Servidor simples.

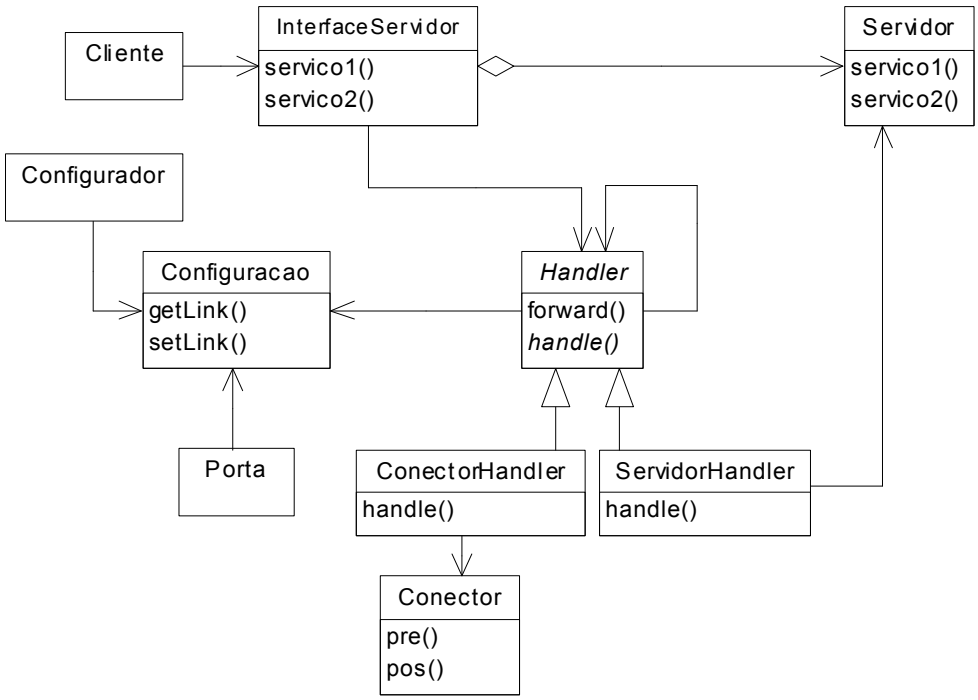


Figura 2. Diagrama de classes de uma possível solução para a arquitetura Cliente-Servidor utilizando *Architecture Configurator*

As classes `Cliente` e `Servidor` representam os módulos funcionais e a classe `ConectorHandler` representa o conector da arquitetura. A programação da configuração está representada pela classe `Configuracao`, que mantém a informação de descrição arquitetural.

`Configurador` é uma classe que define um mecanismo que interpreta as instruções de uma ADL, e a partir da descrição interpretada, define os tipos para componentes, conectores e portas presentes na aplicação.

A classe abstrata `Handler` é responsável pelo encadeamento entre conectores e componentes, conforme a descrição da arquitetura. No modelo, as classes `ConectorHandler` e `ServidorHandler` são encadeadas por `Handler`. `ServidorHandler` representa a classe `Servidor` no encadeamento e possui uma referência a esta última.

As requisições feitas por `Cliente` são interceptadas por `InterfaceServidor`, que busca em `Configuracao` a referência ao próximo conector `ConectorHandler` e invoca a operação *`handle()`* do mesmo.

Conforme a porta de entrada configurada no conector, *`handle()`* invoca a operação adequada. Cada operação descrita no conector invoca *`forward()`*, responsável por dar seqüência ao encadeamento controlado por `Handler`. Na seqüência, `ServidorHandler` tem sua operação *`handle()`* solicitada, a qual encaminha a requisição original para `Servidor`, finalizando o encadeamento.

A aplicação do padrão para arquiteturas genéricas (múltiplos clientes, servidores e conectores) é possível designando-se para cada conector configurado na arquitetura um manipulador específico, acontecendo o mesmo para os servidores. O encaminhamento das requisições terá como base as ligações descritas na arquitetura. Essa informação é consultada no momento da chamada *`forward()`*, de modo que as solicitações feitas por um cliente sigam seu caminho correto na seqüência de componentes configurados, até chegar ao servidor ligado a ele.

Para ilustrar a participação das classes no padrão, a seguir é apresentada a tabela 1, com as respectivas responsabilidades e colaborações.

Tabela 1. Resumo das classes participantes em *Architecture Configurator*, suas responsabilidades e colaborações

Classes	Responsabilidades	Colaboradores
Cliente	Utiliza a interface fornecida por Interface Servidor para requisitar um serviço particular;	InterfaceServidor
Servidor	Implementa um ou mais serviços particulares;	-
InterfaceServidor	Fornece a interface do servidor aos clientes; Recupera a referência do conector interligado ao cliente no nível da configuração, ou do próprio servidor, caso não haja nenhum conector envolvido; Repassa a solicitação do cliente ao conector recuperado, ou ao servidor, no caso de não haver conectores; Retorna ao cliente resposta oriunda do servidor;	Servidor Configurador ConectorHandler
Handler	Serve como classe abstrata base para o servidor e para os conectores; Realiza o encadeamento de conectores e componentes a partir da configuração estabelecida;	Configuracao
ConectorHandler	Implementa serviços relacionados à funcionalidade de um conector; Invoca uma de suas operações correspondente à porta de entrada requisitada; Requisita junto ao próximo conector configurado a operação correspondente a uma de suas portas de saída, através da operação *forward()* da classe Handler.	Configuracao

Tabela 1. Resumo das classes participantes em *Architecture Configurator*, suas responsabilidades e colaborações (cont.)

Classes	Responsabilidades	Colaboradores
ServidorHandler	Encaminha ao servidor a requisição vinda originariamente do cliente;	Servidor
Configurador	Define a arquitetura da aplicação a partir de uma ADL; Recebe instruções a partir de uma determinada ADL e invoca serviços da classe Configuração para executá-las;	Configuracao
Configuracao	Fornece a configuração estabelecida entre componentes e conectores; Disponibiliza serviços para configurar componentes e conectores e iniciar a aplicação; Mantém a descrição da arquitetura, bem como informações quanto à execução da mesma;	Porta
Porta	Fornece as portas configuradas de conectores e componentes com suas respectivas assinaturas;	-

3. Utilização de *Architecture Configurator* em um Ambiente de Suporte

3.1. Considerações iniciais

Com base em *Architecture Configurator*, um Ambiente de Suporte para configuração de arquiteturas deve possuir uma estrutura que consiga, a partir da interpretação de comandos presentes na descrição arquitetural, escrita em uma ADL, obter a informação de descrição do sistema, armazenando-a em estruturas de dados que permitam sua recuperação posterior, e também manter a informação de execução do sistema, que é derivada da descrição – as referências a instâncias de componentes e interligação entre elas. A partir dessa informação de execução, o ambiente deve instanciar os módulos e conectores do sistema, realizando as ligações entre eles e efetivando a execução do sistema.

A implementação de um Ambiente de Suporte pode ter como ponto de partida o diagrama de classes apresentado na figura 2, no qual é feita uma descrição de *Architecture Configurator*. Basicamente, essa implementação consiste em:

- Criação do módulo `Configurador` para agir sobre as classes `Configuracao` e `Porta`, alimentando seus objetos com os dados referentes à arquitetura do sistema;

- Elaboração de estruturas de dados para armazenar a situação arquitetural do sistema – os módulos, portas, instâncias e ligações presentes na topologia do mesmo. Esse é o papel da classe `Configuracao` em *Architecture Configurator*;
- Implementação das funcionalidades de interceptação e encaminhamento de requisições entre objetos `Cliente` e `Servidor`, realizadas pelas classes `Interface Servidor` e `Handler`.

A figura 3 destaca as classes de *Architecture Configurator* envolvidas na implementação do Ambiente de Suporte.

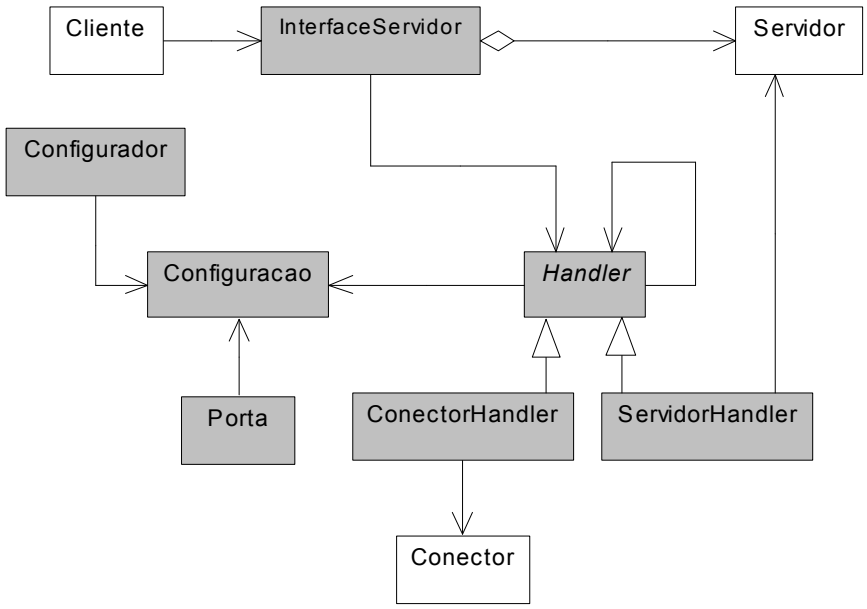


Figura 3. Diagrama simplificado de *Architecture Configurator* mostrando as classes envolvidas na implementação de um Ambiente de Suporte

3.2. Uma proposta de implementação

Após a definição dos pontos de integração de *Architecture Configurator* com um ambiente de suporte, é apresentada aqui uma proposta para implementação prática. Deve ser salientado que a abordagem utilizada refere-se a uma maneira de implementar um ambiente de suporte, não devendo ser entendida como uma forma absoluta e inflexível para a obtenção dos resultados desejados.

A implementação de um Ambiente de Suporte visa a consolidação dos conceitos propostos em *Architecture Configurator*. O ambiente proposto para tal foi concebido como uma composição de três módulos de execução:

- um Módulo de Linha de Comando, que recebe solicitações do usuário e contém o interpretador da ADL. A partir da interpretação de uma descrição é gerada a informação de configuração, que será repassada ao Coordenador;

- um Módulo Coordenador, que recebe mensagens do módulo de Linha de Comando para atualização da configuração (no caso de interpretação de uma descrição) e controle de execução da aplicação. O Coordenador contém as estruturas de dados da classe `Configuracao`, que serão distribuídas para cada módulo Executor presente no ambiente;
- um Módulo Executor, responsável pela instanciação e execução dos módulos funcionais da arquitetura configurada. Pode ser replicado no ambiente para a execução distribuída de componentes, em vários*hosts* de rede.

A figura 4 ilustra a estrutura para a implementação do Ambiente de Suporte proposto neste artigo.

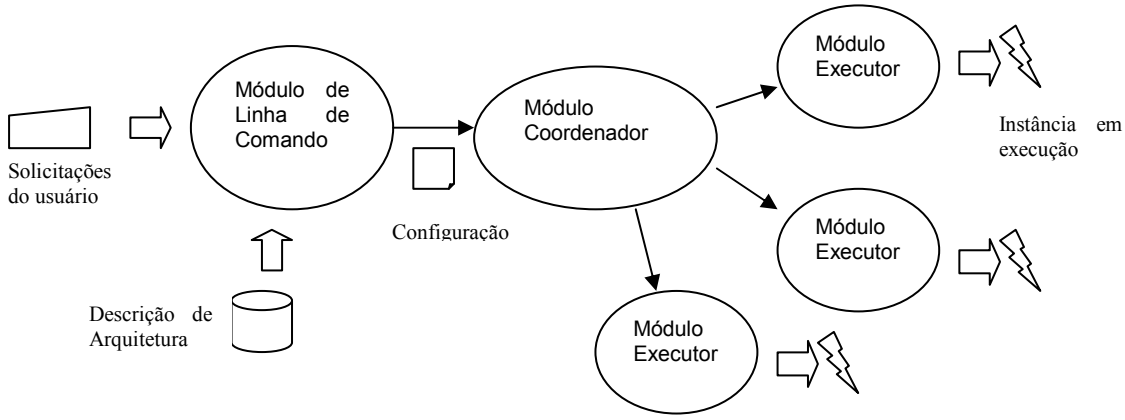


Figura 4. Estrutura proposta para o Ambiente de Suporte à configuração de arquiteturas

3.3. Descrição dos módulos de execução

O módulo de Linha de Comando possui duas funções básicas:

- ***Interpretar a descrição arquitetural escrita em uma ADL:*** é a função da classe `Configurador` de *Architecture Configurator*. Através da interpretação da descrição, é criada a informação sobre a topologia da aplicação, a ser armazenada na classe `Configuracao`. Na proposta apresentada, a função da classe `Configuracao` é feita por uma estrutura de dados que envolve informações sobre os módulos, portas, instâncias e ligações. Essa estrutura pode ser implementada, por exemplo, na forma de tabelas de símbolos, tabelas dinâmicas ou qualquer outro tipo de repositório de objetos, de acordo com a linguagem de programação utilizada.
- ***Receber a intervenção manual do usuário sobre a execução do sistema:*** assim, é possível a interação com a execução da aplicação, permitindo, entre outras coisas, a reconfiguração em tempo de execução.

O módulo Coordenador possui a função de receber solicitações do módulo de Linha de Comando, que podem ser de duas naturezas: o carregamento de uma nova descrição, ou a intervenção do usuário para mudar o estado da aplicação. No primeiro caso, o Coordenador recebe a estrutura de dados com a informação de configuração e a

distribui para os módulos Executores. No segundo caso, o Coordenador interfere na execução das instâncias de componentes do sistema.

O módulo Executor cuida efetivamente da instanciação e execução dos componentes da aplicação. Sua atividade pode ser dividida em três etapas:

- ***Armazenamento da configuração:*** o Executor mantém uma cópia da configuração da aplicação. Essa cópia lhe é enviada pelo Coordenador assim que o mesmo recebe a informação de descrição vinda do módulo de Linha de Comando.
- ***Instanciação de componentes:*** no momento da inicialização da aplicação, o Coordenador envia ao Executor um sinal para a criação das instâncias de componentes que estarão hospedadas em seu *host*, e que serão gerenciadas por ele. O Executor então procede com a criação das instâncias dos componentes, do *proxy* do servidor (`InterfaceServidor`) e dos manipuladores de componentes (`ConectorHandler` e `ServidorHandler`).
- ***Encaminhamento de solicitações:*** as solicitações são encaminhadas segundo o encadeamento de componentes previsto na descrição arquitetural. Cabe ao Executor, consultando a informação de descrição que mantém, descobrir o próximo componente presente do encadeamento, para que a solicitação feita possa ser tratada e encaminhada adiante. É a função da classe `Handler` de *Architecture Configurator*.

4. Conclusão

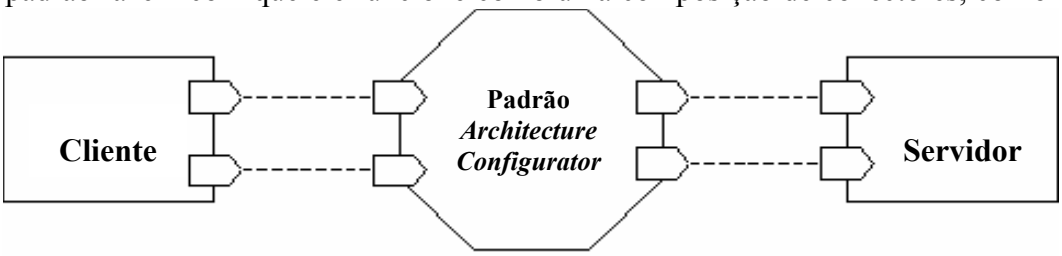
A implementação da proposta apresentada neste artigo permite que seja atingido o objetivo de aplicar na prática os conceitos propostos em *Architecture Configurator*. Com a implementação dos pontos principais propostos no padrão – um interpretador para a descrição arquitetural, um esquema para interceptação e encaminhamento de requisições e um esquema para gerenciamento da configuração através de estruturas para armazenamento e consulta – torna-se possível a obtenção dos principais requisitos desejados na aplicação de arquiteturas: reutilização de software, abstração, separação de interesses, flexibilidade, extensibilidade, entre outros [Lisbôa 2003].

Em comparação com outras abordagens existentes, a utilização de *Architecture Configurator* em um ambiente de suporte tem como diferenciais:

- A disponibilização da informação de descrição arquitetural, armazenada em estruturas de dados, permite que esta seja utilizada para análises estruturais de comportamentais do sistema (verificação de propriedades, validação, testes, etc.), e também guiar atividades de reconfiguração dinâmica, de modo a adaptar o sistema a mudanças no ambiente operacional, através de, por exemplo, contratos de qualidade de serviço diferenciada [Cerqueira *et al.* 2003]. Essa possibilidade não é contemplada em abordagens como as utilizadas no padrão *Component Configurator* [Schmidt *et al.* 2000], e no servidor JBoss [Fleury, Reverbel 2003], por exemplo. Nestas, são privilegiados somente os aspectos de ligação entre componentes, sem haver uma preocupação com a arquitetura.

- O esquema de interceptação e encaminhamento de requisições é encapsulado no ambiente, de modo que os componentes do sistema não precisam ter seu código adaptado para sua utilização, que é feita de maneira transparente. Isso aumenta a possibilidade de reutilização de componentes. No servidor JBoss, por exemplo, os clientes devem ter suas chamadas modificadas para a interação com um componente específico do ambiente (o *MBeans*), que faz a ligação entre os clientes e o *middleware* de suporte à configuração.

O emprego de arquiteturas de software de forma transparente em aplicações baseadas em interação de componentes cliente-servidor, com a utilização do Ambiente de Suporte, faz com que *Architecture Configurator* funcione como um conector de configuração de software, pois ocorre a intermediação da interação entre cliente e servidor, com isolamento e separação de interesses. Na verdade, todos os requisitos não-específicos da aplicação implementados pelo encadeamento de conectores configurado no padrão fazem com que ele funcione como uma composição de conectores, conforme



ilustrado na figura 5.

Figura 5. *Architecture Configurator* funcionando como um conector para configuração de arquiteturas

Referências

Cerqueira, R. *et al.* (2003) "Deploying Non-Functional Aspects by Contract". The Second Workshop on Reflective Adaptative Middleware, International Middleware Conference – MW2003, junho.

Fleury, M., Reverbel, F. (2003) "The JBoss Extensible Server". The Second Workshop on Reflective Adaptative Middleware - MW2003, junho.

Lisbôa, J., Carvalho, S. e Loques, O. (2002) "Um *Design Pattern* para Configuração de Arquiteturas de Software". SugarLoafPLoP 2002 Proceedings, impresso por ICMC-USP, dezembro, p. 37-54.

Lisbôa, J. (2003) "Utilização do *Design Pattern Architecture Configurator* em um Ambiente de Suporte à Configuração de Arquiteturas". Dissertação de Mestrado. IC/UFF, fevereiro.

Loques, O. *et al.* (2000) "On the Integration of Configuration and Meta-Level Programming Approaches". Lecture Notes in Computer Science, vol. 1826. Cazzolla, W. Stroud, R. Tizato, F. (eds). , p. 189-208.

Schmidt, D. *et al.* (2000) "Pattern Oriented Software Architecture Vol. 2: Patterns for Concurrent and Networked Objects". Wiley & Sons.

Shaw, M. e Garlan, D. (1996) “Software Architecture: Perspectives on an Emerging Discipline”. Prentice Hall.