Um Meta-Modelo para Especificação de Arquiteturas de Software em Camadas

Lyrene Fernandes da Silva e Virgínia C. Carneiro de Paula
Universidade Federal do Rio Grande do Norte (UFRN)
Departamento de Informática e Matemática Aplicada (DIMAp)
{lyrene, vccpaula}@ufrnet.br

Resumo

A crescente complexidade dos sistemas de software tem levado a comunidade de pesquisadores e projetistas a criar ou melhorar técnicas e métodos para o desenvolvimento, de maneira a aumentar a satisfação de clientes e a produtividade dos projetistas. A reutilização de código já não é suficiente para a grande demanda por softwares de alta qualidade nem para a engenharia deles. Desenvolver componentes de alta abstração é necessário. Porém, reutilizar ou integrar estes componentes a outros são tarefas que demandam um certo trabalho. A escolha de uma boa arquitetura é o primeiro ponto a se pensar. Uma expressiva e completa representação do sistema ou partes dele é outro fator crucial para o desenvolvimento de sistemas baseados em componentes reutilizáveis. O presente trabalho tem o intuito de apresentar um meta-modelo para especificação de arquiteturas de software em camadas que visa facilitar a comunicação entre arquitetos e projetistas e a transformação da arquitetura em design detalhado do software. Este meta-modelo é fruto da integração de algumas tecnologias que estão sendo largamente utilizadas por desenvolvedores de software, a UML (Unified Modeling Language) e os conceitos de estilos e visões arquiteturais priorizando a reusabilidade.

Abstract

The growing complexity of the software systems has been leading the researchers and designers to create or improve techniques and methods for the development, in a way to increase the customers satisfaction and the designers productivity. The code reuse is no longer enough for the great demand for high quality software nor for their engineering. Developing components of high abstraction is necessary. However, reusing or integrating these components the others are tasks that demand an accurate work. The choice of a good architecture is the first point for thinking of. An expressive and complete representation of the system or parts of him is another crucial factor for the development of systems based on components reuse. This work presents a meta-model for the specification of layered software architectures. It facilite the comunication between architects and designers and the transformation of the architecture in software detailed design. This meta-model is the result of the integration of some technologies that are being broadly used by software developers: the UML (Unified Modeling Language), the concept of architectural styles, and the concept of views prioritizing the reusability.

1 – Introdução

Atingir a satisfação de clientes e usuários através de sistemas de software que supram seus requisitos e que possuam o maior tempo de vida possível é o objetivo de desenvolvedores e pesquisadores da área de engenharia de software. No entanto, a complexidade imposta pelos sistemas atuais exige que técnicas e métodos sejam desenvolvidos e usados para tratar ou gerenciar tal complexidade.

Atributos como manutenibilidade (modificabilidade e extensibilidade) e reusabilidade devem guiar os projetistas durante o desenvolvimento de um sistema para que se possa

minimizar os custos e maximizar a produtividade com os projetos. Levantar e solucionar problemas antecipadamente são os resultados do uso de arquiteturas de software bem projetadas, para um sistema ou para uma família deles.

A arquitetura de software tem desempenhado papel fundamental no desenvolvimento de sistemas de software. Ela oferece um maior entendimento da aplicação por dividi-la em um conjunto de componentes que interagem entre si para realizar parte de uma, uma ou várias funcionalidades do sistema. Além disso, dispõe de métodos que garantem menores custos e tempo no desenvolvimento por provê reusabilidade de componentes ou estilos arquiteturais já existentes e por permitir que vários times trabalhem paralelamente em partes (subsistemas) diferentes do sistema.

Desta forma, a complexidade intrínseca dos sistemas de software atuais pode ser melhor gerenciada. A divisão de um sistema em subsistemas menores e menos complexos provê uma maneira de fazer com que o desenvolvedor concentre seus esforços em pedaços especializados em algum serviço e garanta que eles sejam completos e validados.

Estilos arquiteturais, visões e linguagens de modelagem facilitam o processo de *design* da arquitetura de um sistema e são instrumentos fundamentais para alcançar tais resultados. Um dos estilos arquiteturais mais utilizados para estruturar um sistema de software é o estilo camadas [1][2]. Ele proporciona, aos projetistas, uma maior flexibilidade de desenvolvimento por subdividir o sistema em camadas especializadas em tipos de serviços, que se bem projetadas podem ser facilmente reutilizadas.

A representatividade do projeto é outro fator muito importante quando se pensa em desenvolvimento, reutilização e manutenção. Para tanto, a UML (Unified Modeling Language) tem se mostrado adequada e tem se tornado linguagem padrão para modelagem por apresentar características como flexibilidade, facilidade de aprender e usar. A estruturação e representatividade de um sistema, também, podem ser intensificadas através do uso de visões. Assim como as camadas, as visões ajudam a gerenciar a complexidade de um sistema.

O presente trabalho tem o intuito de apresentar um meta-modelo para especificação de arquiteturas de software em camadas. O objetivo deste meta-modelo é fornecer as diretrizes básicas para especificar este tipo de arquitetura e facilitar, não somente a comunicação entre arquitetos e projetistas mas, também, a transformação da arquitetura em *design* detalhado do software. Este meta-modelo é fruto da integração do estilo camadas, a UML e visões arquiteturais e prioriza a reusabilidade.

A Seção 2, a seguir, fornece uma visão geral da arquitetura de software e dos estilos arquiteturais. As seções 3 e 4 apresentam a UML no contexto da arquitetura de software. A arquitetura em camadas e o meta-modelo para o desenvolvimento de sistemas em camadas são apresentados nos capítulo 5 e 6, respectivamente. Por fim, as conclusões e trabalhos futuros na Seção 7.

2 - Arquitetura de Software e Estilos Arquiteturais

A arquitetura de software surgiu para contornar o *gap* existente entre a análise e o *design* detalhado no processo de desenvolvimento de sistemas de software. Ela fornece um melhor entendimento das partes que constituirão o software e de sua estrutura global. E assim, dá origem a sistemas mais flexíveis, dinâmicos e reais.

A arquitetura de um sistema computacional é uma estrutura ou conjunto de estruturas que incluem componentes de software, as propriedades destes componentes que são visíveis externamente, e relacionamentos entre os componentes [3]. Os *componentes* arquiteturais podem ser objetos, processos, bibliotecas, bancos de dados ou outros. Os *relacionamentos*

representam informações de como os componentes interagem entre si e as *propriedades* são informações que indicam o comportamento dos componentes, podendo assim, um componente ser diferenciado de outros.

A arquitetura representa um sistema sob um alto nível de abstração, de onde o sistema pode ser visto como um todo. Ela representa os atributos funcionais e não-funcionais requeridos para o sistema, e oculta todo e qualquer detalhe de implementação. Para fornecer uma representação de todos os aspectos arquiteturais de um sistema pode-se fazer uso de visões ou pontos de vista [2]. As visões ajudam a gerenciar a complexidade por separar diferentes aspectos em visões distintas (Genericamente, estes aspectos são funcionais, não-funcionais e organizacionais) [2].

Com o uso de arquiteturas, padrões arquiteturais empregados em diversos sistemas e domínios foram identificados [2][4]. Um padrão arquitetural ou estilo define uma família de sistemas em termos de sua organização estrutural [4][5]. Ele define o vocabulário de componentes e conectores que podem ser usados em instâncias daquele estilo, juntamente com um conjunto de restrições de como eles podem ser combinados. O estilo também deve definir como construí-lo e quando utilizá-lo.

O uso de estilos proporciona diversas vantagens para o *design* tais como, documentam a experiência de *design* de arquiteturas já provadas, provêem vocabulário e entendimento comum, ajudam a construir arquiteturas de software complexas e heterogêneas, provêem um esqueleto de comportamento funcional e ajudam a gerenciar a complexidade de sistemas [5]. Eles estão sendo largamente utilizados, puros ou híbridos[1] [3].

A utilização prática destes estilos e arquiteturas deu origem a duas novas categorias de estilos arquiteturais [2]: Arquitetura de referência ou Arquitetura de software de domínio específico e Arquitetura de linha de produto. *Arquiteturas de referência* são estilos que se aplicam a domínios de aplicação específicos. Eles definem como a funcionalidade do domínio é mapeada para os elementos da arquitetura. *Arquiteturas de linha de produto* são estilos que se aplicam a um conjunto de produtos de uma organização ou companhia.

O uso de estilos depende exclusivamente das propriedades (requisitos estruturais, comportamentais e de qualidade) que o sistema ou subsistema deverá apresentar [Shaw, Garlan e Buschmann in 6][7]. Não se deve, em hipótese alguma, escolher um estilo e adaptar o sistema a este estilo, pois desta forma, o modelo obtido não retratará a realidade pretendida. Os exemplos de estilos mais conhecidos são: *Pipe and Filters*, Camadas, Cliente-Servidor, Repositórios dentre outros.

Uma solução globalmente aceita para a estruturação de um sistema é o uso do estilo camadas [1][2][4]. Este estilo favorece a reusabilidade de componentes especializados em fornecer algum tipo de serviço como por exemplo, serviços de gerenciamento de banco de dados, de aplicações específicas, etc. Na Seção 5 maiores detalhes deste estilo serão apresentados.

3 - Arquitetura de software e UML

A UML (Unified Modeling Language) surgiu por volta de 1995 com o intuito de padronizar uma linguagem para modelagem de sistemas. Ela se estende a qualquer tipo de sistema e se propõe a dar suporte a todo o processo de desenvolvimento de software [8].

[1] Estilos arquiteturais puros são os próprios estilos em sua forma primária. Arquiteturas híbridas ou heterogêneas são aquelas que integram vários estilos para descrição do sistema. A maioria dos sistemas são descritos por uma arquitetura heterogênea.

A UML é composta por um conjunto de nove diagramas com sintaxe e semântica bem definidas. São eles: Diagrama de casos de uso, de classes, de objetos, de seqüência, de colaboração, de estados, de atividades, de implantação e de componentes. Cada um deles retrata aspectos diferentes do sistema. Conjuntamente, eles oferecem todas as visões do sistema.

A UML vem sendo largamente utilizada pelas diversas subáreas da engenharia de software devido a sua expressividade e flexibilidade. Pesquisadores da área de arquitetura de software têm mostrado grande interesse em integrar a UML às técnicas e linguagens existentes para descrição de arquiteturas [9][10].

Entretanto, uma arquitetura apresenta um conjunto de características que deve ser modelado por linguagens de descrição adequadas, denominadas ADLs (Architecture Description Language) [11][12]. A UML representa boa parte destas características embora não seja uma ADL [9][13].

Um dos aspectos importantes não satisfeito pela UML é quanto à descrição de estilos. Este aspecto é endereçado, apenas, parcialmente pela UML. A UML não possui estilos pré-definidos, e nem mesmo o conceito de estilos. Entretanto, devido a sua flexibilidade, a UML é capaz de descrever qualquer tipo de sistema a qualquer nível de abstração, englobando tanto seus aspectos estáticos quanto dinâmicos. Assim, é possível descrever meta-modelos e através destes representar estilos [13].

Estes meta-modelos podem ser instanciados para arquiteturas específicas. Obtendo-se arquiteturas descritas em nível de projeto seguindo algum padrão arquitetural específico. A representação de estilos em UML pode proporcionar maior consistência entre arquiteturas de software e modelos de *design*, além de oferecer melhor comunicação entre arquitetos e projetistas.

Para que esta prática traga os benefícios almejados é necessário que os estilos sejam modelados de maneira bastante fiel, sem que se perca nenhuma das propriedades definidas por eles. O uso da Linguagem de Restrições de Objetos (OCL – Object Constraint Language) pode ser necessário. A descrição de restrições é um ponto muito importante para os estilos e deve ser descrito de maneira precisa. Na próxima Seção apresentaremos o conceito de estilo usando a UML.

4 - Estilos e UML

Um meta-modelo baseado nos elementos arquiteturais - componentes, inter-relacionamentos e interfaces - foi definido por Homeister et al. [2], denominado Visão Conceitual da arquitetura. Homeister et al. definiram quatro meta-modelos ou visões que representam aspectos diferenciados de uma arquitetura e juntos fornecem todos os pontos-de-vista arquiteturais. A Visão Conceitual da arquitetura de software pode ser instanciada para criação de qualquer arquitetura (veja Figura 1).

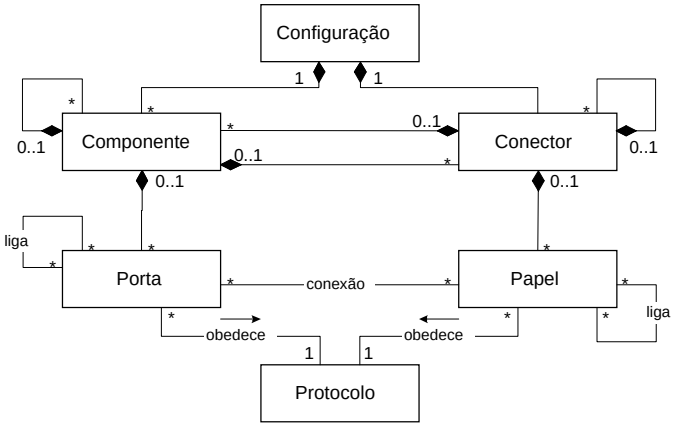

Figura 1: Meta-modelo da visão conceitual da arquitetura [2].

- *Configuração*: A configuração define os tipos de componentes e conectores e restringe como instâncias destes tipos podem ser interconectadas.
- *Componente*: É uma parte encapsulada de um sistema de software. Um componente possui uma interface para interação com o mundo externo que ocorre através de pontos de interação denominados, *portas*. Cada porta possui um *protocolo* associado que indica como as operações de entrada e saída podem ser realizadas.
- *Conector*: É a relação que denota a comunicação entre componentes. Da mesma maneira que componentes possuem portas, conectores possuem *papéis* como pontos de interação com os outros elementos da arquitetura. Os papéis também obedecem a um *protocolo*.
- *Protocolo*: O protocolo define as regras de comunicação entre os pontos de interação, portas e papéis. Ele fornece a infraestrutura de comunicação para a arquitetura.

Assim como as arquiteturas, um estilo é constituído de um vocabulário (os tipos de componentes e conectores) e uma configuração, além de ter um protocolo associado. Esta configuração é uma instanciação do modelo da Figura 1 para o vocabulário e restrições desejadas. A visão conceitual fornece algumas restrições genéricas. Cada estilo deve herdar estas restrições básicas e estabelecer as propriedades intrínsecas ao próprio estilo. As características comportamentais do estilo também devem ser modeladas.

Através da UML, os aspectos estáticos e dinâmicos dos estilos podem ser modelados. Os diagramas de classes, interações, estados e atividades oferecem este subsídio. A flexibilidade da UML provê maneiras de criar novos elementos para modelagem através dos estereótipos. Os estereótipos podem fornecer uma linguagem (vocabulário visual) adaptada a cada estilo.

Utilizar a UML para modelar estilos significa dispor de todas as vantagens de uma linguagem fácil de aprender e usar, flexível e expressiva e que conta com uma grande popularidade. Isto facilitará o uso de estilos arquiteturais, já que estes estarão mais próximos do *design*. E facilitará o uso da UML para descrever arquiteturas.

A seguir, apresentaremos o estilo camadas. A instanciação da visão conceitual para este estilo será mostrada em seguida, na Seção 6. Os diagramas de estados e seqüência serão utilizados para modelar seus aspectos comportamentais.

5 – O Estilo Camadas

O estilo camadas organiza o sistema em camadas hierárquicas. Uma camada pode ser, livremente, definida como um conjunto de sub-sistemas com o mesmo grau de generalidade. Camadas mais altas são específicas da aplicação, camadas mais baixas são de propósito geral, ou seja, utilizadas por diferentes aplicações. Elas funcionam como servidoras para as camadas mais altas e como clientes para as camadas mais baixas. Tradicionalmente, as interações ocorrem somente entre camadas adjacentes.

Um exemplo clássico de arquitetura baseada neste estilo é o padrão OSI da ISO para redes de computadores, onde cada camada é responsável por um aspecto específico da comunicação entre computadores, dos detalhes de transmissão de bits (sinais elétricos) pelo meio físico até o alto nível lógico da aplicação.

Os sistemas em camadas possuem várias propriedades desejáveis: eles suportam o *design* baseado em níveis crescentes de abstração - isto permite implementadores dividirem um problema complexo em uma seqüência de passos incrementais; eles suportam mudanças - porque cada camada interage somente com suas camadas adjacentes, então mudanças em uma camada afetam no máximo duas outras camadas; eles suportam reuso – pois, diferentes implementações de uma mesma camada podem ser usadas, desde que elas possuam as mesmas interfaces para as camadas adjacentes; e a segurança pode ser suportada por camadas específicas para este serviço [4].

Porém, sistemas em camadas também possuem algumas desvantagens: nem todos os sistemas são facilmente estruturados em camadas; para os sistemas que podem ser estruturados em camadas a questão de performance é delicada, pois há um certo tempo e processamento desperdiçado na comunicação entre camadas; e falhas em uma camada pode tornar o sistema indisponível [4].

Uma arquitetura típica para aplicações de software é a arquitetura de quatro camadas apresentada por Jacobson et al. [1] (veja Figura 2). O número de camadas, o nome e o conteúdo delas não é fixo, e varia de situação para situação.

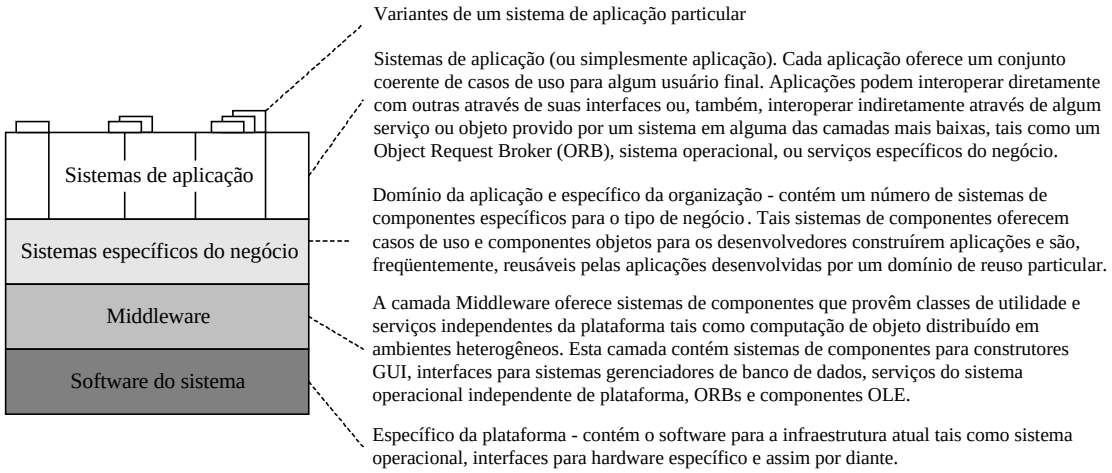

Figura 2: Uma arquitetura em camadas típica para aplicações de negócio [1]

As camadas apresentadas na Figura 2 são estruturais e representam o sistema completamente sob um ponto de vista de alta granularidade[2], devendo assim ser refinadas até um nível de abstração onde sejam, apropriadamente, implementadas em uma linguagem de programação.

Os sistemas constituintes de cada camada podem, também, ser estruturados em outras camadas ou seguir algum outro estilo arquitetural existente, puro ou heterogêneo. Esta decisão dependerá da complexidade e características globais de cada uma das camadas [4].

Um exemplo prático do uso desta arquitetura é a Arquitetura de Gerenciamento de Objetos (OMA) definida pelo OMG (Object Manegement Group) para integrar aplicações distribuídas [14]. Esta arquitetura fornece o grau de transparência desejado para construir sistemas distribuídos sem se preocupar com problemas complexos tais como a distribuição e heterogeneidade de plataformas [15][16][17].

As quatro visões definidas por Homeister et al. [2] - Visão Conceitual, Visão de Módulo, Visão de Execução e Visão de Código é um outro trabalho que mostra a necessidade de se estruturar um sistema em camadas. A Visão de Módulo, definida por eles, estrutura os componentes e relacionamentos em módulos, e organiza-os em camadas hierárquicas (ou parcialmente hierárquicas). Homeister et al. [2] definiram as quatro visões como o mínimo a se pensar quando desenvolver uma arquitetura, assim eles consideram fundamental organizar os sistemas em camadas.

Variantes do estilo camadas é o *Relaxed Layered System* [3] que é menos restritivo sobre a relação entre camadas. Cada camada pode usar serviços de outra camada que não a que está imediatamente abaixo. Uma camada pode também ser parcialmente opaca, isto significa que alguns serviços só são visíveis para a camada imediatamente superior, mas outros são visíveis para qualquer camada. Outras variantes deste estilo podem ser obtidas por modificar o tipo das camadas – que pode ser *black*, *white* ou *gray*, o tipo de comunicação entre elas – *push*, *pull* e *callback* ou o sentido da comunicação entre elas – *top-down* e *bottom-up*.

Outra variação é o *Layering Through Inheritance* [5] onde camadas mais baixas são implementadas como classes básicas. As camadas de nível mais alto requisitam serviços das camadas mais baixas por herdar a implementação destas camadas. Conseqüentemente, elas podem requisitar serviços das classes básicas. Uma vantagem deste esquema é que as camadas mais altas podem modificar os serviços das classes básicas de acordo com suas necessidades. Por outro lado, a relação de herança conduz a uma ligação muito forte entre camadas de nível baixo e nível alto.

Na próxima Seção, apresentaremos um meta-modelo para facilitar a utilização do estilo camadas.

6 – Um Meta-modelo para especificação de arquiteturas de software em camadas

A proposta deste trabalho é especificar o estilo camadas de maneira que ele possa ser facilmente entendido e utilizado. Para tanto, reunimos várias descrições textuais sobre estilos arquiteturais, em especial, sobre o estilo camadas [2][3][4][5], e propomos um meta-modelo,

[2] Componentes (no nosso caso camadas) podem ter níveis de granularidade diferentes tais como pequeno, médio e grande. Os pequenos são, em geral, mais fáceis de desenvolver e mais difíceis de compor. Eles são similares às classes de objeto e são, normalmente, de propósito geral. Componentes médios e grandes são mais difíceis de desenvolver porém a composição deles é mais fácil do que a de componentes pequenos. Estes são, geralmente, de propósito específico a um domínio. Componentes maiores são constituídos de componentes menores [18].

especificado em UML, para a descrição de arquiteturas de software em camadas. A UML oferece várias características para tornar um estilo mais facilmente utilizado, como definido nas seções 3 e 4. Este trabalho constitui o primeiro passo para que a UML dê suporte à descrição de estilos[3].

Como definido anteriormente, o estilo camadas ajuda a estruturar aplicações que podem ser decompostas em um grupo de sub-sistemas com níveis de generalidade diferentes. Este estilo é normalmente utilizado em grandes sistemas que requerem decomposição e nos quais não haja dependências diretas entre as partes decompostas (camadas).

Veremos a seguir a especificação da estrutura estática do meta-modelo para o estilo camadas, modelada através do diagrama de classes e a sua estrutura dinâmica ou comportamental modelada através dos diagramas de seqüências e de estados da UML.

6.1 - Descrição estática através do Diagrama de classes

A especificação do estilo camadas tem como ponto de partida a Visão Conceitual definida na Seção 4. Os elementos do diagrama de classes da Figura 1 têm correlação direta com os elementos do diagrama de classes da Figura 3, a seguir. Ou seja, Configuração está correlacionado com Configuração em Camadas, Camadas com Componentes, Conectores com Conectores e assim sucessivamente.

O diagrama da Figura 3 retrata os elementos do estilo camadas. Os elementos e as operações básicas são fornecidos. As Notas oferecem uma visão geral do que constitui cada operação. Estereótipos podem ser utilizados para encapsular camadas e portas em um único "componente" e conectores e papéis em outro.

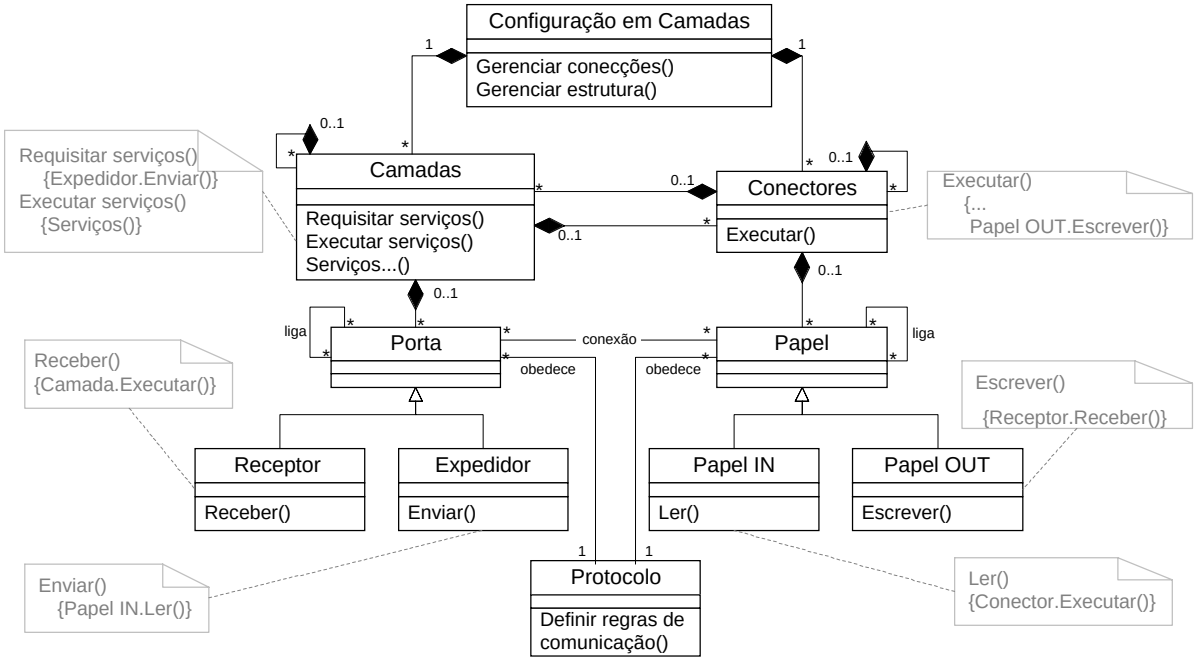

Figura 3: Estrutura estática do estilo camadas

[3] Descrever estilos significa que a linguagem reconhece o vocabulário, estrutura e restrições de determinados estilos. Esta é uma das propriedades desejáveis às ADLs.

As classes Porta e Papel são classes abstratas das classes Receptor, Expedidor, Papel IN, e Papel OUT que de fato regem uma conexão. Portanto, elas são apenas organizacionais neste modelo, eventualmente elas poderiam exercer alguma funcionalidade extra. As classes Protocolo e Configuração em Camadas são classes administrativas, ou seja, gerenciam a estrutura e a interação entre os componentes. Estas classes devem ter suas operações cuidadosamente especificadas para que se possa garantir que as restrições do estilo estão sendo obedecidas. O uso da OCL (Object Constraint Language) é uma maneira de atingir este propósito.

A seguir uma breve descrição, textual e através do cartão CRC[4], de cada um dos elementos e restrições arquiteturais para o estilo camadas é apresentada.

A *configuração em camadas* trabalha em termos de camadas (para os componentes) e conectores. Ela é responsável por garantir que interações ocorram somente entre camadas adjacentes. Normalmente, as camadas superiores solicitam serviços às camadas inferiores, numa comunicação *top-down.* A comunicação *bottom-up,* quando camadas inferiores requisitam serviços às camadas mais acima, ou a combinação de *top-down* e *bottom-up* também pode ocorrer. O resumo destas propriedades é mostrado na Figura 4.

Classe *Configuração em Camadas*	Colaboradores Camada Conector
Responsabilidade Definir o tipo de comunicação (*bottom-up, top-down*) Gerenciar estrutura (quais camadas requerem/ fornecem serviços de/para quais camadas)	

Figura 4: Resumo das características da *Configuração em Camadas*

As *camadas* são as entidades provedoras de serviços para o sistema (Figura 5). Elas provêem os serviços requisitados a elas por realizar algumas tarefas e por requisitarem outros serviços a outras camadas. As requisições, normalmente, não necessitam de resposta, ou seja, uma camada não precisa esperar que as tarefas de outras camadas finalizem para que ela complete algum serviço. Cada camada pode ser internamente estruturada seguindo o estilo camadas ou qualquer outro. O nosso meta-modelo não faz restrições ao uso de qualquer estilo em uma camada. Cada camada pode ou não conhecer o trabalho interno de suas camadas adjacentes.

Classe *Camada*	Colaboradores Receptor Expedidor
Responsabilidade Prover serviços usados pelas camadas (adjacentes) Delegar sub-tarefas para as camadas (adjacentes)	

Figura 5: Resumo das características da *Camada*

[4] O cartão CRC (Class-Responsability-Collaborators) descreve, informalmente, um componente em termos de suas responsabilidades e colaboradores [5].

A interação entre camadas ocorre através de seus pontos de interação, denominados *Portas,* e dos conectores. Há dois tipos de portas denominadas, neste trabalho, de *Receptor* e *Expedidor* (Figura 6 e 7). O Receptor é responsável por receber requisições de algum serviço fornecido pelo componente que a contém. O Expedidor é responsável por iniciar a interação com outros componentes através da requisição de algum serviço.

Classe *Receptor*	Colaboradores Papel OUT Camada
Responsabilidade Receber as requisições (e dados) do Papel OUT	

Figura 6: Resumo das características do *Receptor*

Classe *Expedidor*	Colaboradores Papel IN Camada
Responsabilidade Enviar as requisições (e dados) ao Papel IN	

Figura 7: Resumo das características do *Expedidor*

As camadas são independentes umas das outras. Sendo assim, cada porta conhece somente os conectores que estão diretamente ligados a elas. Os *conectores* transferem as requisições de um componente para outro e podem realizar alguma tarefa adicional para melhorar a comunicação entre componentes como por exemplo, transformar tipos de dados, verificar se o componente no final do conector está ativo e apto a receber requisições etc (Figura 8).

Classe *Conector*	Colaboradores Papel IN Papel OUT
Responsabilidade Realizar alguma tarefa de comunicação	

Figura 8: Resumo das características do *Conector*

Os conectores, através de seus pontos de interação denominados Papéis, interagem com as camadas. Assim como as portas, os papéis são de dois tipos, Papel IN e Papel OUT (Figuras 9 e 10). O Papel IN é responsável por ler as requisições do Expedidor e o Papel OUT por escrever no Receptor do componente que irá fornecer o serviço desejado.

Classe *Papel IN*	Colaboradores Expedidor Conector
Responsabilidade Ler requisições (e dados) no Expedidor	

Figura 9: Resumo das características do *Papel IN*

Classe *Papel OUT*	Colaboradores Conector Receptor
Responsabilidade Escrever requisições (e dados) no Receptor	

Figura 10: Resumo das características da *Papel OUT*

Os Papéis e as Portas devem obedecer a um *protocolo* para que a comunicação realmente aconteça. O protocolo é responsável por determinar as regras de comunicação tais como o sentido da comunicação[5] etc. Deve-se garantir que Papéis IN só irão interagir com Expedidores e Papéis OUT com Receptores e que os tipos de dados passados entre Portas e Papéis são os mesmos ou compatíveis (Figura 11).

Classe *Protocolo*	Colaboradores Papel Porta
Responsabilidade Definir regras de comunicação: tipos de dados, tipo de comunicação	

Figura 11: Resumo das características do *Protocolo*

6.2 - Descrição dinâmica através do Diagrama de seqüências

O diagrama de seqüências da Figura 12 retrata a seqüência de interações quando um componente (camada) precisa se comunicar com outro. Uma camada pode requisitar serviços de outras camadas através da interação existente entre portas e papéis. O protocolo é a infra-estrutura necessária para esta comunicação.

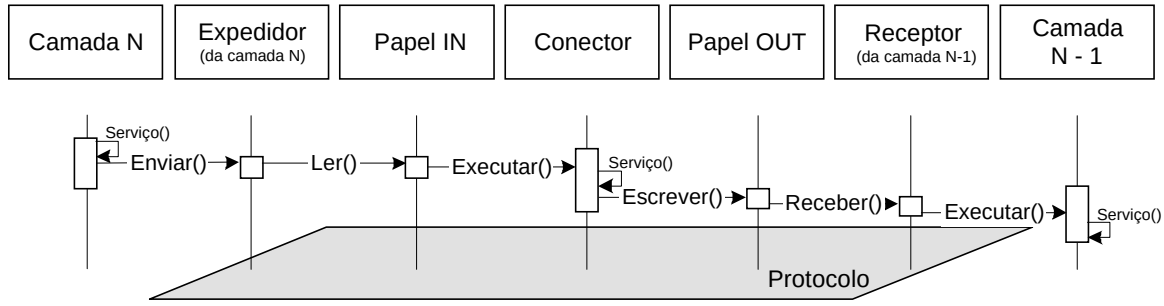

Figura 12: Estrutura dinâmica do estilo camadas

Uma camada não conhece suas camadas adjacentes. O conector pode conhecer que camadas estão em suas extremidades, Papel IN e Papel OUT. Esta informação é gerenciada pela configuração. A configuração deve saber que serviços determinada camada está apta a fornecer ou requisitar, e permitir que conexões sejam estabelecidas somente entre camadas que fornecem-requisitam serviços, e que camadas são adjacentes.

Como mostrado no diagrama de seqüências, a linha de vida de cada classe inicia quando ela recebe uma chamada e termina quando conclui sua tarefa. Para a classe camada isto significa que, por exemplo, ao solicitar um serviço ela não aguarda respostas. O diagrama de estados para a classe camada é apresentado a seguir. Seus possíveis estados são Executando ou Requisitando serviços, não aguardando pelo resultado da requisição de serviços de outras camadas (Figura 13).

O diagrama indica que após executar ou/e requisitar algum serviço, a camada chega ao seu estágio final, ou seja, pronto para receber novas requisições. Isto não significa que uma

[5] O tipo de comunicação mais utilizado é o push, onde as camadas requisitam serviços a camadas mais baixas, qualquer informação é passada como parte do serviço chamado. Este é o modelo especificado neste trabalho [5].

camada não possui nenhuma atividade interna enquanto aguarda requisições, mas sim que sob uma visão externa a camada está simplesmente pronta para receber outras requisições.

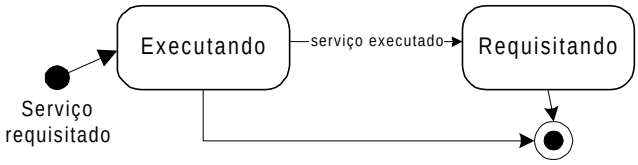

Figura 13: Estrutura dinâmica do estilo camadas

A elaboração do meta-modelo aqui apresentado está em andamento. Definiremos, ainda, regras para a sua utilização, as restrições do estilo em OCL, suas variantes e estereótipos para facilitar ainda mais a modelagem, o entendimento e a comunicabilidade de arquiteturas baseadas no estilo camadas. O intuito não é só catalogar o estilo camadas e sim fornecer subsídios para o desenvolvimento de ferramentas que dêem supote ao desenvolvimento de sistemas baseados no estilo camadas e, futuramente, em outros estilos.

7 – Conclusões e Trabalhos Futuros

Este trabalho une tecnologias amplamente utilizadas na engenharia de software, os estilos e visões arquiteturais e a UML. Propomos um modelo para especificação de arquiteturas em camadas. O estudo realizado levou-nos a reconhecer nesta união uma prática promissora para tornar os estilos mais utilizados por projetistas e a UML mais utilizada pelos arquitetos de software.

Esta união significa parte da automação da utilização de estilos no desenvolvimento de sistemas pois, com o uso da UML transferem-se todos os benefícios desta linguagem para o uso de padrões arquiteturais. Por exemplo, ferramentas CASE, ferramentas e técnicas de análise, integração dos modelos de *design* às linguagens de programação e linguagens de descrição de arquiteturas, dentre outros.

Nosso meta-modelo descreve o estilo camadas de maneira a torná-lo mais facilmente utilizado pelos projetistas visto que, ele retrata as restrições, características, operações e estrutura básica de um sistema em camadas. Como continuidade ao nosso trabalho nos empenharemos em: definir regras para utilização do modelo, definir as restrições do estilo em OCL, definir estereótipos para facilitar ainda mais a modelagem, expandir o meta-modelo para que ele atenda também as variantes do estilo camadas e o aplicaremos em alguns estudos de caso para atestar sua validade.

Em longo prazo, muitos outros trabalhos podem ser realizados: descrever outros estilos como o definido aqui, definir uma metodologia para a utilização deles tanto na engenharia quanto na reengenharia, aplicá-lo e testá-lo em vários contextos, especificá-lo formalmente, definir arquiteturas de referência e arquiteturas de linha de produto. Tudo isto é indispensável para se obter uma biblioteca de estilos inteiramente modelados e prontos para serem instanciados.

8 - Referências Bibliográficas

[1] JACOBSON, Ivar, GRISS, Martin and JONSSON, Patrick: Software Reuse, Addison Wesley, 1997.

[2] HOFMEISTER Christine, NORD Robert, SONI Dilip: Applied Software Architecture, Addison Wesley, 2000.
[3] SHAW Mary, GARLAN David: Software Architecture – Perspectives on an Emerging Discipline, Prentice-Hall, 1996.
[4] GARLAN David, SHAW Mary: An Introduction to Software Architecture, CMU Software Engineering Institute Technical Report – CMU/SEI-94-TR-21, january 1994.
[5] BUSCHMANN Frank, MEUNIER Regine, ROHNERT Hans, SOMMERLAD Peter, STAL Michael: A system of patterns – Pattern-Oriented Software Architecture, Wiley, 1996.
[6] BOSCH, Jan: Design & Use of Software Architectures, Addison Wesley, 2000.
[7] KLEIN, Mark e KAZMAN, Rick: Attribute-based Architectural Styles,CMU/SEI-99-TR-022, 1999.
[8] BOOCH, Grady, RUMBAUGH, James e JACOBSON, Ivar: The Unified Modeling Language User Guide, Addison Wesley, 1999.
[9] MARANHÃO Dina: Integração da Linguagem de descrição de arquitetura software ZCL com UML, Relatório de graduação, UFRN, junho, 2000.
[10] ROBBINS Jason, REDMILES David, ROSEMBLUM David: Integrating C2 with the Unified Modeling Language, From the Proceedings of the 1997 California Software Symposium, Invirne, CA, November 1997.
[11] CLEMENTS Paul: A survey of architecture description languages. Eighth International Workshop on Software Specification and Design, Germany, march 1996.
[12] MEDVIDOVIC Neno: A classification and comparision framework for software architecture description languages, Technical report UCI-ICS-97-02, University of California, 1997.
[13] SILVA, Lyrene F. e PAULA, Virgínia C.C.: A Arquitetura de Software e a UML, I Workshop Técnico-Científico – 15 Aos Dimap, UFRN, Brazil,2000.
[14] SILVA, Lyrene F. e PAULA, Virgínia C.C.: Arquitetura em Camadas para o Desenvolvimento Baseado em Componentes, Workshop sobre Desenvolvimento Baseado em Componentes, Maringá-PR, Junho, 2001.
[15] OMG; A discussion of the object management architecture, www.omg.org
[16] OMG; The object model, www.omg.org
[17] CARDOSO, Gustavo D., CARDOSO, Mércia e FRANCIOLI: Capítulo 9 – O padrão Corba: uma arquitetura baseada em objetos distribuídos, www.nautilus.com.br/~francioli/
[18] FARIAS, Cléver R.G., SINDEREN, Marten V. and PIRES, Luís Ferreira: A Systematic Approach for Component-Based Software Development, In proceeding of the seventh European Concurrent Engineering Conference, Leicester-United Kingdom, pp.127-131, april/2000.