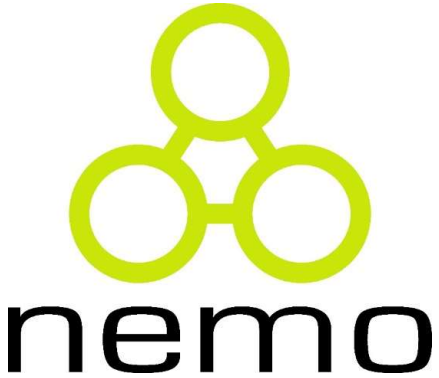


Projeto de Sistemas de Software

Jordana S. Salamon

jssalamon@inf.ufes.br

jordanasalamon@gmail.com

DEPARTAMENTO DE INFORMÁTICA

CENTRO TECNOLÓGICO

UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

Projeto da Lógica de Negócio

- A camada de lógica de negócio engloba o conjunto de classes que vai realizar toda a lógica do sistema de informação.

- As demais camadas são derivadas ou dependentes dessa camada e, portanto, é interessante iniciar o projeto dos componentes da arquitetura do sistema por ela.

- **Os modelos construídos na fase de análise são os principais insumos para o projeto dessa camada, em especial o modelo conceitual estrutural e o modelo de casos de uso.**

Projeto da Lógica de Negócio

- Os diagramas de classes da fase de análise são a base para a construção dos diagramas de classes da fase de projeto.
- A versão inicial do modelo estrutural de projeto (diagramas de classes de projeto) da lógica de negócio é uma cópia do modelo conceitual estrutural.
- Durante o projeto, esse modelo será refinado, visando incorporar informações importantes para a implementação, tais como distribuição de responsabilidades entre as classes (definição de métodos das classes) e definição de navegabilidades e visibilidades.

Projeto da Lógica de Negócio

- Além disso, alterações na estrutura do diagrama de classes podem ser necessárias para tratar requisitos não funcionais, tais como usabilidade e desempenho.

- Para organizar a lógica de negócio, um bom ponto de partida são os padrões arquitetônicos relativos a essa camada.

- A literatura apresenta quatro padrões: Script de Transação (Transaction Script), Modelo de Domínio (Domain Model), Módulo de Tabelas (Table Module) e Camada de Serviço (Service Layer).

Projeto da Lógica de Negócio

- No contexto do desenvolvimento de Sistemas de Informação Orientados a Objetos, merecem destaque os padrões Modelo de Domínio e Camada de Serviço.

- Uma questão bastante importante tratada por esses padrões é a distribuição de responsabilidades ao longo das classes que compõem o sistema.

- No mundo de objetos, uma funcionalidade é realizada através de uma rede de objetos interconectados, colaborando entre si. Objetos encapsulam dados e comportamento.

Projeto da Lógica de Negócio

- O posicionamento correto do comportamento na rede de objetos é um dos principais problemas a serem enfrentados durante o projeto da lógica de negócio.

- Neste contexto, pode-se observar que a lógica de negócio é, na verdade, composta por dois tipos de lógica: a lógica de domínio do problema, que tem a ver puramente com as classes previamente identificadas na fase de análise; e lógica da aplicação, que se refere às funcionalidades descritas pelos casos de uso.

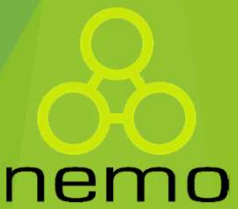
- Neste contexto, um desafio a mais se coloca: **que classes vão comportar as funcionalidades descritas pelos casos de uso (lógica de aplicação)?**

Projeto da Lógica de Negócio

- Duas formas básicas são comumente adotadas:

- • Distribuir as responsabilidades para a execução dos casos de uso ao longo dos objetos do domínio do problema;

- Essa abordagem é refletida no padrão **Modelo de Domínio** e considera que as funcionalidades relativas aos casos de uso do sistema (lógica de aplicação) estarão *distribuídas nas classes previamente identificadas na fase de análise*.



Projeto da Lógica de Negócio

- • Considerar que a lógica de negócio é, na verdade, composta por dois componentes:
- O Componente de Domínio do Problema, que tem a ver puramente com as classes previamente identificadas na fase de análise e que trata apenas da lógica de domínio;
- E o Componente de Gerência de Tarefas, que se refere às funcionalidades descritas pelos casos de uso e trata, portanto, da lógica de aplicação.
- Esta segunda opção é a essência do padrão **Camada de Serviço**.

Diagramas de Interação

- Seja qual for a opção escolhida, para apoiar a definição dos métodos das classes, pode ser útil elaborar **diagramas de interação**.

- Os diagramas de interação da UML são utilizados para modelar aspectos dinâmicos dos sistemas, com ênfase na distribuição de responsabilidades entre classes e no fluxo de controle ao longo das classes.

- A modelagem de aspectos dinâmicos de sistemas pode ser feita construindo-se *roteiros de cenários* (p.ex., fluxos de eventos de casos de uso, fragmentos deles ou operações) que envolvem a interação entre certos objetos de interesse e as mensagens trocadas entre eles. Na UML, a modelagem desses roteiros é feita com o uso dos **diagramas de interação.**

Diagramas de Interação

- Tipicamente, um diagrama de interação ilustra o *comportamento de um grupo de objetos colaborando para a realização de um dado caso de uso*, mostrando um número de instâncias concretas ou prototípicas de classes e as mensagens que são trocadas entre elas no contexto do caso de uso.

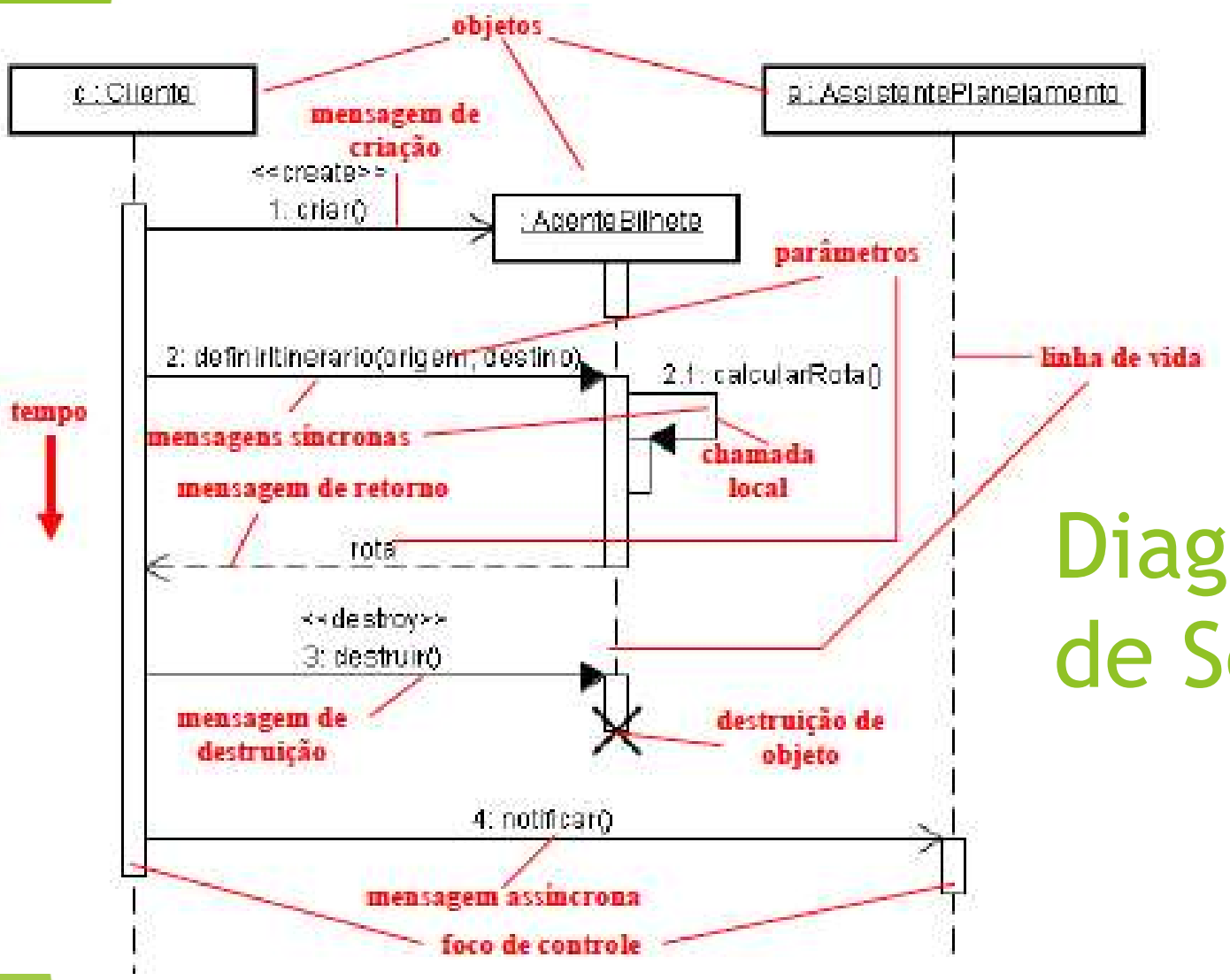
- Há dois tipos de diagramas de interação: **diagramas de sequência** e **diagramas de comunicação**.

Diagramas de Interação

- Os diagramas de sequência dão ênfase à ordenação temporal das mensagens;
- Os diagramas de comunicação dão ênfase à organização estrutural dos objetos.
- Esses dois tipos de diagramas são semanticamente equivalentes. Ou seja, é possível converter um no outro sem perda de informação .

Diagramas de Sequência

- Um diagrama de sequência é um diagrama de interação que dá ênfase à ordenação temporal das mensagens.
- Ele é organizado ao longo de dois eixos. Os objetos que participam da interação são colocados no eixo horizontal, na parte superior do diagrama. O objeto que inicia a interação é colocado à esquerda e os objetos subordinados vão aparecendo à direita.
- Quando um diagrama de sequência modela um cenário de execução de um caso de uso iniciado por um ator, tipicamente uma instância desse ator é o objeto mais à esquerda no diagrama.
- As mensagens que os objetos enviam e recebem são colocadas ao longo do eixo vertical, na ordem cronológica de envio, de cima para baixo. Isso proporciona ao leitor uma clara identificação do fluxo de controle ao longo do tempo.



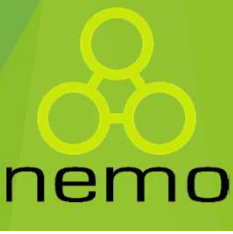
Diagramas de Sequência

Diagramas de Sequência

- Há diferentes tipos de mensagens:

- • Uma **mensagem de criação** indica que o objeto receptor está sendo criado naquele momento.

- Assim, ao contrário dos demais objetos, ele aparece nivelado na altura do envio da mensagem e não na parte superior do diagrama.



Diagramas de Sequência

- • Uma **mensagem síncrona de chamada** invoca uma operação no objeto receptor, passando o controle para esse objeto.
- Um objeto pode enviar uma mensagem para ele mesmo, resultando em uma chamada local de uma operação.

- • Uma **mensagem assíncrona** envia um sinal para o objeto receptor, mas o objeto emissor continua sua própria execução. O objeto receptor recebe o sinal e decide de forma independente o que fazer.

- A representação de mensagens assíncronas é ligeiramente diferente da representação de mensagens síncronas. A primeira tem uma seta fina, enquanto a segunda tem uma seta grossa.

Diagramas de Sequência

- • Uma **mensagem de retorno** indica uma resposta a uma mensagem síncrona (retorno de chamada) e não uma nova mensagem.

- Para diferenciar, mensagens de retorno são simbolizadas por uma linha tracejada com uma seta fina. A mensagem de retorno pode ser omitida, já que há um retorno implícito após qualquer chamada. Contudo, muitas vezes é útil mostrar os valores de retorno.

- • Por fim, uma **mensagem de destruição** elimina um objeto, deixando o mesmo de existir. Assim, mensagens de destruição são acompanhadas do símbolo de destruição de objeto.

Diagramas de Sequência

- O foco de controle mostra o período durante o qual um objeto está desempenhando uma ação, diretamente ou por meio de um procedimento subordinado.
- Ele é mostrado como um retângulo estreito sobre a linha de vida do objeto. A parte superior do retângulo é alinhada com o início da ação; a parte inferior é alinhada com a sua conclusão e pode ser delimitada por uma mensagem de retorno.
- Durante a modelagem do comportamento de um sistema, muitas vezes é importante mostrar fluxos condicionais, laços e a execução concorrente de sequências.
- Para modelar essas situações, operadores de controle podem ser utilizados.

Diagramas de Sequência

- Um operador de controle é apresentado como uma região retangular no diagrama de sequência, contendo um rótulo no canto superior esquerdo, informando o tipo de operador de controle.

- Os tipos mais comuns são:

- • *Execução opcional* (rótulo opt): o corpo do operador de controle é executado somente se a condição de guarda na entrada do operador for verdadeira.

- • *Execução paralela* (rótulo par): o corpo do operador de controle é dividido em sub-regiões por linhas horizontais tracejadas, sendo que cada sub-região representa uma computação paralela (concorrente).

Diagramas de Sequência

- • *Execução alternativa* (rótulo alt): o corpo do operador de controle é dividido em sub-regiões por linhas horizontais tracejadas, sendo que cada sub-região representa um ramo condicional e é executada somente se sua condição de guarda for verdadeira.

- • *Execução iterativa* (rótulo loop): o corpo do operador de controle é executado repetidamente enquanto a condição de guarda na entrada do operador for verdadeira. A condição de guarda é testada no início de cada iteração.

- Mensagens trocadas entre objetos em um diagrama de sequência devem ser mapeadas como operações na classe do objeto receptor da mensagem. Assim, toda mensagem que chega a um objeto aponta para a necessidade de um método com mesma assinatura na classe desse objeto, contribuindo de maneira decisiva para a identificação de métodos nas classes.

Padrões Arquitetônicos para o Projeto da Lógica de Negócio

- Um importante aspecto do projeto da Camada de Lógica de Negócio diz respeito à organização das classes e distribuição de responsabilidades entre elas, o que vai definir, em última instância, os métodos de cada classe dessa camada.

- Pode-se dividir a lógica de negócio em dois tipos principais:

- A **lógica de domínio do problema**, que tem a ver puramente com as classes previamente identificadas na fase de análise;
- A **lógica de aplicação**, que se refere às funcionalidades descritas pelos casos de uso.

- Como consequência, é importante definir onde posicionar os métodos que vão cumprir esses diferentes tipos de responsabilidades em um modelo de classes de projeto.

Padrões Arquitetônicos para o Projeto da Lógica de Negócio

- Fowler (2003) apresenta alguns padrões para organizar a lógica de negócio, a saber: *Script de Transação* (Transaction Script), ***Modelo de Domínio*** (Domain Model), *Módulo de Tabelas* (Table Module) e ***Camada de Serviço*** (Service Layer).

- Em ambos os casos, é útil ter uma classe que vai representar o sistema como um todo, dita uma classe controladora do sistema.
- Essa classe servirá como uma fachada para receber requisições da interface com o usuário e direcioná-las para os objetos capazes de tratá-las.

- Contudo, dependendo do padrão arquitetônico adotado, esse tratamento das requisições será ligeiramente diferente

Padrões Arquitetônicos para o Projeto da Lógica de Negócio

- O padrão **Modelo de Domínio** preconiza que o modelo de classes do domínio incorpore tanto dados quanto comportamento. As classes de domínio identificadas na Análise de Requisitos são as responsáveis por tratar tanto a lógica de domínio quanto a lógica de aplicação (casos de uso).

- Assim, dados e processos são combinados e pode-se dizer que o padrão Modelo de Domínio captura a essência da orientação a objetos.

Padrões Arquitetônicos para o Projeto da Lógica de Negócio

- Já na abordagem do padrão **Camada de Serviço**, um conjunto de classes gerenciadoras de casos de uso fica responsável por tratar a lógica de aplicação, controlando o fluxo de eventos dentro do caso de uso.

- Grande parte da lógica de negócio ainda fica a cargo das classes do domínio do problema, cabendo aos gerenciadores de casos de uso apenas centralizar o controle sobre a execução do caso de uso.

- Assim, a camada de serviço é construída, de fato, sobre a camada de domínio.

Padrões Arquitetônicos para o Projeto da Lógica de Negócio

- A motivação principal para esse padrão é o fato de algumas funcionalidades (casos de uso) não serem facilmente distribuídas nas classes de domínio do problema, principalmente aquelas que operam sobre várias classes.

- Assim, criam-se classes gerenciadoras de casos de uso (gerenciadores ou coordenadores de tarefas), responsáveis pela realização de tarefas (casos de uso).

- Tipicamente, esses gerenciadores de tarefas agem como aglutinadores, unindo outros objetos para dar forma a um caso de uso.

Padrões Arquitetônicos para o Projeto da Lógica de Negócio

- Consequentemente, gerenciadores de tarefa são normalmente encontrados diretamente a partir dos casos de uso.

- Os tipos de funcionalidade tipicamente atribuídos a gerenciadores de tarefa incluem comportamento relacionado a transações e sequências de controle específicas de um caso de uso.

- O padrão *Camada de Serviço* define uma fronteira da lógica de negócio usando uma camada de serviços que estabelece um conjunto de operações disponíveis e coordena as respostas do sistema para cada uma das operações.

- A camada de serviço encapsula a lógica de negócio do sistema, controlando transações e coordenando respostas na implementação de suas operações.

Padrões Arquitetônicos para o Projeto da Lógica de Negócio

- A argumentação em favor desse padrão é que misturar lógica de domínio e lógica de aplicação nas mesmas classes torna essas classes menos reutilizáveis transversalmente em diferentes aplicações, bem como pode dificultar a manutenção da lógica de aplicação, uma vez que a lógica dos casos de uso não é diretamente perceptível em nenhuma classe.

- A identificação das operações necessárias na camada de serviço é fortemente apoiada nos casos de uso do sistema. Uma opção é considerar que cada caso de uso vai dar origem a uma classe de serviços, dita classe gerenciadora de caso de uso.

- Por exemplo, um caso de uso de cadastro, envolvendo funcionalidades de inclusão, alteração, consulta e exclusão, pode ser mapeado em uma classe com operações para tratar essas funcionalidades.

Padrões Arquitetônicos para o Projeto da Lógica de Negócio

- Uma maneira de se implementar o padrão Camada de Serviço consiste em ter uma ou mais classes gerenciadoras de casos de uso, as quais encapsulam a lógica da aplicação.

- Para realizar um caso de uso, a classe gerenciadora de caso de uso invoca métodos da camada de domínio do problema, tal como ocorre no padrão Modelo de Domínio.

- A diferença entre os dois padrões reside, neste caso, no fato da classe gerenciadora de tarefa centralizar o controle do caso de uso, evitando delegar responsabilidades a classes que não têm como tratá-las.

Projeto da Lógica de Domínio do Problema

- No projeto orientado a objetos, os modelos conceituais estruturais (diagramas de classes) produzidos na fase de análise estão diretamente relacionados à lógica de domínio do problema e podem ser incorporados a um Componente de Domínio do Problema (CDP).

- Como ponto de partida para a elaboração do diagrama de classes do CDP, deve-se utilizar uma cópia do diagrama de classes de análise.

- A partir dessa cópia, alterações serão feitas para incorporar as decisões de projeto. Vale ressaltar que o trabalho deve ser efetuado em uma cópia, mantendo o modelo conceitual original intacto para efeito de documentação e manutenção do sistema.

Projeto da Lógica de Domínio do Problema

- Para se poder conduzir o projeto do CDP de maneira satisfatória, algumas informações acerca da plataforma de implementação são essenciais, dentre elas a linguagem de programação e o mecanismo de persistência de objetos a serem adotados.

- Além disso, informações relativas aos requisitos não funcionais e suas prioridades são igualmente vitais para se tomar decisões importantes relativas ao projeto do CDP.

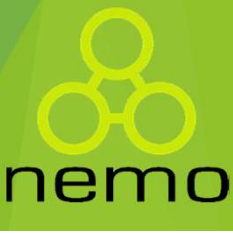
- As alterações básicas a serem incorporadas em um diagrama de classes do CDP são:

Projeto da Lógica de Domínio do Problema

- • *Alteração de informações relativas a tipos de dados de atributos*: Na fase de análise, é comum especificar tipos de dados gerais para atributos.

- Na fase de projeto, contudo, os atributos devem ser mapeados em variáveis de um tipo de dados provido pela linguagem de implementação.

- Além disso, muitas vezes, atributos podem dar origem a novas classes (ou tipos de dados enumerados) para atender a requisitos de qualidade, tais como usabilidade, manutenibilidade e reusabilidade.



Projeto da Lógica de Domínio do Problema

- • *Adição de navegabilidades nas associações*: Na fase de análise, as associações são consideradas navegáveis em todas as direções.

- O mesmo não ocorre na fase de projeto, quando se pode definir que certas associações são navegáveis apenas em um sentido, indicando que apenas um dos objetos terá uma referência para o outro (ou para coleções de objetos, no caso de associações com multiplicidade *).

- Esta decisão deve ser feita, sobretudo, considerando os usos frequentes das funcionalidades do sistema e requisitos não funcionais, com destaque para desempenho, mesmo que ele não seja um atributo condutor da arquitetura.

- Além disso, essa definição pode ser influenciada, dentre outros, pelo mecanismo de persistência de objetos a ser adotado.

Projeto da Lógica de Domínio do Problema

- • *Adição de informações de visibilidade de atributos e associações*: De maneira geral, na fase de análise não se especifica a visibilidade de atributos e associações.

- Na fase de análise, as associações são tipicamente consideradas navegáveis e visíveis em todas as direções. Já os atributos são considerados públicos. Porém essa não é uma boa estratégia para a fase de projeto, já que ocultar informações é um importante princípio de projeto.

- Assim, atributos só devem poder ser acessados pela própria classe ou por suas subclasses. Uma classe não deve ter acesso aos atributos de uma classe a ela associada.

Projeto da Lógica de Domínio do Problema

- • *Adição de métodos às classes*: Muitas vezes, as classes de um diagrama de classes de análise não têm informação acerca das suas operações. Mesmo quando elas têm essa informação, ela pode ser insuficiente, tendo em vista que é no projeto que se decide efetivamente como abordar a distribuição de responsabilidades para a realização de funcionalidades.

- Assim, durante o projeto do CDP atenção especial deve ser dada à definição de métodos nas classes. Para apoiar esta etapa, diagramas de sequência podem ser utilizados para modelar a interação entre objetos na realização de funcionalidades do sistema.

- A escolha de um padrão arquitetônico para o projeto da lógica de negócio também tem influência na distribuição de responsabilidades.

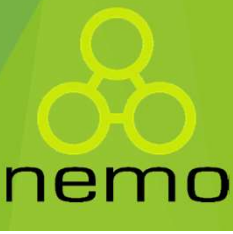
nemo

Projeto da Lógica de Domínio do Problema

- • *Eliminação de classes associativas*: Caso o diagrama de classes de análise contenha classes associativas, recomenda-se substituí-las por classes normais, criando novas associações.

- Isso é importante, pois as linguagens de programação não têm construtores capazes de implementar diretamente esses elementos de modelo.

- Além das alterações básicas a que todos os diagramas de classes do CDP estarão sujeitos, outras fontes de alteração incluem:



Projeto da Lógica de Domínio do Problema

- • *Reutilizar projetos anteriores e classes já programadas*: é importante que na fase de projeto seja levada em conta a possibilidade de se reutilizar classes já projetadas e programadas (desenvolvimento com reúso), bem como a possibilidade de se desenvolver classes para reutilização futura (desenvolvimento para reúso).

- Tipicamente, ajustes feitos para incorporar tais classes envolvem alterações na estrutura do modelo, podendo atingir hierarquias de generalização-especialização, de modo a tratar as classes do domínio do problema como subclasses de classes de bibliotecas pré-existentes.

- Também ao incorporar um padrão de projeto (design pattern), muito provavelmente a estrutura do diagrama de classes de projeto sofrerá alterações.

Projeto da Lógica de Domínio do Problema

- • *Ajustar hierarquias de generalização-especialização*: muitas vezes, as hierarquias de herança da fase de análise não são adequadas para a fase de projeto.

- Dentre os fatores que podem provocar mudanças na hierarquia de herança destacam-se:

- o Ajustar hierarquias de generalização-especialização para adequação ao mecanismo de herança suportado pela linguagem de programação a ser usada na implementação: se, por exemplo, o modelo de análise envolve herança múltipla e a linguagem de implementação não oferece tal recurso, alterações no modelo são necessárias.

Projeto da Lógica de Domínio do Problema

- o Ajustar hierarquias de generalização-especialização para aproveitar oportunidades decorrentes da definição de operações: as definições de operações nas classes podem também conduzir a alterações na hierarquia de generalização-especialização.

- De fato, pode ser que durante a fase de análise não sejam exploradas todas as oportunidades de herança. É útil reexaminar o diagrama de projeto procurando observar se determinadas classes têm comportamento parcialmente comum, abrindo-se espaço para a criação de uma superclasse encapsulando as propriedades (atributos e operações) compartilhadas, abstraindo o comportamento comum.

Projeto da Lógica de Domínio do Problema

- A criação de interfaces também pode ser interessante para garantir uma separação da interface contratual de uma classe de sua implementação.

- A reutilização pode ser um fator motivador para a criação de novas superclasses. Contudo, deve-se tomar cuidado com a refatoração da hierarquia de classes.

- Criar uma nova classe para abstrair comportamento comum somente se justifica quando há, de fato, uma relação de subtipo entre as classes existentes e a nova classe criada; ou seja, pode-se dizer que a subclasse é semanticamente um subtipo da superclasse. **Não se deve alterar a hierarquia de classes simplesmente para herdar uma parte do comportamento, quando as classes envolvidas não guardam entre si uma relação efetivamente de subtipo.**

Projeto da Lógica de Domínio do Problema

- • *Ajustar o modelo para melhorar o desempenho*: Visando melhorar o desempenho do sistema, o projetista pode alterar o diagrama de classes do CDP para melhor acomodar os ajustes necessários. Atributos e associações redundantes podem ser adicionados para evitar recomputação, bem como podem ser criadas novas classes para registrar estados intermediários de um processo.

- • *Ajustar o modelo para facilitar o projeto de interfaces com o usuário amigáveis*: com o objetivo de incorporar o atributo de qualidade usabilidade, pode ser importante considerar novas classes (ou tipos enumerados de dados) que facilitem a apresentação de listas para seleção do usuário.

Projeto da Lógica de Domínio do Problema

- • *Ajustar o modelo para incorporar aspectos relacionados à segurança*: táticas como autenticação e autorização requerem novas funcionalidades que, por sua, vez, requerem novas classes do CDP. Em casos como esse, pode ser útil separar as classes relativas a essas funcionalidades em um novo pacote, visando ao reúso.

- O CDP pode ser alterado, ainda, para comportar outros requisitos não funcionais, tais como testabilidade, confiabilidade etc.

- O CDP é um componente obrigatório, tanto quando se adota o padrão Modelo de Domínio quanto quando se adota o padrão Camada de Serviço. No padrão Modelo de Domínio, o CDP é a própria camada de lógica de negócio, tendo em vista que não há classes gerenciadoras de tarefas (gerenciadoras de casos de uso). No caso do padrão Camada de Serviço, além do CDP, a camada de lógica de negócio tem outro componente, o *Componente de Gerência de Tarefas*.

Projeto da Lógica de Aplicação – Padrão Modelo de Domínio

- Wazlawick (2004) propõe uma maneira de projetar a lógica de aplicação usando o padrão Modelo de Domínio, que consiste em considerar **uma** classe controladora do sistema no modelo de domínio do problema, relacionando-a a todos os conceitos independentes, correspondentes aos cadastros do sistema, ou seja, os elementos a serem cadastrados para a operação do sistema.

- O fluxo de controle sempre inicia em uma instância da classe controladora. Essa classe recebe as requisições da interface e, para tratá-las, o controlador invoca métodos que tratam a lógica de aplicação posicionados nas classes do domínio do problema, em uma abordagem chamada de delegação.

Projeto da Lógica de Aplicação – Padrão Modelo de Domínio

- A delegação consiste em capturar uma operação que trata uma mensagem em um objeto e reenviar essa mensagem para um outro objeto associado ao primeiro que seja capaz de tratar a requisição.

- Neste caso, a execução é delegada para objetos do domínio do problema, procurando efetuar uma cadeia de delegação sobre as linhas de visibilidade das associações já existentes, até se atingir um objeto capaz de atender à requisição.

- Novas linhas de visibilidade só devem ser criadas quando isso for estritamente necessário. Deste modo, mantém-se fraco o acoplamento entre as classes.

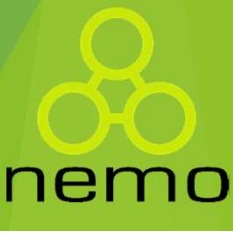
Projeto da Lógica de Aplicação – Padrão Modelo de Domínio

- Assim, partindo-se do controlador do sistema, deve-se identificar qual o objeto a ele relacionado que melhor pode tratar a requisição e delegar essa responsabilidade a ele.

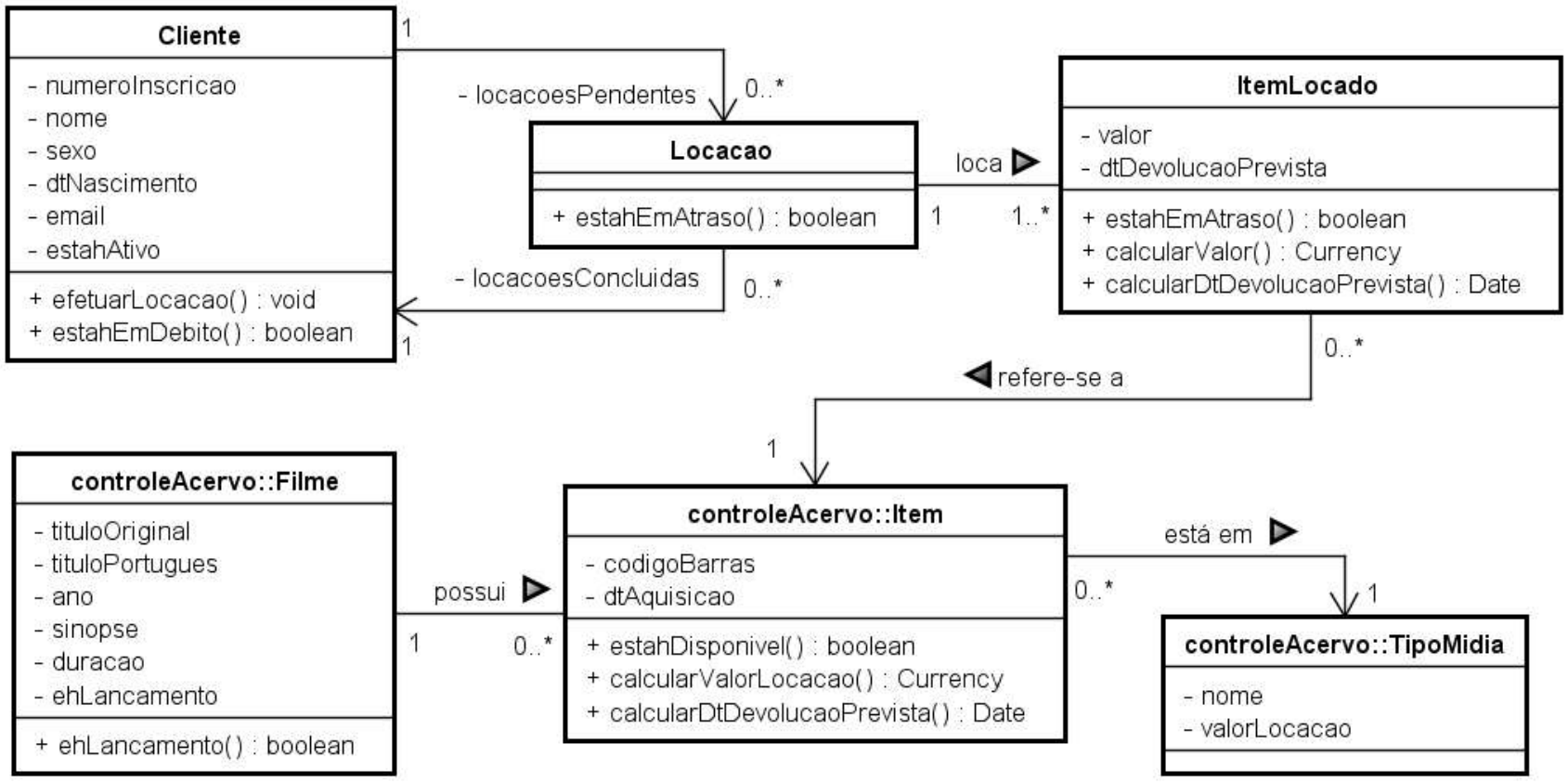
- Esse objeto, por sua vez, vai colaborar com outros objetos para a realização da funcionalidade.

- De maneira geral, um objeto só deve mandar mensagens para outros objetos que estejam a ele associados ou que foram passados como parâmetro no método que está sendo executado.

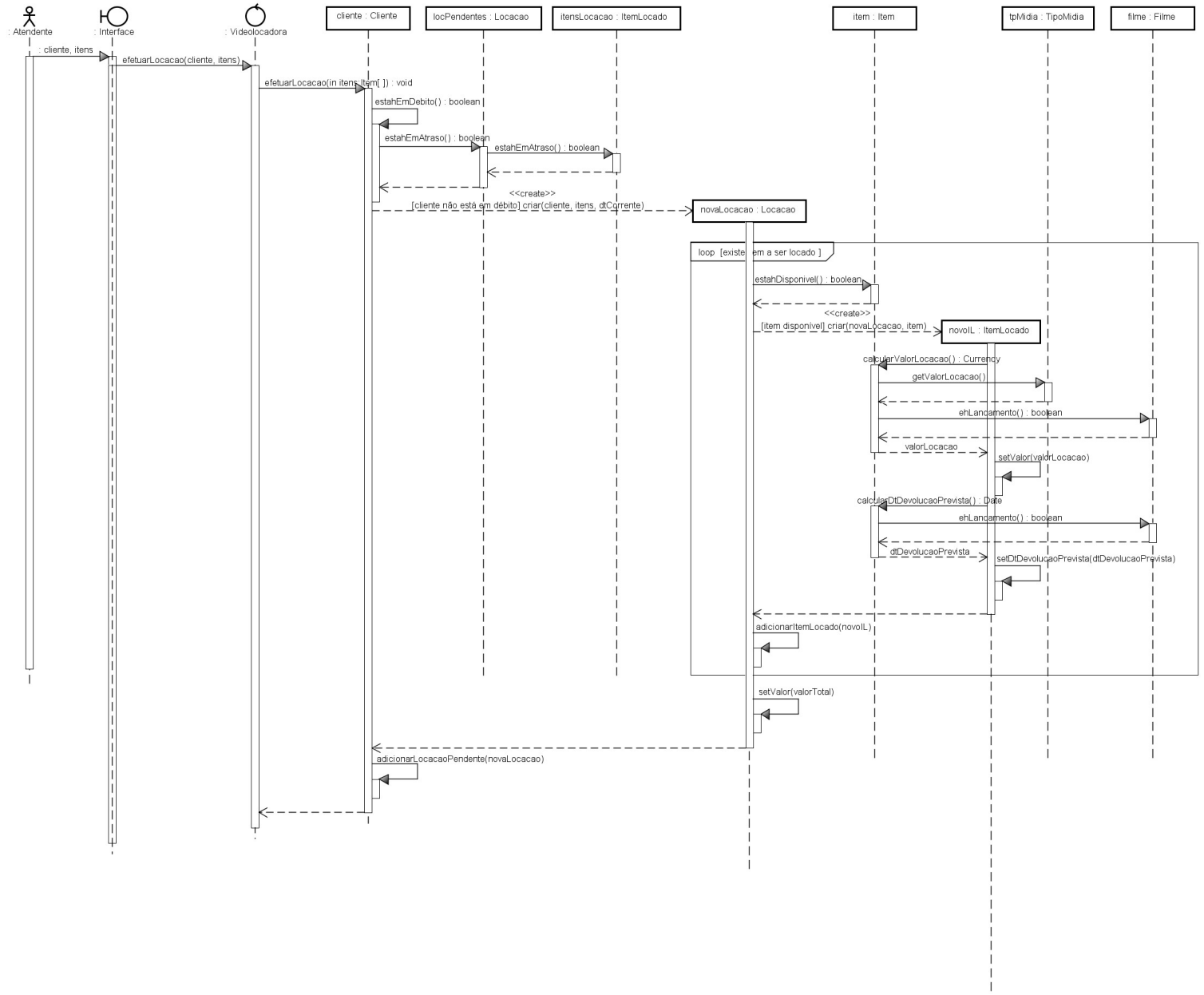
- Nessa abordagem, deve-se evitar obter um objeto como retorno de um método (visibilidade local) para mandar mensagens a ele.



Projeto da Lógica de Aplicação – Padrão Modelo de Domínio



nemo


: Atendente
: Interface
: Videolocadora
cliente : Cliente
locPendentes : Locacao
itensLocacao : ItemLocado
item : Item
tpMidia : TipoMidia
filme : Filme
: cliente, itens
efetuarLocacao(cliente, itens)
efetuarLocacao(in itens:Item[ ]) : void
estahEmDebito() : boolean
estahEmAtraso() : boolean
estahEmAtraso() : boolean
<<create>>
[cliente não está em débito] criar(cliente, itens, dtCorrente)
novaLocacao : Locacao
loop [existe item a ser locado]
estahDisponivel() : boolean
<<create>>
[item disponível] criar(novaLocacao, item)
novoIL : ItemLocado
calcularValorLocacao() : Currency
getValorLocacao()
ehLancamento() : boolean
valorLocacao
setValor(valorLocacao)
calcularDtDevolucaoPrevista() : Date
ehLancamento() : boolean
dtDevolucaoPrevista
setDtDevolucaoPrevista(dtDevolucaoPrevista)
adicionarItemLocado(novoIL)
setValor(valorTotal)
adicionarLocacaoPendente(novaLocacao)

Projeto da Lógica de Aplicação – Padrão Camada de Serviço

- No padrão Camada de Serviço, um novo componente, o Componente de Gerência de Tarefas (CGT), fica responsável por tratar a lógica de aplicação, controlando os fluxos de eventos dos casos de uso.

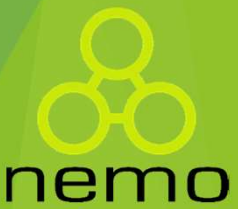
- Esse componente define classes gerenciadoras de tarefas (ou gerenciadoras de casos de uso).

- Quando o padrão Camada de Serviço é aplicado, a camada de lógica de negócio é dividida em duas outras camadas. A Camada de Domínio do Problema (CDP) e, sobre ela, a Camada de Gerência de Tarefa (CGT).

Projeto da Lógica de Aplicação – Padrão Camada de Serviço

- Em um esboço preliminar, pode-se atribuir um gerenciador de tarefas para cada caso de uso, sendo que os seus fluxos de eventos principais dão origem a operações da classe que representa o caso de uso (classe gerenciadora de caso de uso ou classe de aplicação).

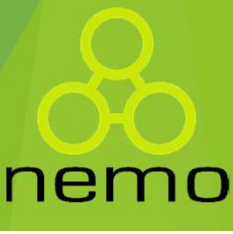
- Deste modo, a manutenibilidade pode ser facilitada, uma vez que, detectado um problema em um caso de uso, é fácil identificar a classe que trata do mesmo.



Projeto da Lógica de Aplicação – Padrão Camada de Serviço

- Uma solução diametralmente oposta consiste em definir uma única classe de aplicação para todo o sistema. Neste caso, os fluxos de eventos de todos os casos de uso dão origem a operações dessa classe.

- Fica evidente que, exceto para sistemas muito pequenos, essa classe tende a ter muitas operações e será extremamente complexa e, portanto, essa opção tende a não ser prática.



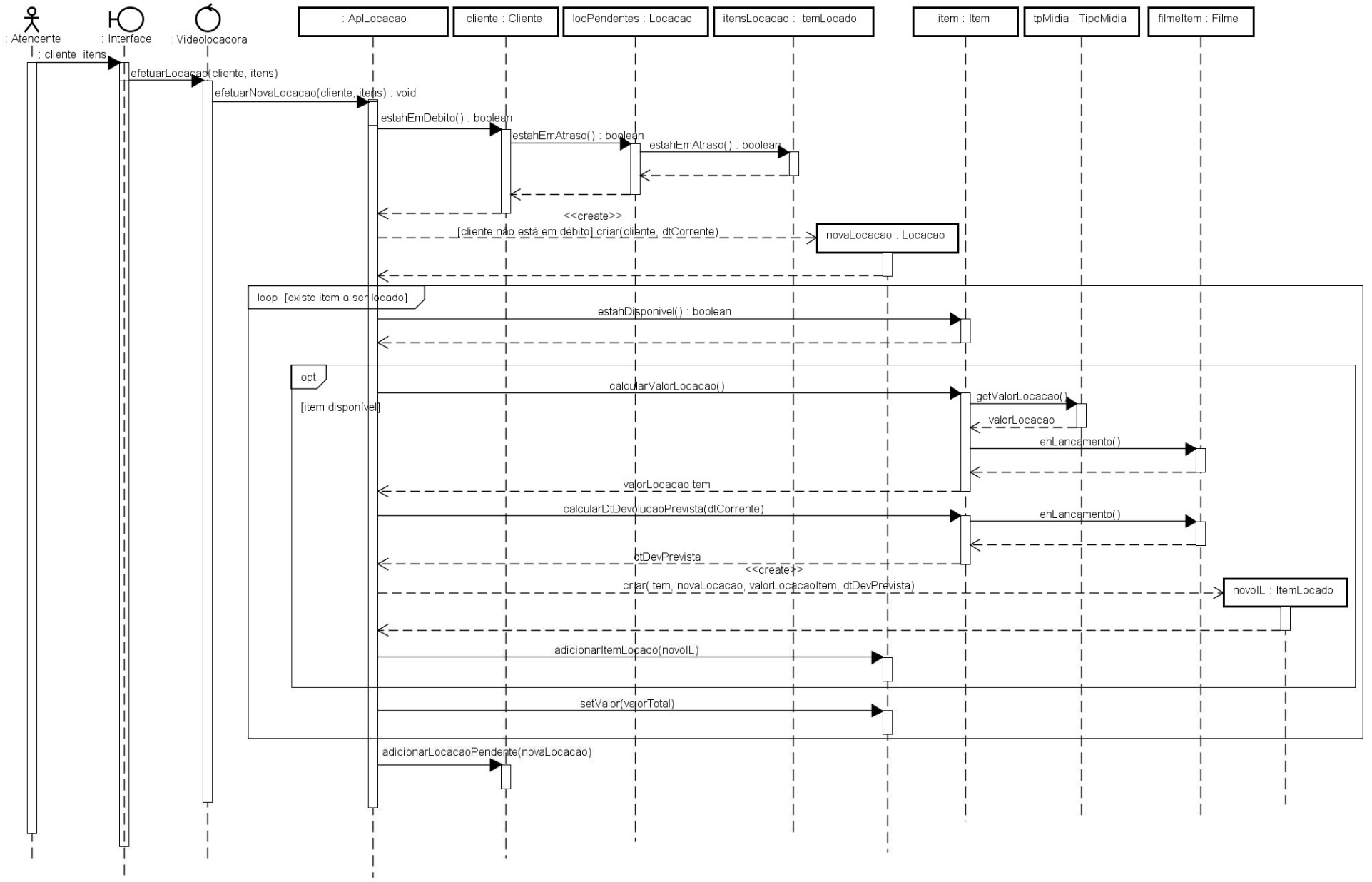
Projeto da Lógica de Aplicação – Padrão Camada de Serviço

- Normalmente, uma solução intermediária entre as duas anteriormente apresentadas conduz a melhores resultados.

- Nessa abordagem, casos de uso complexos são designados a classes de gerência de tarefas específicas. Casos de uso mais simples e de alguma forma relacionados são tratados por uma mesma classe de gerência de tarefas.

- No caso da aplicação do padrão Camada de Serviço, não há restrições de que a classe gerenciadora de tarefa obtenha um objeto como retorno de um método (visibilidade local) e mande mensagens a ele.

- Assim, a classe gerenciadora de tarefa pode ter referência a diversos objetos do domínio, tipicamente aqueles envolvidos na realização do caso de uso correspondente.



Projeto da Lógica de Aplicação – Padrão Camada de Serviço

- O conjunto de tarefas a serem apoiadas pelo sistema oferece um recurso bastante útil para a definição das janelas, menus e outros componentes de interface com o usuário necessários para cada uma dessas tarefas.

- Assim os projetos dos componentes de gerência de tarefa e de apresentação estão bastante relacionados e devem ser realizados conjuntamente, uma vez que, muitas vezes, são as tarefas que determinam a necessidade de elementos de interface com o usuário para sua execução.

That’s all Folks!