repositórios

Em um projeto orientado a modelo, os objetos têm um ciclo de vida que começa com a criação e terminando com exclusão ou arquivamento. Um construtor ou uma fábrica cuida de criação do objeto. Todo o propósito de objetos criando é usá-los. Em uma linguagem orientada a objetos, é preciso manter uma referência a um objeto, a fim de ser capaz de usá-lo. Para ter referência tal, o cliente deve criar o objeto ou obtê-la a partir de outro, atravessando uma associação existente. Por exemplo, para obter um objeto de valor de um agregado, o cliente deve solicitá-lo a partir da raiz do agregado. O problema agora é que o cliente deve ter uma referência para a raiz. Para grandes aplicações, isso se torna um problema porque é preciso certificar-se de que o cliente tem sempre uma referência para o objeto necessário, ou para outro que tem uma referência para o respectivo objecto. Usando essa regra no projeto irá forçar os objetos para segurar uma série de referências que eles provavelmente não iria manter o contrário. Isso aumenta o acoplamento, criando uma série de associações que não são realmente necessários.

Para usar um meio objeto o objeto já foi criado. Se o objeto é a raiz de um agregado, então é uma entidade, e as chances são de que vai ser armazenado em um estado persistente em um banco de dados ou outra forma de persistência. Se for um objeto de valor, pode ser obtido a partir de uma Entidade atravessando uma associação. Acontece que uma grande quantidade de objetos podem ser obtidos diretamente do banco de dados. Isto resolve o problema de obter referência de objetos. Quando um cliente quer usar um objeto, ele acessa o banco de dados, recupera o objeto dele e usa-lo. Esta parece ser uma solução rápida e simples, mas tem impactos negativos sobre o design.

Bancos de dados são parte da infra-estrutura. A má solução é para que o cliente estar ciente dos detalhes necessários para acessar um banco de dados. Por exemplo, o cliente tem que criar consultas SQL para recuperar os dados desejados. A consulta de banco de dados pode retornar um conjunto de registros, expondo ainda mais de seus detalhes internos. Quando muitos clientes têm de criar objetos diretamente do banco de dados, verifica-se que

tal código está espalhado por todo o domínio. Nesse ponto, o modelo de domínio torna-se comprometida. Ele tem que lidar com muitos detalhes de infra-estrutura, em vez de lidar com conceitos do domínio. O que acontece se uma decisão é feita para alterar o banco de dados subjacente? Tudo o que necessita de código dispersos de ser alterado para ser capaz de acessar o novo armazenamento. Quando o código do cliente acessa um banco de dados diretamente, é possível que ele irá restaurar um objeto interno para um agregado. Isso quebra o encapsulamento do agregado com consequências desconhecidas.

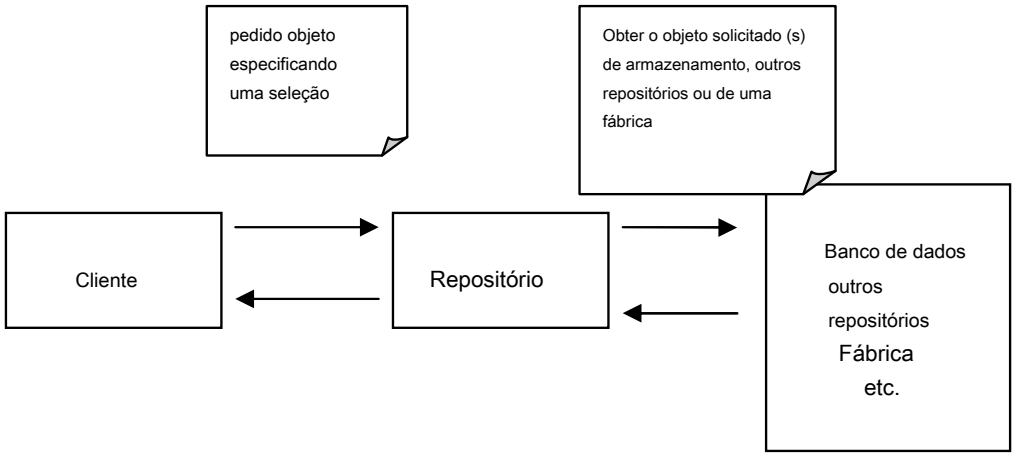
Um cliente precisa de um meio prático de adquirir referências a preexistente objetos de domínio. Se a infra-estrutura torna mais fácil para fazer isso, os desenvolvedores do cliente pode adicionar associações mais traversable, atrapalhando o modelo. Por outro lado, eles podem usar consultas para puxar os dados exatos de que precisam a partir do banco de dados, ou para puxar objetos alguns específico em vez de navegar a partir de raízes de agregação. Domínio lógica move-se em consultas e código de cliente, e as entidades e objetos de valor se tornam meros recipientes de dados. A complexidade técnica pura de aplicação de mais infra-estrutura de acesso à base de dados rapidamente inunda o código do cliente, o que leva os desenvolvedores a dumb-se a camada de domínio, o que torna o modelo irrelevante. O efeito geral é que o foco de domínio está perdido e que o projeto é comprometida.

Portanto, usar um repositório, cuja finalidade é encapsular toda a lógica necessária para obter referências de objeto. Os objetos de domínio não terá que lidar com a infra-estrutura para obter as referências necessárias para outros objetos do domínio. Eles só vão levá-los a partir do Repositório eo modelo é recuperar sua clareza e foco.

O Repositório pode armazenar referências a alguns dos objetos. Quando um objeto é criado, ele pode ser salvo no repositório, e recuperados de lá para ser usado mais tarde. Se o cliente solicitou um objeto a partir do Repositório, eo repositório não tê-lo, pode obtê-lo a partir do armazenamento. De qualquer maneira, o Repositório atua como um local de armazenamento para objetos globalmente acessíveis.

O Repositório pode também incluir uma estratégia. Pode aceder a um armazenamento de persistência ou outra baseada na Estratégia especificado. isto

podem utilizar diferentes locais de armazenamento para diferentes tipos de objetos. O efeito geral é que o modelo de domínio é dissociado da necessidade de armazenar objetos ou suas referências, e acessar a infra-estrutura de persistência subjacente.



Para cada tipo de objeto que precisa de acesso global, criar um objeto que pode fornecer a ilusão de uma coleção na memória de todos os objetos desse tipo. Configurar o acesso através de uma interface global, bem conhecido. Fornecer métodos para adicionar e remover objectos, os quais irão encapsular a inserção real ou remoção de dados no armazenador de dados. Fornecer métodos que selecionar objetos com base em alguns critérios e retornam objetos totalmente instanciado ou coleções de objetos cujos valores atributo cumprir os critérios,

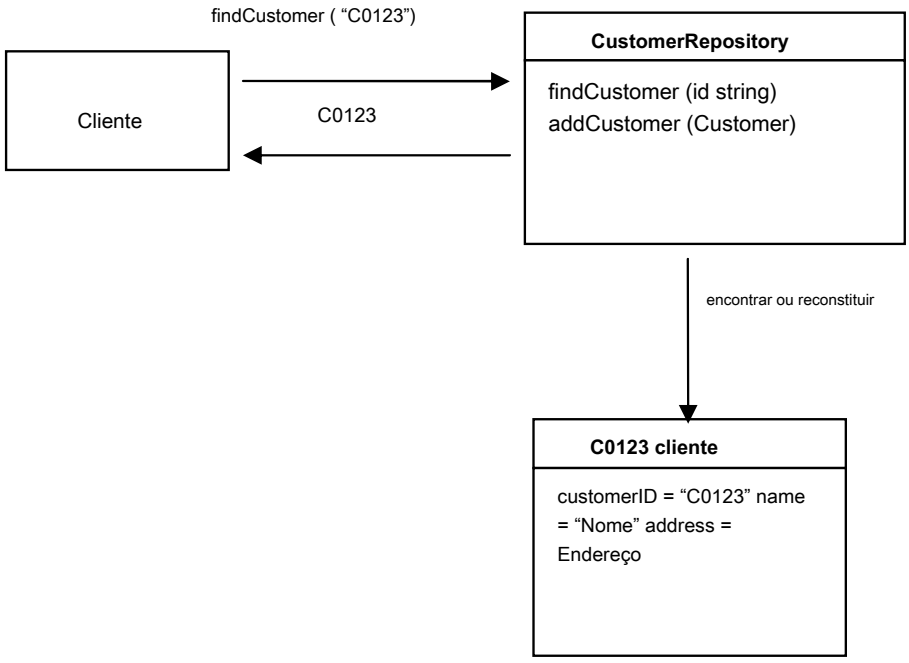
assim

encapsular o armazenamento real e consulta tecnologia. Fornecer repositórios somente para raízes agregado que realmente precisam de acesso direto. Manter o cliente focado no modelo, delegando todo o armazenamento de objetos e acesso aos Repositórios.

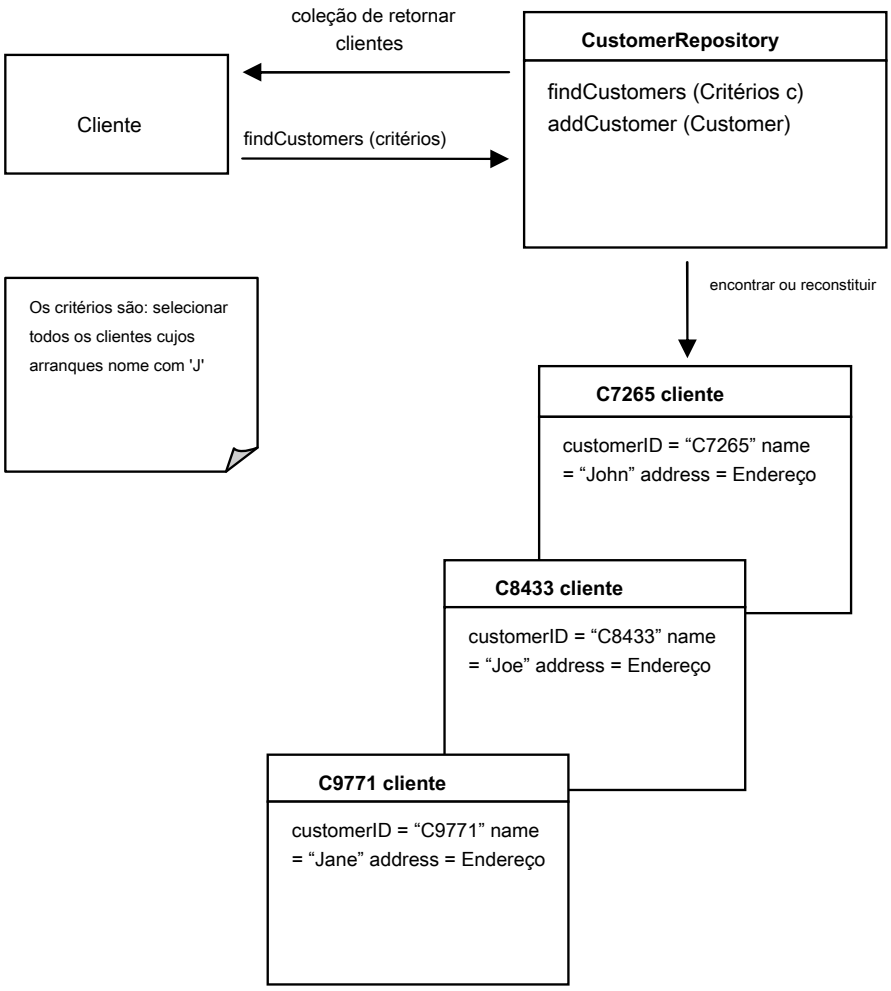
Um repositório pode conter informações detalhadas usadas para o acesso a infra-estrutura, mas sua interface deve ser simples. Um repositório deve ter um conjunto de métodos usados para recuperar objetos. O cliente chama esse método uma e passa um ou mais

parâmetros que representam os critérios de selecção utilizados para seleccionar um objecto ou um conjunto de objectos correspondentes. Uma entidade pode ser facilmente especificado, passando a sua identidade. Outros critérios de seleção podem ser composta por um conjunto de atributos de objeto. O Repositório irá comparar todos os objetos contra esse conjunto e irá retornar aqueles que satisfazem os critérios. A interface Repositório podem conter métodos utilizados para a realização de alguns cálculos suplementares, tais como o número de objectos de um determinado tipo.

Pode-se notar que a implementação de um repositório pode estar intimamente gostava da infra-estrutura, mas que a interface de repositório será modelo de domínio puro.

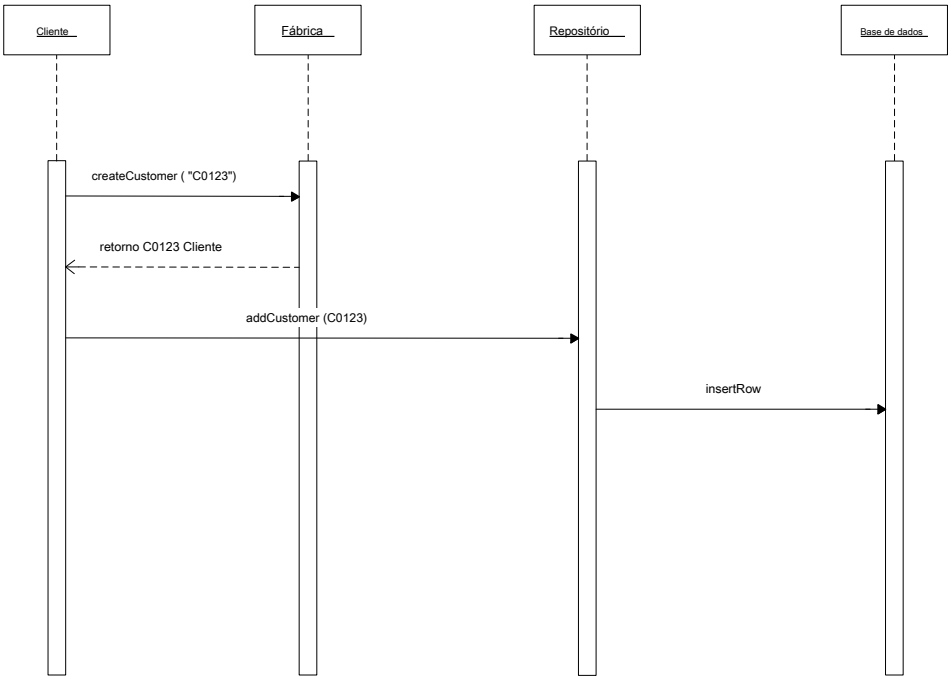


Outra opção é especificar um critério de seleção como uma especificação. A especificação permite a definição de um critério mais complexas, tais como a seguir:



Existe uma relação entre a fábrica e Repositório. Eles são ambos os padrões de design orientado a modelos, e ambos nos ajudam a gerenciar o ciclo de vida de objetos de domínio. Enquanto a fábrica está preocupado com a criação de objetos, o Repositório cuida de objetos já existentes. O Repositório pode cache de objetos localmente, mas na maioria das vezes ele precisa recuperá-los a partir de um armazenamento persistente. Os objectos são, quer criado utilizando um construtor ou eles são passados para uma fábrica para ser construído. Por esta razão, o repositório pode ser visto como uma fábrica, porque cria

objetos. Não é uma criação a partir do zero, mas uma reconstituição de um objeto que existia. Não devemos misturar um repositório com uma fábrica. A fábrica deve criar novos objetos, enquanto o repositório deve encontrar objetos já criados. Quando um novo objecto é para ser adicionado ao Repositório, deve ser criado pela primeira vez utilizando a fábrica, e em seguida, deve ser dado para o repositório que armazená-lo como no exemplo abaixo.



Outra forma como isso é observado é que as fábricas são “puro domínio”, mas que os repositórios podem conter links para a infra-estrutura, por exemplo, o banco de dados.