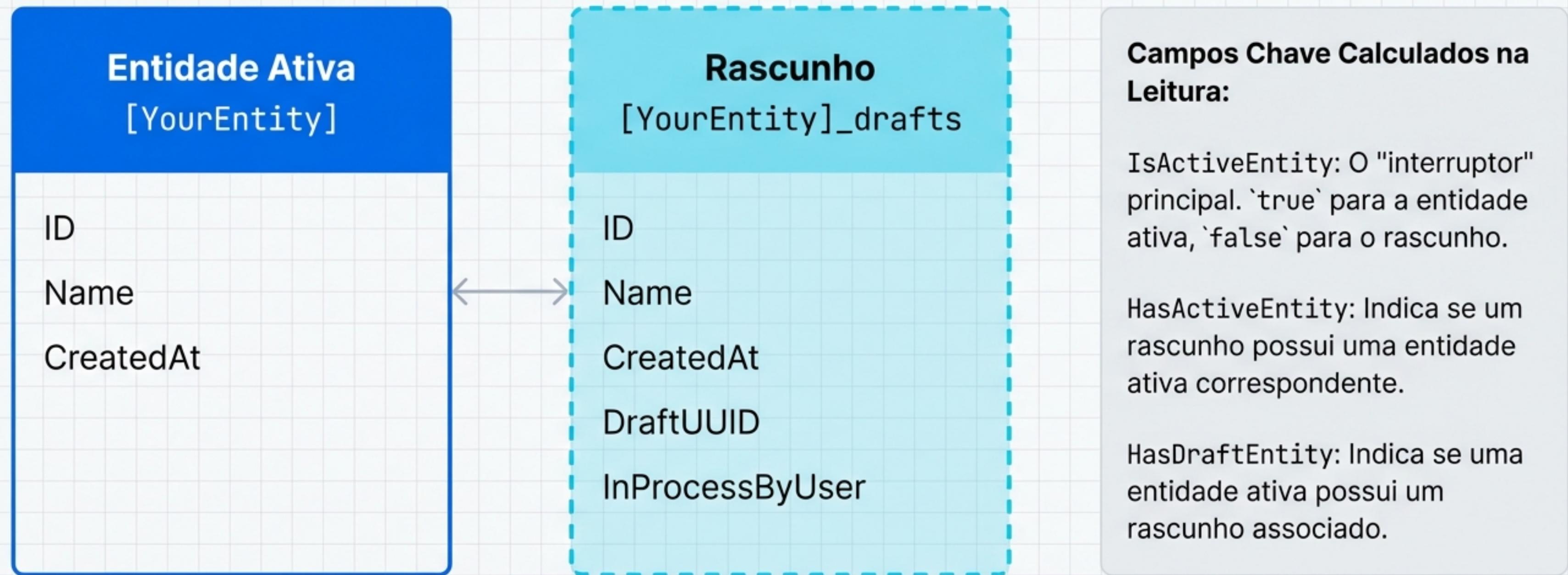


Dominando Fiori Drafts no CAP Java

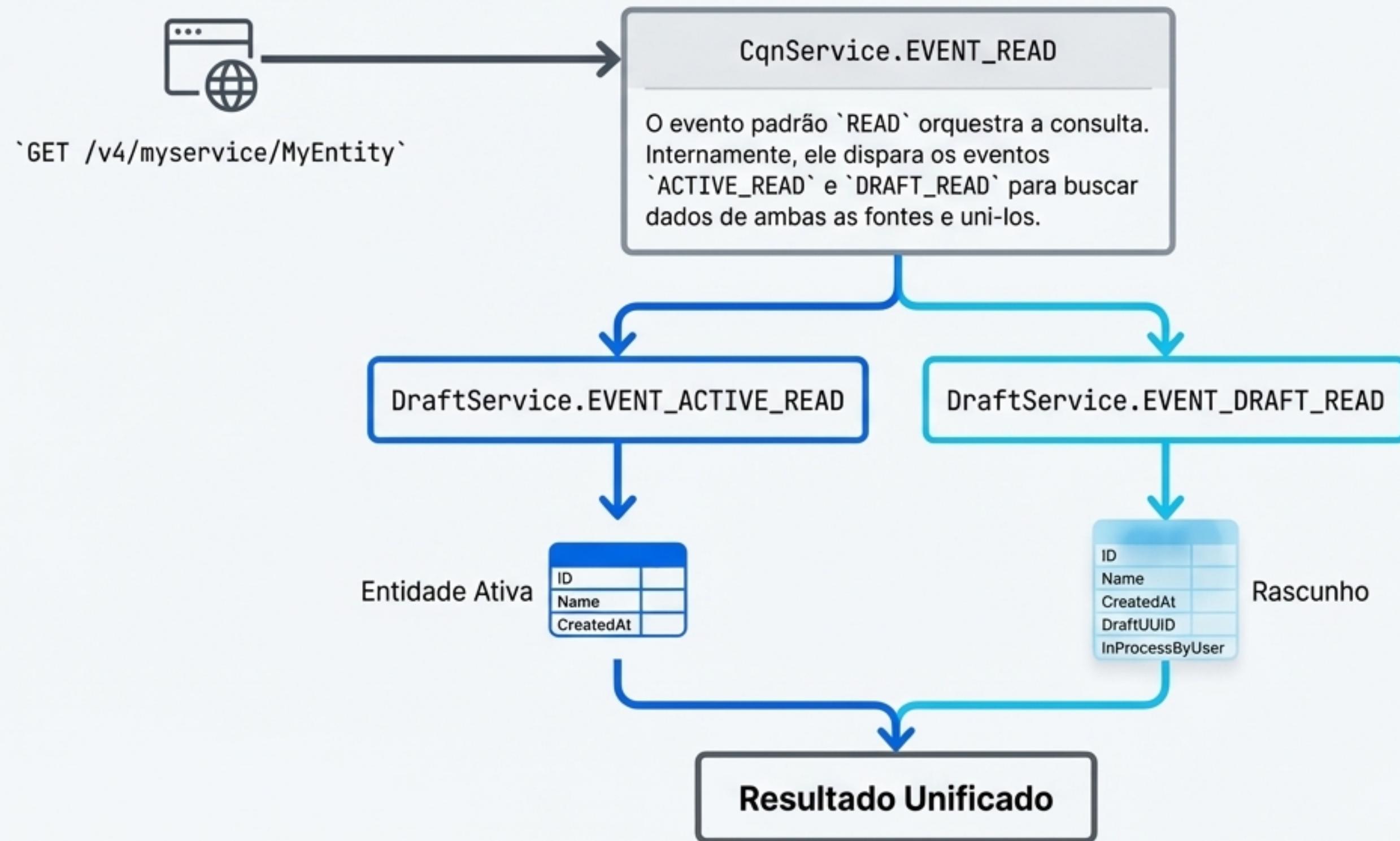
Um Guia Arquitetural para Desenvolvedores

A Arquitetura Fundamental: Entidades Ativas e Rascunhos

Quando uma entidade é habilitada para draft, o framework cria um **conjunto adicional de tabelas** no banco de **dados**. Essas tabelas armazenam os **rascunhos**, mantendo-os separados dos dados "ativos" e produtivos.



O Fluxo de Leitura: Unindo Dados Ativos e Rascunhos



Quando usar cada evento?

@Before / **@After**: Use o evento **'READ'** para modificar queries ou resultados de forma geral.

@On: Use **'ACTIVE_READ'** ou **'DRAFT_READ'** para handlers personalizados que operam especificamente em dados ativos ou rascunhos.

Cenário Avançado: Lendo Entidades Ativas Remotas

E se os dados ativos não estiverem no mesmo banco de dados, como em um sistema S/4HANA remoto?

Configuração

```
cds.drafts.persistence: split
```

Esta propriedade força uma separação estrita nas consultas, evitando JOINs entre tabelas ativas e de rascunho. Consultas a entidades ativas não conterão campos específicos de draft (como IsActiveEntity).

Implementação

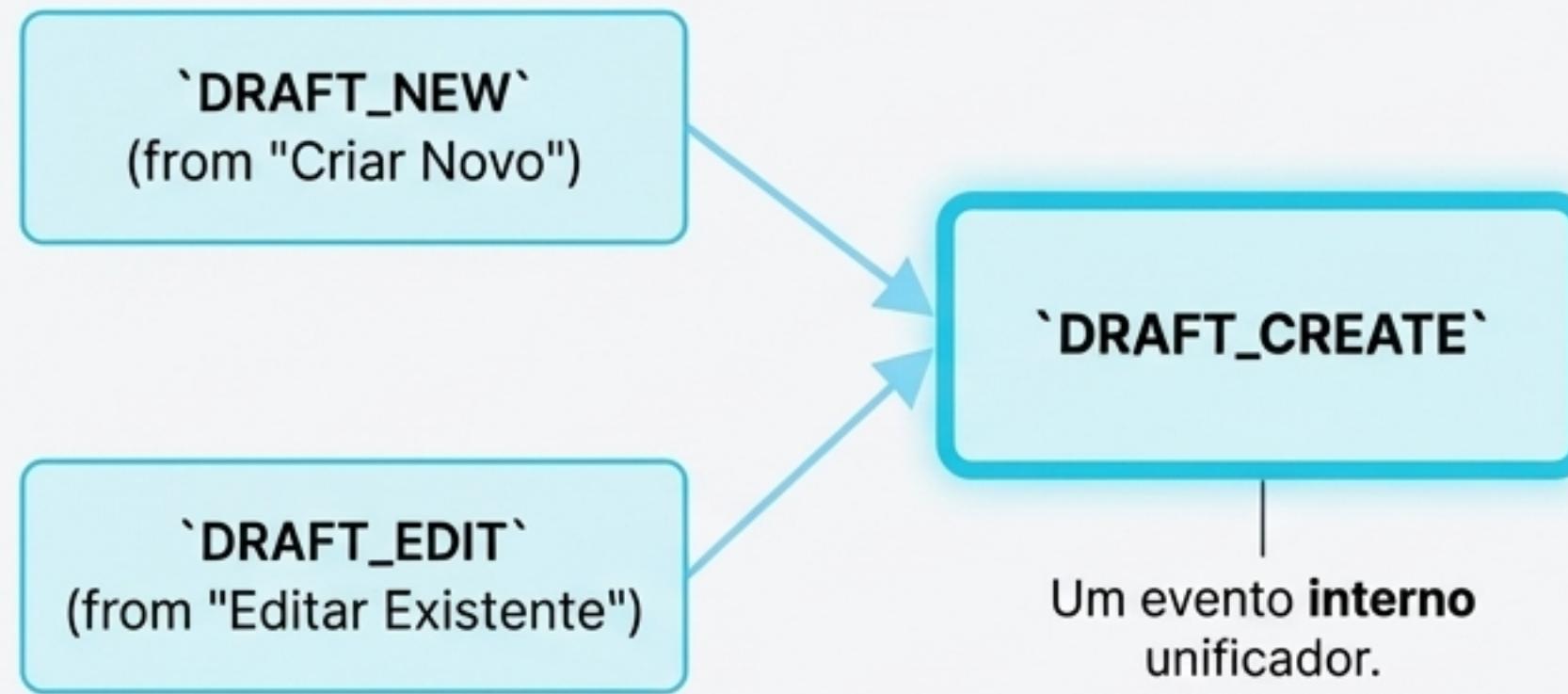
```
// Delegando a leitura de entidades ativas para um sistema remoto  
@On(entity = MyRemoteDraftEnabledEntity_.CDS_NAME)  
public Result delegateToS4(ActiveReadEventContext context) {  
    return remoteS4.run(context.getQn());  
}
```

Isso é útil apenas se você também delegar os eventos CREATE, UPDATE e DELETE para o sistema remoto.

O Ciclo de Vida da Edição de um Rascunho

Ação do Usuário (UI)	Requisição HTTP / OData	Evento Principal (DraftService)	Descrição
 Criar Novo	POST /Entity	DRAFT_NEW	Cria um novo rascunho vazio.
 Editar Existente	POST /Entity(key)/draftEdit	DRAFT_EDIT	Cria um rascunho a partir de uma entidade ativa.
 Salvar Alterações	PATCH /Entity(key, IsActiveEntity=false)	DRAFT_PATCH	Atualiza um rascunho existente.
 Cancelar Rascunho	DELETE /Entity(key, IsActiveEntity=false)	DRAFT_CANCEL	Descarta um rascunho existente.

O Ponto de Extensão Chave: O Evento Interno `DRAFT_CREATE`



Concept

O `DRAFT_CREATE` não é acionado diretamente por uma requisição OData. Ele é um evento **interno**, disparado tanto pelo fluxo `DRAFT_NEW` (criação do zero) quanto pelo `DRAFT_EDIT` (criação a partir de um ativo).

Why It Matters

Isso o torna o local ideal para centralizar lógicas de inicialização, como preenchimento de valores padrão ou calculados, independentemente de como o rascunho foi criado.

```
@Before  
public void prefillOrderItems(DraftNewEventContext context, OrderItems orderItem) {  
    // Lógica para preencher campos com valores padrão  
    // Ex: orderItem.setQuantity(1);  
}
```

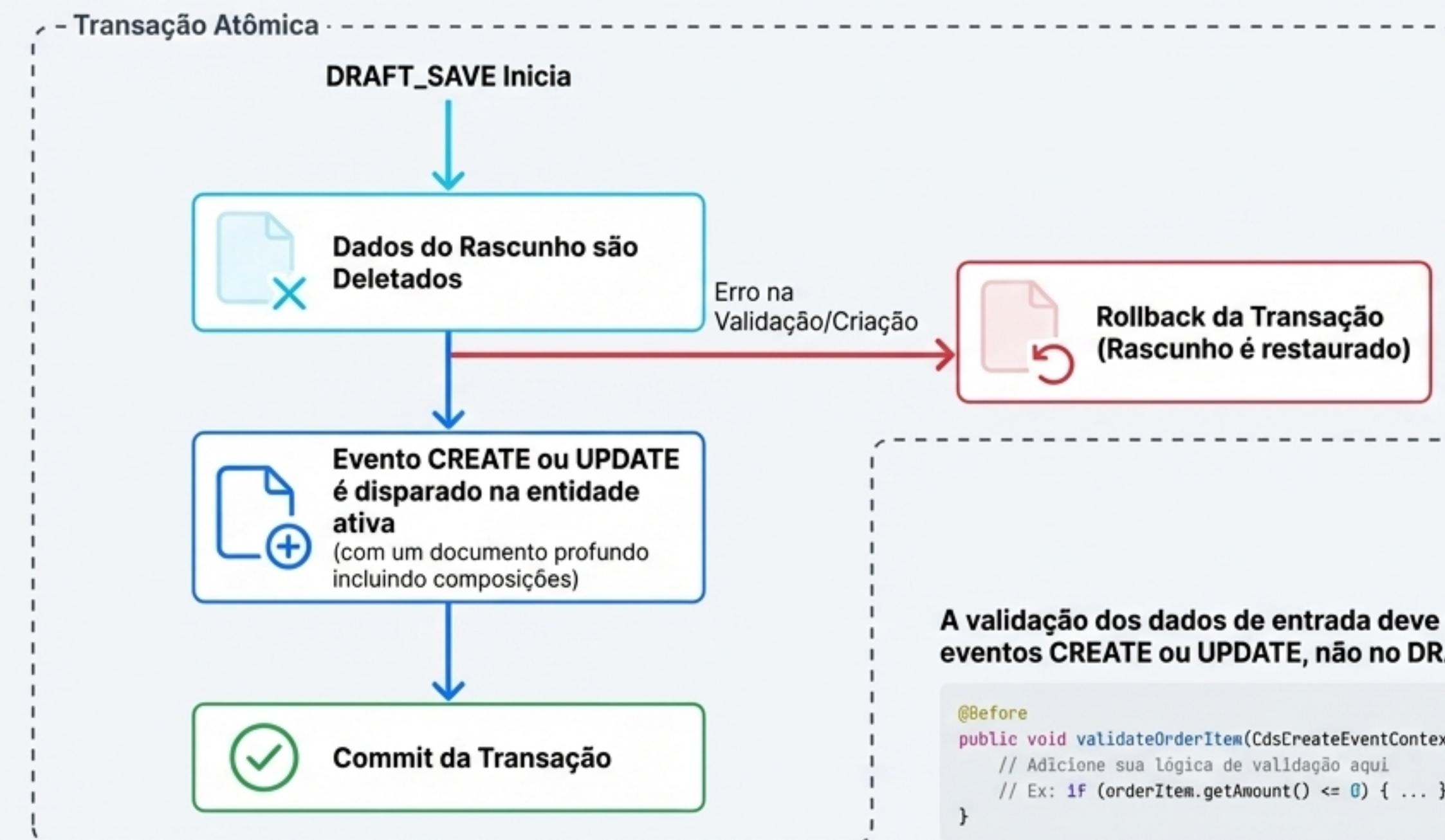
Este código é executado antes que um novo rascunho seja salvo, garantindo que os valores padrão estejam presentes quando o usuário começar a editar.

Ativação: Transformando Rascunhos em Dados Reais

Ação do Usuário: Ativar

Requisição: POST /Entity(key, IsActiveEntity=false)/draftActivate

Evento Principal: DraftService.EVENT_DRAFT_SAVE



Gerenciando Exclusões em Entidades com Rascunho

Excluindo o Rascunho

Cenário

O usuário cancela a edição.

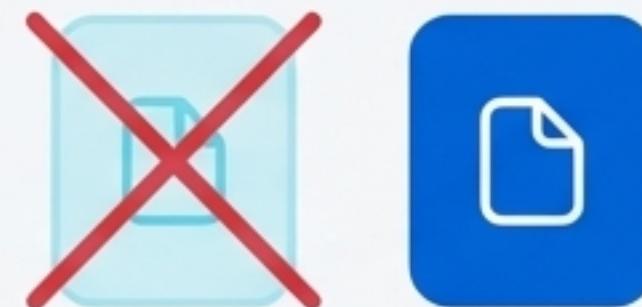
Requisição

```
DELETE /.../(IsActiveEntity=false, ID=<key>)
```

Evento Disparado

DRAFT_CANCEL

Resultado



Apenas o rascunho é removido. A entidade ativa permanece intacta.

Excluindo a Entidade Ativa

Cenário

O registro principal é deletado.

Requisição

```
DELETE /.../(IsActiveEntity=true, ID=<key>)
```

Eventos Disparados

Uma sequência: `DELETE` (na entidade ativa) e `DRAFT_CANCEL` (no rascunho associado).

Resultado



Ambos, a entidade ativa e seu rascunho, são removidos.

Acesso Direto: Bypassing do Fluxo de Rascunho

Útil para pré-carregar dados, integrações ou processos técnicos que precisam manipular entidades ativas diretamente.

Requisição HTTP / OData	Evento CAP Java	Ação
POST com IsActiveEntity: true no payload	CqnService.EVENT_CREATE	Cria a entidade ativa diretamente.
PUT com IsActiveEntity=true na URI	CqnService.EVENT_UPDATE	Atualiza (full) a entidade ativa.
PATCH com IsActiveEntity=true na URI	CqnService.EVENT_UPDATE	Atualiza (sparse) a entidade ativa.



Acesso Direto Não Ignora o Bloqueio (Draft Lock)!

Se uma entidade ativa estiver bloqueada por um rascunho existente, qualquer tentativa de atualização direta será bloqueada. Isso garante que as alterações do usuário no rascunho não sejam perdidas.

O Painel de Controle: Ajustando o Comportamento dos Rascunhos



Bloqueio de Rascunho (Draft Lock)

Uma entidade com um rascunho é bloqueada para edição por outros usuários.

```
cds.drafts.cancellationTimeout: 1h
```

Padrão: 15 minutos.

Limpeza Automática (Garbage Collection)

Rascunhos antigos (stale) são automaticamente removidos.

```
cds.drafts.deletionTimeout: 8w
```

Padrão: 30 dias.

 **Developer Tip:** Para ser notificado quando um rascunho é limpo, registre um handler no evento `DRAFT_CANCEL`.

Desabilitando a Limpeza

É possível desativar completamente a limpeza automática.

```
cds.drafts.gc.enabled: false
```

Customizando a Criação de Rascunhos com Ações

Por padrão, o Fiori usa um `POST` com corpo vazio para criar um rascunho. Você pode substituir esse comportamento por uma ação customizada para, por exemplo, passar parâmetros iniciais.

1.

1. Defina a Ação no CDS

Crie uma `action` vinculada (*bound*) à entidade.

```
action MyDraftAction(in : many $self, ...) returns MyEntity;
```

- O parâmetro de vínculo deve ser do tipo `many $self`.

2.

2. Anote a Entidade

Use a anotação `@Common.DraftRoot.NewAction: '<nome_da_acao>'`.

```
@Common.DraftRoot.NewAction: 'MyDraftAction'  
entity MyEntity ...
```

- Isso informa ao Fiori qual ação utilizar.

3.

3. Implemente a Ação em Java

A implementação **deve** chamar o método `newDraft(CqnInsert)` da `DraftService`.

- A ação deve retornar a entidade de rascunho recém-criada.

```
@Action(name = "MyDraftAction", service = ...)  
public MyEntity myDraftAction(CdsCreateEventContext context) {  
    // ... prepara CqnInsert ...  
    DraftService draftService = ...;  
    return draftService.newDraft(cqnInsert).as(MyEntity.class);  
}
```

Exemplo Prático: Ação de Criação Customizada

Definição no `service.cds`

```
service AdminService {  
    @odata.draft.enabled  
    @Common.DraftRoot.NewAction: 'AdminService.createDraft'  
    entity Orders as projection on my.Orders actions {  
        action createDraft(in: many $self, orderNo: String)  
            returns Orders;  
    };  
}
```

Informa ao Fiori qual ação usar.

→ Assinatura da ação vinculada.

Implementação no `Handler.java`

```
@On(entity = Orders_.CDS_NAME)  
public void createDraft(CreateDraftContext context) {  
    Orders order = Orders.create();  
    order.setOrderNo(context.getOrderNo()); → Usa o parâmetro passado pela ação.  
  
    // A chamada para o DraftService é essencial!  
    context.setResult(adminService.newDraft(  
        Insert.into(Orders_.class).entry(order)  
    ));  
}
```

→ **Chamada crucial**** para criar o rascunho.

Controle Programático com a Interface `DraftService`

Qualquer `ApplicationService` que contenha uma entidade com draft habilitado também implementa a interface `DraftService`.

Ela fornece uma camada de API para interagir com o ciclo de vida dos rascunhos diretamente no código Java.

As principais métodos outro API em JetBrains Mono com:

- `newDraft(...)` : Cria um novo rascunho.
- `patchDraft(...)` : Atualiza um rascunho existente.
- `saveDraft(...)` : Ativa um rascunho (equivalente ao `draftActivate`).
- `editDraft(...)` : Coloca uma entidade ativa em modo de edição, criando um rascunho.



****PersistenceServices` não conhecem drafts!****

Sempre use o `DraftService` ou `ApplicationService` para executar queries que precisam da lógica de unificação de rascunhos. Queries via `PersistenceService` verão apenas as tabelas físicas separadas.

‘DraftService’ em Ação: Um Ciclo de Vida Completo

```
// 1. Obtenha a instância do serviço  
DraftService adminService = ...;  
  
// 2. Crie um novo rascunho  
Orders order = adminService.newDraft(Insert.into(ORDERS)).single(Orders.class); ←  
  
// 3. Preencha os dados ←  
order.setOrderNo("DE-123456");  
  
// 4. Atualize (patch) o rascunho  
adminService.patchDraft(Update.entity(ORDERS).data(order)  
    .where(o -> o.ID().eq(order.getId()).and(o.IsActiveEntity().eq(false))));  
  
// 5. Ative (save) o rascunho  
CqnSelect orderDraft = Select.from(ORDERS)  
    .where(o -> o.ID().eq(order.getId()).and(o.IsActiveEntity().eq(false)));  
adminService.saveDraft(orderDraft); ←  
  
// 6. Coloque a entidade ativa de volta em modo de edição  
CqnSelect orderActive = Select.from(ORDERS)  
    .where(o -> o.ID().eq(order.getId()).and(o.IsActiveEntity().eq(true))); ←  
adminService.editDraft(orderActive, true);
```

Resumo e Melhores Práticas



Pense no Modelo de Duas Tabelas

Sempre visualize a arquitetura de entidades Ativas e de Rascunho. O campo IsActiveEntity é o que as diferencia em queries unificadas.



Use a API Correta para a Tarefa

DraftService é a sua ferramenta para manipulação programática. Evite PersistenceService para operações que precisam estar cientes dos rascunhos.



Escolha o Evento Certo para a Lógica Certa

Use DRAFT_CREATE para valores padrão. Use os eventos CREATE/UPDATE na entidade ativa para validações finais durante a ativação.



Conheça Suas Ferramentas de Configuração

Ajuste o cancellationTimeout e deletionTimeout para controlar o bloqueio e a limpeza automática, adaptando o comportamento às necessidades da sua aplicação.