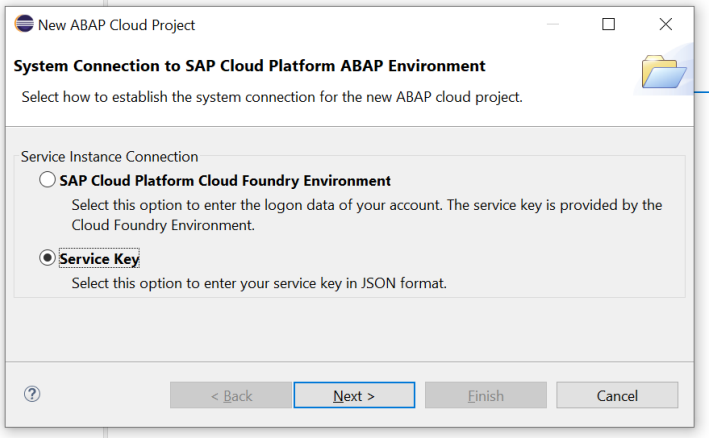
Anexo ao relatório

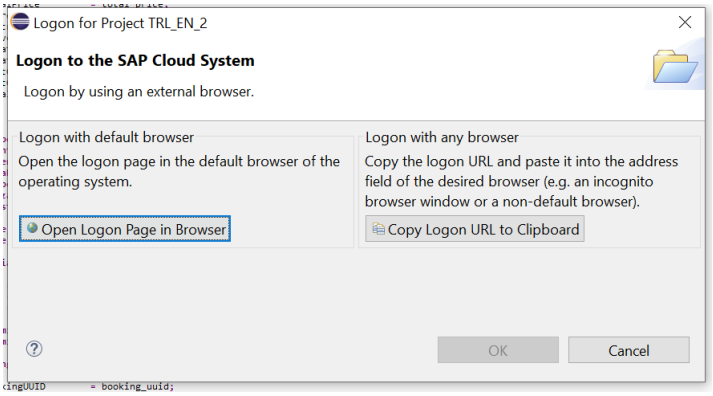
O objetivo proposto foi criar uma aplicação para uma agência de viagens como já referido no relatório. Neste anexo vou explicar como construí a aplicação, no entanto há partes que são repetidas/semelhantes a nível de raciocínio e por isso não acho que se justifique referir.

O primeiro passo é criar uma conta SAP que, dentro da service instances criada do tipo abap trial, e dentro desta criamos uma nova instância. Dentro desta instância criamos uma key service. Precisamos desta key, porque é a que vamos usar para configurar manualmente a conexão com a instância criada do ambiente ABAP, bem como com as Ferramentas de Desenvolvimento ABAP no futuro. A seguir, vamos configurar o Eclipse para que fique com a perspetiva de ABAP Development Tools. Após tudo configurado criamos um projeto ("create an



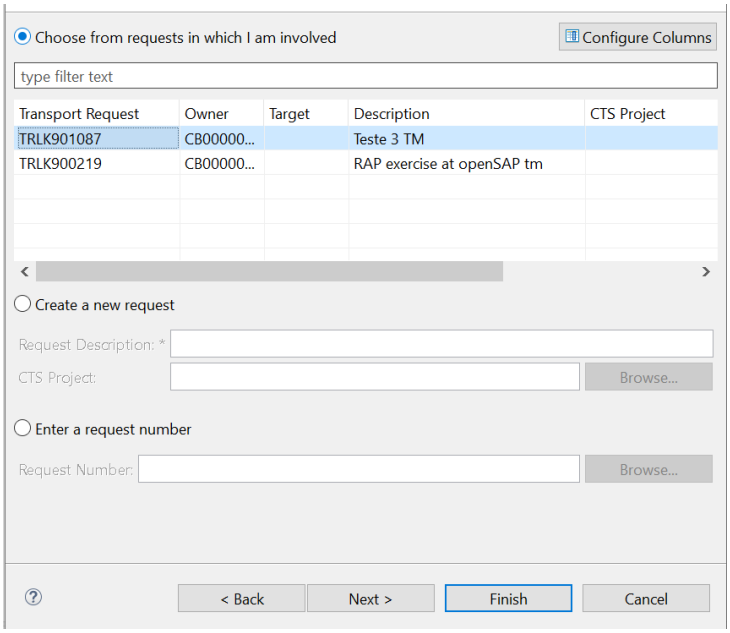
ABAP cloud project") e selecionamos o service key, pois é através do ficheiro json, que foi criado no site SAP no início, que vamos fazer a ligação.

Após fazer essa ligação basta fazer log on.



Quando chegamos aqui já temos tudo pronto para poder começar a desenvolver a aplicação. O primeiro passo nesta fase consiste em criar um packet geral "ZLOCAL", que vamos associar um transport request, que também terá que ser criado. Este transport request vai estar sempre

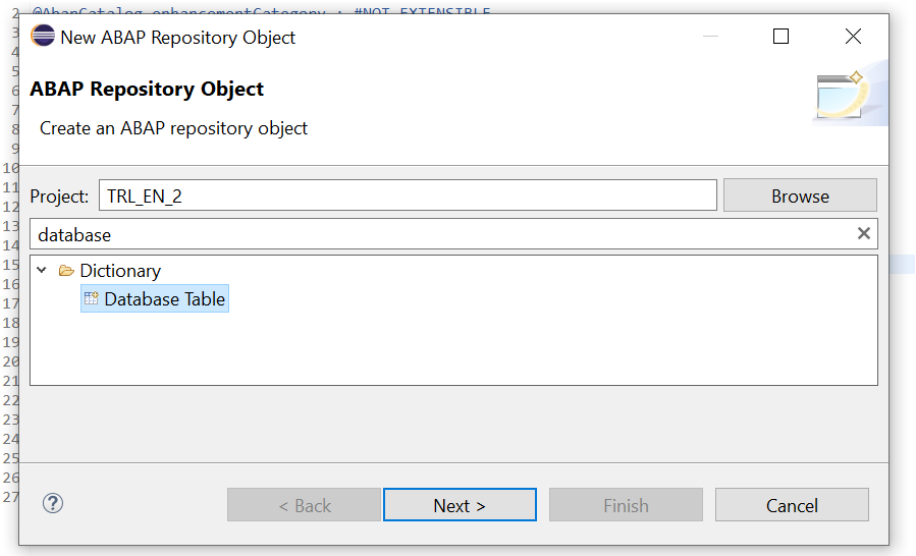
associado a qualquer funcionalidade que criamos. O Transport request é uma espécie de 'Container / Collection' de mudanças que são feitas no sistema de desenvolvimento.



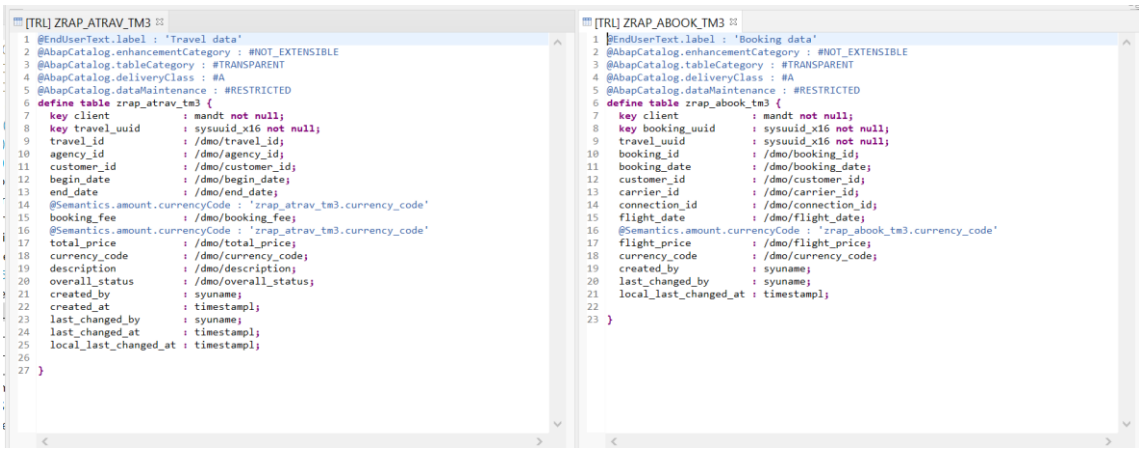
Como referi no relatório, o ponto de partida do desenvolvimento da construção da aplicação (o development flow) é a criação das tabelas database, a que se seguem o data model CDS, a projeção do modelo de dados e o enriquecimento com a semântica da IU. Em seguida, criaremos o serviço OData e visualizaremos a aplicação de viagens SAP Fiori. Finalmente vamos implementar a autorização básica para acesso a dados usando funções CDS.

Antes de começar a criar as tabelas database é importante perceber que o cenário de referência de vôo ABAP pertence ao namespace / DMO /, por isso, é necessário adicionar o pacote principal, / DMO / FLIGHT, aos meus pacotes de favoritos. O cenário de referência contém exemplos de implementações dos diferentes recursos RAP bem como várias legacy implementations, dictionary objects, tabelas de database, domínios e elementos de dados e demo data.

Comecemos então por criar duas tabelas (Travel e Booking) database onde vão ser armazenados os dados das viagens e das reservas.



Os atributos específicos da tabela e do campo são especificados por meio de anotações. Os campos da tabela são especificados entre as chaves, e a keyword "key" é usada para especificar



[TRL] ZRAP_ATRAV_TM3

```
@EndUserText.label : 'Travel data'
@AbapCatalog.enhancementCategory : #NOT_EXTENSIBLE
@AbapCatalog.tableCategory : #TRANSPARENT
@AbapCatalog.deliveryClass : #A
@AbapCatalog.dataMaintenance : #RESTRICTED
define table zrap_atrav_tm3 {
  key client           : mandt not null;
  key travel_uuid      : sysuuid_x16 not null;
  travel_id            : /dmo/travel_id;
  agency_id            : /dmo/agency_id;
  customer_id          : /dmo/customer_id;
  begin_date           : /dmo/begin_date;
  end_date             : /dmo/end_date;
  @Semantics.amount.currencyCode : 'zrap_atrav_tm3.currency_code'
  booking_fee          : /dmo/booking_fee;
  @Semantics.amount.currencyCode : 'zrap_atrav_tm3.currency_code'
  total_price          : /dmo/total_price;
  currency_code        : /dmo/currency_code;
  description          : /dmo/description;
  overall_status       : /dmo/overall_status;
  created_by           : syuname;
  created_at           : timestampl;
  last_changed_by      : syuname;
  last_changed_at      : timestampl;
  local_last_changed_at : timestampl;

}
```

[TRL] ZRAP_ABOOK_TM3

```
@EndUserText.label : 'Booking data'
@AbapCatalog.enhancementCategory : #NOT_EXTENSIBLE
@AbapCatalog.tableCategory : #TRANSPARENT
@AbapCatalog.deliveryClass : #A
@AbapCatalog.dataMaintenance : #RESTRICTED
define table zrap_abook_tm3 {
  key client           : mandt not null;
  key booking_uuid     : sysuuid_x16 not null;
  travel_uuid          : sysuuid_x16 not null;
  booking_id           : /dmo/booking_id;
  booking_date         : /dmo/booking_date;
  customer_id          : /dmo/customer_id;
  carrier_id           : /dmo/carrier_id;
  connection_id        : /dmo/connection_id;
  flight_date          : /dmo/flight_date;
  @Semantics.amount.currencyCode : 'zrap_abook_tm3.currency_code'
  flight_price         : /dmo/flight_price;
  currency_code        : /dmo/currency_code;
  created_by           : syuname;
  last_changed_by      : syuname;
  local_last_changed_at : timestampl;

}
```

os campos-chave. Os atributos técnicos da tabela são definidos na parte de cima da definição da tabela, antes da keyword "define". Como se pode ver no screenshot, vemos as anotações usadas para especificar a referência da currency key para o amount fields, booking fee, total price, e flight price. A tabela agora consiste nos key fields, client e travel_uuid, e table fields, como o travel_id, o agency_id, o customer_id, o total_price e o overall_status. Para além destes também inclui os dados de administração standard, como o respectivo utilizador ou, por exemplo a created_at dos dados. A tabela field currency_code é especificada como o campo de referência para os campos de valor booking_fee e total_price usando uma anotação semântica. Se nós correrrmos a aplicação como está ainda não aparece nada, porque a travel list report está vazia. A solução para resolver esse problema é criar uma ABAP Class.

Na criação desta classe vamos adicionar a interface IF_OO_ADT_RUN. Relativamente ao código aqui apresentado, primeiro quaisquer entradas existentes em ambas as tabelas são apagadas (linha 16 e 17). Depois os dados são selecionados da tabela / dmo / viagens e inseridos na nova

tabela de viagens. A função SQL uuid () é usada para definir o valor do key field travel_uuid. O trabalho de commit statement é executado para manter os dados. Neste caso a seleção de dados foi limitada até 200 registos de viagem.

[TRL] ZCL_GENERATE_DEMO_DATA_TM3

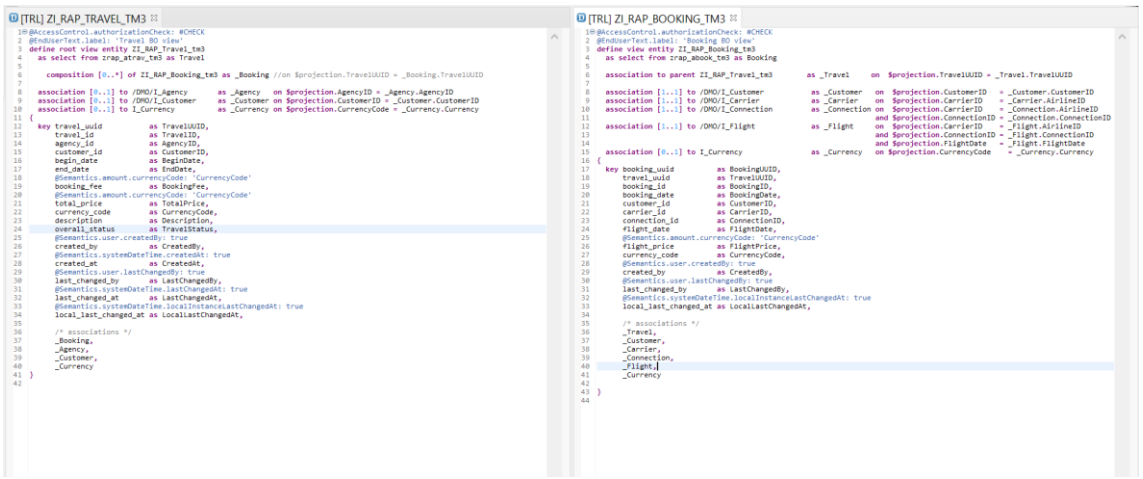
```
CLASS zcl_generate_demo_data_tm3 DEFINITION
  PUBLIC
  FINAL
  CREATE PUBLIC .

  PUBLIC SECTION.
    INTERFACES if_oo_adt_classrun.
  PROTECTED SECTION.
  PRIVATE SECTION.
ENDCLASS.
```

[TRL] ZCL_GENERATE_DEMO_DATA_TM3

```
CLASS zcl_generate_demo_data_tm3 IMPLEMENTATION.
  METHOD if_oo_adt_classrun~main.

    " delete existing entries in the database table
    DELETE FROM zrap_atrav_tm3.
    DELETE FROM zrap_abook_tm3.

    " insert travel demo data
    INSERT zrap_atrav_tm3 FROM (
        SELECT
          FROM /dmo/travel
          FIELDS
            uuid(  )      AS travel_uuid           ,
            travel_id     AS travel_id             ,
            agency_id     AS agency_id             ,
            customer_id   AS customer_id           ,
            begin_date    AS begin_date            ,
            end_date      AS end_date              ,
            booking_fee   AS booking_fee           ,
            total_price   AS total_price           ,
            currency_code AS currency_code         ,
            description   AS description           ,
            CASE status
              WHEN 'B' THEN 'A' " accepted
              WHEN 'X' THEN 'X' " cancelled
              ELSE 'O'          " open
            END           AS overall_status        ,
            createdby     AS created_by            ,
            createdat     AS created_at            ,
            lastchangedby AS last_changed_by       ,
            lastchangedat AS last_changed_at       ,
            lastchangedat AS local_last_changed_at
            ORDER BY travel_id UP TO 200 ROWS
      ).
    COMMIT WORK.

    " insert booking demo data
    INSERT zrap_abook_tm3 FROM (
        SELECT
          FROM   /dmo/booking    AS booking
            JOIN zrap_atrav_tm3 AS z
            ON   booking~travel_id = z~travel_id
          FIELDS
            uuid( )                AS booking_uuid          ,
            z~travel_uuid          AS travel_uuid           ,
            booking~booking_id     AS booking_id            ,
            booking~booking_date   AS booking_date          ,
            booking~customer_id    AS customer_id           ,
            booking~carrier_id     AS carrier_id            ,
            booking~connection_id  AS connection_id         ,
            booking~flight_date    AS flight_date           ,
            booking~flight_price   AS flight_price          ,
            booking~currency_code  AS currency_code         ,
            z~created_by           AS created_by            ,
            z~last_changed_by      AS last_changed_by       ,
            z~last_changed_at      AS local_last_changed_by
      ).
    COMMIT WORK.

    out->write( 'Travel and booking demo data inserted.').
  ENDMETHOD.

ENDCLASS.
```

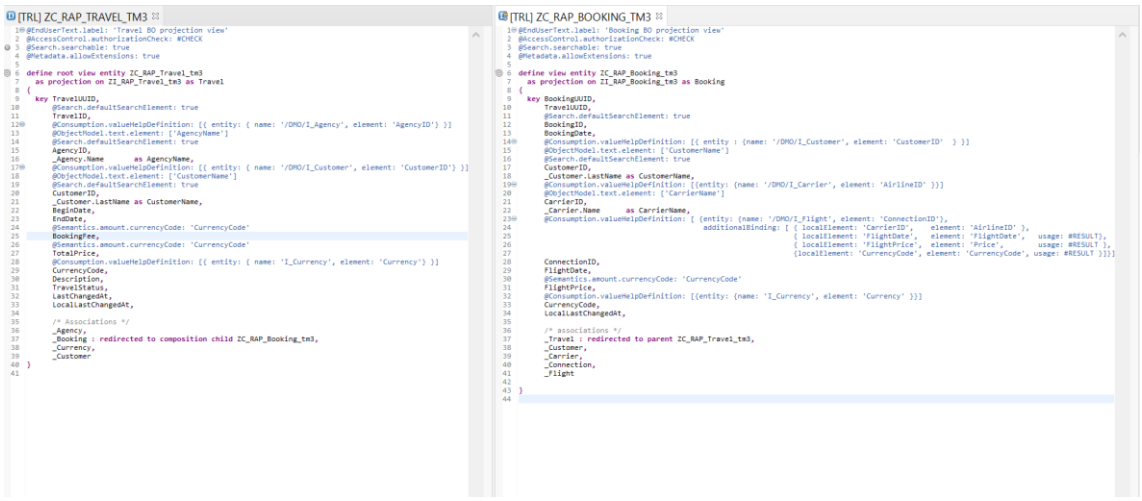
Depois de já termos as database criadas e a funcionar, começamos por definir o modelo de dados CDS clássico e, depois, melhoramo-lo para definir a estrutura do business object. Assim, começamos com a criação da view da viagem: clica-se na tabela da viagem e seleciona-se New Data Definition. A tabela das viagens é usada como fonte de dados e os campos da tabela são inseridos automaticamente na lista das projeções entre as chaves. O campo do cliente é tratado implicitamente pela aplicação, porque é uma visão específica do cliente. A seguir, definimos os aliases para a fonte de dados e visualizamos as colunas usando a keyword "as" e melhoramos o modelo de dados CDS com associações para definir a sua relação com outras entidades.

As associações às entidades Booking, Agency, Customer e Currency são definidas e expostas na lista de projeções. Uma associação é definida com uma cardinalidade, uma entidade CDS de destino, um alias opcional e uma condição on. Na lista de projeções, o elemento de visualização Overall_status é renomeado para TravelStatus.
Depois, podemos adicionar semantics annotations ao currency e administrative fields para garantir um processamento de dados uniforme do lado do consumidor. Podemos também especificar o currencyCode como o campo de referência dos campos de currency BookingFee e TotalPrice usando uma semantics annotations. A anotação dos administrative fields CreatedBy, CreatedAt, LastChangedBy, LastChangedAt e LocalLastChangedAt são uma etapa de preparação para uma próxima etapa no futuro, onde o transacional behaviour da aplicação Travel List Report será ativado. Para estes administrative fields são necessárias anotações para permitir a atualização automática dos campos de administração em cada operação. Aplica-se o mesmo raciocínio na criação do booking view. Aqui no screenshot estão os dois códigos relativamente às viagens e às reservas.



Para terminar esta parte vamos definir a estrutura do nosso business object. Esta etapa não é obrigatória na criação da aplicação do tipo read-only, mas fazemo-la já para preparar o ambiente, o transactional enablement da nossa aplicação de viagen, às necessidades futuras. Assim, na parte da view da viagem, especifico a entidade de viagem como o nó root do BO composition usando a keyword "root" (linha 3 da travel). Na linha 6 da travel deve-se especificar a view da reserva como “filho” do BO usando a associação especial, composition. Relativamente ao booking temos que mudar a view da reserva e especificar a entidade viagem como nó “pai” usando a associação especial, association to parent (linha 6 do booking).

O próximo passo é projetar a parte do CDS que nos é realmente importante para a nossa aplicação. O objetivo é termos o modelo de dados CDS construído a partir de duas projeções do CDS views: uma view de projeção da viagem e uma view de projeção da reserva.



No screenshot, a linha 3, a anotação @Search, representa a ativação da projeção para a pesquisa e especifíca o element view para ser considerado numa pesquisa. A linha 4 está a permitir que haja facilidade nas modificações de ambas as projections views com as anotações de IU num documento separado: este documento é a extensão dos metadados CDS (em suma MDEs) que os separa das anotações relacionadas com o business objects. Através da anotação @ObjectModel vamos adicionar alguns novos campos, como Agency_Name, customer_Name e o Carrier_Name, na lista de projeção e estabelecer a conjunção de um campo com as suas unidades de texto descritivo independente da lingua. Também definimos o value help para os campos Agency ID, Customer ID, and Carrier ID usando a anotação @Consumption, e redirecionamos os nós de composition da viagem e da reserva para as entidades correspondentes da camada de projeção do business object. Com as projections views, podemos expôr apenas aqueles elementos que são relevantes para o serviço específico, incluindo a desnormalização do modelo de dados subjacente. O value help, a pesquisa e a semântica da IU também podem ser definidos.

A seguir, temos que especificar a view da projeção do BO como a entidade root, e fornecer um alias para a projection view da viagem que já está atribuída. Todas as views, elementos e associações são inseridos automaticamente na lista de projeção.

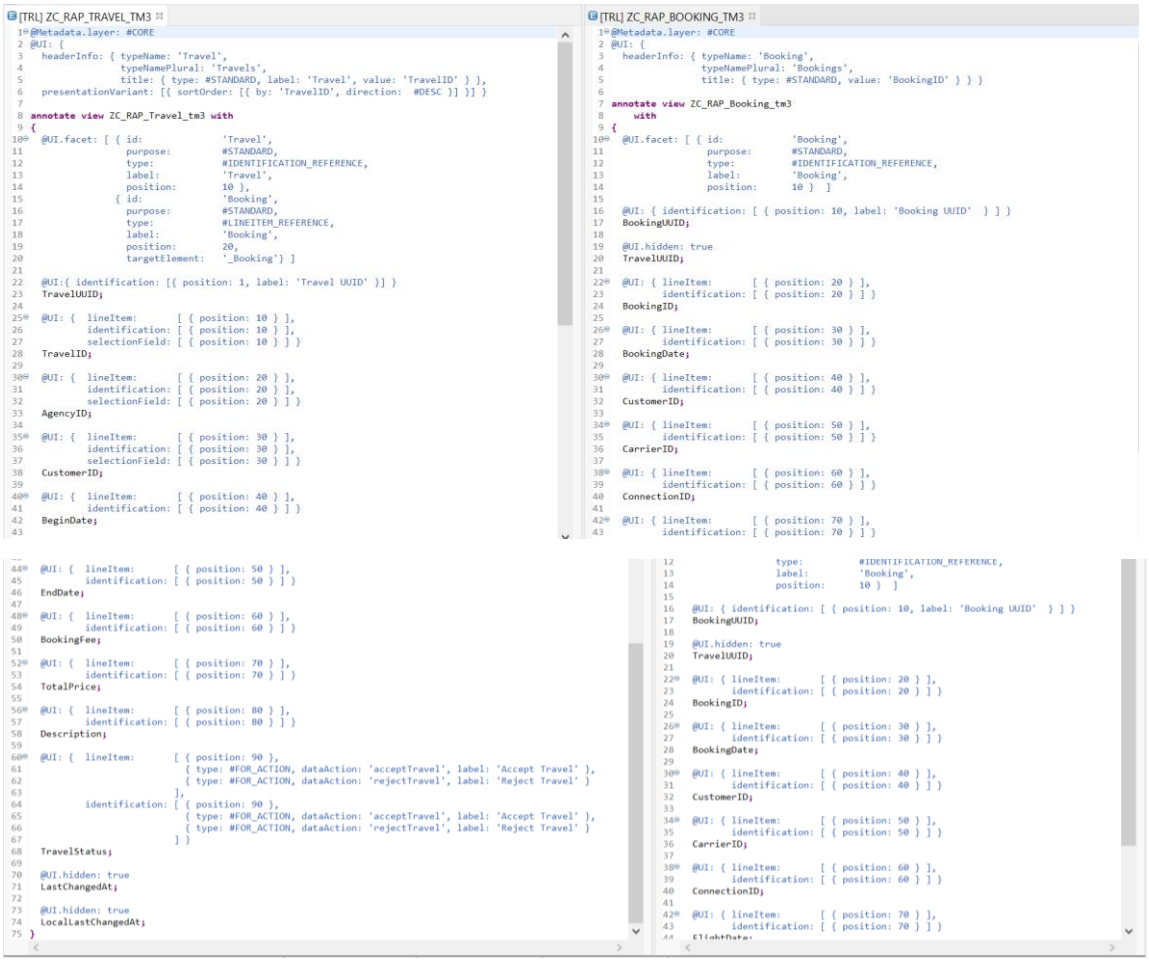
Nesta fase, vamos adicionar algumas semânticas específicas do serviço ao modelo de dados projetado, porque permite que a projection view seja aprimorada com extensões de metadados separadas e porque permite a pesquisa usando as anotações @Metadata.allowExtensions e @Search.searchable.

Relativamente à lista de projeções, a anotação @Search.DefaultSearchElement ativa a pesquisa das colunas TravelID, AgencyID e CustomerID. A anotação @ObjectModel.text.element especifica os elementos adicionados anteriormente AgencyName e CustomerName como descrições para os elementos AgencyID e CustomerID, respectivamente. A anotação @Consumption.ValueHelpDefinition define o value help para os elementos de visualização AgencyID, CustomerID e CurrencyCode. Aqui, o nome da entidade CDS que atua como um value help, é o nome do elemento que está vinculado ao elemento local que tem que ser especificado. É possível completar automáticamente ao especificar a entidade do value help de destino.

Vamos também especificar o elemento currency como o campo de referência para os campos de currency BookingFee e TotalPrice, usando o elemento annotation @Semantics.amount.currency. Manteremos todas as associações expostas para caso seja

necessário usar, mas redirecionaremos o BO “filho” da entidade booking para a visualização de projeção do BO Booking. A anotação @ObjectModel.text.element serve para fornecer uma descrição aos elementos CustomerId e CarrierId. Para o CustomerID, o CarrierID, o ConnectionID e o CurrencyCode os value help são definidos de formas diferentes. Para o value help do ConnectionID temos que adicionar uma condição de ligação extra, que é definida para retornar valores do value help que já foram guardados para os elementos de visualização locais CarrierID, FlightDate, FlightPrice e Currency (linha 23 do booking). O campo de visualização CurrencyCode é especificado como o campo de referência para o campo de currency FlightPrice (linha 31 booking).

De modo a enriquecer ainda mais a projeção do modelo de dados CDS com a semântica da IU vamos usar extensões de metadados CDS (MDEs). As extensões dos metadados do CDS MDEs permitem que se melhore sem alterações no modelos de dados. Com o CDS, podemos adicionar semânticas para diferentes domínios para uma entidade CDS existente usando anotações, normalmente, anotações de IU. Ao usar CDS MDEs, conseguimos uma separação do modelo de dados com todos os business objects da semântica da IU. Essa separação de interesses permite uma melhor gestão de mudança simplificada sem modificação. Graças à abordagem por causa das camadas de CDS MDEs, podemos definir mais de uma extensão de metadados CDS para uma entidade CDS. Ficamos assim com duas extensões de metadados CDS: um MDE CDS para a view de viagem projeção e outro para a projeção de reserva.



Cada extensão de metadados é atribuída a uma camada, como CORE#, PARTNER# ou CUSTOMER#. A camada determina a prioridade da avaliação. Então com a view diversificada e os campos das anotações da IU específicos, podemos ver o nome da entidade CDS que está

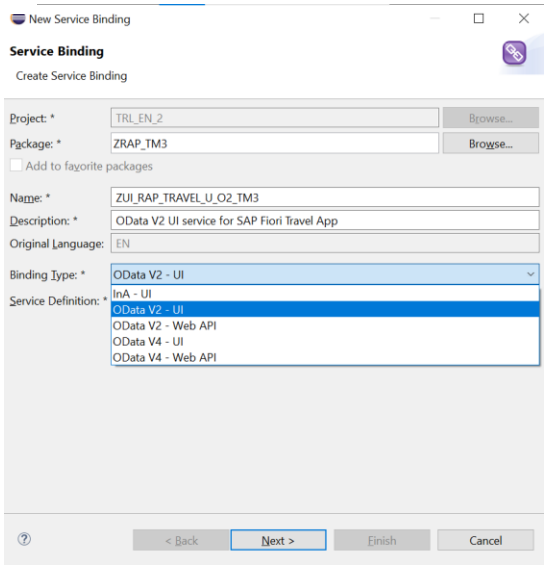
enriquecida. Nesta altura, temos a anotação @Metadata.layer, o nome da projection view do BO após a instrução ANNOTATE VIEW e uma entrada fictícia entre as chaves. Devemos especificar o #CORE como a camada de metadados. Quando existem várias extensões de metadados definidas para uma determinada entidade CDS, a layer determina a prioridade dos metadados. #CORE tem a prioridade mais baixa e #CUSTOMER a prioridade mais alta. Em cima devemos definir também algumas informações do cabeçalho, como o nome do tipo e o título. Neste caso, os dados de viagem apresentados serão classificados em ordem decrescente pelo elemento TravelID na lista, por isso, nas chaves, estamos a usar @UI.facet anotações para definir a pesquisa para a página do objeto e do seu layout. A página de objetos de viagem tem duas facetas: a referência de identificação das entidades de viagens e a referência do item da linha da entidade do booking, com a composition _booking especificada como elemento de destino.

A seguir, devemos especificar uma posição e possivelmente um rótulo para cada elemento usando as respectivas anotações de elemento @UI. O @UI.lineItem é a anotação usada para especificar as informações de layout de cada elemento mostrado como uma coluna na lista. A anotação @UI.identification é usada para especificar as informações de layout de cada elemento mostrado na secção de identificação da página do objeto. E a anotação @UI.selectionField é usada para permitir que um elemento seja selecionado na funcionalidade filtro. Com a anotação @UI.hidden: true, evitamos que os elementos sejam exibidos na IU ou nas configurações. Os elementos TravelID, AgencyID e CustomerID estão disponíveis para serem selecionado nos filtro e também mostrados na lista de viagens.

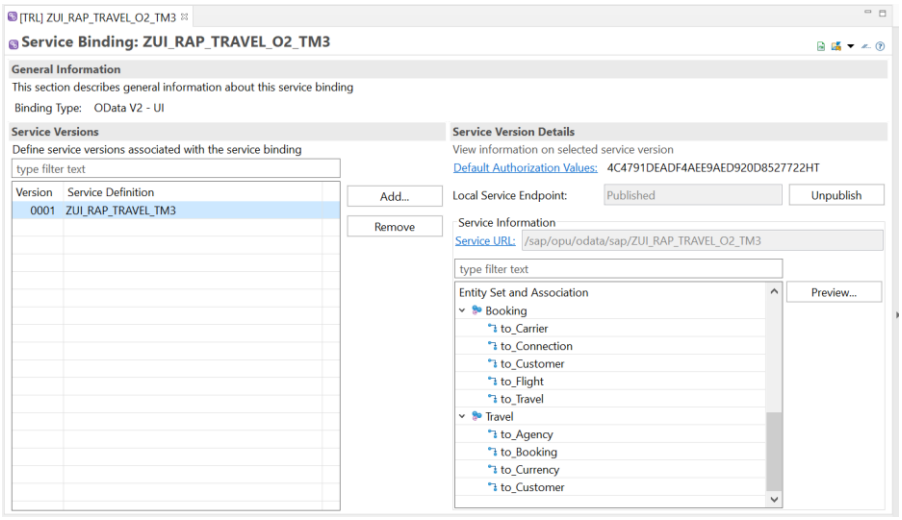
O próximo passo é criar o serviço Odata e visualizar a aplicação. Passos a seguir: criar uma definição de serviço para especificar o scope do serviço Odata, e de seguida criar uma service binding para ligar o service definition ao protocolo OData como um serviço de IU. O scope do serviço é definido expondo o que é relevante: cds views e os metadados.

[TRL] ZUI_RAP_TRAVEL_TM3

```
@EndUserText.label: 'Serv Definition for Travel App'
define service ZUI_RAP_TRAVEL_tm3 {
  expose ZC_RAP_Travel_tm3 as Travel;
  expose ZC_RAP_Booking_tm3 as Booking;
  expose /DMO/I_Agency as Agency;
  expose /DMO/I_Customer as Customer;
  expose /DMO/I_Flight as Flight;
  expose /DMO/I_Carrier as Carrier;
  expose /DMO/I_Connection as Connection;
  expose /DMO/I_Airport as Airport;
  expose I_Currency as Currency;
  expose I_Country as country;
}
```



Atualmente apenas OData V2 é compatível, no entanto, OData V4 está planeado para ser usado.



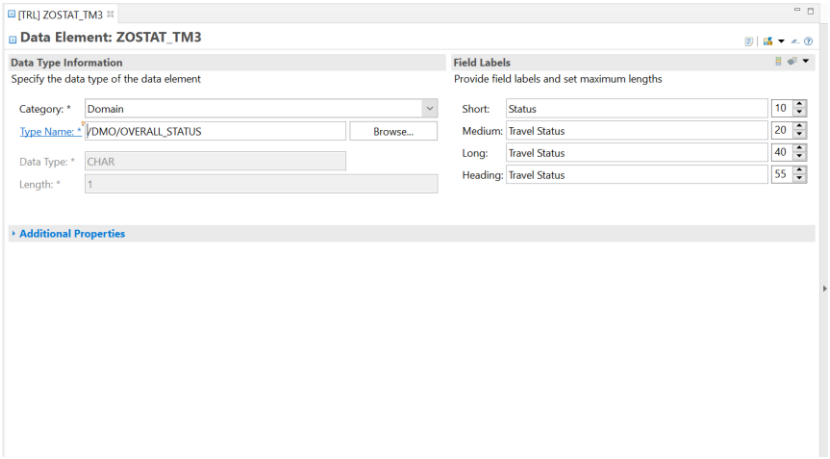
No service binding temos o service URL, os conjuntos de entidades expostas e as associações. Se clicarmos no link do service URL podemos ver os metadados do serviço no browser.
Para podermos ver a aplicação selecionamos a "Travel" e clicamos duas vezes ou clicamos com o botão direito, e aparece a opção abrir com Fiori Elements app.
A aplicação é aberta no browser e podemos ver as colunas de pesquisa: Agency ID e Customer ID que estão de acordo com as anotações de pesquisa definidas na projection view do BO de viagem. Podemos filtrar como por exemplo, pelo ID da agência, podemos usar o value help, podemos personalizar a lista adicionando, movendo e removendo colunas através da personalização das configurações. Por exemplo, podemos adicionar o UUID e movê-lo para a primeira posição. E podemos também facilmente restaurar o layout padrão.

Para finalizar, vamos implementar uma autorização básica para dados de acesso usando funções CDS para o business objects de viagens. O ABAP CDS fornece uma linguagem de controle de dados, DCL, para definir o acesso de autorização aos dados usando uma função CDS. A autorização clássica usada na plataforma ABAP requer verificações explícitas de autoridade codificada nos dados selecionados, que são armazenados em tabelas internas na camada de aplicação. O CDS DCL, em vez disso, oferece uma alternativa declarativa com base em verificações de autorização implícitas que ocorrem durante as tentativas de acesso à entidade CDS. As funções do CDS podem ser definidas usando condições PFCG, condições literais, condições do utilizador e condições de herança. Devem ser definidas explicitamente para cada entidade CDS, quando necessário. Fazem parte do modelo de dados e, portanto, também são enviadas para a database para que apenas os dados autorizados sejam devolvidos ao ler um data model CDS.

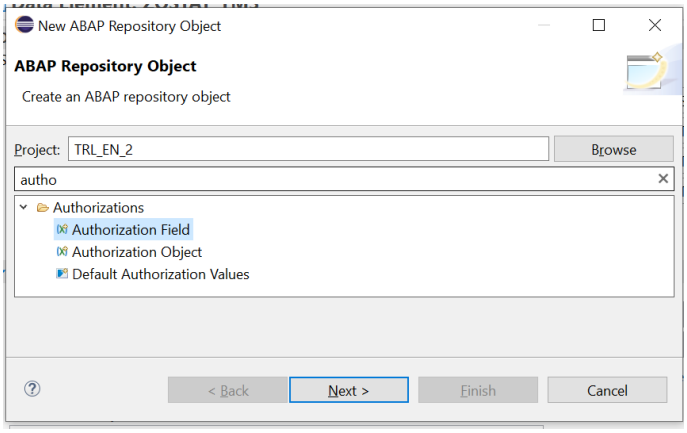
As funções do CDS são definidas uma vez e usadas automaticamente em todo o lado. Ao aceder às entidades CDS, teremos definido um authorization object e duas funções de CDS para a entidade de viagem, uma na camada de modelação de dados e outra na camada de provisionamento de business service para a projeção do modelo de dados. A função do CDS na camada de modelação de dados será definida com condições literais e PFCG, e criaremos um objeto de autorização para o efeito. O PFCG é um transaction code cujo objetivo é manter administration para manage roles e authorization data. A segunda função CDS irá herdar a

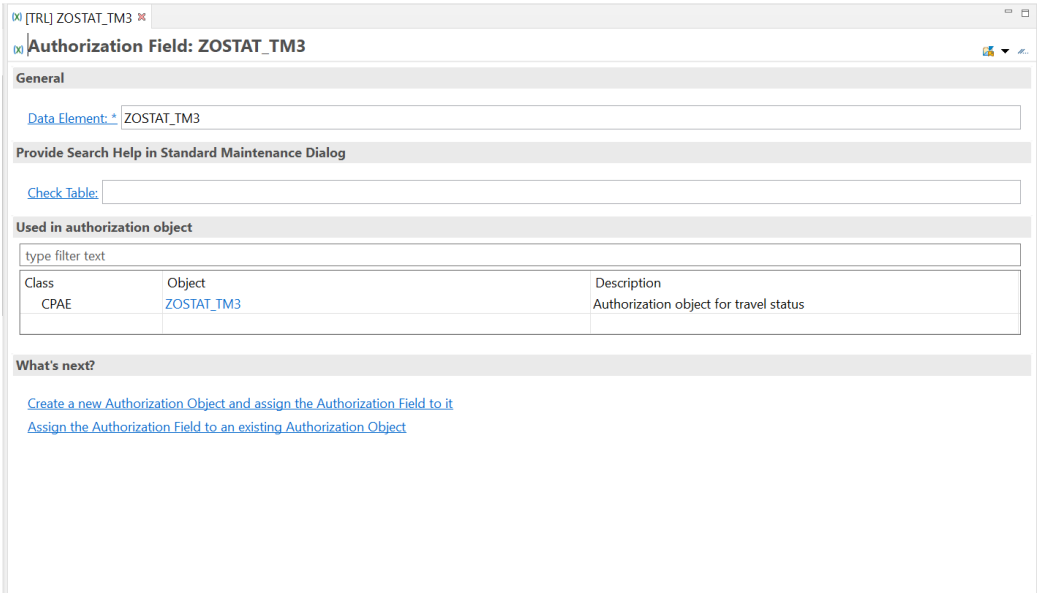
condição da camada subjacente. As regras de acesso consistirão numa condição literal para o elemento de visualização CurrencyCode e uma condição PFCG para o elemento de visualização TravelStatus. Para a definição do PFCG, criaremos um authorization object, incluindo campo de autorização e elemento de dados.

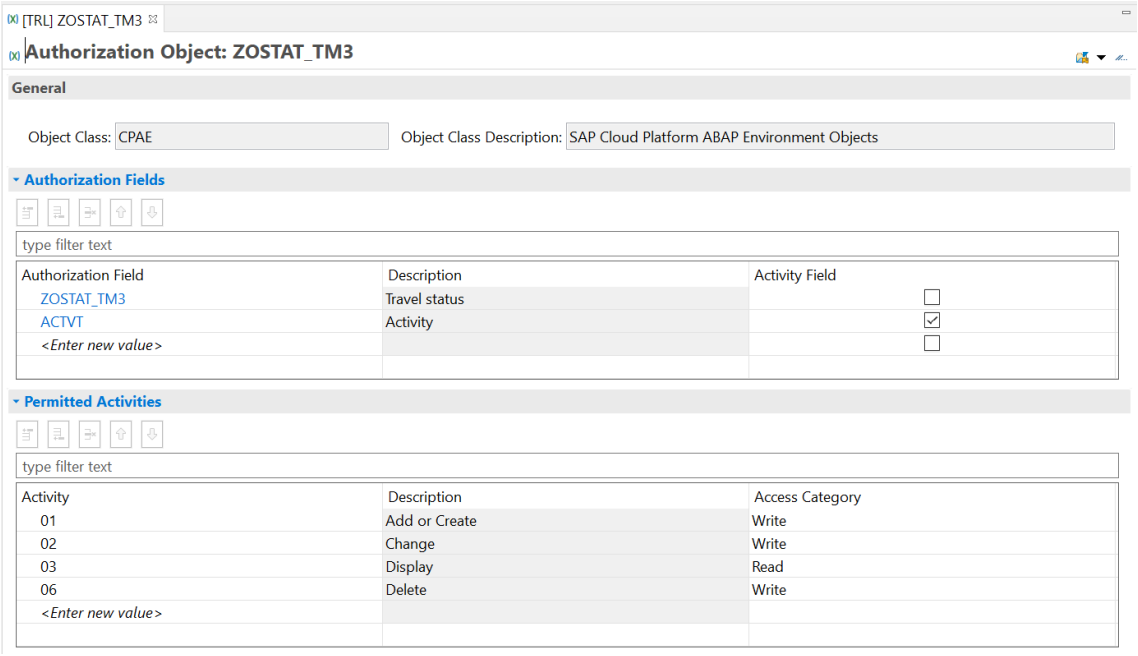
Criar Data Element



No campo Category temos que pôr “Domain” e inserir o nome do tipo no campo Type name. Nos Filed labels como mostra o screenshot, devemos preencher com “Status” e “Travel Status”.

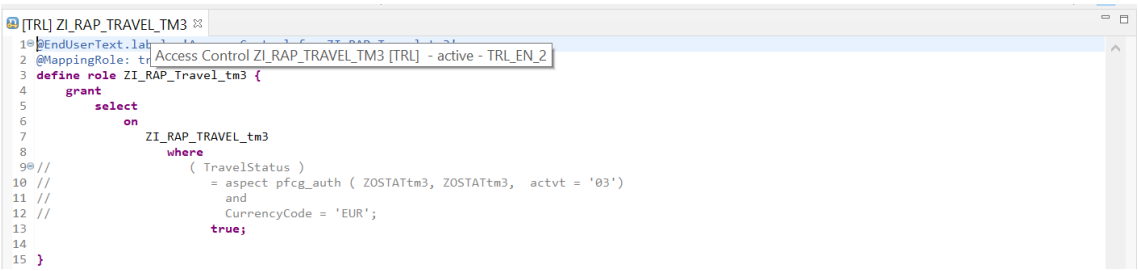




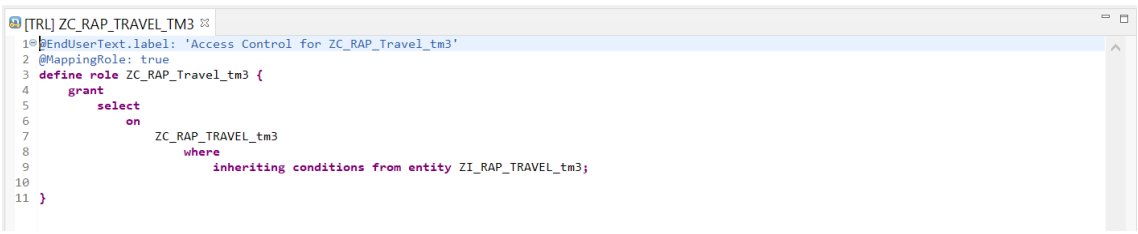


Neste screenshot estamos a permitir adicionar, criar, alterar, fazer display e apagar.

No screenshot abaixo estamos a definir o papel do CDS para a view do BO da viagem. Assim, vamos criar um novo controle de acesso. A anotação @MappingRole: true é definida no topo para atribuir a função CDS a todos os utilizadores, independentemente do cliente. O nome do CDS protegido é especificado após o select na declaração e as regras de acesso do utilizador são definidas na cláusula where. Em comentário está definido uma condição no elemento CurrencyCode. Apenas registos com o código de moeda EURO devem ser selecionados. Está em comentário porque nós queremos ter o acesso total aos dados e para isso podemos adicionar uma condição ou “true” na cláusula where do controle de acesso para o BO de viagem.



Conforme já explicado, uma função CDS deve ser definida explicitamente para cada entidade CDS. Não há herança implícita de regras de acesso. Portanto, vamos agora definir as regras de acesso para a visão de projeção do BO de viagem.



A anotação @MappingRole: true é definida na parte superior. O nome do CDS é especificado após a instrução grant select on, e, na cláusula da condição where, podemos definir a entidade CDS da qual as condições devem ser herdadas.

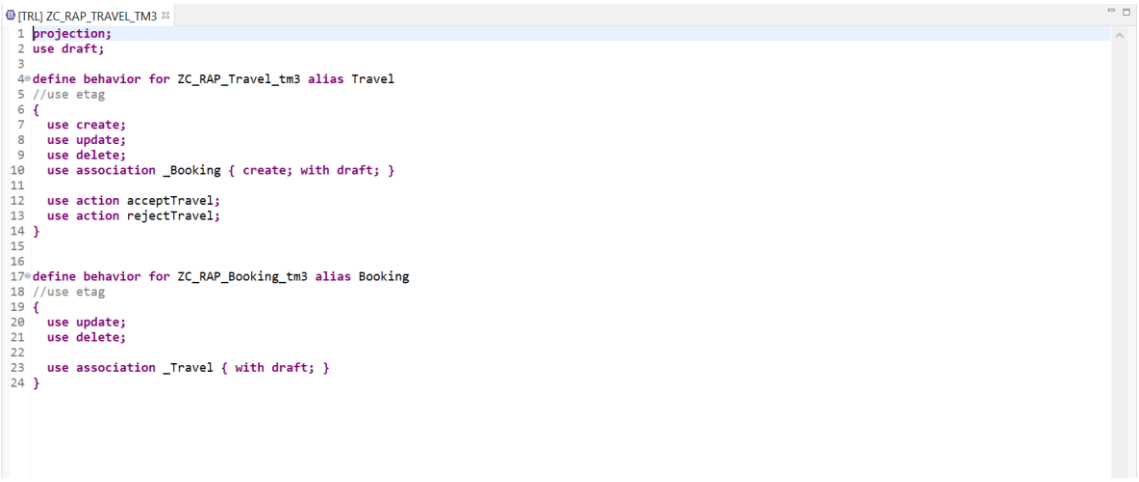
A próxima fase é ativar os recursos transacionais da nossa aplicação, criando o behavior definition do business object: neste teremos a definição básica do comportamento, que define o nosso business object das viagens managed com o create, update e delete, bem como as associações necessárias. É muito importante que o nome do behavior definition tenha o mesmo nome da root view do CDS. Neste anexo apenas estou a implementar a aplicação do tipo managed.
A primeira coisa que definimos é o alias para a viagem e a entidade de reserva. Depois, especificamos a persistência dando os nomes das tabelas da database para ambas as entidades. Isso permite o runtime managed de execução para executar as operações create, update e delete diretamente nas nossas tabelas database. O próximo passo é especificar o master lock para a entidade root. Para isso, adicionamos a linha 8 "lock master". O nó “filho” da reserva torna-se dependente do lock e faz uso da associação definida na CDS view para permitir ao transacional fazer a associação de viagens explicitamente listadas na entidade de reserva. Como este é um cenário baseado em UUID, queremos que o runtime managed forneça uma chave quando novas instâncias são criadas. Para isso, precisamos de fazer a numeração TravelUUID managed e somente de leitura. O mesmo é necessário para a entidade de reserva.

[TRL] ZI_RAP_TRAVEL_TM3

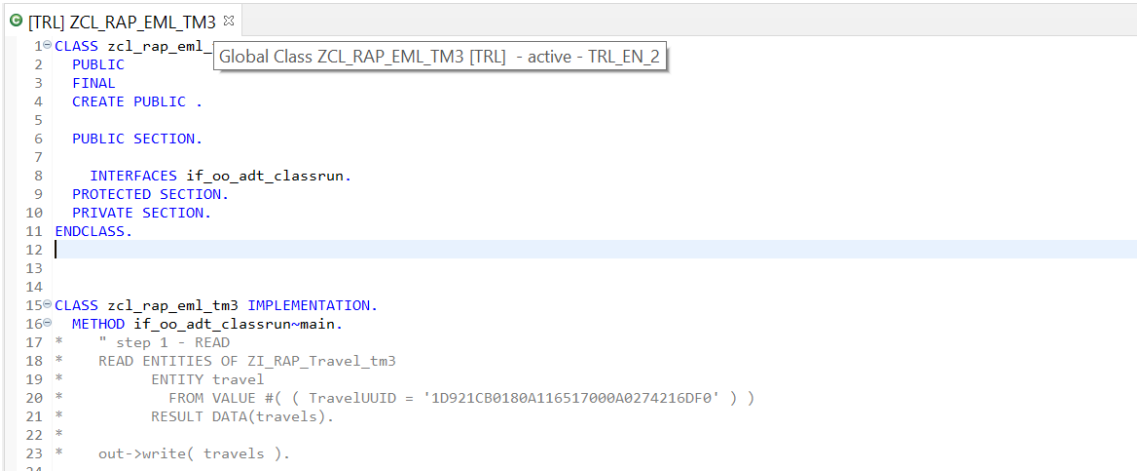
```
managed;
with draft;

define behavior for ZI_RAP_Travel_tm3 alias Travel
implementation in class zbp_i_rap_travel_tm3 unique
persistent table zrap_atrav_tm3
draft table zrap_dtrav_tm3
lock master total etag LastChangedAt
authorization master ( instance )
etag master LocalLastChangedAt
{
  create;
  update;
  delete;
  association _Booking { create; with draft; }

  field ( numbering : managed, readonly ) TravelUUID;
  field ( readonly  ) TravelId, TotalPrice, TravelStatus;
  field ( readonly  ) LastChangedAt, LastChangedBy, CreatedAt, CreatedBy, LocalLastChangedAt;
  field ( mandatory ) AgencyID, CustomerID;

  action ( features : instance ) acceptTravel result [1] $self;
  action ( features : instance ) rejectTravel result [1] $self;
  internal action recalcTotalPrice;

  determination setInitialStatus    on modify { create; }
  determination calculateTotalPrice on modify { field BookingFee, CurrencyCode; }
  determination calculateTravelID   on save   { create; }

  validation validateAgency   on save { field AgencyID; create; }
  validation validateCustomer on save { field CustomerID; create; }
  validation validateDates    on save { field BeginDate, EndDate; create; }

  draft determine action Prepare  {
    validation validateAgency;
    validation validateCustomer;
    validation validateDates;
  }

  mapping for zrap_atrav_tm3
  {
    TravelUUID        = travel_uuid;
    TravelID          = travel_id;
    AgencyID          = agency_id;
    CustomerID        = customer_id;
    BeginDate         = begin_date;
    EndDate           = end_date;
    BookingFee        = booking_fee;
    TotalPrice        = total_price;
    CurrencyCode      = currency_code;
    Description       = description;
    TravelStatus      = overall_status;
    CreatedBy         = created_by;
    CreatedBy         = created_by;
    CreatedAt         = created_at;
    LastChangedBy     = last_changed_by;
    LastChangedAt     = last_changed_at;
    LocalLastChangedAt = local_last_changed_at;
  }
}

define behavior for ZI_RAP_Booking_tm3 alias Booking
implementation in class zbp_i_rap_booking_tm3 unique
persistent table zrap_abook_tm3
draft table zrap_dbook_tm3
lock dependent by _Travel
authorization dependent by _Travel
etag master LocalLastChangedAt
{
  update;
  delete;

  association _Travel { with draft; }

  field ( numbering : managed, readonly ) BookingUUID;
  field ( readonly ) TravelUUID, BookingId;
  field ( readonly ) CreatedBy, LastChangedBy, LocalLastChangedAt;

  determination calculateBookingID  on modify { create; }
  determination calculateTotalPrice on modify { field FlightPrice, CurrencyCode; }

  mapping for zrap_abook_tm3
  {
    BookingUUID       = booking_uuid;
    TravelUUID        = travel_uuid;
    BookingID         = booking_id;
    BookingDate       = booking_date;
    CustomerID        = customer_id;
    CarrierID         = carrier_id;
    ConnectionID      = connection_id;
    FlightDate        = flight_date;
    FlightPrice       = flight_price;
    CurrencyCode      = currency_code;
    CreatedBy         = created_by;
    LastChangedBy     = last_changed_by;
    LocalLastChangedAt = local_last_changed_at;
  }
}
```

Vai aparecer um warning indicando que o TravelUUID na entidade de reserva deve ser definido para somente leitura, pois é usado na condição de associação. A solução para o problema é definir o master ETag em ambas as entidades. O outro warning que aparece diz respeito às informações de mapeamento que faltam. Como fornecemos aliases nas visualizações de CDS da interface para os nomes dos elementos, precisamos de dizer ao framework como mapear os nomes dos elementos no modelo de dados CDS para os campos da tabela correspondente. O mesmo precisa de ser feito para a entidade de reserva.

Passamos agora para o behavior definition, que projeta os recursos transacionais da behavior definition base. O nome da behavior definition deve ser idêntico ao da visão raiz do CDS.



As operações de create, update e delete são automaticamente assumidas a partir do behavior definition base por meio da keyword "use". A primeira coisa que definimos é o alias para a entidade viagem e reserva. Nós também queremos permitir a manipulação de Etag e para isso precisamos de adicionar isso para todas as entidades.
Relativamente à EML, é a linguagem de manipulação de entidades usadas para, por exemplo, adicionar determinações, validações ou ações ao behavior definition do business object. O EML padrão permite de forma segura read e modify o acesso a dados. Na criação desta funcionalidade no ADT é necessário adicionar a interface if_oo_adt_classrun. O valor uuid que aparece é retirado da tabela das viagens, porque funciona como key.
A seguir, conforme apresentado na fotografia, temos a primeira operação que vemos que é a operação de leitura, que permite o desempenho de uma leitura transacional de instâncias de business objects.



Leitura transacional significa que a leitura está a ocorrer no buffer transacional. Se a instância não está presente no buffer, ele é lido automaticamente no buffer da database pelo runtime managed.
As operações de modify são usadas para realizar alterações no buffer transacional. Se um registo não estiver presente no buffer, ele é lido da database antes da operação a ser executada. A modify create é usada para criar a instância raiz.
A modify update é usada para atualizar instâncias, sendo possível, através do Content ID, também atualizar instâncias que já foram criadas antes, mas que ainda não foram mantidas na database. A modify delete é usada para apagar instâncias.

```
[TRL] ZCL_RAP_EML_TM3
24
25 *    " step 2 - READ with Fields
26 *    READ ENTITIES OF ZI_RAP_Travel_tm3
27 *      ENTITY travel
28 *        FIELDS ( AgencyID CustomerID )
29 *      WITH VALUE #( ( TravelUUID = '1D921CB0180A116517000A0274216DF0' ) )
30 *      RESULT DATA(travels).
31 *
32 *    out->write( travels ).
33
34 *   " step 3 - READ with All Fields
35 *    READ ENTITIES OF ZI_RAP_Travel_tm3
36 *      ENTITY travel
37 *        ALL FIELDS
38 *      WITH VALUE #( ( TravelUUID = '1D921CB0180A116517000A0274216DF0' ) )
39 *      RESULT DATA(travels).
40 *
41 *    out->write( travels ).
42 *
43 *    " step 4 - READ By Association
44 *    READ ENTITIES OF ZI_RAP_Travel_tm3
45 *      ENTITY travel BY \_Booking
46 *        ALL FIELDS WITH VALUE #( ( TravelUUID = '1D921CB0180A116517000A0274216DF0' ) )
47 *      RESULT DATA(bookings).
48 *
49 *    out->write( bookings ).
50
51 *    " step 5 - Unsuccessful READ
52 *    READ ENTITIES OF ZI_RAP_Travel_tm3
53 *      ENTITY travel
54 *        ALL FIELDS WITH VALUE #( ( TravelUUID = '11111111111111111111111111111111' ) )
55 *      RESULT DATA(travels)
56 *      FAILED DATA(failed)
57 *      REPORTED DATA(reported).
58 *
59 *    out->write( travels ).
60 *    out->write( failed ).    " complex structures not supported by the console output
61 *    out->write( reported ).  " complex structures not supported by the console output
```

```
[TRL] ZCL_RAP_EML_TM3
63 *    " step 6 - MODIFY Update
64 *    MODIFY ENTITIES OF ZI_RAP_Travel_tm3
65 *      ENTITY travel
66 *        UPDATE
67 *          SET FIELDS WITH VALUE
68 *            #( ( TravelUUID  = '1D921CB0180A116517000A0274216DF0'
69 *                 Description = 'I like RAP@openSAP' ) )
70 *
71 *     FAILED DATA(failed)
72 *     REPORTED DATA(reported).
73 *
74 *    " step 6b - Commit Entities
75 *    COMMIT ENTITIES
76 *      RESPONSE OF ZI_RAP_Travel_tm3
77 *      FAILED     DATA(failed_commit)
78 *      REPORTED   DATA(reported_commit).
79 *
80 *    out->write( 'Update done' ).
81
82 *    " step 7 - MODIFY Create
83 *    MODIFY ENTITIES OF ZI_RAP_Travel_tm3
84 *      ENTITY travel
85 *        CREATE
86 *          SET FIELDS WITH VALUE
87 *            #( ( %cid       = 'MyContentID_1'
88 *                 AgencyID   = '70012'
89 *                 CustomerID = '14'
90 *                 BeginDate  = cl_abap_context_info=>get_system_date( )
91 *                 EndDate    = cl_abap_context_info=>get_system_date( ) + 10
92 *                 Description = 'I like RAP@openSAP' ) )
93 *
94 *     MAPPED DATA(mapped)
95 *     FAILED DATA(failed)
96 *     REPORTED DATA(reported).
97 *
98 *    out->write( mapped-travel ).
99 *
100 *    COMMIT ENTITIES
101 *      RESPONSE OF ZI_RAP_Travel_tm3
102 *      FAILED     DATA(failed_commit)
103 *      REPORTED   DATA(reported_commit).
104 *
105 *    out->write( 'Create done' ).
106
107 *    " step 8 - MODIFY Delete
108 *    MODIFY ENTITIES OF ZI_RAP_Travel_tm3
109 *      ENTITY travel
110 *        DELETE FROM
111 *          VALUE
112 *            #( ( TravelUUID  = '1D921CB0180A116517000A0274216DF0' ) )
113 *
114 *     FAILED DATA(failed)
115 *     REPORTED DATA(reported).
116 *
117 *    COMMIT ENTITIES
118 *      RESPONSE OF ZI_RAP_Travel_tm3
119 *      FAILED     DATA(failed_commit)
120 *      REPORTED   DATA(reported_commit).
121 *
122 *    out->write( 'Delete done' ).
123
124   ENDMETHOD.
125
126 ENDCLASS.
```

Na implementação do behavior, vamos usar as nossas próprias mensagens através de uma classe de exceção, “T100” geradas. Por isso, primeiro precisamos de criar uma classe para as nossas mensagens. Na criação é preciso ter em conta que a superclasse precisa de ser CX_STATIC_CHECK, e, na parte pública devemos adicionar a interface if_abap_behv_message.

[TRL] ZRAP_MSG_TM2

Message Class: ZRAP_MSG_TM2 (Total Messages: 5)

type filter text

Locked	Number	Short Text	Self E...	Last Changed ...	Changed On
	001	Begin date &1 must not be after end date &2 for travel &3	☑	CB0000000152	2021-02-11
	002	Begin date &1 must be on or after system date	☑	CB0000000152	2021-02-11
	003	Customer &1 unknown	☑	CB0000000152	2021-02-11
	004	Agency &1 unknown	☑	CB0000000152	2021-02-11
	005	You are not authorized to perform this activity	☑	CB0000000152	2021-02-11
	<Enter new val...		☑		

Vamos criar cinco mensagens, por exemplo com os números de um a cinco, adicionando as constantes para todas as cinco mensagens. O behavior implementation acontece nas Local

[TRL] ZCM_RAP_TM3

```
CLASS zcm_rap_tm3 DEFINITION
  PUBLIC
  INHERITING FROM cx_static_check
  FINAL
  CREATE PUBLIC .

  PUBLIC SECTION.

    INTERFACES if_t100_dyn_msg .
    INTERFACES if_t100_message .
    INTERFACES if_abap_behv_message.

    CONSTANTS:
      BEGIN OF date_interval,
        msgid TYPE symsgid VALUE 'ZRAP_MSG_tm3',
        msgno TYPE symsgno VALUE '001',
        attr1 TYPE scx_attrname VALUE 'BEGINDATE',
        attr2 TYPE scx_attrname VALUE 'ENDDATE',
        attr3 TYPE scx_attrname VALUE 'TRAVELID',
        attr4 TYPE scx_attrname VALUE '',
      END OF date_interval .
    CONSTANTS:
      BEGIN OF begin_date_before_system_date,
        msgid TYPE symsgid VALUE 'ZRAP_MSG_tm3',
        msgno TYPE symsgno VALUE '002',
        attr1 TYPE scx_attrname VALUE 'BEGINDATE',
        attr2 TYPE scx_attrname VALUE '',
        attr3 TYPE scx_attrname VALUE '',
        attr4 TYPE scx_attrname VALUE '',
      END OF begin_date_before_system_date .
    CONSTANTS:
      BEGIN OF customer_unknown,
        msgid TYPE symsgid VALUE 'ZRAP_MSG_tm3',
        msgno TYPE symsgno VALUE '003',
        attr1 TYPE scx_attrname VALUE 'CUSTOMERID',
        attr2 TYPE scx_attrname VALUE '',
        attr3 TYPE scx_attrname VALUE '',
        attr4 TYPE scx_attrname VALUE '',
      END OF customer_unknown .
    CONSTANTS:
      BEGIN OF agency_unknown,
        msgid TYPE symsgid VALUE 'ZRAP_MSG_tm3',
        msgno TYPE symsgno VALUE '004',
```

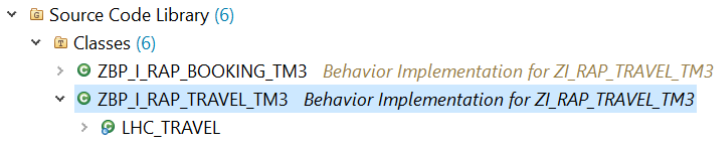
[TRL] ZCM_RAP_TM3

```
      END OF customer_unknown .
    CONSTANTS:
      BEGIN OF agency_unknown,
        msgid TYPE symsgid VALUE 'ZRAP_MSG_tm3',
        msgno TYPE symsgno VALUE '004',
        attr1 TYPE scx_attrname VALUE 'AGENCYID',
        attr2 TYPE scx_attrname VALUE '',
        attr3 TYPE scx_attrname VALUE '',
        attr4 TYPE scx_attrname VALUE '',
      END OF agency_unknown .
    CONSTANTS:
      BEGIN OF unauthorized,
        msgid TYPE symsgid VALUE 'ZRAP_MSG_tm3',
        msgno TYPE symsgno VALUE '005',
        attr1 TYPE scx_attrname VALUE '',
        attr2 TYPE scx_attrname VALUE '',
        attr3 TYPE scx_attrname VALUE '',
        attr4 TYPE scx_attrname VALUE '',
      END OF unauthorized .

    METHODS constructor
      IMPORTING
        severity   TYPE if_abap_behv_message=>t_severity DEFAULT if_abap_behv_message=>severity-error
        textid     LIKE if_t100_message=>t100key OPTIONAL
        previous   TYPE REF TO cx_root OPTIONAL
        begindate  TYPE /dmo/begin_date OPTIONAL
        enddate    TYPE /dmo/end_date OPTIONAL
        travelid   TYPE /dmo/travel_id OPTIONAL
        customerid TYPE /dmo/customer_id OPTIONAL
        agencyid   TYPE /dmo/agency_id  OPTIONAL
      .

    DATA begindate TYPE /dmo/begin_date READ-ONLY.
    DATA enddate TYPE /dmo/end_date READ-ONLY.
    DATA travelid TYPE string READ-ONLY.
    DATA customerid TYPE string READ-ONLY.
    DATA agencyid TYPE string READ-ONLY.

  PROTECTED SECTION.
  PRIVATE SECTION.
ENDCLASS.
```

Types tab geradas pela tool numa classe herdada, designada por cl_abap_behavior_handler. Na parte privada do código, adicionamos constantes para o status da viagem.

[TRL] ZBP_I_RAP_TRAVEL_TM3

```
CLASS zbp_i_rap_travel_tm3 DEFINITION PUBLIC ABSTRACT FINAL FOR BEHAVIOR OF zi_rap_travel_tm3.
ENDCLASS.

CLASS zbp_i_rap_travel_tm3 IMPLEMENTATION.
ENDCLASS.
```



O primeiro método que temos que implementar é o método acceptTravel, que define se o status é aceite para todas as keys fornecidas usando uma instrução de modification EML. Ele usa "in local mode", porque, por exemplo, é o que nos permite até mesmo mudar campos apenas de leitura enquanto saltamos o recurso e o controle de autorização.

Uma pequena nota: o uso de %tky significa chave de transação. No caso de a usarmos sem a funcionalidade draft, consideramos o mesmo valor da chave percentual, que é a chave da entidade relacionada. O uso da chave transacional reduz a necessidade para voltar a desenvolver a implementação quando permite por exemplo o rascunho, e é semelhante à ação de aceitar a viagem ou de a rejeitar. Agora a chave transactional vai ter automaticamente o indicador is_draft.

A primeira validação que vamos implementar é o método validateAgency: esta validação verifica se o AgencyID fornecido ao guardar, ou quando uma instância é criada, foi alterada.

[TRL] ZBP_I_RAP_TRAVEL_TM3

```
CLASS lhc_Travel DEFINITION INHERITING FROM cl_abap_behavior_handler.
  PRIVATE SECTION.

    CONSTANTS:
      BEGIN OF travel_status,
        open     TYPE c LENGTH 1  VALUE 'O', " Open
        accepted TYPE c LENGTH 1  VALUE 'A', " Accepted
        canceled TYPE c LENGTH 1  VALUE 'X', " Cancelled
      END OF travel_status.

    METHODS CalculateTravelID FOR DETERMINE ON SAVE
      IMPORTING keys FOR travel~CalculateTravelID.

    METHODS setInitialStatus FOR DETERMINE ON MODIFY
      IMPORTING keys FOR Travel~setInitialStatus.

    METHODS validateAgency FOR VALIDATE ON SAVE
      IMPORTING keys FOR Travel~validateAgency.

    METHODS validatecustomer FOR VALIDATE ON SAVE
      IMPORTING keys FOR travel~validatecustomer.

    METHODS validateDates FOR VALIDATE ON SAVE
      IMPORTING keys FOR Travel~validateDates.

    METHODS acceptTravel FOR MODIFY
      IMPORTING keys FOR ACTION Travel~acceptTravel RESULT result.

    METHODS rejectTravel FOR MODIFY
      IMPORTING keys FOR ACTION Travel~rejectTravel RESULT result.

    METHODS get_features FOR FEATURES
      IMPORTING keys REQUEST requested_features FOR Travel RESULT result.

    METHODS get_authorizations FOR AUTHORIZATION
      IMPORTING keys REQUEST requested_authorizations FOR Travel RESULT result.

    METHODS recalctotalprice FOR MODIFY
      IMPORTING keys FOR ACTION travel~recalcTotalPrice.

    METHODS calculateTotalPrice FOR DETERMINE ON MODIFY
      IMPORTING keys FOR Travel~calculateTotalPrice.

    METHODS is_update_granted IMPORTING has_before_image      TYPE abap_bool
                                        overall_status        TYPE /dmo/overall_status
                              RETURNING VALUE(update_granted) TYPE abap_bool.

    METHODS is_delete_granted IMPORTING has_before_image      TYPE abap_bool
                                        overall_status        TYPE /dmo/overall_status
                              RETURNING VALUE(delete_granted) TYPE abap_bool.

    METHODS is_create_granted RETURNING VALUE(create_granted) TYPE abap_bool.

ENDCLASS.
```

Validações são implementações que geralmente começam com a leitura dos dados necessários usando EML. No nosso caso, queremos ler o ID da agência para as chaves fornecidas.

```
METHOD validateAgency.
  " Read relevant travel instance data
  READ ENTITIES OF zi_rap_travel_tm3 IN LOCAL MODE
    ENTITY Travel
      FIELDS ( AgencyID ) WITH CORRESPONDING #( keys )
    RESULT DATA(travels).

  DATA agencies TYPE SORTED TABLE OF /dmo/agency WITH UNIQUE KEY agency_id.

  " Optimization of DB select: extract distinct non-initial agency IDs
  agencies = CORRESPONDING #( travels DISCARDING DUPLICATES MAPPING agency_id = AgencyID EXCEPT * ).
  DELETE agencies WHERE agency_id IS INITIAL.

  IF agencies IS NOT INITIAL.
    " Check if agency ID exist
    SELECT FROM /dmo/agency FIELDS agency_id
      FOR ALL ENTRIES IN @agencies
      WHERE agency_id = @agencies-agency_id
      INTO TABLE @DATA(agencies_db).
  ENDIF.

  " Raise msg for non existing and initial agencyID
  LOOP AT travels INTO DATA(travel).
    " Clear state messages that might exist
    APPEND VALUE #(  %tky               = travel-%tky
                     %state_area        = 'VALIDATE_AGENCY' )
      TO reported-travel.

    IF travel-AgencyID IS INITIAL OR NOT line_exists( agencies_db[ agency_id = travel-AgencyID ] ).
      APPEND VALUE #( %tky = travel-%tky ) TO failed-travel.

      APPEND VALUE #( %tky        = travel-%tky
                      %state_area = 'VALIDATE_AGENCY'
                      %msg        = NEW zcm_rap_tm3(
                                        severity = if_abap_behv_message=>severity-error
                                        textid   = zcm_rap_tm3=>agency_unknown
                                        agencyid = travel-AgencyID )
                      %element-AgencyID = if_abap_behv=>mk-on )
        TO reported-travel.
    ENDIF.
  ENDLOOP.
ENDMETHOD.
```

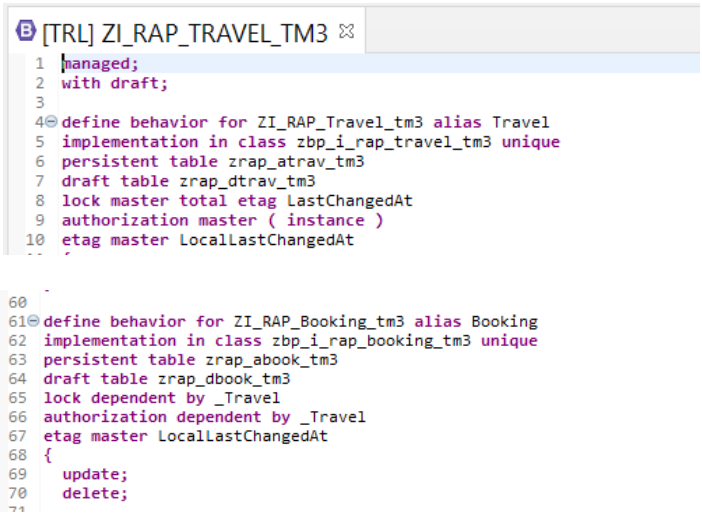
```
METHOD acceptTravel.
  " Set the new overall status
  MODIFY ENTITIES OF zi_rap_travel_tm3 IN LOCAL MODE
    ENTITY Travel
       UPDATE
         FIELDS ( TravelStatus )
         WITH VALUE #( FOR key IN keys
                         ( %tky          = key-%tky
                           TravelStatus = travel_status-accepted ) )
    FAILED failed
    REPORTED reported.

  " Fill the response table
  READ ENTITIES OF zi_rap_travel_tm3 IN LOCAL MODE
    ENTITY Travel
      ALL FIELDS WITH CORRESPONDING #( keys )
    RESULT DATA(travels).

  result = VALUE #( FOR travel IN travels
                      ( %tky   = travel-%tky
                        %param = travel ) ).
ENDMETHOD.
```

Foi criada uma tabela interna com todos os Ids das diversas agências e foi feito um select da database para confirmar que existem, verificando se um Agency ID foi fornecido e se está ativo. Se o AgencyID estiver vazio ou não existe na tabela, inserimos uma mensagem usando a nossa classe de exceção.

Existem mais métodos da travel, mas não acho que seja relevante expô-los todos aqui. Para além de que também temos que o fazer para o booking.

Uma nota para as determinações que precisam de ser idempotentes, isto é o resultado mantém-se sendo executadas várias vezes para a mesma key.

De seguida, vamos adicionar controle de autorização ao nosso business object.



A linha 8 "authorization master (instance)", é o que declara o nó raiz como authorization master. Relativamente à parte do booking fazemos a mesma mudança mais a especificação à associação _Travel. A authorization master diz que a entidade de reserva é dependente da autorização. Vão aparecer uns warnings que, usando a correção rápida, gera automaticamente as implementações para todos os três métodos.

Com desenvolvimento destas funcionalidades resultou a aplicação apresentada nos anexos do relatório.

Teresa Monteiro, 52597

18/02/2021