Generalitat de Catalunya
Centre de Telecomunicacions
i Tecnologies de la Informació

Framework Corporatiu J2EE

Connector GeCat

Barcelona, 17 / febrer / 2006

Històric de modificacions

Data	Autor	Comentaris	Versió
13/01/2006	Atos Origin, sae openTrends	Versió inicial del document	1.0

Llegenda de Marcadors

Índex

1. Introducció

1.1. Propósit

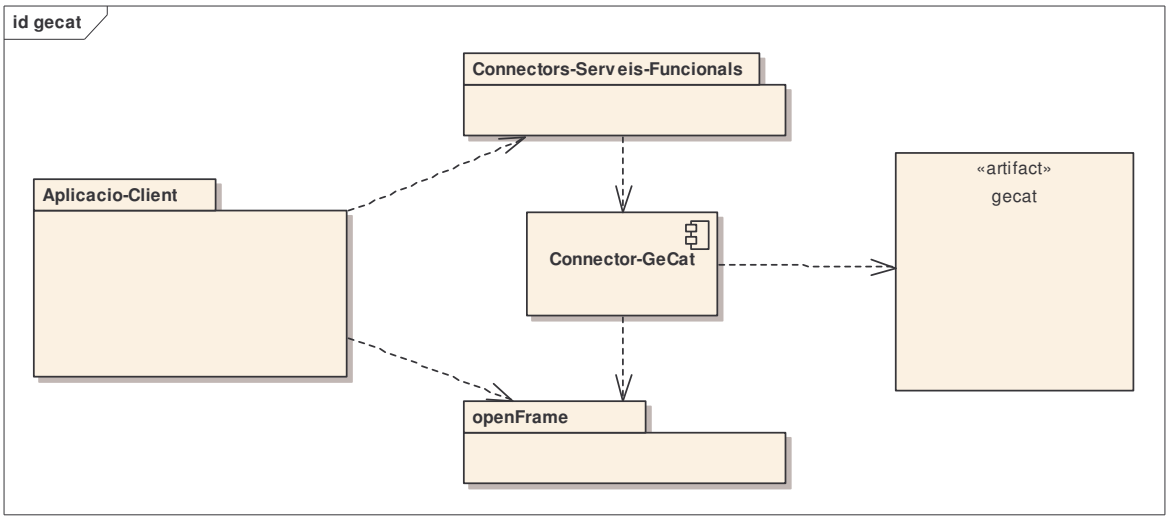
L'objectiu del present document és descriure la metodologia a seguir en la utilització del connector al sistema SAP del Gecat des de qualsevol aplicació Java del Framework J2EE.

L'abast d'aquest connector es basa en la utilització de les funcions d'alta de factures, consultes i reserves online del SAP de Gecat, així com totes les funcions batch.

1.2. Context i Escenaris d'Ús

Aquest connector podrà ser utilitzat des de qualsevol aplicació del Framework J2EE.

El connector permet l'accés al SAP mitjançant objectes de consulta. El connector transforma aquests objectes en una cadena de caràcters vàlida per al SAP. La cadena de retorn és transformada novament en un objecte que conté els registres de retorn.



1.3. Versions i Dependències

En el present apartat es mostren quines són les versions i dependències necessàries per fer ús del Servei.

Nom	Tipus	Versió	Descripció
openFrame-services-validation	Jar	1.0	
openFrame-services-exceptions	Jar	1.0	

Nom	Tipus	Versió	Descripció
openFrame-services-logging	Jar	1.0	
openFrame-services-i18n	Jar	1.0	
openFrame-services-sap	Jar	1.0	
jaxb-api-1.0.1.jar	Jar	1.0.1	
jaxb-libs-1.0.1.jar	Jar	1.0.1	
jaxb-impl-1.0.1.jar	Jar	1.0.1	
jax-qname-1.1.jar	Jar	1.1	
namespace-1.0.1.jar	Jar	1.0.1	
relaxngDatatype-1.0.1.jar	Jar	1.0.1	
xsdlib-1.1.2.jar	jar	1.1.2	

1.4. A qui va dirigit

Aquest document va dirigit als següents perfils:

- Programador. Per conéixer l'ús del connector.
- Arquitecte. Per conéixer quins són els components i la configuració del connector.

1.5. Documents i Fonts de Referència

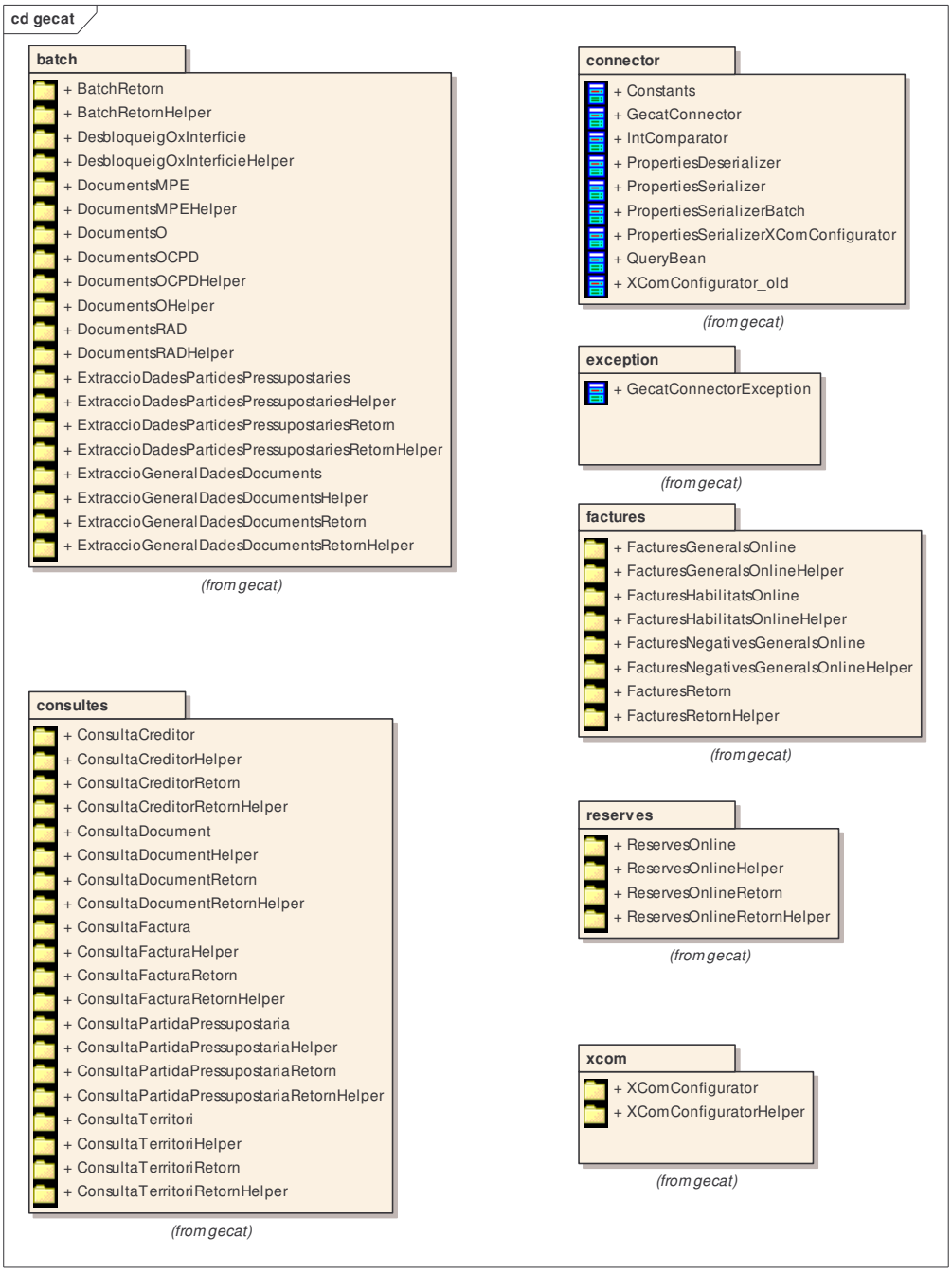
[1]	Anàlisi Funcional - Connector Serveis Funcionals - 16-Nov.doc	Document d'anàlisi dels connectors de serveis funcionals.
[2]	Consultes a SAP_Annex.doc	
[3]	XCOM_GECAT.doc	
[4]	XCOM_SAP_to_REMOTS.doc	
[5]	XCOM-HOST.doc	
[6]	Crides Java a GECAT.pdf	

2. Descripció Detallada

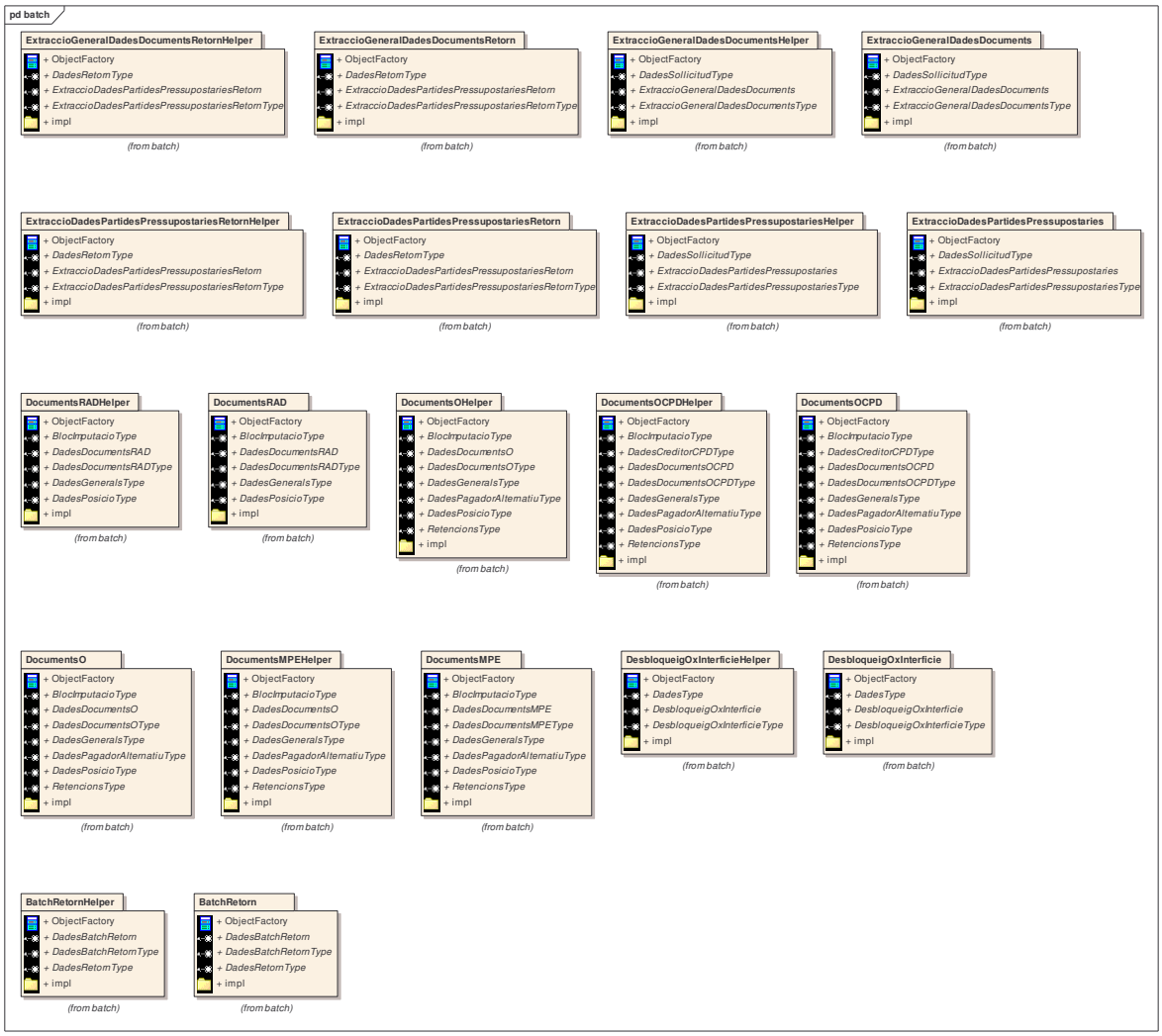
2.1. Arquitectura i Components

2.1.1. Interfícies i Components Genérics

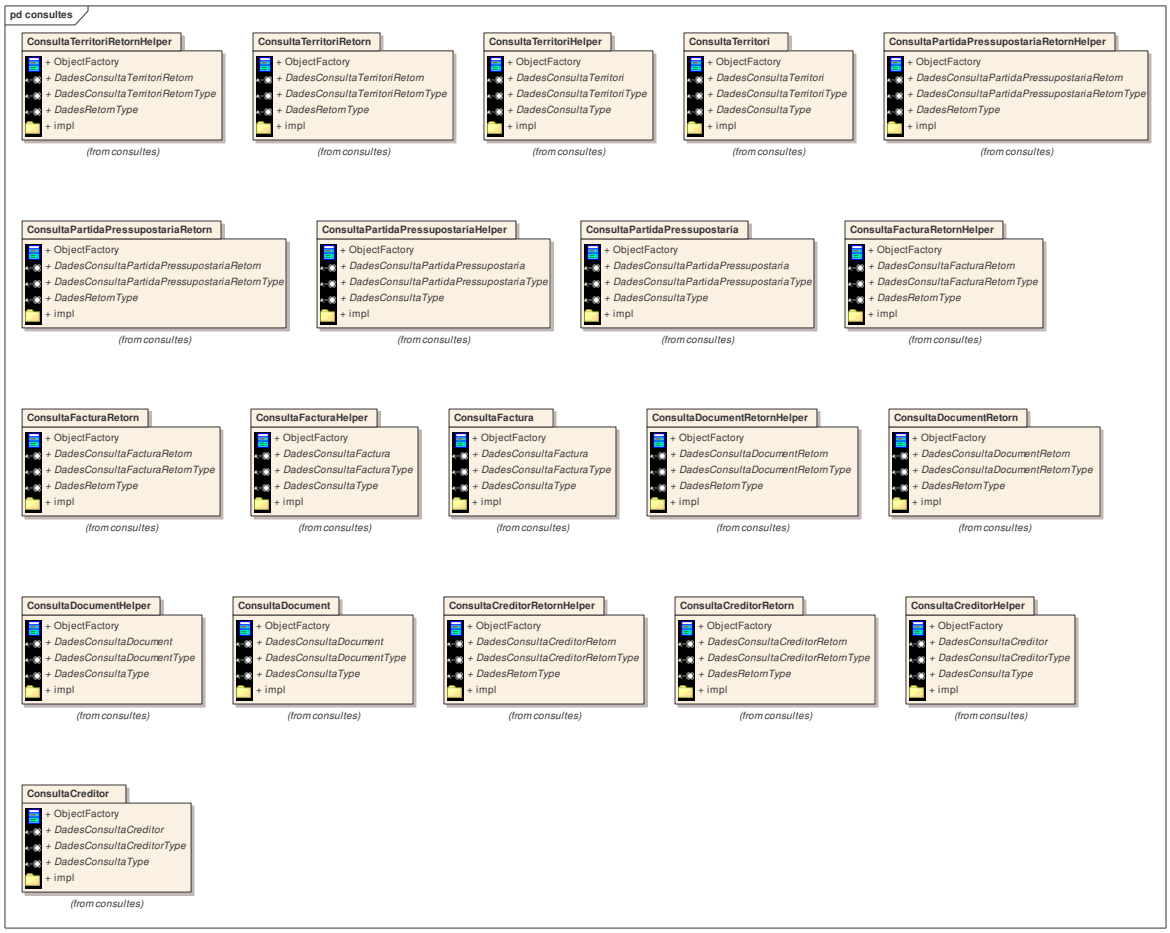
El connector defineix les següents interfícies:



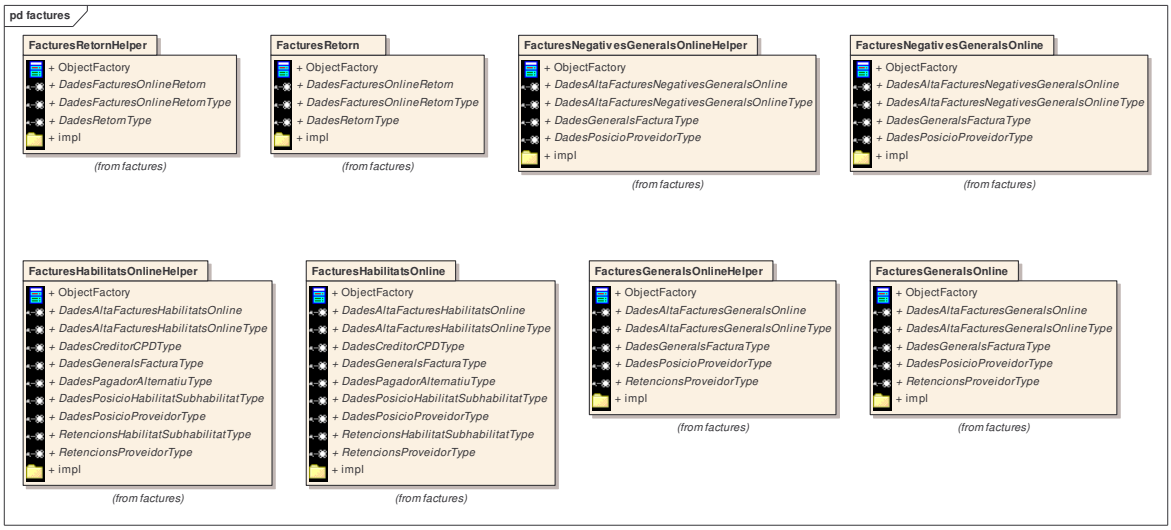
Dins el package *batch*:



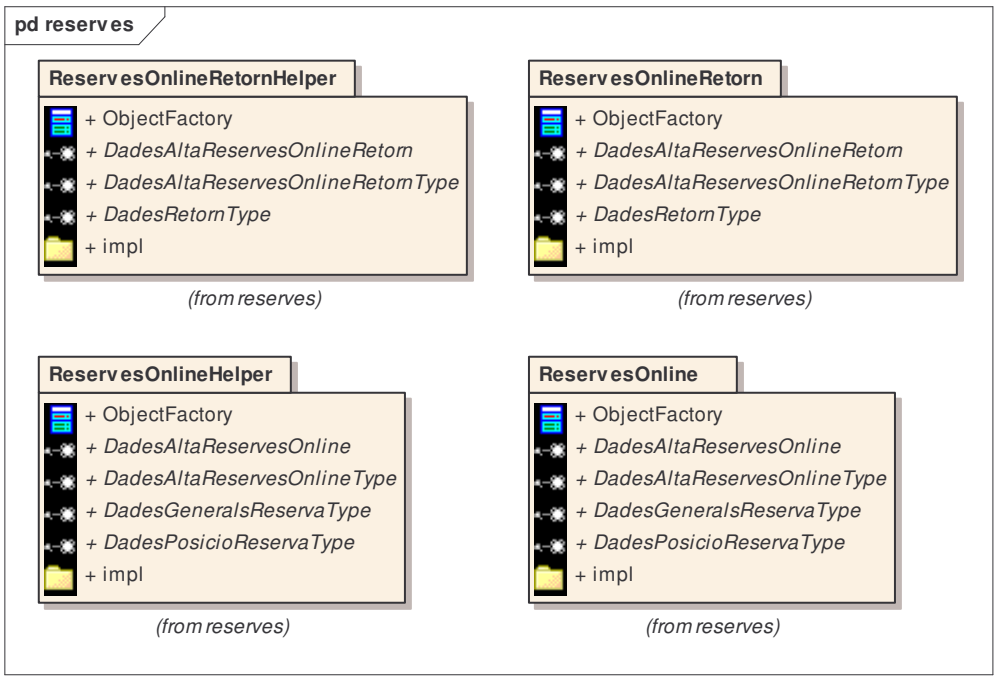
Dins el package *consultes*:



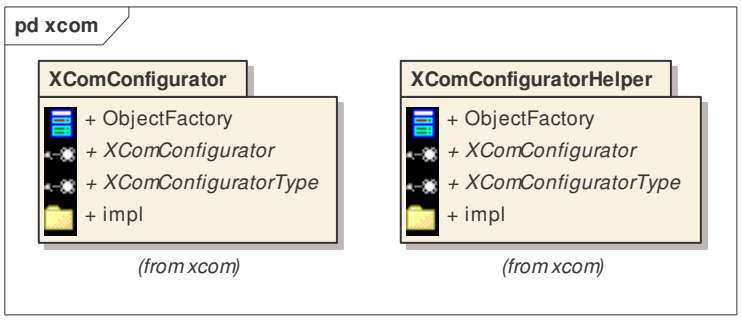
Dins el package *factures:*



Dins *reserves*:



Dins *xcom* :



2.1.2. Components basats en Commons i Spring

Component	Package	Descripció
GecatConnectorException	net.gencat.gecat.exception	Extends SystemException.

2.2. Instal·lació i Configuració

2.2.1. Instal·lació i Configuració

Per poder utilitzar el connector haurem de configurar el projecte perque inclogui 2 llibreries dll i diferents jars:

Les llibreries són **sapjcorfc.dll** i **librfc32.dll** i s'han de copiar al directori system32.

Per fer les operacions sobre el sistema SAP del Gecat es necessària la utilització del jar **sapjco-2.1.6**, que és el connector propi del SAP per fer les crides a les seves funcions BAPI (RFC).
Les classes que utilitzarem per fer els objectes d'operacions estan en el jar del connector Gecat **openFrame-connectors-gecat-1.0-SNAPSHOT.jar**
Les classes que utilitzarem per fer les operacions han estat generades amb una eina Open Source anomenada **JAXB** (Java API for Xml Binding), que genera classes java a partir d'un esquema de xml (XMLSchema). Aquesta eina ens permet no solament generar automàticament classes java sinó també fer validacions per comprovar que les dades que contenen els objectes són vàlides. Per això també haurem d'incloure els jars que necessita JAXB:

- **jaxb-api-1.0.1.jar**
- **jaxb-libs-1.0.1.jar**
- **jaxb-impl-1.0.1.jar**
- **jax-qname-1.1.jar**
- **namespace-1.0.1.jar**
- **relaxngDatatype-1.0.1.jar**
- **xsdlib-1.1.2.jar**

Aquests jars els inclourem mitjançant les dependencies de project.xml, tal com acabem d'explicar.
Ara executarem el goal de Maven 'opentrends:generate-classpath' per regenerar la configuració del classpath del projecte en funció del fitxer de project.xml. Aquest goal està definit dins de maven.xml:

```
<goal name="opentrends:generate-classpath"
        description="Shortcut in Eclipse for the goal eclipse:generate-classpath">
        <attainGoal name="eclipse:generate-classpath"/>
</goal>
```

2.3. Utilització del Servei

La utilització del Connector es basa principalment en la configuració. L'ús directe des dels clients es permet mitjançant les interfícies definides.

El connector del Gecat ofereix tres operacions: alta de factures online, consultes i reserves. Cada operació té una classe que conté tota la informació de l'operació (o crida) i un altra amb la informació que retorna SAP per aquesta operació. Cada classe conté un altra classe per cada línia que tingui l'operació.

Si per exemple tenim l'operació consultaFactura, que conté una sola línia, daddadesConsulta, haurem de crear un objecte de la classe DadesConsultaFacturaType i un altra de DadesConsultaType. Aquesta última l'omplirem amb totes les dades que facin falta mitjançant els seus mètodes setters (setSocietat, setCodiCreditor, etc..). Un cop omplert del tot aquest objecte (que representa una sola línia), l'l'assignarem a l'altre (que és el que representa tota la consulta) mitjançant també un mètode setter: dadesConsultaFactura.setDadesConsulta(dadesConsulta). Si tingués més línies, hauríem de fer el mateix per a totes.

En el cas en que una mateixa línia pugui aparèixer més d'una vegada la forma de fer les operacions canvia, ja que el conjunt de totes les línies del mateix tipus aniran dins d'una llista. Un cop hem creat l'objecte de crida amb totes les dades necessàries, utilitzarem la classe GecatConnector per obtenir l'objecte de retorn amb les dades retornades pel SAP.

2.4. Integració amb Altres Serveis

2.4.1. Integració amb el Servei de Internacionalització

En els fitxers de configuració es defineixen claus que permeten especificar quins missatges retornar en cas errors. Per a poder traduir aquestes claus és necessari especificar que el connector usarà el Servei d'Internacionalització (veure Configuració).

3. Exemples

3.1. Crides a Gecat sense línies repetides

1. Retorn sense línies repetides

Per mostrar aquest tipus de crida agafarem com exemple la consulta de partida pressupostària.

Primerament crearem tots els objectes que necessitarem per fer el procediment:

```
        net.gencat.gecat.consultes.ConsultaPartidaPressupostaria.ObjectFactory
objectFactory = new
net.gencat.gecat.consultes.ConsultaPartidaPressupostaria.ObjectFactory();
```

La classe ObjectFactory és, com el seu nom indica, una classe per crear objectes d'altres classes. Aquestes classes són les que ofereix JAXB per crear objectes amb una certa estructura predefinida. Crearem un objecte de la classe ObjectFactory per crear els objectes que serviran per fer la crida (objectFactory).

Seguidament crearem un objecte que representarà la crida completa. Aquest objecte contindrà un objecte per a cada línia que tingui la crida. Aquests objectes són els que s'han de completar amb la informació de la crida.

```
                DadesConsultaPartidaPressupostariaType
dadesConsultaPartidaPressupostaria =
objectFactory.createDadesConsultaPartidaPressupostariaType();

        DadesConsultaType dadesConsulta = objectFactory.createDadesConsultaType();

        dadesConsulta.setCentreGestor("1006");

        dadesConsulta.setEntitatCP("1000");

        dadesConsulta.setExercici("2002");

        dadesConsulta.setPosicioPressupostaria("D/220100100/4551");

        dadesConsulta.setVinculacio("S");

        dadesConsultaPartidaPressupostaria.setDadesConsulta(dadesConsulta);
```

Amb aquest objecte ja podem fer la crida al SAP mitjançant el mètode estàtic *consultaPartidaPressupostaria* de la classe **GecatConnector**. Aquesta funció requereix de l'objecte de consulta.

```
      DadesConsultaPartidaPressupostariaRetornType dadesConsultaRetorn =
GecatConnector.consultaPartidaPressupostaria(dadesConsultaPartidaPressupostaria);
```

En aquest punt ja tenim l'objecte de retorn llest per a ser utilitzat per l'aplicació.

```
System.out.println("getDenominacio():" +
dadesConsultaRetorn.getDadesRetorn().getDenominacio());
```

2. Retorn amb línies repetides

Per mostrar aquest tipus de crida agafarem com exemple la consulta de creditor.

Primerament crearem tots els objectes que necessitarem per fer el procediment:

```
        net.gencat.gecat.consultes.ConsultaCreditor.ObjectFactory objectFactory =
new net.gencat.gecat.consultes.ConsultaCreditor.ObjectFactory();
```

La classe ObjectFactory és, com el seu nom indica, una classe per crear objectes d'altres classes. Aquestes classes són les que ofereix JAXB per crear objectes amb una certa estructura predefinida. Crearem un objecte de la classe ObjectFactory per crear els objectes que serviran per fer la crida (objectFactory).

Seguidament crearem un objecte que representarà la crida completa. Aquest objecte contindrà un objecte per a cada línia que tingui la crida.

```
        DadesConsultaCreditorType dadesConsultaCreditor =
objectFactory.createDadesConsultaCreditorType();

        DadesConsultaType dadesConsulta = objectFactory.createDadesConsultaType();

        dadesConsulta.setNIFoCodiCreditor("A53062927");

        dadesConsulta.setRegistreInicial("001");

        dadesConsulta.setSocietat("1000");

        dadesConsultaCreditor.setDadesConsulta(dadesConsulta);
```

Haurem de passar al connector l'objecte de consulta mitjançant el mètode estàtic *consultaCreditor* de la classe **GecatConnector**:

```
     DadesConsultaCreditorRetornType dadesConsultaRetorn =
GecatConnector.consultaCreditor(dadesConsultaCreditor);
```

Ara ja tenim l'objecte de retorn llest per a ser utilitzat:

```
List finalList = dadesConsultaRetorn.getDadesRetorn().getDadaRetorn();

      System.out.println("dadaRetorn.getAdreca:" +
              ((DadaRetornType)finalList.get(0)).getAdreca());
```

3.2. Crides a Gecat amb línies repetides

Per mostrar aquest altre tipus de crida prendrem com exemple l'alta de factures habilitats.

Crearem tots els objectes que necessitarem per fer el procediment:

```
      net.gencat.gecat.factures.Factures.ObjectFactory objectFactory = new
net.gencat.gecat.factures.Factures.ObjectFactory();
```

La classe ObjectFactory és, com el seu nom indica, una classe per crear objectes d'altres classes. Aquestes classes són les que ofereix JAXB per crear objectes amb una certa estructura predefinida. Crearem un objecte de la classe ObjectFactory per crear els objectes que serviran per fer la crida (objectFactory).

Seguidament crearem un objecte que representarà la crida completa. Aquest objecte contindrà un objecte per a cada línia (o conjunt de línies) que tingui la crida.

```
      DadesAltaFacturesHabilitatsOnlineType dadesAltaFacturaHabilitats =
objectFactory.createDadesAltaFacturesHabilitatsOnlineType();

       DadesGeneralsFacturaType dadesGeneralsFactura =
objectFactory.createDadesGeneralsFacturaType();

      dadesGeneralsFactura.setDataCompt("24051980");
      dadesGeneralsFactura.setDataDocument("01");
       .
       .
       .
```

```
        dadesGeneralsFactura.setTransaccio("ZD21");

        dadesAltaFacturaHabilitats.setDadesGeneralsFactura(dadesGeneralsFactura);
```

Si la línia pot estar repetida, com en el cas de *retencions habilitat subhabilitat*, haurem de crear una llista d'objectes que representen una línia:

```
        RetencionsHabilitatSubhabilitatType retencionsHS =
objectFactory.createRetencionsHabilitatSubhabilitatType();

        List retencionsHSList = retencionsHS.getRetencioHabilitatSubhabilitat();

        RetencioHabilitatSubhabilitatType retencioHS =
objectFactory.createRetencionsHabilitatSubhabilitatTypeRetencioHabilitatSubhabilit
atType();

        retencioHS.setImportBase("1500");
        retencioHS.setImportRetencio("25000");
        retencioHS.setIndicadorRetencio("01");
        retencioHS.setTipusRegistre("3");
        retencioHS.setOrder(new BigInteger("0"));

        retencionsHSList.add(retencioHS);

        retencioHS =
(RetencioHabilitatSubhabilitatType)retencioHS.getClass().newInstance();

        retencioHS.setImportBase("1600");
        retencioHS.setImportRetencio("26000");
        retencioHS.setIndicadorRetencio("01");
        retencioHS.setTipusRegistre("3");
        retencioHS.setOrder(new BigInteger("1"));

        retencionsHSList.add(retencioHS);


dadesAltaFacturaHabilitats.setRetencionsHabilitatSubhabilitat(retencionsHS);
```

Fixem-nos que és important donar un ordre a les línies contingudes en la llista:

```
retencioHS.setOrder(0);
```

Amb aquest objecte ja podem fer la crida al SAP, mitjançant el mètode estàtic *altaFacturesHabilitats* de la classe **GecatConnector**:

```
        DadesConsultaCreditorRetornType dadesConsultaRetorn =
    GecatConnector.consultaCreditor(dadesConsultaCreditor);
```

Ara ja podem disposem de l'objecte de retorn per a ser utilitzat:

```
            List finalList = dadesAltaFacturaRetorn.getDadesRetorn().getDadaRetorn();

            System.out.println("dadaRetorn.getClasseDocument():" +
                    ((DadaRetornType)finalList.get(0)).getClasseDocument());
```