E5.1V2 SERVICIOS DE INTEGRACIÓN

ESPACIO DE DATOS LINGÜÍSTICO (SER 15/23 OTT)

Resumen

El documento describe la integración entre la plataforma European Language Grid (ELG) y el espacio de datos lingüístico (EDL) INESData. Se ha creado un conector exclusivo para ELG en el EDL que se encarga de publicar en el catálogo federado del EDL los recursos lingüísticos alojados en ELG. Se describe el proceso técnico de creación del conector de ELG, incluyendo la configuración de la base de datos, almacenamiento y autenticación. Para publicar los recursos de ELG en EDL el conector de ELG usa dos componentes principales: un script para cargar metadatos desde ELG al espacio de datos lingüístico, filtrando recursos relevantes, y un servicio web intermediario que facilita la descarga de recursos desde ELG. Ambos componentes están dockerizados y configurados para su despliegue en el EDL.

Cristian Berrío Aroca
Andrés García-Silva

19/12/2024
Expert.ai Expert.ai Language Technology Research Lab

Historia

revisión	Fecha	Descripción	Autor
V2.0	11/12/24	Tabla de contenidos	Cristian Berrío
V2.1	18/12/24	Primer borrador	Cristian Berrío
V2.2	19/12/24	Revisión del documento	Andrés García-Silva
V2.3	19/12/24	Añadido diagrama de componentes en introducción	Cristian Berrío

Contenido

1 Introducción

Este documento contiene información de la implementación del servicio de integración de la plataforma European Language Grid (ELG) en el espacio de datos lingüístico EDL. La integración entre ambas plataformas se realiza por medio de un conector creado exclusivamente para ELG en el EDL. Este conector usa un servicio que permite la extracción de metadatos de los recursos en ELG para su posterior carga como activos en el EDL. El servicio, además, permite la descarga de recursos alojados en ELG, a través de un adaptador. Observe que el conector de ELG solo importa los metadatos de los recursos, los datos permanecen en la plataforma ELG y solo son descargados directamente desde ELG cuando un usuario del EDL lo requiera. En la Figura 1 se presenta un diagrama con los diferentes componentes para la integración de ELG en el espacio de datos lingüístico.

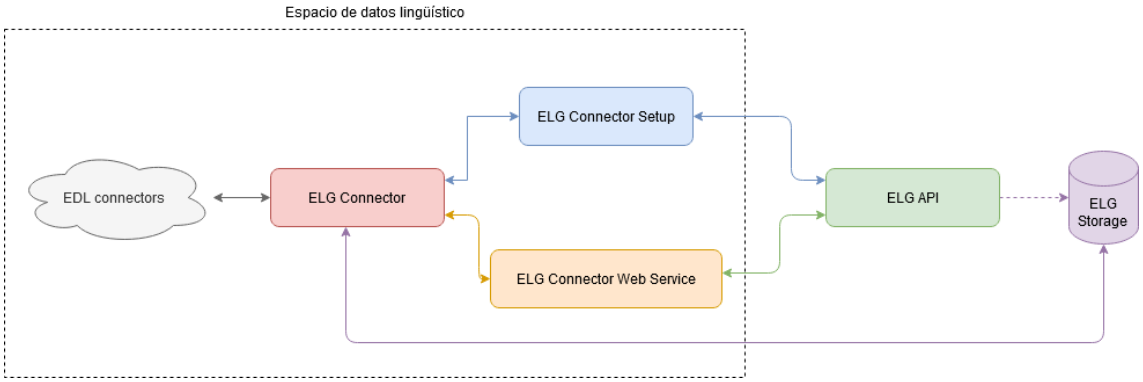

Figura 1. Diagrama de los componentes del conector para ELG en el espacio de datos lingüístico.

2 Integración de ELG en el espacio de datos lingüístico

En la primera versión de este entregable (E5.1v1), se hizo una pequeña presentación de la plataforma ELG. Finalmente, se ha decidido optar por el desarrollo de un conector para esta plataforma. El conector de ELG expone en el EDL los recursos lingüísticos de tipo Corpus, o léxico/conceptuales que estén alojados en el grid de ELG que tengan una licencia de tipo abierta. Los recursos de ELG se integran en el EDL como activos del conector ELG que además les asigna un contrato con políticas de acceso y contratación por defecto para que sean visibles al resto de participantes en el EDL. Se exige que el recurso este alojado en el grid de ELG para garantizar que pueda descargarse automáticamente cuando se realiza la transferencia en el EDL.

2.1 Carga de datos en la configuración del conector de ELG

Para la carga de datos de ELG en el conector del espacio de datos lingüístico, se ha desarrollado un script en Python. Este script utiliza la librería en Python de ELG[1] para hacer las búsquedas en su catálogo. En concreto los parámetros de búsqueda son los siguientes:

- Tipo de recurso: se buscan recursos que sean del tipo “Corpus” o “Lexical/Conceptual resource", excluyendo de esta forma otros tipos de recursos como ejemplo, servicios.
- Licencia: Se han identificado cuáles son las licencias abiertas de los recursos en ELG, obteniendo la siguiente lista:
    - Apache-2.0
    - CC-BY-4.0
    - CC-BY-NC-4.0
    - CC-BY-NC-ND-4.0
    - CC-BY-NC-SA-2.0

[1] https://pypi.org/project/elg/

  - CC-BY-NC-SA-2.5
  - CC-BY-NC-SA-4.0
  - CC-BY-SA-4.0
  - CC0-1.0

  De esta forma, los recursos que tengan licencias privativas o de otro tipo, no se incluyen en el catálogo del conector.

- Alojamiento en ELG: se filtra por recursos que estén alojados en ELG, excluyendo aquellos que puedan estar en otros sitios.

Una vez se realiza la búsqueda en ELG, se añade al catálogo del conector aquellos recursos que cumplan con los filtros y que previamente no se encuentren ya en este. A la hora de crear el activo, en el campo "baseUrl" de “dataAddress” se incluye una referencia a un servicio web adaptador que intermedia entre EDL y ELG para poder descargarse el recurso. El servicio web adaptador es descrito en la siguiente sección. Una vez se han insertado todos los recursos, el script comprueba si existe al menos un contrato, y de no existir, crea uno asociado a la política “always-true”, preexistente en el conector. Esto se hace para que los recursos puedan ser accesibles en el catálogo del EDL para otros conectores.

Se ha creado una imagen de Docker con el script, para que dentro del docker-compose del EDL INESData[2], se carguen los datos una vez se ha levantado el conector de ELG. El script a su vez utiliza variables de entorno para su configuración, los valores por defecto de estas variables son:

AUTHENTICATION_ENDPOINT = http://keycloak:8080/realms/dataspace/protocol/openid-connect/token

ELG_CONNECTOR_ENDPOINT = http://connector-elg:39193

ELG_WS_ENDPOINT = http://elg-connector-ws:5000/

Para las credenciales que se utilizan para hacer el inicio de sesión de cara al conector, se utilizan las variables de entorno USER y PASSWORD. Estas variables de entorno se pueden configurar en el docker-compose de la siguiente forma:

```
setup-elg-connector:
  image: setup-elg-connector
  depends_on:
    - connector-elg
  environment:
    - AUTHENTICATION_ENDPOINT=http://keycloak:8080/my_new_endpoint
    - USER=myuser
    - PASSWORD=mypassword
```

El código del script junto a los archivos necesarios para la dockerización se encuentra en https://github.com/oeg-upm/inesdata-espacio-linguistico-elg-connector.

2.2 Servicio web para descarga de recursos alojados ELG

La forma de descargar archivos alojados en ELG es haciendo una petición POST al endpoint https://live.european-language-grid.eu/catalogue_backend/api/management/download/{id}/, sin embargo, esta url no se puede usar a la hora de crear el activo en el EDL, puesto que cuando el conector se intente descargar el recurso, lo hará con una petición GET. Por tanto, se ha desarrollado un servicio web intermediario que resuelve este problema.

En concreto, se trata de un servicio web que utiliza el framework Flask[3]. El servicio recibe el id del recurso en ELG, y a través del cliente para Python de ELG, se trae los metadatos necesarios

[2] https://github.com/oeg-upm/inesdata-espacio-linguistico

[3] https://flask.palletsprojects.com/en/stable/

para hacer la descarga, en concreto el parámetro “elg-resource-distribution-id”. En ELG un recurso puede tener asociado más de una forma de distribución, por defecto, el servicio web descargará la primera de ellas en caso de que tenga más de una distribución.

Por otra parte, la llamada al *endpoint* para las descargas en ELG requiere de un token de acceso. Este token generalmente se obtiene a través de un inicio de sesión con credenciales de forma interactiva, pero ELG permite crear *offline* tokens sin una fecha de expiración a corto plazo. Este token se puede guardar en un archivo json, y posteriormente el servicio web lo puede usar para realizar la autenticación y obtener el token de acceso.

Una vez se hace la llamada al *endpoint*, este devuelve una URL desde donde se puede descargar el recurso. El servicio web por tanto devuelve esta URL con un estado de respuesta 307 (redirección temporal), para que el conector finalmente descargue el recurso de dicha URL.

Al igual que el script para la carga de datos, se ha creado una imagen Docker para el servicio web. Esta imagen de docker se incluye dentro del docker compose del EDL INESData[4] junto al conector para ELG. El servicio web dentro del contenedor escucha las peticiones por el puerto 5000, pudiéndose configurar este puerto a través del archivo “app.ini”. El servicio web en el docker compose tiene la siguiente forma:

```
elg-connector-ws:
  image: elg_connector_ws
  ports:
    - "5000:5000"
```

El código del servicio web del conector de ELG se encuentra en https://github.com/oeg-upm/inesdata-espacio-linguistico-elg-web-service. Este repositorio contiene las instrucciones de cómo generar el *offline* token requerido para la autenticación en ELG, así como el comando para crear la imagen docker del servicio web.

2.3 Conector ELG en el espacio de datos lingüístico INESData

A continuación, se presentan los pasos que se ha seguido para crear un nuevo conector basado en la versión 0.8.0 del conector de INESData[5].

2.3.1 Base de datos postgres

Dentro del docker-compose.yml del EDL INDESData[6], en la variable de entorno POSTGRES_MULTIPLE_DATABASES del servicio “postgres-common”, se ha añadido una nueva base de datos para el conector de ELG, en concreto se ha añadido a la lista “connectorelg”. Por otra parte, en la lista de volúmenes del servicio “postgres-common”, se ha añadido el siguiente mapeo:

```
- ./resources/db-init/connector/:/docker-entrypoint-
initdb.d/connectorelg
```

A demás, se ha modificado el archivo resources/db-init/registrationservice/participant.sql, añadiendo el siguiente comando SQL:

```
INSERT INTO edc_participant (participant_id,url,created_at,shared_url)
VALUES ('connector-elg','
http://connector-elg:39194/protocol',42893849,'http://connector-
elg:39196/shared');
```

[4] https://github.com/oeg-upm/inesdata-espacio-linguistico

[5] https://github.com/oeg-upm/inesdata-local-env/tree/v0.8.0

[6] https://github.com/oeg-upm/inesdata-espacio-linguistico

2.3.2 Repositorio de almacenamiento MinIO

Dentro del docker-compose.yml, se ha creado un nuevo servicio llamado “minio-elg”, basado en los servicios ya existentes para los conectores 1 y 2. Específicamente, el nuevo servicio ha quedado de la siguiente forma, marcando en negrita las principales diferencias con respecto a los servicios minio de los otros conectores:

```
minio-elg:
  image: minio/minio
  ports:
    - '9020:9000'
    - '9021:9021'
  volumes:
    - mdata-elg:/data
  environment:
    - MINIO_ROOT_USER=inesdata
    - MINIO_ROOT_PASSWORD=inesdata
    - MINIO_DEFAULT_BUCKETS=bucket-connector-elg
  command: server --console-address ":9021" /data
  healthcheck:
    test: timeout 5s bash -c ':> /dev/tcp/127.0.0.1/9000' || exit 1
    interval: 30s
    timeout: 20s
    retries: 3
```

En volúmenes (volumes), hay un nuevo volumen para el conector de ELG que se mapea a /data, este nuevo volumen por tanto se ha añadido a la lista de volúmenes del docker-compose.

```
volumes:
  …
  mdata-elg:
    driver: local
```

El servicio para la configuración de minio, “setup-minio”, se modificado añadiendo “minio-elg” a la lista de “depends_on”. Además, a la lista de comandos del entrypoint, se han añadido los siguientes comandos antes del “exit 0”:

```
/usr/bin/mc alias set minio-elg http://minio-elg:9000 inesdata
inesdata;
/usr/bin/mc admin user svcacct add minio-elg inesdata --access-key
lw308Hm1qu1nQ3PGy6Uw --secret-key
2ouuTRWWv2AqPafyu6bSMU1ABIwHRx9dabVKGoTN;
/usr/bin/mc mb minio-elg/bucket-connector-elg;
```

El access key utilizado para el minio del conector de ELG se ha generado levantando uno de los minio del local environment en una máquina local, y accediendo a http://localhost:9001/access-keys/new-account, allí se puede generar un nuevo access key que se puede utilizar para el nuevo conector.

2.3.3 Servicio de Autenticación y credenciales Keycloak

El principal archivo de configuración de Keycloak es el resources/keycloak/dataspace-realm.json. Se ha añadido la configuración necesaria para un nuevo conector, modificado el realm

"dataspace", luego se ha exportado la nueva configuración, utilizando esta nueva versión como se explica a continuación.

En http://keycloak:8080/admin/master/console/#/dataspace/roles, se ha creado un realm role llamado "connector-elg", en "Attributes", se ha puestos los siguientes pares "Key: Value": "connector: connector-elg" y "connector-type: inesdata-connector".

En http://keycloak:8080/admin/master/console/#/dataspace/clients, se ha creado un cliente de tipo OpenID Connect con *Client ID* "connector-elg", *Name* connector-elg y *Description* "Inesdata: Cliente para el conector connector-elg", en *Capability Config*, se ha activado la opción "Client authentication" y en *Authentication flow* se ha dejado activadas sólo las opciones "Direct access grants" y "Service accounts roles", en *Login settings* se ha dejado los valores vacíos. Una vez creado el cliente, pulsando en este, en la pestaña de *Credentials* se ha seleccionado el *Client Authenticator* "Signed jwt". En la pestaña *Client scopes,* pulsando en *Add client scope* seleccionando "inesdata-nbf-claim" y "inesdata-local-dataspace-audience" y luego en el botón de *Add,* se ha seleccionado la opción *Default.*

En http://keycloak:8080/admin/master/console/#/dataspace/groups, se ha creado el grupo "connector-elg", y una vez creado, pulsando en este. En la pestaña de *Role mapping* pulsando en el botón *Assign role,* se han seleccionado los roles "connector-elg" y "connector-user".

Una vez guardados todos los cambios en http://keycloak:8080/admin/master/console/#/dataspace/realm-settings, pulsando en *Action* y *Partial export,* se han activado las opciones *Include groups and roles* y *Include clients* y se ha exportado la nueva configuración del realm de dataspace.

Finalmente en el archivo generado, real-export.json, se ha modificado manualmente el json, para que dentro de la lista del campo "users", contenga la lista de usuarios del archivo dataspace-realm.json original y se ha añadido un nuevo usuario llamado "user-elg", miembro del grupo "connector-elg". Los elementos añadidos son los siguientes:

```
{
      "username": "user-c1",
      "createdTimestamp": 1710521195533,
      "email": "user-c1@inesdata.com",
      "firstName": "user-c1",
      "lastName": "user-c1",
      "attributes": {
        "region": "eu"
      },
      "emailVerified": true,
      "enabled": true,
      "credentials": [
          {
              "type": "password",
              "value": "user-c1"
          }
      ],
      "groups": [
          "connector-c1"
      ]
    },
    {
      "username": "user-c2",
      "createdTimestamp": 1710521195633,
```

```
    "email": "user-c2@inesdata.com",
    "firstName": "user-c2",
    "lastName": "user-c2",
    "attributes": {
      "region": "eu"
    },
    "emailVerified": true,
    "enabled": true,
    "credentials": [
        {
            "type": "password",
            "value": "user-c2"
        }
    ],
    "groups": [
        "connector-c2"
    ]
  },
  {
    "username": "user-elg",
    "createdTimestamp": 1710521195633,
    "email": "user-elg@inesdata.com",
    "firstName": "user-elg",
    "lastName": "user-elg",
    "attributes": {
      "region": "eu"
    },
    "emailVerified": true,
    "enabled": true,
    "credentials": [
        {
            "type": "password",
            "value": "user-elg"
        }
    ],
    "groups": [
        "connector-elg"
    ]
  },
  {
    "username": "user-strapi-c1",
    "createdTimestamp": 1710521195633,
    "email": "user-strapi-c1@inesdata.com",
    "firstName": "user-strapi-c1",
    "lastName": "user-strapi-c1",
    "attributes": {
      "region": "eu"
```

```
    },
    "emailVerified": true,
    "enabled": true,
    "credentials": [
        {
            "type": "password",
            "value": "user-strapi-c1"
        }
    ],
    "groups": [
        "connector-c1"
    ]
  }
```

Por otra parte, dentro de la lista “clients”, para el objeto con *clientId* “connector-elg”, en el campo “attributes” -> “jwt.credential.certificate”, se ha usado el siguiente *string*:

```
MIIDETCCAfkCFBEOuzF+mRgrzrxVv70zNDUEv+nUMA0GCSqGSIb3DQEBCwUAMEUxCzAJBg
NVBAYTAkFVMRMwEQYDVQQIDApTb21lLVN0YXRlMSEwHwYDVQQKDBhJbnRlcm5ldCBXaWRn
aXRzIFB0eSBMdGQwHhcNMjQwMTMxMTIyNTUyWhcNMjUwMTMwMTIyNTUyWjBFMQswCQYDVQ
QGEwJBVTETMBEGA1UECAwKU29tZS1TdGF0ZTEhMB8GA1UECgwYSW50ZXJuZXQgV2lkZ2l0
cyBQdHkgTHRkMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAxHbsW755sB/5Yw
jPyk0xPJwyQkxkGV2SS4sQQCCz8KV/QoEilk/PzUqUsE2e7LqavWL5+FPSic79ZKEljeeN
B1z5xoE2YnKEs55MLi/PgfLJHJ9MbA42lJBYI37MeGv0kGkIQDIrtpDjBfSgGSINXeTjs0
T/l5sXXPTfSRm4URWX3I+QO9ACYLb7Cz//G8HdYHp8MHNa38x4BeBku7cT7xgEUPfhu4LC
nBW1u0pMbeT8A2M9zz/TIOJKwUeupacXjJ/tqHGLYMgBLCSRNK4zFJiGzhDbEZA6+NMUp6
60t7Su6M5ix3WIffKlIcxi0bnk9slhjK5NynJ3H/uvbfYnCwIDAQABMA0GCSqGSIb3DQEB
CwUAA4IBAQBAttZGxh1k87S8OHA6L8fmu/PEtjdkJV3Cu7J3xTGfmSBVMTStl2NI4D4iFJ
UFdTJjvEibEaQEETz2zJakwdp7JGG7ip7e/dKsP0wVMC63MZ8eeDzST7IzBLk6mjb+Co0+
2p/9rHPBz1AFpHRcJ9jaORh8FlctfcxcBsr6dMptGtCZMzam26WwePxapNd+wBNFHuAXHy
TzHl3NIq2Bd+56Roen5muZKv4xqI7iVBbVrejvVyXNDhrqh9QBVushjclGdc2GSdCVpUhi
v1srX59/yX5HBZG8HRQSgCXpcSiGIhiMmd1gaU/qrXa8kwmd6tdKUcAiLU7qXEJX4iBzio
bm
```

El real-export.json modificado se usa como nuevo dataspace-realm.json.

2.3.4 Gestor de secretos Vault

En la carperta resources/vault-init/json-secrets se han creado los ficheros private-key-elg.json y public-key-elg.json, el contenido de estos ficheros es el mismo que por ejemplo private-key-c2.json y public-key-c2.json respectivamente.

En la carpeta resources/vault-init/secrets se ha creado el fichero connector-elg.properties, su contenido es el siguiente:

```
access-key-elg=lw308Hm1qu1nQ3PGy6Uw
secret-key-elg=2ouuTRWWv2AqPafyu6bSMU1ABIwHRx9dabVKGoTN
```

Estas llaves serán usadas por el conector para acceder al minio y son las que fueron generadas en la sección Repositorio de almacenamiento MinIO.

2.3.5 Conector de ELG en EDL INESData

Se ha creado un nuevo servicio en el archivo docker-compose con el siguiente contenido:

```
connector-elg:
  image: ghcr.io/oeg-upm/inesdata-connector:0.8.0
  ports:
    - '39191:39191'
    - '39192:39192'
    - '39193:39193'
    - '39194:39194'
    - '39195:39195'
    - '39291:39291'
    - '39196:39196'
  volumes:
    - ./resources:/opt/connector/resources
  environment:
    - EDC_KEYSTORE=/opt/connector/resources/certs/store.pfx
    - EDC_KEYSTORE_PASSWORD=inesdata
    - EDC_FS_CONFIG=/opt/connector/resources/configuration/connector-
elg-configuration.properties
  depends_on:
    postgres-common:
      condition: service_healthy
    minio-c2:
      condition: service_healthy
    keycloak:
      condition: service_healthy
    vault:
      condition: service_healthy
    registration-service:
      condition: service_healthy
```

Como se puede ver en el nuevo servicio, aparte de cambiar los puertos del conector con respecto a los conectores ya existentes, se ha creado un nuevo archivo de configuración llamado connector-elg-configuration.properties en la carpeta resources/configuration. El contenido de este archivo es el siguiente:

```
edc.participant.id=connector-elg
edc.dsp.callback.address=http://connector-elg:39194/protocol
web.http.port=39191
web.http.path=/api
web.http.management.port=39193
web.http.management.path=/management
web.http.protocol.port=39194
web.http.protocol.path=/protocol
edc.transfer.proxy.token.signer.privatekey.alias=private-key-elg
edc.transfer.proxy.token.verifier.publickey.alias=public-key-elg
edc.web.rest.cors.enabled=true
edc.web.rest.cors.headers=origin, content-type, accept, authorization,
x-api-key
web.http.public.port=39291
web.http.public.path=/public
```

```
web.http.control.port=39192
web.http.control.path=/control
web.http.catalog.port=39195
web.http.catalog.path=/federatedcatalog
edc.dataplane.api.public.baseurl=http://connector-elg:39291/public
web.http.shared.port=39196
web.http.shared.path=/shared

edc.vault.hashicorp.url=http://vault:8201
edc.vault.hashicorp.token=00000000-0000-0000-0000-000000000000

edc.datasource.default.url=jdbc:postgresql://postgres-
common:5432/connectorelg
edc.datasource.default.user=connectorelg
edc.datasource.default.password=connectorelg
edc.datasource.default.pool.maxIdleConnections=10
edc.datasource.default.pool.maxTotalConnections=10
edc.datasource.default.pool.minIdleConnections=5

edc.aws.access.key=access-key-elg
edc.aws.secret.access.key=secret-key-elg
edc.aws.endpoint.override=http://minio-elg:9000
edc.aws.region=us-east-1
edc.aws.bucket.name=bucket-connector-elg

edc.oauth.token.url=http://keycloak:8080/realms/dataspace/protocol/ope
nid-connect/token
edc.oauth.provider.audience=http://keycloak:8080/realms/dataspace
edc.oauth.endpoint.audience=http://keycloak:8080/realms/dataspace
edc.oauth.provider.jwks.url=http://keycloak:8080/realms/dataspace/prot
ocol/openid-connect/certs
edc.oauth.certificate.alias=public-key-elg
edc.oauth.private.key.alias=private-key-elg
edc.oauth.client.id=connector-elg
edc.oauth.validation.nbf.leeway=10

edc.api.auth.oauth2.allowedRoles.1.role=connector-admin
edc.api.auth.oauth2.allowedRoles.2.role=connector-management
edc.api.auth.oauth2.allowedRoles.3.role=connector-user

edc.catalog.configuration.participant.1.name = connector-c1
edc.catalog.configuration.participant.1.id = connector-c1
edc.catalog.configuration.participant.1.targetUrl = http://connector-
c1:19194/protocol
edc.catalog.configuration.participant.2.name = connector-c2
edc.catalog.configuration.participant.2.id = connector-c2
```

```
edc.catalog.configuration.participant.2.targetUrl = http://connector-c2:29194/protocol
edc.catalog.configuration.participant.3.name = connector-elg
edc.catalog.configuration.participant.3.id = connector-elg
edc.catalog.configuration.participant.3.targetUrl = http://connector-elg:39194/protocol
edc.catalog.registration.service.host = http://registration-service:59191/api
edc.catalog.cache.execution.period.seconds=15
edc.catalog.cache.partition.num.crawlers=2
edc.catalog.cache.execution.delay.seconds=5
edc.participants.cache.execution.period.seconds=1800
edc.vocabularies.task.execution.period.seconds=1800
edc.data.plane.self.unregistration=true
```

Además se han actualizado los archivos de configuración connector-c1-configuration.properties y connector-c2-configuration.properties añadiendo lo siguiente:

```
edc.catalog.configuration.participant.3.name = connector-elg
edc.catalog.configuration.participant.3.id = connector-elg
edc.catalog.configuration.participant.3.targetUrl = http://connector-elg:39194/protocol
edc.catalog.registration.service.host = http://registration-service:59191/api
```

3 Conclusiones

Se ha presentado el mecanismo de integración del espacio de datos lingüísticos con la plataforma ELG. La integración ha consistido en la creación de un nuevo conector dentro del EDL INESData, donde un servicio de carga de metadatos crea activos en el catálogo del conector de ELG en el EDL con los metadatos de los recursos lingüísticos procedentes de ELG. A su vez este conector se apoya en otro servicio intermediario para la descarga de recursos en ELG. Por último, se han descrito los pasos llevados a cabo para la creación del nuevo conector, con el objetivo de que esta información pueda ser de ayuda de cara a una posible nueva creación de otros conectores en el espacio de datos lingüístico.