4. DSAM - Linee guida, strumenti e risorse software per l’estensione della base informativa (SIN2_DSAM_D7) - 1.1

Premessa

Scopo del presente documento è descrivere le logiche di gestione dell’acquisizione dati da molteplici fonti informative geografiche. Il documento è diviso in due parti:

- una presentazione della logica complessiva di interoperabilità, che include anche una descrizione del modello dati e dell’ontologia Intense
- la descrizione dell’architettura dei servizi di acquisizione dei dati.

Modello dati e Interoperabilità Semantica

Nella visione del progetto RAS CICLO/INTENSE, l’integrazione con le fonti dati esterne deve avvenire nel rispetto del principio dell’*Interoperabilità Semantica*, in base a cui è possibile per sistemi diversi interagire attraverso una visione condivisa del modello dati e dei servizi.

Varie sono le strategie per poter attuare questo paradigma: in particolare quello utilizzato nel web (nonché nell’IoT) da schema.org si basa sulla definizione di un data model di riferimento (Core Information Model) da usare per descrivere i dati (sia a livello di modello che di istanza) dei sistemi interoperanti. Per permettere l’integrazione quindi è necessario adattarsi all’uso di questo modello.

Questo è l’approccio che abbiamo proposto per gestire l’integrazione di fonti dati esterne con il SIN2. Il primo passo quindi è stato quello di formalizzare il modello dati a cui i servizi di popolamenti da fonti esterne devono adeguarsi per poter far importare i loro dati nel sistema.

Per il progetto INTENSE è stato definito un Data Model (cfr. paragrafo seguente) assai ricco e strutturato che è necessariamente il riferimento di partenza della presente implementazione. È un modello che può essere anche formalizzato attraverso un’ontologia (cfr. punto sotto), quindi perfettamente aderente ai principi del Semantic Web.

Nel progetto è stato proposto un modello dati basato sul formato GeoJSON, vista la natura prettamente geografica delle informazioni gestite, opportunamente adattato per rappresentare le informazioni del Modello INTENSE. Esso si basa su:

- Schede Intense
- OST, a loro volta di 4 tipi
  - Sentiero_Percorso
  - Attrattore
  - Servizio
  - Waypoint/Punti sosta

Abbiamo poi aggiunto il concetto di “Tappa”, di fatto un raccoglitore di OST nella Scheda Intense.

Data Model Intense (aggiornato)

Premessa

La prima versione del data model Intense era stato descritta nel Documento di analisi dei dati e delle fonti per la creazione degli OST v1.0, redatto nella prima fase di progetto. Presentiamo in questo paragrafo la versione aggiornata.

Il modello dati

Per l'integrazione con le fonti esterne, secondo il principio dell'*Interoperabilità Semantica*, è stato definito un Modello Dati Intense, un *core vocabulary* a cui le informazioni da importare devono essere adeguate.

Si tratta di un'implementazione del Modello Intense ufficiale del progetto, accessibile alle seguenti URL:

- Modello dati: https://gisepi.crs4.it/intense/intense_documents.zip
- Tassonomie: https://gisepi.crs4.it/intense/

I dati vengono serializzati in GeoJSON, un formato estendibile basato su JSON, che valorizza le informazioni geografiche. Di seguito inseriamo lo schema aggiornato del modello dati.

```
[
    {
        "type": "FeatureCollection",
        "features": [
            {
                "type": "Feature",
                "properties": {
                    "fonteDati": [
                        "< STRINGA DI DESCRIZIONE DELLA FONTE >"
                    ],
                    "class": "Scheda",
                    "title": "< TITOLO SCHEDA INTENSE >",
                    "body": "< BODY >",
                    "nid": < ID UNIVOCO >,
                    "tipoFondo": [...],
                    "tipologia": [...],
                    "tematica": [...],
                    "ambito": [...],
                    "fruizione": [...],
                    "pavimentazione": [...],
                    "presenzaSegnaletica": [...],
                    "gradoDifficolta": [...],
                    "immagini": [{
                        "name": "< NOME FILE >",
                        "url": "...",
                        "id": < ID >,
                        "rawfile": < BASE 64 ENCODE >
                    }],
                    "immaginePrincipale": {
                        "name": "< NOME FILE >",
                        "url": "...",
                        "id": < ID >,
                        "rawfile": < BASE 64 ENCODE >
                    },
                    "allegati": [
                        {
                            "name": "< NOME FILE >",
                            "url": "...",
                            "id": < ID >,
                            "rawfile": < BASE 64 ENCODE >
                            "__typename": "FileType"
```

```
                },

            ],
            "immagini": [
                {
                    "name": "< NOME IMMAGINE >"
                    "url": "...",
                    "id": < ID >,
                    "rawfile": < BASE 64 ENCODE >
                }
            ],
            "mezziTrasporto": [...],
            "periodoFruizione": [{
                "festivita": <nome festivita>,
                "dataEsatta": <data>,
                "dataEsattaDa": <data con anno>,
                "dataEsattaA": <data con anno>,
                "periodoFruizioneDa": <data senza anno>,
                "periodoFruizioneA": <data senza anno>
              }],
            "stato": "< draft | validazione | pubblicato >",
            "locale": "< it | en | fr >"
        }
    },
    {
        "type": "Feature",
        "properties": {
            "class": "Tappa",
            "title": "< TITOLO >",
            "tappaId": < ID TAPPA >,
            "locale": "< it | en | fr >"
        }
    },
    {
        "type": "Feature",
        "properties": {
            "fonteDati": [
                "< STRINGA DI DESCRIZIONE DELLA FONTE >"
            ],
            "nid": < ID UNIVOCO >,
            "class": "Ost",
            "subclass": "ost_sentiero_percorso",
            "title": "< TITOLO OST >",
            "descrizione": "< BODY >",
            "tappaId": < ID TAPPA DI RIFERIMENTO >,
            "disponibilita": < 1 | 0 >,
            "tipologia": [...],
            "tematica": [...],
            "presenzaSegnaletica": [...],
            "statoFondo": [...],
```

```
                    "gradoDifficolta": [...],
                    "accessibilitaDisabili": [...],
                    "rete": [...],
                    "ambito": [...],
                    "fruizione": [...],
                    "esposizioneARischi": [...],
                    "percorrenza": "< Monodirezionale | Bidirezionale |
Non determinata >",
                    "idCastasto": "< ID CATASTO >",
                    "idExt": "< ID >",
                    "allegati": [{
                        "name": "< NOME FILE >",
                        "url": "...",
                        "id": < ID >,
                        "rawfile": < BASE 64 ENCODE >
                        "__typename": "FileType"
                    }],
                    "immagini": [{
                        "name": "< NOME FILE >",
                        "url": "...",
                        "id": < ID >,
                        "rawfile": < BASE 64 ENCODE >
                    }],
                    "immaginePrincipale":  {
                      "name": "< NOME FILE >",
                      "url": "...",
                      "id": < ID >,
                      "rawfile": < BASE 64 ENCODE >
                    },
                    "statoPavimentazione": [
                      {
                        "percentuale": <0...1>,
                        "statoFondoTerm": <termine>
                      }
                    ],
                    "tipologiaDelFondo": [
                      {
                        "percentuale": <0...1>,
                        "tipologiaFondo": <termine>
                      }
                    ]
                    "locale": "< it | en | fr >"
                }
            },
            {
                "type": "Feature",
                "properties": {
                    "field_fonte_dati": [
                        "< STRINGA DI DESCRIZIONE DELLA FONTE >"
                    ],
```

```
            "nid": < ID UNIVOCO >,
            "class": "Ost",
            "subclass": "ost_waypoint",
            "title": "< TITOLO OST >",
            "descrizione": "< BODY >",
            "tappaId": < ID TAPPA DI RIFERIMENTO >,
            "disponibilita": < 1 | 0 >,
            "tipologia": [...],
            "idCastasto": "< ID CATASTO >",
            "idExt": "< ID >",
            "visitabile": < 1 | 0 | null >,
            "allegati": [{
                "name": "< NOME FILE >",
                "url": "...",
                "id": < ID >,
                "rawfile": < BASE 64 ENCODE >
                "__typename": "FileType"
            }],
            "immagini": [{
                "name": "< NOME FILE >",
                "url": "...",
                "id": < ID >,
                "rawfile": < BASE 64 ENCODE >
            }],
            "immaginePrincipale":  {
              "name": "< NOME FILE >",
              "url": "...",
              "id": < ID >,
              "rawfile": < BASE 64 ENCODE >
            },
            "locale": "< it | en | fr >"
        }
    },
    {
        "type": "Feature",
        "properties": {
            "fonteDati": [
                "< STRINGA DI DESCRIZIONE DELLA FONTE >"
            ],
            "nid": < ID UNIVOCO >,
            "class": "Ost",
            "subclass": "ost_attrattore",
            "title": "< TITOLO OST >",
            "descrizione": "< BODY >",
            "tappaId": < ID TAPPA DI RIFERIMENTO >,
            "disponibilita": < 1 | 0 >,
            "tipologia": [...],
            "tematica": [...],
            "contatti": [...],
            "riconoscimenti": [...],
```

```
            "accessibilitaDisabili": [...],
            "idCastasto": "< ID CATASTO >",
            "idExt": "< ID >",
            "visitabile": < 1 | 0 | null >,
            "allegati": [{
                "name": "< NOME FILE >",
                "url": "...",
                "id": < ID >,
                "rawfile": < BASE 64 ENCODE >
                "__typename": "FileType"
            }],
            "immagini": [{
                "name": "< NOME FILE >",
                "url": "...",
                "id": < ID >,
                "rawfile": < BASE 64 ENCODE >
            }],
            "immaginePrincipale":  {
              "name": "< NOME FILE >",
              "url": "...",
              "id": < ID >,
              "rawfile": < BASE 64 ENCODE >
            },
            "locale": "< it | en | fr >"
        }
    },
    {
        "type": "Feature",
        "properties": {
            "fonteDati": [
                "< STRINGA DI DESCRIZIONE DELLA FONTE >"
            ],
            "nid": < ID UNIVOCO >,
            "class": "Ost",
            "subclass": "ost_servizi",
            "title": "< TITOLO OST >",
            "descrizione": "< BODY >",
            "tappaId": < ID TAPPA DI RIFERIMENTO >,
            "disponibilita": < 1 | 0 >,
            "tipologia": [...],
            "tematica": [...],
            "contatti": [...],
            "riconoscimenti": [...],
            "accessibilitaDisabili": [...],
            "idCastasto": "< ID CATASTO >",
            "idExt": "< ID >",
            "visitabile": < 1 | 0 | null >,
            "allegati": [{
                "name": "< NOME FILE >",
                "url": "...",
```

```
                    "id": < ID >,
                    "rawfile": < BASE 64 ENCODE >
                    "__typename": "FileType"
                }],
                "immagini": [{
                    "name": "< NOME FILE >",
                    "url": "...",
                    "id": < ID >,
                    "rawfile": < BASE 64 ENCODE >
                }],
                "immaginePrincipale":  {
                  "name": "< NOME FILE >",
                  "url": "...",
                  "id": < ID >,
                  "rawfile": < BASE 64 ENCODE >
                },
                "locale": "< it | en | fr >"
            }
        },
    ],

},
{ //INFORMAZIONI GEOGRAFICHE IN FORMATO GEOJSON
    "type": "FeatureCollection",
    "features": [
        { // ES. OST "POI"
            "type": "Feature",
            "properties": {
                "ostid": < ID >,
                "tappaId": < ID TAPPA >,
            },
            "geometry": {
                "type": "Point",
                "coordinates": [
                    lon,
                    lat
                ]
            }
        },
        { // ES. OST "TRATTO"
            "type": "Feature",
            "properties": {
                "ostid": < ID >,
                "tappaId": < ID TAPPA >,
                "parziale": < false | true >,
            },
            "geometry": {
                "type": "LineString",
                "coordinates": [
                    [
```

```
                        lon,
                        lat
                    ],
                    [...],
                ]
            }
        },
    ]
```

```
    }
]
```

Per tutti gli attributi non esplicitamente specificati in questo file JSON di esempio, bisogna fare riferimento ai valori corrispondenti delle Tassonomie Intense.

Basta specificarli come stringhe esatte per essere interpretate correttamente dal servizio di importazione dati del SIN2. Per semplicità il servizio di acquisizione fa un confronto case insensitive sulle stringhe delle tassonomie specificate sul sito https://gisepi.crs4.it/intense/.

Ad esempio, per specificare il tipo di fondo di un OST di tipo sentiero percorso, andrò a specificare qualcosa simile al seguente frammento (cfr. https://gisepi.crs4.it/intense/intense_it/index.php?tema=4905&/pavimentazione-di-un-segmento-di-percorso):

```
"tipologiaDelFondo": [
    {“percentuale”:0.5,“tipologiaFondo”:“FONDO IN ASFALTO”},
    {“percentuale”:0.1,“tipologiaFondo”:“FONDO IN TERRA BATTUTA”},
    {“percentuale”:0.3,“tipologiaFondo”:“FONDO NATURALE”},
    {“percentuale”:0.1,“tipologiaFondo”:“DATO SUL FONDO NON PRESENTE”},
]
```

Si noti come per gli OST di tipo Sentiero_Percorso gli attributi:

- Grado di protezione percorso ciclabile
- Grado di difficoltà Intense
- Tempo di percorrenza

sono calcolati, come specificato nel documento del Modello Dati Intense. A questo riguardo, qualora venga specificato nel file GeoJSON un tempo di percorrenza, esso verrebbe comunque sovrascritto da quello calcolato (per cui è inutile indicarlo).

Ontologia del Data Model Intense

Il Data Model può essere descritto anche in forma semantica attraverso un’Ontologia, costruita secondo i principi qui descritti.

Ontologie principali Importate

- purl.org/geojson/vocab# geojson
- www.w3.org/2006/time#2016 time
- www.w3.org/TR/skos-reference/skos-owl1-dl.rdf skos
- schema.org schema.org
- purl.org/dc Dublin Core (terms)

Descrizione Ontologia

Sono stati definiti i concetti di SchedaIntense, Tappa e OST. Sono state definite le proprietà hasTappa, che ha per dominio SchedaIntense e per co-dominio Tappa e la proprietà hasOST, che ha per dominio Tappa e per co-dominio OST.

La classe OST è stata definita come sottoclasse della classe Feature di GeoJSON, in quanto ogni OST rappresenta una feature geografica. La classe OST ha quattro sottoclassi (OSTAttrattore, OSTSentieroPercorso, OSTServizi, OSTWaypoint) che rappresentano, ciascuna, uno specifico tipo di OST. OSTAttrattore è stato definito come equivalente all’unione delle classi LandmarksOrHistoricalBuidings, TouristAttraction e TouristDestination di Schema.org.

Ogni OST ha una geometria rappresentata dalla proprietà geometry che ha come dominio Feature e come co-dominio Point e MultiLineString (classi di GeoJSON). Le istanze di queste due ultime classi sono collegate dalle proprietà hasCoordinate e hasCoordinates alla classe Coordinate, rispettivamente. La classe Coordinate ha due dataproperty xCoordinate e yCoordinate, di tipo float, che rappresentano i valori possibili che le coordinate possono assumere. Sono stati definiti i seguenti vincoli logici:

1. La classe Point di GeoJSON è specializzata come l’insieme di punti di OSTServizi, OSTAttrattore e OSTWaypoint
2. Ogni istanza della classe Point così ridefinita ha esattamente UN’ISTANZA della classe Coordinate

3. La classe MultiLineString di GeoJSON è specializzata come OSTPercorso
4. Ogni istanza della classe MultiLineString così ridefinita ha necessariamente almeno DUE ISTANZE della classe Coordinate

Ciascuna tassonomia è stata rappresentata come una classe, i valori che la tassonomia definisce sono stati rappresentati come individui della classe corrispondente. Le classi rappresentanti le tassonomie sono state definite come sottoclasse di skos:Concept.

Le SchedeIntense e gli OST sono legati alle tassonomie dalle corrispondenti objectproperty (ad esempio l'objectproperty AccessibilitaDisabili ha per dominio OSTSentieroPercorso e SchedaIntense e per codominio la classe Accessibilita).

I vari attributi che una scheda o un OST possono avere sono stati definiti come dataproperty.

Le proprietà che rappresentano date sono state rappresentate come data property con co-dominio il datatype xsd:dateTime introdotto dall'ontologia importata OWLTime (orariDataApertura…).

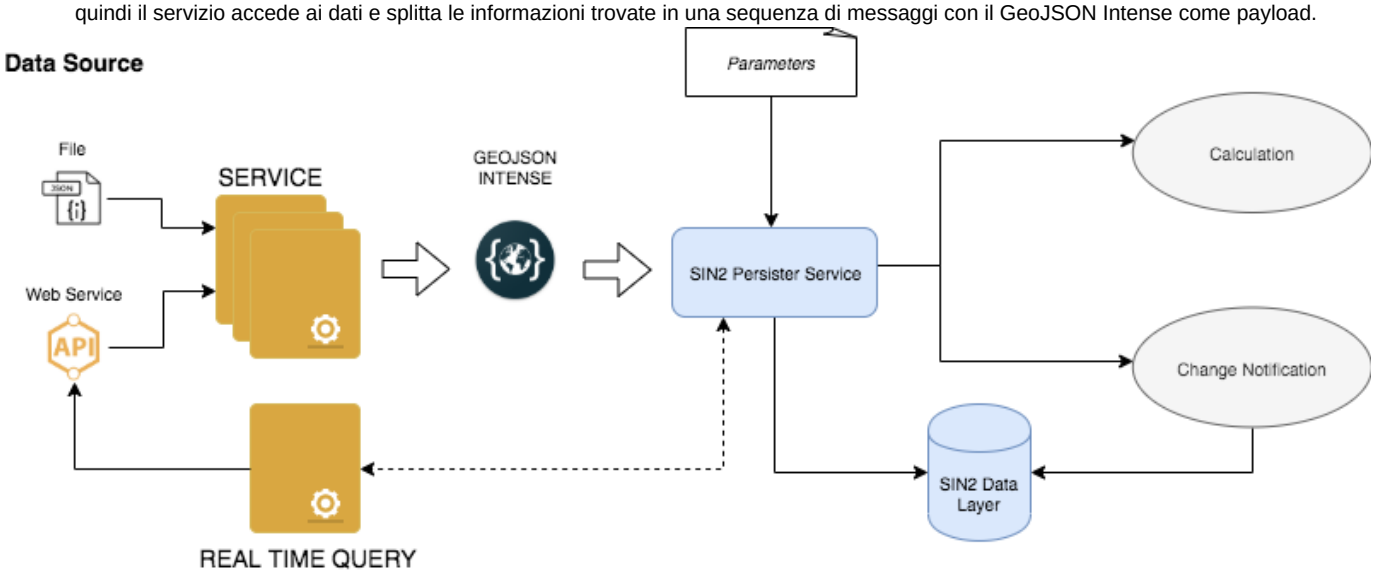
L'ontologia può essere scaricata da questo indirizzo (allegato ad una pagina Confluence). Come *base url* è stata scelta

```
https://api.turismoattivo.sardegnaturismo.it/ontologies/sin2
```

Architettura della piattaforma di acquisizione dati per la base informativa

L'implementazione della piattaforma è basata su Apache Camel. La logica è quella di avere un servizio dedicato per ciascuna fonte dati da integrare che:

- recupera i dati. Assumiamo qui 2 casi
  - lettura da file system in una directory dedicata alla specifica fonte dati
  - lettura da web service, es. i servizi di Sardegna Turismo.
- estrae le informazioni e le adatta al data model GeoJSON Intense. In ottica di processing a "pipeline", al servizio Drupal (SIN2 Persister Service) viene mandato un solo oggetto per volta, quindi un OST o una Scheda Intense i cui OST però sono stati inviati prima. Di fatto quindi il servizio accede ai dati e splitta le informazioni trovate in una sequenza di messaggi con il GeoJSON Intense come payload.



Logica parametrica per aggiornamento OST

Questi sono i vincoli che sono stati posti a questa implementazione:

- l'oggetto principale dell'importazione dalle fonti esterne è l'OST. Per comodità di progetto assumiamo anche di poter importare Schede Intense già costruite ma la massima attenzione a livello di modello deve essere dedicata agli OST
- è necessario assumere l'ipotesi che i dati un OST possano provenire da più fonti diverse. Diventa necessario quindi pensare ad una riconciliazione tra gli attributi ricevuti, in cui l'OST importato è l'unione degli attributi prelevati da più fonti.

Queste invece le assunzioni che facciamo:

1. le operazioni sul SIN2 sono solo di tipo "Insert" e "Update" dove l'update viene automaticamente dedotto dal fatto che si tratta di un OST già presente nel SIN2 (*Upsert*).
2. definiamo le regole con cui, a fronte di un insieme di OST e di una richiesta di operazione, il sistema definisce la riconciliazione tra fonti

dati diverse

3. assumiamo che la logica sia sempre di tipo push, ovvero dalla fonte esterna (o meglio, da un servizio che recupera gli OST dalla fonte esterna) al servizio di importazione nel SIN2.
4. IDENTIFICAZIONE UNIVOCA OST, basata su
    a. ID
    b. attributi geografici (punti e tracciato).

La logica parametrica di gestione degli aggiornamenti da più fonti (NOTA: solo per gli OST) è stata implementata facendo queste assunzioni:

- Identificazione forte degli OST in base ad un ID esterno o ad attributi geografici. L'ID deve essere lo stesso usato da più sorgenti dati (cfr. https://sardegnaturismo.atlassian.net/wiki/spaces/IN/pages/731742233/3.+Progettazione+software+di+dettaglio+della+componente+DSAM+SIN2+DSAM+D6#Processo-di-importazione)
- Identificazione univoca della fonte: facile da gestire in quanto la codifica sarà gestita dal servizio corrispondente su Camel.

L'implementazione si basa sulla definizione di una gerarchia di logiche di sostituzione per ciascun campo di ciascun OST.

Es. per OST

- Titolo: NULL => l'ultimo che arriva sovrascrive
- Body:
    1. Forestas
    2. OpenStreetMap
- Altitudine:
    1. OpenStreetMap

Questo vorrebbe dire ad esempio che se mi arriva un OST da Forestas ma l'ultimo aggiornamento sull'altitudine è stato fatto da OpenStreetMap non lo vado a sostituire, mentre invece posso sostituire il body.

A tale scopo si mantiene per ogni campo il riferimento della fonte dati che l'ha modificato l'ultima volta: questo è stato fatto aggiungendo un attributo al content type che mantiene in formato JSON la mappa dell'ultima fonte che ha aggiornato l'entità corrente. Esempio:

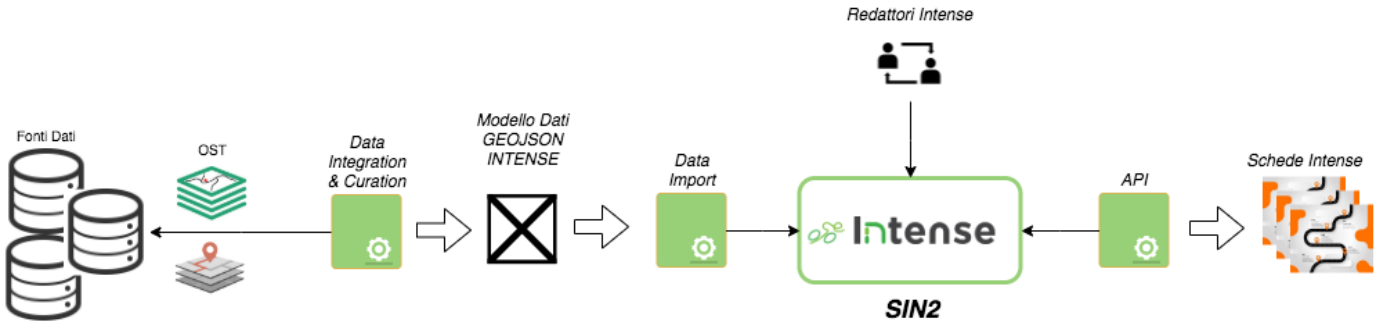
```
{ titolo: openstreetmap,

  body: forestas

  altitudine: openstreetmap,

… }
```

La struttura di configurazione per la logica di interoperabilità semantica viene gestita da back-end Drupal. Di seguito alleghiamo un esempio definito nel SIN2.

```
{
"title":
["sardegnasentieri",
"Frontend editoriale",
"forestas"],
"body":
["sardegnasentieri"],
"field_geojson":
["sardegnasentieri",
"Frontend editoriale",
"forestas"],
"field_geometry":
["Frontend editoriale",
"forestas",
"sardegnasentieri"],
"field_ost_type":
["sardegnasentieri",
"Frontend editoriale",
"forestas"]
}
```


Documentazione sugli adapter sviluppati

La logica di processing dei dati da fonti esterne nel progetto Intense è mostrata nell’immagine che segue.



Lo scopo degli Adapter è quello di gestire la prima fase, ovvero:

- il recupero dati dalle fonti
- l'adattamento al modello dati GeoJSON Intense
- l'invio del GeoJSON al web service Drupal che ne curerà il processing e la persistenza tramite la componente DSAM.

Per quest'ultimo punto, essenzialmente le operazioni svolto sono:

- inserimento di un OST
- modifica di un OST
- "indisponibilità" di un OST

Nel progetto sono stati sviluppati 3 adattatori, per gestire l'acquisizione:

- da un file dati in formato GeoJSON già conforme al modello Intense. Di fatto questo è l'esempio da cui partire per poter sviluppare dei nuovi adapter
- dai POI del sito SardegnaTurismo.it, che recuperiamo da questo endpoint API: `https://www.sardegnaturismo.it/rest/attrattori.json`
- dei dati sui trasporti. Permette di importare i dati sulle fermate come OST di tipo "Servizio" a partire dai file pubblicati sul sito Open Data della Regione Sardegna. Tale dato è presente a questo indirizzo:`http://www.sardegnamobilita.it/opendata/punto_fermata.geojson`

Linee guida per lo sviluppo di un nuovo connettore

Lo sviluppo di un nuovo connettore può procedere in diversi modi.

Il più semplice, è quello di scrivere internamente alla propria applicazione un adapter rendendolo compliant col formato del modello Intense. In questo modo la propria applicazione dovrà soltanto curarsi del mettere un file *.json nella cartella su cui è in ascolto Camel. Sarà possibile come sviluppo futuro che il connettore Intense sia connesso a un bucket S3 al posto di una cartella fisica sulla macchina sulla quale gira.

Il secondo caso richiede sviluppo ed è più complesso.
Come punti di attenzione, prima ancora di illustrare come aggiungere questo connettore, è necessario tenere presente che:

- le code su cui vengono scambiati i messaggi su Camel sono code di tipo SQS presenti su Amazon, è pertanto quindi necessario crearle da management console AWS.
- L'adapter si aspetta un singolo OST alla volta: per questo il messaggio viene prima mandato a uno splitter che eventualmente divide geometrie più complesse

Andando invece a dover creare su Camel un Adapter, è necessario procedere come segue:

- Scaricarsi il sorgente disponibile sul repo git a questo indirizzo git@bitbucket.org:sardegnaturismo/intense-sin2-interoperabilita.git
- Aggiungere nella classe: <repo_path>/src/main/java/org/net7/router/Routes.java una rotta con un timer (ad intervalli regolari o con notazione cron) che specifica ogni quanto recuperare dalla fonte dati (vedi esempio sottostante):

```
public static String SARDEGNATURISMO_TIMER =
"timer://sardegnaturismo?fixedRate=true&period=12000000";
    public static String SARDEGNAMOBILITA_TIMER =
"timer://sardegnamobilita?fixedRate=true&period=12000000";
```

- Creare su AWS una coda SQS da utilizzare come "splitter" del messaggio

- Aggiungere nel file precedente anche questa informazione, come ad esempio:

```
public static String SARDEGNATURISMO_ROUTE =
"aws-sqs://SardegnaTurismoFetcherRoute" + "?concurrentConsumers=" +
numConsumers;
  public static String SARDEGNAMOBILITA_ROUTE =
"aws-sqs://SardegnaMobilitaFetcherRoute" + "?concurrentConsumers=" +
numConsumers;
```

- A questo punto è necessario agire sul file di routing, ossia: <repo_path>/src/main/java/org/net7/router/IntenseRouteBuilder.java

E' necessario anzitutto inizializzare la rotta per il recupero temporizzato delle informazioni (dentro il metodo configure della classe stessa):

```
from(Routes.SARDEGNAMOBILITA_TIMER).process(new
EmptyBodyProcessor()).to(Routes.SARDEGNAMOBILITA_ROUTE);
```

Dopodichè questo messaggio, come detto precedentemente, va diviso in N pacchetti, uno per OST, da mandare all'adapter "generico", utilizzando la coda SQS precedentemente creata.

Per fare ciò va scritto un Processor, che recupera il dato e lo salva in un array di elementi di tipo "`IntenseAdapterMessage`".

Fondamentale è specificare la fonte dati, che serve per l'adapter (è la proprietà pubblica **fonteDati** della classe di cui sopra) , oltre alle informazioni da adattare, che vanno nella proprietà pubblica **inputData** sempre della stessa classe.

Riporto per praticità il codice di un Procressor di esempio (`SardegnaMobilitaFetcherProcessor`):

```
package org.net7.processors;

import com.google.gson.*;
import org.apache.camel.Exchange;
import org.apache.camel.Processor;
import org.apache.http.HttpHeaders;
import org.net7.model.IntenseAdapterMessage;
import org.net7.router.Routes;
import org.net7.utils.HttpUtils;
import org.net7.utils.IntenseDrupalCostants;
import org.net7.utils.LoggingUtils;

import java.util.*;

public class SardegnaMobilitaFetcherProcessor implements Processor {
    private static LoggingUtils loggingUtils =
LoggingUtils.getInstance("SardegnaTurismoFetcherProcessor");

    public void process(Exchange exchange) throws Exception {
        loggingUtils.logInfo("Into sardegna mobilita fetcher!");
        HttpUtils http = new HttpUtils();
        Map params = new HashMap<>();
        String jsonResponse =
http.callGet(Routes.SARDEGNAMOBILITA_JSON_FERMATE_ENDPOINT, "", params);
        List<IntenseAdapterMessage> messages = new ArrayList<>();
        JsonElement jsonData = JsonParser.parseString(jsonResponse);
        JsonArray features =
jsonData.getAsJsonObject().get("features").getAsJsonArray();
        for(int i = 0; i < features.size(); i++) {
            JsonObject feature = features.get(i).getAsJsonObject();
            IntenseAdapterMessage message = new IntenseAdapterMessage();
            message.fonteDati = "sardegnaMobilita";
            message.inputData = feature.toString();
            messages.add(message);
        }
        exchange.getIn().setBody(messages);
    }
}
```

A questo punto ci affidiamo alle funzioni base di Camel per split e invio alla coda SQS dell’adapter.

```
from(Routes.SARDEGNAMOBILITA_ROUTE).process(new
SardegnaMobilitaFetcherProcessor())
                .split(simple("${body}"))
                .marshal()
                .json(JsonLibrary.Gson)
                .to(Routes.ADAPTER_SQS_ROUTE);
```

L’adapter, come da nome, adatta il messaggio al formato Intense.

Pertanto sono necessarie due operazioni:

- la prima, è quella di creare una classe Adapter che estende la `GenericAdapter`(che si può trovare qua: <repo_path>/src/main/java/org/net7/adapters/GenericAdapter.java) e di cui riporto il codice:

```
package org.net7.adapters;

public class GenericAdapter {

    protected String inputData = "";
    protected String outputData = "";
    public GenericAdapter(String inputData){
        this.inputData = inputData;
    }

    /**
     * Method that adapts the input to the desired output
     * this default implementation simply returns the input without
processing it
     * @return
     */
    public void adapt(){
        this.outputData = this.inputData;
    }

    public String getOutputData(){
        return this.outputData;
    }


}
```

Come ad esempio è stato fatto per SardegnaMobilità:

```
package org.net7.adapters;

import com.google.gson.JsonArray;
```

```
import com.google.gson.JsonElement;
import com.google.gson.JsonObject;
import com.google.gson.JsonParser;
import org.net7.utils.LoggingUtils;

public class SardegnaMobilitaAdapter extends GenericAdapter {
    private static LoggingUtils loggingUtils =
LoggingUtils.getInstance("SardegnaMobilitaAdapter");
    public String fonteDati = "SardegnaMobilita";

    public SardegnaMobilitaAdapter(String inputData) {
        super(inputData);
    }

    public void adapt() {
        JsonArray mainArray = new JsonArray();
        JsonObject mainObject = new JsonObject();
        JsonObject mainGeometryObject = new JsonObject();
        mainObject.addProperty("type", "FeatureCollection");
        JsonArray parsedGeometryElements = new JsonArray();
        JsonArray parsedElements = new JsonArray();
        JsonElement jsonData = JsonParser.parseString(this.inputData);
        JsonObject element = jsonData.getAsJsonObject();
        //parse the single element!
        parsedElements.add(this.setOstProperties(element, 0));

parsedGeometryElements.add(this.setOstGeometryProperties(element, 0));
        mainObject.add("features", parsedElements);
        mainGeometryObject.add("features", parsedGeometryElements);
        mainArray.add(mainObject);
        mainArray.add(mainGeometryObject);
        loggingUtils.logInfo(mainArray.toString());
        this.outputData = mainArray.toString();
    }

    private JsonObject setOstProperties(JsonObject element, int i) {
        JsonObject elementProperties =
element.getAsJsonObject("properties");
        JsonObject elementGeometry =
element.getAsJsonObject("geometry");
        JsonObject singleElement = new JsonObject();
        singleElement.addProperty("type", "Feature");
        JsonArray arrayProps = new JsonArray();
        arrayProps.add("SardegnaMobilita");
        JsonObject nestedProperties = new JsonObject();
        nestedProperties.add("fonteDati", arrayProps);
        nestedProperties.addProperty("language", "it");
        nestedProperties.addProperty("class", "Ost");
        nestedProperties.addProperty("title",
elementProperties.get("stop_name").getAsString());
```

```
        nestedProperties.addProperty("subclass", "ost_servizi");
        nestedProperties.addProperty("idExt",
elementProperties.get("stop_code").getAsString());
        nestedProperties.addProperty("temporaryId", i);
        singleElement.add("properties", nestedProperties);
        singleElement.add("geometry", elementGeometry);
        return singleElement;
    }

    private JsonObject setOstGeometryProperties(JsonObject element, int
i) {
        JsonObject singleElement = new JsonObject();
        singleElement.addProperty("type", "Feature");
        JsonObject geometryProperties = new JsonObject();
        geometryProperties.addProperty("temporaryId", i);
        JsonObject elementGeometry =
element.getAsJsonObject("geometry");
        singleElement.add("geometry", elementGeometry);
        singleElement.add("properties", geometryProperties);
        return singleElement;
    }
```

```
}
```

A questo punto dobbiamo estendere lo “switch” dell’AdapterProcessor (<repo_path>/src/main/java/org/net7/processors/AdapterProcessor.java) aggiungendo il nostro “case”.

```
switch(fonteDati){
            case "standard":
                adapter = new GenericAdapter(message.inputData);
                break;
            case "sardegnaMobilita":
                adapter = new
SardegnaMobilitaAdapter(message.inputData);
                break;
            case "sardegnaTurismo":
                adapter = new SardegnaTurismoAdapter(message.inputData);
                break;
        }
```

Il resto della procedura a questo punto non lo dobbiamo rivedere: il dato sarà adattato correttamente dalla nostra classe al formato Intense, e il persisting è gestito allo stesso modo di come vengono salvate le schede lato Frontend Editoriale.