May 11th, 2026

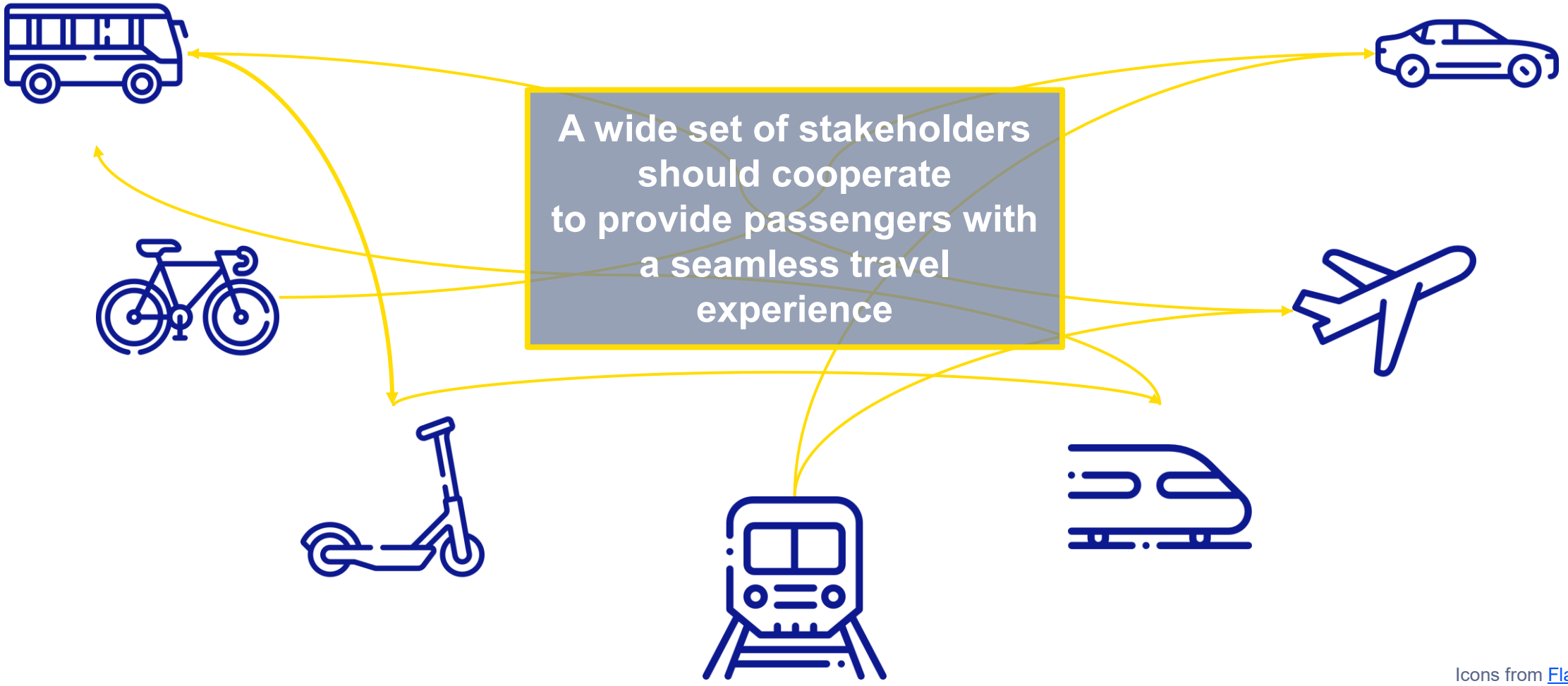
23rd European Semantic Web Conference (ESWC) 2026

Tutorial KG4DI

Knowledge Graphs for Data Interoperability with Chimera (KG4DI)

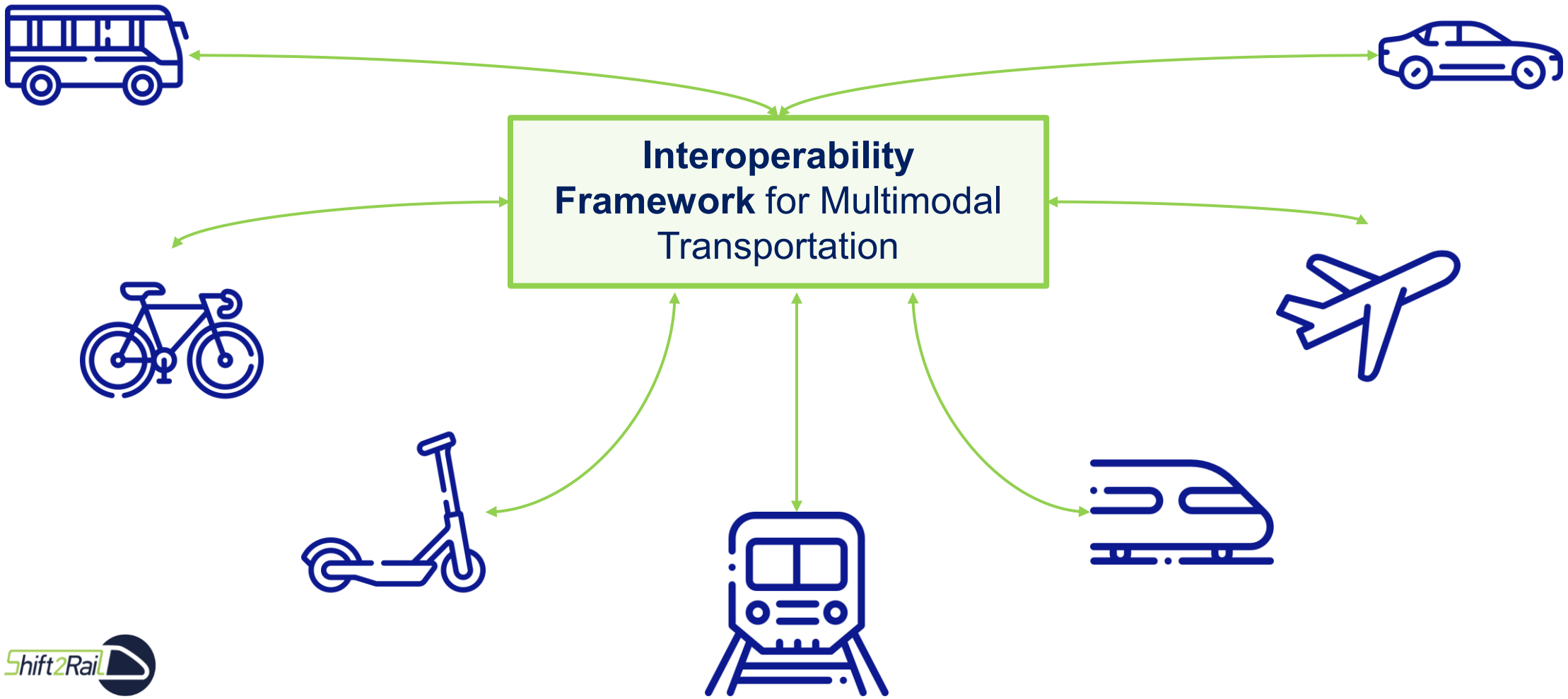
Marco Grassi, Mario Scrocca,
Alessio Carenini, Irene Celino (Cefriel)
name.surname@cefriel.com

A Motivating Scenario



Icons from Flaticon

A Motivating Scenario



https://rail-research.europa.eu/research-development/ip4/

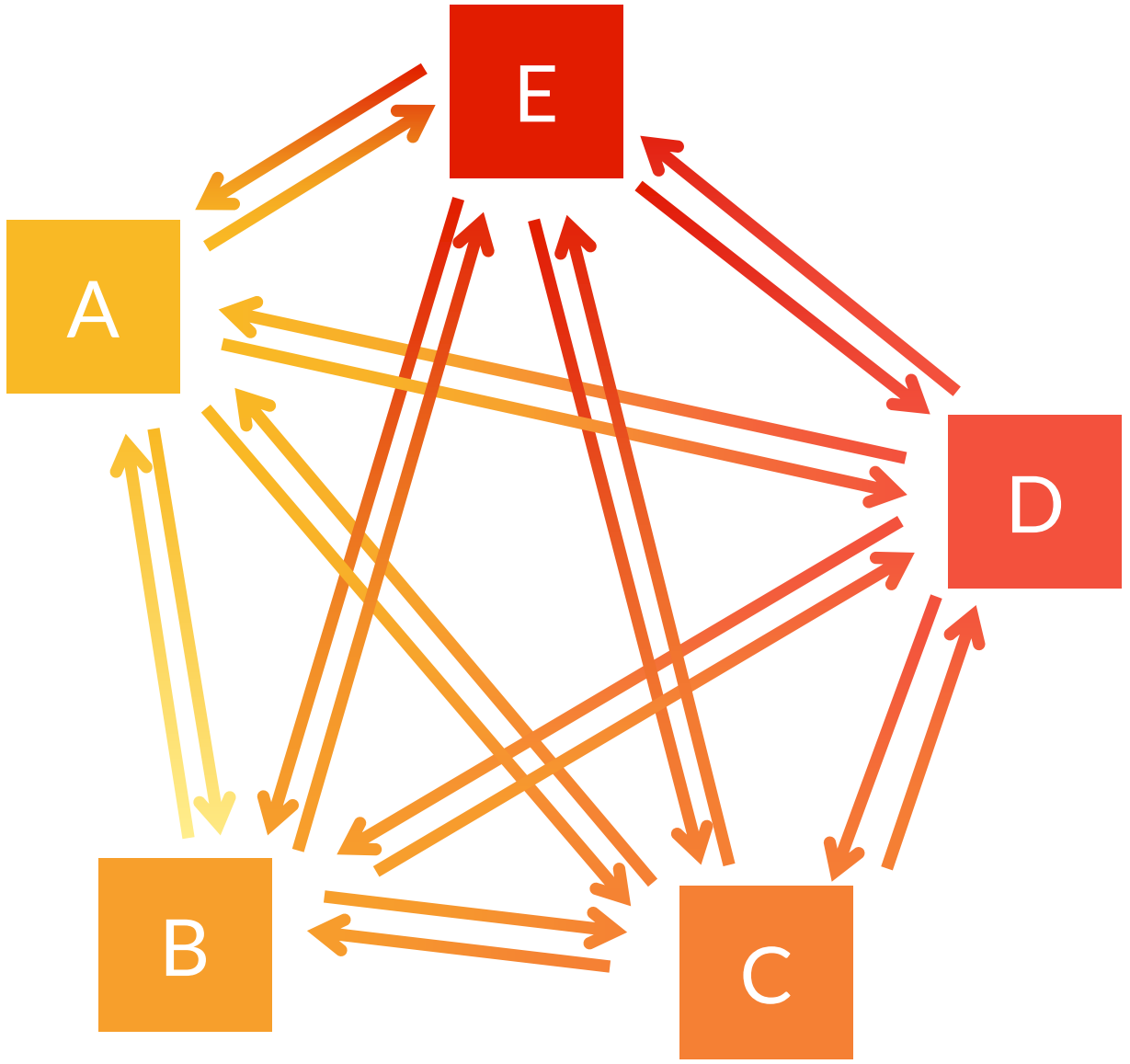
Icons from Flaticon

Why this tutorial?

Two or more systems are **interoperable if they can communicate and exchange information** in an effective way and without loss of meaning.

Considering different systems (also within the same company):

- Many **different data formats**
- **Heterogeneous** specifications and **semantics**
- **Point-to-point** system integration **doesn't scale**



Question 1: **How Knowledge Graphs can support mediated data exchanges?**

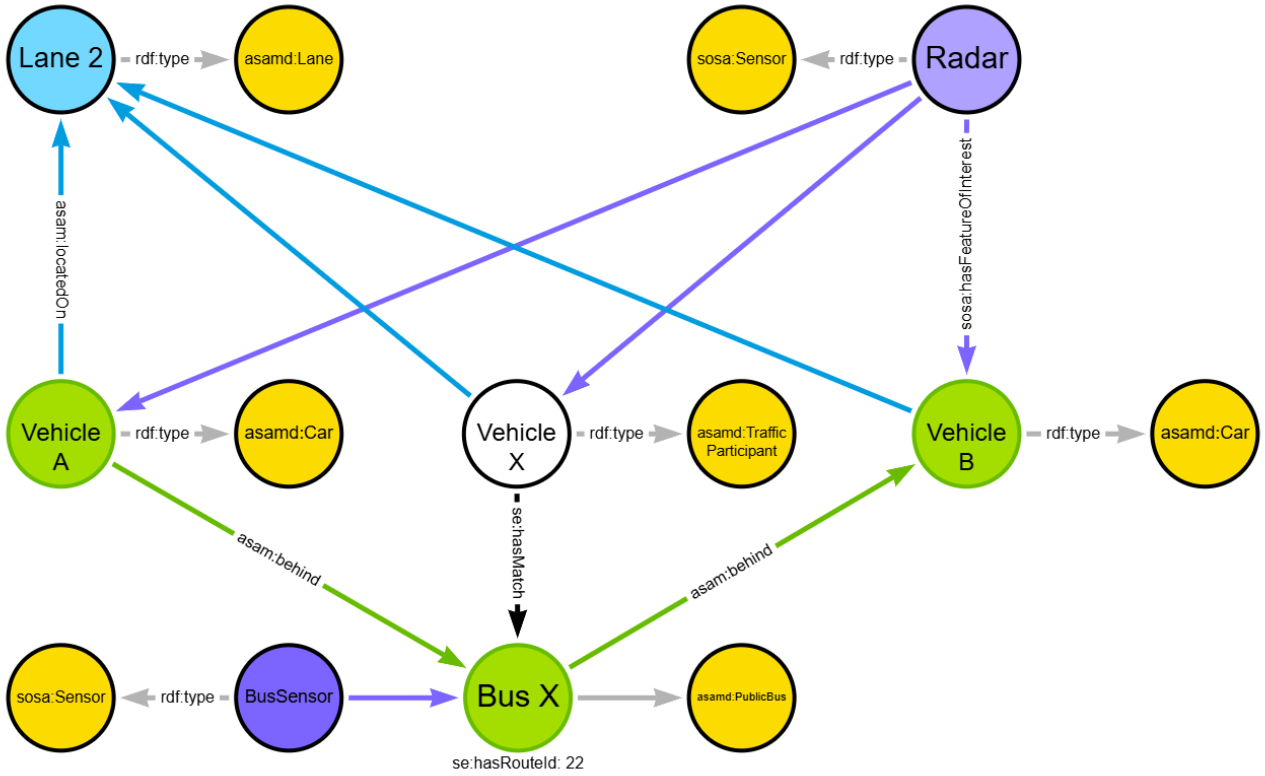
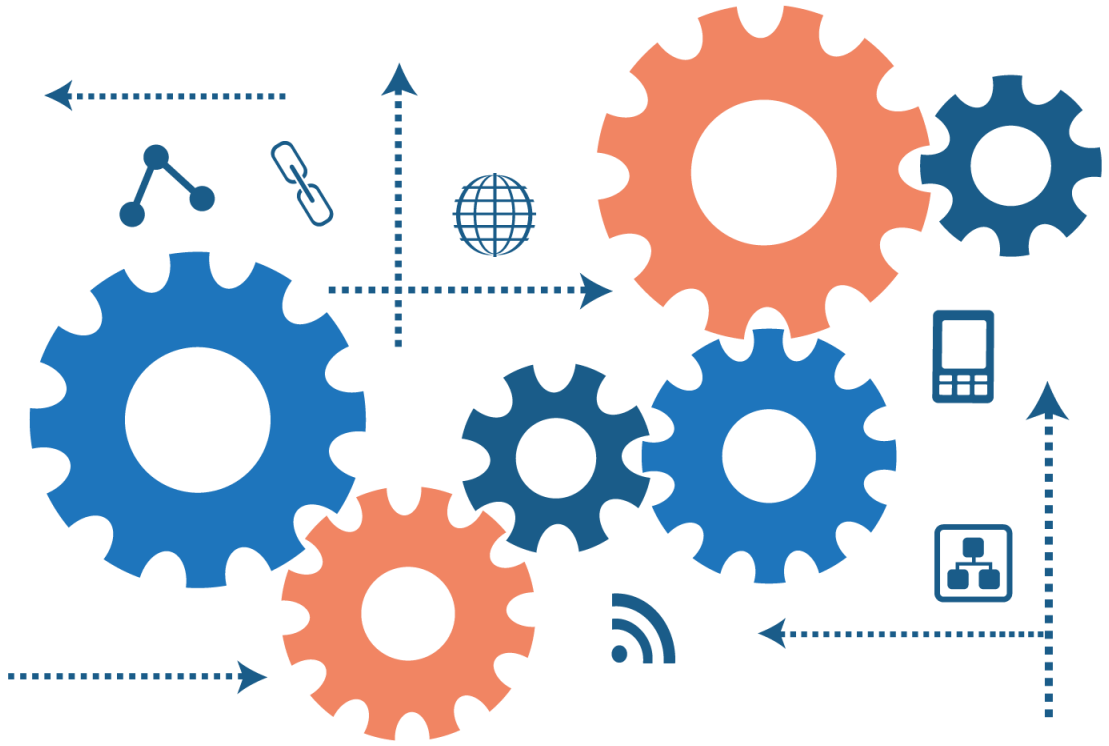


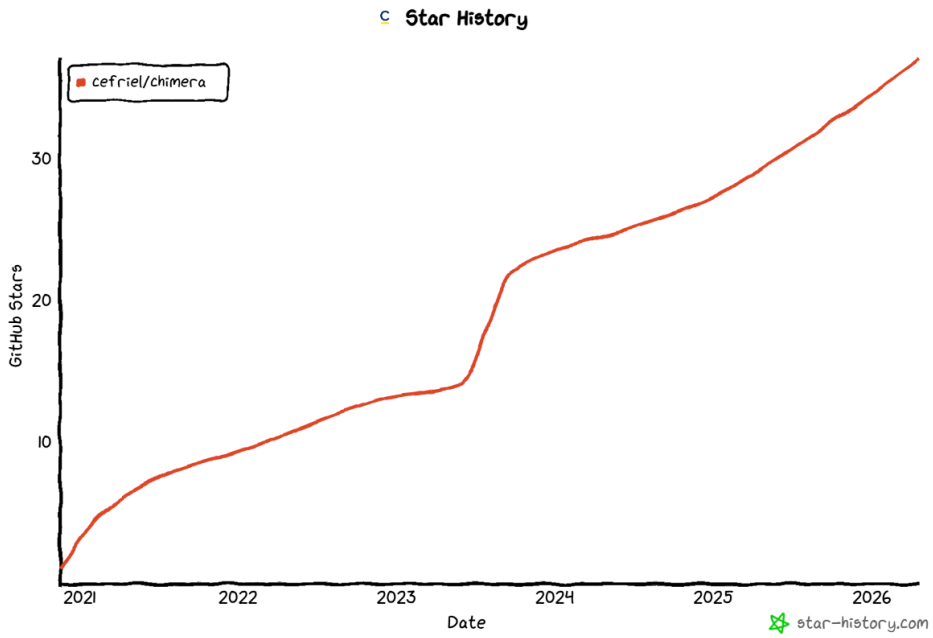
Image from Grassi et al. "*Semantic Low-Code Data Fusion on the Edge to Support Intelligent Traffic Management*" (to appear)

Question 2: **How to support mediated data exchanges between heterogenous information systems?**



Tutorial Objectives

- Learn how **Knowledge Graphs can support data interoperability effectively**
- **Get familiar about Chimera** (we added for this tutorial extensive documentation at https://cefriel.github.io/chimera/)
- **Have fun with our exercise** and experience how cool is to have interoperable data!
- **You choose Chimera for your next data interoperability project!** Help us in spreading the word about this tool and build a community around it



Presenters

MARCO GRASSI
Knowledge Technologies Researcher
Cefriel

MARIO SCROCCA
Senior Knowledge Technologies Researcher
Cefriel

Projects

SMARTY

Urban-Flow

smartedge

Cefriel
POLITECNICO DI MILANO

Discover more at
cefriel.com

100+ PROJECTS PER YEAR

140+ PEOPLE

30 YEARS OF INNOVATION

Cefriel is the partner of Research, Innovation and Education that for over 30 years has accompanied national and international companies in their development path towards digital

10% RESEARCH
20% EDUCATION
70% INNOVATION

Agenda

Segment	Duration	Content
Introduction	09:00 - 09:15	Motivation, objectives and practical information
Part 1 — Data Interoperability Challenges	09:15 - 09:45	Key challenges in heterogeneous data integration; limitations of ad-hoc approaches; advantages of adopting knowledge graphs + [presentation of the tutorial use case]
Part 2 — Mapping Approaches	09:45 - 10:30	State-of-the-art; Implementing an any-to-RDF-to-any pattern; Mapping Template Language (MTL) + [exercises for lifting/lowering mapping rules]
Break	10:30 - 11:00	
Part 3 — Chimera Framework	11:00 - 12:15	Chimera concepts and related components + [guided pipeline construction for the tutorial use case: ingestion, lifting, SPARQL enrichment, construction, lowering, and visualisation]
Part 4 — Chimera in Action	12:15 - 12:30	Practical experiences adopting Chimera to enable data interoperability across various domains

cefriel.github.io/kg4di/

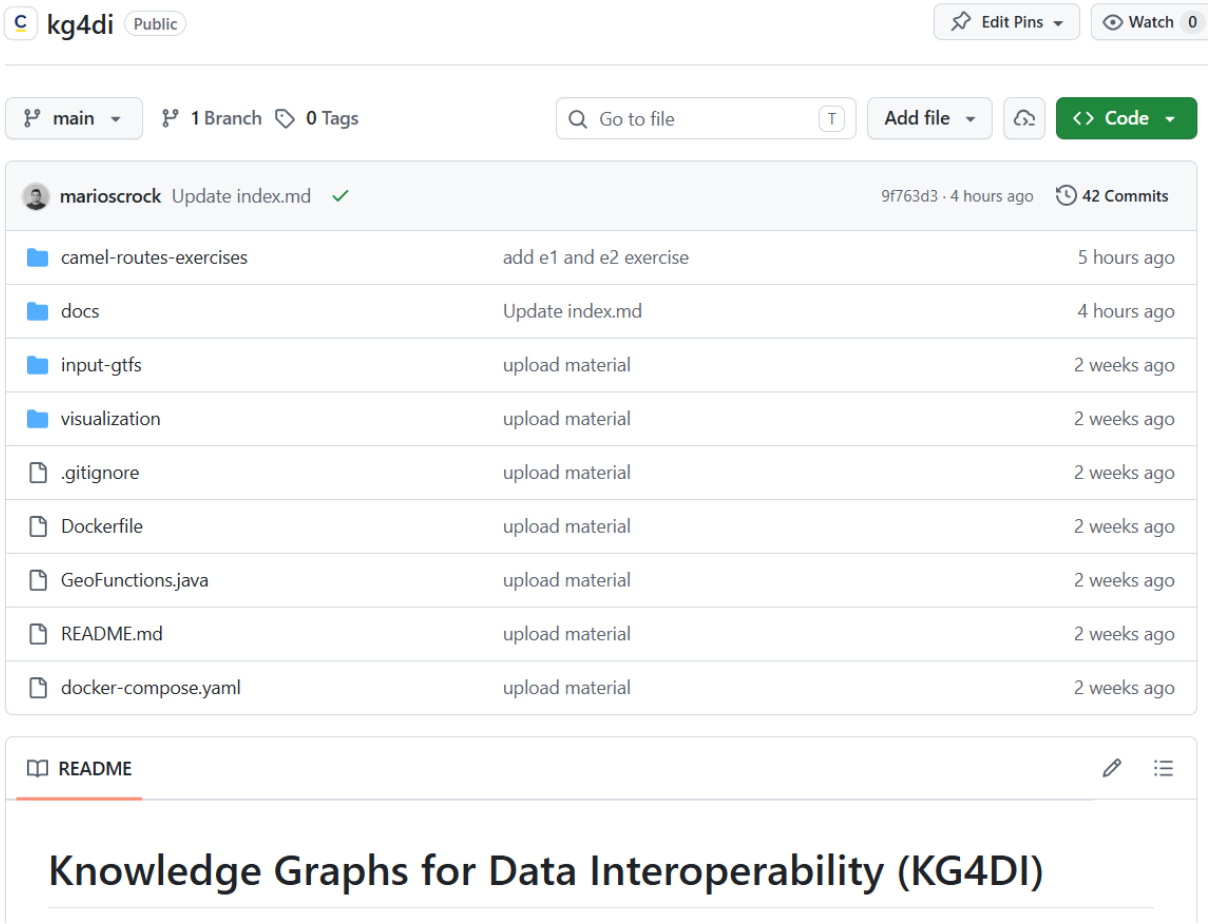
How to follow the exercises

Clone the exercises repository:
`git clone https://github.com/cefriel/kg4di.git`

The tutorial can be followed either by **using a provided Docker image** or by **installing JBang and Camel Jbang** on your machine

Check the README for instructions

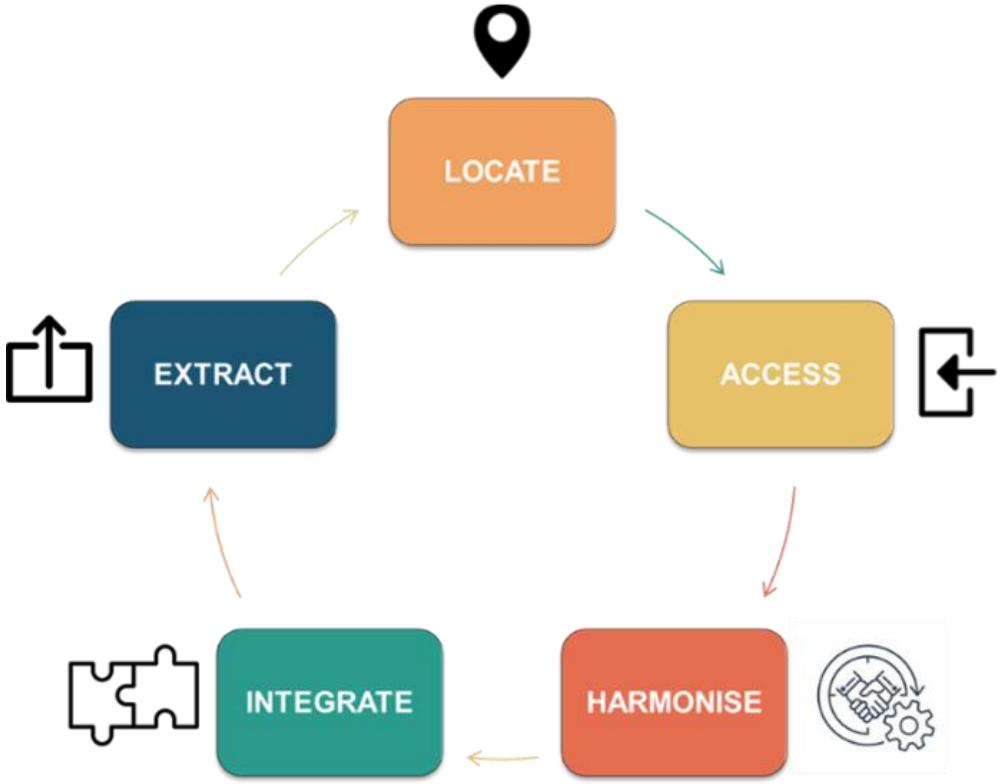
Test that everything works with the `camel-routes-exercise/hello-world` project



DATA INTEROPERABILITY CHALLENGES

PART 1

Data Interoperability Challenge



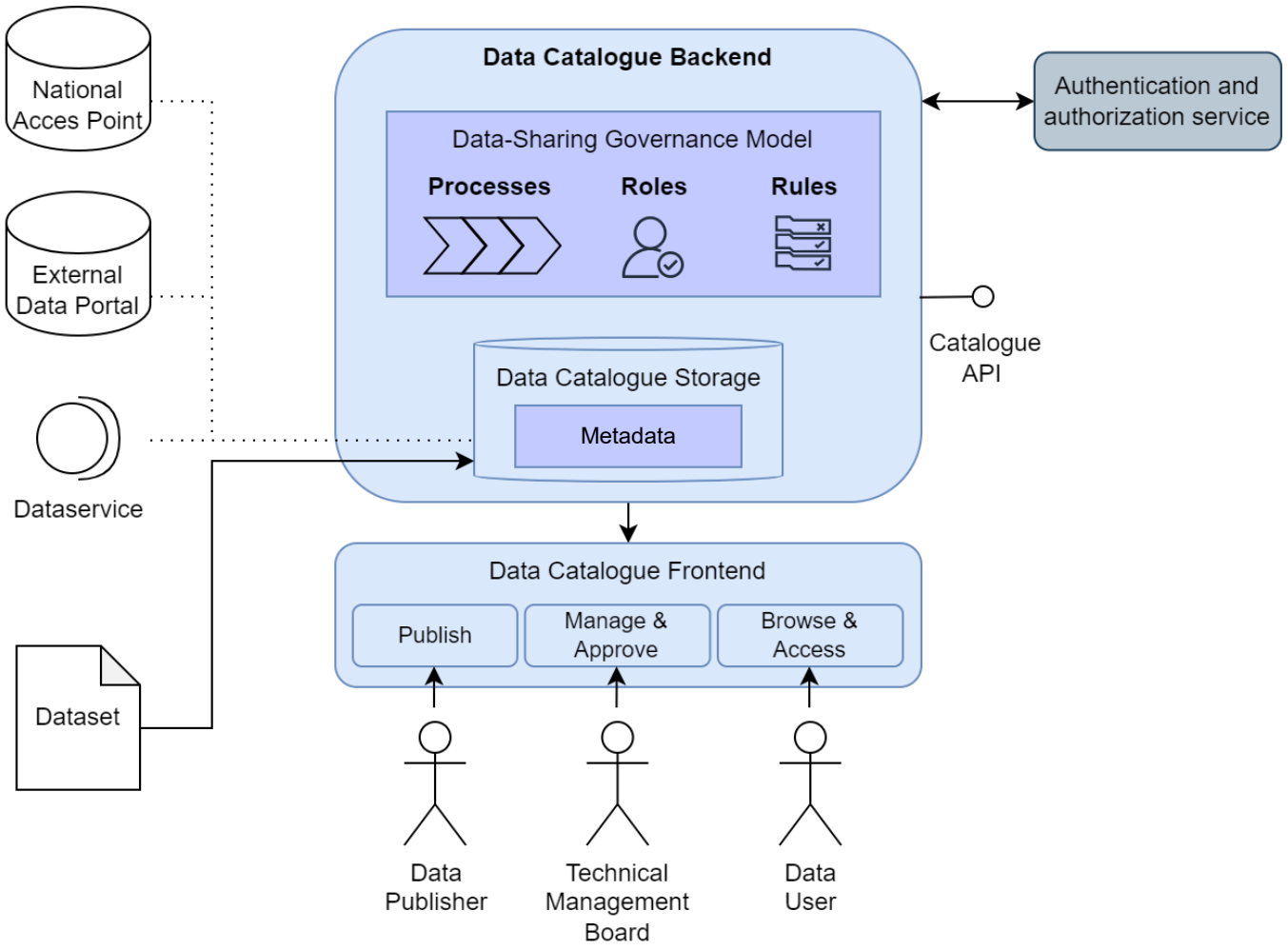
1. **Locate:** *which data is available and where?*
2. **Access:** *how to obtain the data needed?*
3. **Harmonise:** *how to convert data according to a specific data model?*
4. **Integrate:** *how to ensure different data sources can be merged?*
5. **Extract:** *how to consume harmonized and integrated data?*

LOCATE and ACCESS

A Data Catalogue is a **digital platform** supporting the sharing, findability, and accessibility of distributed data sources

- **Description of data sources** adopting a common metadata profile and including information on how to access local/remote data sources
- **Navigation/searching of data sources** through a web interface and according to specific metadata fields
- Implementation of **governance rules**

For this tutorial, **we assume we know target data sources and how to access them**

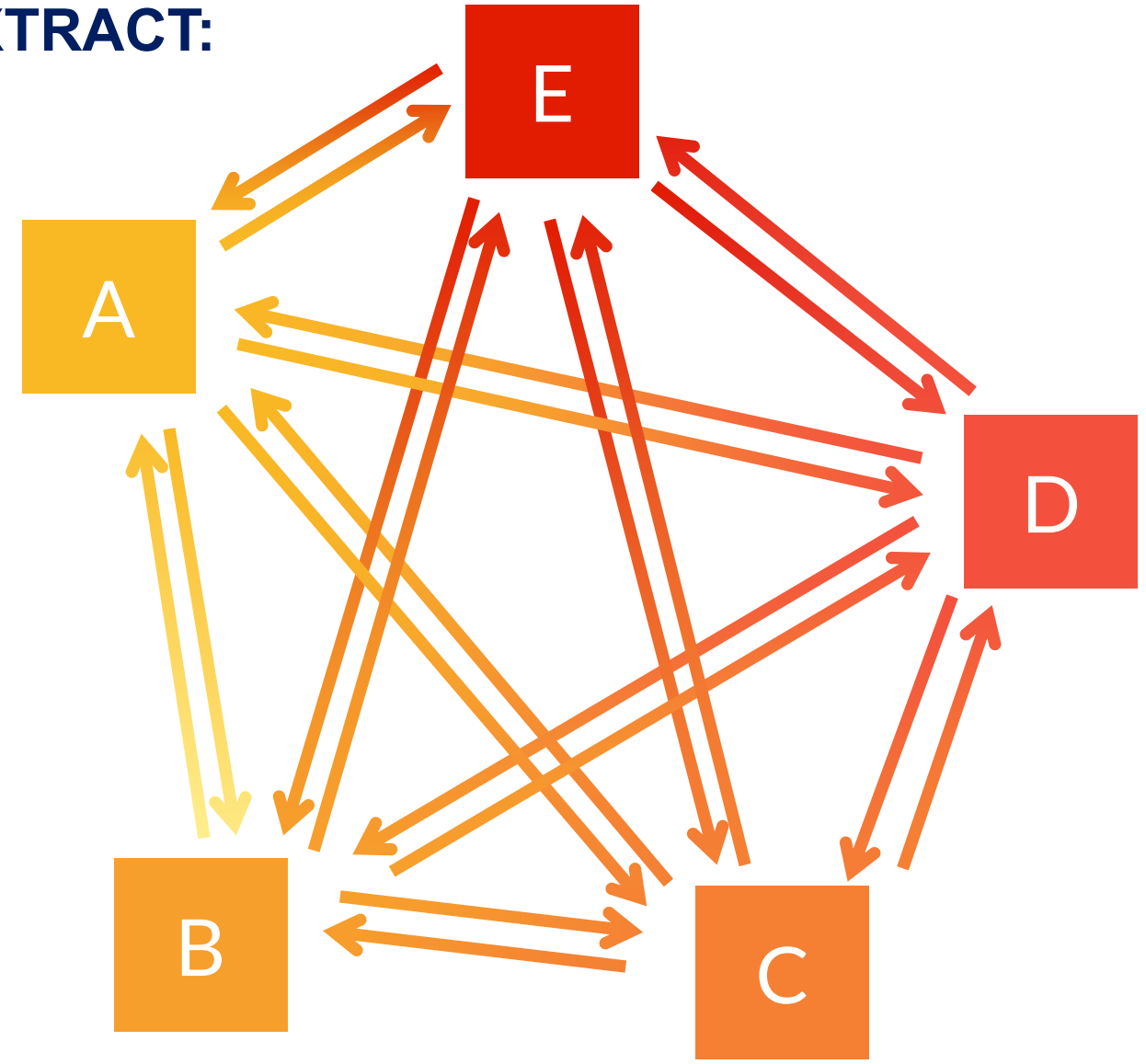


HARMONISE/INTEGRATE/EXTRACT: Naïve Solution

Point-to-point system integration **doesn't scale**

Any-to-any mapping approach

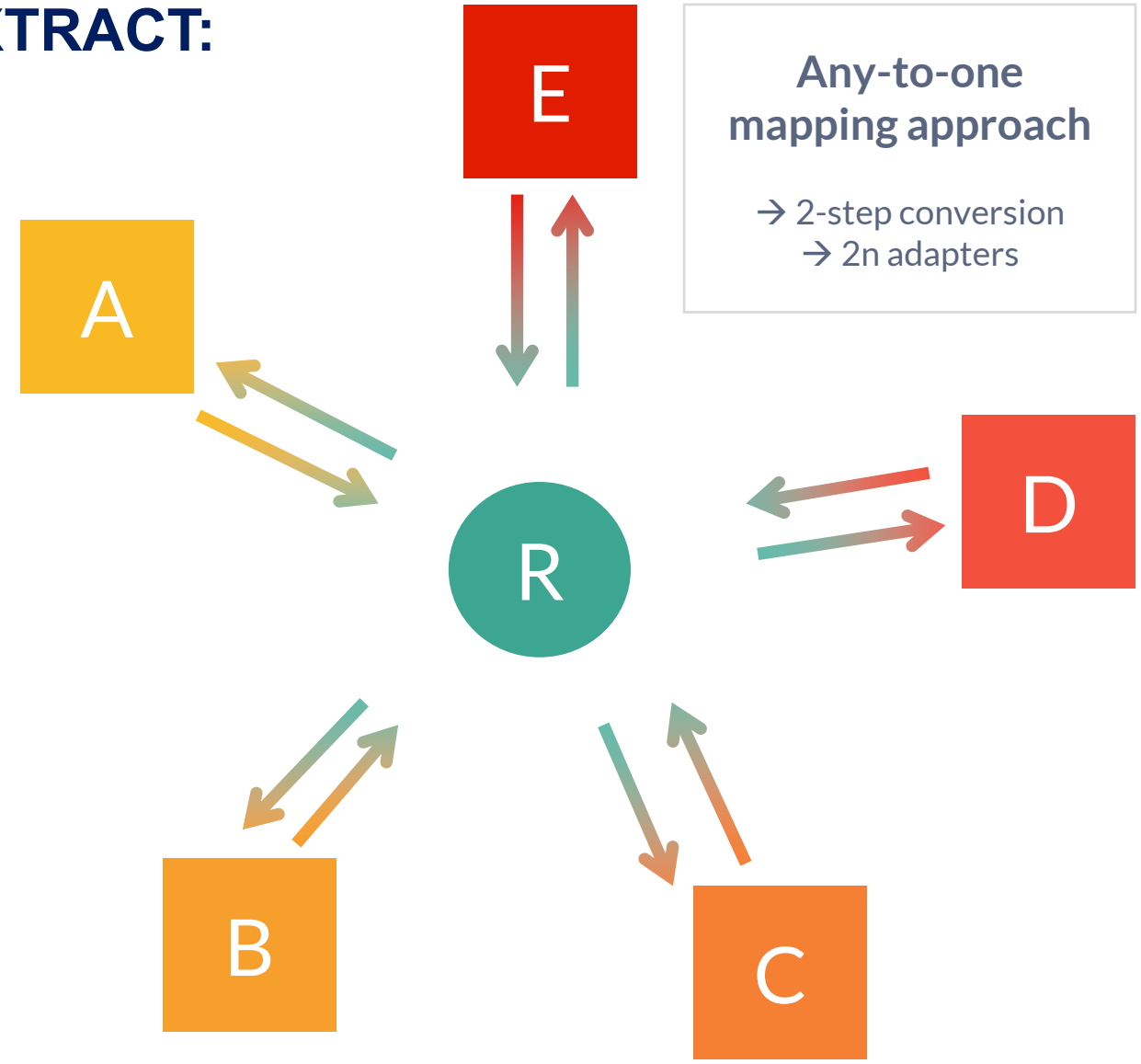
→ 1-step conversion
→ n(n-1) adapters



HARMONISE/INTEGRATE/EXTRACT: Proposed Approach

Adopting an **any-to-one mapping** approach:

- ✓ Stakeholders can **keep using their *current legacy systems***
- ✓ A stakeholder only needs to define **mappings to/from a *reference conceptual* model**
- ✓ ***Scalable* solution** when considering a growing number of systems

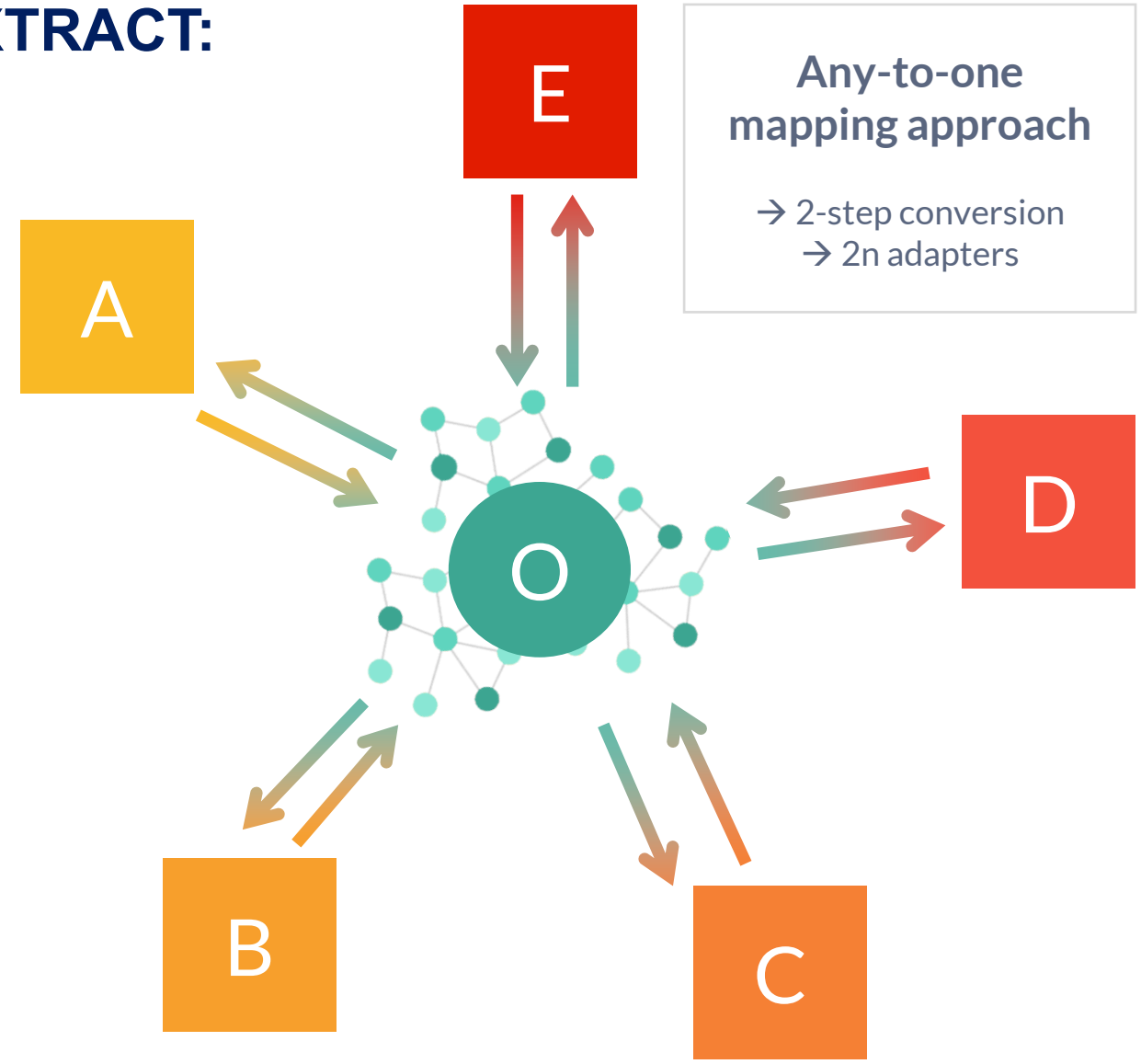


HARMONISE/INTEGRATE/EXTRACT: Proposed Approach

Adopting the **any-to-one mapping** through **Knowledge Graphs**:

- ✓ Conceptual model formalized as (a set of) ***reference ontology***
- ✓ ***Graph representation*** facilitates data fusion and schema flexibility
- ✓ ***Interoperable Knowledge Graph*** as an additional valuable product of the conversion

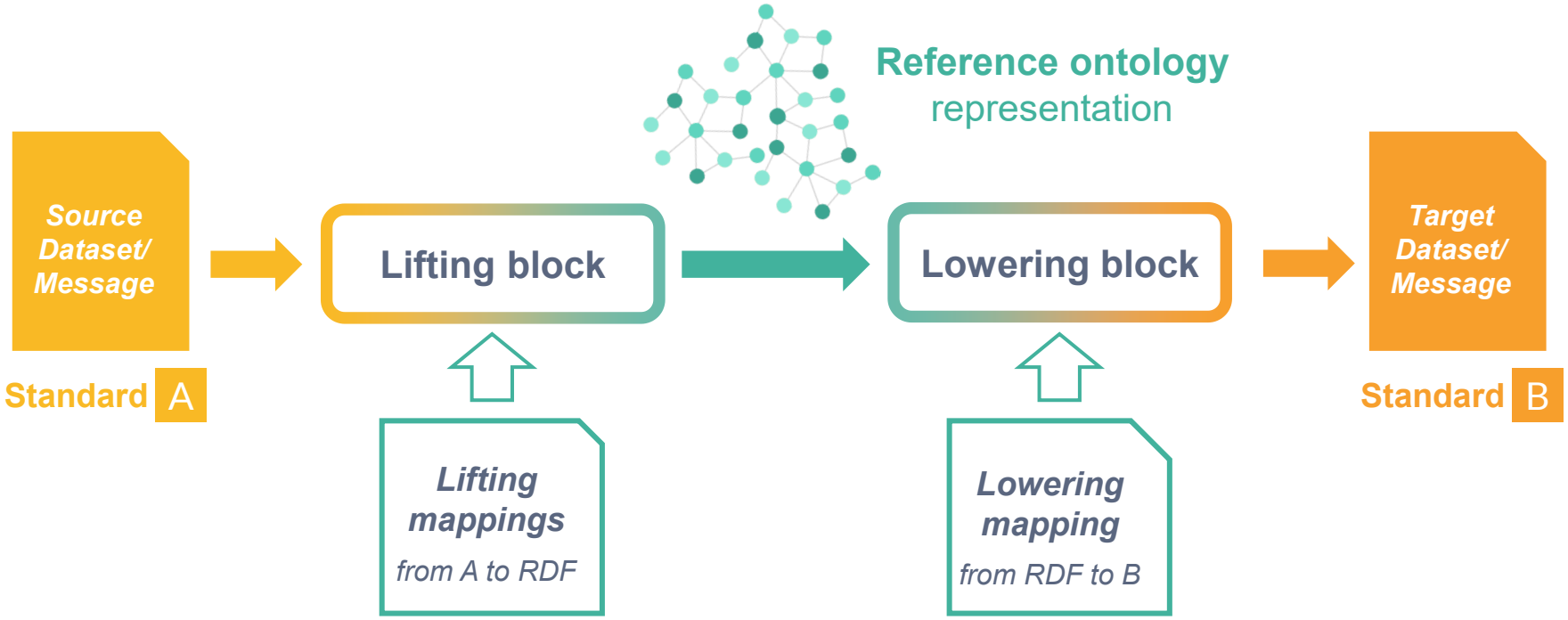
Requires the definition of **semantic converters** to/from the reference ontology



Semantic Conversion with **Declarative Mappings**

We want to leverage declarative mapping rules to define the **semantic conversion** to/from the reference ontology to enable **maintainable and scalable** solutions

- **Reference ontology**: identify common semantics of information to be exchanged
- **Lifting mappings**: we «extract knowledge» from the input according to a reference ontology
- **Lowering mappings**: we «access knowledge» to build the output message



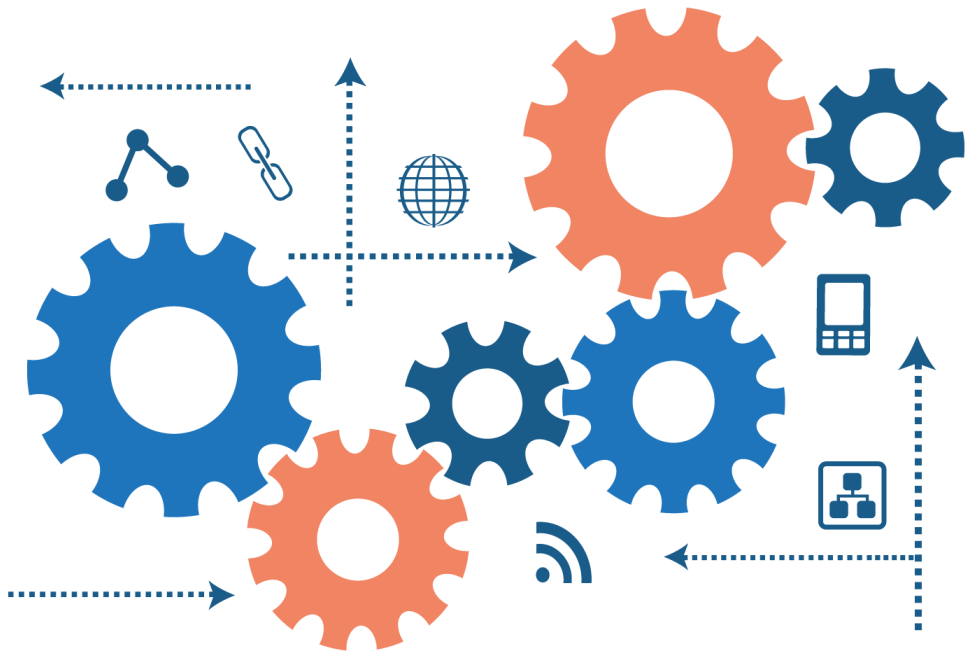
Scrocca M., Comerio M., Carenini A., Celino I. (2020) **Turning Transport Data to Comply with EU Standards While Enabling a Multimodal Transport Knowledge Graph**. In: The Semantic Web – ISWC 2020. Springer. https://doi.org/10.1007/978-3-030-62466-8_26

But it is not that easy…

There is **no single interoperability problem** and, therefore, no single interoperability solution.

A flexible and extensible **set of specialized tools** is needed to cope with different functional requirements for the **integration of semantic converters across heterogeneous information system**

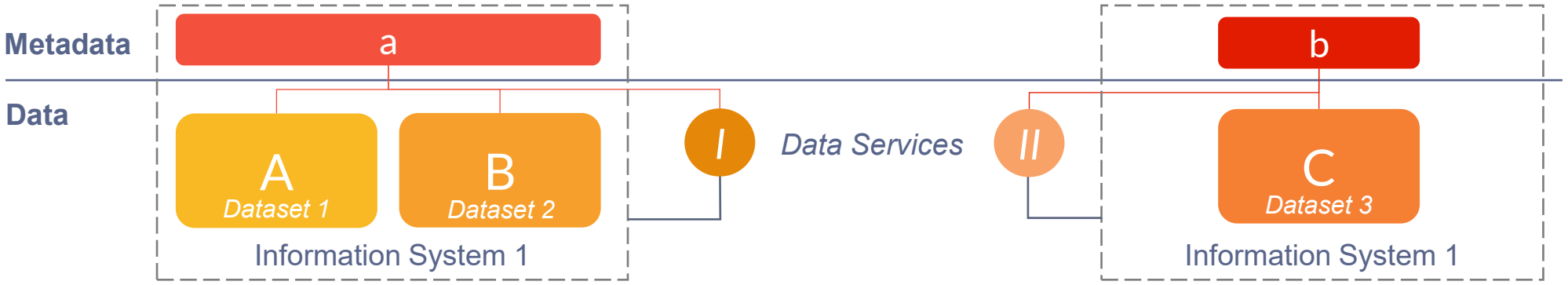
A **modular and configurable solution** is needed to minimise the effort required for integration.



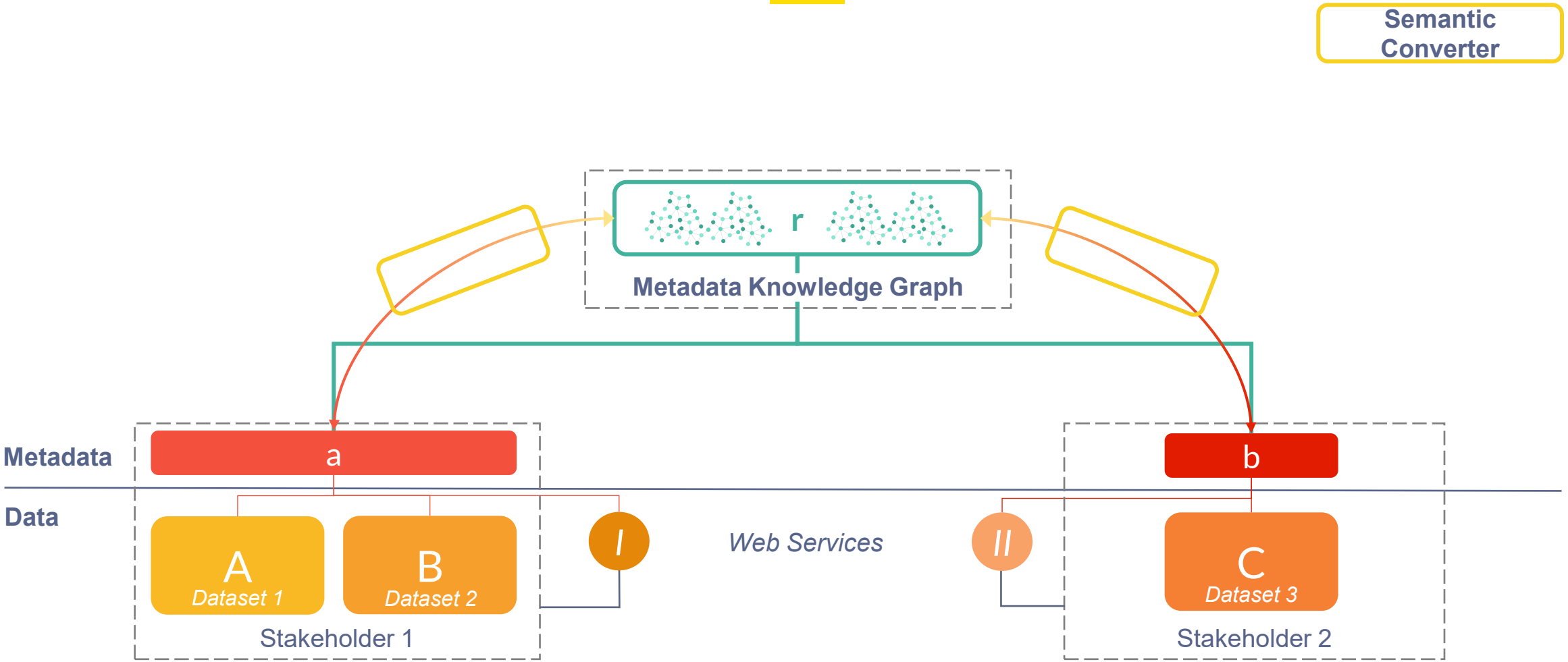
Semantic Conversion **Use Cases**

Given different information systems, we want to enable **interoperability through semantic data converters:**

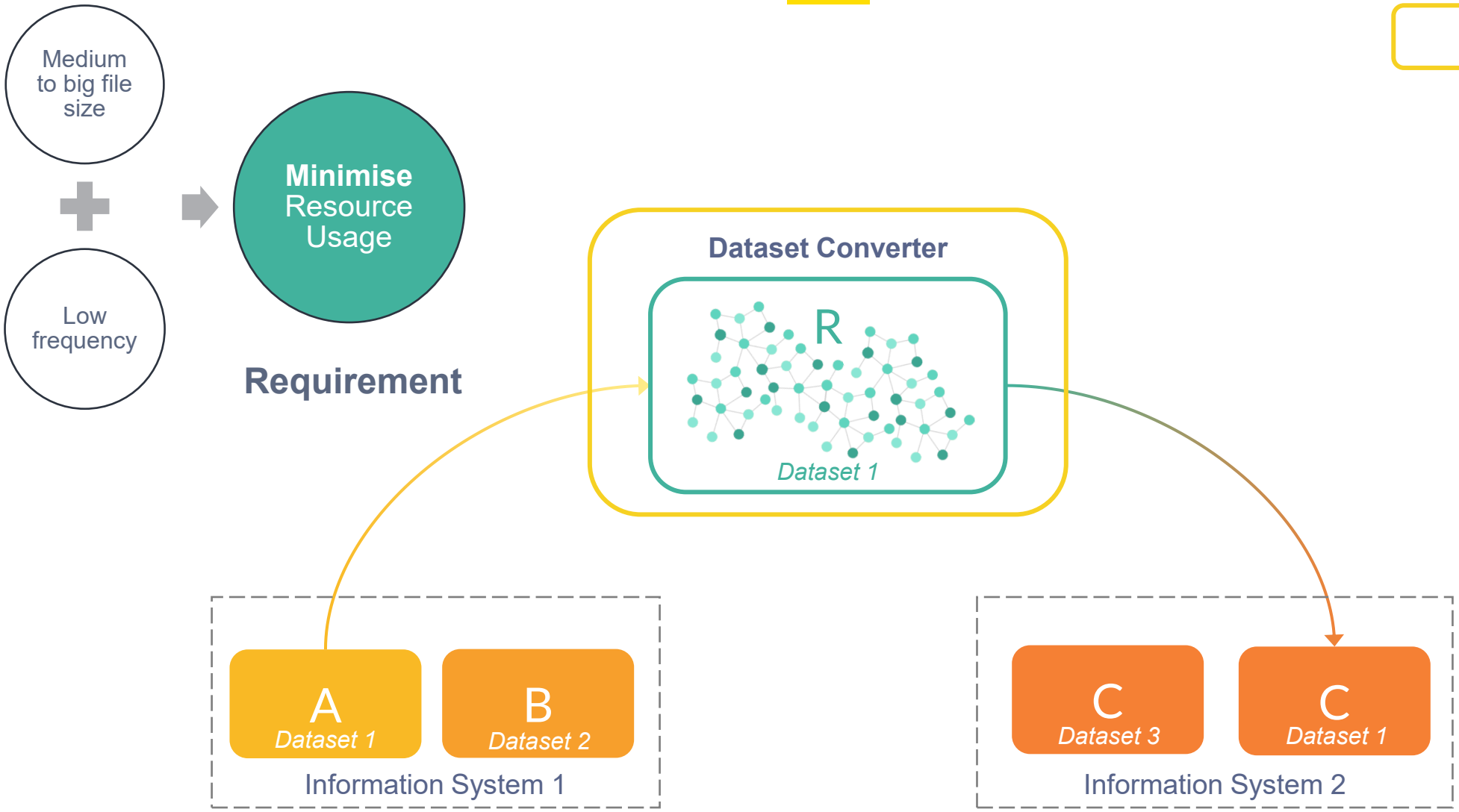
- considering **metadata** using different formats/profiles (a, b)
- considering **datasets** in different formats (A, B, C)
- considering **web services** relying on different API specifications (I, II)
- considering **different requirements** for integration



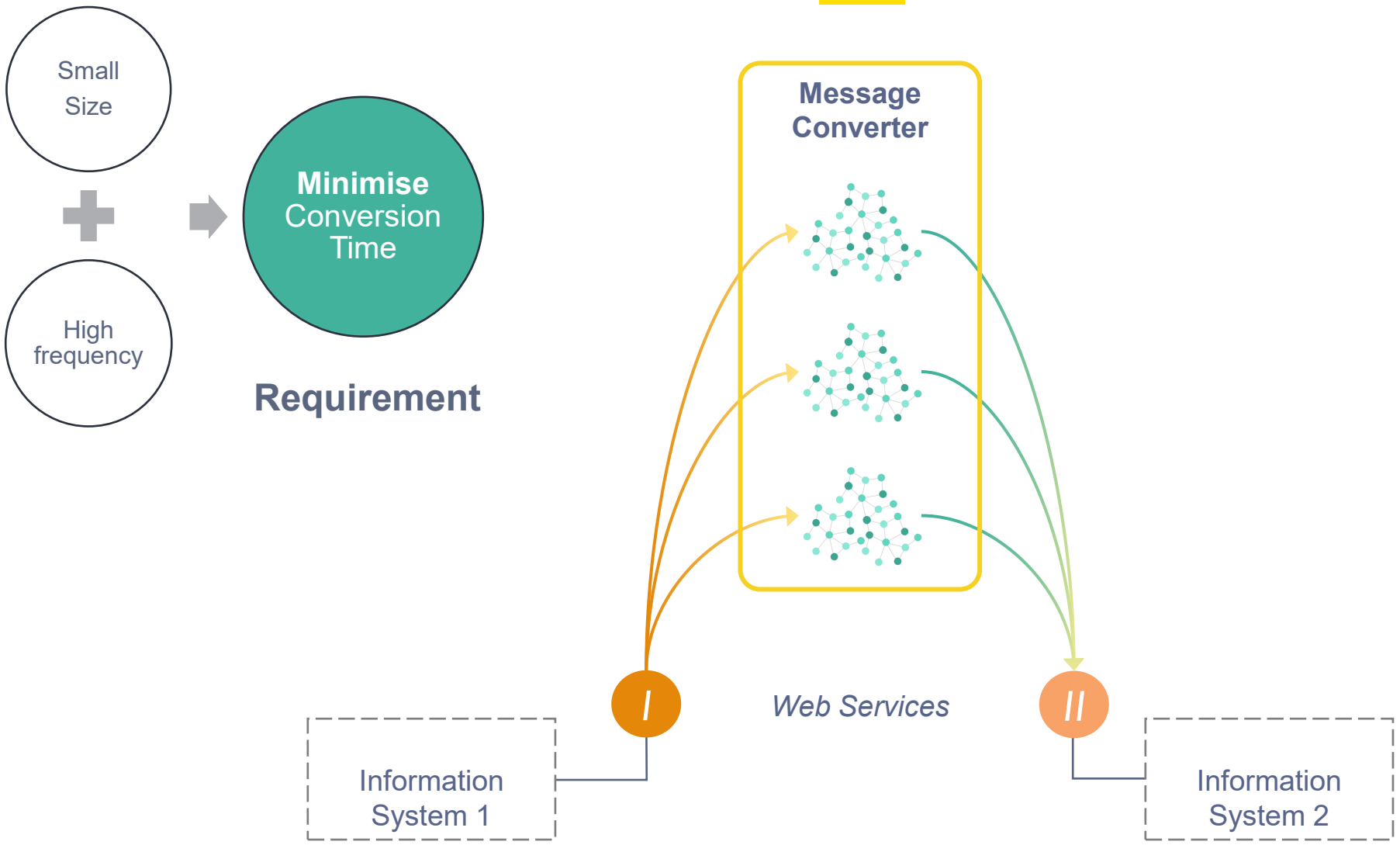
Semantic Data Conversion: **Metadata Harmonisation**



Semantic Conversion: **Batch Conversion Use Case**

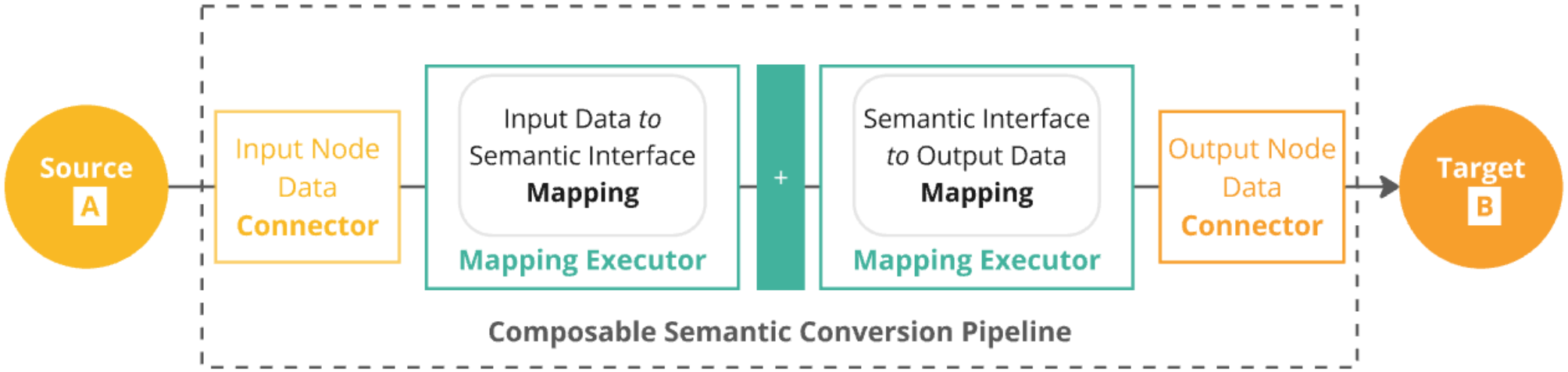


Semantic Conversion: **Service Mediation Use Case**



Composable Semantic Conversion Pipelines

We designed and implemented the **Chimera framework** as a set of modular and configurable components to **define semantic converters for heterogeneous use cases and integration requirements** as composable semantic conversion pipelines.

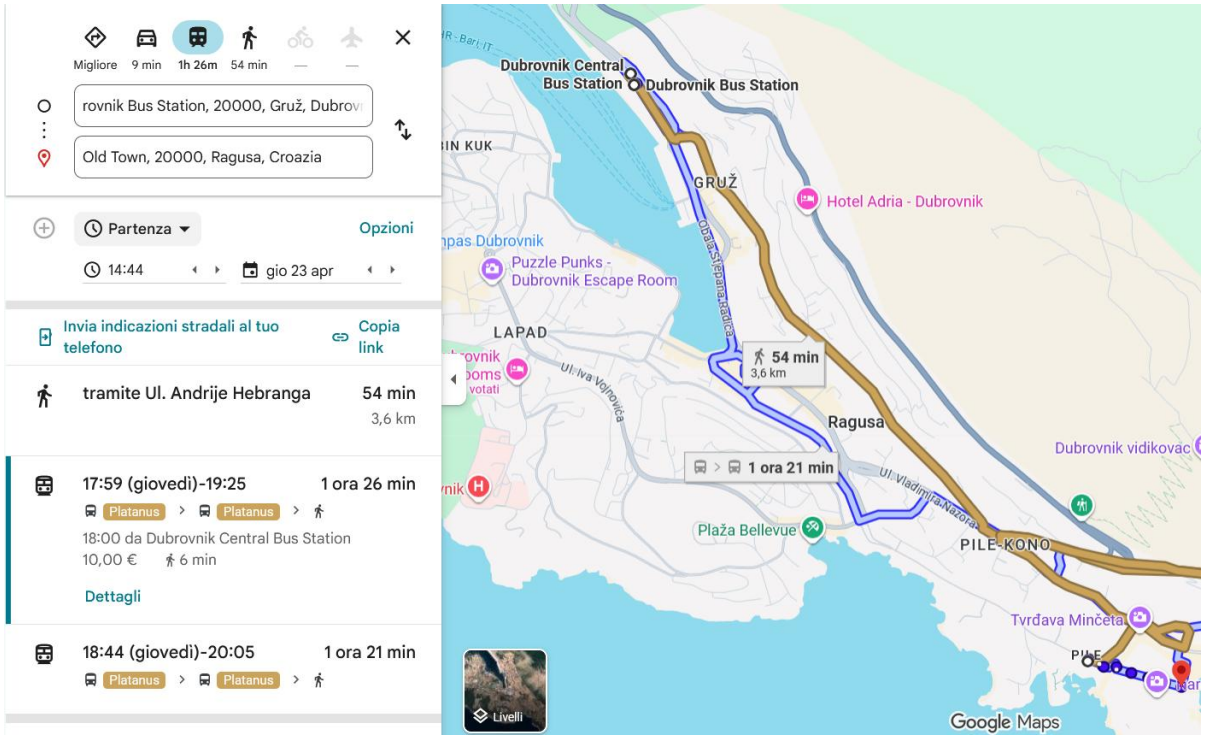


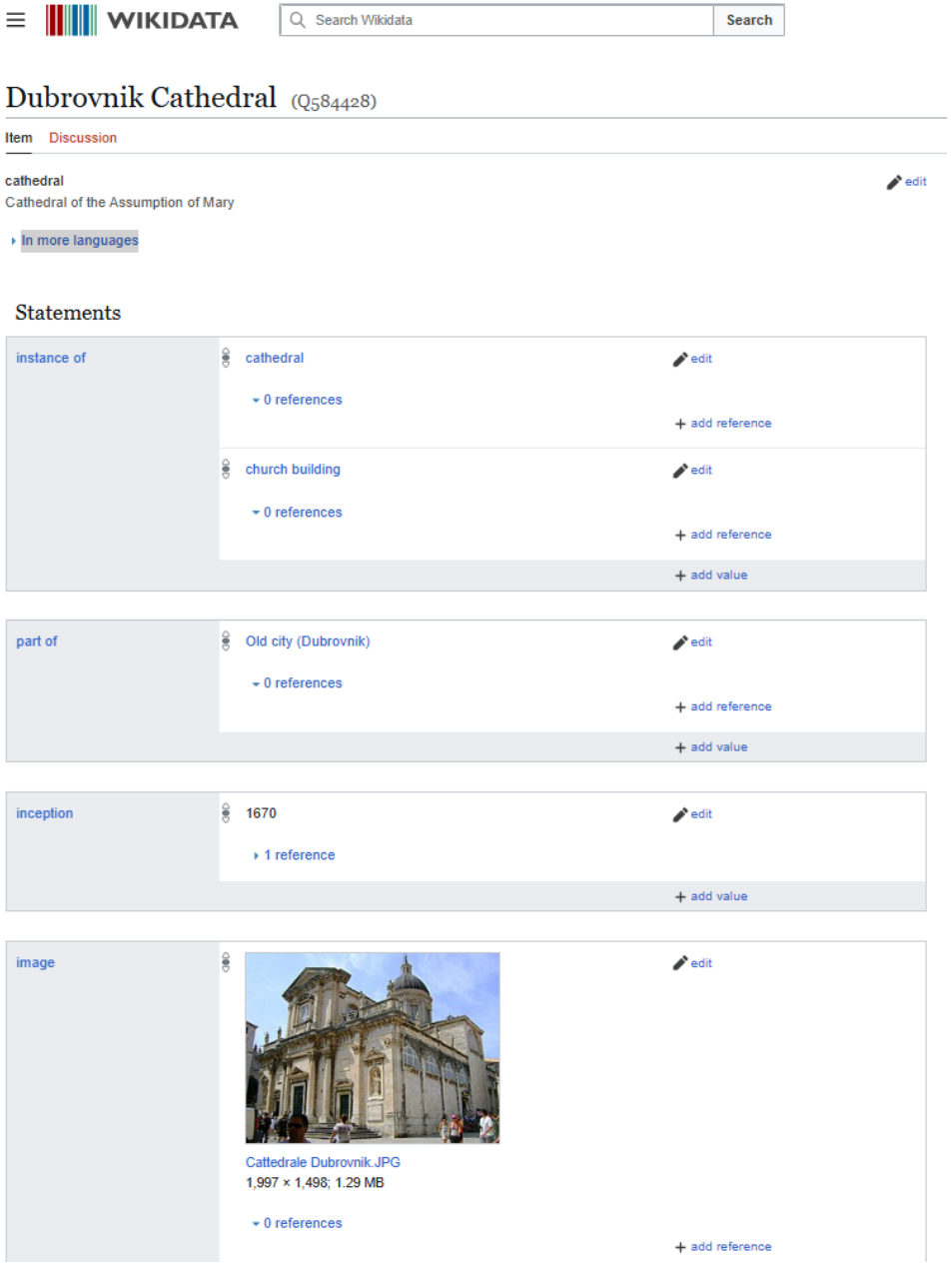
TUTORIAL USE CASE 🗺

TODAY'S SPECIAL 👨‍🍳

Tutorial Use Case

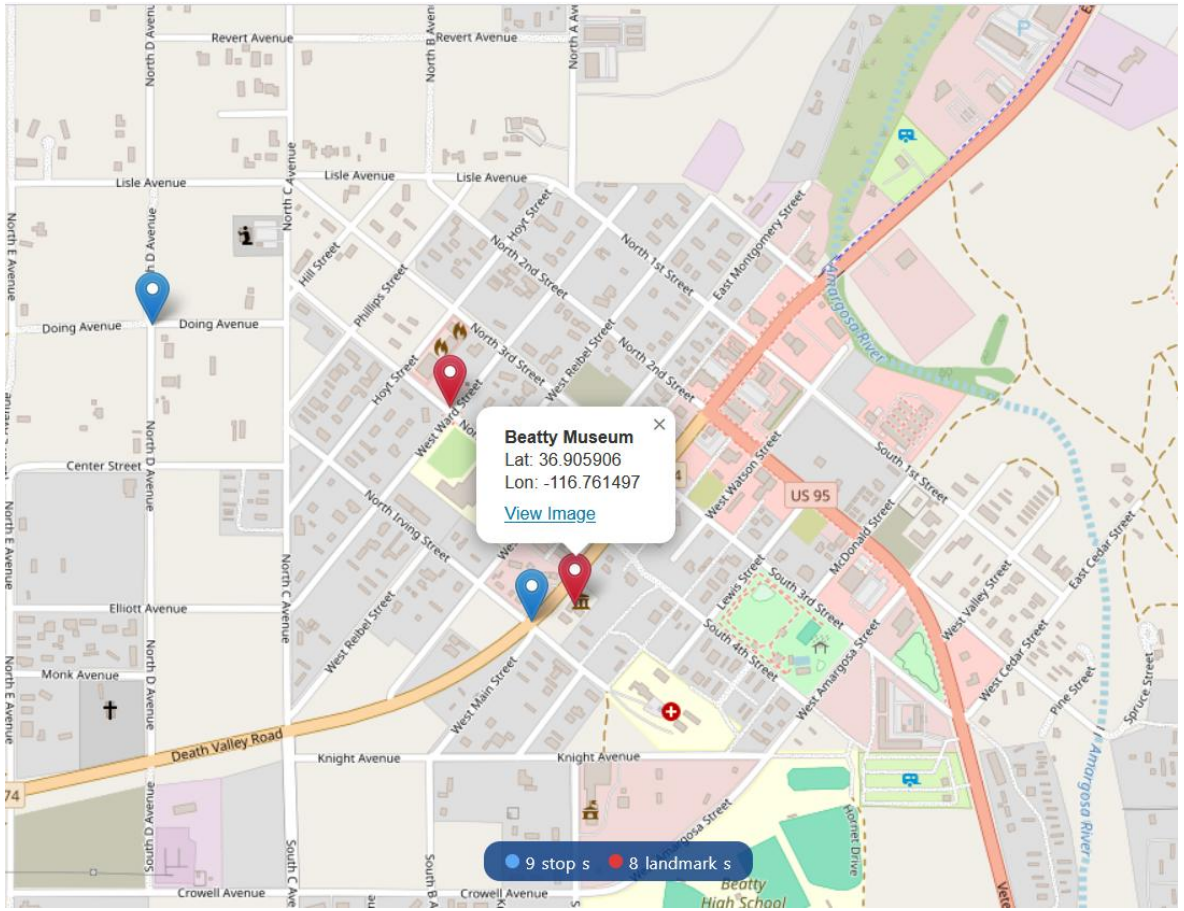
- **Goal:** Interactive dashboard showcasing **public transport stops with nearby landmarks**





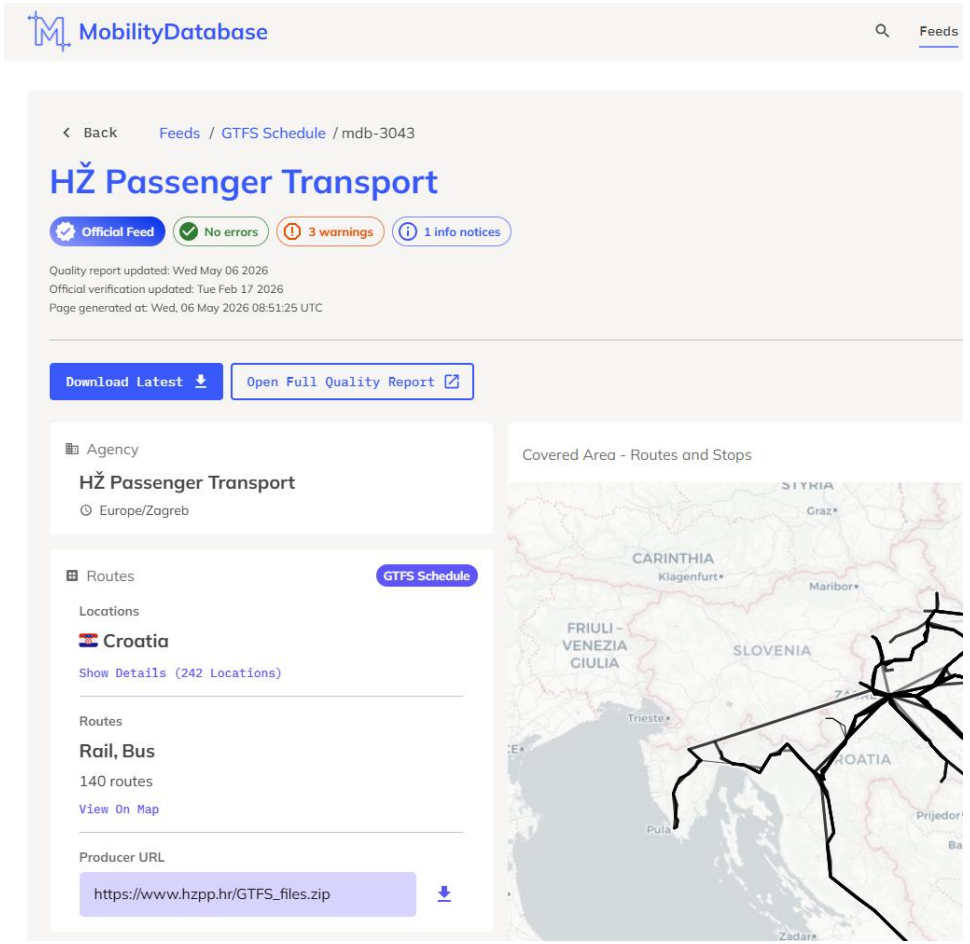
Tutorial Use Case

- **Retrieve public transport stops** for a specific city
- **Integrate information from Wikidata** about landmarks near the identified stops
- Provide information to a **target web application** by using the available **API to upload** stops and landmarks
- **Visualize the dashboard online** and navigate stops and landmarks of all participants!



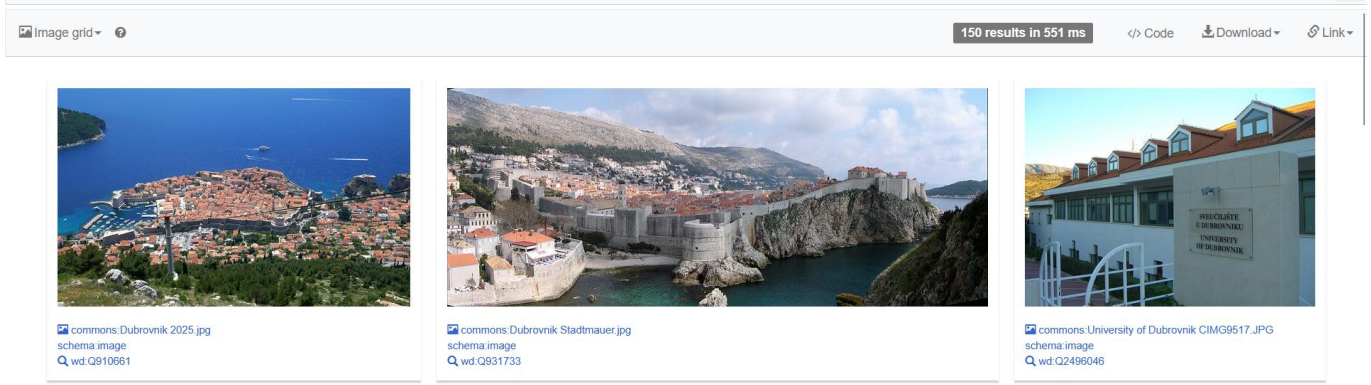
Requirements – ACCESS/HARMONISE

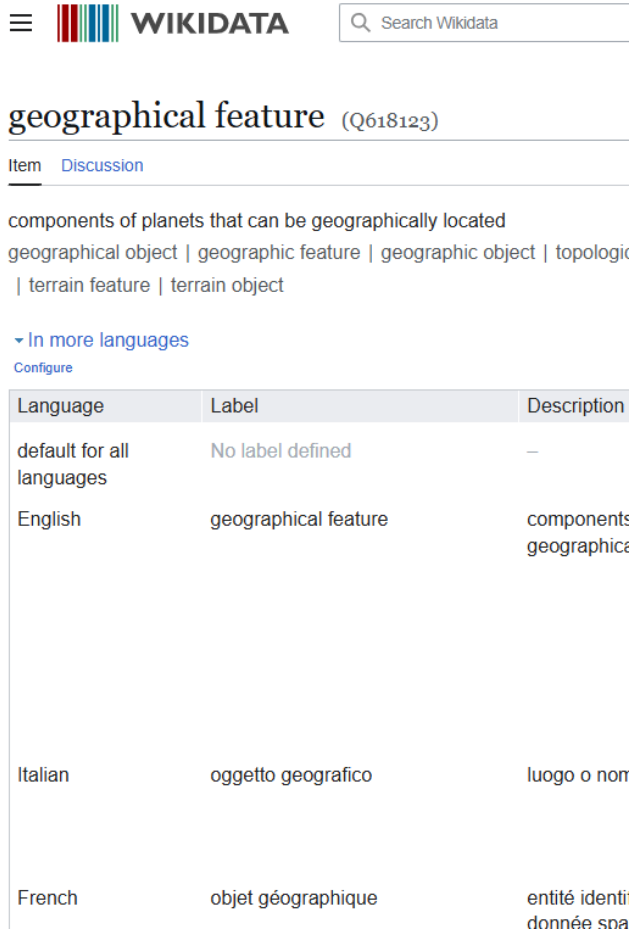
- **Retrieve stops from a GTFS Schedule feed** (ZIP archive with CSV files) used to describe static data about public transport services
  - You can start downloading a GTFS Schedule feed for the city you are from, or a city of your liking from https://mobilitydatabase.org/
  - Not all cities are present, filter for “GTFS Schedule”, download the ZIP (we will use it then in the exercise)
- **Harmonise GTFS stops**: Transform the CSV stops data (file stops.txt in the GTFS feed) into RDF triples according to a given ontology (*lifting mapping rules*)



Requirements – HARMONISE/INTEGRATE

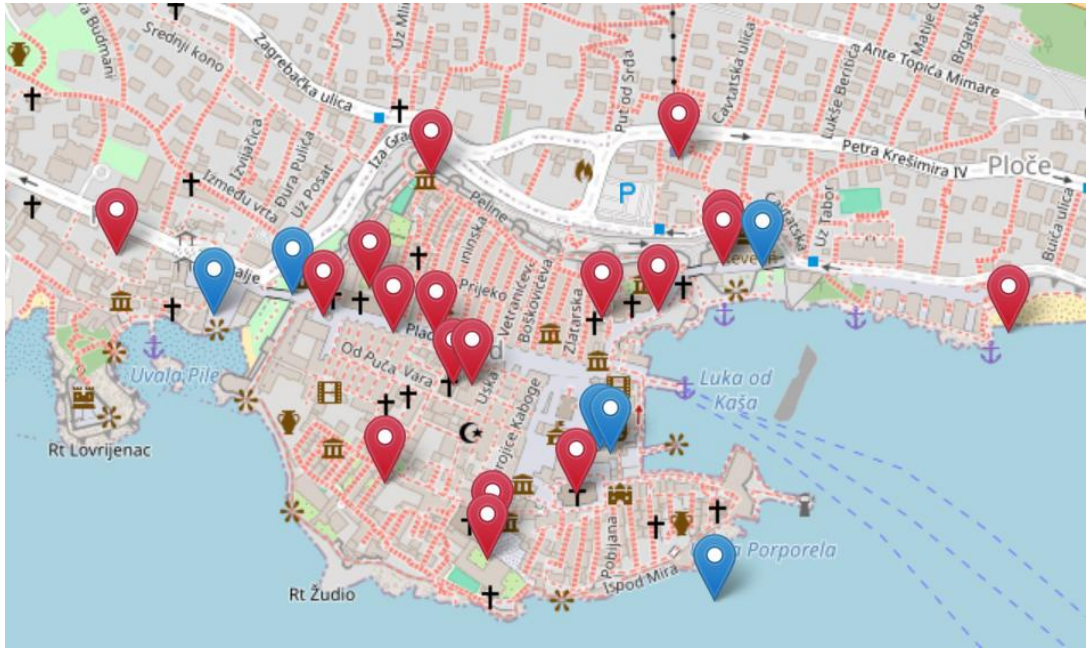
- **Get places of interest near to stops locations from Wikidata:** Generate a query to find in Wikidata the places of interest near to stops locations.
- **Call the remote Wikidata endpoint:** Execute the generated query against the Wikidata SPARQL endpoint and obtain triples on landmarks
- **Aggregate the two graphs:** Merge the graph from GTFS data and the Wikidata subgraph into a single unified graph





Requirements - EXTRACT

- **Transform the graph to target output:** Query the aggregated graph and serialize the result as a flat CSV according to the data format expected by the online dashboard (*lowering mapping rules*)
- **Send the result to the visualization backend:** Make a HTTP POST request with the generated CSV to the remote visualization service



MAPPING APPROACHES

PART 2

Question 1: **How Knowledge Graphs can support mediated data exchanges?**

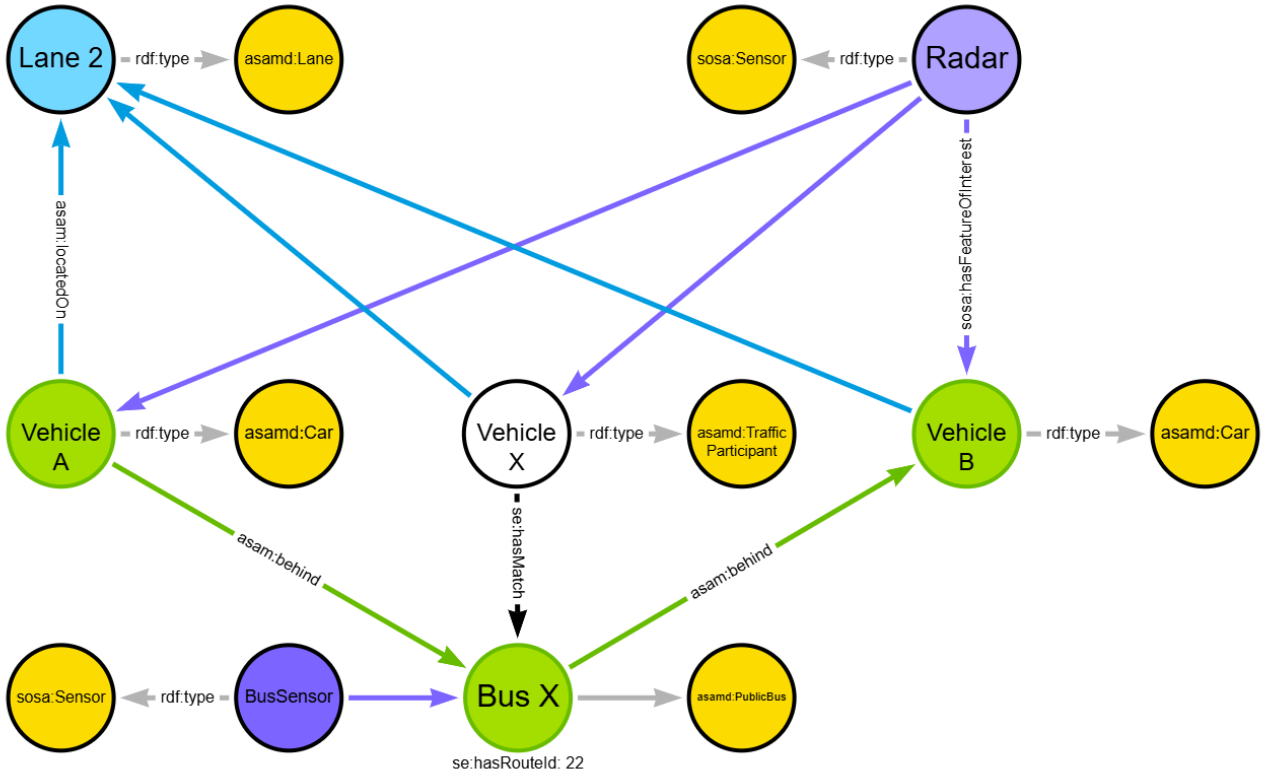
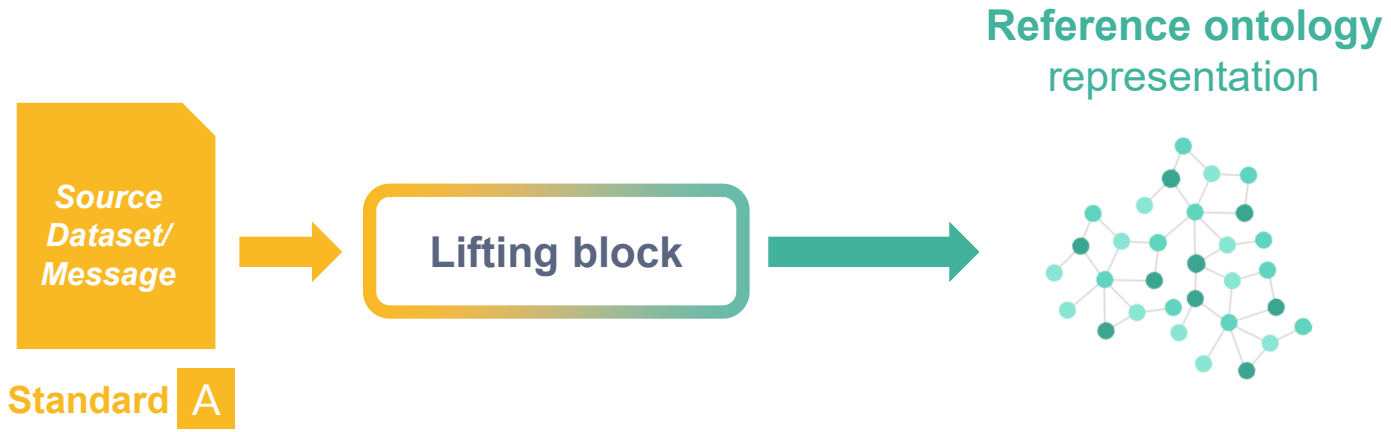


Image from Grassi et al. "*Semantic Low-Code Data Fusion on the Edge to Support Intelligent Traffic Management*" (to be published)

State-of-the-art: **Knowledge Graph Construction**

Lifting mappings «extract knowledge» from the input generating an RDF output according to a reference ontology.



Lifting with **Ad-Hoc Scripting**

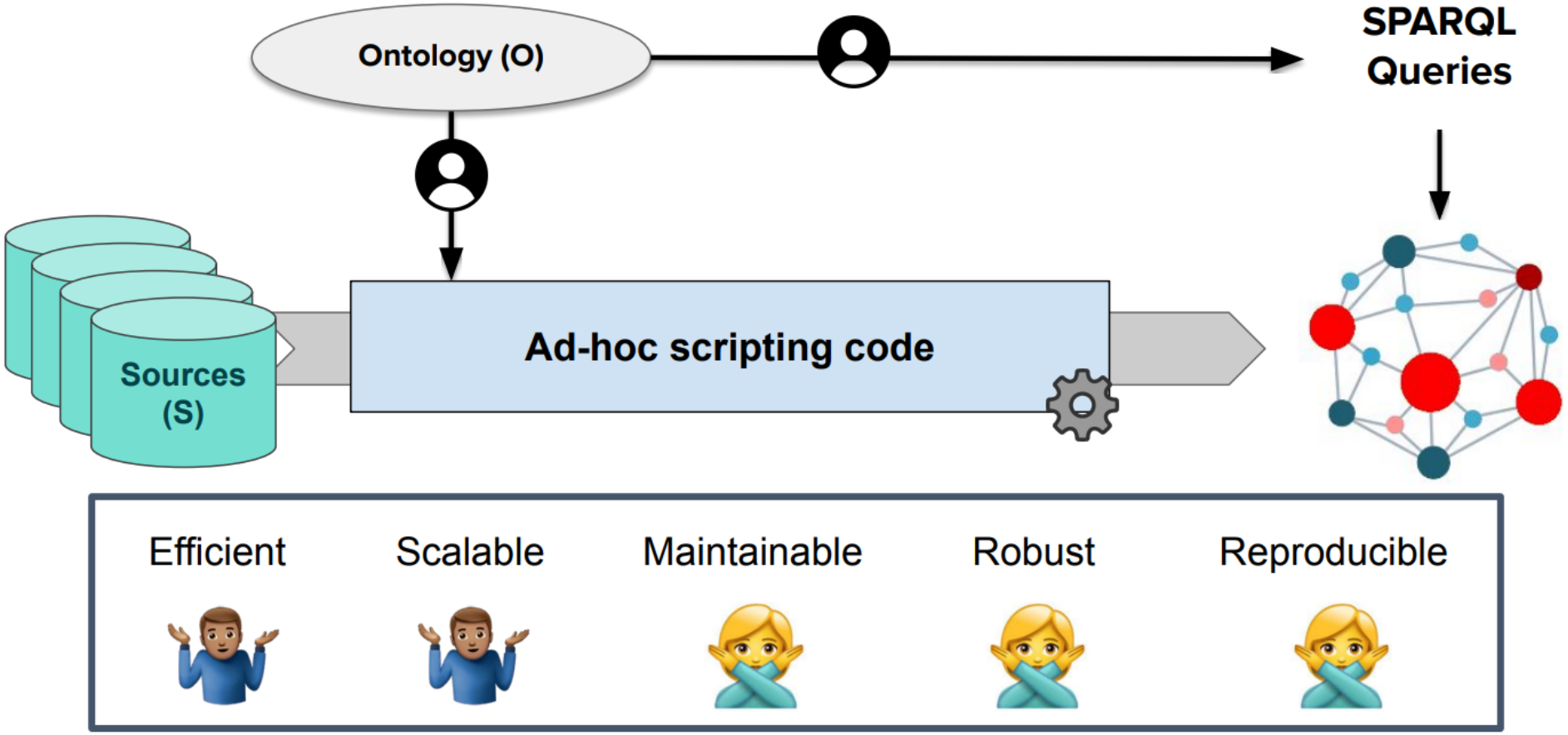
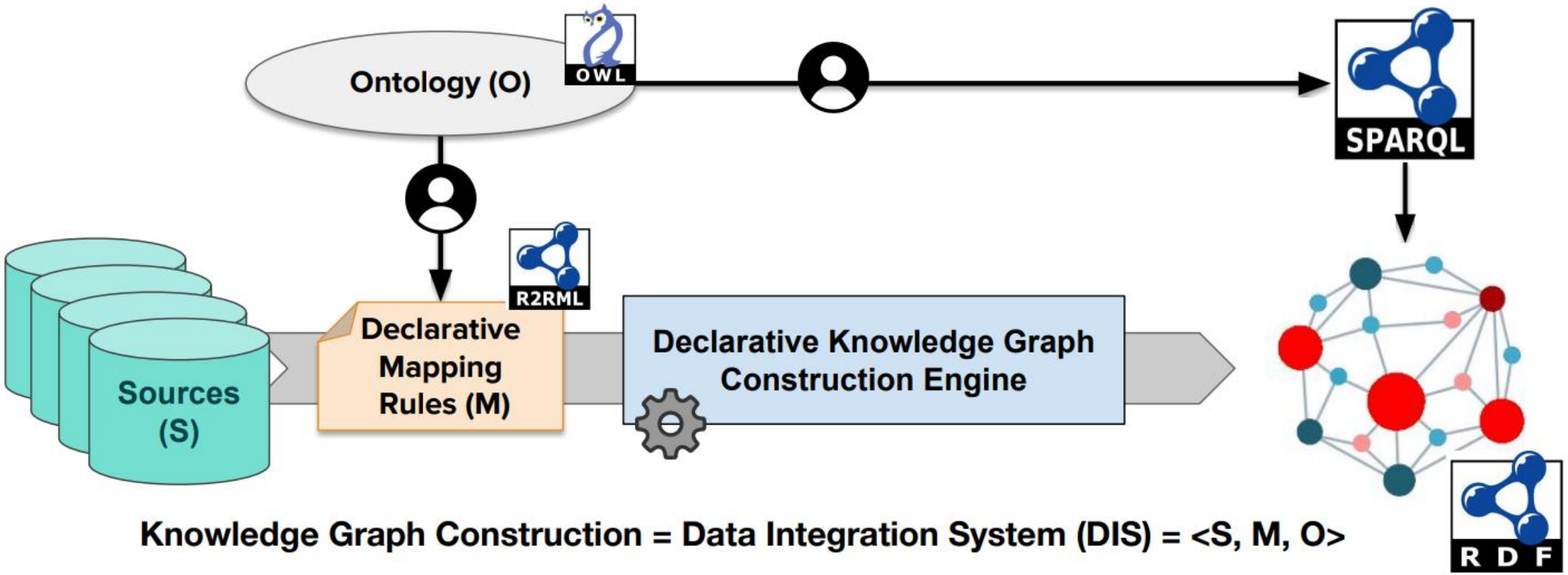


Image from Ana Iglesias-Molina, David Chaves-Fraga, “How do you construct a Knowledge Graph?”, https://davidchavesfraga.com/outcomes/presentations/2024/kgc2024ecai.pdf

Lifting with a **Declarative Approach**



Poggi, A., Lembo, D., Calvanese, D., De Giacomo, G., Lenzerini, M., & Rosati, R. (2008). Linking data to ontologies. In *Journal on data semantics X*
Lenzerini, M. Data integration: A theoretical perspective. In *Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*

Image from Ana Iglesias-Molina, David Chaves-Fraga, “How do you construct a Knowledge Graph?”, https://davidchavesfraga.com/outcomes/presentations/2024/kgc2024ecai.pdf

Mapping Languages for KG Construction

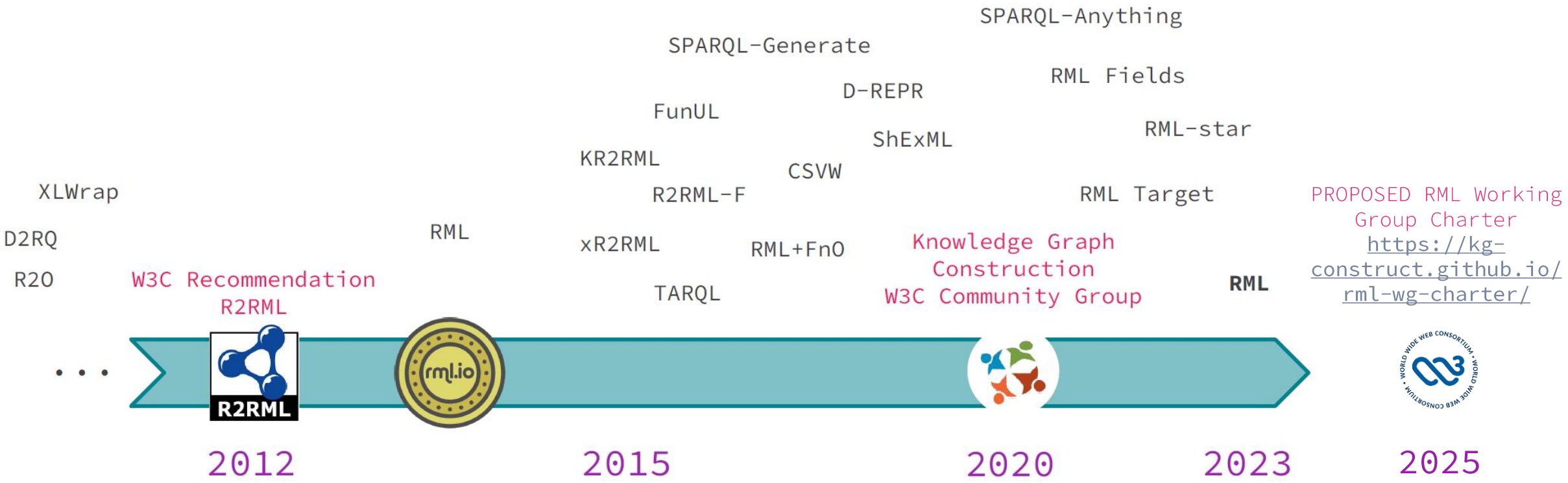
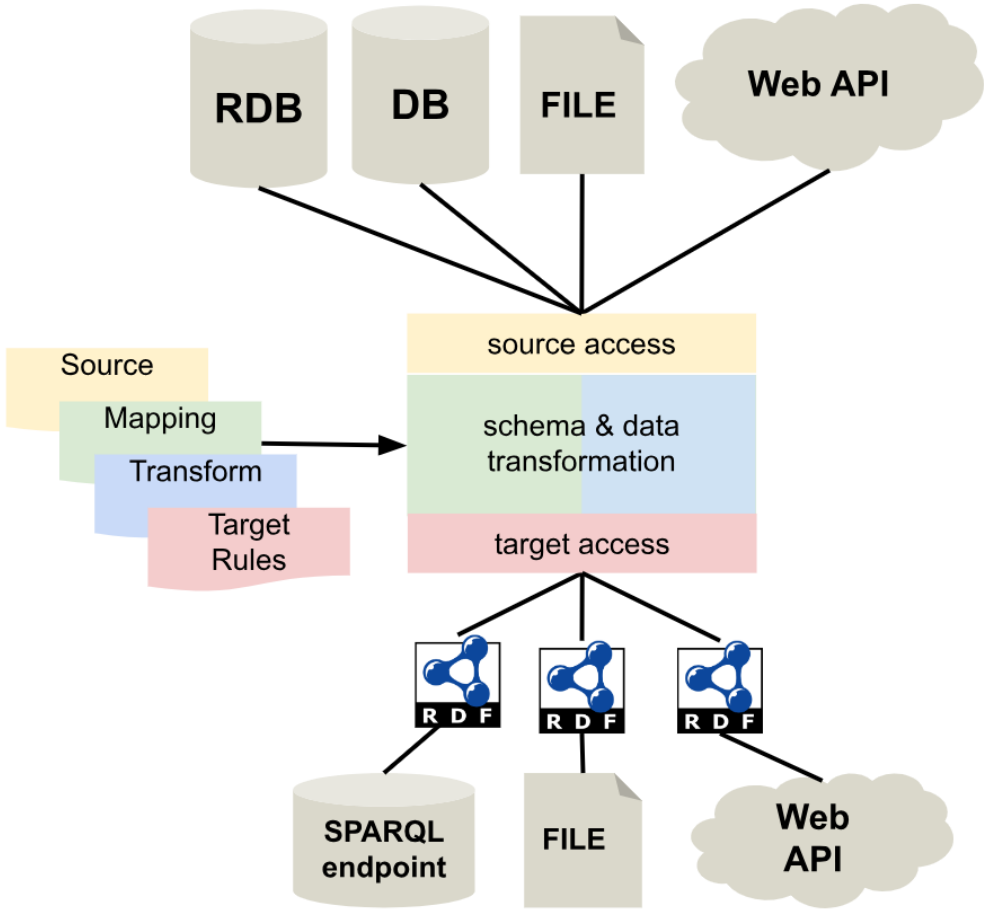


Image from Ana Iglesias-Molina, David Chaves-Fraga, “How do you construct a Knowledge Graph?”, https://davidchavesfraga.com/outcomes/presentations/2024/kgc2024ecai.pdf

RML Mapping Language

- The RML mapping language allows specifying **declarative mapping rules from different data formats to an RDF serialization**
- RML extends the **W3C spec R2RML** (SQL) allowing also mappings from heterogeneous data sources (e.g., CSV, XML, JSON)
- Built-in support:
  - for data enrichment between multiple input data sources (also with different data formats)
  - to apply custom functions in processing the input data
- RML mapping rules are expressed in RDF using a dedicate set of classes and properties. RML is composed of 5 modules (https://w3id.org/rml/portal)
- **YARRRML** (https://rml.io/yarrrml/) is a human-friendly syntax (YAML-based), that can be transformed into RML mapping rules enabling its execution by RML mapping processors

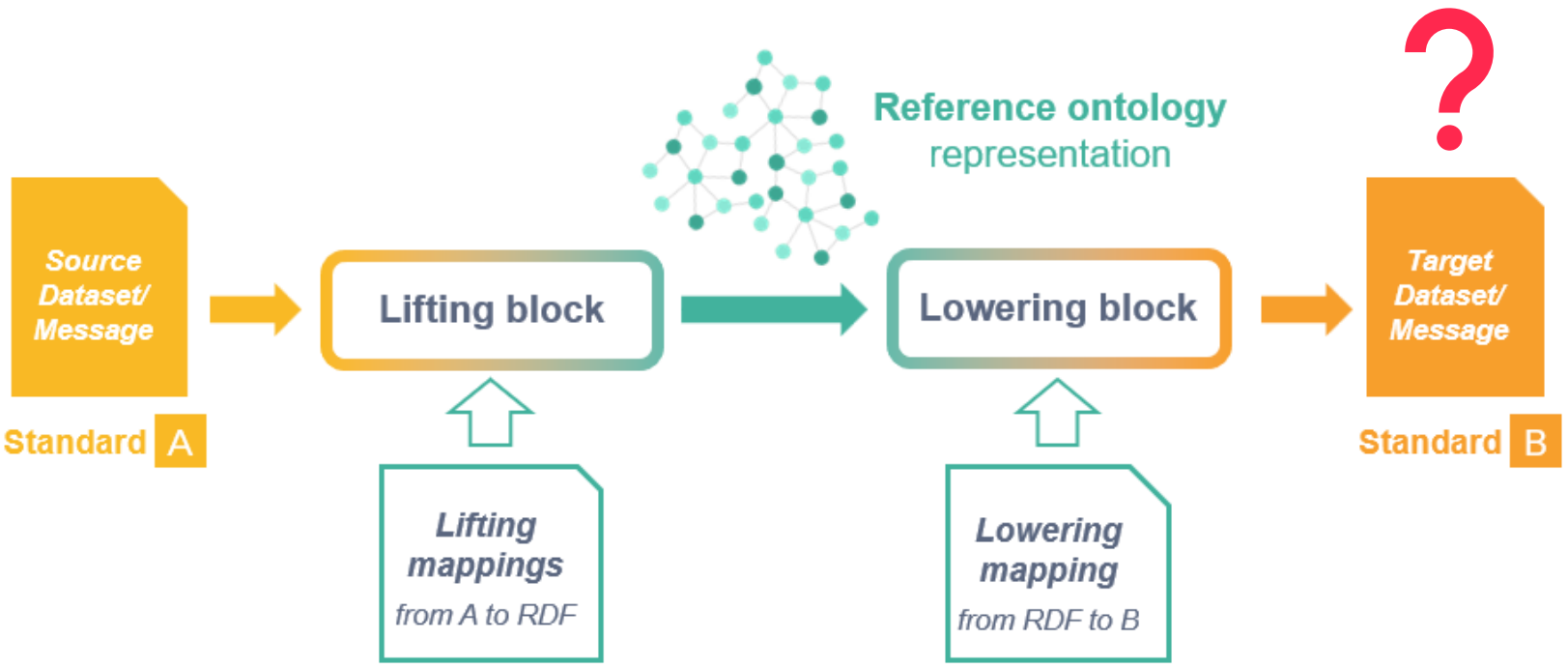


Slide from: https://dylanvanassche.be/assets/leveraging-wot-for-kg-generation

But… not everybody speaks RDF!

The RDF representation enables **interoperability and data fusion** among different stakeholders. However, the **target systems may not be able to "speak" RDF**.

Lowering mappings are needed to build the output message in the target standard.

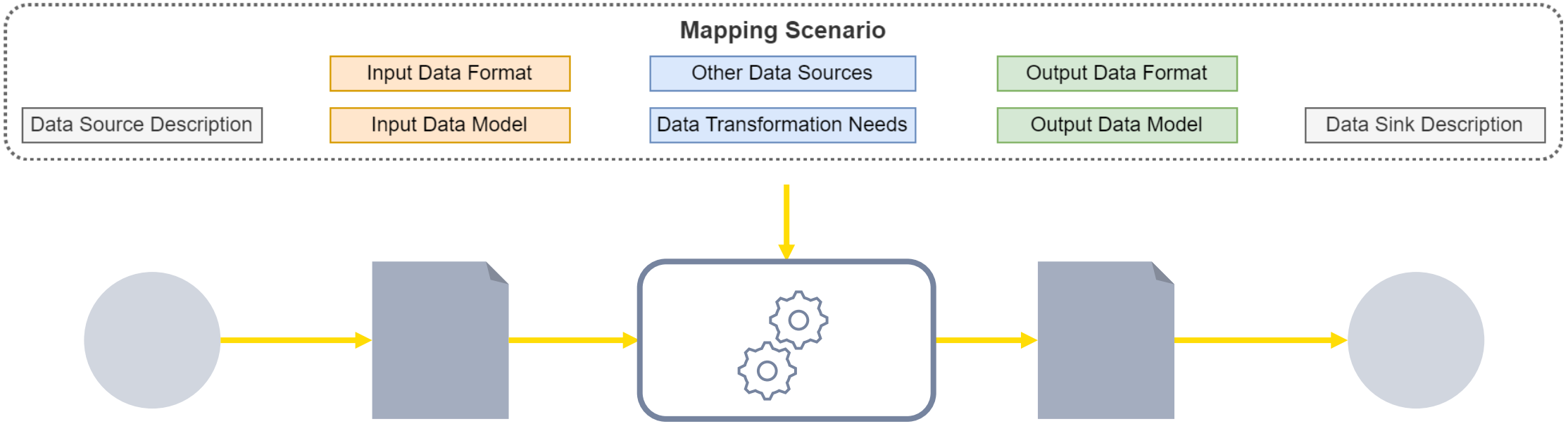


Equivalent mapping language for lowering not existing.

Only tools from RDF to specific target output (e.g., XML).

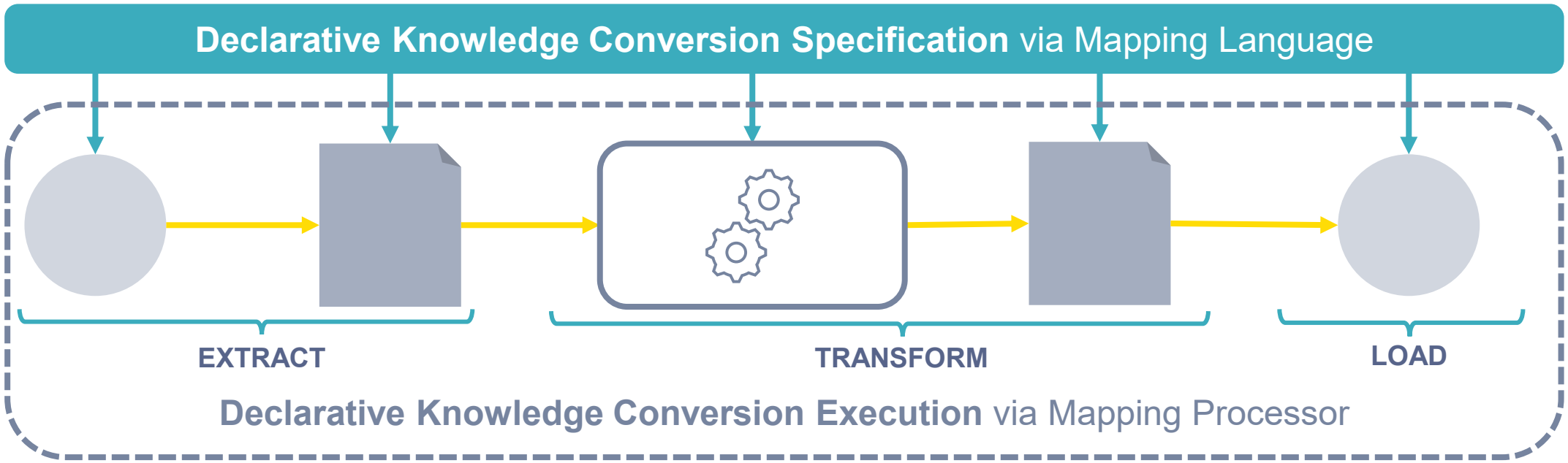
Scrocca, Mario, et al. **"Not everybody speaks RDF: Knowledge conversion between different data representations"**
Fifth International Workshop on Knowledge Graph Construction@ ESWC 2024

A Generic Mapping Scenario



Declarative Knowledge Conversion

Given a **Mapping Scenario**, we want to define a **declarative knowledge conversion** process between different data representations enabled by a **Mapping Language** and a **Mapping Processor** supporting it.



Data Frame Abstraction

SQLTable

Table accessed by Column/Template

R2RML

SQLQuery

SQL Query

Result Set (**Table**)

Tabular (e.g., CSV)

Table accessed by Reference/Template

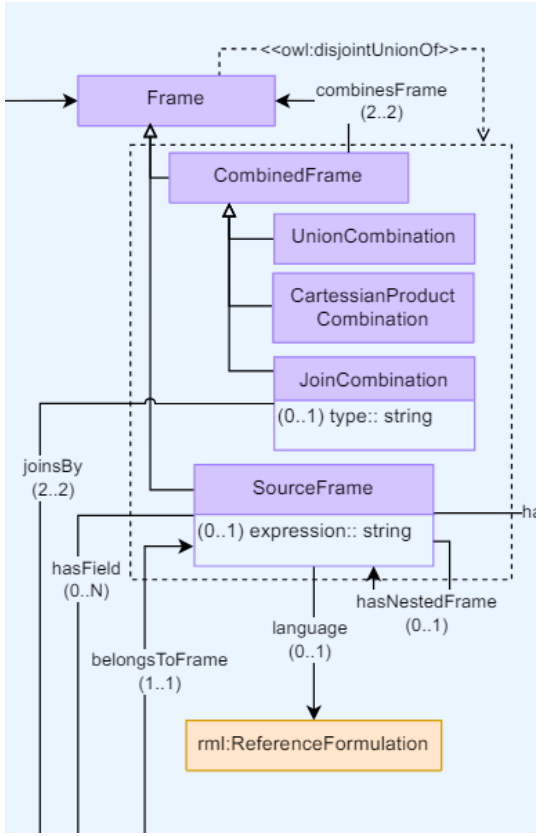
RML

Nested data sources (e.g., XML/JSON)

Iterator

Table: one row for each result of the iterator
Column: access subfields

Conceptual Mapping Ontology [3]



RML Logical Views [4]

W3C Community Group Draft Report

RML Logical Views

Draft Community Group Report 16 March 2026

Latest published version:
https://w3id.org/rml/lv/spec/

Latest editor's draft:
https://w3id.org/rml/lv/spec

Editors:
Pano Maria (ModelDesk)
Els de Vleeschauwer (Ghent University – imec – IDLab)

Authors:
Pano Maria (ModelDesk)
Els de Vleeschauwer (Ghent University – imec – IDLab)
Davide Lanti (Free University of Bozen-Bolzano)

Website
https://github.com/kg-construct/rml-lv/

Abstract

RML logical views is an extension of the RDF Mapping Language (RML) that increases the language's capability to construct RDF datasets from nested input data, to join data sources (also across data hierarchies), and to handle data sources that mix source formats, by allowing to specify a logical view: a flattened, source format-agnostic view over one or more existing data sources. Additionally, it provides a mechanism to express relationships between data sources, as well as additional information about their fields, through structural annotations.

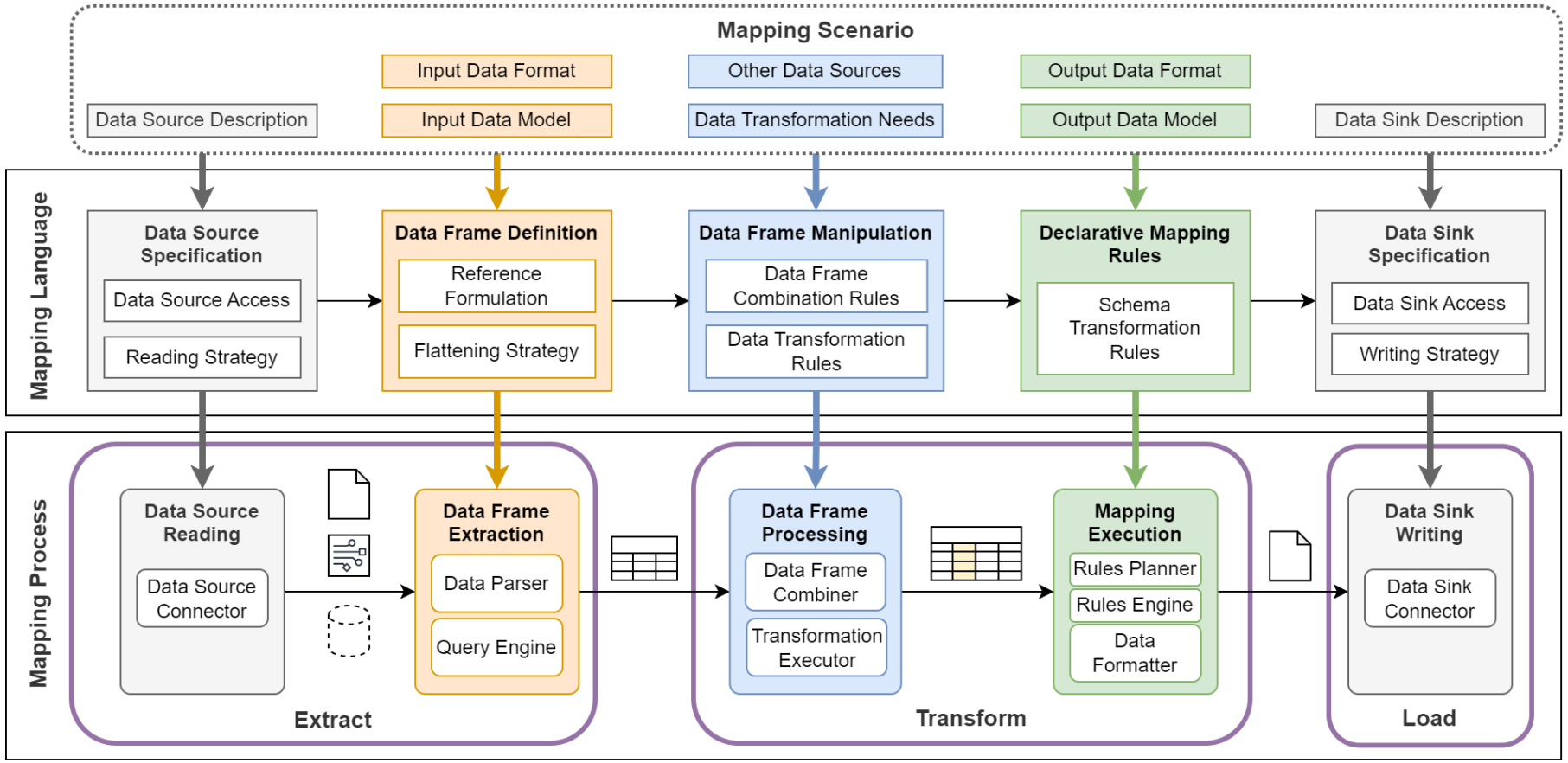
This document describes RML logical views through definitions and examples.

The version of this document is *DRAFT*.

[3] A. Iglesias-Molina et al., **An ontological approach for representing declarative mapping languages**, Semantic Web 15 (2024) 191–221. doi:10.3233/SW-223224.

[4] https://w3id.org/rml/lv/spec/

Mapping Workflow

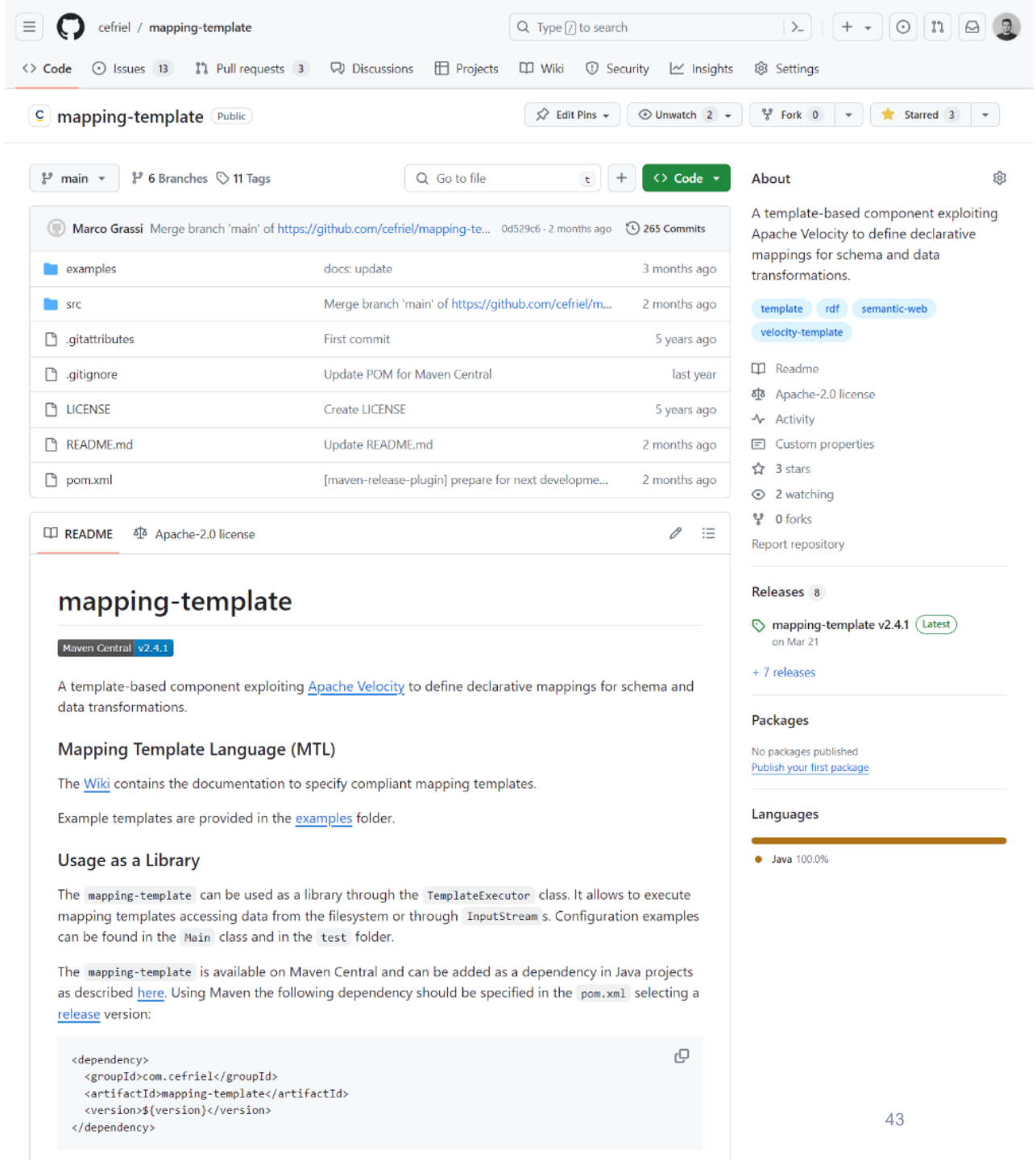


Scrocca, Mario, et al. **"Not everybody speaks RDF: Knowledge conversion between different data representations"**
Fifth International Workshop on Knowledge Graph Construction@ ESWC 2024

mapping-template

Open-source software tool based on the proposed workflow and the Apache Velocity Engine to execute **data and schema transformations**

- https://github.com/cefriel/mapping-template
- Defines a **Mapping Template Language (MTL)** to enable the description of mapping rules based on the data frame abstraction.
- Provides ***Reader*** and ***Formatter*** interfaces to support multiple input/output.
- Currently interfaces for to extract data frames from **CSV**, **JSON**, **XML** and **SQL** (MySQL and Postgres) inputs are implemented.
- Available as a library on Maven Central or as a standalone JAR executable via CLI.
- Supports for **RML** execution (automatic rewriting to MTL)



PRACTICAL SESSION 

MTL MAPPING RULES

Basics Concepts in MTL

- **Built on Apache Velocity (VTL)**, MTL extends VTL syntax to define declarative, template-based mappings for schema and data transformations
- **Format-agnostic approach**, write a single template to transform data into any output format (RDF, JSON, XML, etc.) by mixing target output structure with variable interpolation and control flow
- **Extended with mapping functions**, adds custom utilities beyond VTL: multi-format readers (CSV, JSON, XML, RDF, SQL), and data transformations

```
<html>
  <body>
  #set( $foo = "Velocity" )
  Hello $foo World!
  </body>
</html>
```

Variables in MTL

- Represent dynamic values provided in the template context, for example fields of an object, list of items, etc.

- **Setting variables in templates**
  - You can define/override variables inside a template using the *#set* directive
  - Variable names must start with the '$' character.
  - *#set($var = "value")*

- **How to reference variables**
  - *`$name`* or *`${name}`* to access the variable named `name`.
  - *`$user.name`* to access a property/field `name` on object `user`.
  - If a variable is missing in the context, it may appear unchanged (`$missing`) or be blank depending on configuration.

```
<html>
  <body>
  #set( $foo = "Velocity" )
  Hello $foo World!
  </body>
</html>
```

Iterations in MTL

- `#foreach` lets you iterate over each element in a collection and bind it to a loop variable.
- `#foreach` defines the start of a loop block, and `#end` marks its end.
- Everything between `#foreach' and `#end` is repeated once per element in the collection.

```
#set( $numbers = [1, 2, 3, 4] )

#foreach( $n in $numbers)
  Number: $n
#end
```

Conditions in MTL

- `#if / #elseif / #else` control which block of template code is rendered based on conditions.
- You can chain **multiple** `#elseif` blocks after a single `#if` to test several alternative conditions in order.
- Conditionals can use comparisons (`==`, `!=`, `>`, `<`, `>=`, `<=`) and logical operators (`&&`, `||`, `!`).

```
#set( $hour = 14 )

#if( $hour < 12 )
  Good morning!
#elif( $hour < 18 )
  Good afternoon!
#else
  Good evening!
#end
```

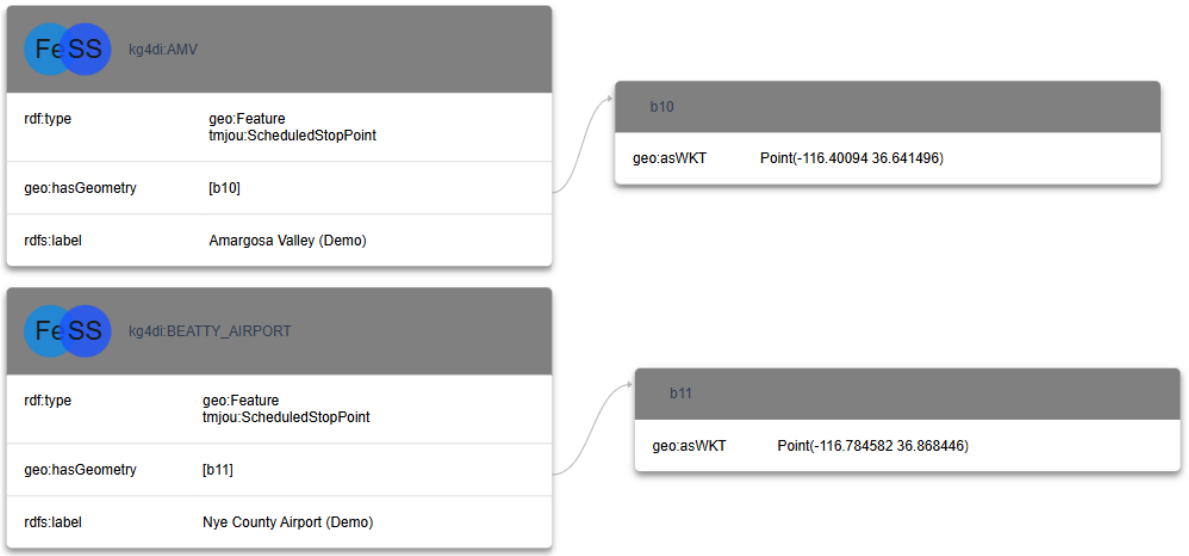
Learning by doing… MTL for our tutorial use case!

INPUT: The **stops.txt** file in a GTFS feed is a CSV file describing public transport stop places.

OUTPUT: We want to map it to the selected reference ontology [1]: every row in **stops.txt** as an RDF resource of type *geo:Feature* and *tmjou:ScheduledStopPoint*, with its label and geographic coordinates as WKT geometry.

[1] https://knowledge.c-innovationhub.com/tangent/schema

```
stop_id,stop_name,stop_desc,stop_lat,stop_lon,zone_id,stop_url
FUR_CREEK_RES,Furnace Creek Resort (Demo),,36.425288,-117.133162,,
BEATTY_AIRPORT,Nye County Airport (Demo),,36.868446,-116.784582,,
BULLFROG,Bullfrog (Demo),,36.88108,-116.81797,,
STAGECOACH,Stagecoach Hotel & Casino (Demo),,36.915682,-116.751677,,
NADAV,North Ave / D Ave N (Demo),,36.914893,-116.76821,,
NANAA,North Ave / N A Ave (Demo),,36.914944,-116.761472,,
DADAN,Doing Ave / D Ave N (Demo),,36.909489,-116.768242,,
EMSI,E Main St / S Irving St (Demo),,36.905697,-116.76218,,
AMV,Amargosa Valley (Demo),,36.641496,-116.40094,,
```



Learning by doing… MTL for our tutorial use case!

CSV File

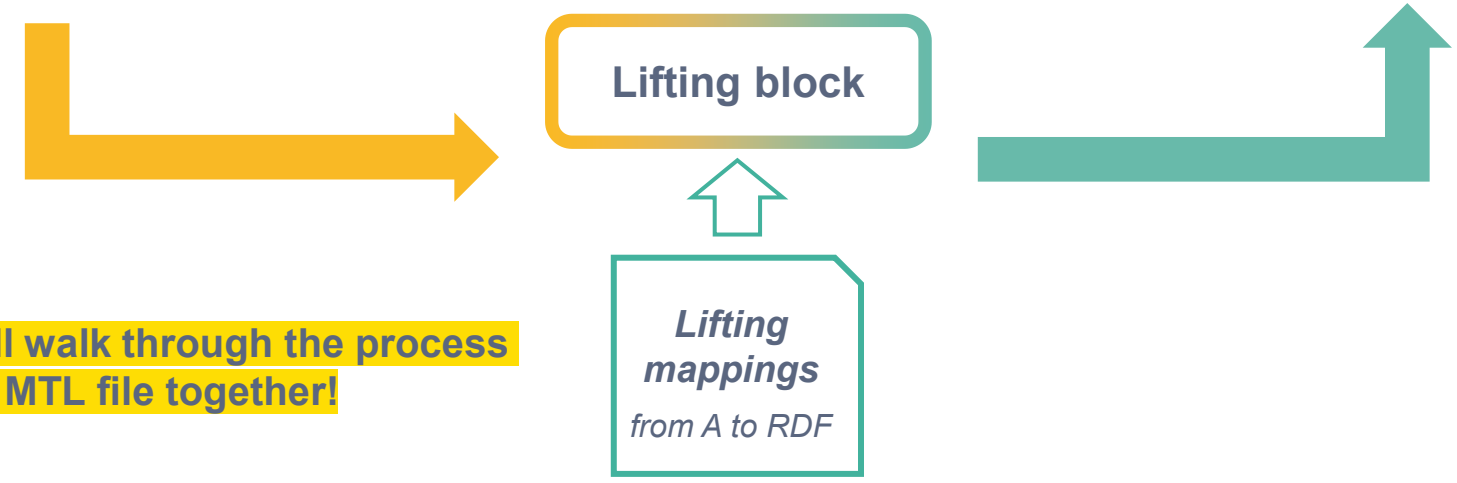
```
stop_id,stop_name,stop_desc,stop_lat,stop_lon,zone_id,stop_url
FUR_CREEK_RES,Furnace Creek Resort (Demo),,36.425288,-117.133162,,
BEATTY_AIRPORT,Nye County Airport (Demo),,36.868446,-116.784582,,
```

Input file: **e0/inbox/stops.txt**

Reference ontology representation

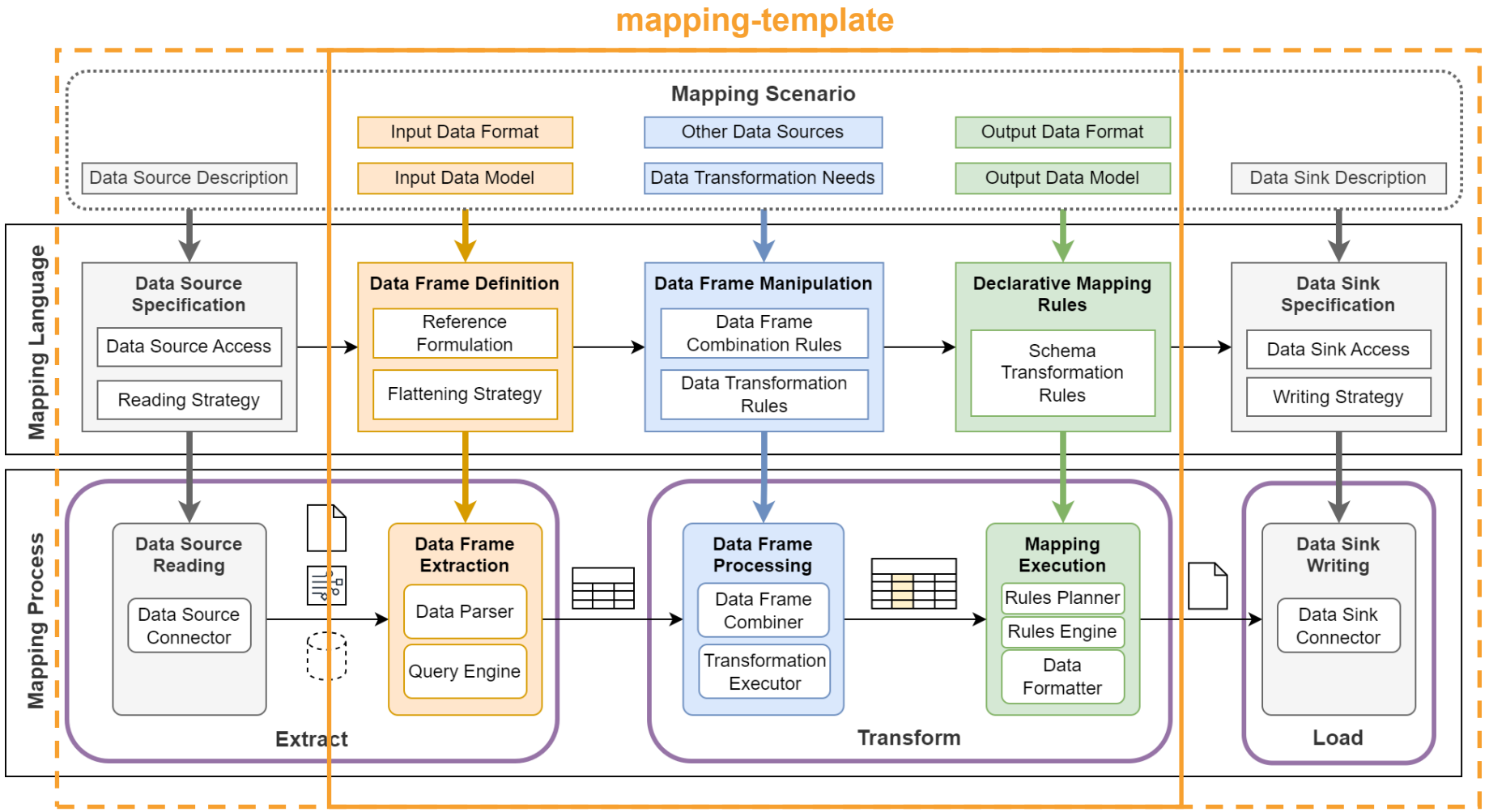
```
@prefix geo: <http://www.opengis.net/ont/geosparql#> .
@prefix tmjou: <https://w3id.org/mobility/transmodel/journeys#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix kg4di: <https://cefriel.github.io/kg4di#> .

kg4di:FUR_CREEK_RES a geo:Feature, tmjou:ScheduledStopPoint;
  geo:hasGeometry [
      geo:asWKT "Point(-117.133162 36.425288)"^^geo:wktLiteral
    ];
  rdfs:label "Furnace Creek Resort (Demo)" .
```

Expected output: **e0/outbox/e0-output.ttl**

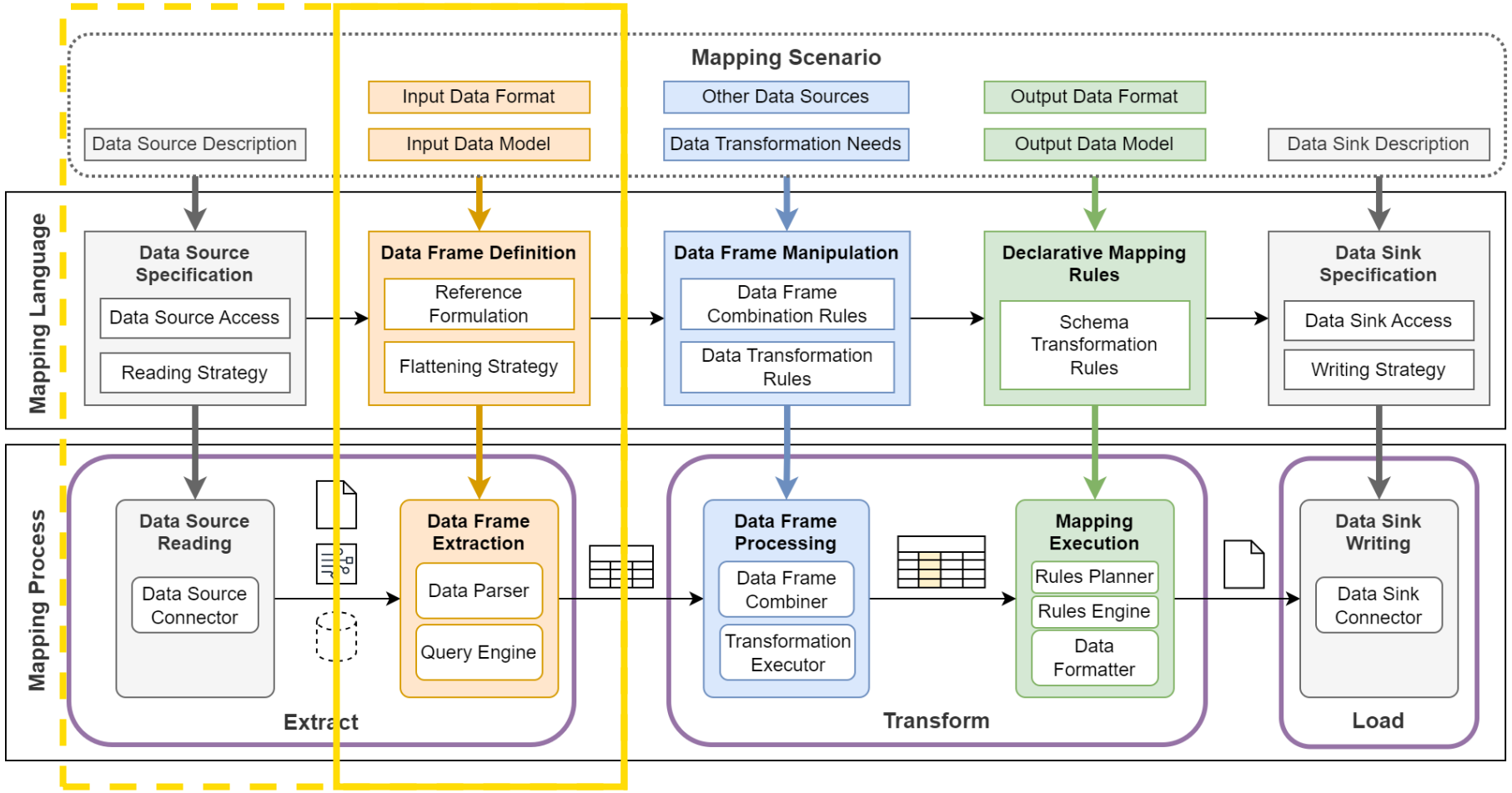


For e0, we will walk through the process of writing the MTL file together!

Mapping Workflow



Mapping Workflow



Readers in MTL

What is a Reader?

- A Reader is an abstraction that provides a **unified** way to **read**, **access** and **query** structured data sources and return results as a *DataFrame*, essentially a list of rows, where each row maps column names to string values.
- MTL supports Readers for the **CSV, JSON, XML, RDF and SQL** (Postgres and MySQL)
- In the mapping-template, Readers can be bound to specific variables by configuring the mapping processor (e.g., via CLI) or instantiated manually inside of an MTL mapping

```
stop_id,stop_name,stop_desc,stop_lat,stop_lon,zone_id,stop_url
FUR_CREEK_RES,Furnace Creek Resort (Demo),,36.425288,-117.133162,,
BEATTY_AIRPORT,Nye County Airport (Demo),,36.868446,-116.784582,,
BULLFROG,Bullfrog (Demo),,36.88108,-116.81797,,
STAGECOACH,Stagecoach Hotel & Casino (Demo),,36.915682,-116.751677,,
NADAV,North Ave / D Ave N (Demo),,36.914893,-116.76821,,
NANAA,North Ave / N A Ave (Demo),,36.914944,-116.761472,,
DADAN,Doing Ave / D Ave N (Demo),,36.909489,-116.768242,,
EMSI,E Main St / S Irving St (Demo),,36.905697,-116.76218,,
AMV,Amargosa Valley (Demo),,36.641496,-116.40094,,
```

```
#set ($stops = $reader getDataframe())
```

```
#set ($charReader = $functions.getCSVReaderFromFile("people.csv"))
#set ($episodeReader = $functions.getCSVReaderFromFile("episodes.csv"))
```

Dataframes in MTL

- A **dataframe** is the expected output of every **Reader**
- Think of it as a **table** in memory, rows and named columns, all as plain text.
- Every *Reader* (CSV, SQL, RDF, XML, JSON) produces the exact same dataframe shape, so your template logic never changes regardless of the source
- A dataframe is obtained by calling ***getDataframe(query)*** on a *Reader* that runs a query (SPARQL, SQL, Xpath, JsonPath) that returns results in a flat, tabular structure

```
stop_id,stop_name,stop_desc,stop_lat,stop_lon,zone_id,stop_url
FUR_CREEK_RES,Furnace Creek Resort (Demo),,36.425288,-117.133162,,
BEATTY_AIRPORT,Nye County Airport (Demo),,36.868446,-116.784582,,
BULLFROG,Bullfrog (Demo),,36.88108,-116.81797,,
STAGECOACH,Stagecoach Hotel & Casino (Demo),,36.915682,-116.751677,,
NADAV,North Ave / D Ave N (Demo),,36.914893,-116.76821,,
NANAA,North Ave / N A Ave (Demo),,36.914944,-116.761472,,
DADAN,Doing Ave / D Ave N (Demo),,36.909489,-116.768242,,
EMSI,E Main St / S Irving St (Demo),,36.905697,-116.76218,,
AMV,Amargosa Valley (Demo),,36.641496,-116.40094,,
```

```
#set ($stops = $reader.getDataframe())
```

Learning by doing… MTL for our tutorial use case!

Input file: **e0/inbox/stops.txt**

```
stop_id,stop_name,stop_desc,stop_lat,stop_lon,zone_id,stop_url
FUR_CREEK_RES,Furnace Creek Resort (Demo),,36.425288,-117.133162,,
BEATTY_AIRPORT,Nye County Airport (Demo),,36.868446,-116.784582,,
```

Expected output: **e0/outbox/e0-output.ttl**

```
@prefix geo: <http://www.opengis.net/ont/geosparql#> .
@prefix tmjou: <https://w3id.org/mobility/transmodel/journeys#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix kg4di: <https://cefriel.github.io/kg4di#> .

kg4di:FUR_CREEK_RES a geo:Feature, tmjou:ScheduledStopPoint;
  geo:hasGeometry [
      geo:asWKT "Point(-117.133162 36.425288)"^^geo:wktLiteral
    ];
  rdfs:label "Furnace Creek Resort (Demo)" .
```

Our Mapping

```
#set ($stops = $reader.getDataframe())
```

The expected RDF output depends on the data in the input file, so we need to read the input data using a **Reader** and obtain a **Dataframe**

- ***Where does this $reader come from?*** When an MTL mapping is run from either the CLI using the mapping-template or as part of a Chimera route using the appropriate component a $reader is instantiated by default with the provided input data
- ***Why aren't we providing a query to the getDataframe method***? A query is provided when we want to go from a hierarchical file structure to a flat, tabular structure. In this case the starting data is a CSV so we do not need to perform a flattening.

Learning by doing… MTL for our tutorial use case!

Input file: **e0/inbox/stops.txt**

```
stop_id,stop_name,stop_desc,stop_lat,stop_lon,zone_id,stop_url
FUR_CREEK_RES,Furnace Creek Resort (Demo),,36.425288,-117.133162,,
BEATTY_AIRPORT,Nye County Airport (Demo),,36.868446,-116.784582,,
```

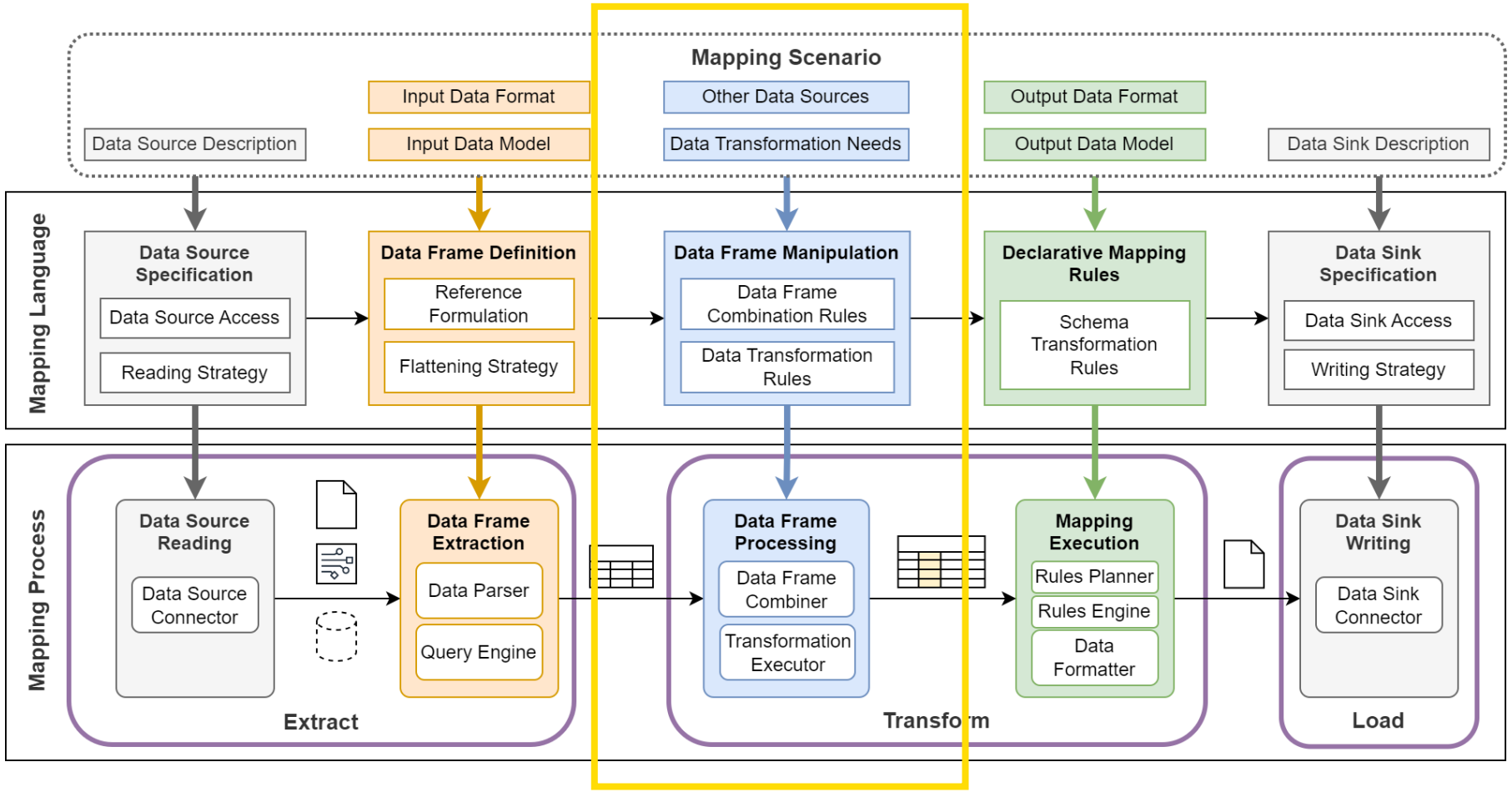
Our Mapping

```
#set ($stops = $reader.getDataframe())
```

The $stops dataframe

stop_id	stop_name	stop_lat	stop_lon
FUR_CREEK_RES	Furnace Creek Resort (Demo)	36.425288	117.133162
BEATTY_AIRPORT	Nye County Airport (Demo)	36.868446	-116.784582

Mapping Workflow



Data Frame Manipulation

- Once obtained one or more **dataframes,** these **can be manipulated** by using utility functions made available in MTL or through custom iterations

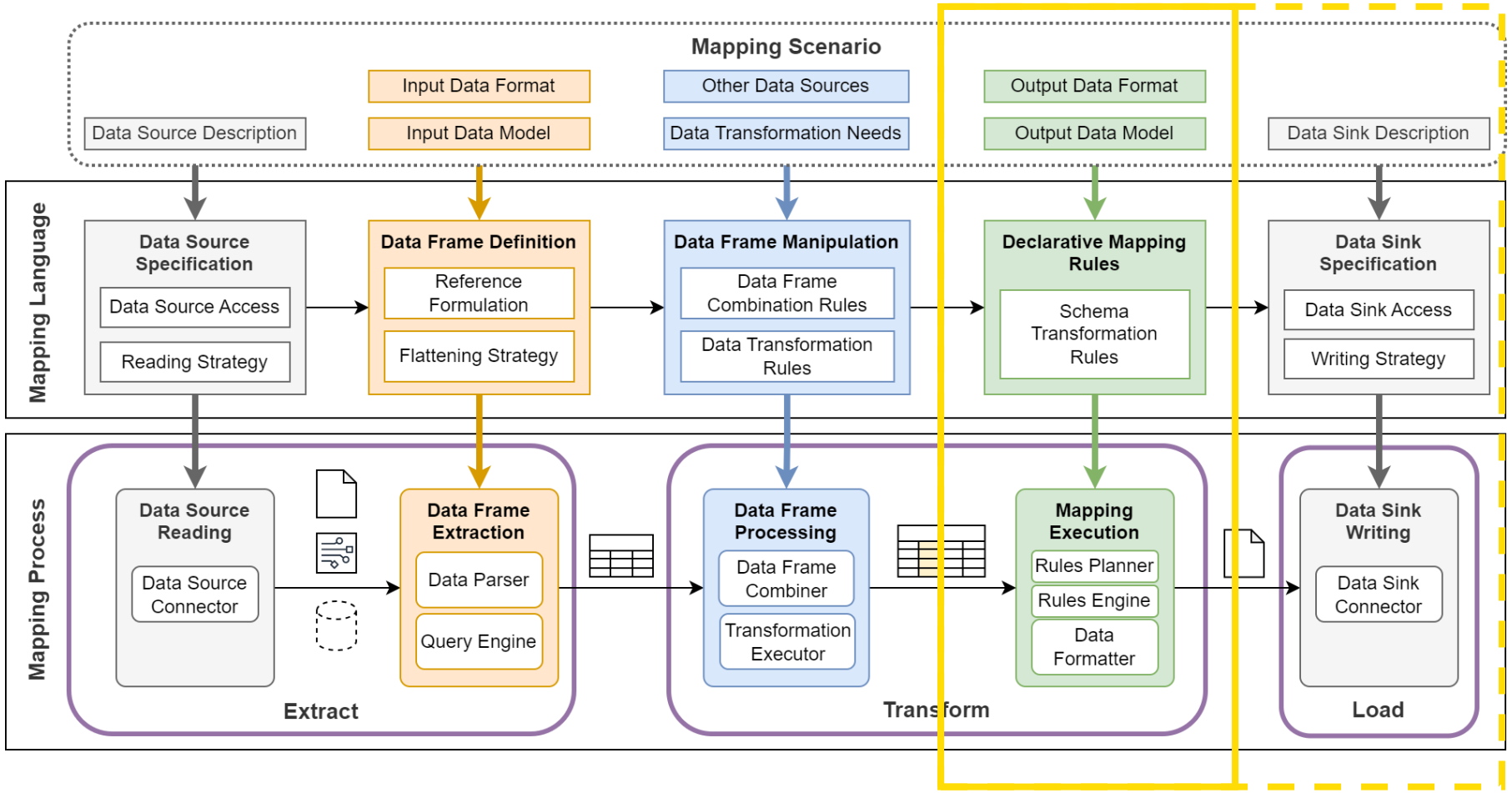
- **Data transformations** (e.g., formatting, string manipulation, etc.) and **join operations** can be applied to dataframes

- Functions can be registered and then called from within Velocity templates to **extend** the template language capabilities with **custom logic**

```
import com.cefriel.template.utils.TemplateFunctions;

public class GeoFunctions extends TemplateFunctions {
    public String getLat(String position) {
        if (position == null || !position.startsWith("Point(")) return null;
        String coords = position.substring(6, position.length() - 1);
        String[] parts = coords.split(" ");
        if (parts.length != 2) return null;
        return parts[1];
    }
```

```
$functions.getLat($stop.coord)
```

Mapping Workflow



Mapping Rules in MTL

- Mapping rules are defined using a **template-based approach**
- Constants portions should be combined with proper variables **considering the target output structure**
- **At runtime variables** are bound to input values within dataframes and the template is **rendered with actual values**
- In the mapping-template, by default the output of the template is written to a file

```
#foreach($ep in $episodes)
ex:episode_$ep.number a schema:Episode ;
    schema:title "$ep.title" .
#end
```

```
<?xml version="1.0" encoding="UTF-8"?>
#set ($bookDF = $reader.getDataframe())
<books>
#foreach($row in $bookDF)
  <book>
    <id>$row.book_id</id>
    <name>$row.title</name>
  </book>
#end
</books>
```

Formatter

- The MTL runtime supports formatters that can reformat output (e.g., convert RDF serialization), allowing for example templates written for Turtle to output other formats.
- This flexibility comes with some serialization conversion overhead.

```
@prefix geo: <http://www.opengis.net/ont/geosparql#> .
@prefix tmjou: <https://w3id.org/mobility/transmodel/journeys#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix kg4di: <https://cefriel.github.io/kg4di#> .

kg4di:FUR_CREEK_RES a geo:Feature, tmjou:ScheduledStopPoint;
  geo:hasGeometry [
      geo:asWKT "Point(-117.133162 36.425288)"^^geo:wktLiteral
    ];
  rdfs:label "Furnace Creek Resort (Demo)" .
```

Turtle

```
{
  "@context": {
    "geo": "http://www.opengis.net/ont/geosparql#",
    "tmjou": "https://w3id.org/mobility/transmodel/journeys#",
    "kg4di": "https://cefriel.github.io/kg4di#",
    "rdfs": "http://www.w3.org/2000/01/rdf-schema#"
  },
  "@id": "kg4di:FUR_CREEK_RES",
  "@type": [
    "geo:Feature",
    "tmjou:ScheduledStopPoint"
  ],
  "geo:hasGeometry": {
    "geo:asWKT": {
      "@value": "Point(-117.133162 36.425288)",
      "@type": "geo:wktLiteral"
    }
  },
  "rdfs:label": "Furnace Creek Resort (Demo)"
}
```

JSON-LD

```
<https://cefriel.github.io/kg4di#FUR_CREEK_RES> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.opengis.net/ont/geosparql#Feature> .
<https://cefriel.github.io/kg4di#FUR_CREEK_RES> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <https://w3id.org/mobility/transmodel/journeys#ScheduledStopPoint> .
<https://cefriel.github.io/kg4di#FUR_CREEK_RES> <http://www.opengis.net/ont/geosparql#hasGeometry> _:b0 .
_:b0 <http://www.opengis.net/ont/geosparql#asWKT> "Point(-117.133162 36.425288)"^^<http://www.opengis.net/ont/geosparql#wktLiteral> .
<https://cefriel.github.io/kg4di#FUR_CREEK_RES> <http://www.w3.org/2000/01/rdf-schema#label> "Furnace Creek Resort (Demo)" .
```

NT

Learning by doing… MTL for our tutorial use case!

Input file: **e0/inbox/stops.txt**

```
stop_id,stop_name,stop_desc,stop_lat,stop_lon,zone_id,stop_url
FUR_CREEK_RES,Furnace Creek Resort (Demo),,36.425288,-117.133162,,
BEATTY_AIRPORT,Nye County Airport (Demo),,36.868446,-116.784582,,
```

Expected output: **e0/outbox/e0-output.ttl**

```
@prefix geo: <http://www.opengis.net/ont/geosparql#> .
@prefix tmjou: <https://w3id.org/mobility/transmodel/journeys#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix kg4di: <https://cefriel.github.io/kg4di#> .

kg4di:FUR_CREEK_RES a geo:Feature, tmjou:ScheduledStopPoint;
  geo:hasGeometry [
      geo:asWKT "Point(-117.133162 36.425288)"^^geo:wktLiteral
    ];
  rdfs:label "Furnace Creek Resort (Demo)" .
```

Our Mapping

- We want to produce an RDF file, so let's start by adding the prefixes.
- As this is a template, these prefixes will be rendered in the exacts same way in the produced output

```
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
PREFIX kg4di: <https://cefriel.github.io/kg4di#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX tmjou: <https://w3id.org/mobility/transmodel/journeys#>
```

Learning by doing… MTL for our tutorial use case!

Input file: **e0/inbox/stops.txt**

```
stop_id,stop_name,stop_desc,stop_lat,stop_lon,zone_id,stop_url
FUR_CREEK_RES,Furnace Creek Resort (Demo),,36.425288,-117.133162,,
BEATTY_AIRPORT,Nye County Airport (Demo),,36.868446,-116.784582,,
```

Expected output: **e0/outbox/e0-output.ttl**

```
@prefix geo: <http://www.opengis.net/ont/geosparql#> .
@prefix tmjou: <https://w3id.org/mobility/transmodel/journeys#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix kg4di: <https://cefriel.github.io/kg4di#> .

kg4di:FUR_CREEK_RES a geo:Feature, tmjou:ScheduledStopPoint;
  geo:hasGeometry [
      geo:asWKT "Point(-117.133162 36.425288)"^^geo:wktLiteral
    ];
  rdfs:label "Furnace Creek Resort (Demo)" .
```

Our Mapping

- Since we know the desired output shape, let's use the power of MTL's template approach!
- We **copy the intended output into the MTL mapping**, then generalize it to fit our input data

```
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
PREFIX kg4di: <https://cefriel.github.io/kg4di#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX tmjou: <https://w3id.org/mobility/transmodel/journeys#>

#set ($stops = $reader.getDataframe())

kg4di:FUR_CREEK_RES a geo:Feature, tmjou:ScheduledStopPoint;
  geo:hasGeometry [
   geo:asWKT "Point(-117.133162 36.425288)"^^geo:wktLiteral ];
   rdfs:label "Furnace Creek Resort (Demo)" .
```

Learning by doing… MTL for our tutorial use case!

Input file: **e0/inbox/stops.txt**

```
stop_id,stop_name,stop_desc,stop_lat,stop_lon,zone_id,stop_url
FUR_CREEK_RES,Furnace Creek Resort (Demo),,36.425288,-117.133162,,
BEATTY_AIRPORT,Nye County Airport (Demo),,36.868446,-116.784582,,
```

Expected output: **e0/outbox/e0-output.ttl**

```
@prefix geo: <http://www.opengis.net/ont/geosparql#> .
@prefix tmjou: <https://w3id.org/mobility/transmodel/journeys#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix kg4di: <https://cefriel.github.io/kg4di#> .

kg4di:FUR_CREEK_RES a geo:Feature, tmjou:ScheduledStopPoint;
  geo:hasGeometry [
      geo:asWKT "Point(-117.133162 36.425288)"^^geo:wktLiteral
    ];
  rdfs:label "Furnace Creek Resort (Demo)" .
```

Our Mapping

- We need to handle all the Dataframe rows from the input csv data, as each row should produce a different set of RDF triples.
- We start by **iterating over the Dataframe using a foreach loop**

```
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
PREFIX kg4di: <https://cefriel.github.io/kg4di#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX tmjou: <https://w3id.org/mobility/transmodel/journeys#>

#set ($stops = $reader.getDataframe())

#foreach ($stop in $stops)
kg4di:FUR_CREEK_RES a geo:Feature, tmjou:ScheduledStopPoint;
  geo:hasGeometry [
   geo:asWKT "Point(-117.133162 36.425288)"^^geo:wktLiteral ];
   rdfs:label "Furnace Creek Resort (Demo)" .
#end
```

Learning by doing… MTL for our tutorial use case!

Input file: **e0/inbox/stops.txt**

```
stop_id,stop_name,stop_desc,stop_lat,stop_lon,zone_id,stop_url
FUR_CREEK_RES,Furnace Creek Resort (Demo),,36.425288,-117.133162,,
BEATTY_AIRPORT,Nye County Airport (Demo),,36.868446,-116.784582,,
```

Expected output: **e0/outbox/e0-output.ttl**

```
@prefix geo: <http://www.opengis.net/ont/geosparql#> .
@prefix tmjou: <https://w3id.org/mobility/transmodel/journeys#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix kg4di: <https://cefriel.github.io/kg4di#> .

kg4di:FUR_CREEK_RES a geo:Feature, tmjou:ScheduledStopPoint;
  geo:hasGeometry [
      geo:asWKT "Point(-117.133162 36.425288)"^^geo:wktLiteral
    ];
  rdfs:label "Furnace Creek Resort (Demo)" .
```

Our Mapping

- All that's left is to **insert variables into the output** wherever values depend on the input data

```
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
PREFIX kg4di: <https://cefriel.github.io/kg4di#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX tmjou: <https://w3id.org/mobility/transmodel/journeys#>

#set ($stops = $reader.getDataframe())

#foreach ($stop in $stops)
 kg4di:$stop.stop_id
   rdf:type geo:Feature ;
   rdf:type tmjou:ScheduledStopPoint ;
   rdfs:label "$stop.stop_name" ;
   geo:hasGeometry [
   geo:asWKT "Point($stop.stop_lon $stop.stop_lat)"^^geo:wktLiteral ] .
#end
```

Exercise 0: Lifting Exercise

- [*Homework*] You can try to rewrite the proposed **lifting MTL mapping and execute it** by following the instructions in **Exercise 0** of the **repository**

Running Exercise 0

We want to lift the CSV lower ./inbox/stops.txt to obtain the the RDF KG contained in ./outbox/e0-output.ttl.

To do this, edit and complete the MTL mapping in ./lifting.vm.

Using Docker Compose

```
cd camel-routes-exercises/e0
docker compose up
```

Docker Command

```
# from root directory
docker run -v ./camel-routes-exercises/e0:/app cefriel/chimera:kg4di
```

JBang Command

```
cd camel-routes-exercises/e0
jbang camel@apache/camel run camel-routes/*.yaml --dep=mvn:com.cefriel:camel-chimera-mapping-template:4.6.0
```

https://github.com/cefriel/kg4di#exercise-0--rdf-lifting

PRACTICAL SESSION 
EXERCISE 1: LOWERING IN MTL

Exercise 1: Write MTL Mapping Rules!

Reference ontology
representation

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix transit: <http://vocab.org/transit/terms/>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix wgs84_pos: <http://www.w3.org/2003/01/geo/wgs84_pos#>.
@prefix ex: <http://airport.example.com/>.


ex:6523 rdf:type transit:Stop ;
  transit:route "25"^^xsd:int ;
  wgs84_pos:lat "50.901389" ;
  wgs84_pos:long "4.484444" .
```

CSV File

```
stop_id,route,latitude,longitude
http://airport.example.com/1847,12,51.219447,4.402464
http://airport.example.com/6523,25,50.901389,4.484444
http://airport.example.com/9031,7,50.850346,4.351721
http://airport.example.com/7712,18,51.054342,3.717424
```

Input file: **e1/inbox/e1-input.ttl**

Your task is to complete the MTL in: e1/inbox/lowering.vm

Lowering block

Lowering mapping
from RDF to CSV

Expected output: **e1/outbox/e1-output.csv**

Check instructions in the e1/README.md to test your MTL mapping rules

Break (30 min)
Back at 11

Knowledge Graphs for Data Interoperability with Chimera (KG4DI)

Discover how Knowledge Graphs can solve data interoperability challenges and then get hands-on with Chimera to build an interactive dashboard from heterogeneous data sources

cefriel.github.io/kg4di/

Repository

Cefriel
POLITECNICO DI MILANO

ESWC26

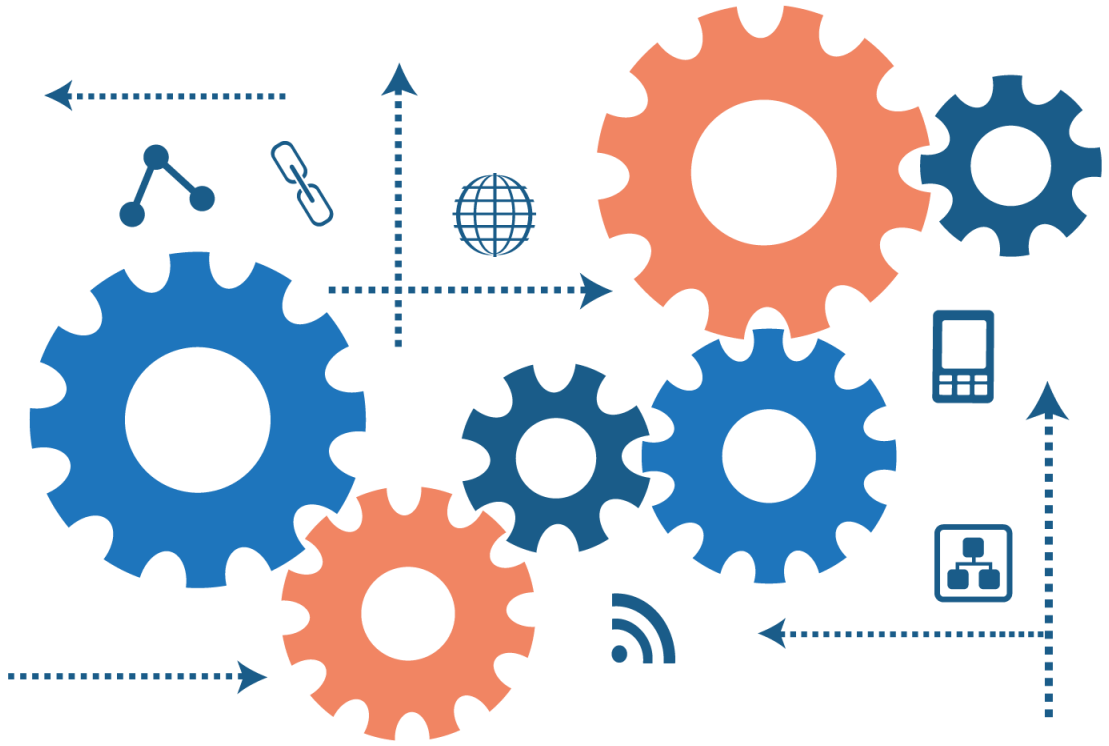
THE CHIMERA FRAMEWORK

PART 3

So far so good…

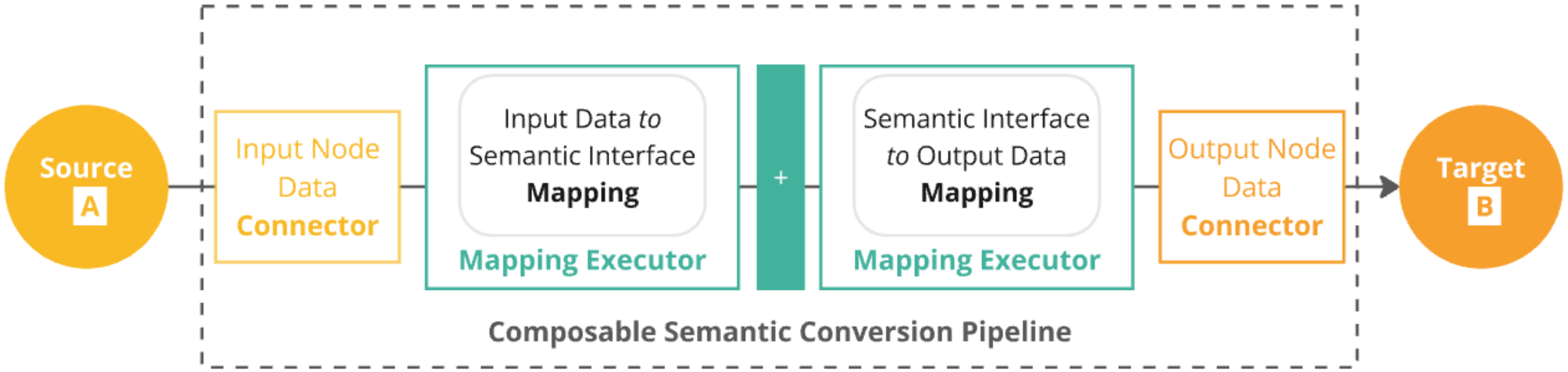
- Knowledge graphs can support semantic data conversion between heterogeneous data formats via an **any-to-one mapping approach**
- An intermediate KG representation **enables data harmonisation and facilitates data fusion**
- Different mapping approaches can be used to implement **declarative lifting and lowering mapping rules**. MTL is based on templates and provides the possibility of specifying both lifting and lowering mapping rules.

Question 2: **How to support mediated data exchanges between heterogenous information systems?**



Composable Semantic Conversion Pipelines

Our objective was a set of modular, composable and configurable components to **define semantic converters for heterogeneous use cases and integration requirements** as composable semantic conversion pipelines



Towards Chimera

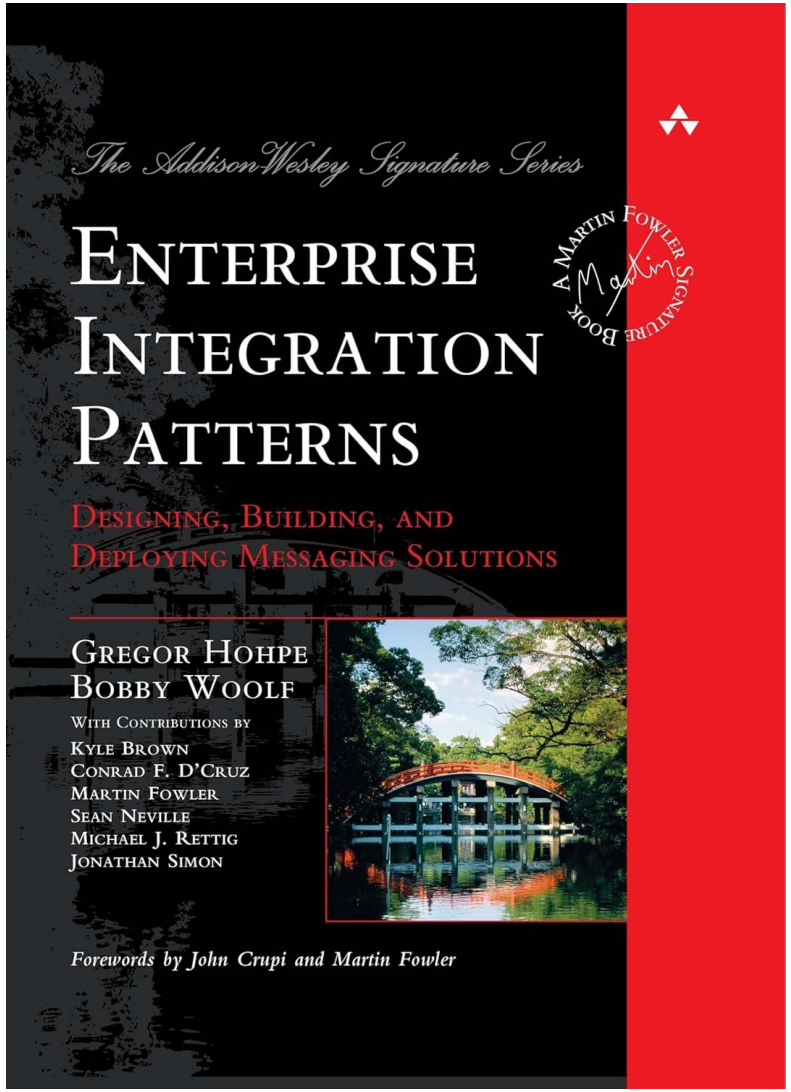
Our **design principles**:

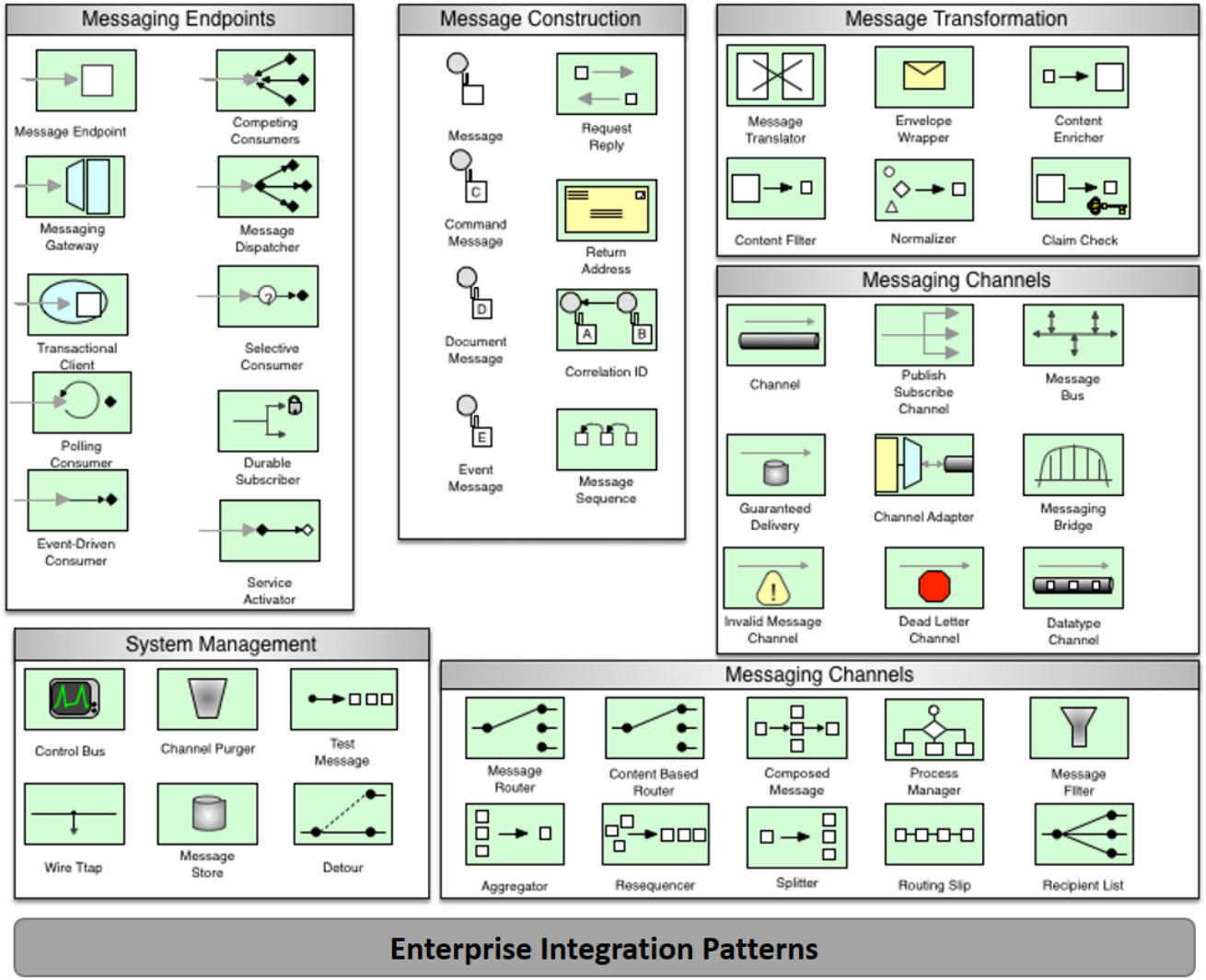
- consider already studied **patterns** and reuse as much as possible **existing solutions**
- provide modular/reusable components to enable **low-code approaches**

For the purpose, we started:

- assessing open-source frameworks for data integration
- defining KG-based operations for semantic conversion

Enterprise Integration Patterns





What is Apache Camel?

Apache Camel is an **open-source framework for message-oriented middleware** with a rule-based routing **used for system integration and creation of ETL pipelines** (Extract, transform and load).

- **EIP:** Camel implements all the Enterprise Integration Pattern that can be used in pipelines (enrich, aggregate, wiretap….).
- **Configurability:** Camel enables low-code configuration of integration pipelines by combining existing components
- **Extensibility:** new components can be added to the list of the >300 already supported ones.

How Apache Camel Works?

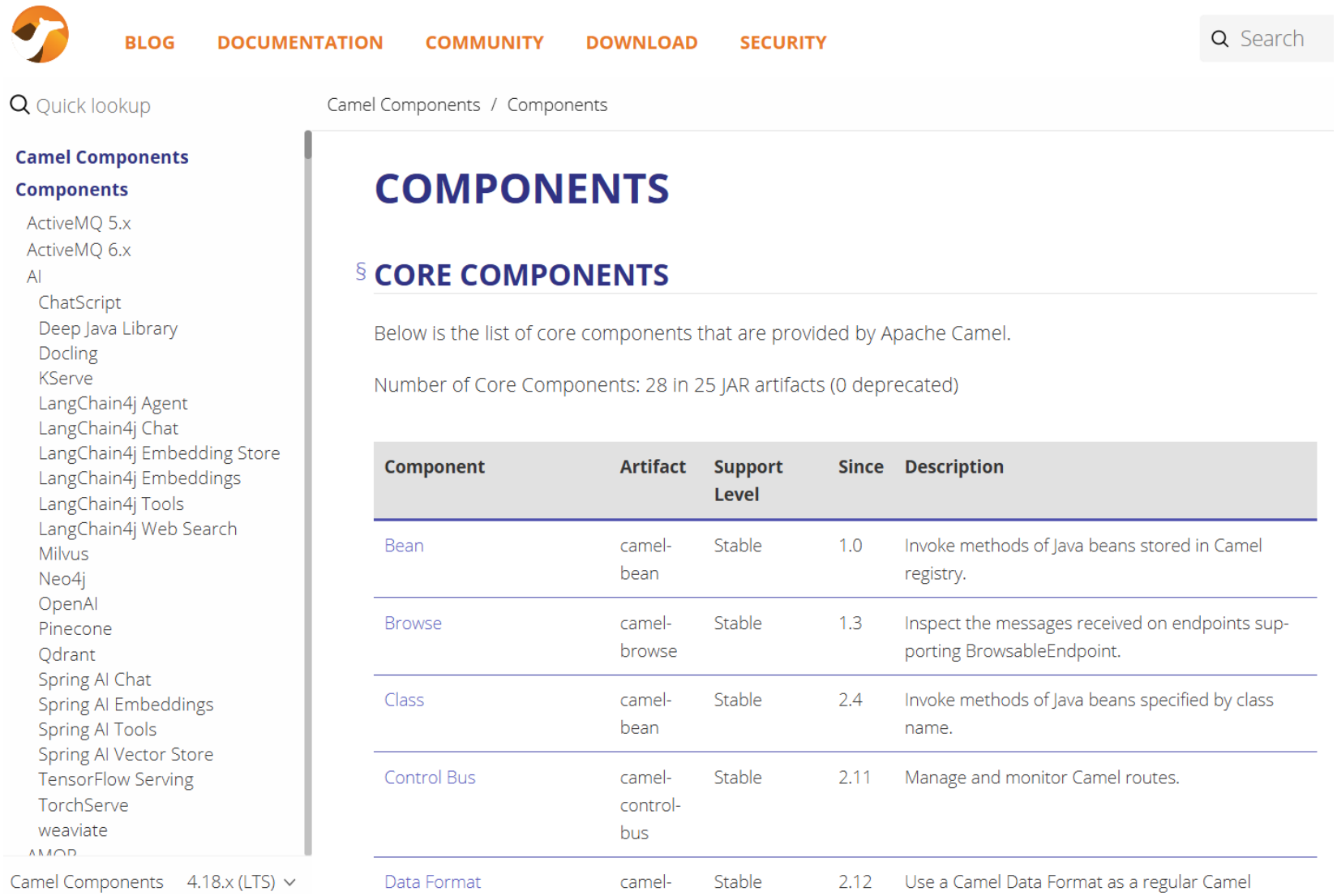
To understand how Apache Camel works and what is an ETL pipeline**, imagine it as a big assembly line**.

- An assembly line in Camel is the camel pipeline and it's called **"route".** You can concatenate more route to obtain a more complex and detailed result.
- The product is called **exchange.** You can imagine the exchange as a box, where the product inside is called **"body"** and all the labels attached to the box are called **"headers".**
- Each machine that composes the line is a **component.**
- When the assembly line is finished, **the exchange could be sent to another line or returned to the caller.**

Apache Camel Components

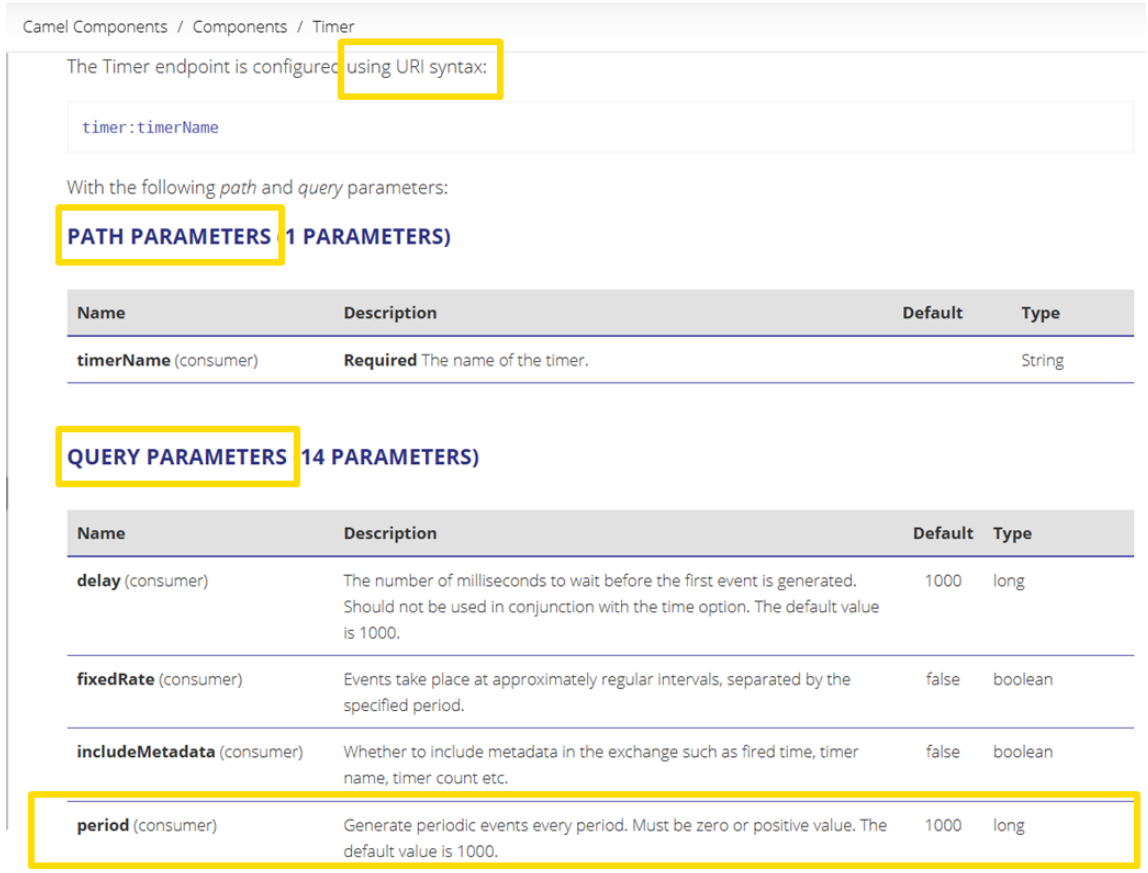
Component

- A pluggable adapter for a specific technology (HTTP, MQTT, FTP, reading from the filesystem…)
- A specific component in a route is used by configuring its parameters



https://camel.apache.org/components/4.18.x/index.html

Apache Camel Pipeline Example



Camel Components / Components / Timer

The Timer endpoint is configured using URI syntax:

```
timer:timerName
```

With the following *path* and *query* parameters:

PATH PARAMETERS (1 PARAMETERS)

Name	Description	Default	Type
timerName (consumer)	**Required** The name of the timer.		String

QUERY PARAMETERS (14 PARAMETERS)

Name	Description	Default	Type
delay (consumer)	The number of milliseconds to wait before the first event is generated. Should not be used in conjunction with the time option. The default value is 1000.	1000	long
fixedRate (consumer)	Events take place at approximately regular intervals, separated by the specified period.	false	boolean
includeMetadata (consumer)	Whether to include metadata in the exchange such as fired time, timer name, timer count etc.	false	boolean
period (consumer)	Generate periodic events every period. Must be zero or positive value. The default value is 1000.	1000	long

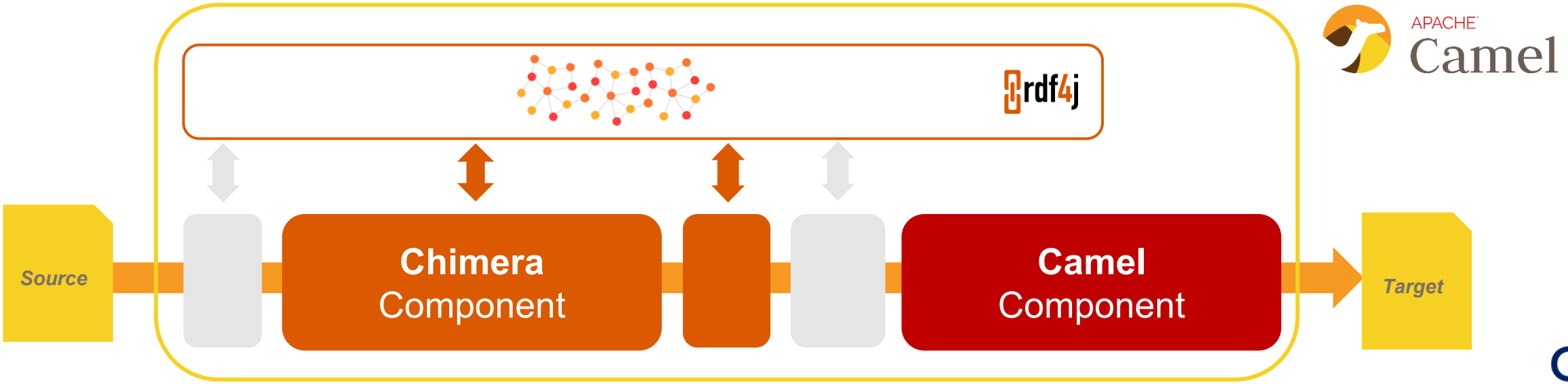
- **Example:**
  - Pipeline executed every 5 seconds using the Timer Component
  - Sets a constant string as body of the exchange
  - Logs the body of the exchange

```
- route:
    id: "hello-world-route"
    from:
        uri: "timer:hello?period=5000"
        steps:
            - setBody:
                constant: "Hello, World!"
            - log:
                message: "${body}"
```

Chimera: Composable Semantic Data Transformation

Chimera is a **modular** and **configurable solution** for the Apache Camel integration framework to minimise the effort required to specify and configure a **semantic data transformation pipeline**

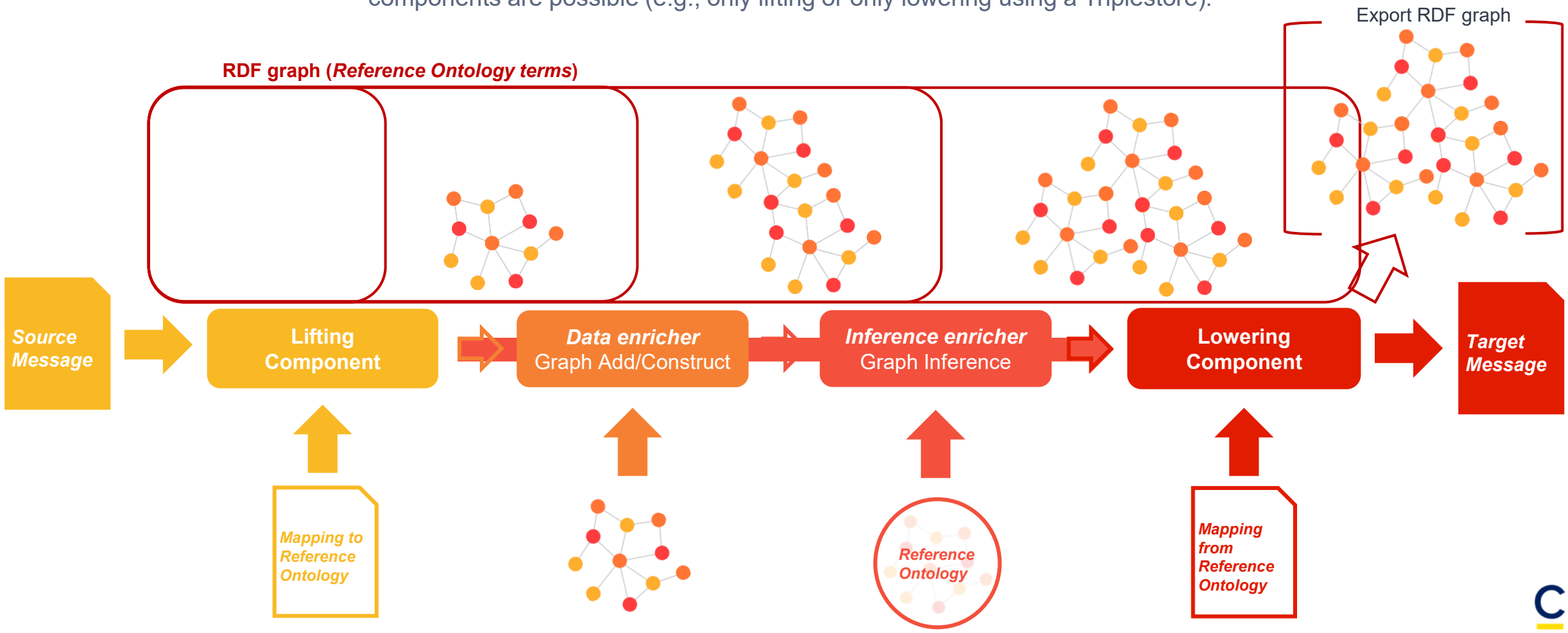
- **Chimera components** implementing (i) *RDF graph operations, (ii) Lifting,* and *(iii) Lowering*
- Production-ready **Camel components** can be used to implement *Enterprise Integration Patterns* and *data sources/sinks integration*
- *Integration with existing ETL pipelines* and additional **custom components** (e.g., pre-processing) can be easily performed



Chimera: https://github.com/cefriel/chimera

Chimera: Semantic Converter Basic Pipeline

A basic pipeline for a Semantic Converter using Chimera is presented. Different combinations of components are possible (e.g., only lifting or only lowering using a Triplestore).



What is a Chimera RDFGraph?

- An RDFGraph is the **graph interface flowing through Chimera pipelines**
- A wrapper around **RDF4J repositories** for managing RDF triples
- Think of it as the **object to access/modify the knowledge graph** during execution

- Link to documentation: https://cefriel.github.io/chimera/rdf-graphs/

RDFGraph Types

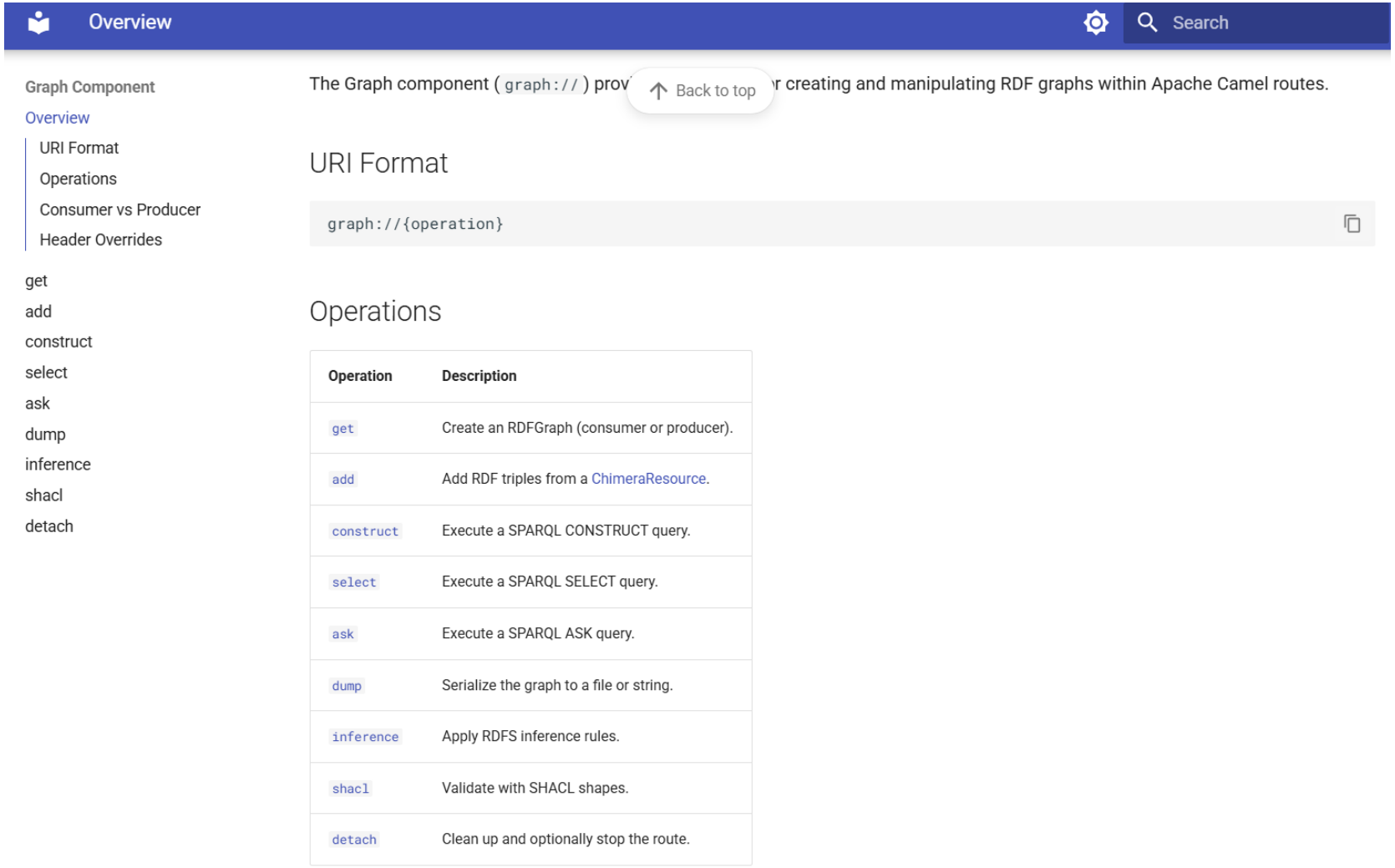
Type	Use Case
MemoryRDFGraph	Default; in-memory, temporary data for single exchange
NativeRDFGraph	Persistent on-disk storage with RDF4J's native index
HTTPRDFGraph	Connect to remote RDF4J Server
SPARQLEndpointGraph	Query any standard SPARQL endpoint (Virtuoso, Fuseki, etc.)
InferenceRDFGraph	In-memory or persistent store with automatic RDFS inference

Chimera Graph Component

The **Chimera Camel Component** defines two TypeConverter and different Camel Endpoints.

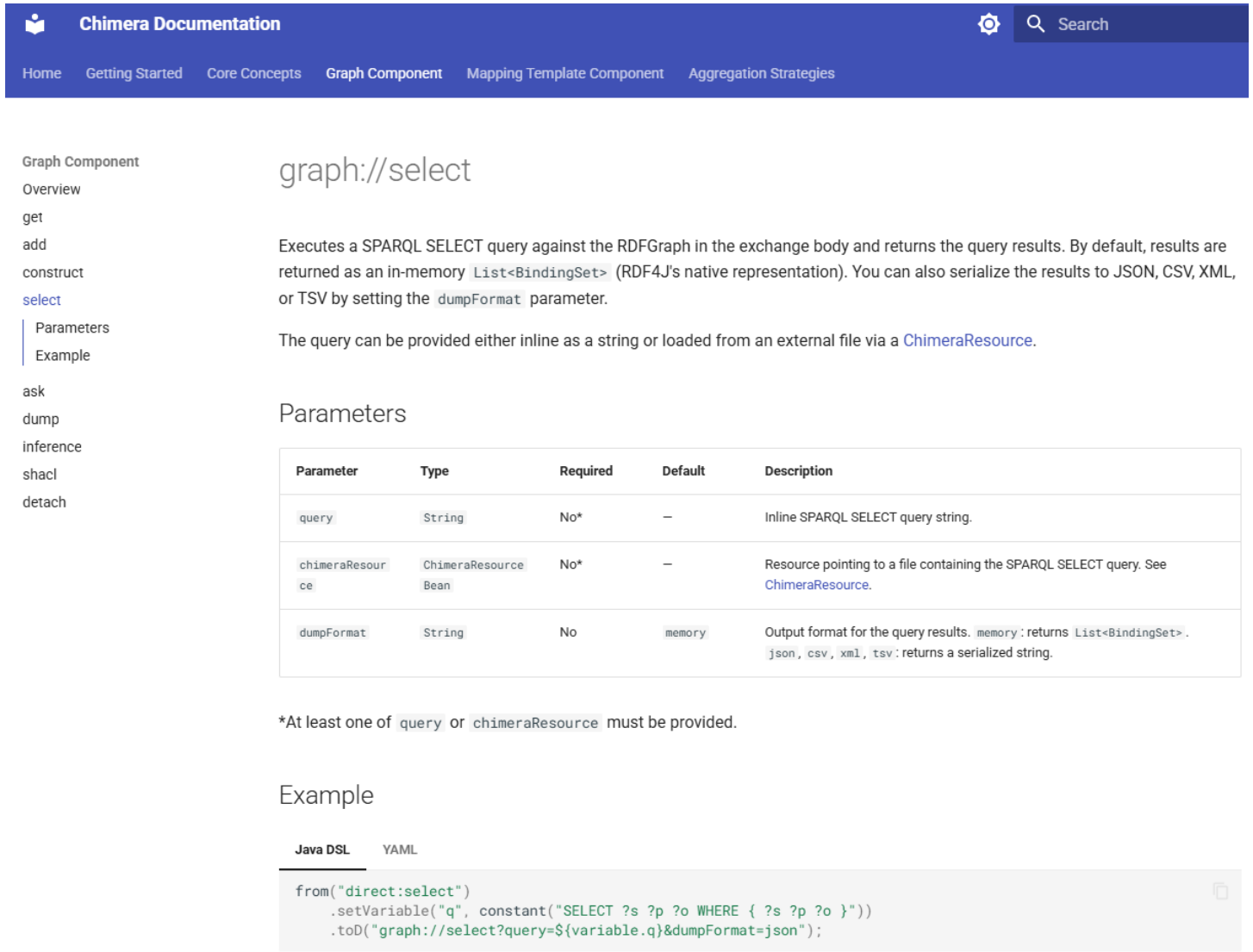
RDF Parser	Parse from RDF serialisation
Graph **Get**	**Get** an RDF graph (in-memory, on disk or connecting to a Triplestore)
Graph **Add**	**Parse** RDF triples and enrich the RDF graph
Graph **Construct**	Execute a SPARQL **Construct** on the RDF graph
Graph **Query**	Execute a SPARQL **Select** on the RDF graph
RDF Serializer	Serialize to RDF
Graph **Inference**	Enable **inference** on the RDF graph
Graph **Detach**	**Detach** the RDF graph deleting all/part of the triples
Graph **Dump**	**Dump** the RDF graph choosing a serialization format
Graph **Validate**	**Validate** the RDF graph using SHACL shapes

Chimera Documentation



https://cefriel.github.io/chimera/

Chimera Documentation



Example

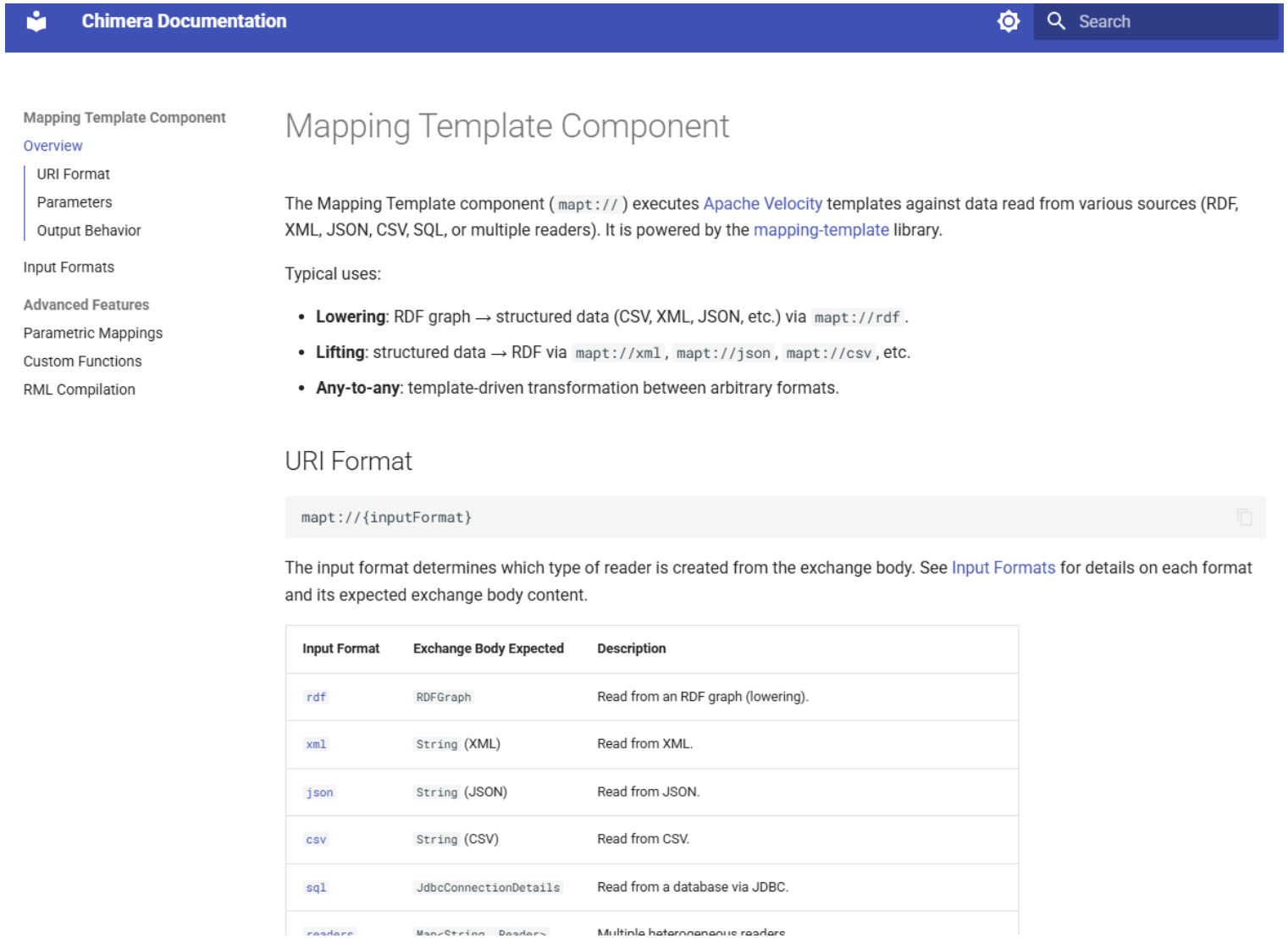
Java DSL YAML

```yaml
- route:
    from: "direct:select"
    steps:
      - setVariable:
          name: q
          constant: "SELECT ?s ?p ?o WHERE { ?s ?p ?o }"
      - toD: "graph://select?query=${variable.q}&dumpFormat=json"
```

https://cefriel.github.io/chimera/

Chimera Mapping Template Component

Chimera Documentation



https://cefriel.github.io/chimera/

What is a Chimera Resource?

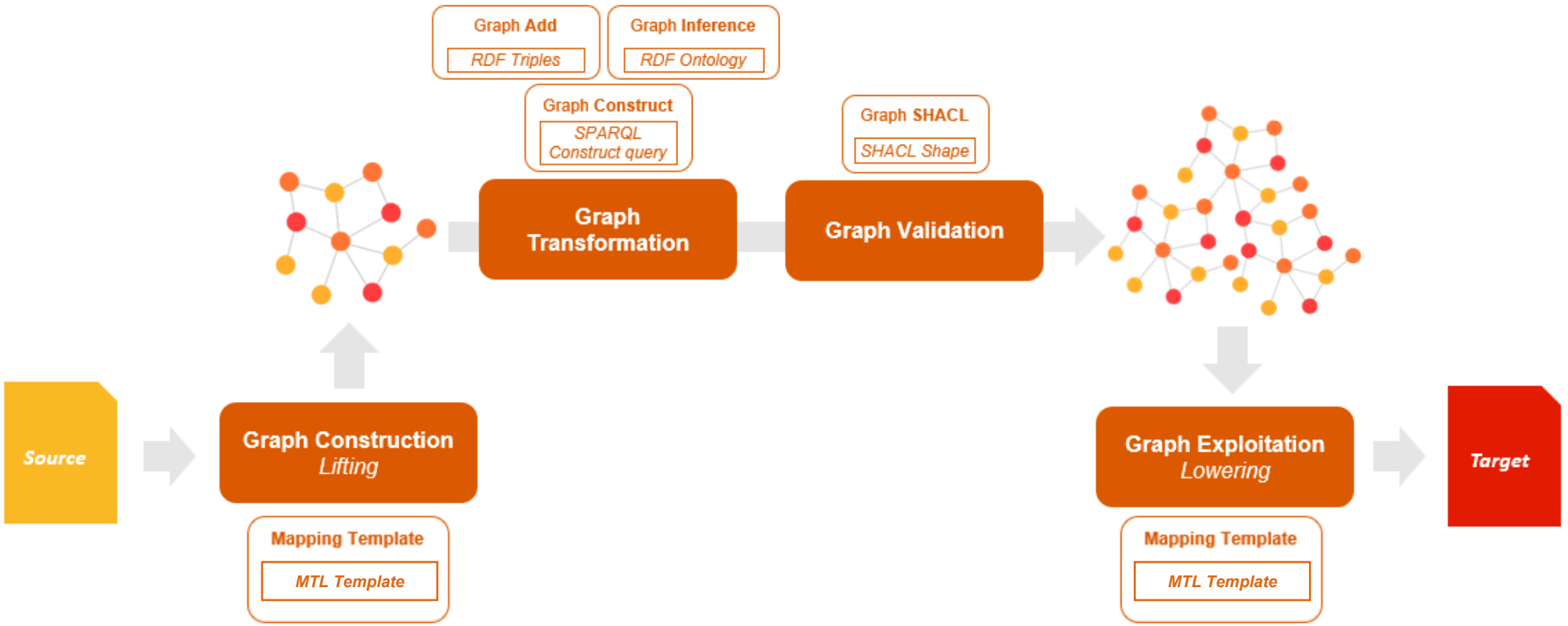
- Uniform wrapper for **referencing external data** for Chimera operations
- Encapsulates **URL** location, **format**, and optional authentication
- Used for **referencing files** containing **SPARQL queries**, **ontologies**, **MTL mappings** and more
- Link to documentation: https://cefriel.github.io/chimera/chimera-resources/

Resource Source Types

Prefix	Source
file://	Local filesystem
http:// / https://	Remote HTTP endpoint
classpath://	Java classpath (embedded rescs)
header://	Camel message header
property://	Camel exchange property
variable://	Camel exchange variable

```
- beans:
    - name: lowering
      type: com.cefriel.util.ChimeraResourceBean
      properties:
        url: "file://./mappings/lowering.vm"
        serializationFormat: "vtl"
```

Composing Chimera Components

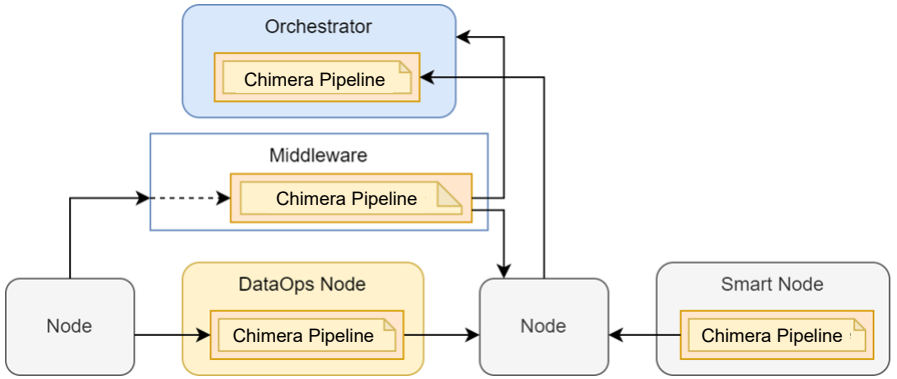


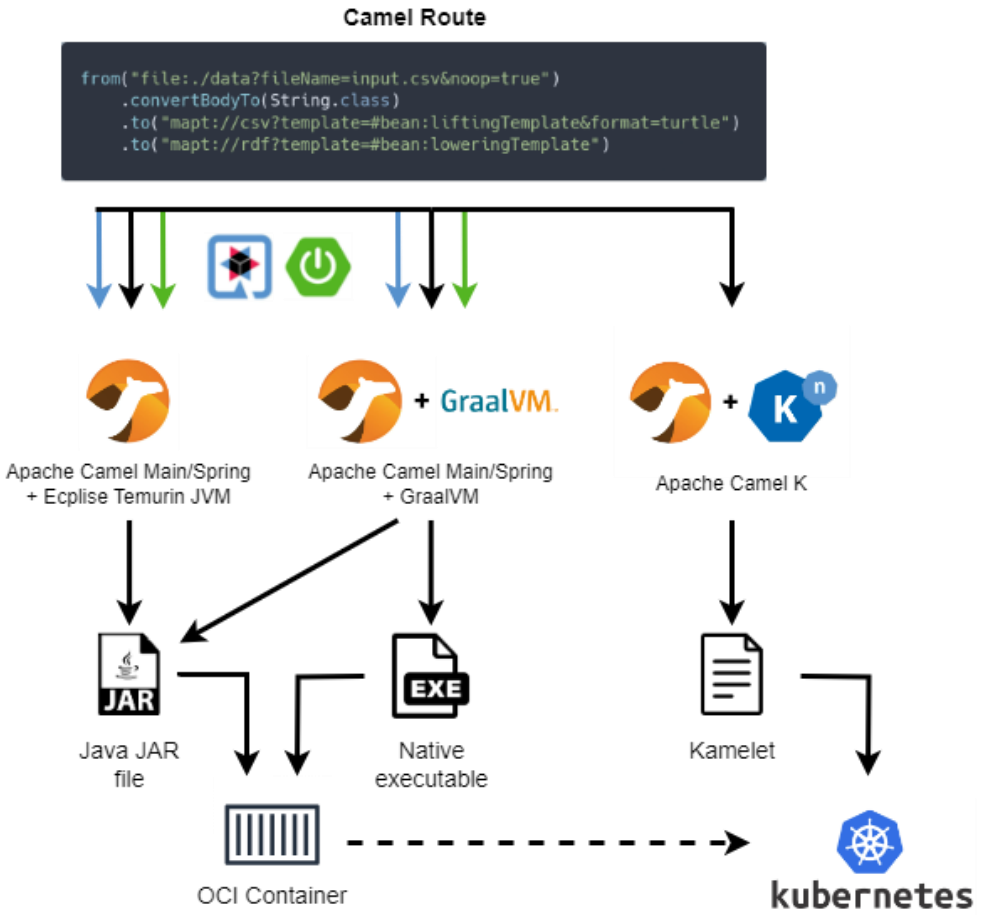
Chimera Deployment Templates

Chimera supports **different strategies for deploying a pipeline**:

- Embedded in a specific node (source/target/orchestrator)
- Dedicated mediation node (mediation service)
- Embedded in the middleware/network layer

The same Chimera pipeline can be **deployed in different environments** by reusing the made available templates.

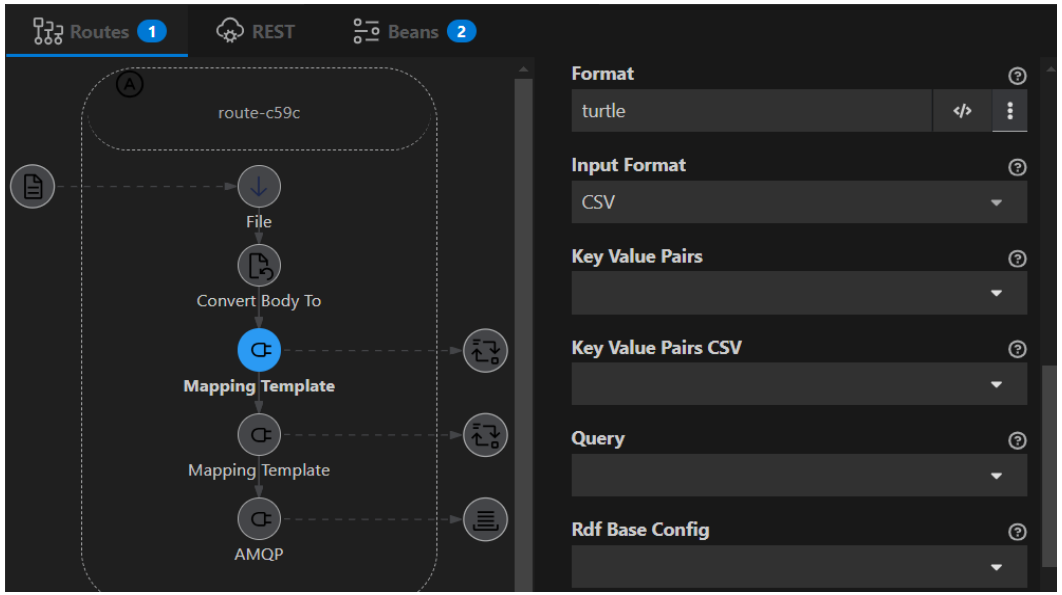




Repository https://github.com/cefriel/chimera-deployment-templates
Comparison of alternatives in D3.2 at https://www.smart-edge.eu/deliverables/

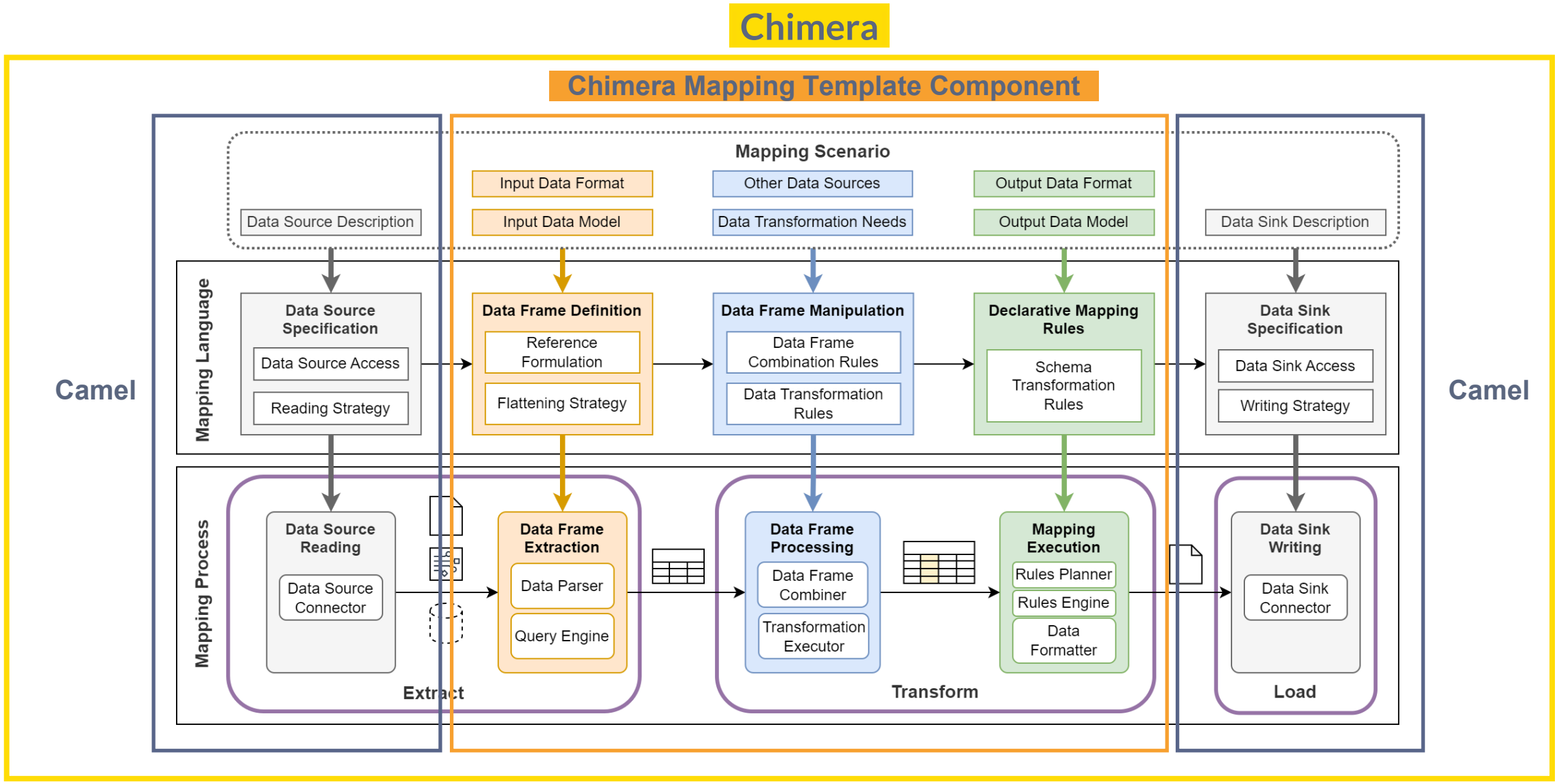
Low-code Chimera Configuration

- **Domain-Specific Languages (DSL),** such as XML, Spring XML, and YAML, to declaratively specify a pipeline. Modifications to a pipeline changing only the declaration files, without the need to rebuild.
- Extension of Apache Camel Karavan to support the Chimera components and provide a **drag-and-drop visual editor**
- Pipelines can be defined as **Kamelets** and **reused across projects**.



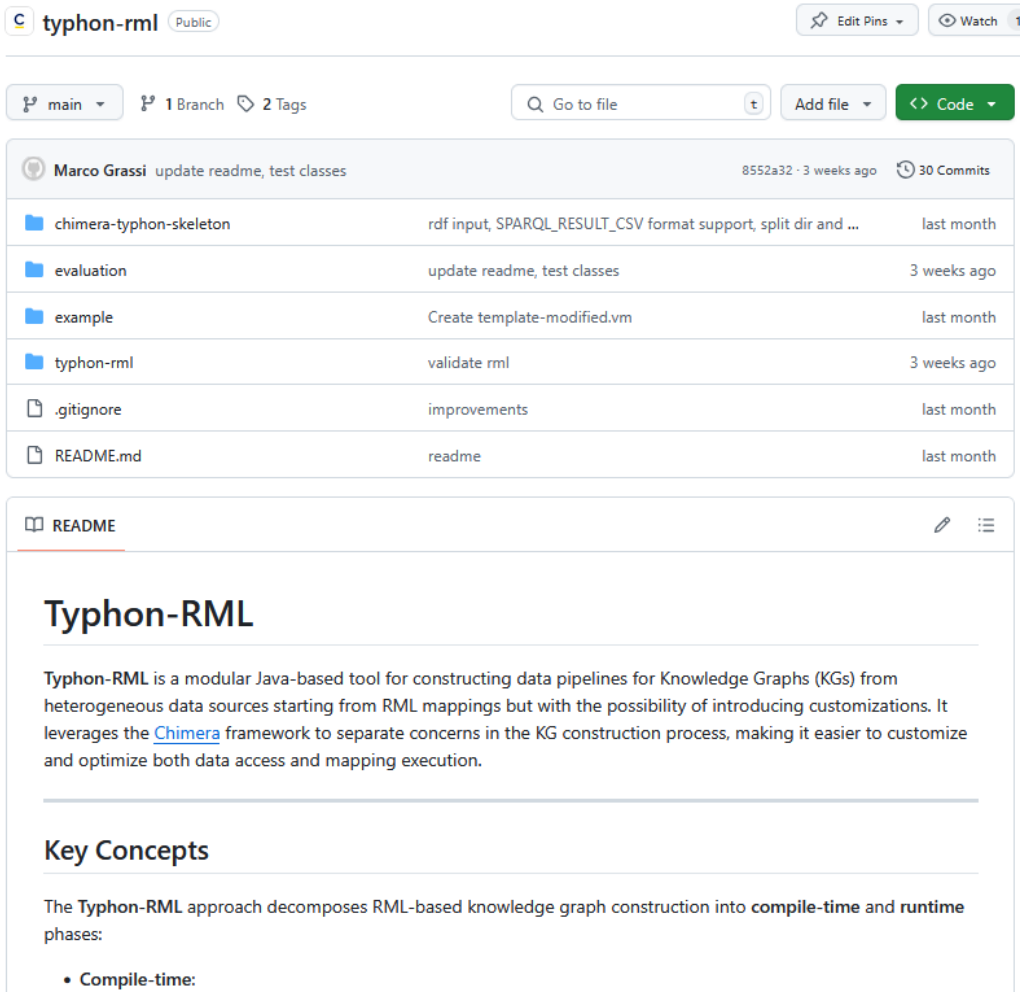

```yaml
- route:
    id: route-c59c
    nodePrefixId: route-e88
    from:
      id: from-41c6
      uri: file
      parameters:
        directoryName: ./data
        fileName: input.csv
        noop: true
      steps:
        - convertBodyTo:
            id: convertBodyTo-69b0
            type: String
        - to:
            id: to-5f43
            uri: mapt
            parameters:
              inputFormat: CSV
              template: "#bean:lifting"
              format: turtle
        - to:
            id: to-4517
            uri: mapt
            parameters:
              inputFormat: rdf
              template: "#bean:lowering"
        - to:
            id: to-d5ae
            uri: amqp
            parameters:
              destinationName: myQueue
- beans:
    - name: lifting
      type: com.cefriel.util.ChimeraResourceBean
      properties:
        serializationFormat: vtl
        url: file://./data/lift.vm
    - name: lowering
      type: com.cefriel.util.ChimeraResourceBean
      properties:
        serializationFormat: vtl
        url: file://./data/lower.vm
```

Chimera and the Mapping Workflow



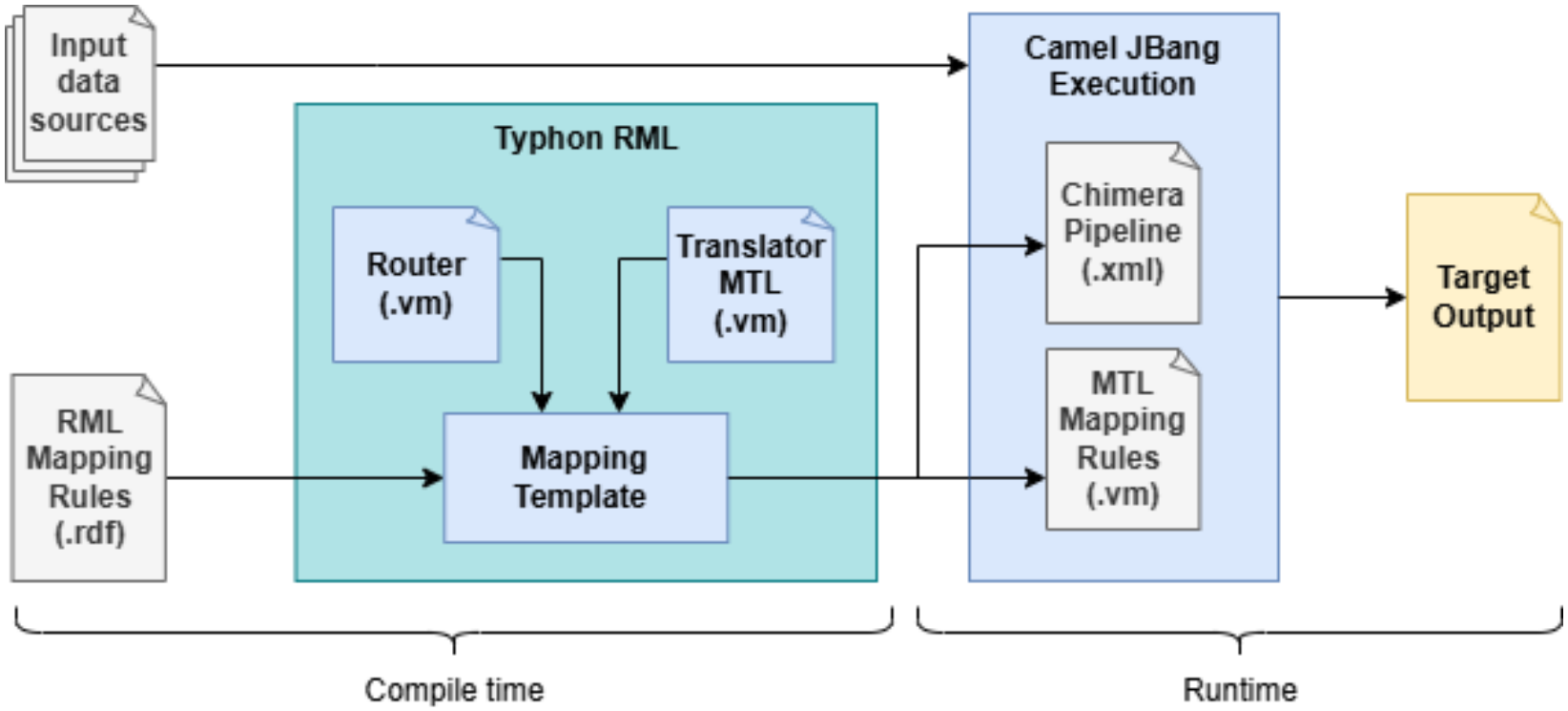
Typhon-RML

- Open-source **Java tool for constructing data pipelines for Knowledge Graphs** (KGs) from heterogeneous data sources
- **Uses RML mappings as inputs** and produces two distinct files, an **Apache Camel route.xml file** to handle data source reading and writing and a **template.vm MTL with mapping rules** equivalent to the ones expressed in the input RML.
- It **leverages the Chimera framework to separate concerns in the KG construction process**, making it easier to customize and optimize both data access and mapping execution.



Typhon-RML: https://github.com/cefriel/typhon-rml

Typhon-RML



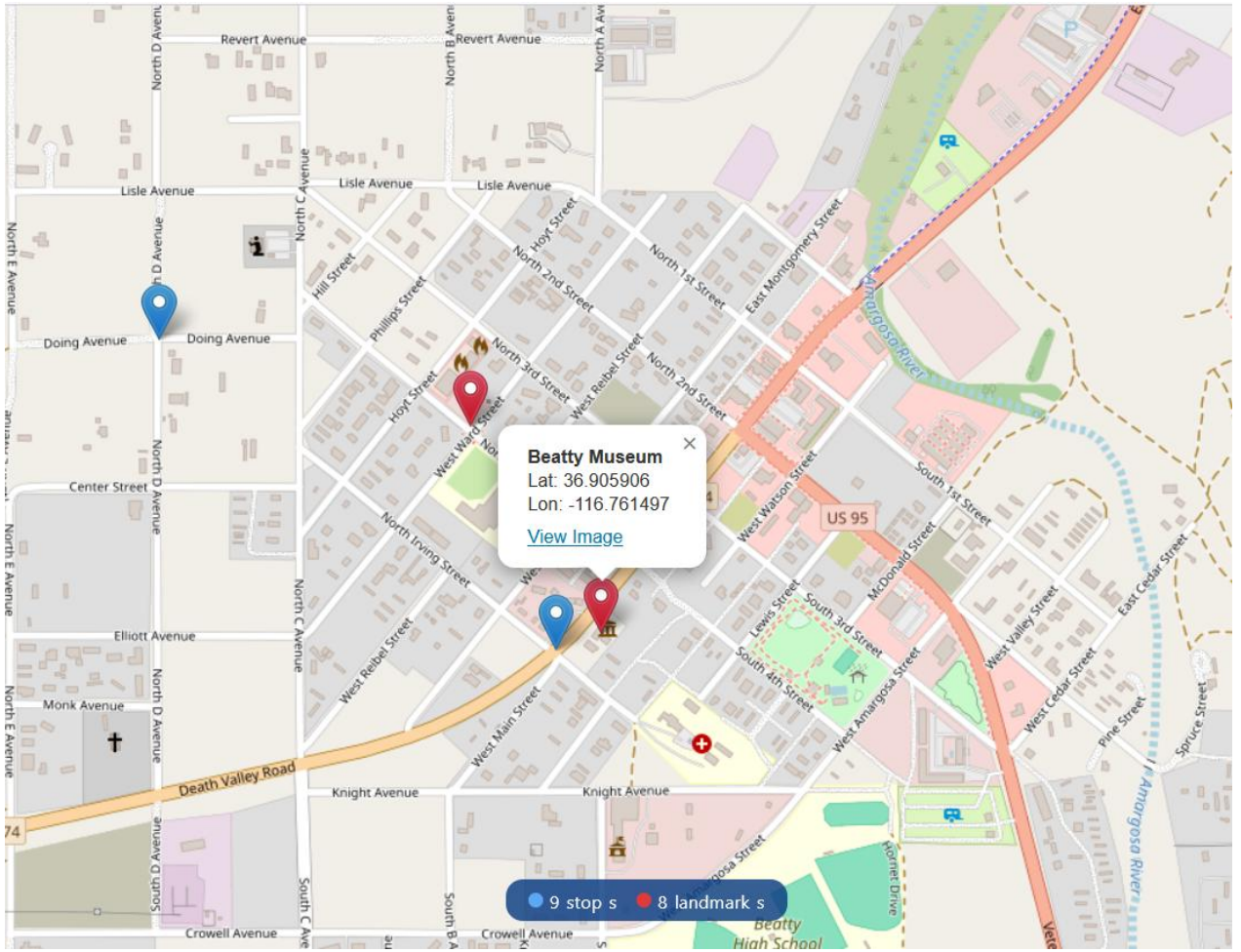
Typhon-RML: https://github.com/cefriel/typhon-rml

PRACTICAL SESSION 
EXERCISE 2 : CHIMERA PIPELINES

E2 – Complete exercise

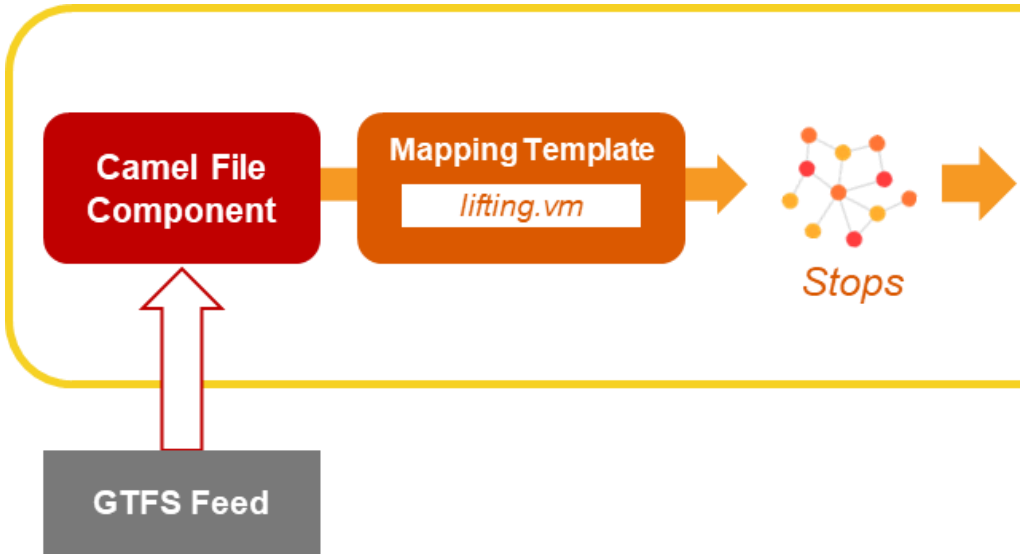
- **The final goal is now in sight!**
- Exercise 2 requires completing partial Chimera routes. […] markers indicate where steps must be added.
- **We want to get here, remember?**

- First task is easy, **replace the sample-feed.zip with the GTFS file you downloaded in folder e2/inbox**



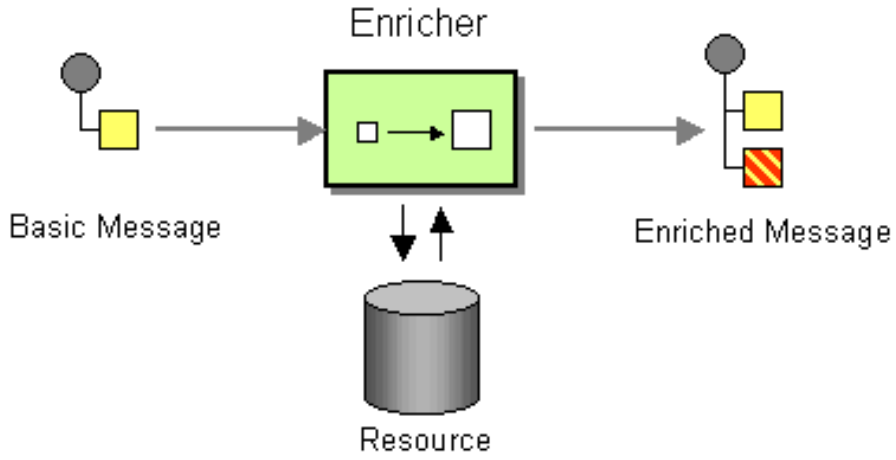
E2 – Lifting the GTFS Data

Pipeline: **e2/camel-routes/lifting.yaml**



E2 – Enriching the GTFS Data

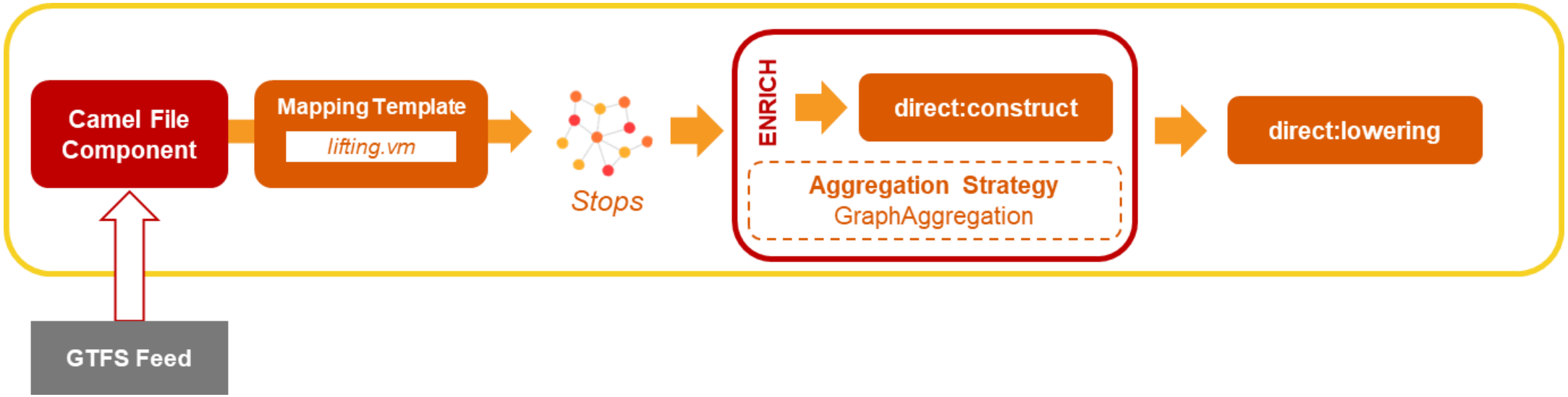
- We can use the **Content Enricher EIP**, implemented in Camel, **to access an external data source in order to augment a message** with missing information
- An **AggregationStrategy** defines how the Basic Message is merged with the incoming data to produce the Enriched Message (by default only the incoming data is kept).
- In Camel the Content Enricher EIP, is implemented as **Enrich** and **PollEnrich** depending on the operation to be performed

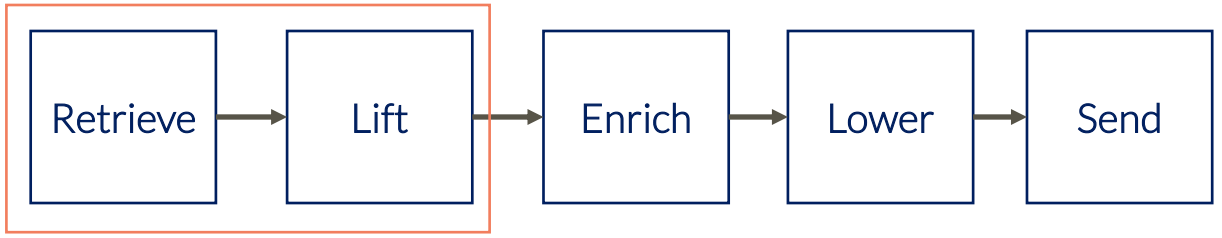


https://camel.apache.org/components/4.18.x/eips/enrich-eip.html

E2 – Enriching the GTFS Data

Pipeline: **e2/camel-routes/lifting.yaml**





E2 – Lifting the GTFS Data

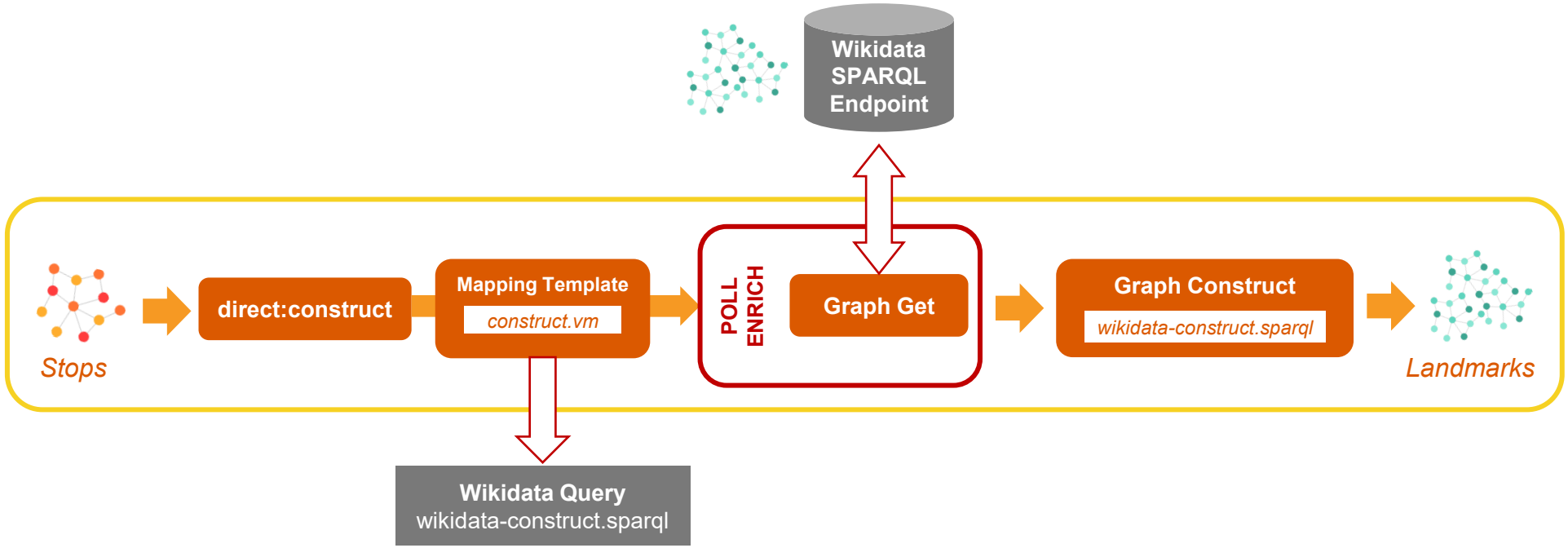
- The ‘lifting.yaml’ route:
  - **Reads** the input GTFS data from ‘*e2/inbox’* folder
  - **Sends** the lifted stops data to be enriched with wikidata data, via the ‘*direct:construct’ route*
  - Takes the enriched data and sends it to be lowered via the and ‘*direct:lowering*’ route
- **Your task**:
  - **Define a ChimeraResourceBean** referencing the file ‘*lifting.vm*’ in the ‘*e2/mappings’* directory (https://cefriel.github.io/chimera/chimera-resources/)
  - **Add a lifting step** using the camel-chimera-mapping-template component (https://cefriel.github.io/chimera/mapt-component/)
- **Hint**: check e0 and e1 if you need help

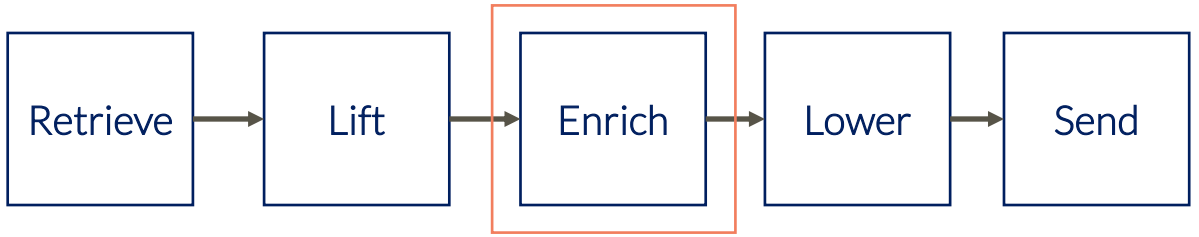
```yaml
- beans:
    # TODO: Define ChimeraResourceBean for lifting MTL mapping
    [...]

- route:
    from:
      uri: "file:inbox"
      parameters:
        antInclude: "*.zip"
        noop: true
      steps:
        - unmarshal:
            zipFile:
              usingIterator: true
        - split:
            simple: "${body}"
            steps:
              - choice:
                  when:
                    - simple: "${header.CamelFileName} == 'stops.txt'"
                      steps:
                        - convertBodyTo:
                            type: "java.lang.String"
                        # TODO: Perform lifting
                        [...]
                        # Load rdf into Chimera RDF memory graph
                        - to:
                            uri: "graph://get"
                            parameters:
                              rdfFormat: "turtle"
                        - log:
                            message: "\n E1 lifting result: ${body}\n"
                        - enrich:
                            simple: "direct:construct"
                            aggregationStrategy: "#class:com.cefriel.component.GraphAggregation"
                        - to:
                            uri: "direct:lowering"
```

e2/camel-routes/lifting.yaml

E2 – Graph Enrichment

Pipeline: **e2/camel-routes/enrich-wikidata.yaml**



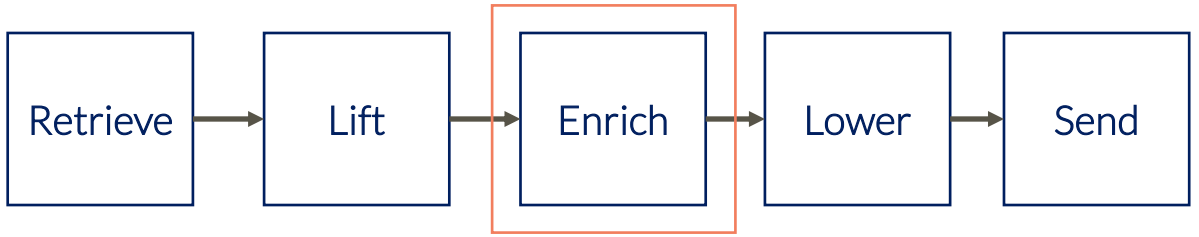


E2 – Graph Enrichment

- Let's **enhance** our stops data by **adding** information about **landmarks located near each stop**.
- We can obtain this information from **Wikidata** using a SPARQL CONSTRUCT query. The `*construct.vm*` MTL mapping file generates the necessary query.
- You heard that right, we are using a MTL mapping to **generate a dynamic SPARQL query**

```
#set ($positions = $reader.getDataframe("
SELECT ?position ?featureCoord
WHERE {
  ?position a tmjou:ScheduledStopPoint ;
             geo:hasGeometry [ geo:asWKT ?featureCoord ] .
}
ORDER BY ?position
LIMIT 10
"))

CONSTRUCT {
  ?place a wd:Q618123 ;
         rdfs:label ?placeLabel ;
         wdt:P625 ?coord ;
         schema:image ?image ;
     geo:hasGeometry [ geo:asWKT ?coord ] .
}
WHERE {

  #foreach($position in $positions)
    {
      SERVICE wikibase:around {
        ?place wdt:P625 ?coord .
        bd:serviceParam wikibase:center
            "$position.featureCoord"^^geo:wktLiteral .
        bd:serviceParam wikibase:radius "0.5" .
      }
    }#if($foreach.hasNext) UNION #end
  #end

  OPTIONAL { ?place wdt:P18 ?image . }

  SERVICE wikibase:label {
    bd:serviceParam wikibase:language "en".
  }
}
LIMIT 25
```

e2/mappings/construct.vm



E2 – Chimera Enrichment

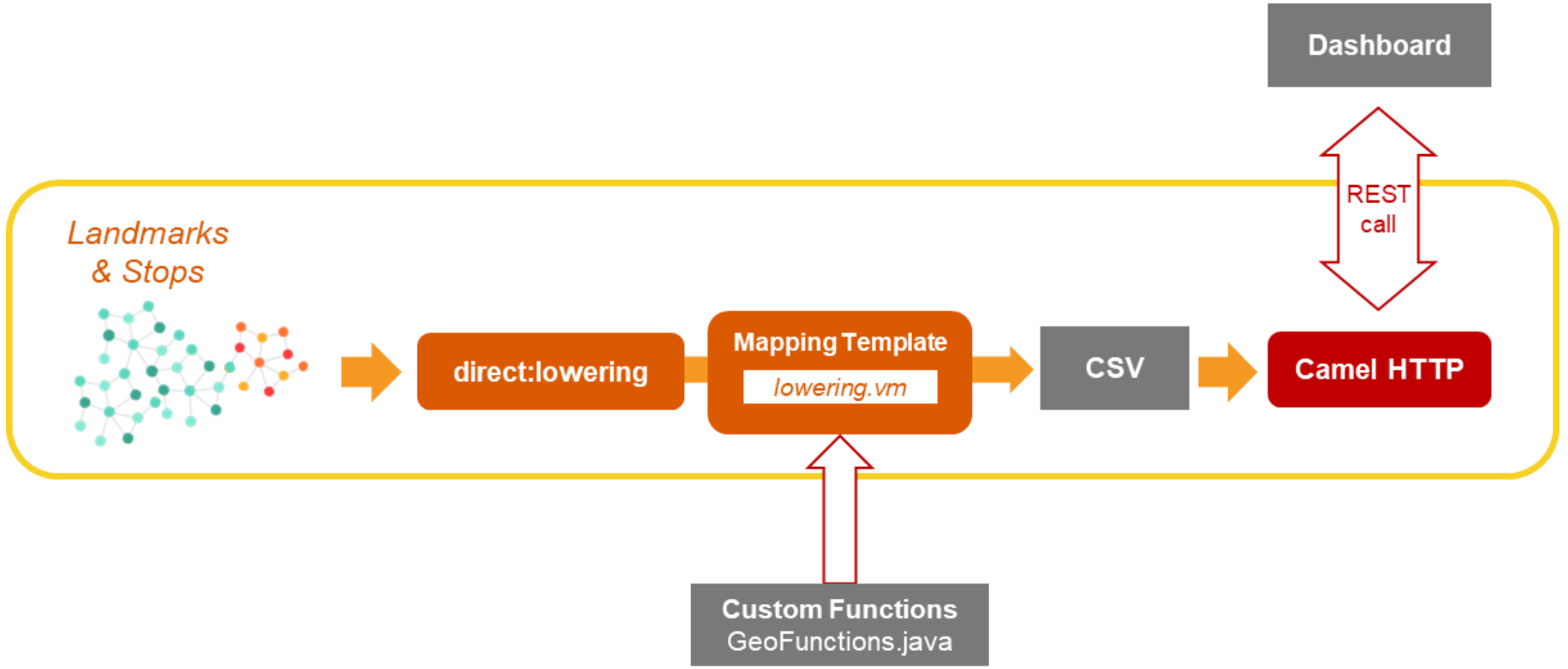
- Now we need to run the generated construct query on **Wikidata** through its **SPARQL endpoint**.
- **Your task is**:
  - Run the CONSTRUCT query using the **appropriate graph component operation** (https://cefriel.github.io/chimera/graph-component/)
  - Here's a **tip**, check the '*newGraph*' **parameter** to avoid trying to update the Wikidata graph

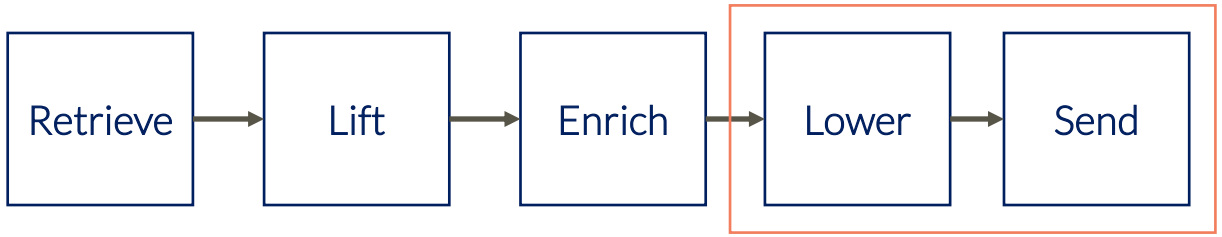
```yaml
- beans:
    - name: constructTemplate
      type: com.cefriel.util.ChimeraResourceBean
      properties:
        url: "file://./mappings/construct.vm"
        serializationFormat: "vtl"
    - name: wikidataQuery
      type: com.cefriel.util.ChimeraResourceBean
      properties:
        url: "file://./wikidata-construct.sparql"
        serializationFormat: "sparql"

- route:
    from:
      uri: "direct:construct"
      steps:
        # generate construct query
        - to:
            uri: "mapt://rdf"
            parameters:
              template: "#bean:constructTemplate"
              filename: "wikidata-construct.sparql"
        # get RDFGraph of type SPARQLEndpoint for Wikidata
        - pollEnrich:
            constant: "graph://get?sparqlEndpoint=https://query.wikidata.org/sparql"
        # TODO: Run construct query
        [...]
```

e3/camel-routes/enrich-wikidata.yaml

E2 – Lower and Send

Pipeline: **e2/camel-routes/lowering.yaml**





E2 – Lower and Send

Now all that is left is to lower the data in our KG and send the data to the visualization.

The '*lowering.yaml*' route:

- Performs a lowering using a **custom function to process coordinates**
- **Sends** the lowered data to the **visualization backend**

Your tasks is:

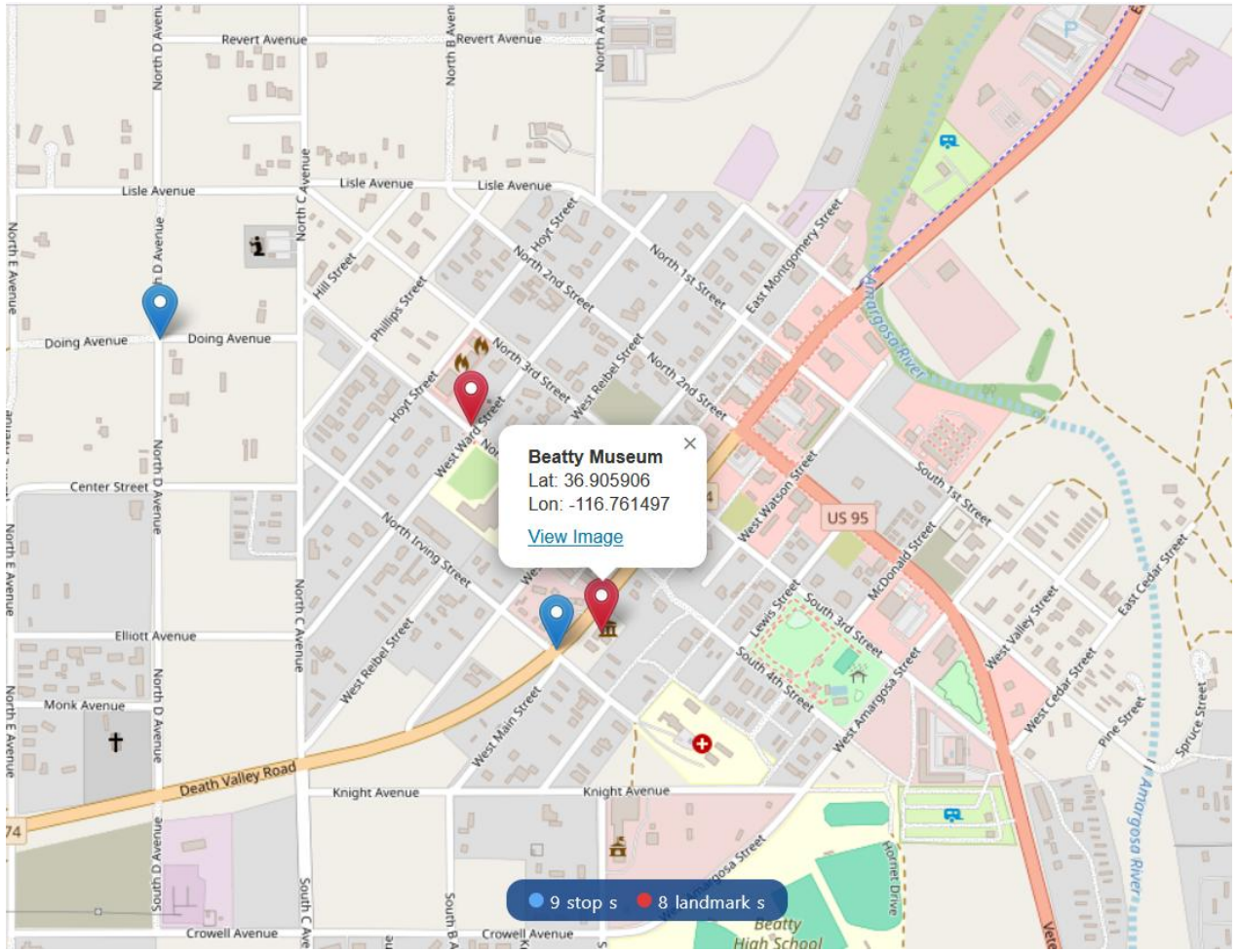
- Use the Camel HTTP component (https://camel.apache.org/components/4.18.x/http-component.html) to send the lowered data to the visualization tool via an **HTTP POST** request

```yaml
- beans:
    - name: lowering
      type: com.cefriel.util.ChimeraResourceBean
      properties:
        url: "file://./mappings/lowering.vm"
        serializationFormat: "vtl"

- route:
    from:
      uri: "direct:lowering"
      steps:
        # perform lowering
        - to:
            uri: "mapt://rdf"
            parameters:
              template: '#bean:lowering'
              customFunctions: "#class:com.cefriel.tutorial.GeoFunctions"
        - log:
            message: "performed lowering"
        - setHeader:
            name: Content-Type
            constant: text/csv
        - setHeader:
            name: Authorization
            constant: Bearer tutorial-kg4di
        - log:
            message: "sending result to visualization..."
        # TODO: Send result to viz using HTTP POST via HTTP component
        - [...]
        - log:
            message: "\n backend response: ${body}\n"
```

e2/camel-routes/lowering.yaml

E2 – Check the web app!

- **Go at** https://knowledge.c-innovationhub.com/kg4di
- Check your stops & landmarks and explore the ones of other participants!



CHIMERA IN ACTION

PART 4

TANGENT Use Case

We consider a scenario from the **TANGENT** project involving the collection of real-time data from several IoT sensors and Web Services.

Real-time data


Semaphores

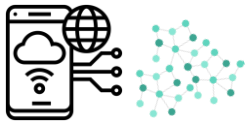
Parking

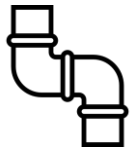

Traffic Data


Events

Solution

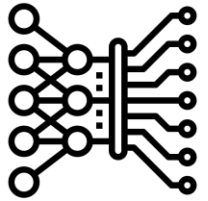
- **Harmonise ingested data** using a common ontological data model for KG-enabled applications
- **Enrich data** ingested using additional data sources from a Data Catalogue
- **Transform data** in a specific output format and send them to a data sink





Objectives

Real-time integrated **dashboard** for **city** and **transport authorities**



Intelligent **incident detection** and **response plan prescription**

Scrocca M. et al., *Intelligent Urban Traffic Management via Semantic Interoperability across Multiple Heterogeneous Mobility Data Sources,* ISWC 2024

Icons from Flaticon

Chimera Pipeline in TANGENT

Data Catalogue

TFGM_CARS_04 Real-time Journey Times Data
Custom JSON Format

TFGM_CARS_18 Journey Times Sensors position
Custom JSON Format

REN_TRAF_02 Real-time road traffic data
Custom JSON Format

Declarative Mappings

Lifting block

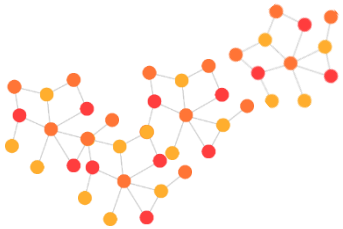
Lifting block

Declarative Mappings

Lifting block

Declarative Mappings

Harmonized and fused representation
based on the
TANGENT Reference Conceptual Model

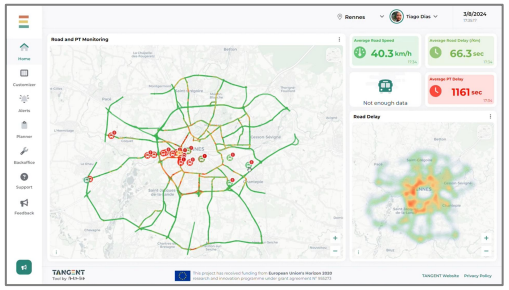


Lowering block

Declarative Mappings

Only these mappings should be updated to accommodate changes in the expected output format

TANGENT Dashboard

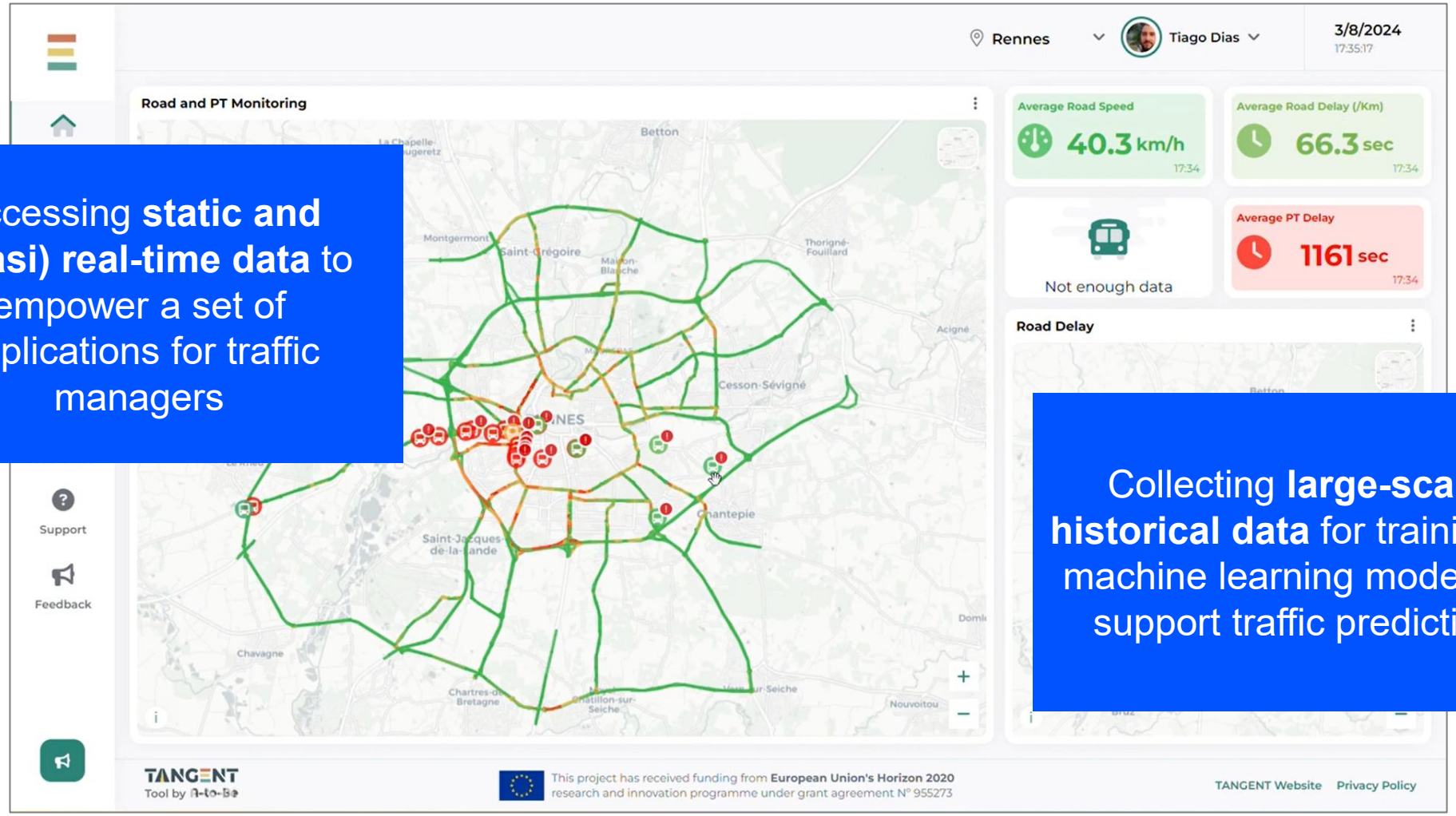


TANGENT Data API
JSON Format

Scrocca M. et al., *Intelligent Urban Traffic Management via Semantic Interoperability across Multiple Heterogeneous Mobility Data Sources,* ISWC 2024

Intelligent Multimodal Traffic Management

Accessing **static and (quasi) real-time data** to empower a set of applications for traffic managers

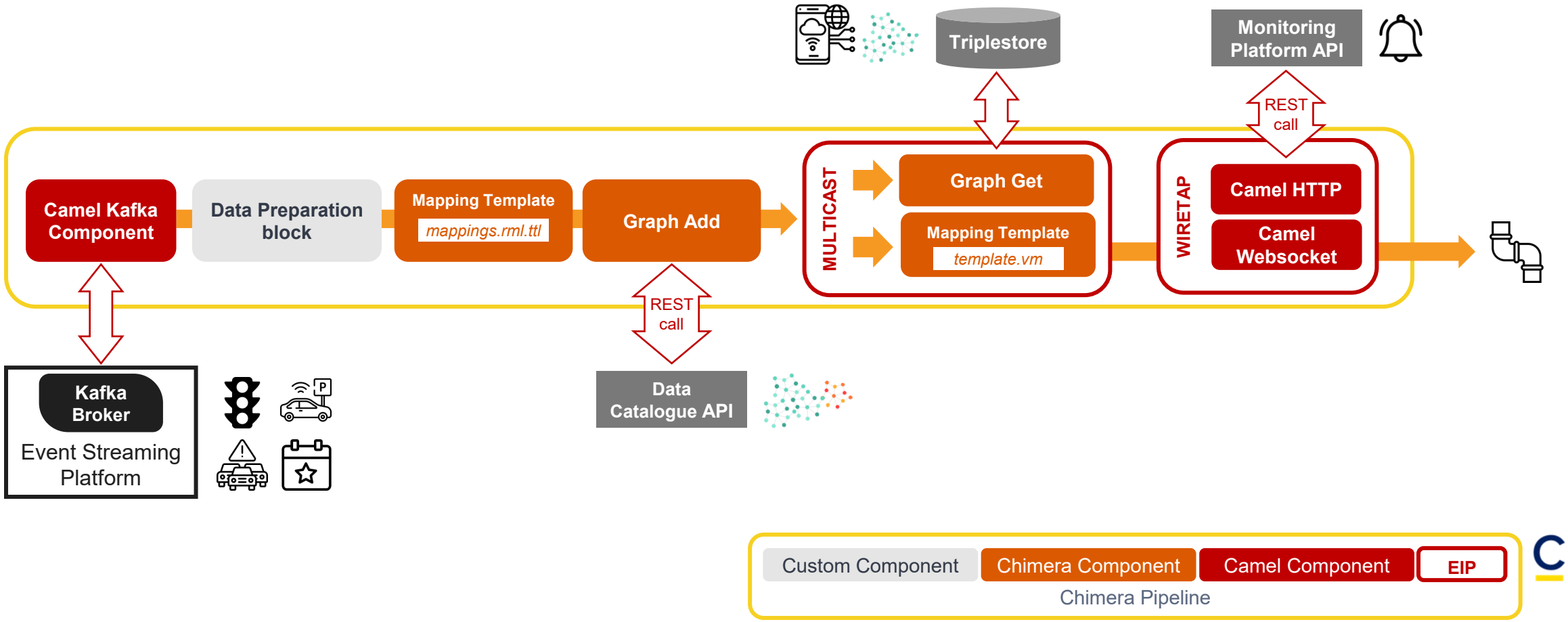


Collecting **large-scale historical data** for training of machine learning models to support traffic prediction

Scrocca M. et al., *Intelligent Urban Traffic Management via Semantic Interoperability across Multiple Heterogeneous Mobility Data Sources,* ISWC 2024

Advanced Pipeline Example

The composition of **Chimera components** and **Apache Camel components** allows configuring pipelines to cover advanced requirements. An example converter for the presented scenario is shown below.



Low-Code Edge Intelligence for Smart Factory

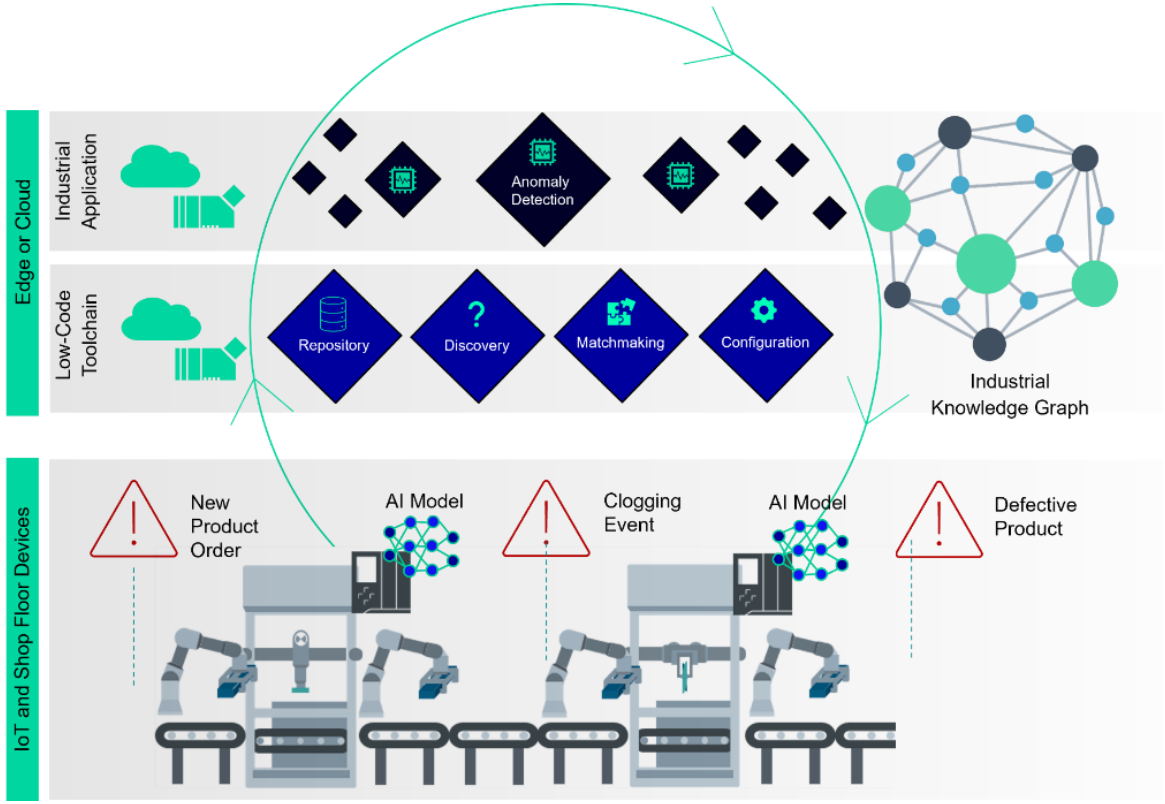
Objectives

- Enable low-code configuration of production lines for order-specific product assembly
- Support flexible reprogramming to meet changing manufacturing demands

Devices: Demonstrator in the Siemens Autonomous Factory Lab (AFL) considering three macro-modules: assembly, mini-backbone (transport/distribution), and shipping.

Challenges

- To enable the discovery and low-code programmability of heterogeneous devices, there is the need for structured and interoperable device descriptions
- OPC UA standard can not be fully expressed using Web of Things (WoT) and needs dedicated transformation and discovery



https://tangere-xperience.siemens.cloud/AFL/

Scrocca M. et al., *A DataOps Toolbox Enabling Continuous Semantic Integration of Devices for Edge-Cloud AI Applications*, ISWC 2025

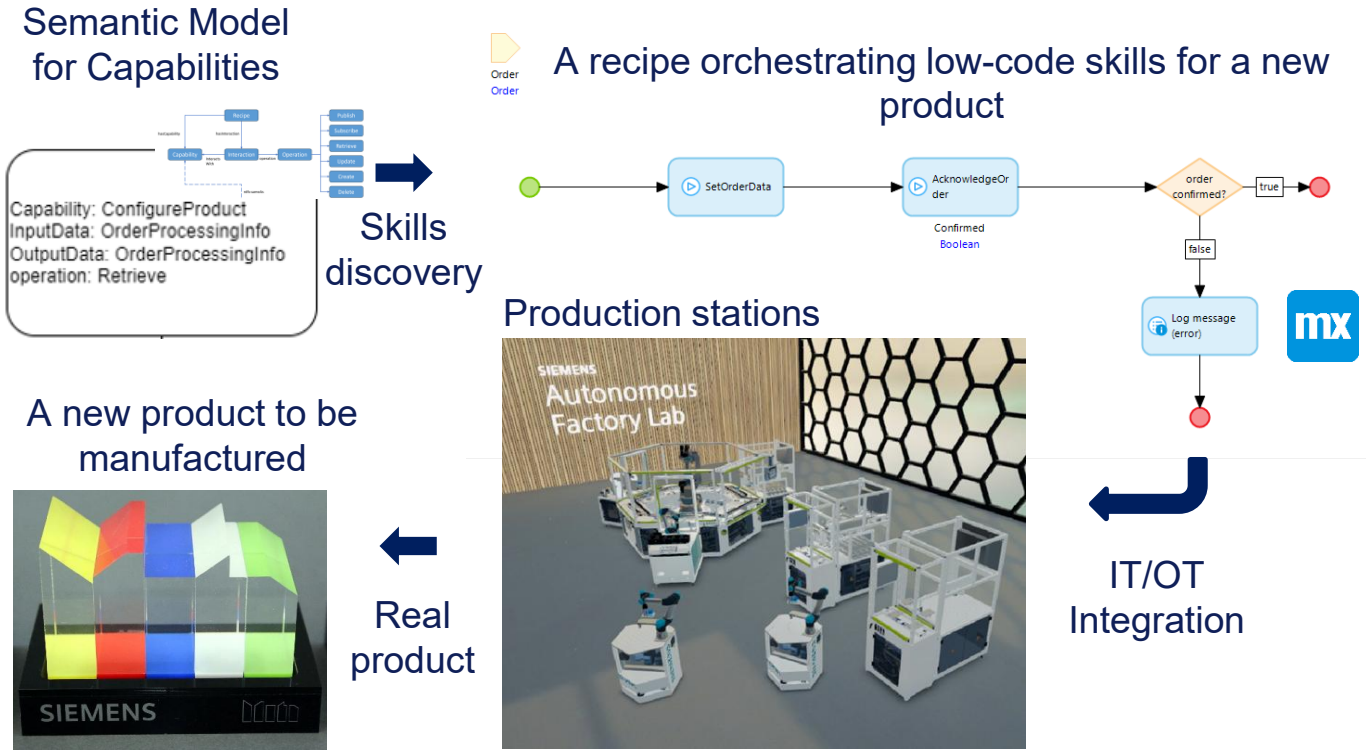
Low-Code Edge Intelligence for Smart Factory

Chimera pipelines:

- **Insertion and retrieval** of OPC UA nodes descriptions in the KG: OPC UA → RDF (lifting) and RDF → OPC UA (lowering)
- Endpoint to **support skills discovery of available devices**. Adds support for OPC UA devices and is aligned with the WoT TD Discovery (Domus TDD).

Validation results:

- **AFL devices** successfully **described in the KG** and made available for discovery
- **LLM-enabled skill discovery** leverage the KG and removes dependency on SPARQL queries
- Technicians could define and execute new **recipes for production workflows** adopting a **low-code** approach via Mendix



Scrocca M. et al., *A DataOps Toolbox Enabling Continuous Semantic Integration of Devices for Edge-Cloud AI Applications*, ISWC 2025

Smart Vehicle to Infrastructure

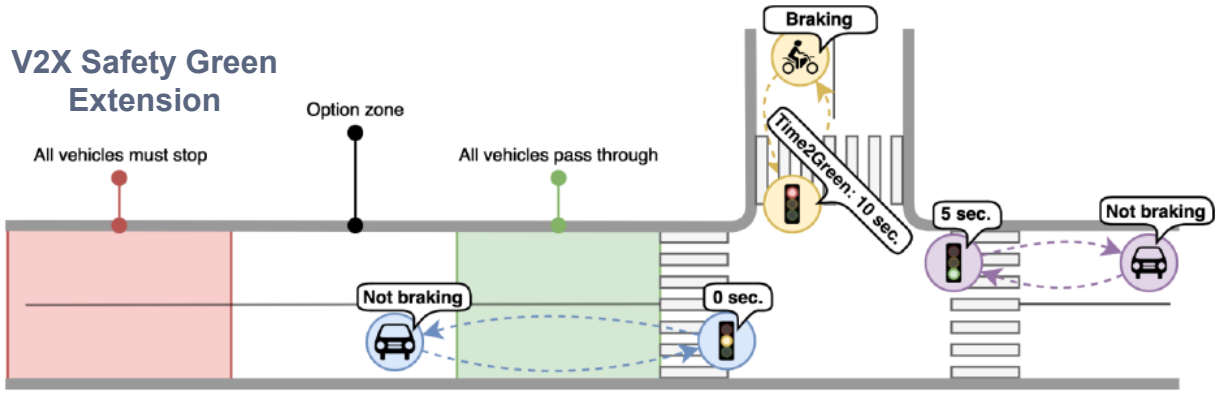
Objectives:

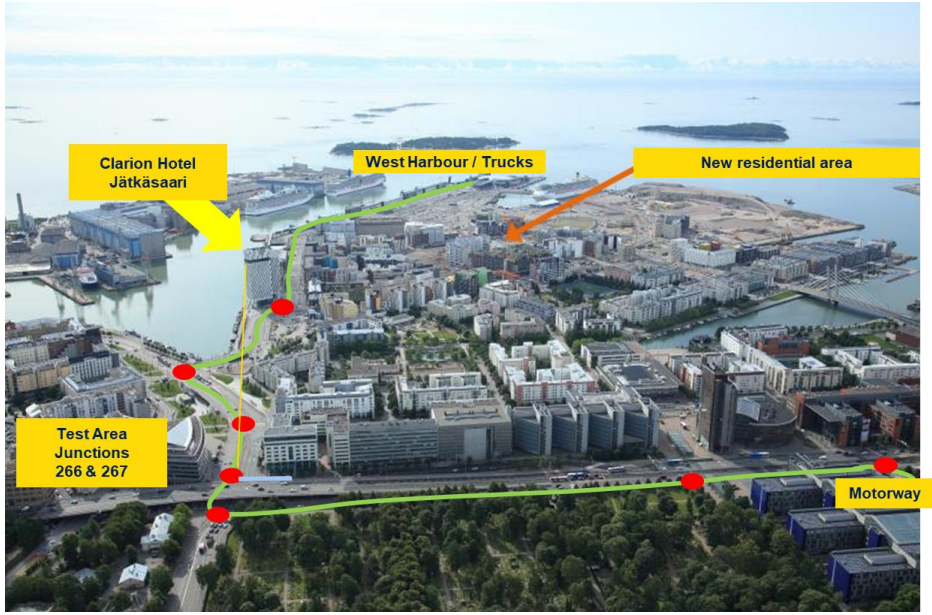
- Develop intelligent AI-enabled traffic management applications with minimal programming.
- Support real-time computation of traffic indicators from heterogeneous data (cameras, lidars, radars, and public transport feeds) and use them to define adaptive traffic control rules and send signals to connected vechicles.

Challenges:

- Heterogeneous data streams to be harmonised and fused at runtime
- Edge computation at each intersection for low-latency decision-making
- Minimal resource consumption on edge hardware

Devices: Demonstrator in Helsinki's Mobility Lab corridor comprising 6 intersections, 17 radars, 21 edge units and ~~2~~ (now 3!) connected vehicles.





Scrocca M. et al., *A DataOps Toolbox Enabling Continuous Semantic Integration of Devices for Edge-Cloud AI Applications*, ISWC 2025

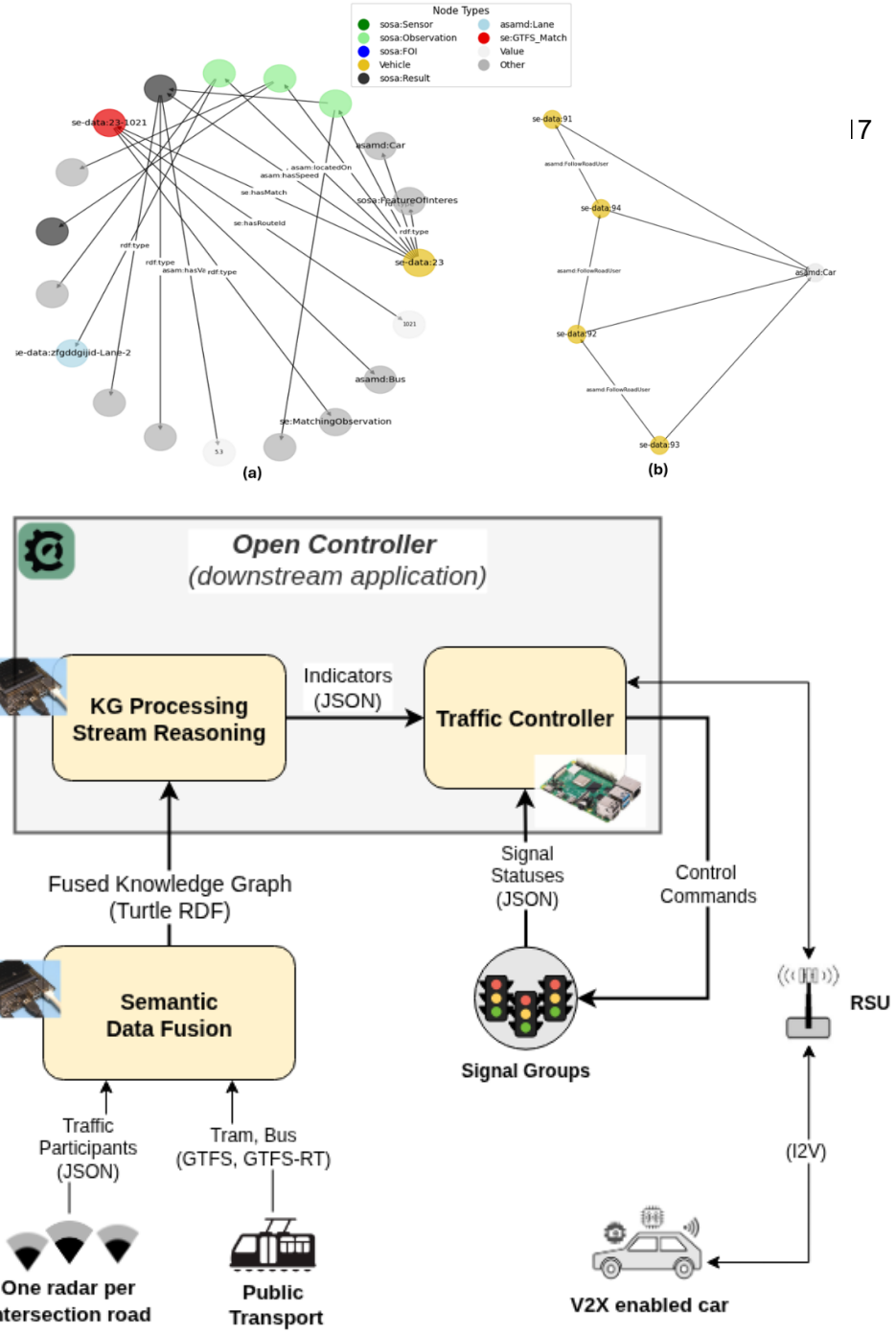
Smart Vehicle to Infrastructure

Chimera pipelines:

- Harmonize **multimodal sensor data and public transport feeds** into RDF using ASAM OpenX and SOSA ontologies
- **Fusion logic** to correct sensor inaccuracies (e.g., tram misidentified as trucks)
- **Deployment via templates** of pipelines on edge units (RSUs) and cloud environments to test performance and scalability

Validation results:

- Deployed in **6 intersections** across Helsinki's Mobility Lab corridor
- RDF stream generated is processed to compute indicators
- Edge deployment: Average pipeline latency **<100 ms**, memory usage **≈100 MB** per edge unit. Successfully met expectations for **10 msg/sec** at intersections (4 KB/msg typical load).
- Laboratory benchmark: processed **up to 400 requests/sec**, average **<7 ms per message** (60 KB payloads).



Scrocca M. et al., *A DataOps Toolbox Enabling Continuous Semantic Integration of Devices for Edge-Cloud AI Applications*, ISWC 2025

Smart Health Rehabilitation

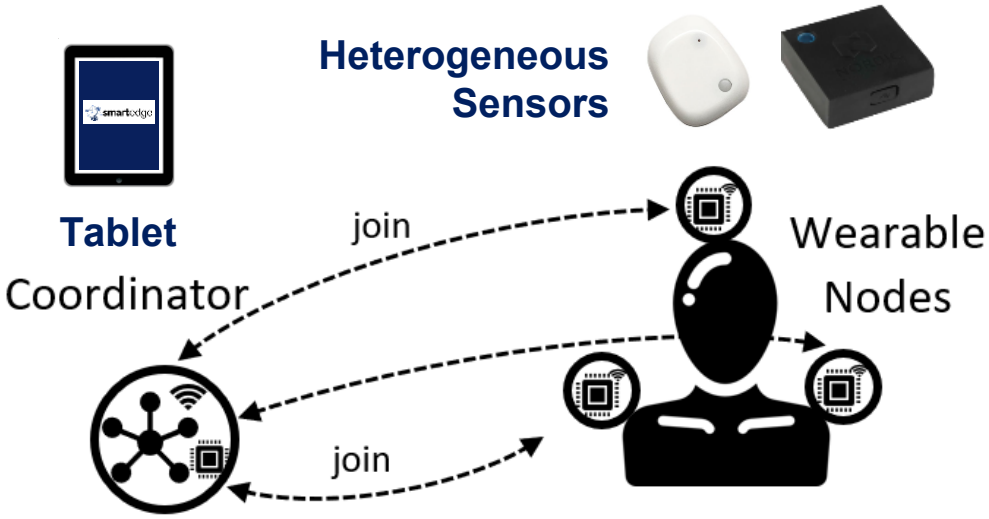
Objectives:

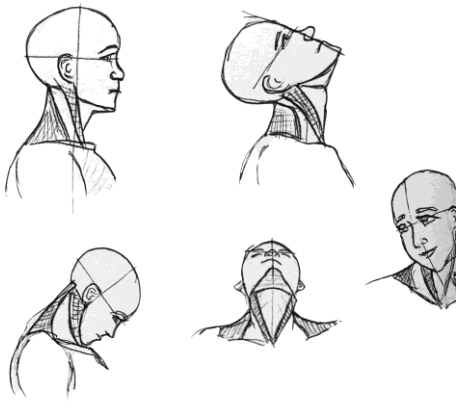
- Support home-based physiotherapy (neck rehabilitation post-surgery or injury recovery) through AI-driven activity tracking
- Enable low-code personalization of exercises by the physiotherapist

Devices: Tested in collaboration with PhysioLab at HES-SO by providing wearables sensors (3 for each patient) and a tablet for exercise execution

Challenges:

- Support remote exercise definition and monitoring by the physiotherapist
- Matching and configuring heterogeneous devices (e.g., personal devices) with low capabilities





Scrocca M. et al., *A DataOps Toolbox Enabling Continuous Semantic Integration of Devices for Edge-Cloud AI Applications*, ISWC 2025

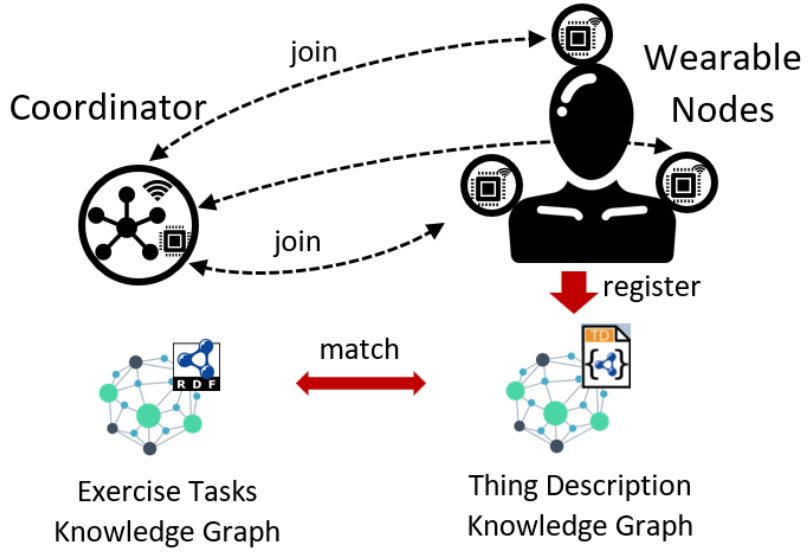
Smart Health Rehabilitation

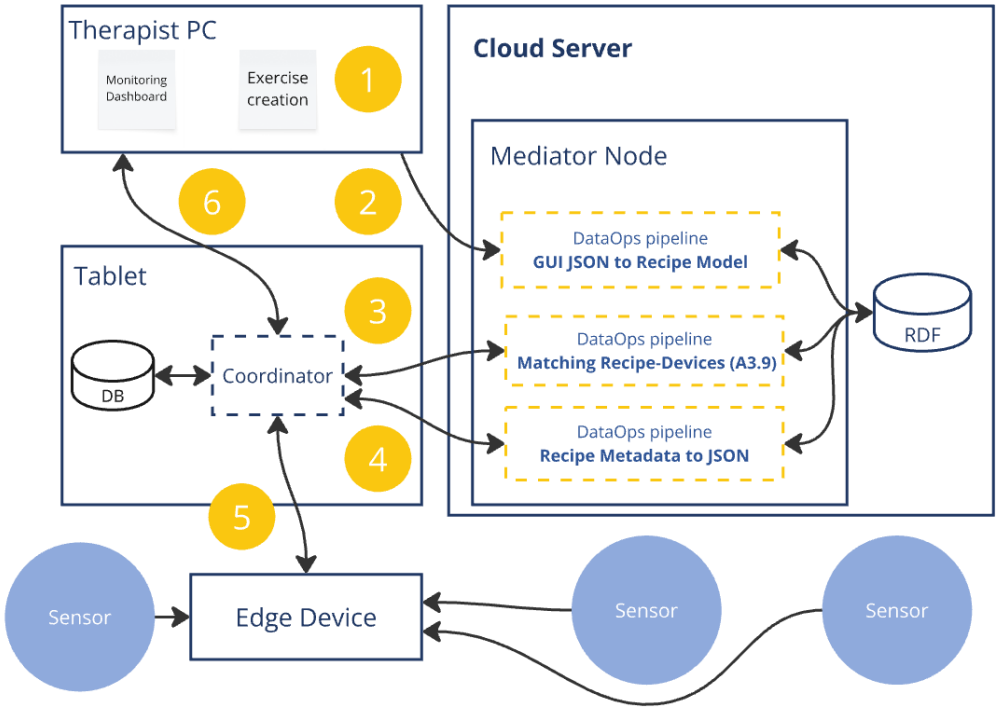
Chimera pipelines:

- Exercise transformation to the **Recipe Model ontology**
- **Device-capability matching** for exercise execution considering available devices
- **Runtime conversion** of recipe metadata for the coordinator
- **Karavan low-code editor** used to define pipelines
- **Cloud deployment of pipelines as a mediator service** invoked by local orchestrators running on the edge

Validation results:

- **Exercise definition** successfully tested by physiotherapy lecturer
- Involvement of students wearing devices and performing exercises under the supervision of the physiotherapist for the **validation of the exercise movements recognition**





Scrocca M. et al., *A DataOps Toolbox Enabling Continuous Semantic Integration of Devices for Edge-Cloud AI Applications*, ISWC 2025

Use Case: Security-Aware Service Orchestration

Problem:

- Allocate network resources (device-edge-cloud) dynamically based on service tier (Gold/Best Effort) and security requirements, responding to vulnerability/attack events in real-time

How Chimera helps:

- Ingest RDF-formatted network status data (describing hardware specs, running software, DPU capabilities, security features) + telemetry streams
- Feeds data to a match-maker engine that matches service requests to optimal resources via semantic reasoning

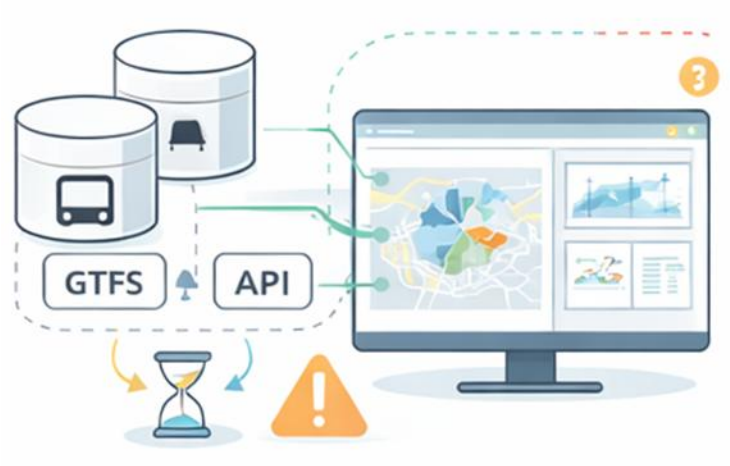
Smart City Data Interoperability

Problem:

- Integrate heterogeneous IoT devices and urban data sources, such as traffic, air quality, and noise, despite differences in data semantics, formats, and access methods.

How Chimera helps:

- Use Chimera to harmonize and unify city data, providing planners, municipalities, and operators with a holistic view of urban operations to identify critical intervention areas and support informed decision-making.



Thank you!

MARCO GRASSI
Knowledge Technologies Researcher
Cefriel

LinkedIn

marco.grassi@cefriel.com

MARIO SCROCCA
Senior Knowledge Technologies Researcher
Cefriel

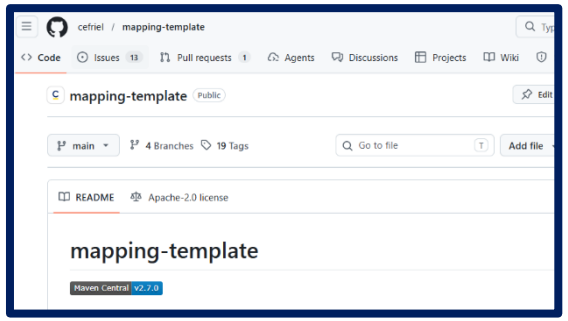
LinkedIn

mario.scrocca@cefriel.com

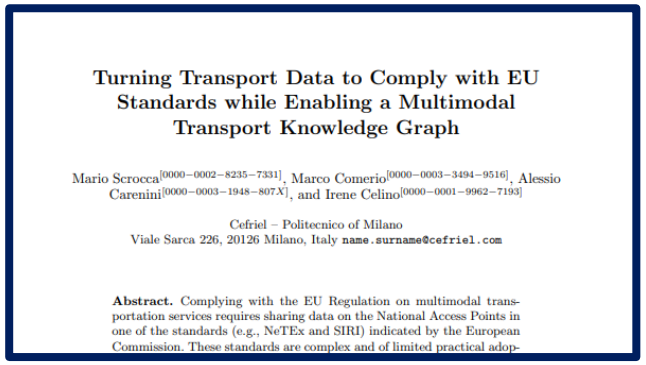
Now, you are ready to start playing with Chimera!

Chimera's story and references to know more

2015-19 *IP4: Interoperability Framework for the Transportation Domain*
Ontology, mapping techniques/tools to transform data to/from RDF



2021-22 *Mapping Template*
- Focus on performance and scalability of the transformation process
- Challenges in KG construction
- Creation of a separate library for the mapping-template to better structure the template-based approach and extend it to generic data transformations (including lifting)



Turning Transport Data to Comply with EU Standards while Enabling a Multimodal Transport Knowledge Graph

Mario Scrocca[0000−0002−8235−7331], Marco Comerio[0000−0003−3494−9516], Alessio Carenini[0000−0003−1948−807X], and Irene Celino[0000−0001−9962−7193]

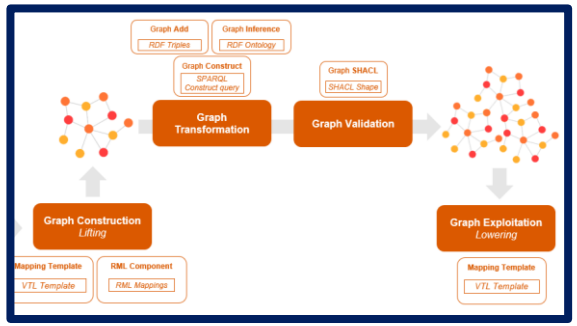
Cefriel – Politecnico of Milano
Viale Sarca 226, 20126 Milano, Italy name.surname@cefriel.com

Abstract. Complying with the EU Regulation on multimodal transportation services requires sharing data on the National Access Points in one of the standards (e.g., NeTEx and SIRI) indicated by the European Commission. These standards are complex and of limited practical adop-

2019-21 *Initial Chimera Implementation*
- Project based on Apache Camel
- RML processor for lifting
- Reversible RML project failure → custom template-based processor for lowering based on Apache Velocity

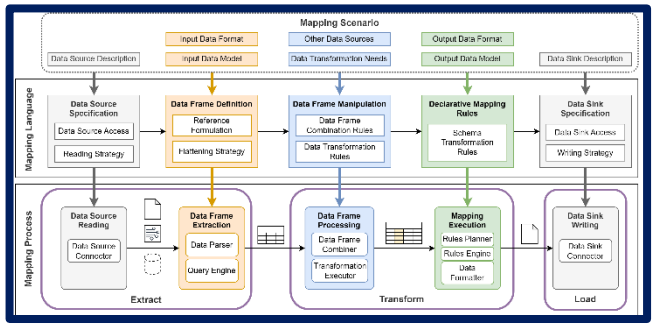
In Use paper on applying Chimera to transport data compliance (conversion GTFS -> NeTEx) [link]

Chimera's story and references to know more



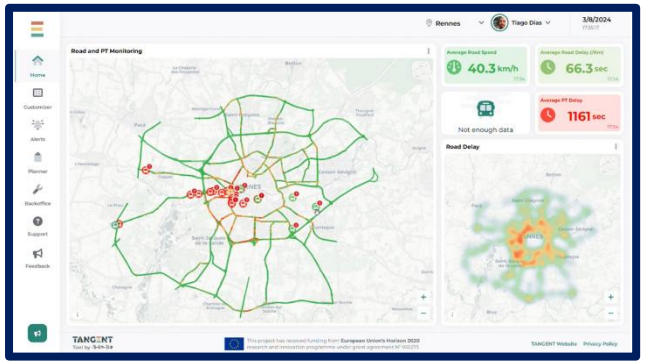
2023-Today *Advancing Chimera*

- Refactor as proper Apache Camel components
- Chimera tutorial and documentation
- Published on Maven Central
- Creation of deployment templates to execute same Chimera pipeline on edge and cloud



2024-Today *Typhon-RML*

- Workflow for declarative knowledge conversion mapping rules and definition of MTL for mapping-template [link]
- Automatic translation of RML mapping rules to Chimera pipelines with Mapping Template rules [link]



TANGENT

smartedge

SMARTY

Urban-Flow

2022-Today *Chimera in use!*

- Intelligent Urban Traffic Management via Semantic Interoperability across Multiple Heterogeneous Mobility Data Sources [link]
- A DataOps Toolbox Enabling Continuous Semantic Integration of Devices for Edge-Cloud AI Applications [link]