FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

20th International Conference on Knowledge Engineering and Knowledge Management

kbs Knowledge-Based Systems Group

A Benchmarking Framework for Stream Processors

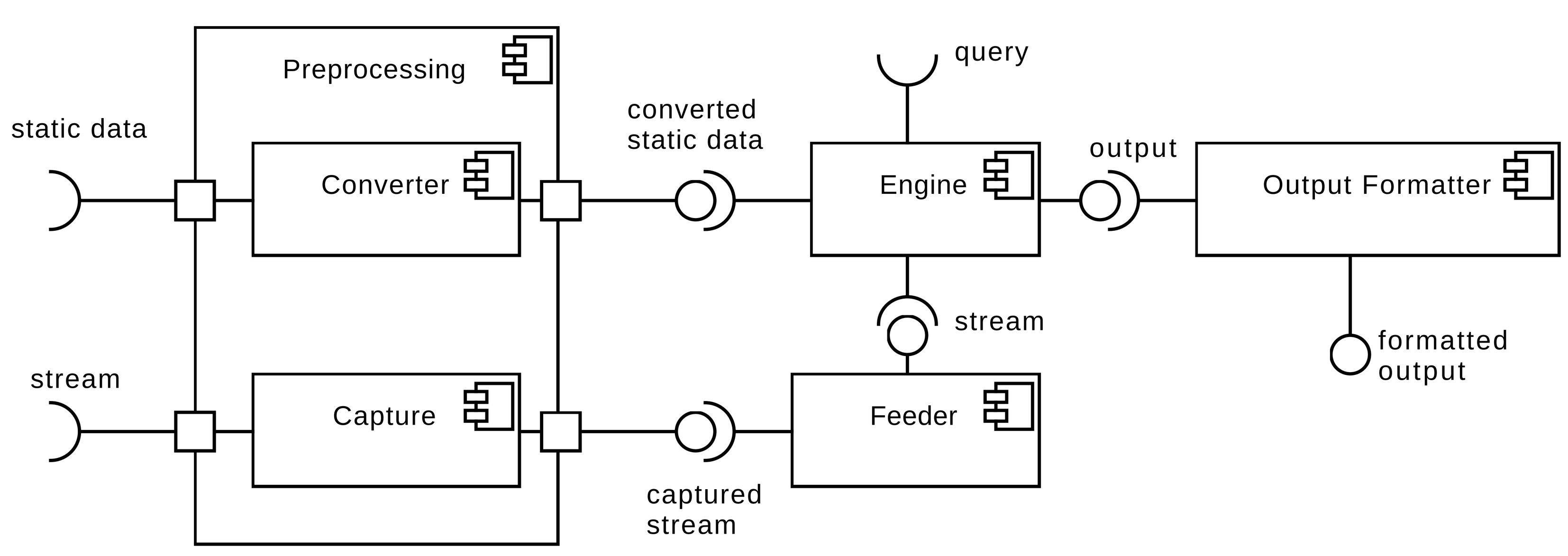
Vienna University of Technology
Institute of Information Systems
Knowledge-Based Systems Group

Andreas Moßburger, Harald Beck, Minh Dao-Tran, Thomas Eiter

Challenges

- The engines work on different data formats
- Tests must be reproducible
- No unified way to query the engines
- Output has to be generated in a unified format for easy comparison

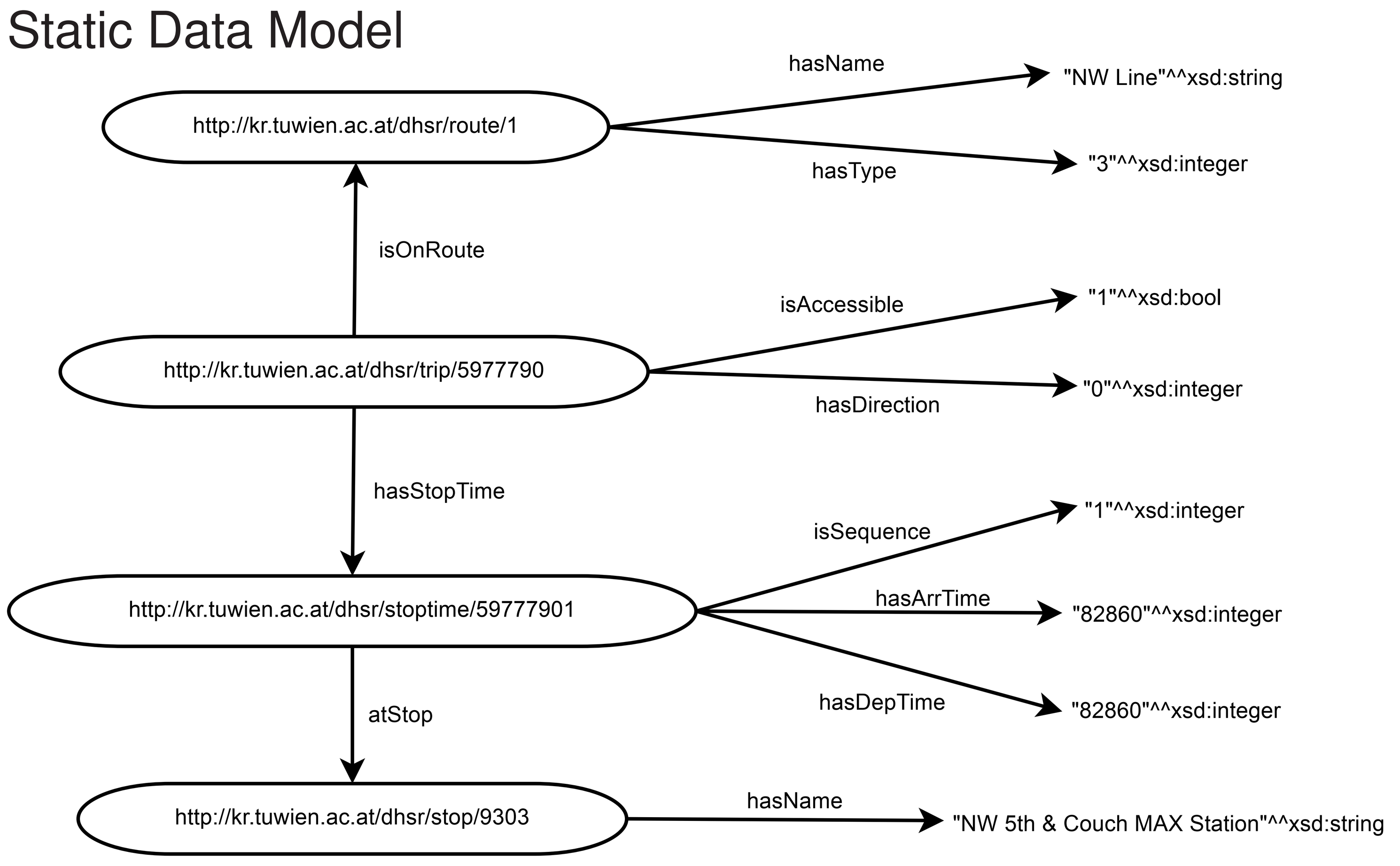
Framework Architecture



- **Converter** is responsible for converting any static data from the provided format to a format that can be read by an engine.
- **Capture** extracts relevant data from a data stream and stores it, allowing reproducible evaluations. The format of the stored data should be generic, so that only minimal conversions are necessary for a particular engine. Additionally, timing information of the captured data should be stored, so it can be played back authentically.
- **Feeder** is responsible for replaying the captured streaming data to the engines. It allows arbitrary fine control over the streaming process. Data may be streamed using authentic or artificial timing, like streaming a certain amount of data per time unit.
- **Engine** wraps the evaluated engine. Wrappers provide a standardized way of accessing/outputting data for different engines but are not allowed to affect their performance.
- **Output Formatter** converts the output data from different engines to a canonical form.

General Transit Feed Specification

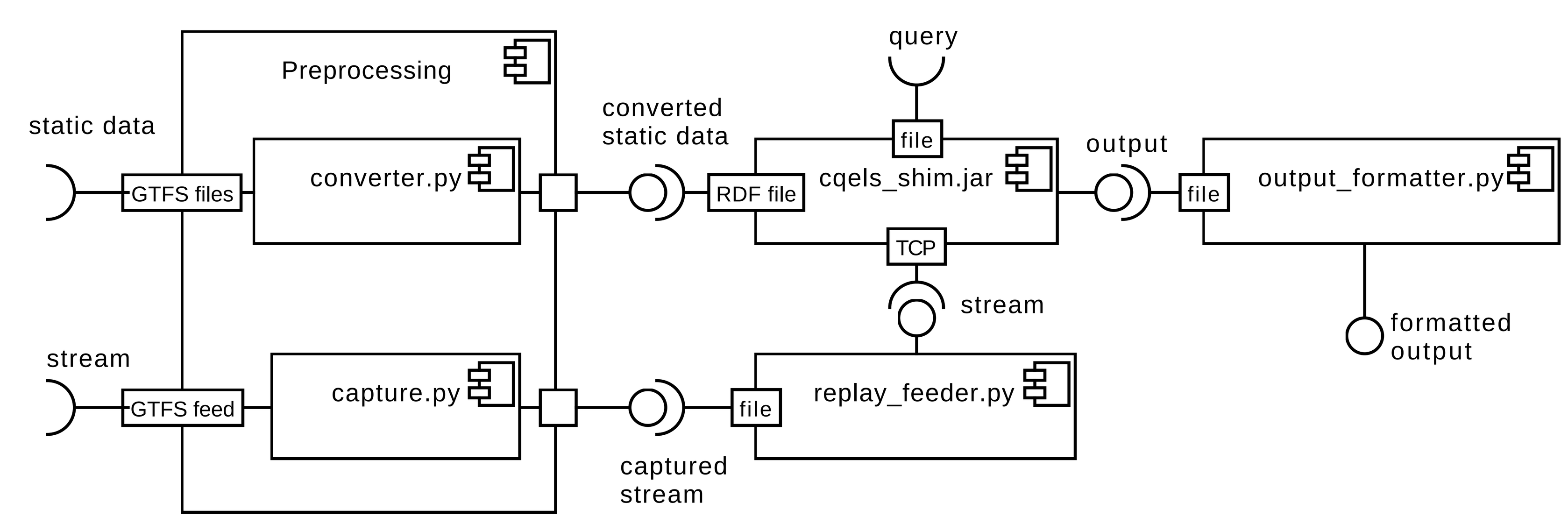
Static Data Model



Streaming Data

- *TripUpdate*: represents a change to a timetable and consists of possibly multiple delays or new arrival times for single stops of a trip.
- *VehiclePosition*: tells the position of a vehicle relative to a stop.

Implementation



Components implemented as Pyhton scripts:

- `gtfs-converter.py` and `gtfs-capture.py` implement the converter and capture modules, resp. These scripts are specific to the GTFS use case. All other scripts and programs are generic and do not make any assumptions about the data domain.
- `simple_feeder.py`, `replay_feeder.py` and `triple_to_asp.py` provide implementations of the feeder module.
- `output_formatter.py` covers the output formatter module.

Different wrappers were implemented to access the engines.

Code and queries are available at
`https://github.com/mosimos/sr_data_generator/`

Evaluation with clingo as an oracle

Queries for evaluating functionality

01 Simply output all *hasArrived* triples.
02 Use `FILTER` to output only *hasDelay* triples with a delay greater than a certain value.
03 Use `UNION` to output both *hasDelay* and *hasArrived* triples.
04 Use `OPTIONAL` to output *hasDelay* and optionally a *hasArrived* triple of the same stop.
05 Calculate a value (delay in minutes) directly in `SELECT` clause.
06 Calculate a value (delay in minutes) using a `BIND` clause.
07 Aggregate function `COUNT`.
08 Aggregate function `COUNT DISTINCT`.
09 Aggregate function `MAX`.
10 `ORDER BY`
11 Simple join combining streaming and static data.
12 Simple join combining streaming and static data, using `OPTIONAL` clause.

	Queries											
	01	02	03	04	05	06	07	08	09	10	11	12
C-SPARQL	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
CQELS	✓	✓	-	-	-	-	✓	✓	-	✓	✓	-
Spark	✓	✓	✓	✓	✓	✓	✓	✓	✓	-	✓	✓
clingo	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Table: Results of functionality test (✓query produced output, - query resulted in error or didn't return anything)

Correctness

- CQELSand Spark conformed to the results predicted by clingo.
- C-SPARQLmisses some output by clingo.

Contact: {mossburger,beck,dao,eiter}@kr.tuwien.ac.at