

# Building a Self-Service IoT Analytics Toolbox

## Basics, Models and Lessons Learned

**Rudi Studer, Dominik Riemer**

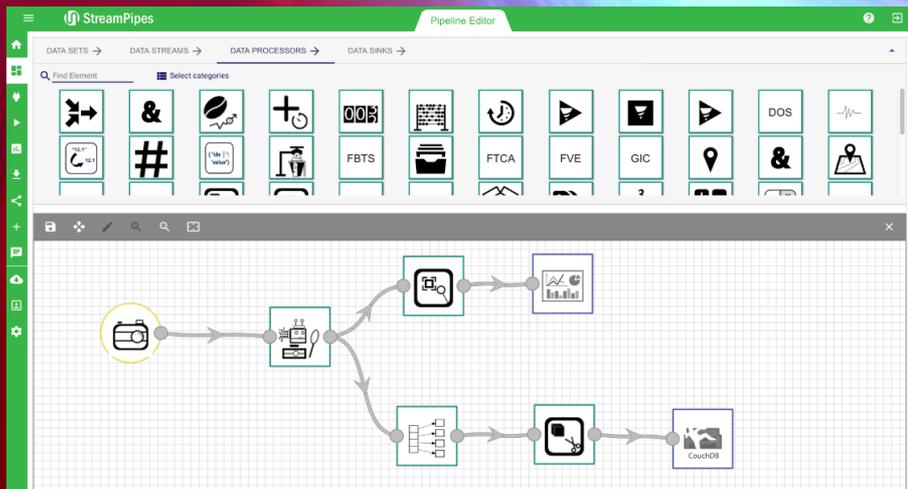
IC3K 2018, Seville, September 20, 2018

Institute of Applied Informatics and Formal Description Methods (AIFB)



# This talk

How to enable **application specialists** to **create** and **execute IoT analytics applications** based on distributed **stream processing** in a **self-service** manner?



Deploy

Distributed Stream  
Processing  
Architecture

# Outline

- **Introduction & Motivation**
- Problems
- Development methodology for event-driven applications
  - Overview
  - Modeling of data streams and processing elements
  - Definition and execution of distributed processing pipelines
- Implementation and evaluation
- Conclusion

# Internet of Things

Data streams anywhere

Energy consumption

Machine data

Logistics

GPS

Traffic sensors

Enterprise applications

Dust particles

Environmental data

# Events

GPS

## Event [Luck02]

A record or an activity within a system

### Event Type

Event Property

```
timestamp : long
vehicleId : string
latitude  : double
longitude : double
```

### Event Object

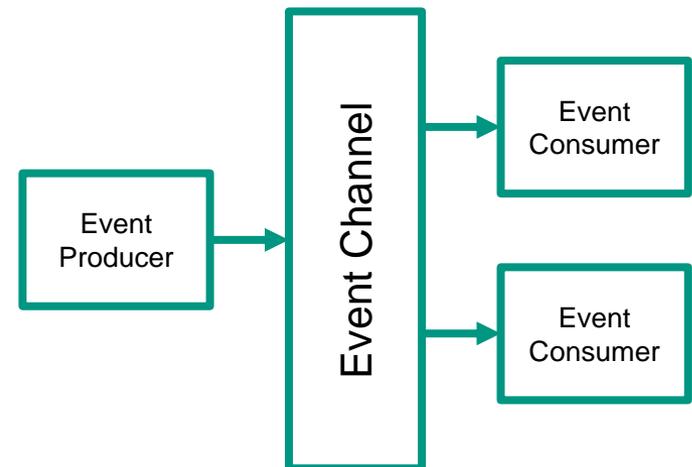
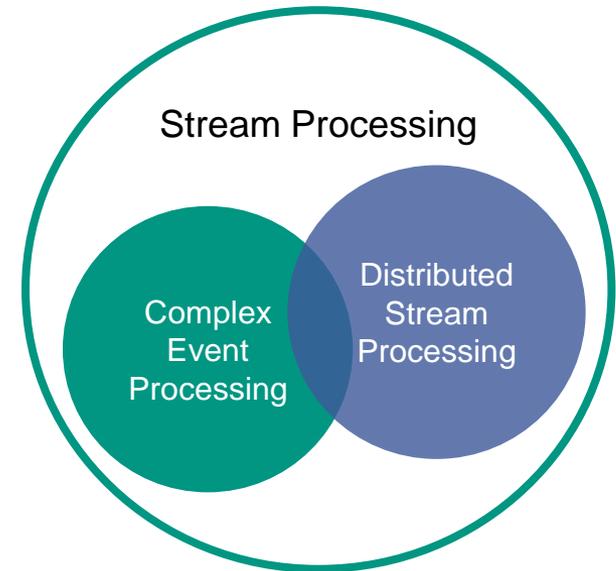
```
timestamp : 1453478150
vehicleId : ID5
latitude  : 48.94
longitude : 8.40
```

### Event Representation (JSON)

```
{
  "timestamp" : 1453478150
  "vehicleId" : "ID5"
  "latitude"  : 48.94
  "longitude" : 8.40
}
```

# Stream Processing

- **Event-Driven Architecture [BrDu10]**
  - Producers and consumers are completely decoupled
  - Publish/Subscribe [EtNi11]
    - Multiple consumers per event
- **Push Interaction [EtNi11]**
  - One-way-communication
  - Producer does not expect that an event is processed by any consumer



[BrDu10] Bruns, Dunkel: Event-Driven Architecture.  
 [EtNi11] Etzion, Niblett: Event Processing in Action

# Complex Event Processing (CEP)

Focus on *Complex Pattern Detection* [WuDR06]

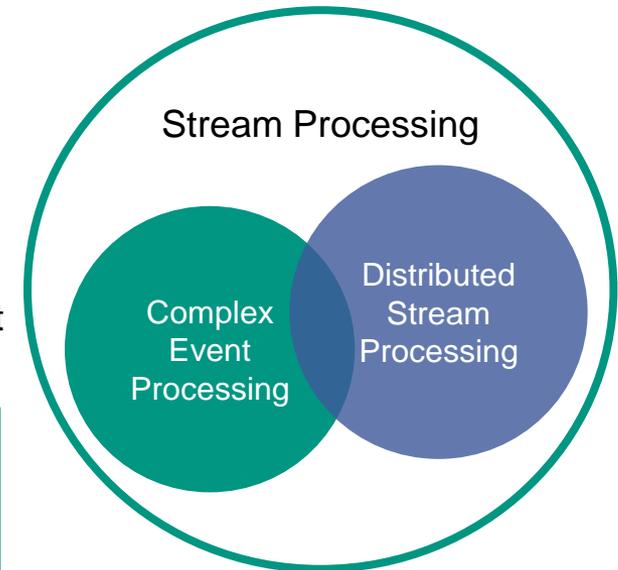
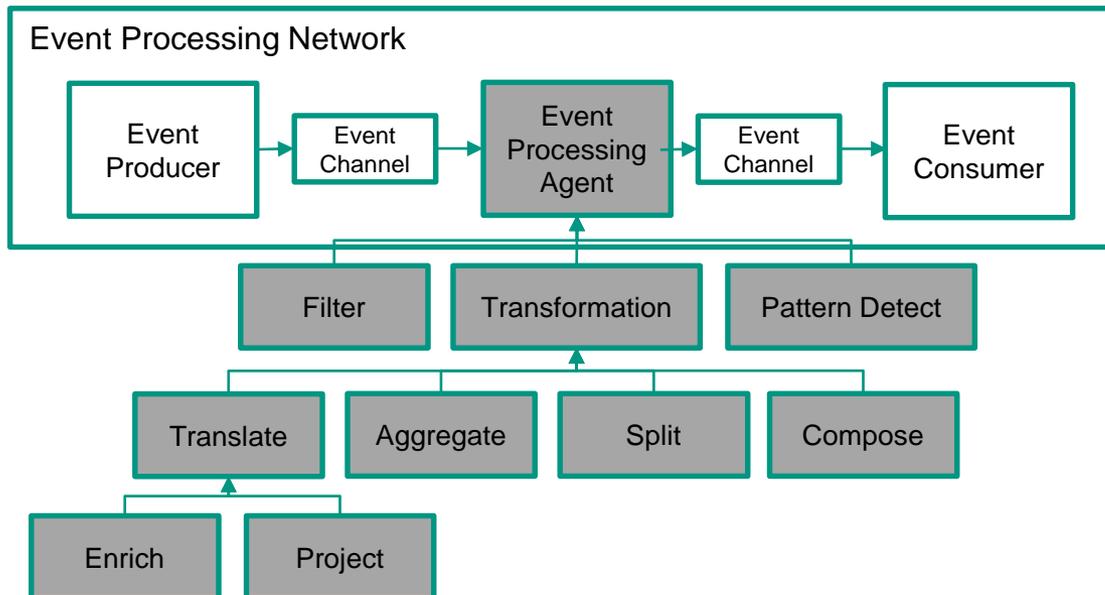
- e.g., absence of events, sequences, sliding windows

Event Processing Agent [EtNi11]

- Software component which processes events

Event Processing Network [EtNi11]

- Set of event producers, event processing agents and event consumers, connected through event channels



## Tools: Examples

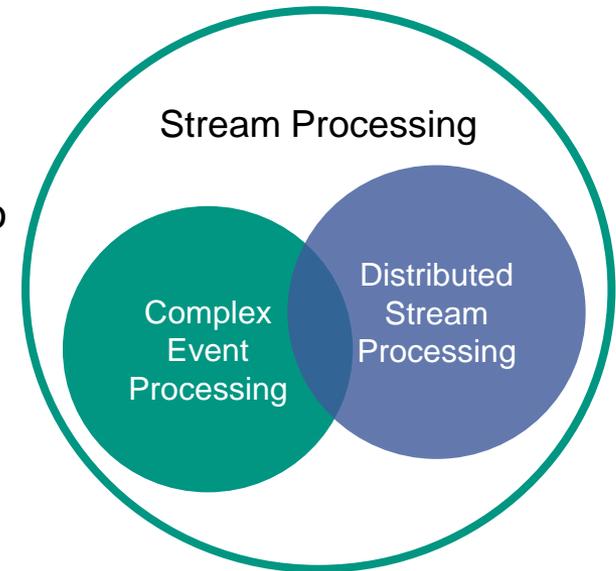


[WuDR06] Wu et.al.: High-Performance Complex Event Processing Over Streams  
 [EtNi11] Etzion, Niblett: Event Processing in Action

# Distributed Stream Processing

Focus on *Scalable Processing of Events* [CBBC+03]

- CEP-Systems usually single-host systems, leading to scalability issues in case of very high throughput [AGDT14]
- High-Level programming APIs

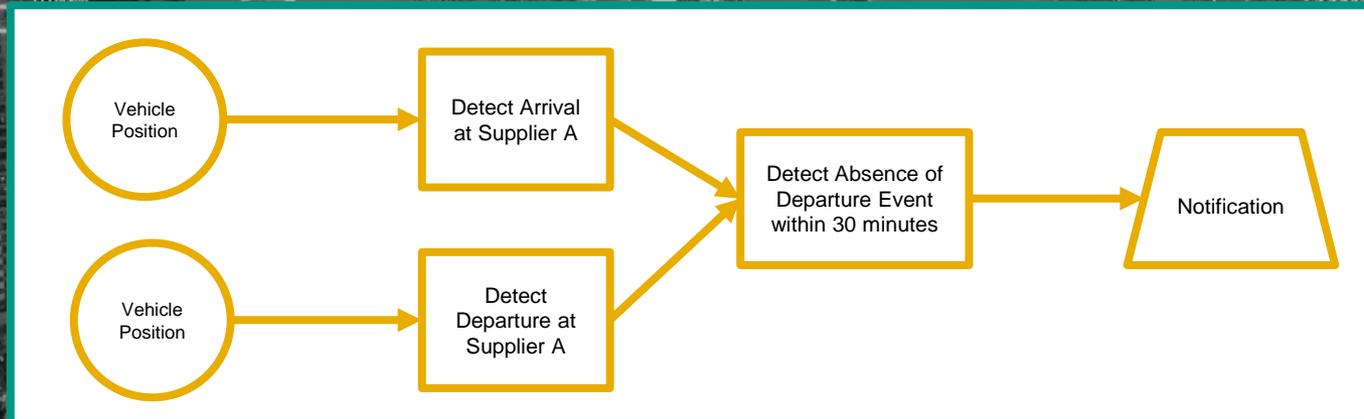


## Tools: Examples



[CBBC+03] Cherniack et.al.: Scalable Distributed Stream Processing.  
 [AGDT14] Andrade et.al.: Fundamentals of Stream Processing.

# A building block of IoT analytics applications: Flexible definition of real-time processing pipelines by application specialists



## Integrated Monitoring

Flexible data integration from heterogeneous sources

## Situational Awareness

Detection of situations and failures based on Complex Event Processing (CEP)

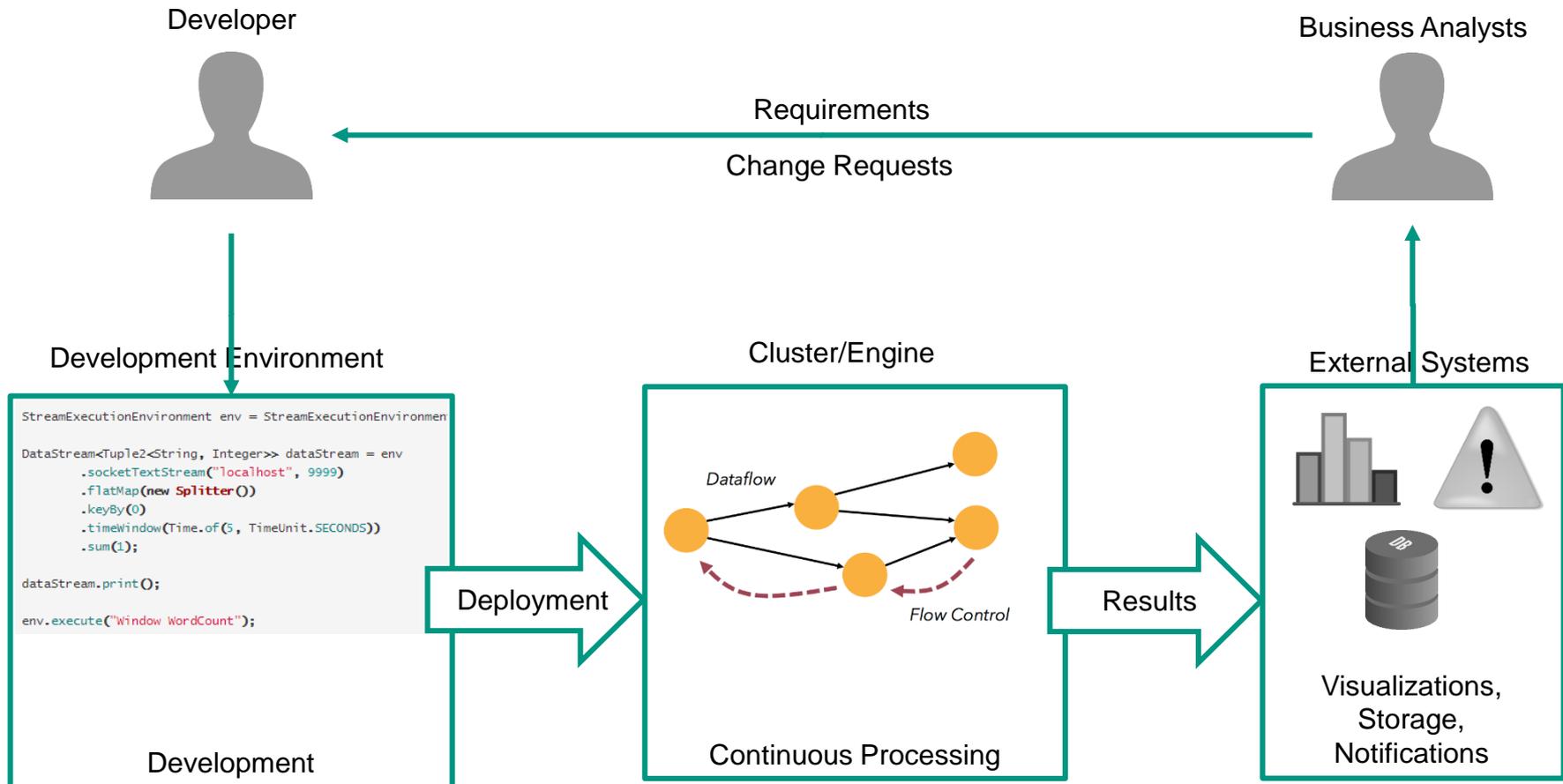
## Continuous Data Harmonization

Continuous pre-processing and data harmonization for third-party systems

# Outline

- Introduction & Motivation
- **Problems**
- Development methodology for event-driven applications
  - Overview
  - Modeling of data streams and processing elements
  - Definition and execution of distributed processing pipelines
- Implementation and evaluation
- Conclusion

# Problem: Slow Development Cycles



# Problem: Slow Development Cycles

Entwickler



Entwicklungs-  
umgebung

```
StreamExecutionEnvironment env = StreamExecutionEnvironment.createLocalEnvironmentWithDrainIO(1);  
  
DataStream<Tuple2<String, Integer>> dataStream = env  
    .socketTextStream("localhost", 9999)  
    .flatMap(new Splitter())  
    .keyBy(0)  
    .timeWindow(Time.of(5, TimeUnit.SECONDS))  
    .sum(1);  
  
dataStream.print();  
  
env.execute("Window WordCount");
```

Detect Arrival

Detect  
Absence of  
Departure

## Complex Event Processing

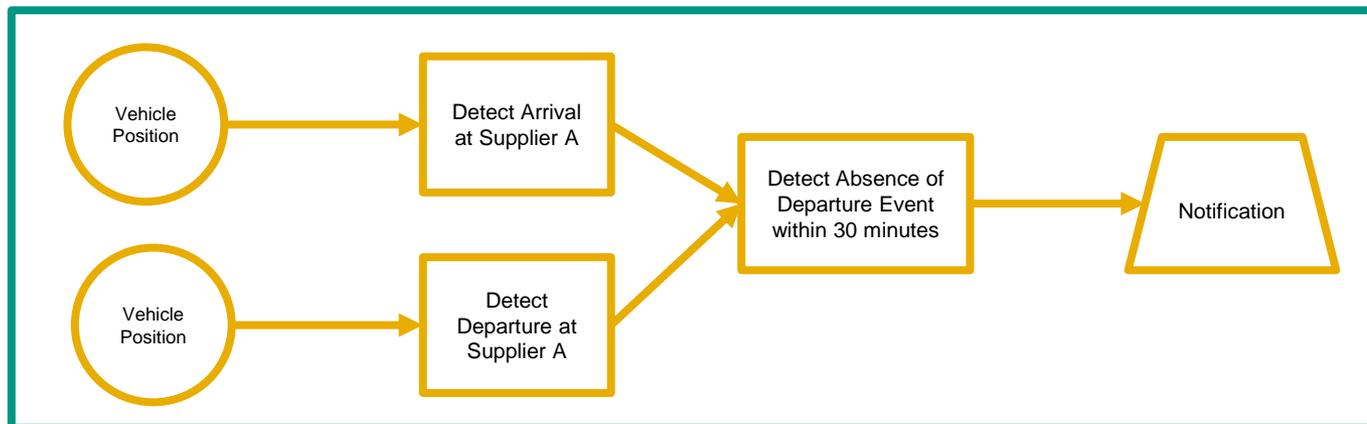
Register event types

```
insert into ArrivalEvent  
select a.timestamp, a.vehicleId, getSupplier(b.latitude,  
b.longitude) from pattern[every (a=PositionEvent ->  
b=PositionEvent)]  
where (!a.isInside(a.lat, a.lng, 49.06, 8.5, 500) and  
b.isInside(b.lat, b.lng, 49.23, 4.27, 500) and  
b.vehicleId=a.vehicleId)
```

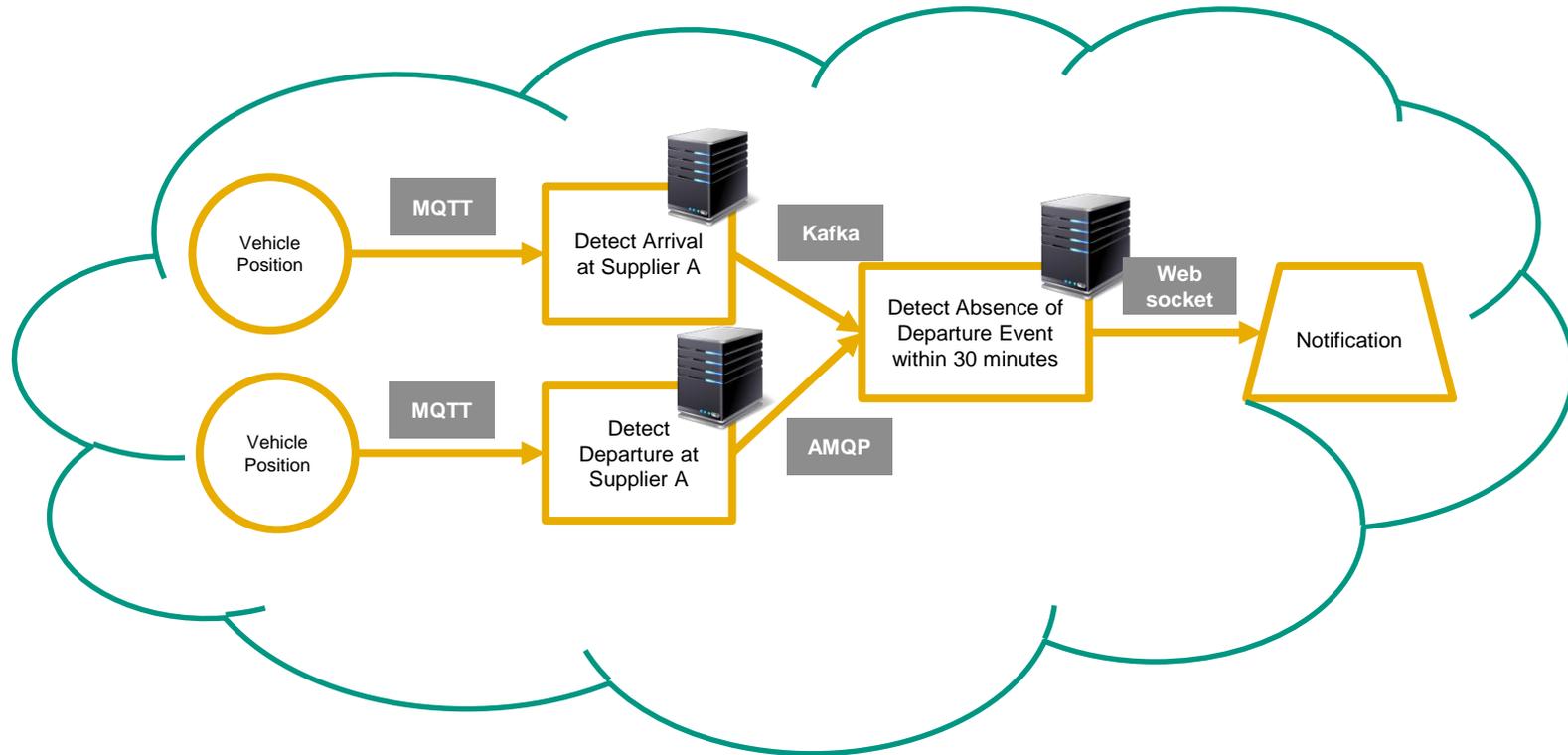
```
select a.vehicleId, (current_timestamp - a.timestamp) as time  
from pattern[every a=ArrivalEvent -> timer:interval(30  
minutes) and not b=DepartureEvent] where  
a.vehicleId=b.vehicleId
```

Implement output adapter

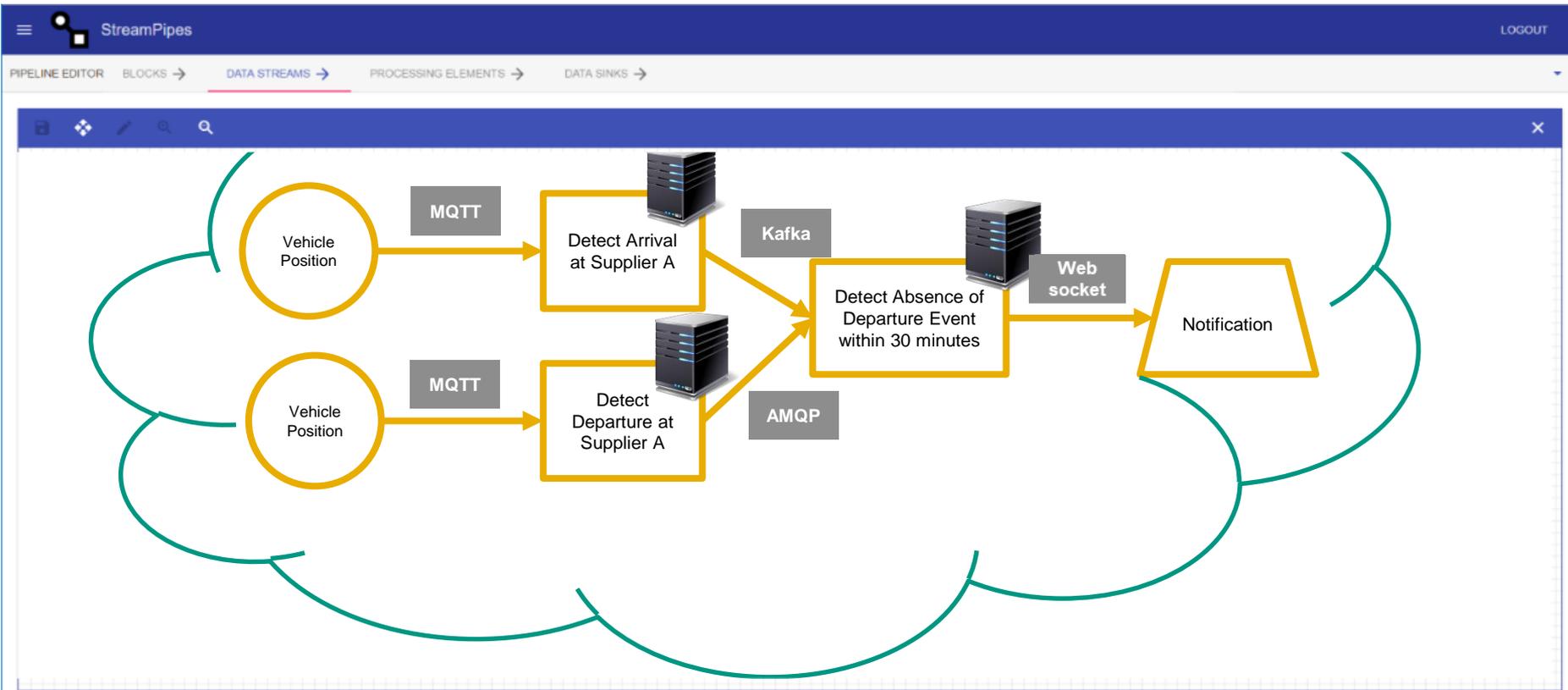
# Requirement: Interoperability



# Requirement: Interoperability



# Requirement: Interoperability



# Problems: Summary

## Observation

- frequent changes of applications required due to:
  - semantic/syntactic changes of event producers
  - new/changing requirements of application specialists
- high effort needed due to slow development cycles
- demand for „**Self-Service Data Analysis**“

*“**Analysts** should be able to process streaming data to gain insight - and once insight has been gained, to **easily refine the processing pipelines for even more insight** or to switch their focus of attention completely **without much latency.**”*

**otto group**

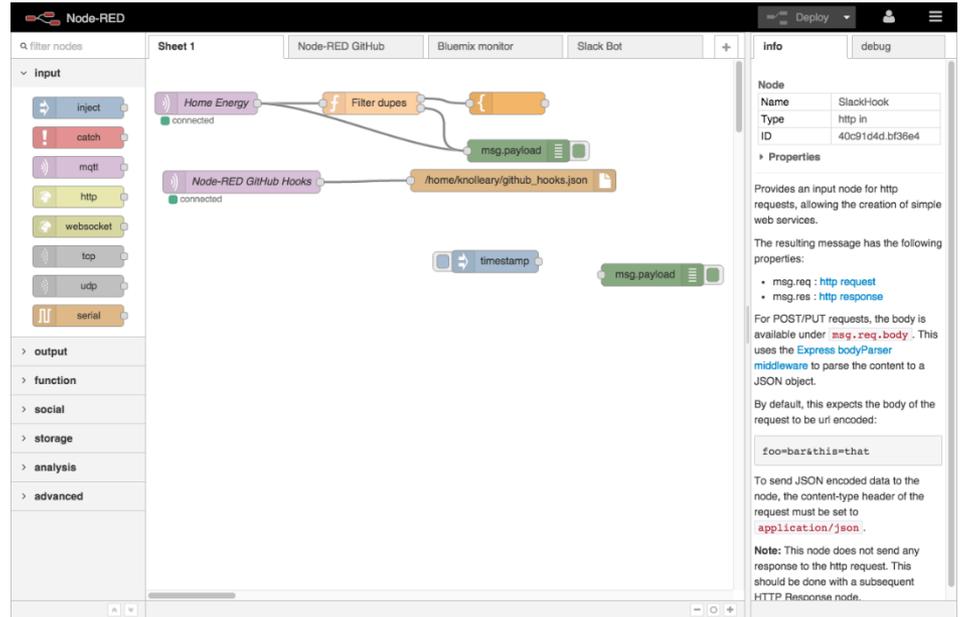
# Example: Node-Red

## Node-Red

- "Wiring the Internet of Things", initially developed by IBM Research

## Differences

- Single-Host system (no distribution of operators)
- Basic consistency checking (based on datatypes only)
- HTML-based configuration
- Relies on JavaScript runtime



The screenshot shows the Node-RED web interface. On the left, there is a sidebar with a search bar and a list of nodes categorized into input, output, function, social, storage, analysis, and advanced. The main workspace displays a flow with several nodes: 'Home Energy' (input), 'Filter dupes' (function), 'msg.payload' (output), 'Node-RED GitHub Hooks' (input), and 'timestamp' (function). The right sidebar shows the 'info' panel for the selected 'StackHook' node, including its name, type, ID, and a detailed description of its properties and usage.

# Outline

- Introduction & Motivation
- Problems
- **Development methodology for event-driven applications**
  - **Overview**
    - Modeling of data streams and processing elements
    - Definition and execution of distributed processing pipelines
- Implementation and evaluation
- Conclusion

# Methodology: Phases

## Preparation

**Development and grounding of re-usable  
event processing components**

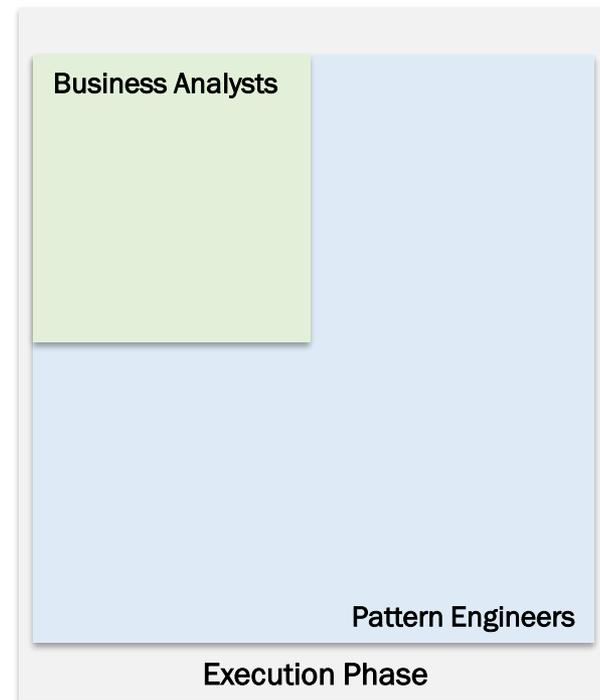
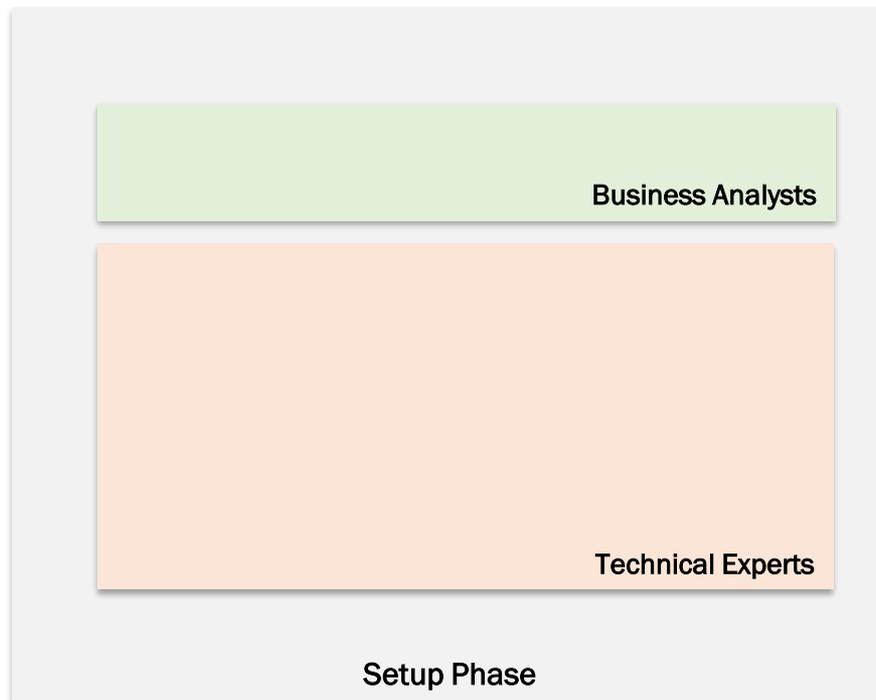
Setup Phase

## Execution

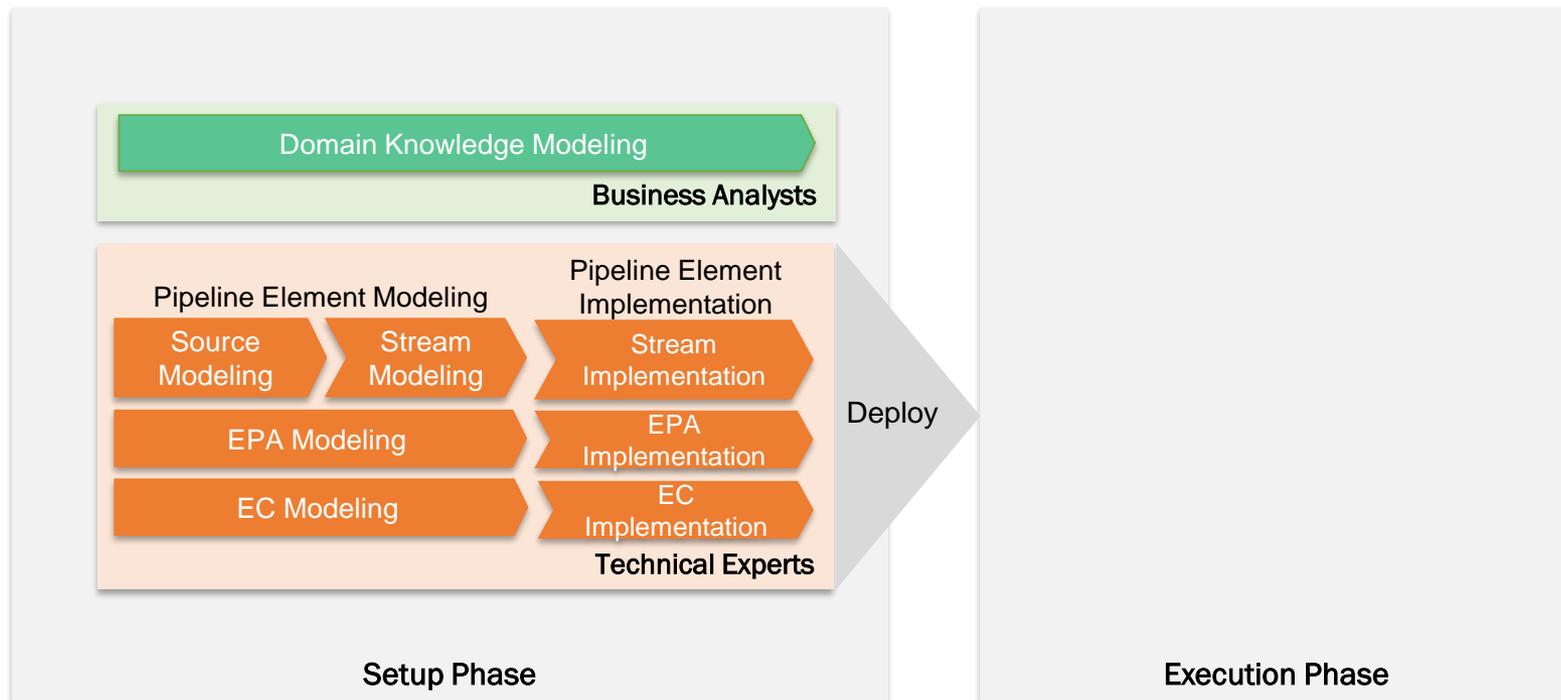
**Definition and deployment of  
processing pipelines based  
on re-usable components**

Execution Phase

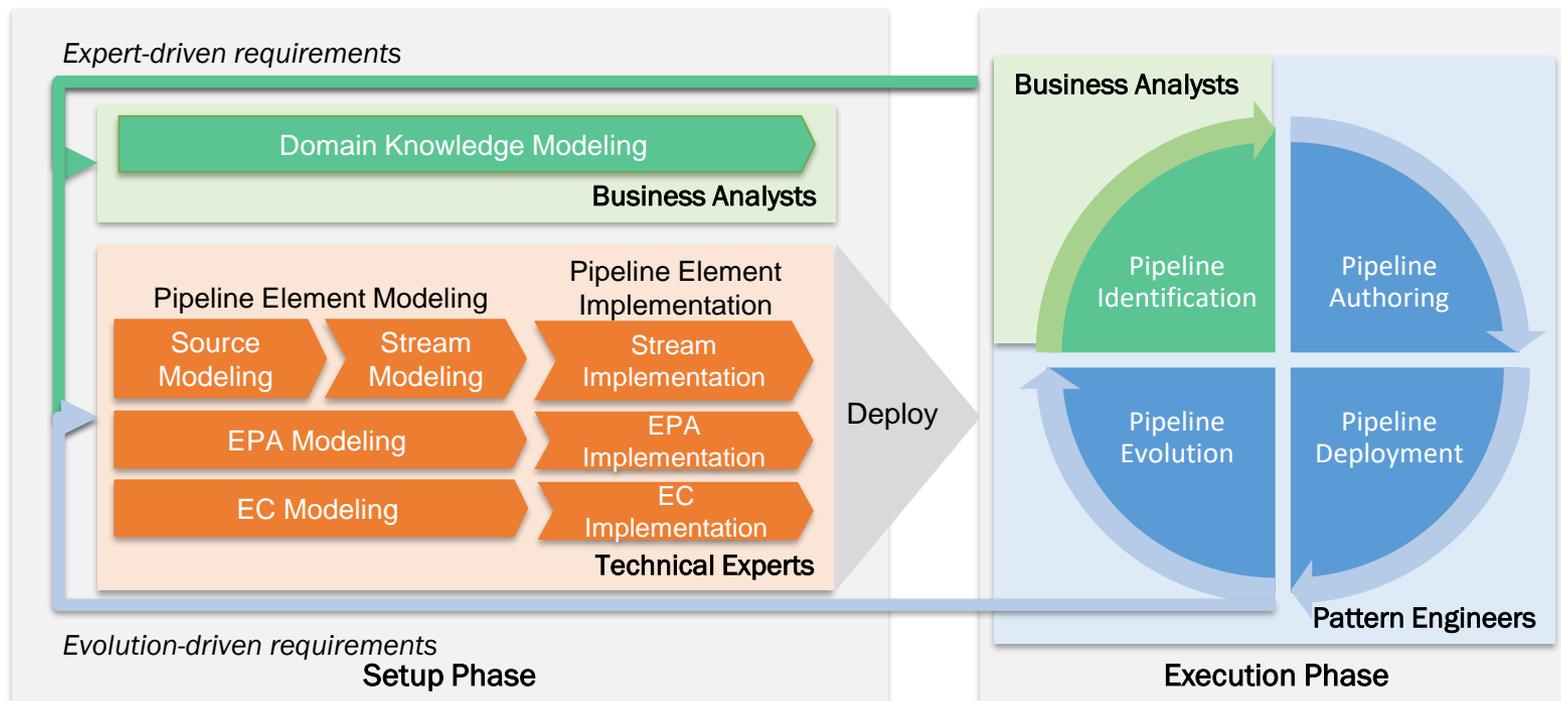
# Methodology: Different Roles



# Methodology: Two Tasks in Setup Phase

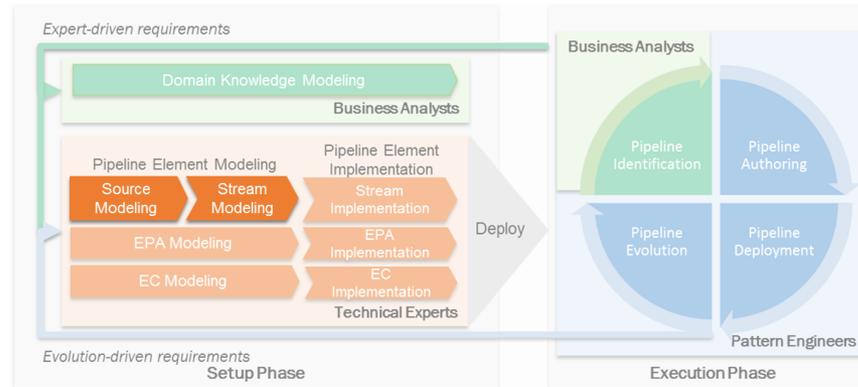


# Methodology: Tasks

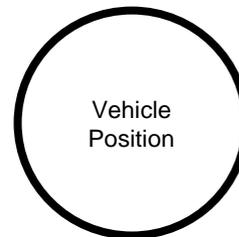


# Outline

- Introduction & Motivation
- Problems
- **Development methodology for event-driven applications**
  - Overview
  - **Modeling of data streams and processing elements**
  - Definition and execution of distributed processing pipelines
- Implementation and evaluation
- Conclusion



## Semantic Modeling of Data Streams



# Related Work: Semantic Sensor Network Ontology (SSN)

## Scope

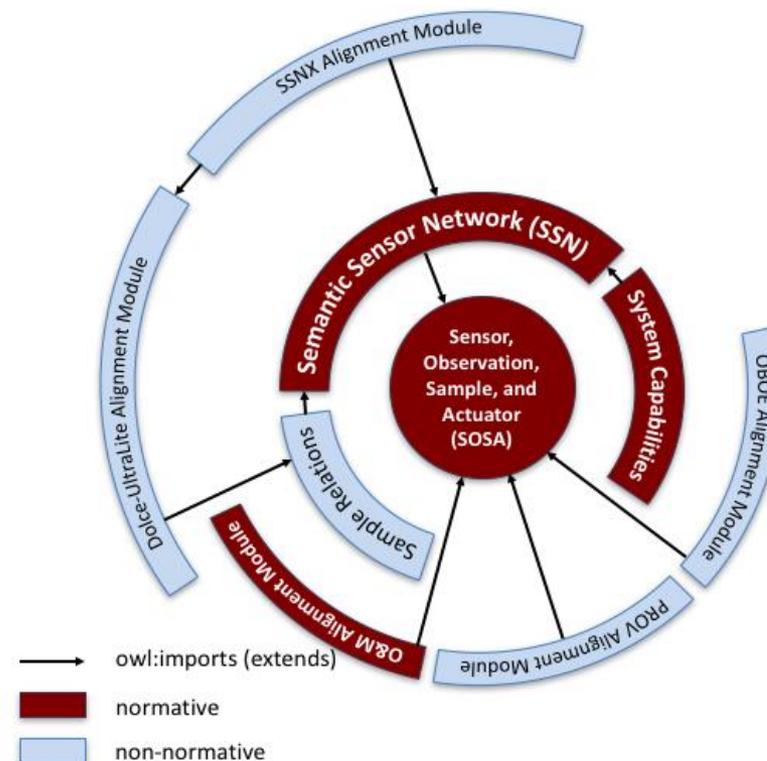
Define capabilities of sensors and sensor networks

## Vocabulary

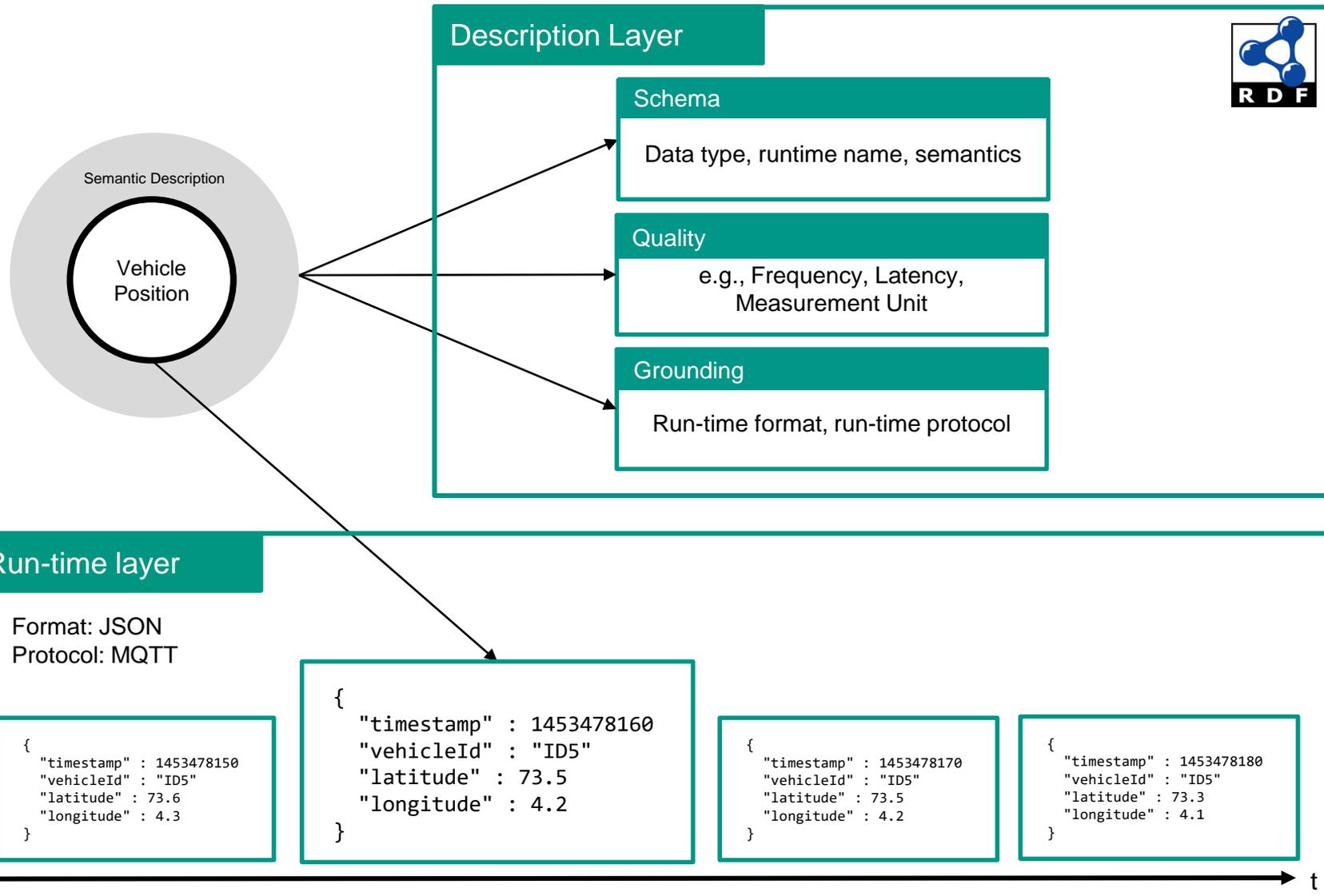
- Provides ways to model platforms, sensors and observations
- Aligned with upper ontologies
- modular (only observations, only sensors)
- Can be extended (e.g., with measurement units)

## Our approach

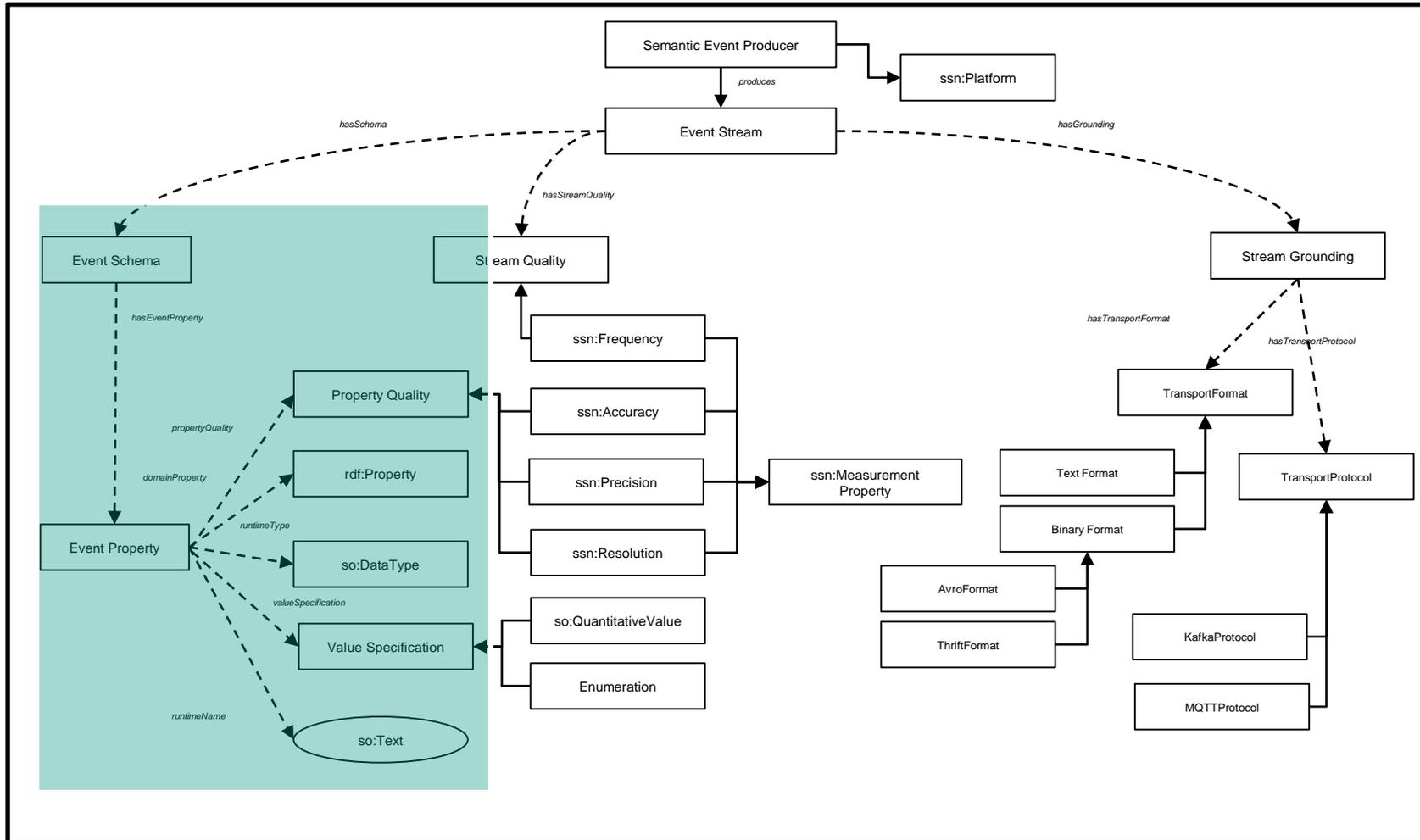
- Reuse parts of SSN (e.g., qualities)
- Use RDF only as metadata description to streams
- Use existing serialization formats for event transmission



# Stream Modeling: Requirements



# Stream Modeling: Vocabulary



# Stream Modeling: Event Schema

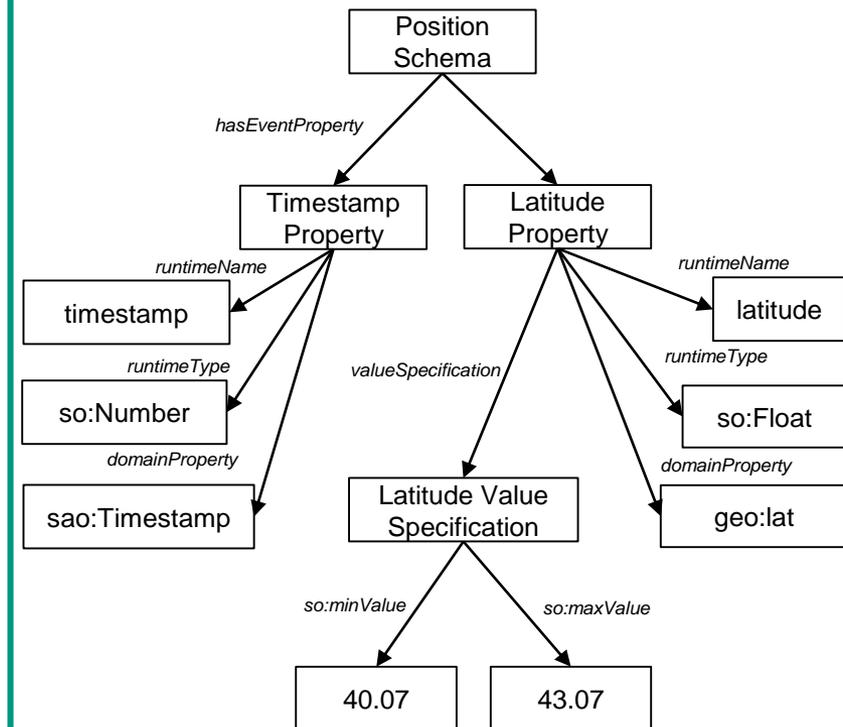
## Model

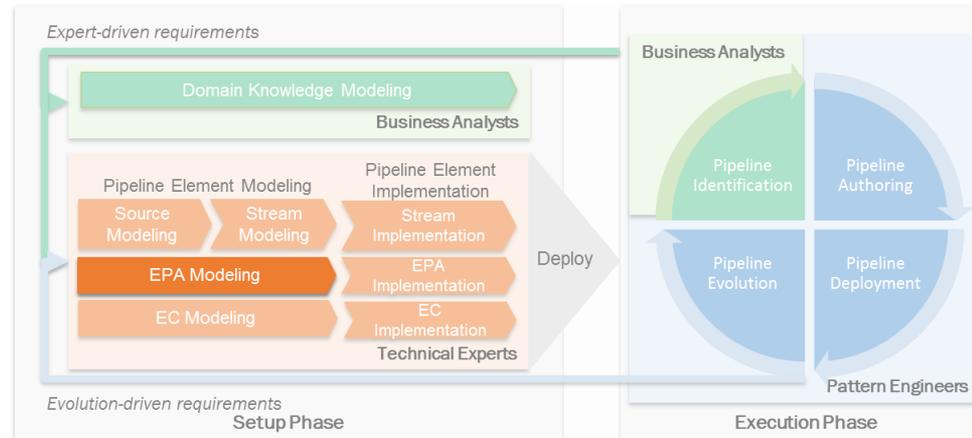
### Event Schema:

Description of the structure of the event at run-time

- **runtimeName**
  - Identifier of event property name in the run-time format
- **runtimeType**
  - Data type of the event property at run-time
- **domainProperty**
  - additional semantic description used for matching during pipeline definition
- **valueSpecification**
  - value specification of the event property
- **measurementUnit**
  - Measurement unit of the event property
- **propertyQuality**
  - Property-specific quality attributes, e.g., accuracy

## Example

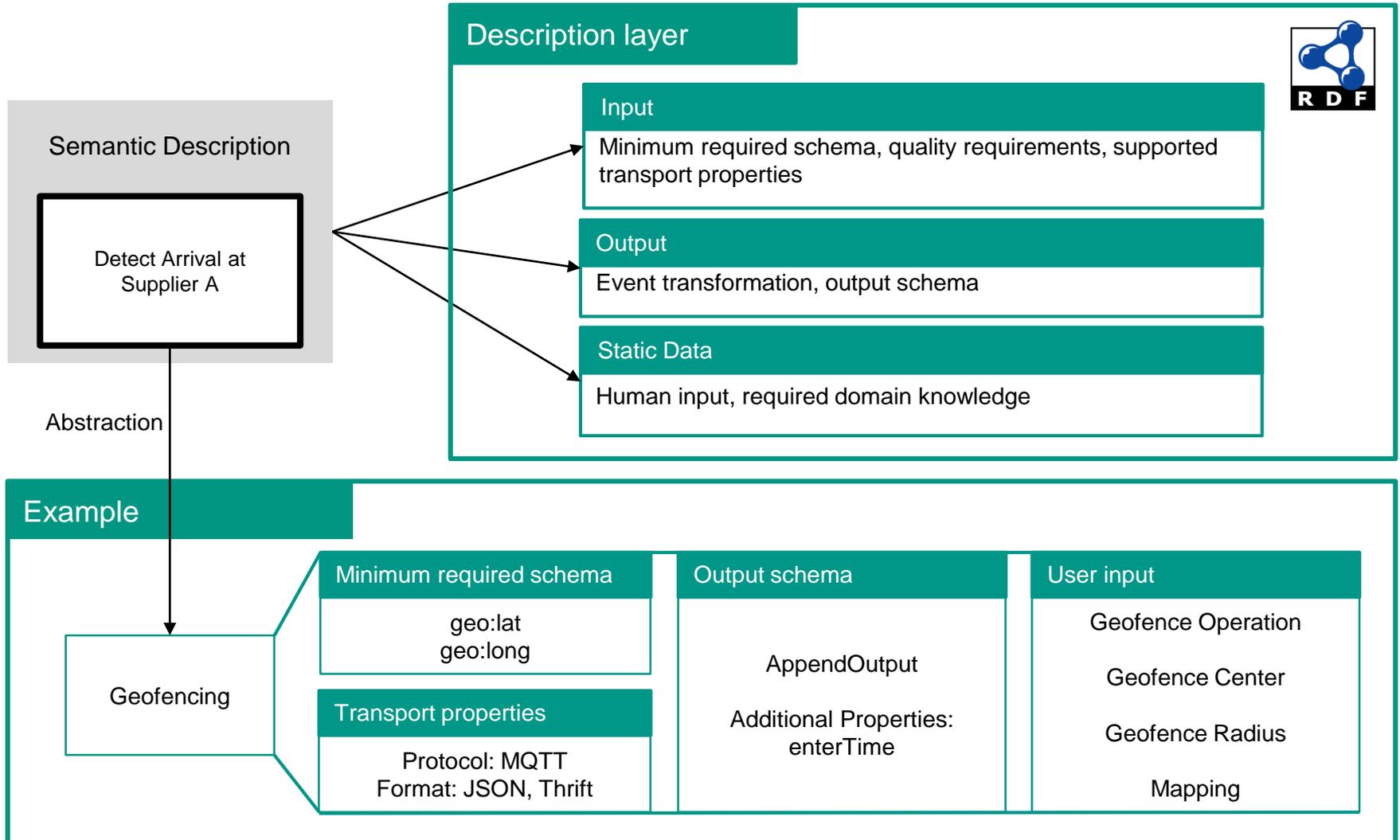




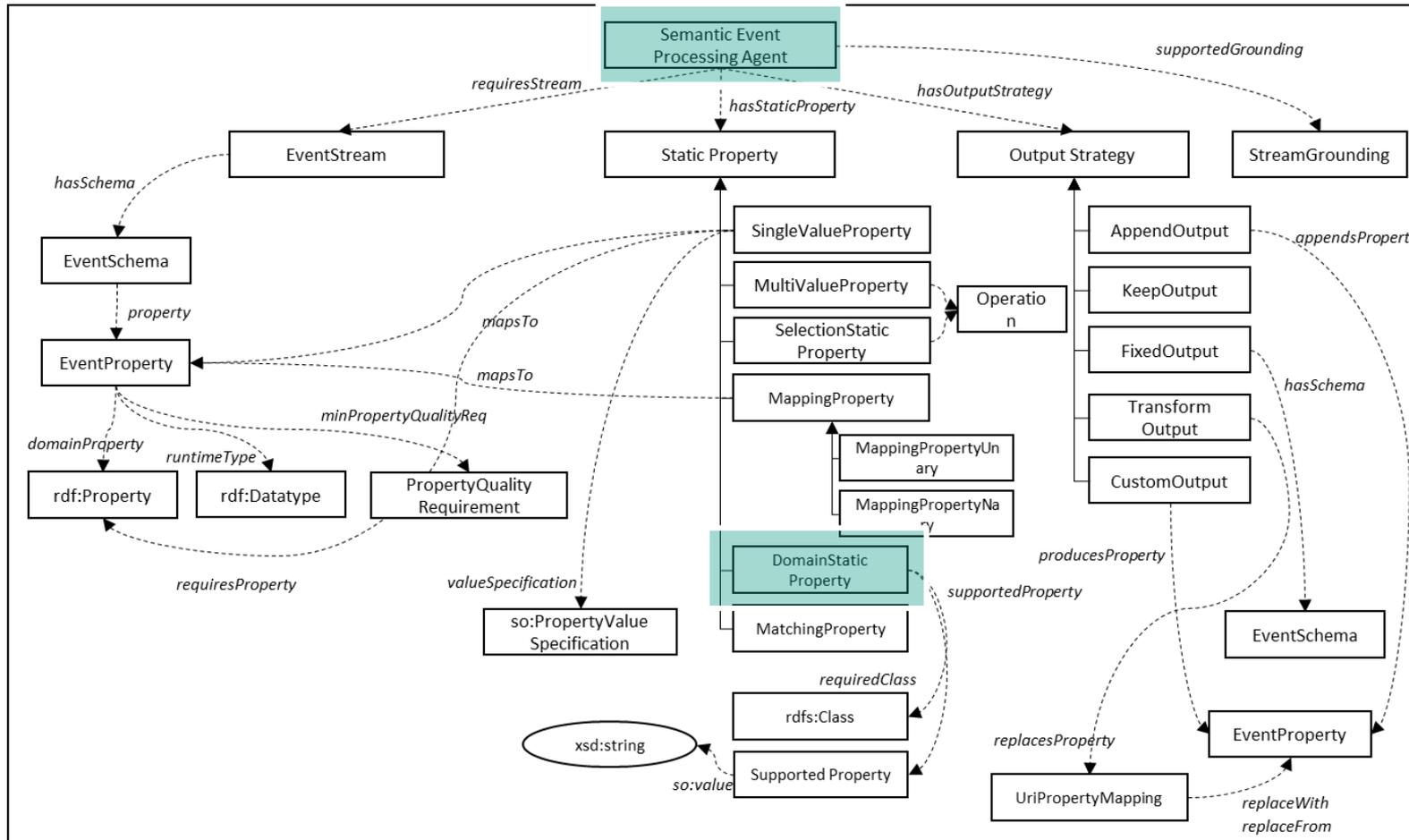
## Semantic Modeling of Event Processing Agents

Detect Arrival at  
Supplier A

# EPA Modeling: Requirements



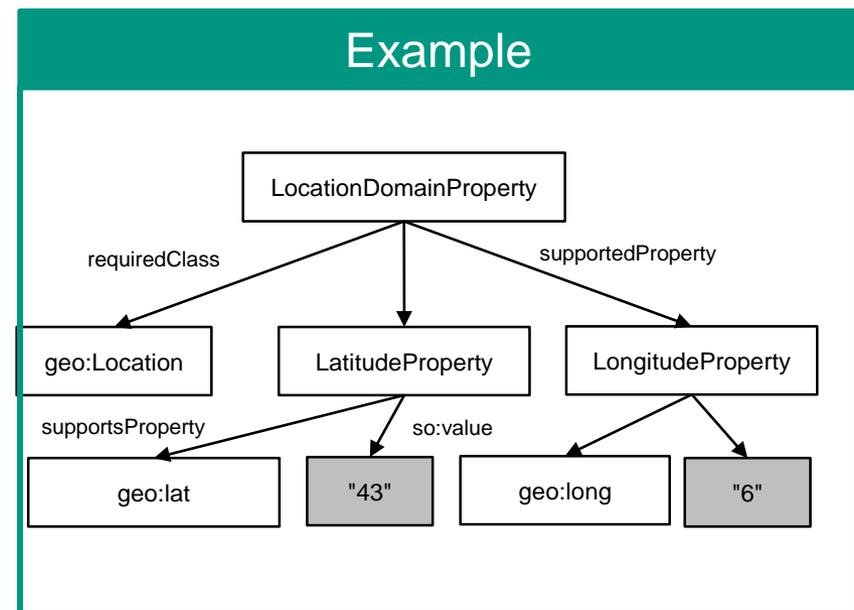
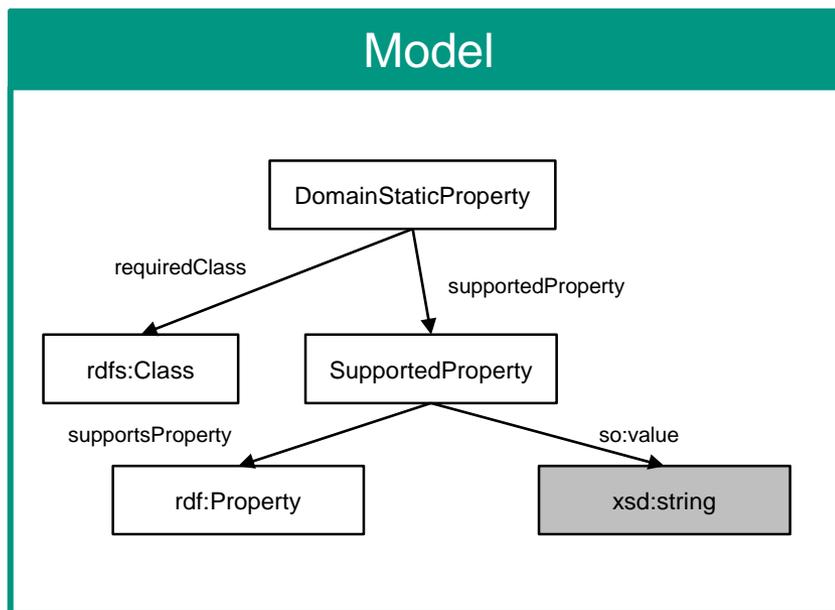
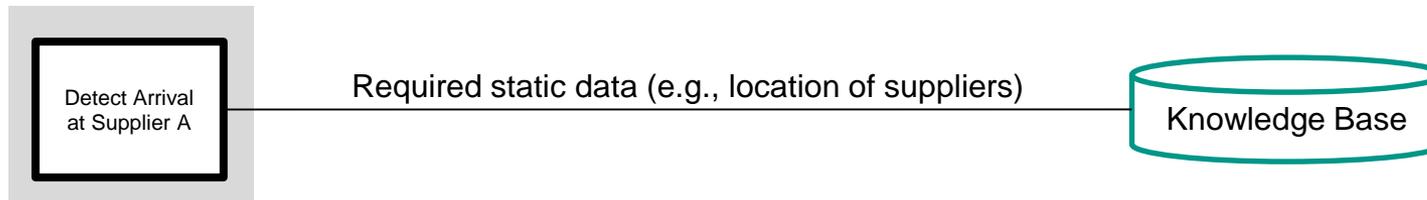
# EPA Modeling: Vocabulary



# EPA Modeling: Domain Knowledge

Example:

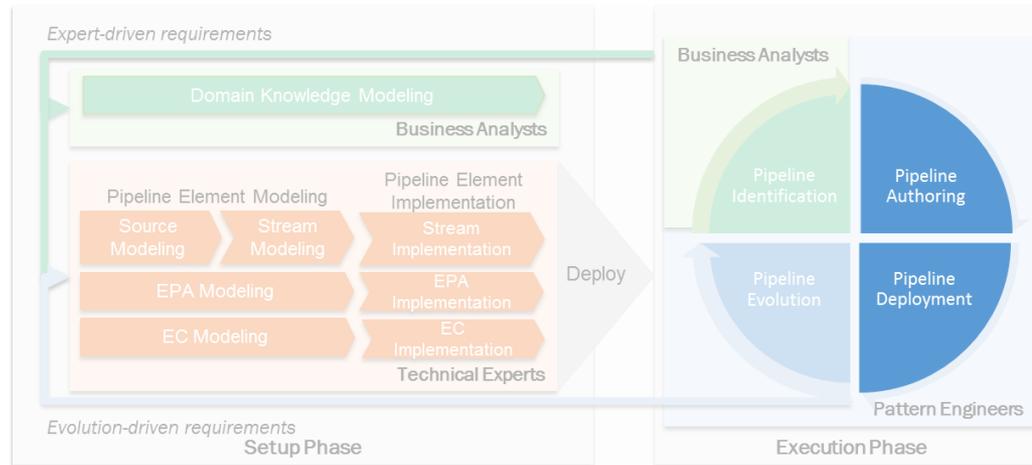
- The geofence EPA requires a coordinate acting as the geofence center
- Locations of suppliers are stored separately as domain knowledge



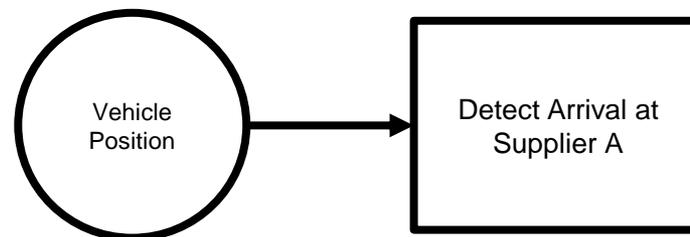
Provided in the setup phase
  Provided in the execution phase

# Outline

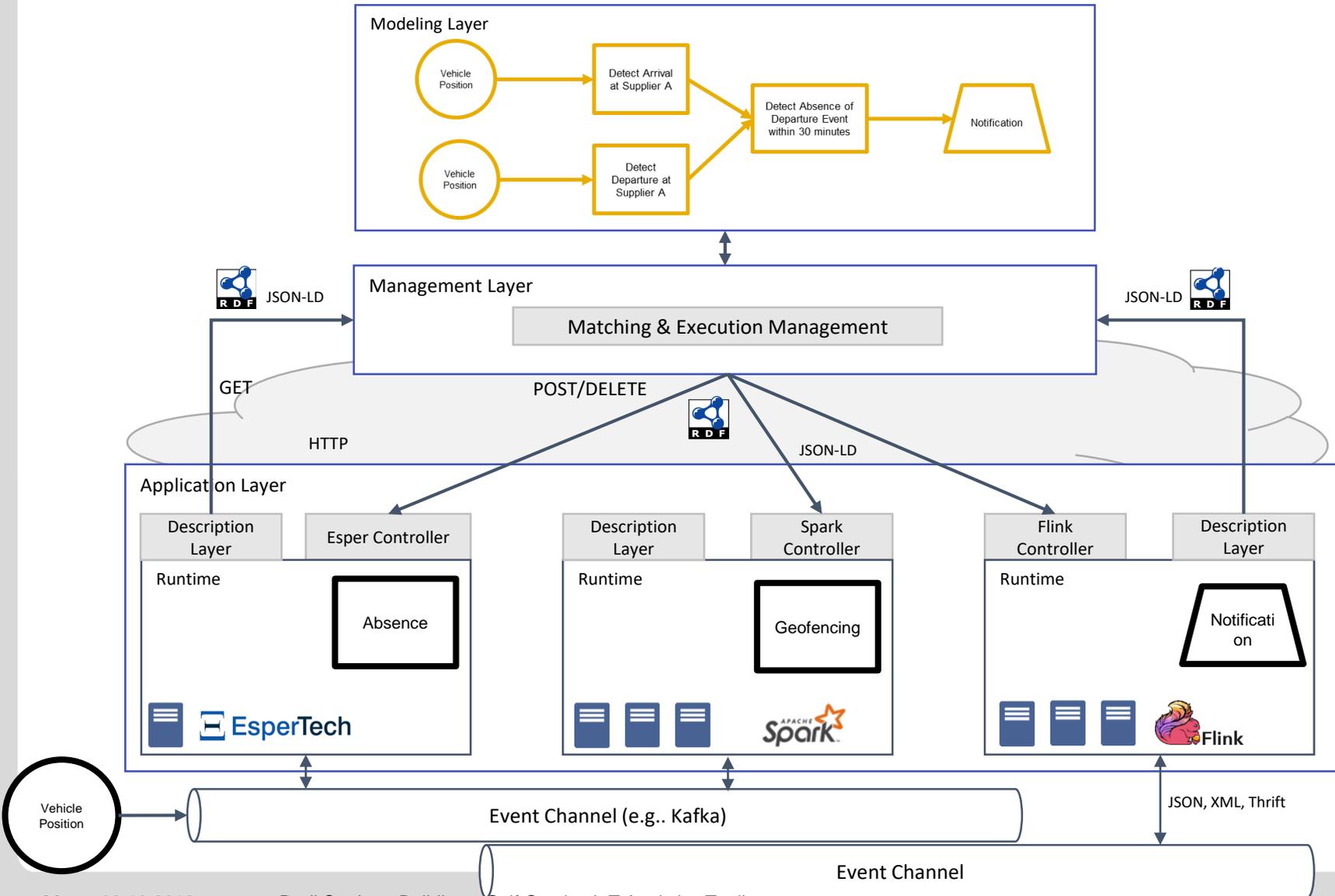
- Introduction & Motivation
- Problems
- **Development methodology for event-driven applications**
  - Overview
  - Modeling of data streams and processing elements
  - **Definition and execution of distributed processing pipelines**
- Implementation and evaluation
- Conclusion



# System



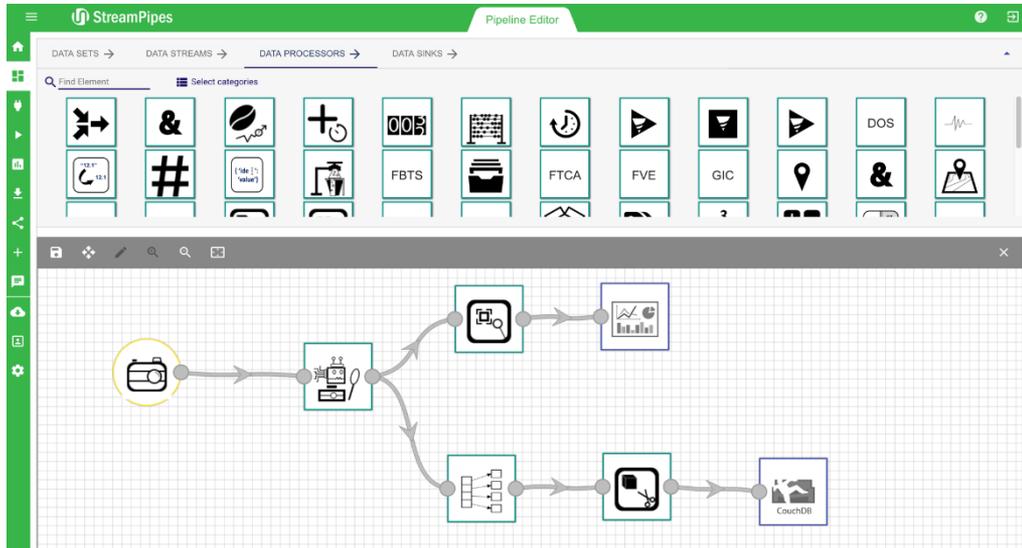
# Geo-distributed processing pipelines



# Outline

- Introduction & Motivation
- Problems
- Development methodology for event-driven applications
  - Overview
  - Modeling of data streams and processing elements
  - Definition and execution of distributed processing pipelines
- **Implementation and evaluation**
- Conclusion

# StreamPipes: Open Source Self-Service Analytics



Semantics-based modeling layer

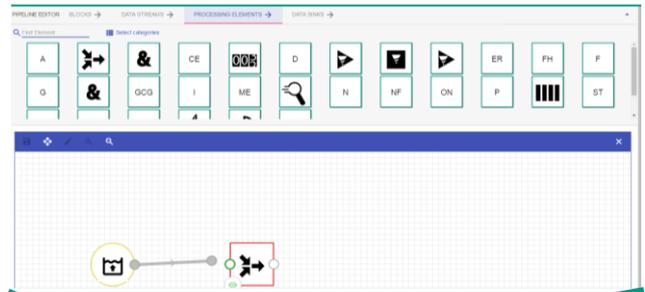
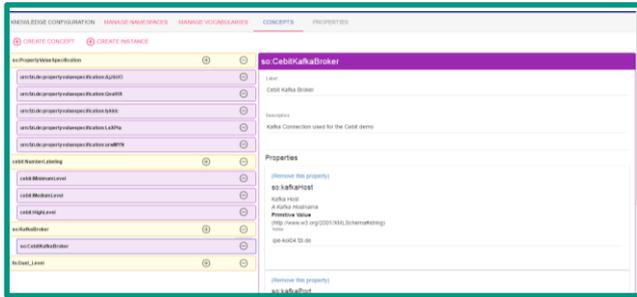
Arbitrary data formats and protocols

Geo-distributed execution

Exchangeable runtime wrappers  
(Single host or distributed)

Open Source: <https://docs.streampipes.org>

# Tool Support: StreamPipes



Expert-driven requirements

**1 Knowledge Editor**

**2 Description Model Editor**

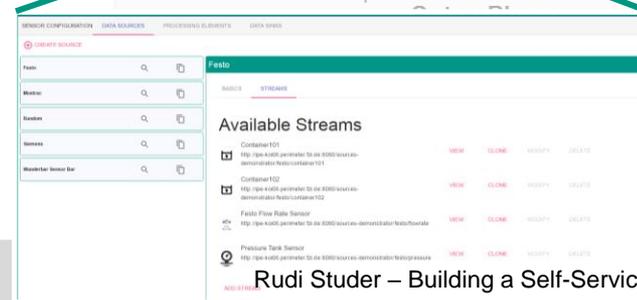
**3 SDK**

**4 Runtime Wrapper**

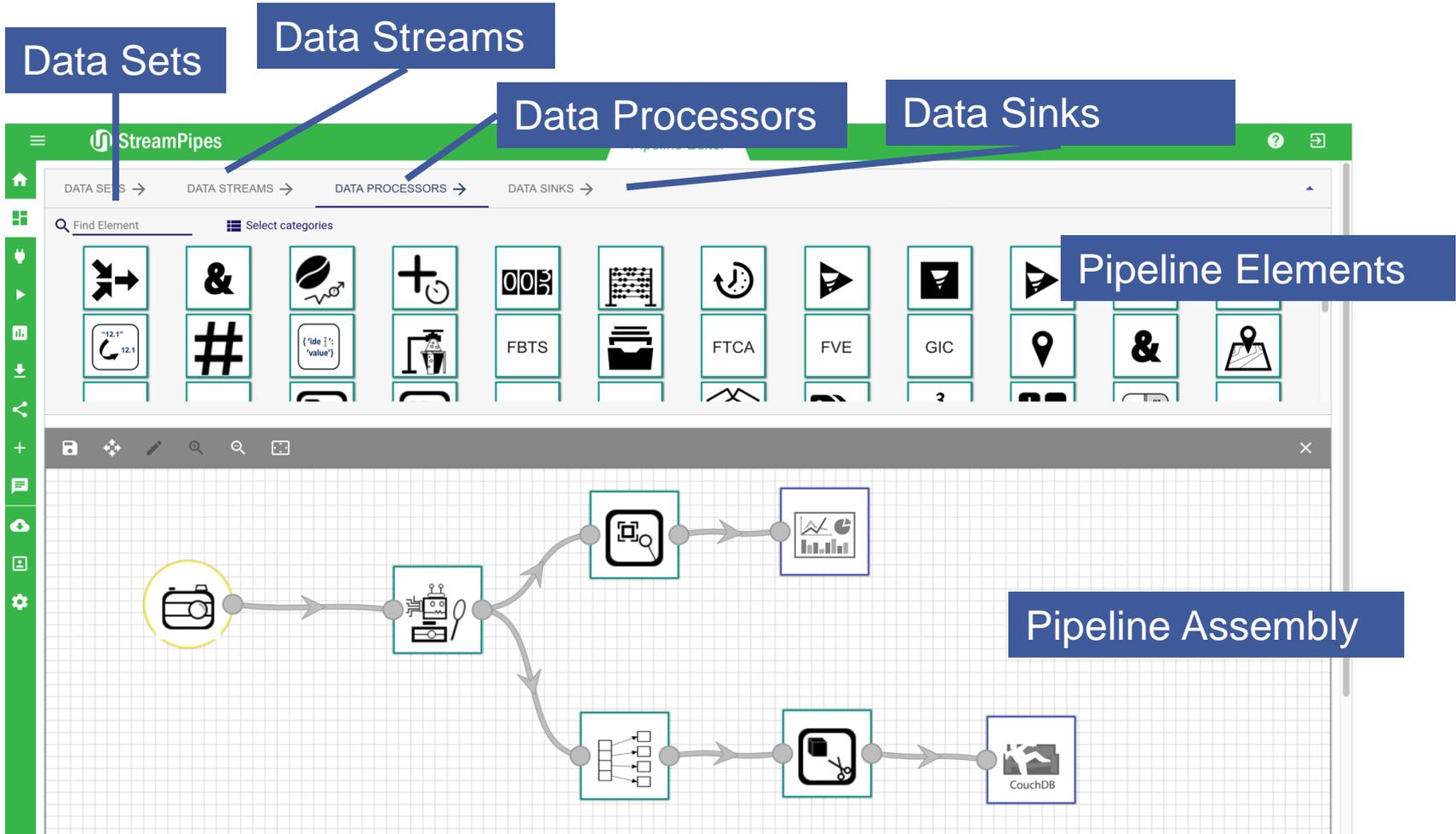
**5 Pipeline Authoring Tool**

**6 Integration & Execution Engine**

Evolution-driven requirements



# Pipeline Authoring Tool



**Data Sets**

**Data Streams**

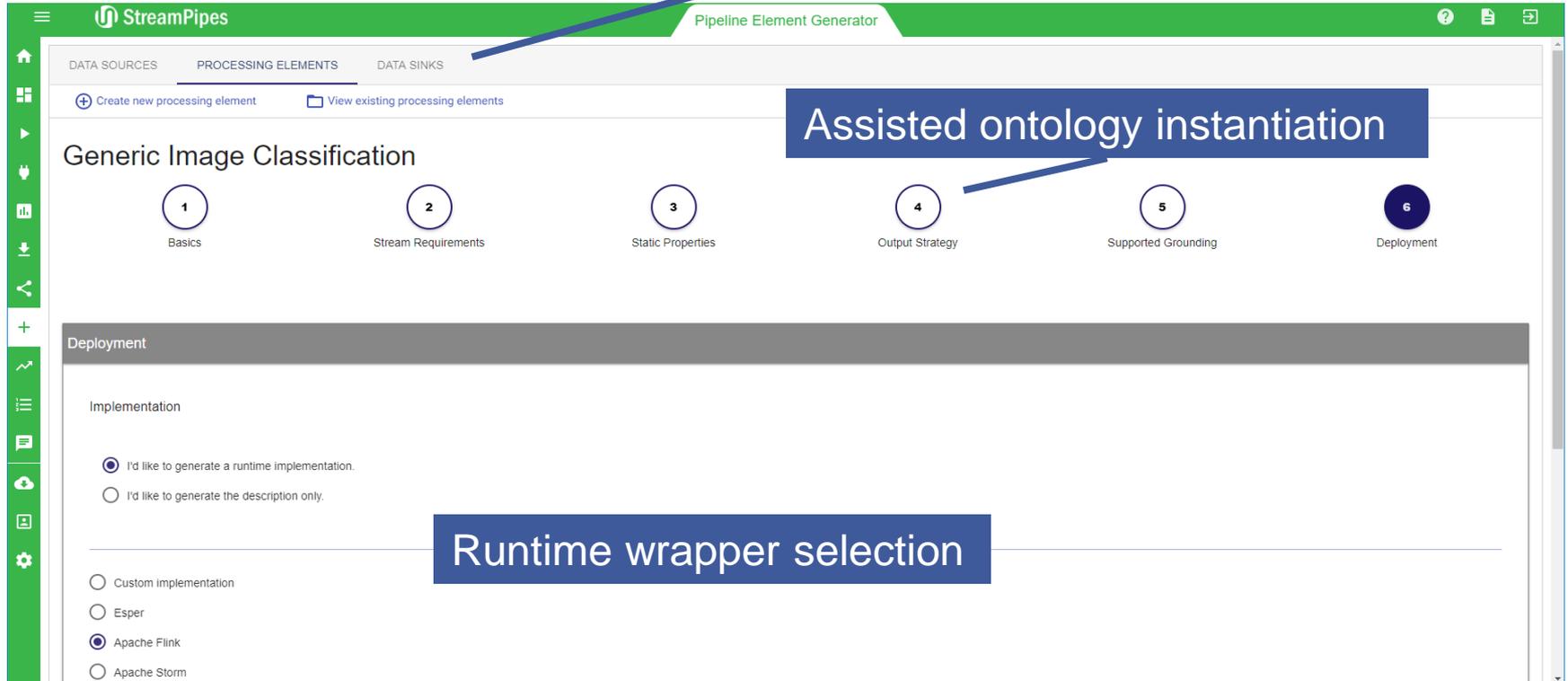
**Data Processors**

**Data Sinks**

**Pipeline Elements**

**Pipeline Assembly**

Selection of pipeline element type

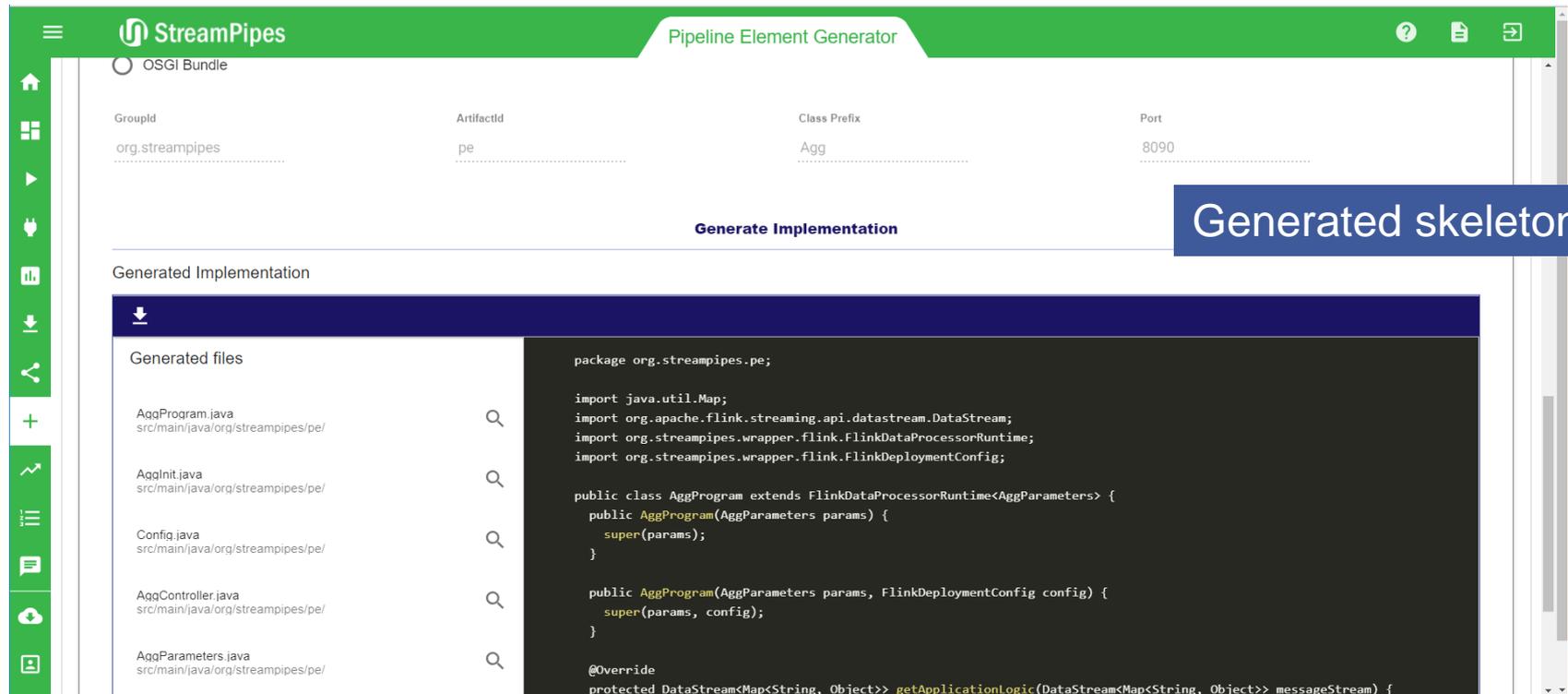


The screenshot shows the 'StreamPipes Pipeline Element Generator' interface. The top navigation bar includes 'DATA SOURCES', 'PROCESSING ELEMENTS', and 'DATA SINKS'. Below this, there are buttons for 'Create new processing element' and 'View existing processing elements'. The main content area is titled 'Generic Image Classification' and features six numbered steps: 1. Basics, 2. Stream Requirements, 3. Static Properties, 4. Output Strategy, 5. Supported Grounding, and 6. Deployment. The 'Deployment' step is currently selected and expanded, showing options for 'Implementation'. Under 'Implementation', there are two radio buttons: 'I'd like to generate a runtime implementation.' (which is selected) and 'I'd like to generate the description only.'. Below these are four radio buttons for 'Runtime wrapper selection': 'Custom implementation', 'Esper', 'Apache Flink' (which is selected), and 'Apache Storm'. A blue callout box labeled 'Assisted ontology instantiation' points to the 'Output Strategy' step. Another blue callout box labeled 'Runtime wrapper selection' points to the 'Apache Flink' option.

Assisted ontology instantiation

Runtime wrapper selection

# Model Editor



The screenshot shows the StreamPipes Model Editor interface. At the top, there is a green header with the StreamPipes logo and the title "Pipeline Element Generator". Below the header, there is a form for defining the pipeline element. The form contains the following fields:

GroupId	ArtifactId	Class Prefix	Port
org.streampipes	pe	Agg	8090

Below the form, there is a "Generate Implementation" button. A blue callout box points to this button with the text "Generated skeleton".

Below the button, there is a section titled "Generated Implementation". On the left, there is a list of generated files:

- AggProgram.java (src/main/java/org/streampipes/pe/)
- AggInit.java (src/main/java/org/streampipes/pe/)
- Config.java (src/main/java/org/streampipes/pe/)
- AggController.java (src/main/java/org/streampipes/pe/)
- AggParameters.java (src/main/java/org/streampipes/pe/)

On the right, there is a code editor showing the generated code skeleton for AggProgram.java:

```
package org.streampipes.pe;

import java.util.Map;
import org.apache.flink.streaming.api.datastream.DataStream;
import org.streampipes.wrapper.flink.FlinkDataProcessorRuntime;
import org.streampipes.wrapper.flink.FlinkDeploymentConfig;

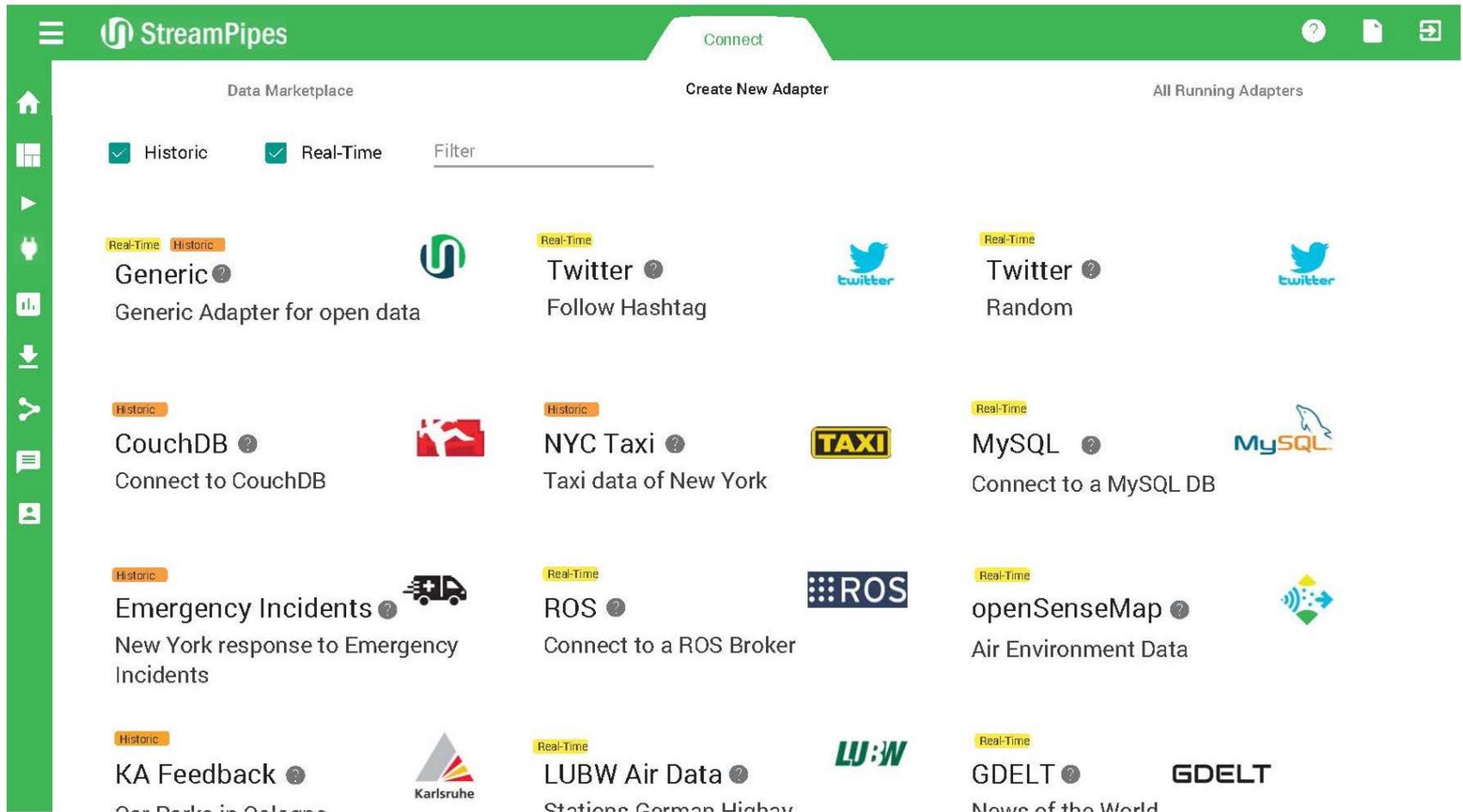
public class AggProgram extends FlinkDataProcessorRuntime<AggParameters> {
    public AggProgram(AggParameters params) {
        super(params);
    }

    public AggProgram(AggParameters params, FlinkDeploymentConfig config) {
        super(params, config);
    }

    @Override
    protected DataStream<Map<String, Object>> getApplicationLogic(DataStream<Map<String, Object>> messageStream) {
```

Only the application logic needs to be added to the generated code!

# Adapter Library: Access to existing data sources



The screenshot shows the StreamPipes Adapter Library interface. At the top, there is a green header with the StreamPipes logo and a 'Connect' button. Below the header, there are three tabs: 'Data Marketplace', 'Create New Adapter', and 'All Running Adapters'. The 'Data Marketplace' tab is active, showing a grid of adapters. Each adapter card includes a status indicator (Real-Time or Historic), a title, a description, and a logo. The adapters listed are:

- Generic** (Real-Time, Historic): Generic Adapter for open data
- Twitter** (Real-Time): Follow Hashtag
- Twitter** (Real-Time): Random
- CouchDB** (Historic): Connect to CouchDB
- NYC Taxi** (Historic): Taxi data of New York
- MySQL** (Real-Time): Connect to a MySQL DB
- Emergency Incidents** (Historic): New York response to Emergency Incidents
- ROS** (Real-Time): Connect to a ROS Broker
- openSenseMap** (Real-Time): Air Environment Data
- KA Feedback** (Historic): Open Feedback in Karlsruhe
- LUBW Air Data** (Real-Time): Stations German Highway
- GDELT** (Real-Time): News of the World

# Self-Service Analytics with StreamPipes

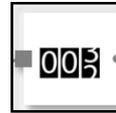
## Example pipeline elements

### Data Streams



- **Sensors**
  - Machines (e.g., OPC)
  - MES
  - Environmental Data
- **Enterprise Applications**
  - Production plans
  - Databases
  - Master data
- **Human Sensors**
  - E.g., mobile applications

### Data Processors



- **Complex Event Processing**
  - Pattern Detection
  - Aggregation
  - Filter
- **Advanced Analytics**
  - Vibration Detection
  - Predictions
  - Online Classification
- **Transformations**
  - Enrichment
  - Conversion
  - Replacement

### Data Sinks



- **Visualization**
  - Charts
  - Geo
  - Tables
- **Notifications**
  - E-Mail
  - Dashboard
- **Storage / Messaging**
  - Elasticsearch
  - Kafka
  - HDFS
- **Actuators**

## Overview

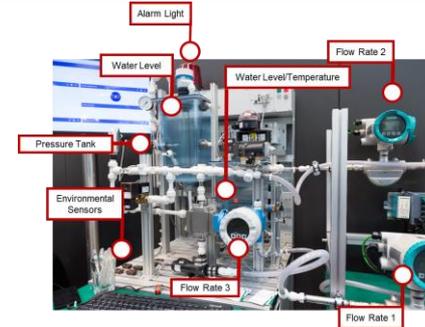


**Production**

> 100 implemented pipeline elements



**ACM DEBS**



**smartAutomation**

## DEBS Grand Challenge

- Common evaluation testbed for event-based systems
- Dataset: New York City Taxi Data (130 M Events)
- ask: geospatial real-time analytics (profitable areas, frequent routes)

## Results

- Re-usable pipeline elements
  - Distance
  - Spatial Aggregations
  - Ranking
- Solved Grand Challenge based on re-usable pipeline elements and pipelines
- Compared performance to baseline (single host and distributed systems)

# Use Case: Image Processing for Logistics

## Task

- Flexible quality tracking for inbound logistics
- Detect quality problems based on
  - Cheap IoT sensors (e.g., vibration)
  - Image data from stationary cameras
- Provide transport planners with a flexible solution to analyse various KPIs

## Sensors

- Cameras
- IoT sensors (Bosch XDK, proprietary sensors)
  - Temperature
  - Light
  - Acceleration

## Pipelines

- Take pictures of incoming products, classify them using deep neural networks and recognize quantity deviations
- Automatically read and recognize parcel labels
- Get insights on proper handling of sensitive products during transport

# Use Case: Production Monitoring (3D Printing)

## Task

- Quality monitoring of industrial-grade 3D printers with a large German manufacturer
- Correlate environment data to product quality
- Constantly monitor 3D printing parameters from different machines

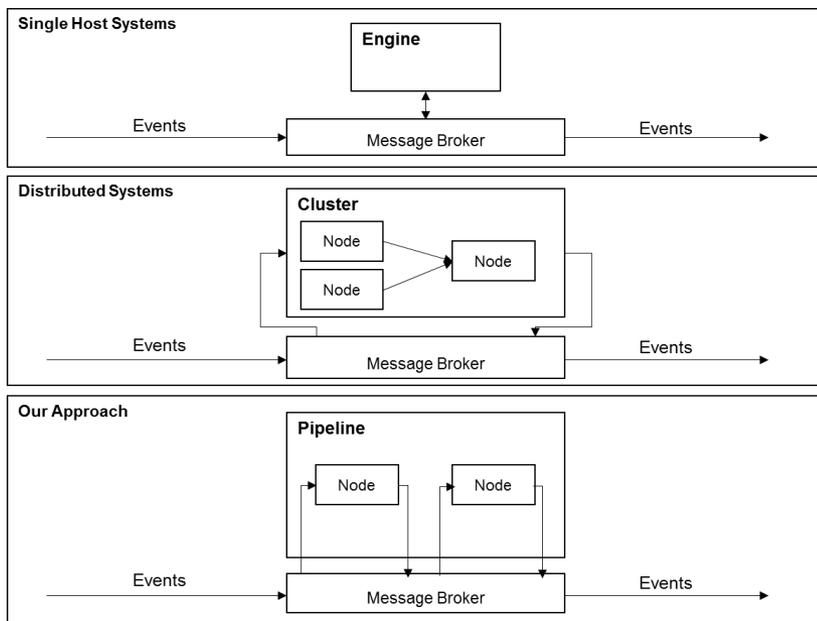
## Sensors

- 3D printers
  - Humidity
  - Temperature
  - Machine settings
- Environmental Sensors
  - Temperature
  - Humidity

## Pipelines

- Constant monitoring: Environmental parameters are very critical for production outcome
- Early detection of potential quality issues based on correlations of internal & external sensors
- Benchmarking (compare outcome to other plants/facilities)

## Performance compared to single host systems

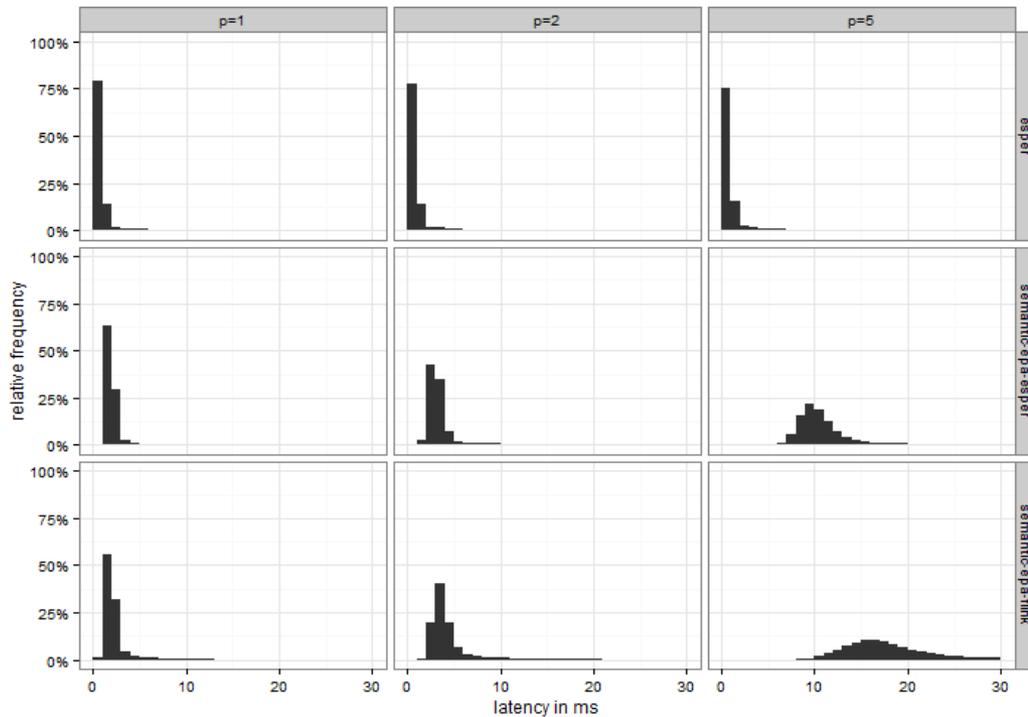


## Setting

- **Hard- and software**
  - 3 Servers (24GB RAM, 12GB, 8GB)
  - CPU: 4x 2,3 Ghz
  - Kafka, Flink, Esper, ActiveMQ
- **Configurations**
  - esper: Single-host system
  - semantic-epa-esper: Distributed system (esper nodes)
  - semantic-epa-flink: Distributed system (flink nodes)
- **Experiment**
  - Pipeline size (p) 1, 2, 5
  - 1000 events/sec, 5000 events/sec
  - 100.000 events

# Performance

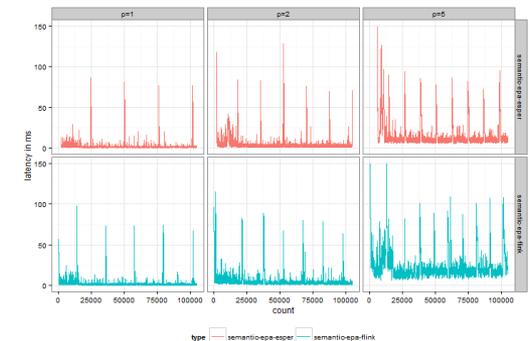
## Performance: Latency



config	p=1	p=2	p=5
esper	1	1	1
semantic-epa-esper	2	3	10
semantic-epa-flink	2	4	16

latency [ms]

## Performance: Latency/Time



# Outline

- Introduction & Motivation
- Problems
- Development methodology for event-driven applications
  - Overview
  - Modeling of data streams and processing elements
  - Definition and execution of distributed processing pipelines
- Implementation and evaluation
- **Conclusion**

## Summary

- **Self-Service Analytics Toolbox for the IoT**
  - **Methodology:** novel 2-phase development methodology for distributed event processing applications
  - **Semantic Models** and **vocabulary** for run-time independent description of event producers, processing agents and consumers
  - **System:** to define and execute distributed event processing pipelines
  - **Open source software artifact (StreamPipes)** as tool support



Intelligent IoT Analytics

## Ongoing research directions

- **Semi-automatic pipeline generation (based on model repository & runtime data)**
- **Dynamic Edge Processing (automatically distribute nodes on edge units)**

## Industrial-grade open source product

- **Extend pipeline element repository with reusable analytics operators**
- **Pipeline monitoring, logging, app system on top of pipelines**

# Bibliography

- Dominik Riemer, Nenad Stojanovic, Ljiljana Stojanovic.  
A Methodology for Designing Events and Patterns in Fast Data Processing. In: *Proceedings of the 25th International Conference on Advanced Information Systems Engineering (CAiSE)*. Valencia, Spain, 2013.
- Dominik Riemer, Ljiljana Stojanovic, Nenad Stojanovic.  
SEPP: Semantics-Based Management of Fast Data Streams. In: *Proceedings of the 7th IEEE International Conference On Service-Oriented Computing and Applications (SOCA)*. Matsue, Japan, 2014.
- Dominik Riemer, Florian Kaulfersch, Robin Hutmacher, Ljiljana Stojanovic.  
StreamPipes: Solving the DEBS Grand Challenge with Semantic Stream Processing Pipelines. In: *Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems (DEBS)*. Oslo, Norway, 2015.
- Philipp Zehnder, Dominik Riemer:  
Modeling self-service machine learning agents for distributed stream processing. In: *Proceedings of the 2017 IEEE Conference on Big Data (IEEE BigData)*. Boston, USA, 2017.

Thank you!

## Questions?

[riemer@fzi.de](mailto:riemer@fzi.de)  
[rudi.studer@kit.edu](mailto:rudi.studer@kit.edu)



[streampipes.org](https://streampipes.org)



[docs.streampipes.org](https://docs.streampipes.org)



[github.com/streampipes/streampipes-ce](https://github.com/streampipes/streampipes-ce)