Semantic Grounding for Stream Reasoning through Ontology-Based Introspection

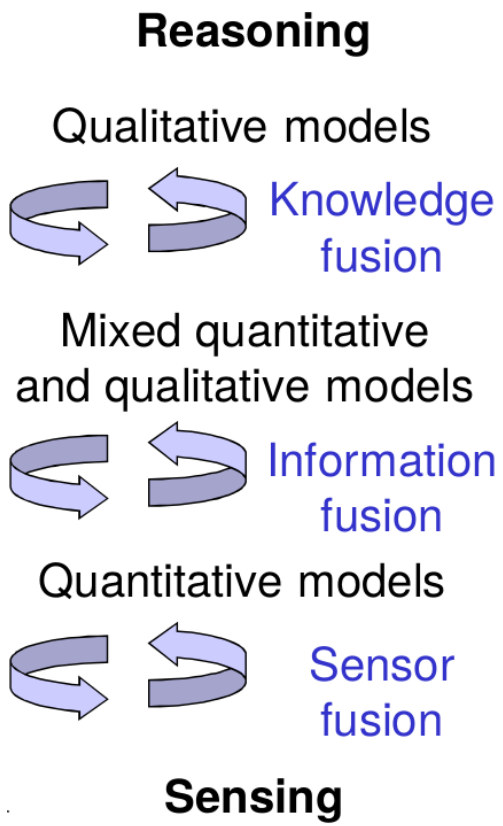
Daniel de Leng

Artificial Intelligence and Integrated Computer Systems
Department of Computer and Information Science
Linköping University, Sweden

Introduction

My research topic: "Grounded spatio-temporal **stream reasoning** with on-demand knowledge acquisition through collaboration and the Semantic Web"

- Background in symbolic artificial intelligence
- Interested in combining subsymbolic with symbolic AI: **sense-reasoning gap**



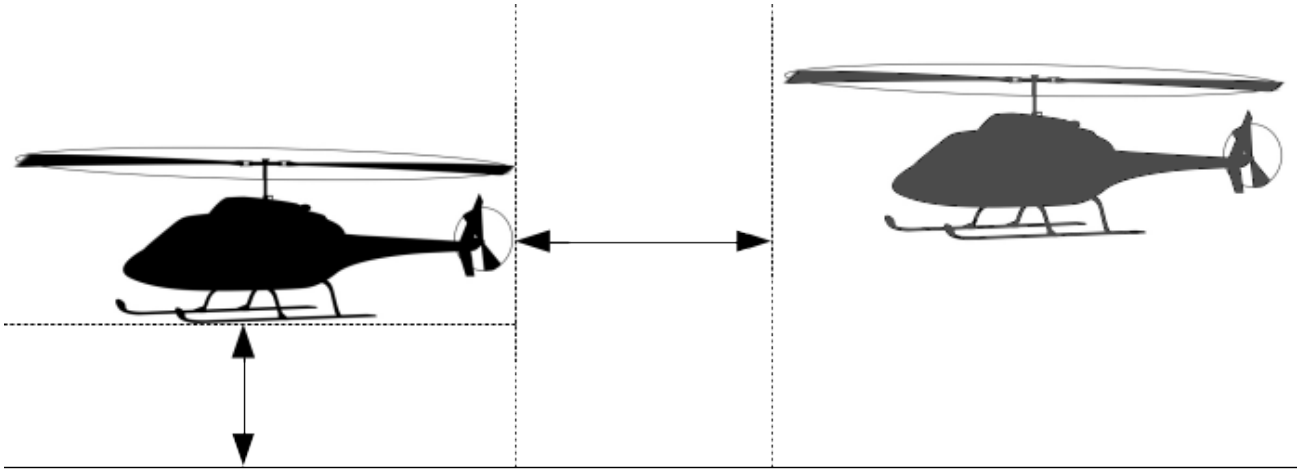
Introduction

1. Introduction
   - Grounding Stream Reasoning
   - Ontology-Based Introspection
   - System Overview
2. Stream Processing with DyKnow
3. Ontology-Based Introspection
   - Ontology for Configuration Modeling
   - Maintaining Correspondence
   - Semantic Matching
4. Conclusions

Grounding Stream Reasoning

A temporal logic formula consists of a number of symbols, including **features**, sorts and objects:

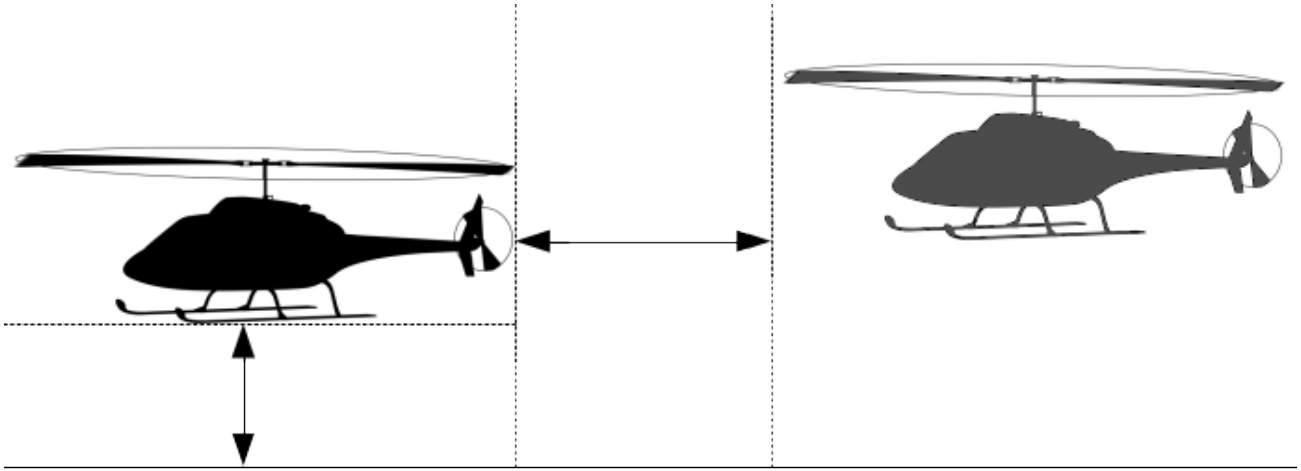
$$\forall u \in \text{UAV} : u \neq \text{uav1} \rightarrow$$
$$\Box(\textbf{Altitude}[\text{uav1}] > 20 \ \vee \ \textbf{XYDist}[u, \text{uav1}] > 10)$$



Grounding Stream Reasoning

A temporal logic formula consists of a number of symbols, including features, **sorts** and objects:

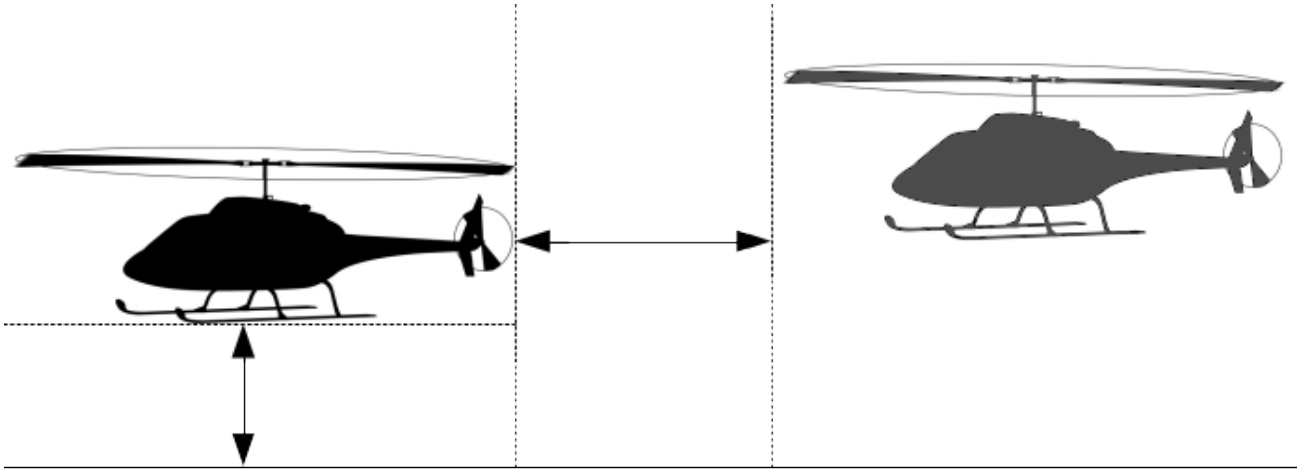
$$\forall u \in \textbf{UAV} : u \neq \text{uav1} \rightarrow$$
$$\Box(\text{Altitude}[\text{uav1}] > 20 \ \vee \ \text{XYDist}[u, \text{uav1}] > 10)$$



Grounding Stream Reasoning

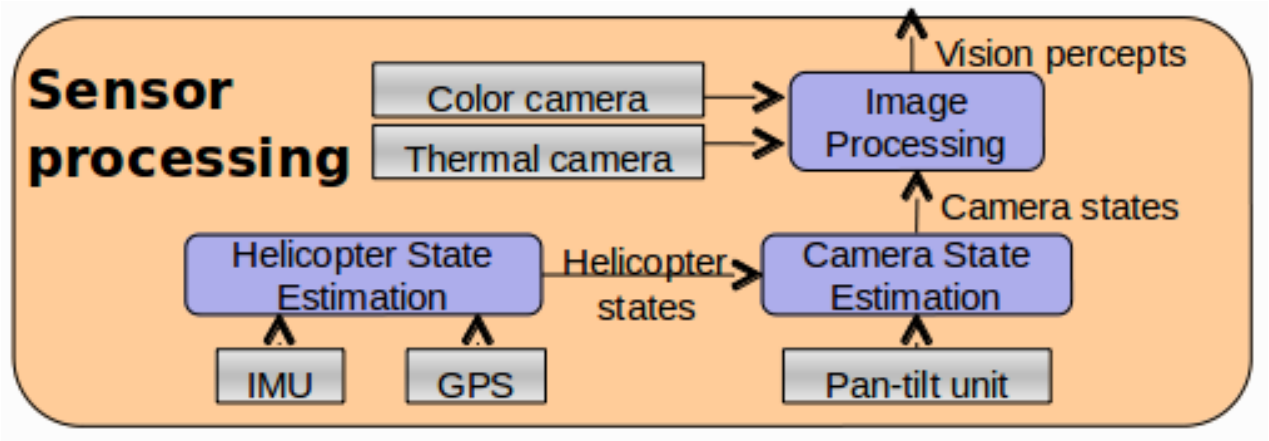
A temporal logic formula consists of a number of symbols, including features, sorts and **objects**:

$$\forall u \in \text{UAV} : u \neq \textbf{uav1} \rightarrow$$
$$\Box(\text{Altitude}[\textbf{uav1}] > 20 \ \vee \ \text{XYDist}[u, \textbf{uav1}] > 10)$$



Grounding Stream Reasoning

Given a functional system, such as a robot, producing streams the integration problem for logic-based stream reasoning is to **connect symbols in formulas to streams in the functional system so that the symbols get their intended meaning**.

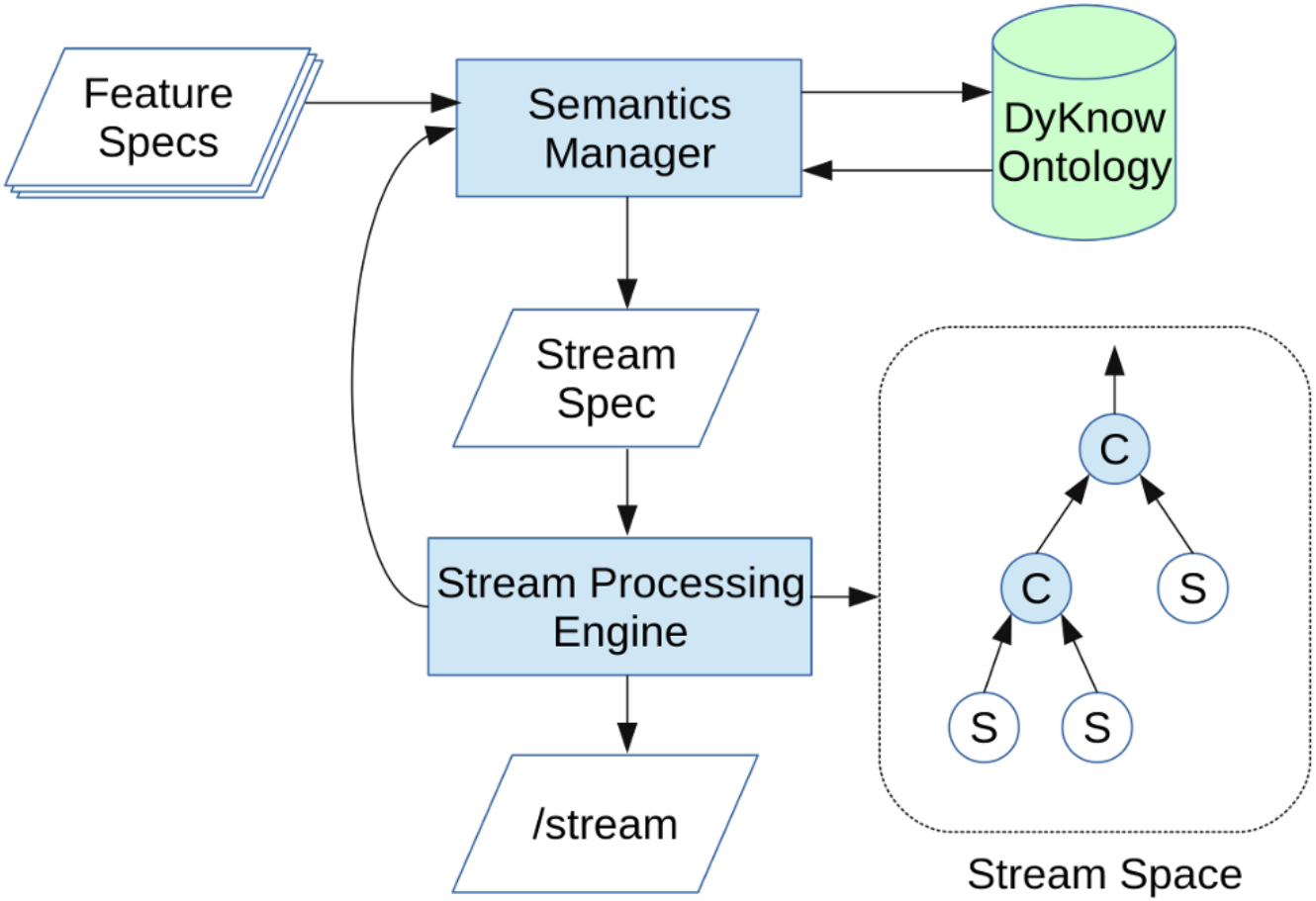


Ontology-Based Introspection

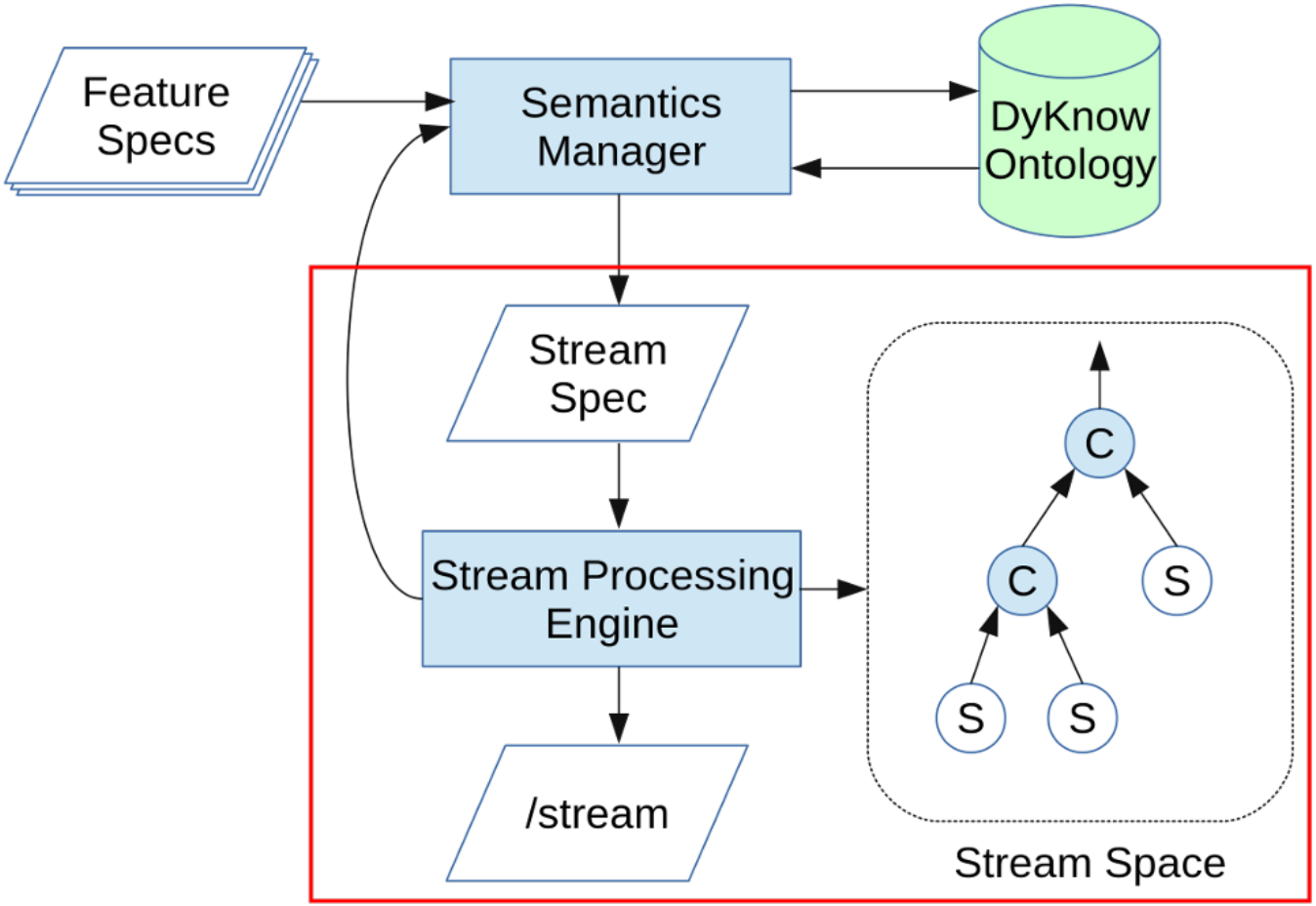
Our approach applies **ontology-based introspection** to model and query streaming configurations. It makes use of

- an **ontology** consisting of concepts and relations between concepts, utilising the Web Ontology Language (OWL),
- **semantic annotations** for stream transformations describing the features they handle, and
- a **feature specification**, such as a feature symbol in a formula.

System Overview



Stream Processing with DyKnow



Stream Processing with DyKnow

Stream processing components:

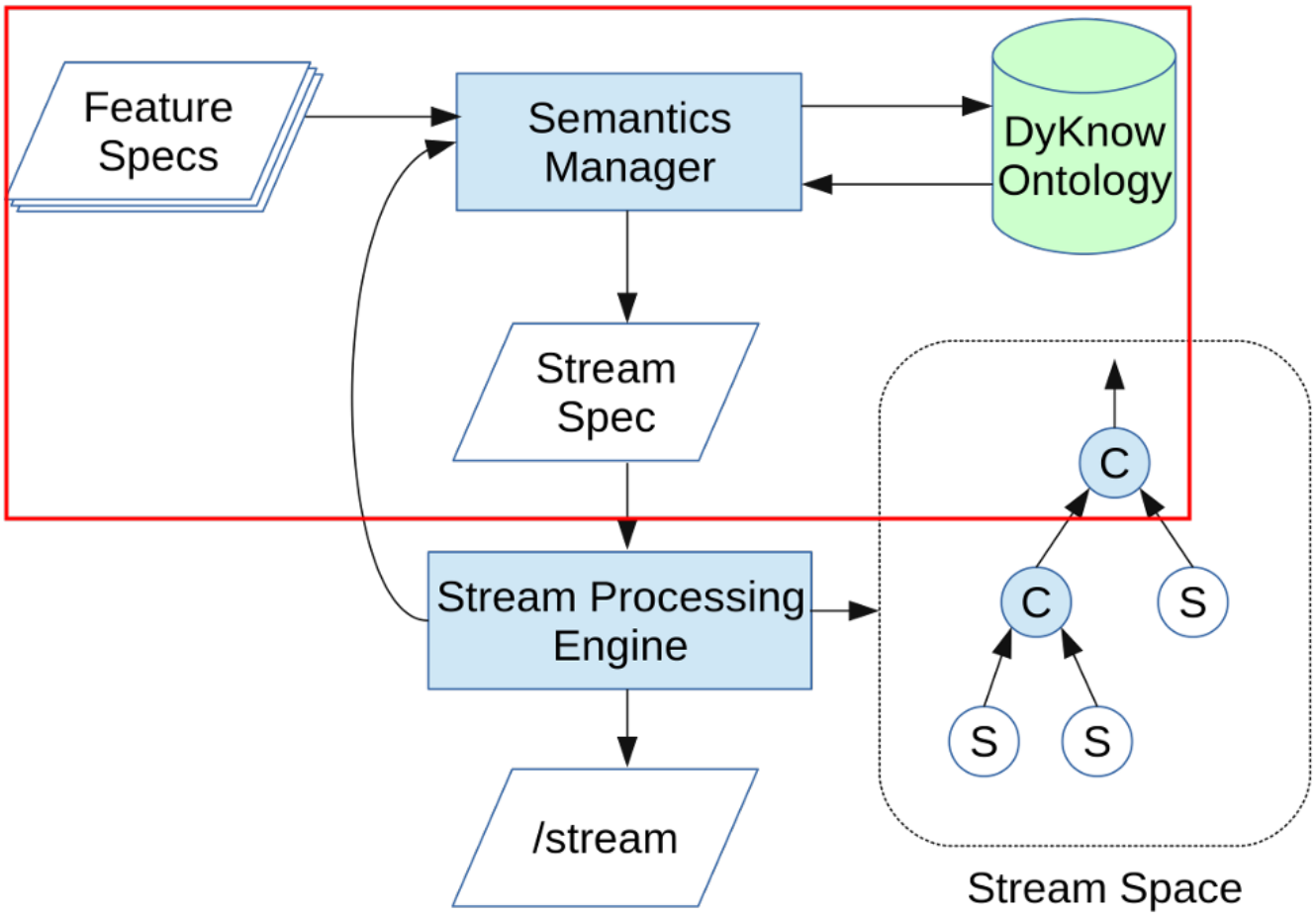
- **Streams** are sequences of time-stamped values.
- **Transformations** are stream generating functions that take streams as input.
- **Computation Units** are transformation instances.

Stream Processing with DyKnow

Configuration specifications can describe how to combine transformations to generate some stream. For example:

```
<spec:specification>
    <spec:insertions>
        <spec:cu name="result" type="project2Dto3D">
            <spec:cu type="fusionRGBIR">
                <spec:cu type="rgbCam" />
                <spec:cu type="irCam" />
            </spec:cu>
            <spec:cu type="GPSto3D">
                <spec:cu type="gps" />
            </spec:cu>
        </spec:cu>
    </spec:insertions>
    <spec:removals>
        <!-- Removals based on names of transformations and CUs -->
    </spec:removals>
</spec:specification>
```

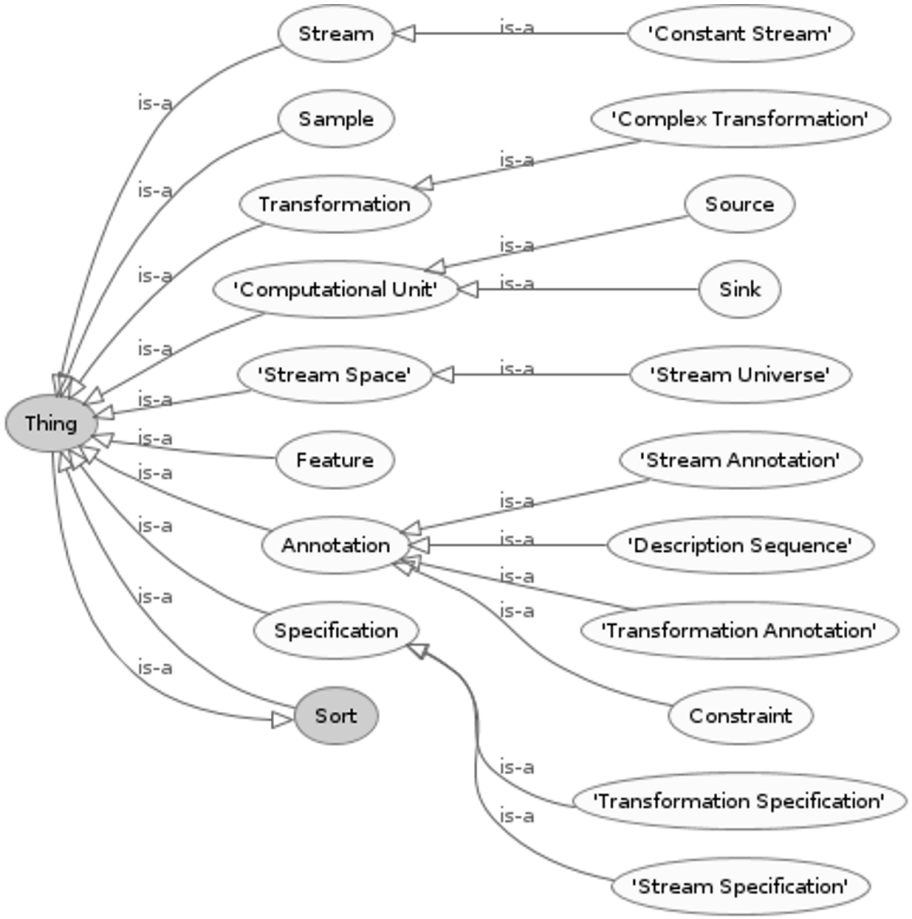
Ontology-Based Introspection



Ontology for Configuration Modeling

Motivation:

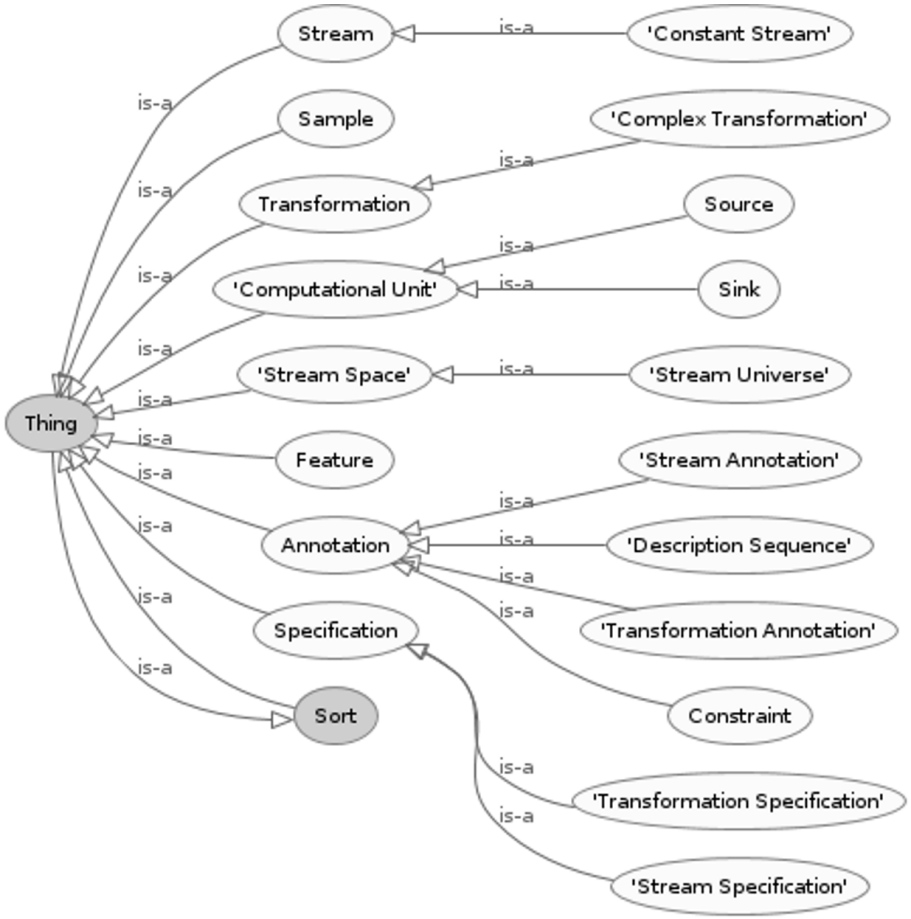
- An ontology gives a **common vocabulary** with which configurations can be described and shared.
- Allows for **semantically annotating** transformations through properties.
- Closely related to e.g. the OWL-S service ontology.



Ontology for Configuration Modeling

Example computation unit:

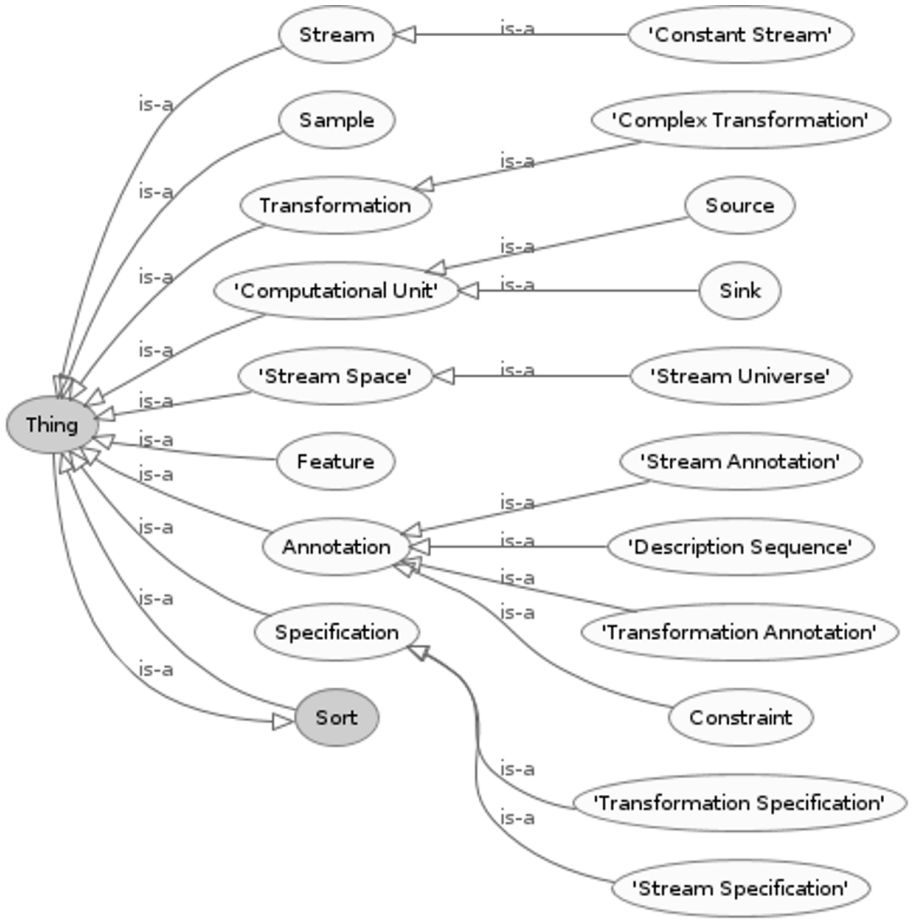
```
1 :dyknow_cu1 a :ComputationalUnit ;
2     :hasInput :dyknow_stream1 ;
3     :hasInput :dyknow_stream2 ;
4     :hasOutput :dyknow_stream3 ;
5     :instantiationOf :fusionRGBIR .
```



Ontology for Configuration Modeling

Example transformation with semantic annotation:

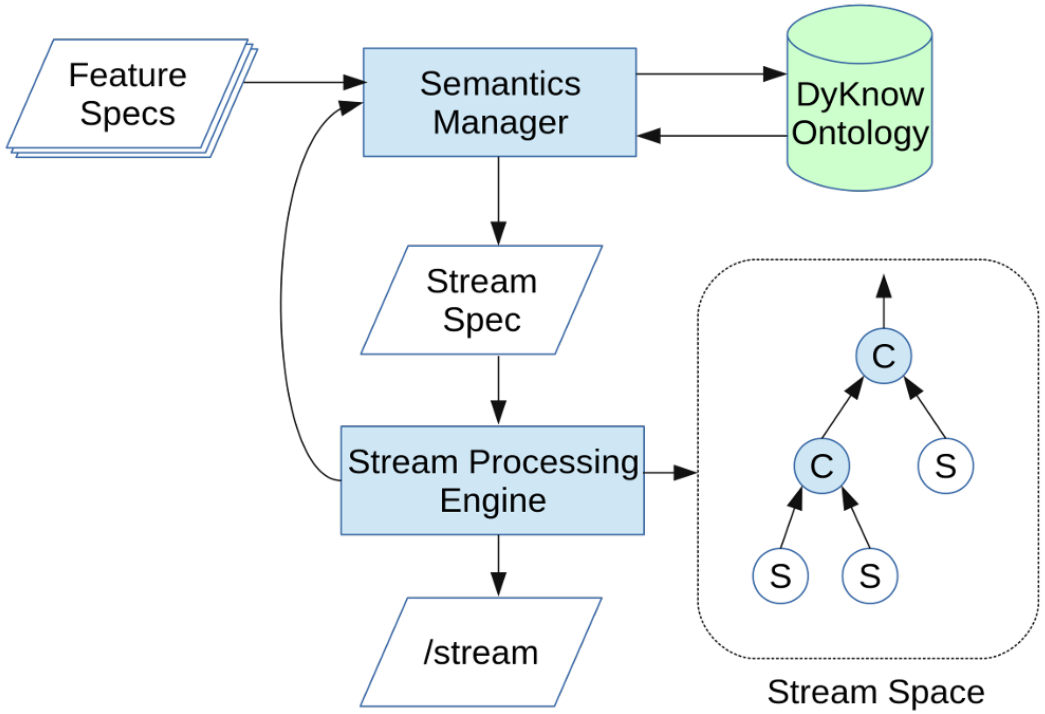
```
1  :fusionRGBIR a :Transformation ;
2      :hasAnnotation [
3          :hasOutputAnnotation [
4              :describesFeature :ImagePosition ;
5              :describesSort :Human
6          ] ;
7          :hasInputAnnotation [
8              :describesFeature :RawRGBCam ;
9              :describesSort :self ;
10             :nextSegment [
11                 :hasInputAnnotation [
12                     :describesFeature :RawIRCam ;
13                     :describesSort :self
14                 ]
15             ]
16         ]
17     ] ;
18     :hasName "fusionRGBIR"^^string .
```



Maintaining Correspondence

The configuration model needs to be **continuously updated** to reflect the actual configuration:

- Stream Processing Engine receives configuration specifications and executes them. Changes are reported over a **status stream**.
- The Semantics Manager listens to the status stream and updates the configuration model.



Semantic Matching

Semantic Matching is the task of providing stream specifications given a desired feature specification.

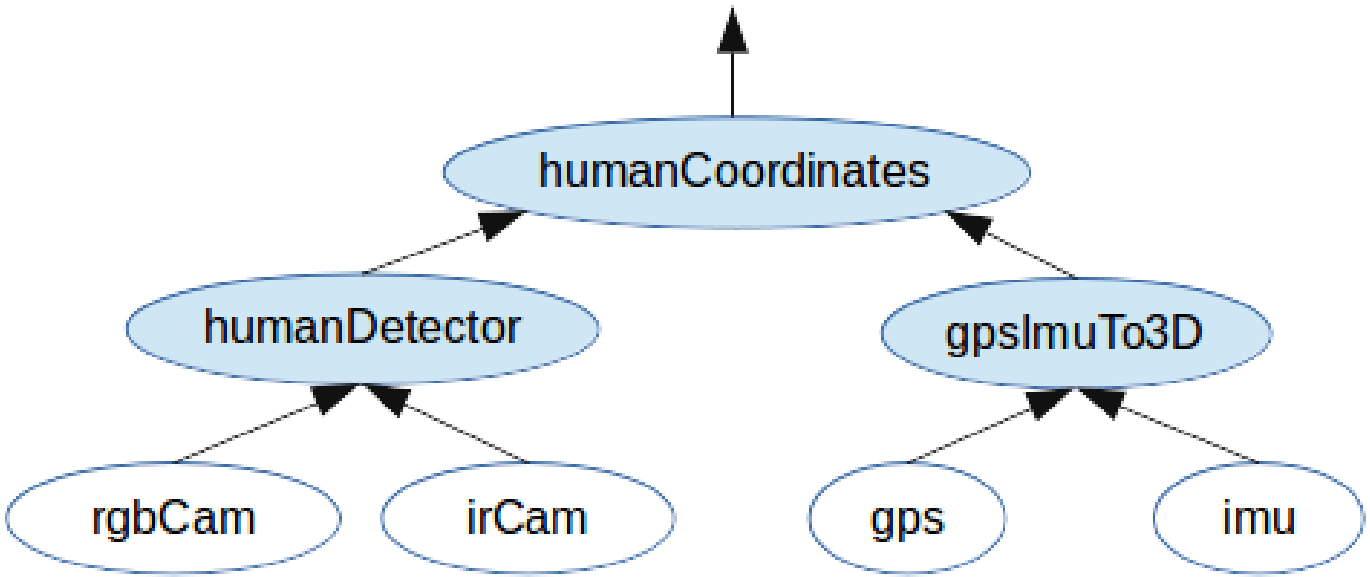
1. Given a desired feature, find all transformations producing that feature.
2. Recursively make those transformations' input features the new desired features.
3. A valid transformation tree has transformations with no inputs as leafs.
4. If multiple transformation trees exist, combine or choose one.

Semantic Matching

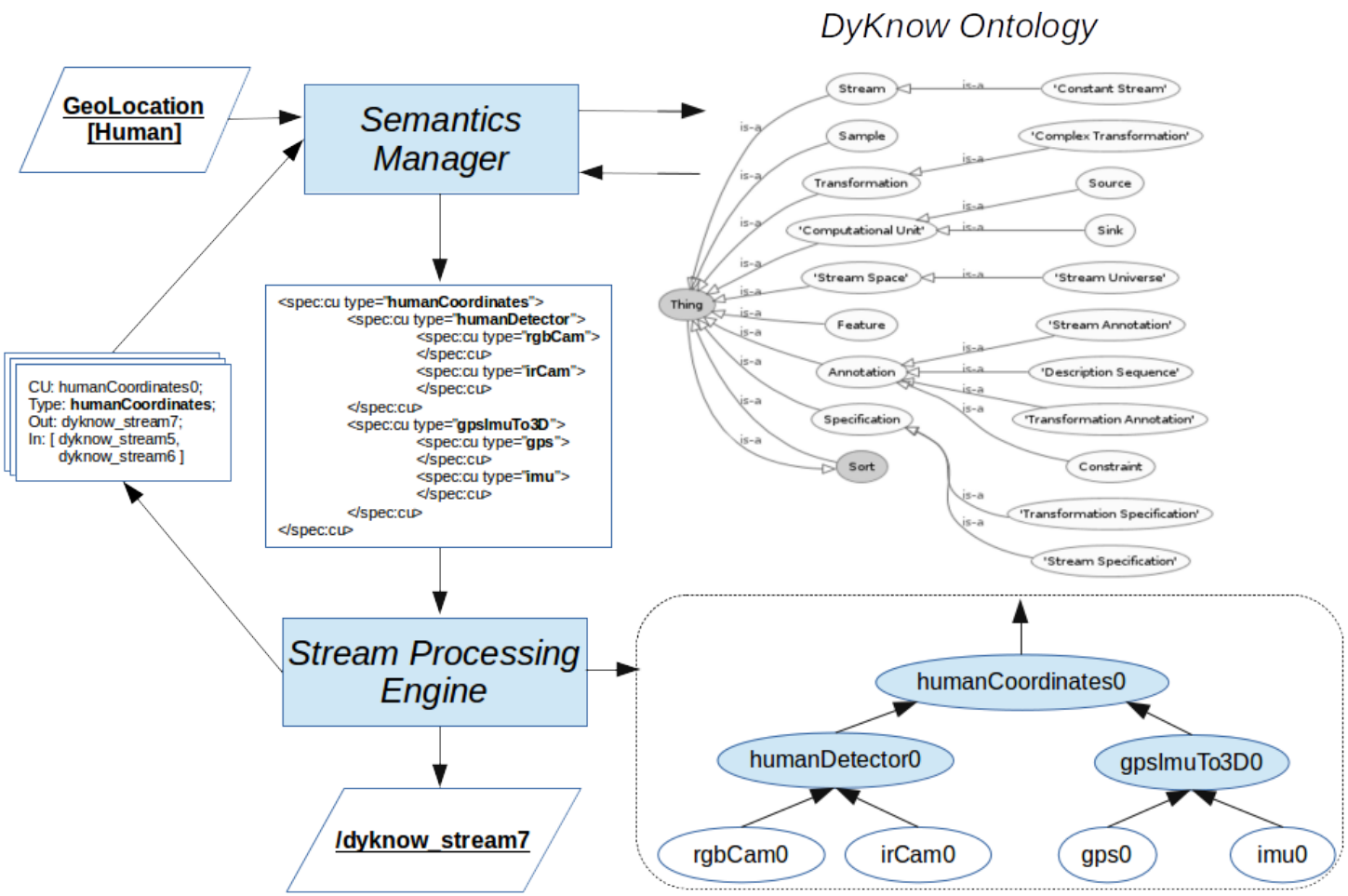
Consider the following example transformations:

- `gps` : $\emptyset \Rightarrow$ GPS[self]
- `imu` : $\emptyset \Rightarrow$ IMU[self]
- `rgbCam` : $\emptyset \Rightarrow$ RGB[self]
- `irCam` : $\emptyset \Rightarrow$ IR[self]
- `attitude` : IMU[Thing] $\Rightarrow$ Attitude[Thing]
- `GPSto3D` : GPS[Thing] $\Rightarrow$ GeoLocation[Thing]
- `humanDetector` : RGB[RMAX], IR[RMAX] $\Rightarrow$ PixelLocation[Human]
- `humanCoordinates` : PixelLocation[Human], GeoLocation[RMAX], Attitude[RMAX] $\Rightarrow$ GeoLocation[Human]

Semantic Matching



Semantic Matching



Conclusions

- Ontologies can be used to model stream processing.
- Semantic annotations provide high-level descriptions of transformations.
- Semantic matching makes use of **ontology-based introspection** to generate stream specifications automatically.
- Interesting opportunities for extension, such as semantic subscriptions.