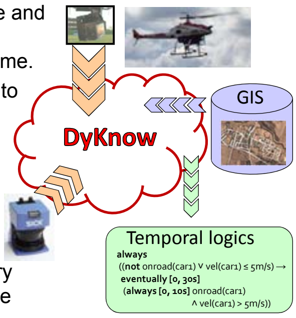
Integrating Stream Reasoning in Robotic Systems using Semantic Technologies

Fredrik Heintz and Zlatan Dragisic
Knowledge Processing Laboratory/UASTech Laboratory
Artificial Intelligence and Integrated Computer Systems Division
Department of Computer and Information Science
Linköping University, Sweden

Stream Reasoning

- Autonomous systems produce and process sequences of values incrementally created at run-time.
- These sequences are natural to model as *streams*.
- *Stream reasoning* is incremental reasoning over streams.
- Stream reasoning captures the continuous reasoning with minimal latency necessary to react to rapid changes in the environment.



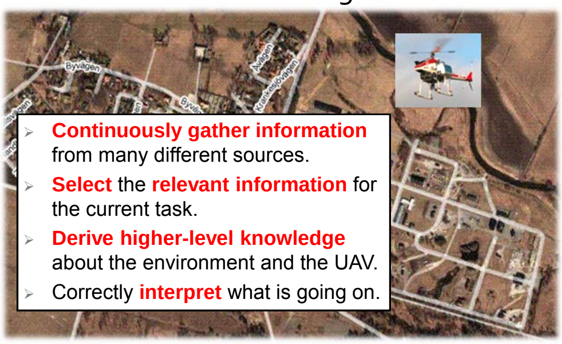
UASTech UAV Platforms

Micro Aerial Vehicles
weight < 500 g,
diameter < 50 cm

Yamaha RMAX
weight 95 kg,
length 3.6 m

LinkQuad weight ~1 kg, diameter ~70cm

A Traffic Monitoring Scenario

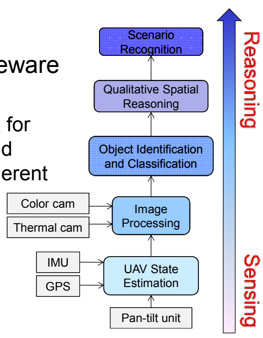


- **Continuously gather information** from many different sources.
- **Select** the **relevant information** for the current task.
- **Derive higher-level knowledge** about the environment and the UAV.
- Correctly **interpret** what is going on.

DyKnow

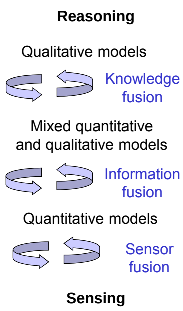
DyKnow is a stream-based knowledge processing middleware framework that provides

- a formal conceptual framework for integrating different sensing and reasoning approaches in a coherent processing framework,
- stream reasoning functionalities, and
- a distributed implementation infrastructure.

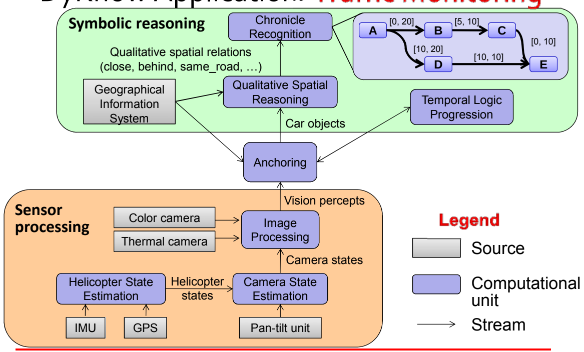


Requirements

- Integrating information from distributed sources.
- Processing at many different levels of abstraction.
- Quantitative and qualitative processing.
- Bottom-up data processing and top-down model-based processing.
- Managing uncertainty on different levels of abstraction.
- Flexible configuration and reconfiguration.
- Declarative specification of the information being generated and the available information processing functionalities.



DyKnow Application: Traffic Monitoring



DyKnow: Fluent Streams

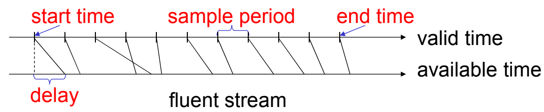
DyKnow models the world in terms of *objects* and *features*. A feature has a value at every time-point.

A *sample* is a stream element <t_a, t_v, *v*> where t_a is the *available time*, t_v is the *valid time*, and *v* is the *value* of the sample. A sample can be used to represent the value of a feature at a particular time-point (the valid time).

A *fluent stream* is a stream where each stream element is a sample. A fluent stream can be used to represent an approximation of the value of a feature over time.

DyKnow: Policy

from 0 **to** 120s **sample every** 100ms **max delay** 200ms

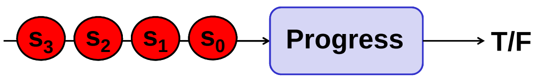



```
FLUENT_STREAM_POLICY := FLUENT_STREAM_CONSTRAINT*
FLUENT_STREAM_CONSTRAINT :=
          VALUE_APPROXIMATION_CONSTRAINT
        | CHANGE_CONSTRAINT
        | DELAY_CONSTRAINT
        | DURATION_CONSTRAINT
        | ORDER_CONSTRAINT
```

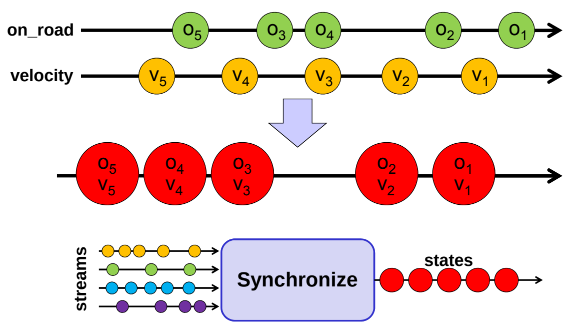

Stream Reasoning in Metric Temporal Logic

always
((**not** on_road(car1) ∨ velocity(car1) ≤ 5m/s) →
eventually [0, 30s]
(**always [0, 10s]** on_road(car1) ∧ velocity(car1) > 5m/s))

The semantics of these formulas is defined over infinite state sequences. Progression is one technique to check whether the current prefix is sufficient to determine the truth value of a formula.



State Streams



Execution Monitoring (ICAPS'08, JAAMAS'09)

If things can go wrong they probably will!

This implies the need for continual monitoring of an autonomous system and its environment in a principled, contextual, task specific manner which can be specified by the system itself.

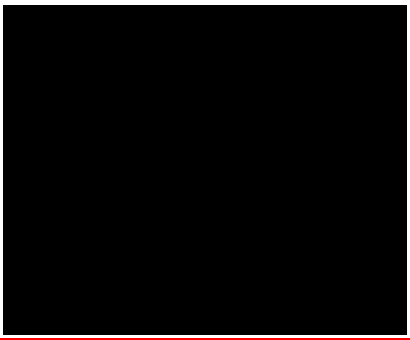
always (eventually [0, t] (always [0,t'] speed(uav) < T))

It should always be the case that within t time units from now, an interval of length t' should start where the UAV's speed stays below threshold T.

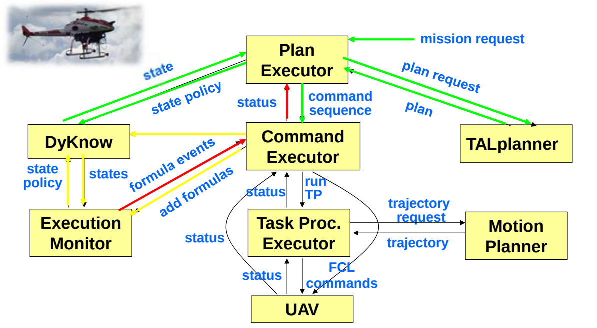
EXEC until [0, 5000] (¬EXEC ∧ altitude(uav) > 7)

The command should take less than 5 seconds to execute and when the execution is finished the altitude of the UAV should be above 7 meters.

Fail to Attach



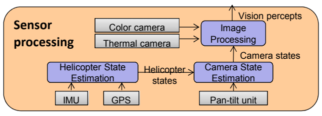
Stream Reasoning for Planning (ICAPS'08, JAAMAS'09)



Integrating Stream Reasoning

- A temporal logical formula contains a number of symbols representing variables whose values over time must be collected and synchronized in order to determine the truth value of the formula.
- 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*.

$forall\ x\ in\ UAV\ Behind[x, uav1]\ and\ Altitude[uav1]\ >\ 10$



Integration using Semantic Technologies

- The general idea is
  - create an ontology for the domain, such as traffic monitoring,
  - annotate streams in the functional system with ontological concepts based on their content,
  - write formulas using the ontological concepts, and
  - given a formula, find the relevant streams by matching the ontological concepts used in the formula to the ontological concepts associated with the streams.

Integration using Semantic Technologies

- Example:

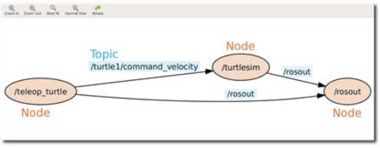
  $forall\ x\ in\ UAV\ Behind[x, uav1]\ and\ Altitude[uav1]\ >\ 10$
  - Ontology
    - Classes: UAV isa MobileObject
    - Objects: UAV = {uav1, uav2, uav3}
    - Unary features: Altitude[UAV]
    - Binary features: Behind[MobileObject, MobileObject]
  - Streams
    - helistate containing HeliState samples with fields id and alt
    - behind containing BinaryRelation samples with fields arg1, arg2 and value

Integration using Semantic Technologies

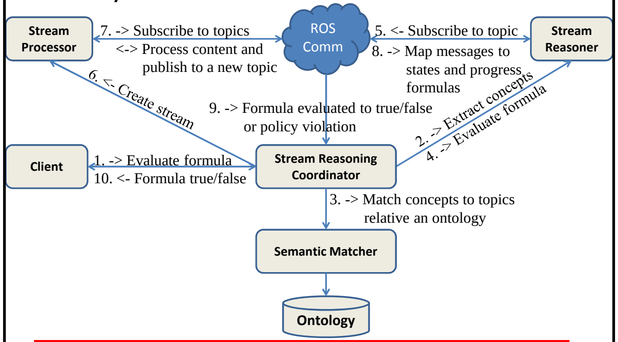
- We need
  - An ontology
  - A method for annotating streams with ontological concepts
  - A method for finding all streams which contain relevant information given an ontological concept
- We use
  - OWL for representing the ontology
  - An XML-based stream annotation language
  - A matching method using the ontology and the annotations for finding all streams for a formula

The Robot Operating System (ROS)

- A framework for robot software development providing operating system-like functionality
- Main concepts
  - nodes
  - Messages
  - topics
  - services



DyKnow Architecture Overview



Relations ontology

Relation examples:

$Altitude \subseteq UnaryRelation \cap \forall arg1.UAV$

$Behind \subseteq BinaryRelation \cap \forall arg1.Object \cap \forall arg2.Object$

Object types ontology

Semantic Annotation of Streams

- To annotate streams we have designed a language called Semantic Specification Language for Topics SSL_T.
- Three categories of streams:
  - Streams containing sorts
  - Streams containing features
  - Streams containing objects
- Topic specifications in SSL_T are represented using XML to support semantic matching.

Semantic Annotation of Streams

- Example: A stream containing the features Altitude and Speed for the sort UAV
- Stream (ROS topic):
  Topic name: `topic1`
  Message type: `UAVMsg`
  Fields: `int id; float alt; float spd`
- SSL_T specification:
  topic `topic1:UAVMsg` **contains** `Altitude(uav1)=msg.alt`
- XML representation:

```
<topic msgtype="UAVMsg" name="topic1">
 <feature name="Altitude" value="msg.alt">
  <object name="uav1"/>
 </feature>
</topic>
```

Semantic Annotation of Streams

- Example: A stream containing the features Behind(x, uav2) for all UAVs x
- Stream (ROS topic):

```
Topic name: topic6
Message type: BinaryRelation
Fields: int id1; int id2; bool value
```

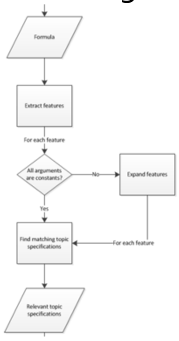
- SSL_T specification:

```
topic topic6:BinaryRelation
contains Behind(UAV,uav2)=msg.value for every UAV=msg.id1
```

- XML representation:

```
<topic msgtype="BinaryRelation" name="topic6">
 <feature name="Behind" value="msg.value">
  <sort name="UAV" value="msg.id1" all_objects="true" />
  <object name="uav2" />
 </feature>
</topic>
```


Semantic Matching – Overview



Semantic Matching – Extracting Features

$forall\ x\ in\ UAV\ Behind[x, uav1]\ and\ Altitude[uav1] > 10$

- *Behind[UAV, uav1]*
- *Altitude[uav1]*

Expanded features for feature *Behind[UAV, uav1]:*

- *Behind[uav1, uav1]*
- *Behind[uav2, uav1]*
- *Behind[uav3, uav1]*

Semantic Matching – Example (1)

```
<topic msgtype="UAVMsg" name="topic1">
 <feature name="Altitude"
   value="msg.alt">
  <object name="uav1"/>
 </feature>
</topic>

<topic msgtype="UAVMsg" name="topic2">
 <feature name="Speed" value="msg.spd">
  <object name="uav1" value="msg.id"/>
 </feature>
</topic>

<topic msgtype="UAVMsg" name="topic3">
 <feature name="Altitude"
   value="msg.alt">
  <sort name="UAV" value="msg.id"
   all_objects="false" />
 </feature>
</topic>
```

```
<topic msgtype="BinaryRel" name="topic4">
 <feature name="Behind" value="msg.value">
  <object name="uav1" value="msg.id1" />
  <object name="uav3" value="msg.id2" />
 </feature>
</topic>

<topic msgtype="BinaryRel" name="topic5">
 <feature name="Behind" value="msg.value">
  <object name="uav2" value="msg.id1" />
  <object name="uav1" value="msg.id2" />
 </feature>
</topic>

<topic msgtype="BinaryRel" name="topic6">
 <feature name="Behind" value="msg.value">
  <sort name="UAV" value="msg.id1"
   all_objects="true" />
  <object name="uav2" value="msg.id2" />
 </feature>
</topic>
```

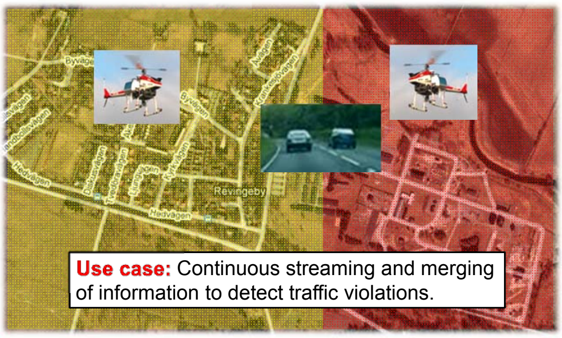

Semantic Matching – Example (2)

```
<topic msgtype="BinaryRel" name="topic4">
 <feature name="Behind" value="msg.value">
  <object name="uav1" value="msg.id1" />
  <object name="uav3" value="msg.id2" />
 </feature>
</topic>

<topic msgtype="BinaryRel" name="topic5">
 <feature name="Behind" value="msg.value">
  <object name="uav2" value="msg.id1" />
  <object name="uav1" value="msg.id2" />
 </feature>
</topic>

<topic msgtype="BinaryRel" name="topic6">
 <feature name="Behind" value="msg.value">
  <sort name="UAV" value="msg.id1" all_objects="true" />
  <object name="uav2" value="msg.id2" />
 </feature>
</topic>
```

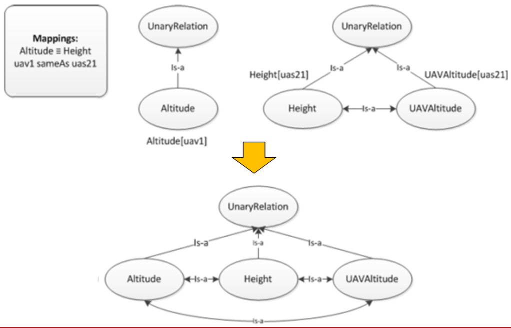

Semantic Matching – Example (3)

```
<topic msgtype="BinaryRel" name="topic4">
 <feature name="Behind" value="msg.value">
  <object name="uav2" value="msg.id1" />
  <object name="uav1" value="msg.id2" />
 </feature>
</topic>

<topic msgtype="BinaryRel" name="topic6">
 <feature name="Behind" value="msg.value">
  <sort name="UAV" value="msg.id1" all_objects="true" />
  <object name="uav2" value="msg.id2" />
 </feature>
</topic>
```

Multi UAV Traffic Monitoring



Semantic Matching with Multiple Robots



Performance evaluation

Evaluating different aspects of the semantic matching process:

1. Size of ontology – number of concepts as well as number of relevant and irrelevant individuals in the ontology
2. Number of topic specifications – number of relevant and irrelevant topic specifications
3. Size of formulas – number of features in a formula
4. Type of features in formulas – quantified and non-quantified arguments

Performance evaluation (1)

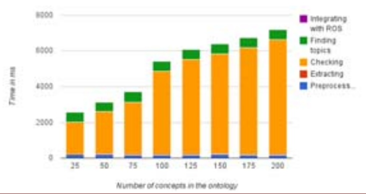
The execution time is divided into five phases:

1. Preprocessing – includes loading of ontologies and topic specifications into memory
2. Extracting features – includes feature extraction from a logical formula
3. Checking features – includes the process of checking extracted features against an ontology
4. Matching topic specifications – process of querying for relevant topic specifications
5. ROS integration – transforming classes

Performance evaluation (2)

Test case 1.1 – the number of concepts in an ontology is varied (0 to 200) while the number of individuals and topic specifications is kept constant (200 and 20 respectively)

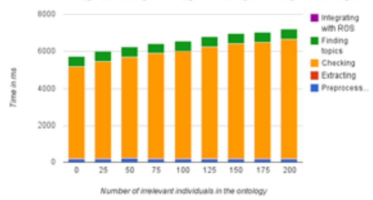
$$X[b1] \; and \; X[b2] \; and \; X[b3] \; and \; Y[b1,b1] \; and \; Y[b1,b2] \; and \; Y[b1,b3] \text{ and } Z[b1,b2,b1] \; and \; Z[b1,b2,b2] \; and \; Z[b1,b2,b3]$$



Performance evaluation (3)

Test case 1.2 – the number of irrelevant individuals is varied (0 to 200) while the number of concepts and topic specifications is kept constant (200 and 20 respectively)

$$X[b1] \; and \; X[b2] \; and \; X[b3] \; and \; Y[b1,b1] \; and \; Y[b1,b2] \; and \; Y[b1,b3] \text{ and } Z[b1,b2,b1] \; and \; Z[b1,b2,b2] \; and \; Z[b1,b2,b3]$$

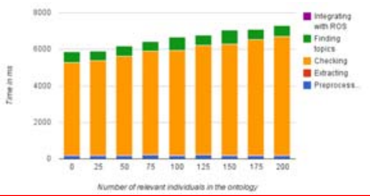


Linköpings universitet

Performance evaluation (4)

Test case 1.3 – the number of relevant individuals is varied (0 to 200) while the number of concepts and topic specifications is kept constant (200 and 20 respectively)

$X[b1]$ and $X[b2]$ and $X[b3]$ and $Y[b1, b1]$ and $Y[b1, b2]$ and $Y[b1, b3]$ and $Z[b1, b2, b1]$ and $Z[b1, b2, b2]$ and $Z[b1, b2, b3]$

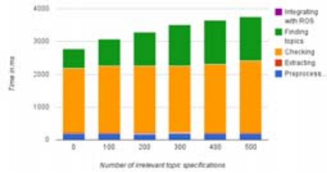


Linköpings universitet

Performance evaluation (5)

Test case 2.1 – the number of irrelevant topic specifications is varied (0 to 500) while the number of concepts, individuals and relevant topic specifications is kept constant (50, 50 and 9 respectively)

$X[b1]$ and $X[b2]$ and $X[b3]$ and $Y[b1, b1]$ and $Y[b1, b2]$ and $Y[b1, b3]$ and $Z[b1, b2, b1]$ and $Z[b1, b2, b2]$ and $Z[b1, b2, b3]$

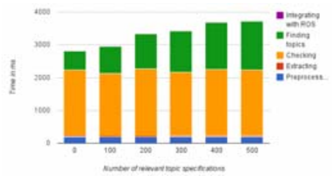


Linköpings universitet

Performance evaluation (6)

Test case 2.2 – the number of relevant topic specifications is varied (100 to 500) while the number of concepts, individuals and irrelevant topic specifications is kept constant (50, 50 and 9 respectively)

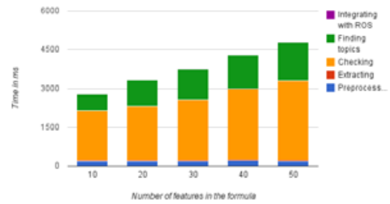
$X[b1]$ and $X[b2]$ and $X[b3]$ and $Y[b1, b1]$ and $Y[b1, b2]$ and $Y[b1, b3]$ and $Z[b1, b2, b1]$ and $Z[b1, b2, b2]$ and $Z[b1, b2, b3]$



Linköpings universitet

Performance evaluation (7)

Test case 3 – the number of features in the formula is varied (10 to 50) while the number of concepts, individuals and topic specifications is kept constant (50, 50 and 50 respectively)

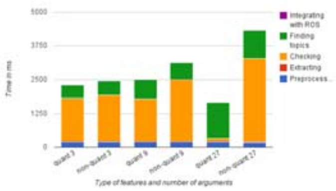


Linköpings universitet

Performance evaluation (8)

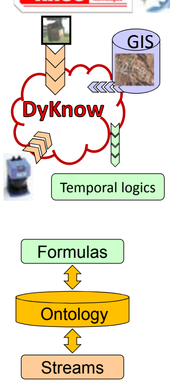
Test case 4 – two versions of the same formula are tested, one with quantifiers and one without. This is done for three formulas with varying number of features (3, 9, 27). There are 27 topic specifications while the ontology contains 50 concepts and 50 individuals.

$$forall\ x\ in\ B\ (Z[x, d1, e1])$$
$$forall\ x\ in\ B, y\ in\ D\ (Z[x, y, e1])$$
$$forall\ x\ in\ B, y\ in\ D, z\ in\ E(Z[x, y, z])$$



Linköpings universitet

Conclusions



- Stream reasoning captures both the incremental nature of sensing and the continuous reasoning with minimal latency necessary for autonomous systems.
- Semantic technologies simplify the integration of stream reasoning in robotic systems by providing a "mediator" in the form of a common ontology a the possibly to find matching concepts.