

Modeling, Representing, and Configuring Restricted Part-Whole Relations

Lothar Hotz

HITeC e.V., University of Hamburg

hotz@informatik.uni-hamburg.de

Overview

- Scene Interpretation as Configuration
- Analysis of Reasoning Tasks
- Experiment
- Summary

Interpretation of Images (and Videos)

- Given: image or video of a technical world
- Compute a description of the things that can be seen in the image or video
- Domains: façade scenes, apron observation, table-laying scenes

Interpretation of Images?

Image →

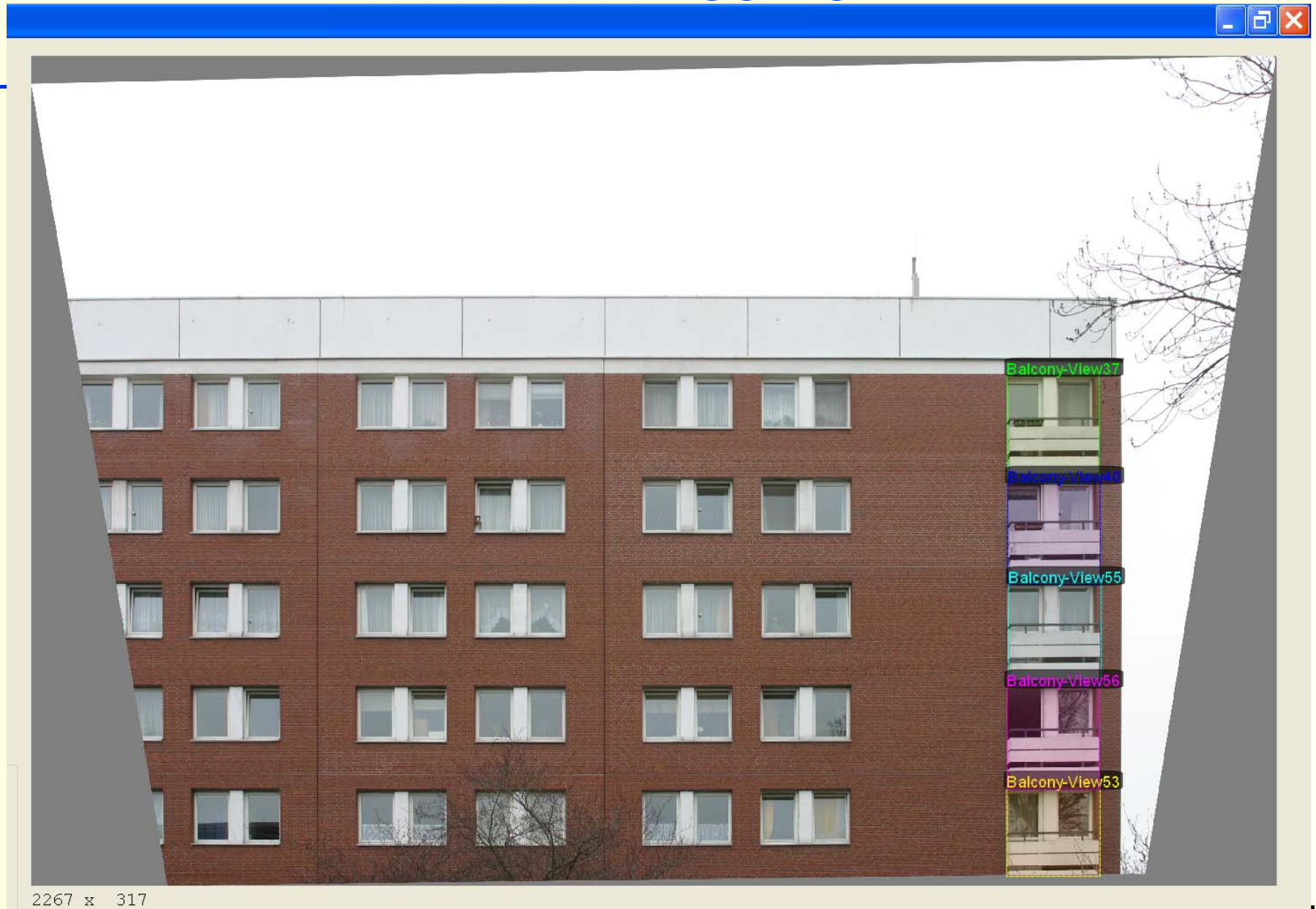
?

↓
Scene description

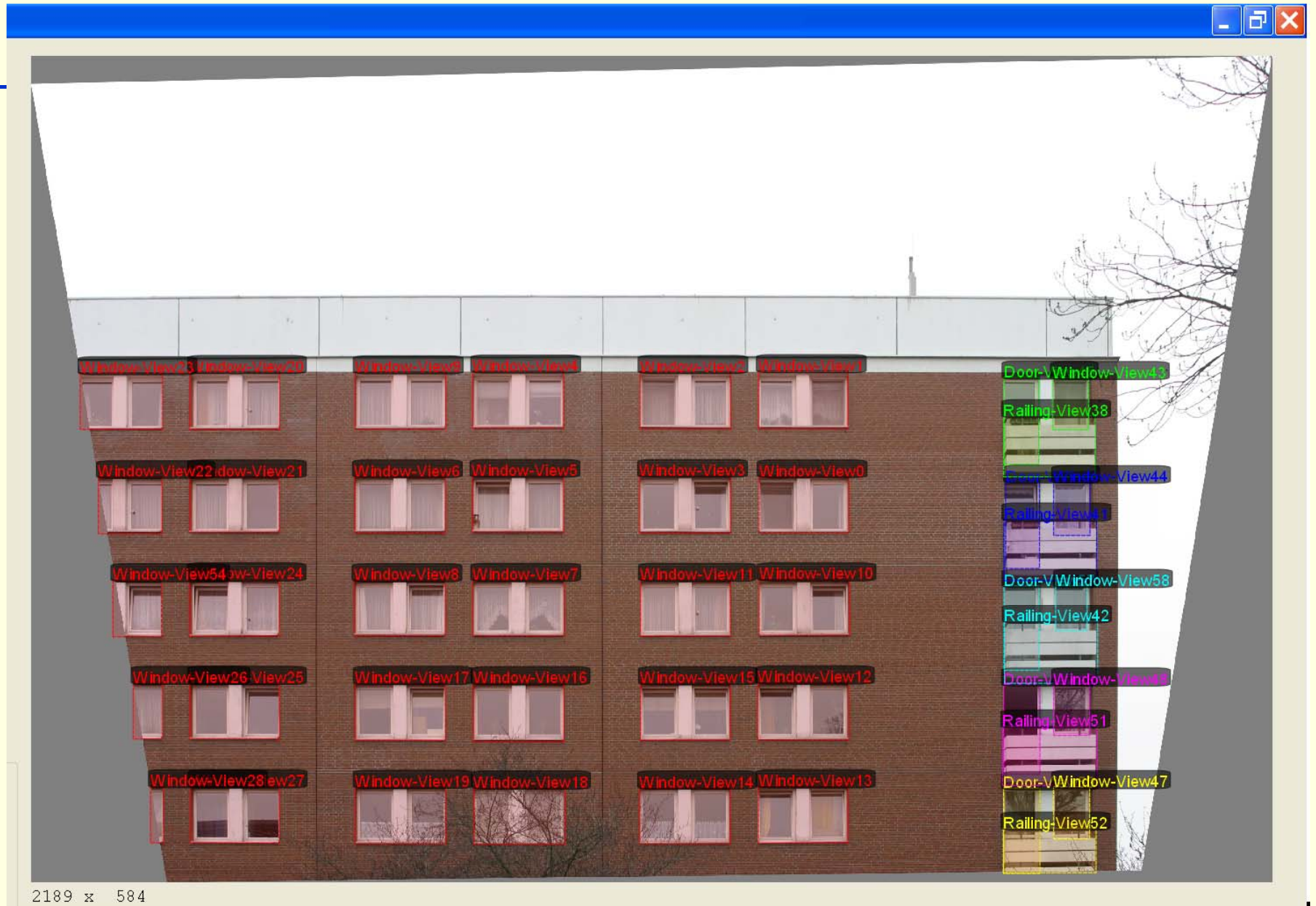
Image



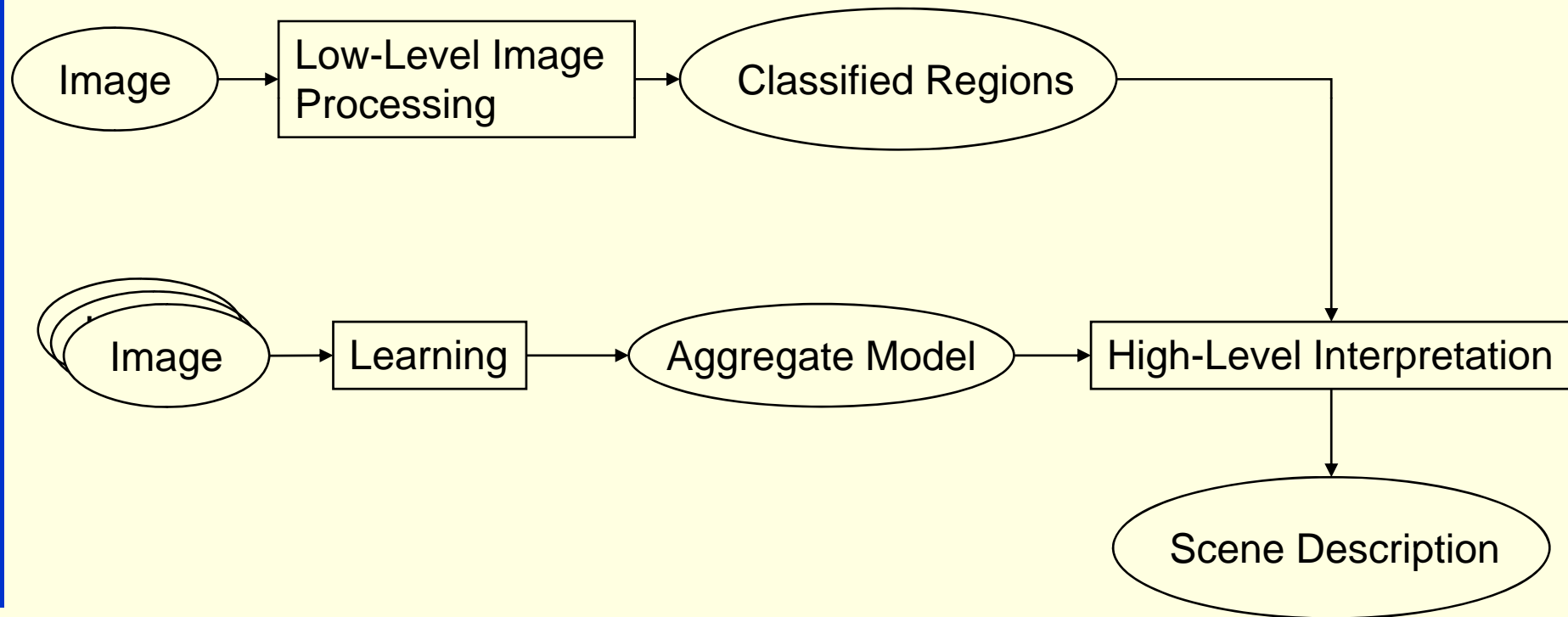
Scene Description with Aggregates and...



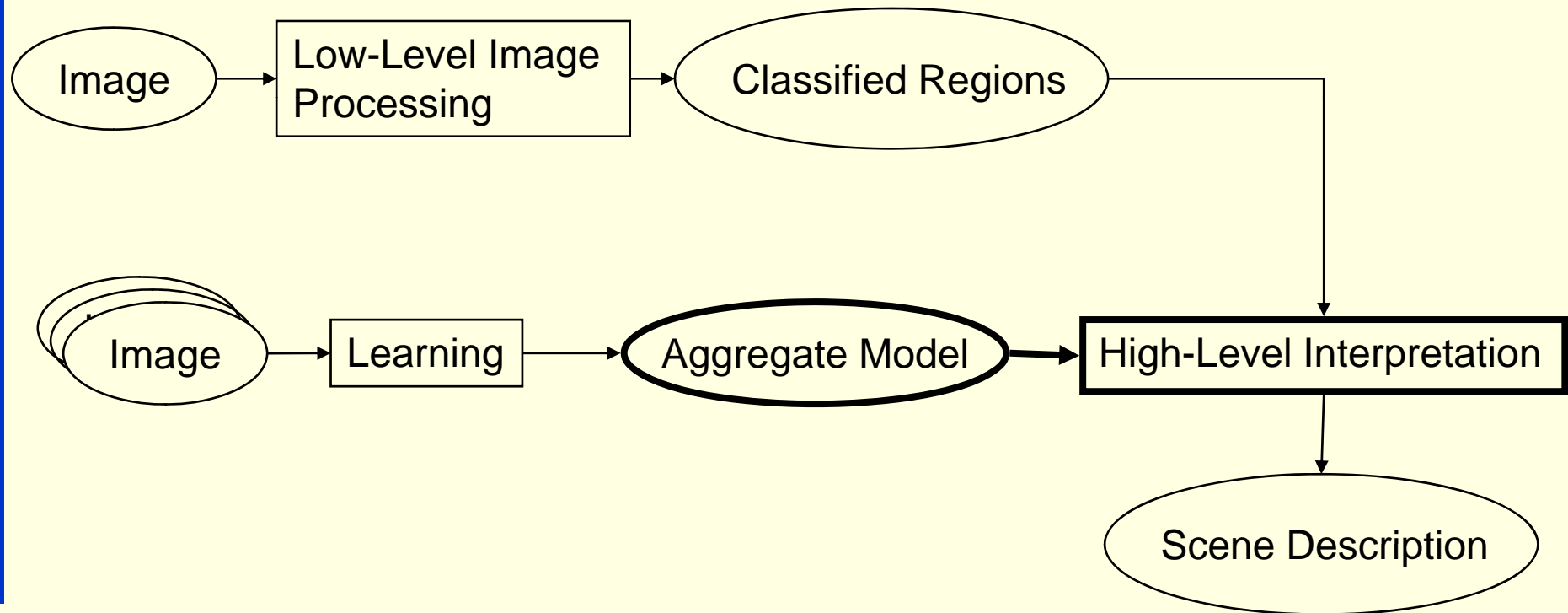
... Parts



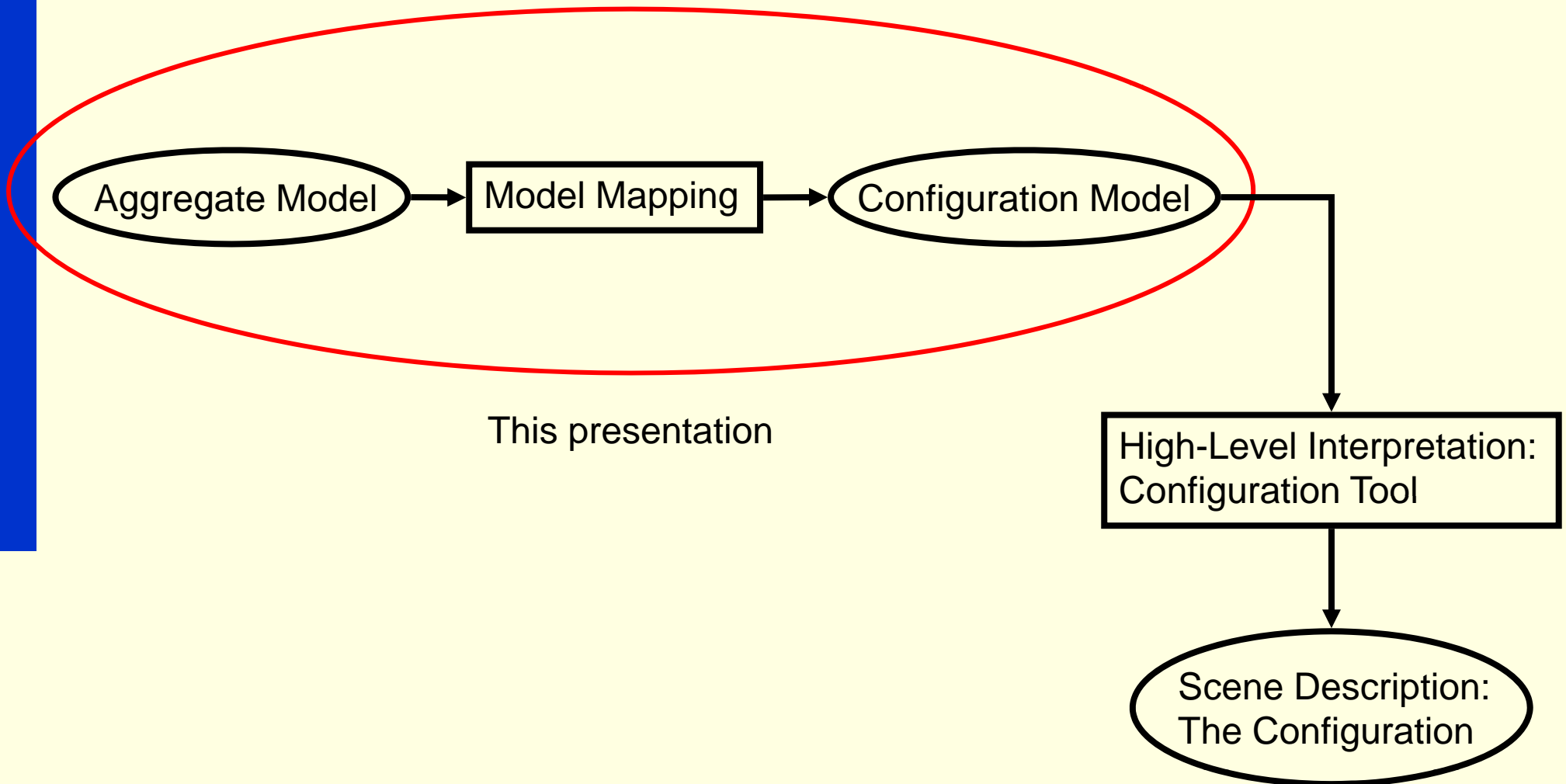
Interpretation of Images



Interpretation of Images



Interpretation of Images: **SCENIC - SCENE** Interpretation as **C**onfiguration



Structural Configuration Approach

- Descriptions of domain objects and their properties (**conceptual model**)
- **Taxonomical** and **compositional** relations, as well as definable **restrictions** between domain objects
- Knowledge about the solution procedure (**procedural knowledge**)
- A description of the purpose to be fulfilled (**task specification**)

Inference Services to Be Used

- Ontology reasoning
 - Subsumption test
 - Automatic specialization
 - Recognizing structural situations
- Constraint propagation
 - Computing variable bindings by considering relations
 - Create constraint net from structural situations
 - Arithmetic and **structural** constraint relations:
 - `create-instance`, `ensure-relation`, `less`, `greater`, `equal`,
...

The Façade Domain

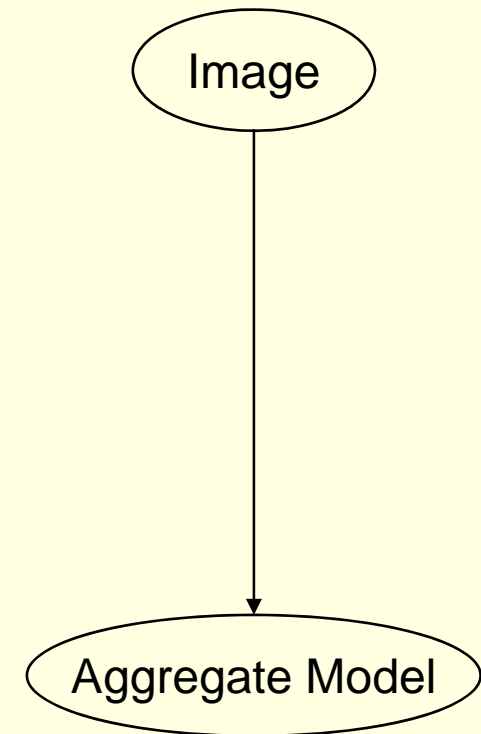
- Aggregates: balcony, entrance, façade
- Parts: door, window, sign, canopy
- Spatial Relations: In balcony the window is right-of door
- Predicates: Bounding box of aggregates covers all parts.

Modeling Guidelines

- Local representations of aggregates
- No restrictions between parts of different aggregates
- A part type may be part type of diverse aggregates
- N-ary restrictions between parts
- Discriminators that indicate restrictions or parts that are **sufficient conditions** for the existence of the aggregate.

Modeling Decisions

- Quantitative physical parameters
 - Size and position of objects
 - Implicit relations between objects
- Spatial predicates
 - Identify certain spatial relations
 - E.g. **above-p**, **left-of**
- Qualitative spatial relations
 - Explicit relations between objects
 - **o1 above o2, o2 left-of o3**
 - Used in aggregate models
 - Abstract from concrete numbers



Aggregate Representation from Learning

```
(define-aggregate :name Entrance
  :super Scene-Aggregate
  :parameters
  ((size-x [184 295])
   (size-y [299 420])
   (parts-top-left-x-variability [7 131])
   (parts-top-left-y-variability [1 284])
   (parts-bottom-right-x-variability [7 131])
   (parts-bottom-right-y-variability [1 284])
   (top-left-x [0 inf]) (top-left-y [0 inf])
   (bottom-right-x [0 inf]) (bottom-right-y [0 inf])))
  :parts
  ((:name ?stairs0 :type Stairs)
   (:name ?door1 :type Door)
   (:name ?sign2 :type Sign)
   (:name ?railing3 :type Railing)
   (:name ?canopy4 :type Canopy))
  :restrictions
  ((:name ?bn-s-d :relation belowNeighbor
    :subject ?stairs0 :object ?door1)
   (:name ?b-s-c :relation below
    :subject ?stairs0 :object ?canopy4)
   (:name ?o-s-r :relation overlap
    :subject ?stairs0 :object ?railing3)
   (:name ?bn-s-s :relation belowNeighbor
    :subject ?stairs0 :object ?sign2)
   (:name ?an-d-s :relation aboveNeighbor
```



```
(define-aggregate :name Entrance
```

```
:super Scene-Aggregate
```

```
:parameters
```

```
(/size-x [184 295]  
(size-y [299 420]))
```

```
(parts-top-left-x-variability [7 131])
```

```
(parts-top-left-y-variability [1 284])
```

```
(parts-bottom-right-x-variability [7 131])
```

```
(parts-bottom-right-y-variability [1 284])
```

```
(top-left-x [0 inf]) (top-left-y [0 inf])
```

```
(bottom-right-x [0 inf]) (bottom-right-y [0 inf])))
```

```
:parts
```

```
((:name ?stairs0 :type Stairs)
```

```
(:name ?door1 :type Door)
```

```
(:name ?sign2 :type Sign)
```

```
(:name ?railing3 :type Railing)
```

```
(:name ?canopy4 :type Canopy))
```

```
:restrictions
```

```
((:name ?bn-s-d :relation belowNeighbor
```

```
:subject ?stairs0 :object ?door1)
```

```
(:name ?b-s-c :relation below
```

```
:subject ?stairs0 :object ?canopy4)
```

```
(:name ?o-s-r :relation overlap
```

```
:subject ?stairs0 :object ?railing3)
```

```
(:name ?bn-s-s :relation belowNeighbor
```

```
:subject ?stairs0 :object ?sign2)
```

```
(:name ?an-d-s :relation aboveNeighbor
```

```
:subject ?door1 :object ?stairs0)
```

```
(:name ?bn-d-c :relation belowNeighbor
```

```
:subject ?door1 :object ?canopy4)
```

```
:discriminators
```

```
((?bn-s-d) (?b-s-c) (?o-s-r)
```

```
(?bn-s-s) (?an-d-s) (?bn-d-c)))
```

Aggregate Representation from Learning

Hartz, J., Neumann, B.: Learning a knowledge base of ontological concepts for high-level scene interpretation. In: *International Conference on Machine Learning and Applications*, Cincinnati (Ohio, USA) (December 2007)

Key Situations and Main Task

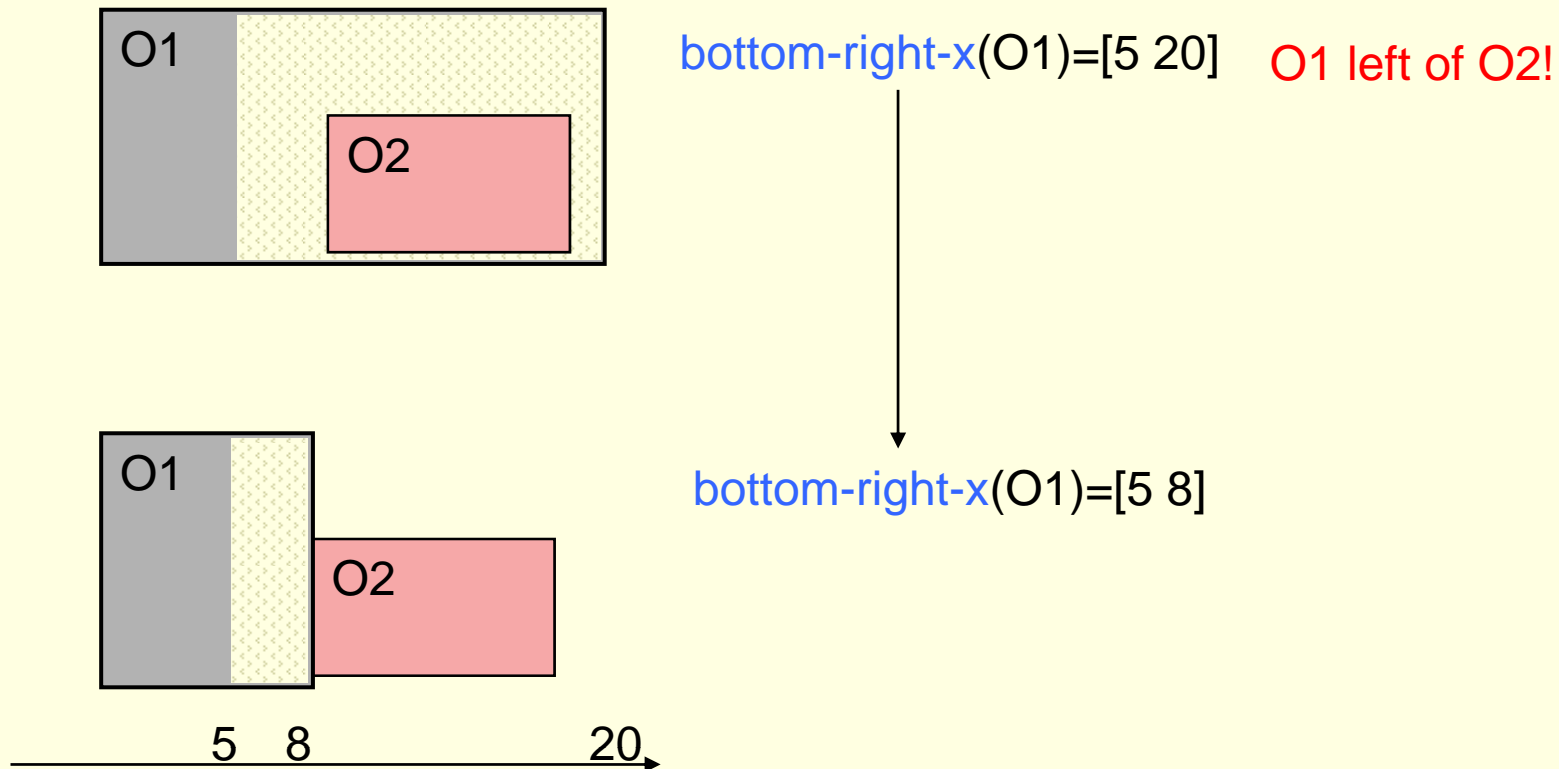
- Parts given with or without relations
- Aggregates given with or without parts
- Task:
 - Develop general aggregation reasoning chunks for each key situation, which will construct complete aggregates and which will guarantee the validity of the restrictions.

Construct Reasoning Chunks that should:

- Allow the construction of aggregates when parts are given (*bottom-up structuring*)
- Allow the construction of parts when aggregates are given (*top-down integration*)
- Integrate given parts in existing aggregates (*top-down decomposition*)
- Use given parts for decomposing existing aggregates (*top-down decomposition*)
- Check restrictions for an aggregate with parts (*aggregate consistency*)
- Establish the aggregate restrictions, if parts belong to an aggregate (*restriction establishment*)
- Determine types of parts, when not given (*object specialization*)
- Select an aggregate type, if several are possible (*type selection*)
- Map quantitative and qualitative parameters (*quantitative/qualitative mapping*)

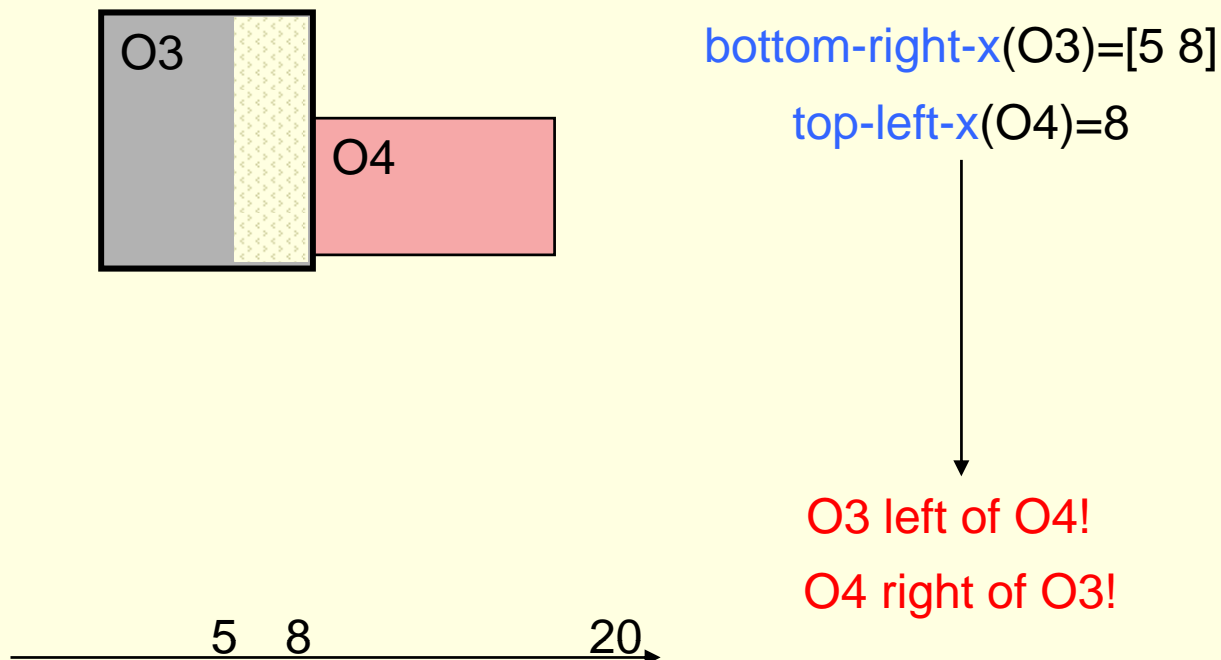
Examples

- If an **explicit relation** between objects is given, the **physical parameters** should be changed, so that the geometry holds between the parameters.



Examples

- If an implicit relation between **physical parameters** is given, the **explicit relation** between objects should be established.



Overview

- Scene Interpretation as Configuration
- **Analysis of Reasoning Tasks**
- Experiment
- Summary

Analysis

- Three dimensions of variability:
 - Aggregate structure may be given or not
 - Spatial relations may be given or not
 - Physical parameters may be given or not
 - (Part or aggregate type given or not)

Cases to be Considered

No	Aggregate structure for A and p_i	Spatial relation for p_i	Physical parameter for p_i
1	To be computed	Given	To be computed
2	To be computed	To be computed	Given
3	Given	To be computed	To be computed
4	Given	Given	To be computed
5	Given	To be computed	Given
6	To be computed	Given	Given
7	Given	Given	Given

“To be computed” = original value given in the aggregate model

The Reasoning Chunks: Case 2 and Case 5

```
(define-conceptual-constraint
  :name Spatial-Relations-from-Physical-Parameters
  :structural-situation
    ((:name ?o1      :type Scene-Object)
     (:name ?o2      :type Scene-Object
      :relations
        ((self
          #'(aboveNeighbor-p *it* ?o1))))))
  :action-part
    ((ensure-relation (above-neighbor ?o2 ?o1)
                     (below-neighbor ?o1 ?o2))))
```

For all aggregate types

Spatial predicate

Structural constraint relation

Spatial relations

The Reasoning Chunks: Case 1 and Case 4

```
(define-conceptual-constraint
  :name Physical-Parameters-from-Spatial-Relation
  :structural-situation
    ( (:name ?o1      :type Scene-Object)
      (:name ?o2      :type Scene-Object
        :relations
          ((above-neighbor *it* ?o1))))
  :action-part
    ((less (bottom-right-y ?o1) (top-left-y ?o2))
      (less (top-left-x ?o2) (top-left-x ?o1))
      (greater (bottom-right-x ?o2) (bottom-right-x ?o1))))
```

For all aggregate types

Spatial relation

Arithmetic constraint relation

Physical parameters

The Reasoning Chunks: Case 3

```
(define-conceptual-constraint
  :name Entrance-Spatial-Relation
  :structural-situation
    ((:name ?e      :type Entrance)
     (:name ?stairs0 :type Stairs
      :relations ((part-of ?e)))
     (:name ?door1  :type Door
      :relations ((part-of ?e))))
  :action-part
  ((ensure-relation
    (above-neighbor ?door1 ?stairs0)
    (below-neighbor ?stairs0 ?door1))))
```

Compositional relation known

Establish spatial relations

Specific for the aggregate entrance.

One reasoning chunk needed for each restriction.

The Reasoning Chunks: Case 6

```
(define-conceptual-constraint
  :name Entrance-Creation
  :structural-situation
    ((:name ?stairs0 :type Stairs
      :relations
        ((part-of #'(free-p *it*))))
     (:name ?door1 :type Door
      :relations
        ((part-of #'(free-p *it*))
         (above-neighbor ?stairs0))))
  :action-part
  ((create-instance Entrance
    (part-of ?stairs0)
    (part-of ?door1))))
```

Spatial relations of parts known

Create aggregate

The Reasoning Chunks: Case 6

```
(define-conceptual-constraint
  :name Entrance-Terrace-Creation
  :structural-situation
    ((:name ?stairs0 :type Stairs
      :relations
        ((part-of #'(free-p *it*))))
     (:name ?door1 :type Door
      :relations
        ((part-of #'(free-p *it*))
         (above-neighbor ?stairs0))))
  :action-part
  ((create-instance (Entrance Terrace)
    (part-of ?stairs0)
    (part-of ?door1))))
```

Spatial relations of parts known

Create aggregate

Multiple aggregates may have same relations.

The Reasoning Chunks: Case 4, Part Integration

```
(define-conceptual-constraint
  :name Entrance-Has-Parts-Door-Stairs
  :structural-situation
    ((:name ?stairs0 :type Stairs)
     (:name ?door1 :type Door
      :relations
        ((aboveNeighbor ?stairs0)))
     (:name ?e :type Entrance
      :relations
        ((has-parts
          #'(free-or-in-agg-p ?stairs ?doors1 *it*)
          #'(check-bounding-box *it*
            ?stairs0 ?door1))))))
  :action-part
  ((ensure-relation (part-of ?stairs0 ?e)
                   (has-parts ?e ?stairs0))
   (ensure-relation (part-of ?door1 ?e)
                    (has-parts ?e ?door1))))
```

Parts are in spatial relations

Parts fit to aggregate

Summary of Reasoning Chunks

1. Quantitative parameters → Explicit relations
2. Explicit relations → Quantitative parameters
3. Discriminative explicit relations → Aggregate creation
4. All given → Check restrictions
5. Potential parts and aggregates → Integrate while considering aggregate restrictions

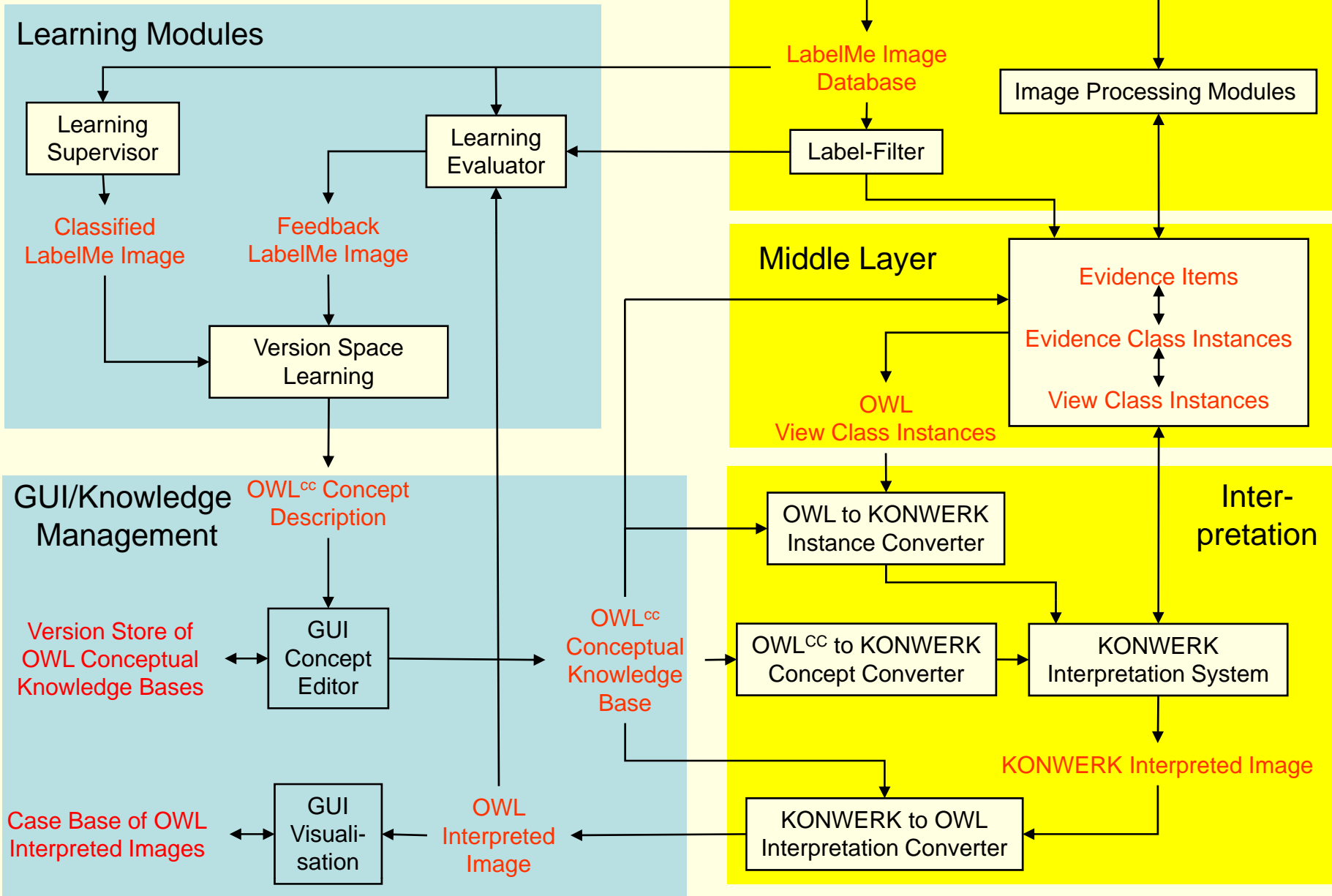
Overview

- Scene Interpretation as Configuration
- Analysis of Reasoning Tasks
- **Experiment**
- Summary

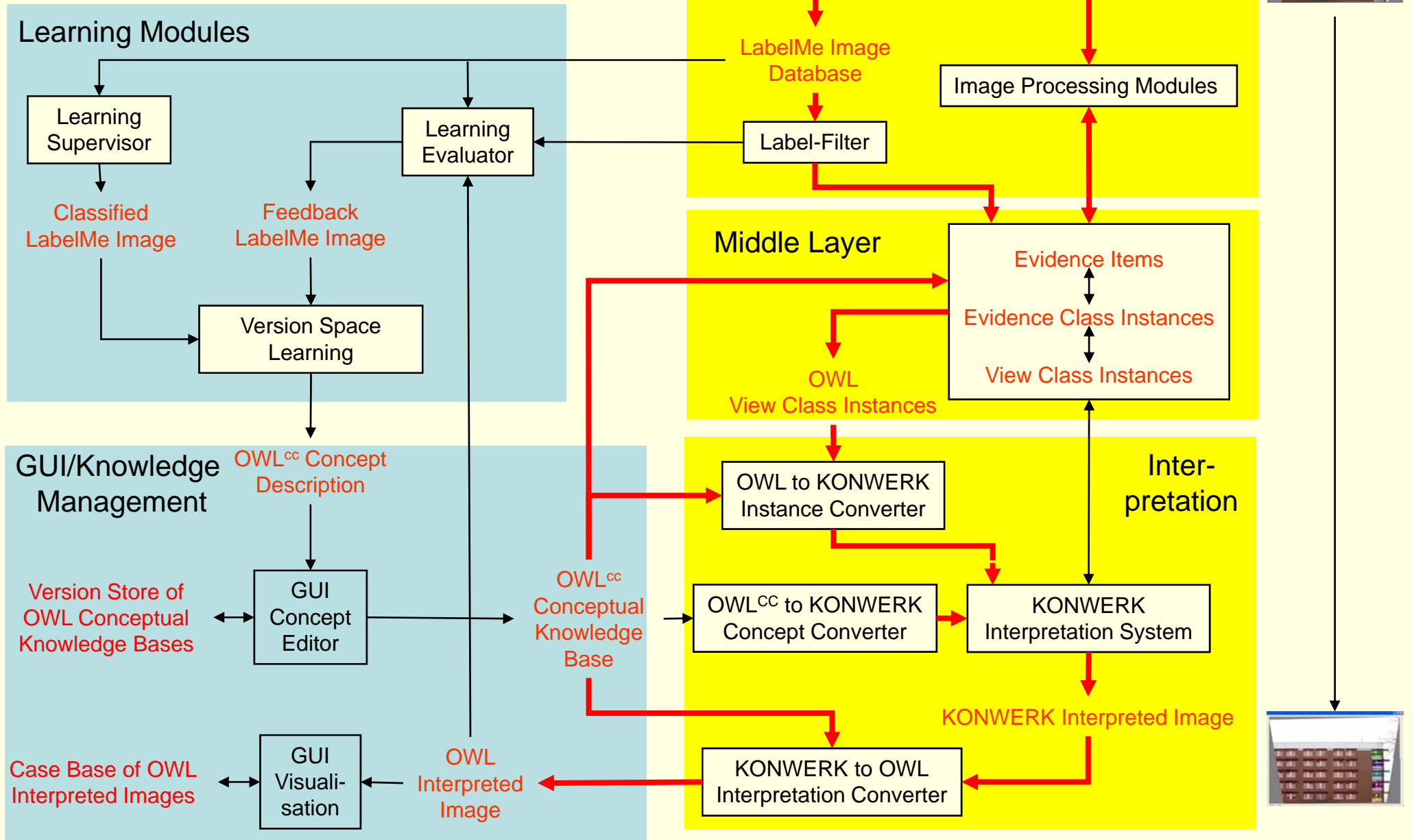
Experiment: Use Mapping for Iteratively Refining Concept Models during Feedback Learning

- Learn balcony model
- Map it to reasoning chunks
- Interpret new image
- Evaluate result
- Feedback learning
- Map new balcony model to reasoning chunks
- Interpret the image again
- Evaluate result
- ...

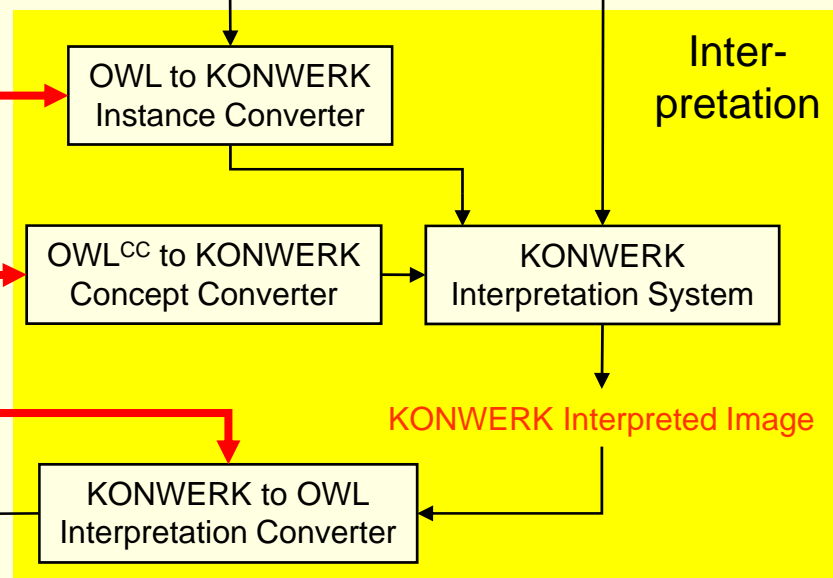
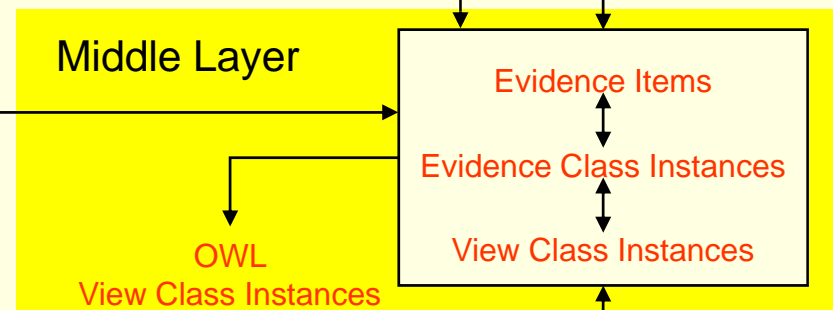
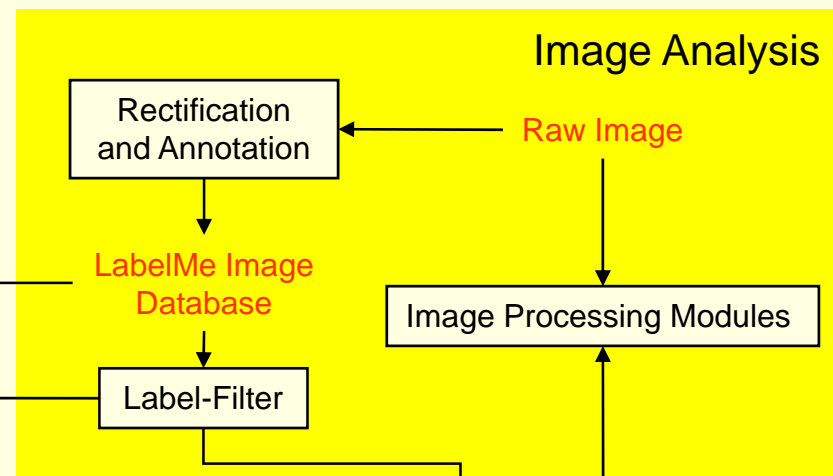
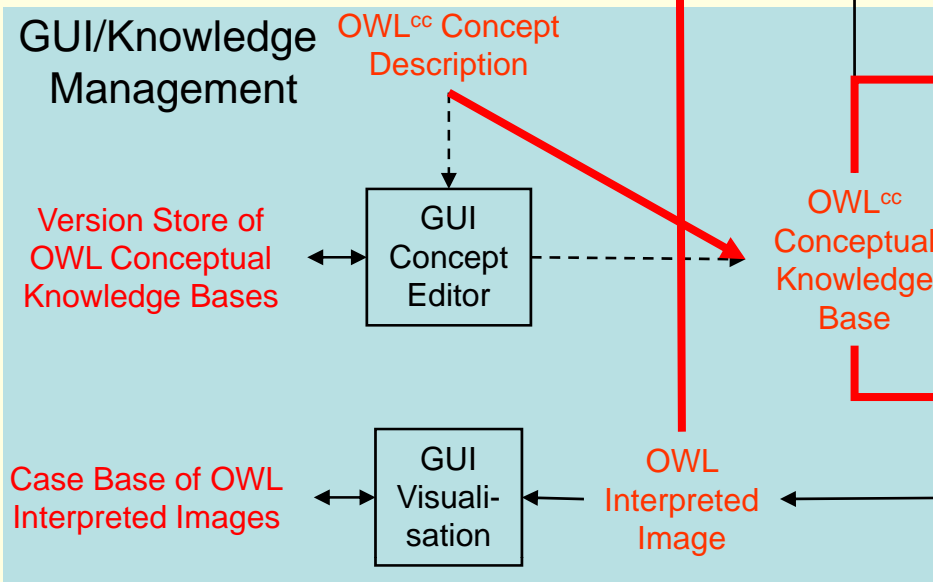
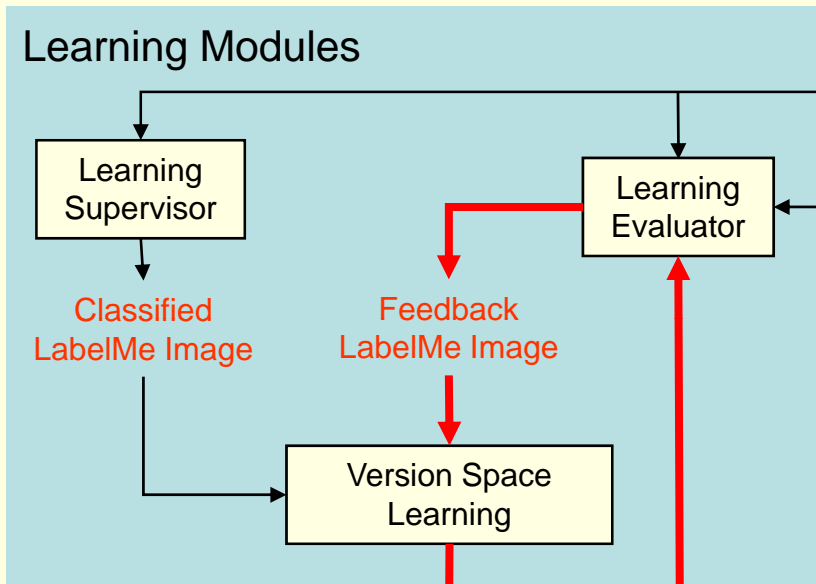
System Overview



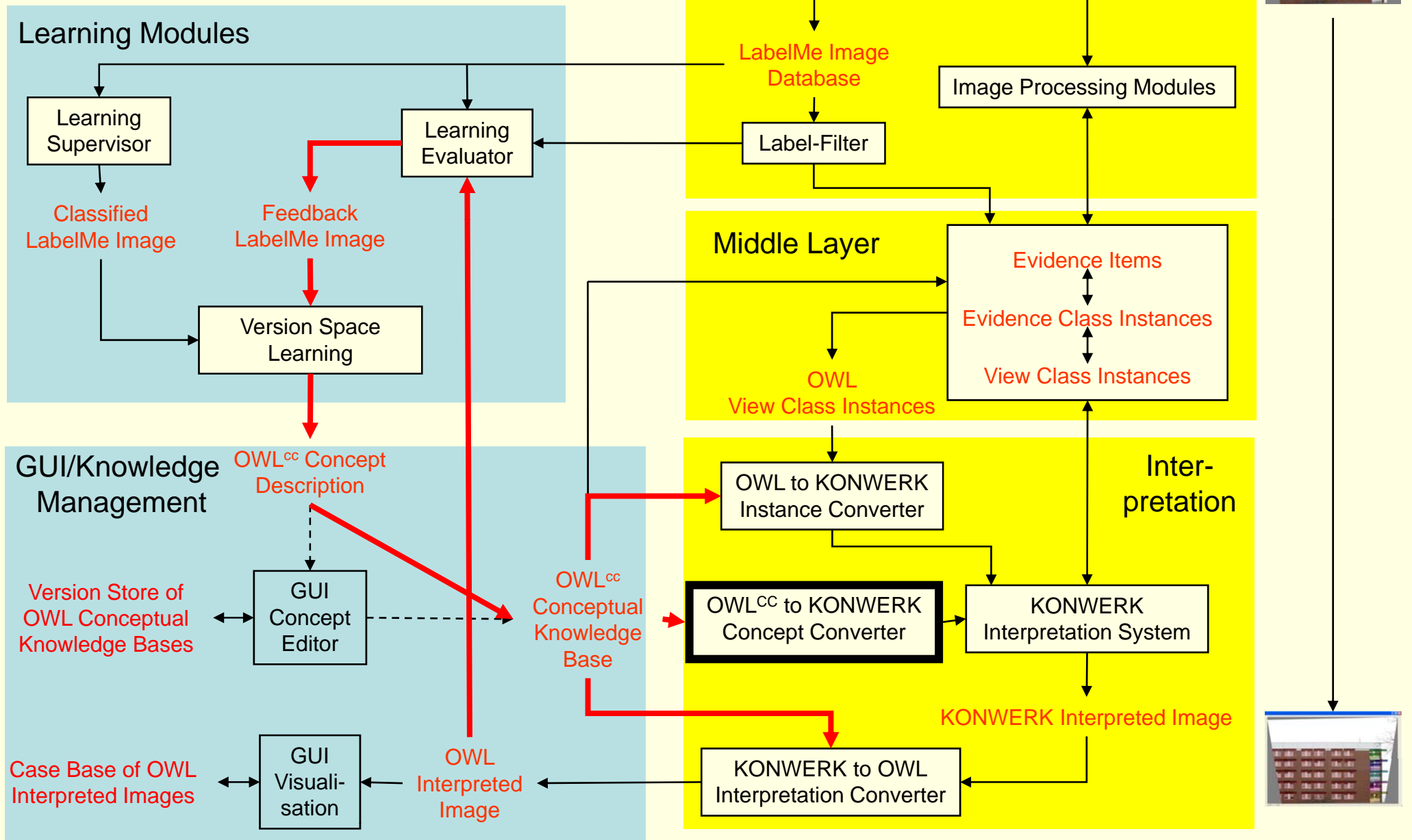
Interpretation Process



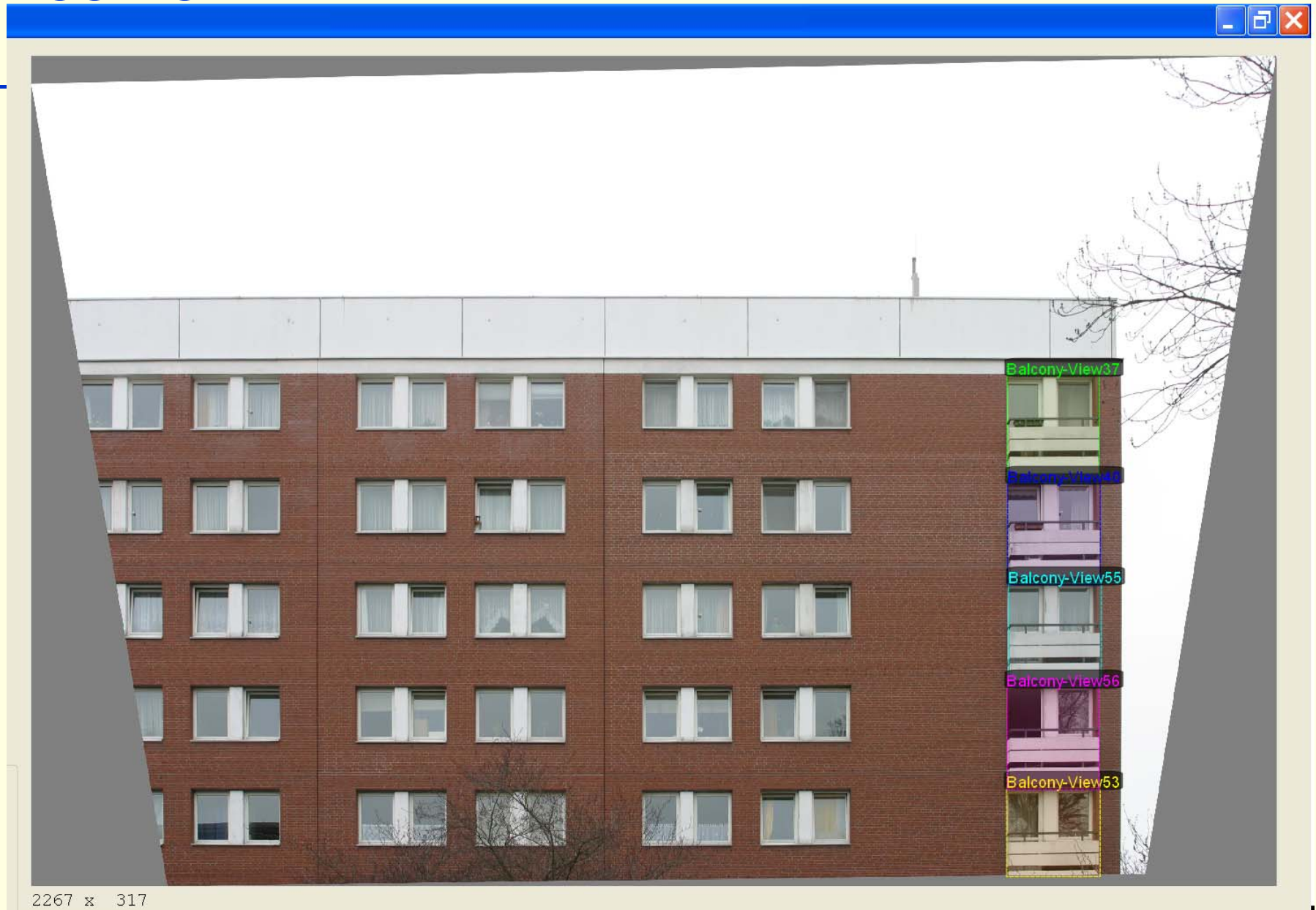
Feedback Learning



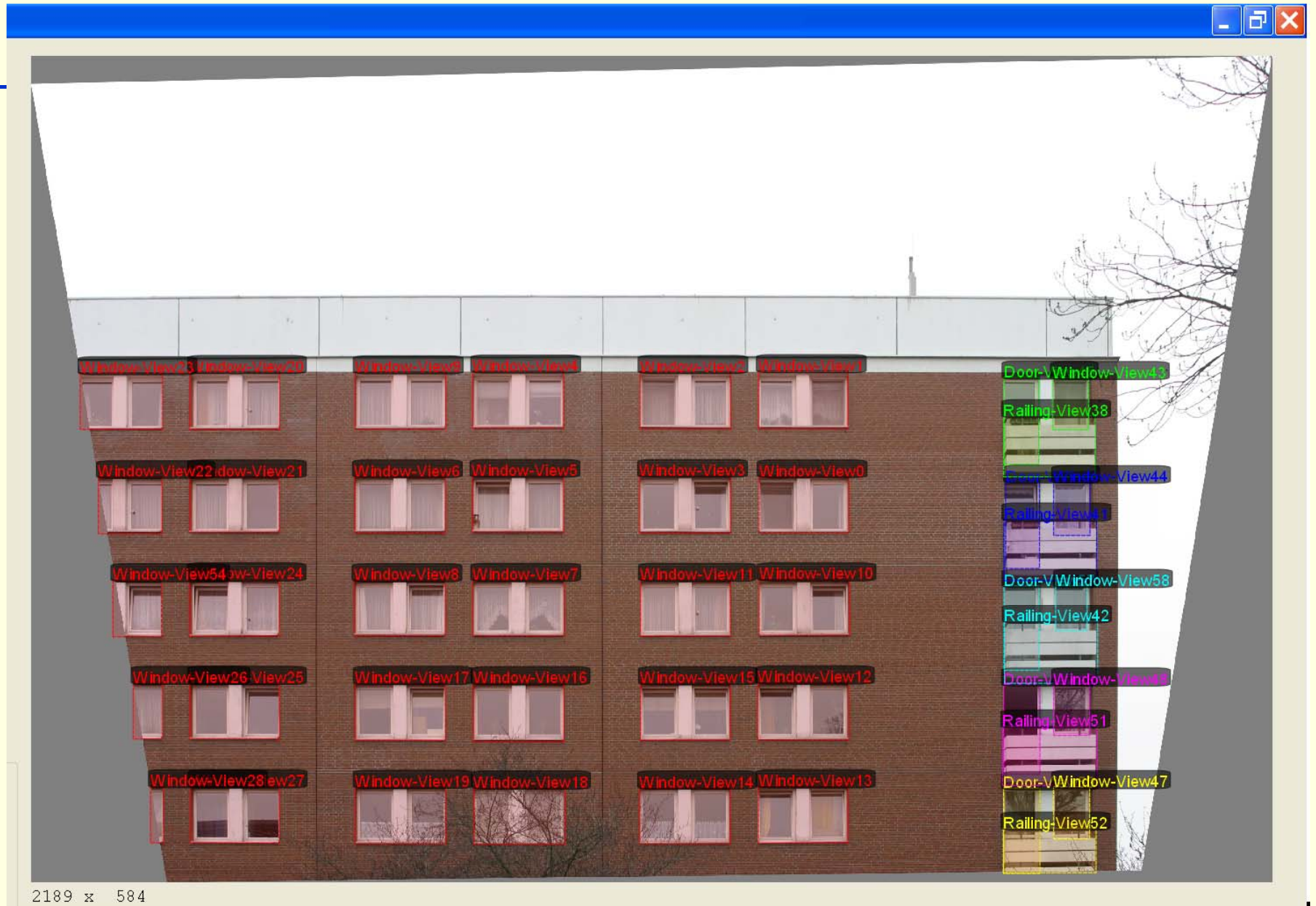
Feedback Learning



Aggregates and...



... Parts



Initial Learnt Balcony Concept:

```
(define-aggregate :name balcony-12088
  :super Scene-Aggregate
  :parameters
  ((Parts-Top-Left-X-Variability [0.0000 1425.5403])
   (Parts-Top-Left-Y-Variability [0 INF])
   (Parts-Bottom-Right-X-Variability [0.0000 1605.6350])
   (Parts-Bottom-Right-Y-Variability [1.0000 INF])
   ...)
  :parts
  ((:type Scene-Object :number-restriction [0 INF])
   (:type door :number-restriction [1.0000 INF])
   (:type railing :number-restriction [1.0000 INF])
   (:type window :number-restriction [0.0000 5.0000]))
  :restrictions
  ((:relation Overlap :subject ?railing0 :object ?window2)
   (:relation Overlap :subject ?railing0 :object ?door1)
   (:relation Overlap :subject ?window2 :object ?railing0)
   (:relation RightNeighbour :subject ?window2 :object
                                ?door1)
   (:relation Overlap :subject ?door1 :object ?railing0)
   (:relation LeftNeighbour :subject ?door1 :object
                                ?window2)))
```



```

:constraint-aufrufe
((integrate-instances (?Balcony has-elements)
  (?Balcony self) (?railing0 self) (?railing0 element-of))
 (integrate-instances (?Balcony has-elements)
  (?Balcony self) (?window2 self) (?window2 element-of))))

```

Reasoning Chunks

```

(define-conceptual-constraint :name restriction3-trigger-cc
:structural-situation
((?door1 :name Door :relationen
  ((element-of (:condition '(free-p *it*))))))
(?window2 :name Window :relationen
  ((element-of (:condition '(free-p *it*)))
  (self (:condition
    '(check-variability-and-bb-p-concept 'Balcony *it*
      ?door1))))
  (right-neighbour
    (:condition '(aggs::is-in-set ?door1 *it*))))))
:constraint-aufrufe
((create-instance 'Balcony 'has-elements (?door1 self)
  (?window2 self) (?door1 element-of) (?window2 element-of)))

```

```

(define-conceptual-constraint :name
Balcony-restriction3-spatial-relation-cc
:structural-situation
((?Balcony :name Balcony)
  (?door1 :name Door :relationen
    ((element-of

```

Given: Only parts and no balcony, but Façade.

python



Display Options

- Show Labels
- Show Views
- Show Primitives
- Show Aggregates
- Show Non-Buildings

Evidence:

Views:

- Building-View13
- Door-View10
- Door-View7
- Facade-View11
- Railing-View14
- Railing-View8
- Roof-View16
- Roof-View17
- Vegetation-Object-View
- Window-View0
- Window-View1
- Window-View15
- Window-View18
- Window-View19
- Window-View2
- Window-View3
- Window-View4
- Window-View5

Given: Only parts and no balcony, but Façade.

python



Display Options

- Show Labels
- Show Views
- Show Primitives
- Show Aggregates
- Show Non-Buildings

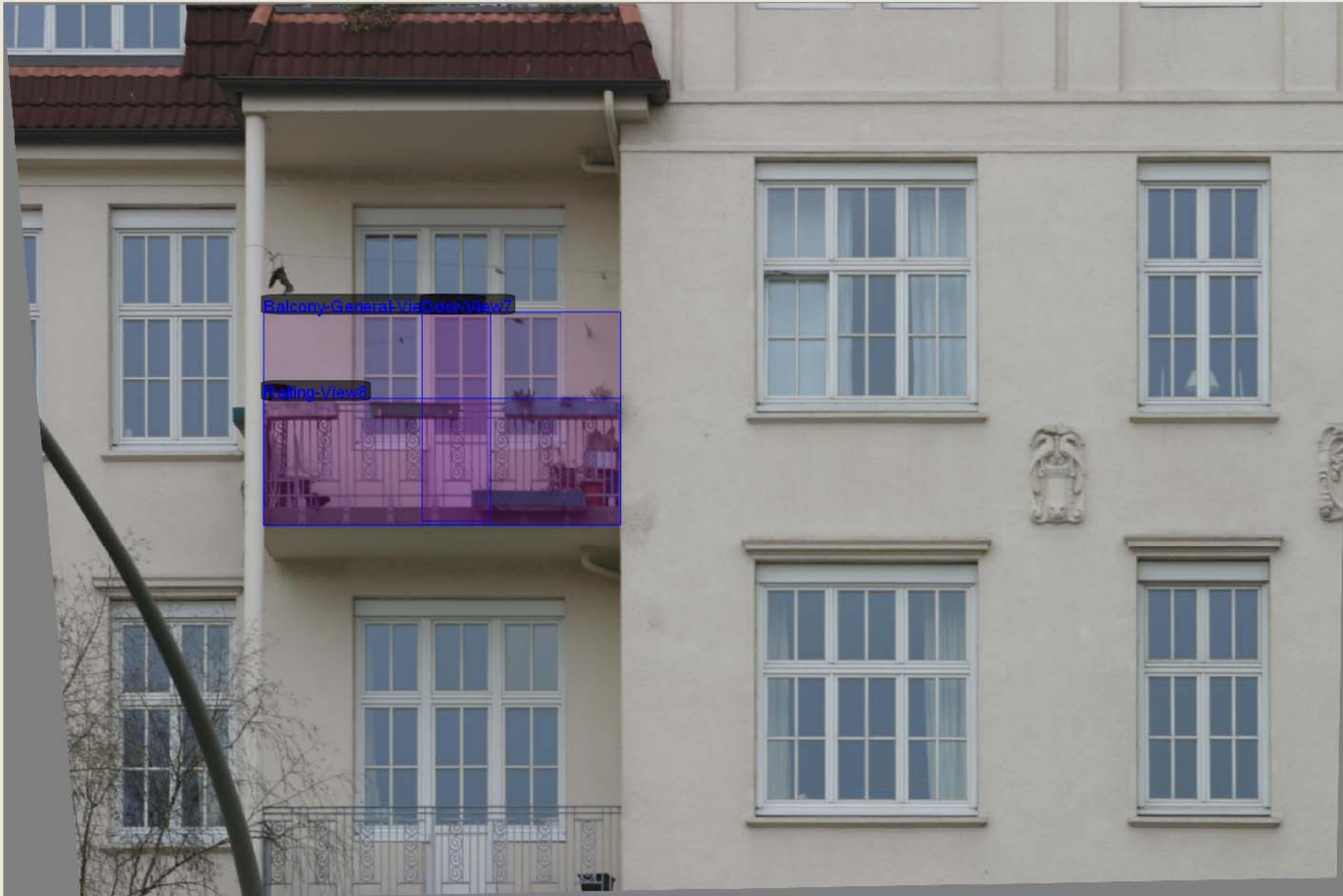
Evidence:

Views:

- Building-View13
- Door-View10
- Door-View7
- Facade-View11
- Railing-View14
- Railing-View8
- Roof-View16
- Roof-View17
- Vegetation-Object-View
- Window-View0
- Window-View1
- Window-View15
- Window-View18
- Window-View19
- Window-View2
- Window-View3
- Window-View4
- Window-View5

Result: Identified Balcony but with wrong parts, because of non-appropriate model

python



Display Options

- Show Labels
- Show Views
- Show Primitives
- Show Aggregates
- Show Non-Buildings

Evidence:

Views:

- Balcony-General-View
- Balcony-General-View
- Building-View13
- Door-View10
- Door-View10007
- Door-View7
- Facade-View11
- Railing-View14
- Railing-View8
- Roof-View16
- Roof-View17
- Vegetation-Object-View
- Window-View0
- Window-View1
- Window-View15
- Window-View18
- Window-View19
- Window-View2
- Window-View3
- Window-View4
- Window-View5



Balcony Concept after Feedback Learning

```
(define-aggregate :name Balcony-boundary-conjunction-34510
  :super Scene-Aggregate
  :parameters
    ((Parts-Top-Left-X-Variability [0.0000 1425.5403])
     (Parts-Top-Left-Y-Variability [0.0000 525.0465])
     (Parts-Bottom-Right-X-Variability [7.0465 1605.6350])
     (Parts-Bottom-Right-Y-Variability [1.0000 472.0233])
     ...)
  :parts
    ((:type door :number-restriction [1.0000 1.0000])
     (:type railing :number-restriction [1.0000 3.0000])
     (:type roof :number-restriction [0.0000 3.0000])
     (:type stairs :number-restriction [0.0000 3.0000])
     ...)
  :restrictions
    ((:relation Overlap :subject ?railing39 :object
      ?door46)
     (:relation Overlap :subject ?railing39 :object
      ?window44)
     (:relation Overlap :subject ?door46 :object
      ?railing39)
     (:relation Overlap :subject ?window44 :object
      ?railing39))
```

Result: Identified Balcony with correct parts

python



- Display Options
- Show Labels
 - Show Views
 - Show Primitives
 - Show Aggregates
 - Show Non-Buildings

Evidence:

Views:

- Balcony-General-View
- Balcony-General-View
- Building-View13
- Door-View10
- Door-View7
- Facade-View11
- Railing-View14
- Railing-View8
- Roof-View16
- Roof-View17
- Vegetation-Object-View
- Window-View0
- Window-View1
- Window-View15
- Window-View18
- Window-View19
- Window-View?
- Window-View3
- Window-View4
- Window-View5

Result: Identified Balcony with correct parts

python



- Display Options
- Show Labels
 - Show Views
 - Show Primitives
 - Show Aggregates
 - Show Non-Buildings

Evidence:

Views:

- Balcony-General-View
- Balcony-General-View
- Building-View13
- Door-View10
- Door-View7
- Facade-View11
- Railing-View14
- Railing-View8
- Roof-View16
- Roof-View17
- Vegetation-Object-View
- Window-View0
- Window-View1
- Window-View15
- Window-View18
- Window-View19
- Window-View2
- Window-View3
- Window-View4
- Window-View5

Summary

- Interpretation of images as configuration task
- Integration of learnt aggregate models in the configuration model through reasoning chunks
- Map quantitative parameters from sensors to qualitative relations for modeling
- Reasoning chunks compute all kinds of entities: qualitative relations, quantitative parameters, aggregates, and parts
- Domain independent mapping given
- May be applicable to pure rule systems
- Next step:
 - *Exhaustive experiments for learning diverse aggregates*



Thank You for Your Attention

Lothar Hotz

HITeC e.V., University of Hamburg

hotz@informatik.uni-hamburg.de