

Towards the Implementation of an Ontology-Based Reasoning System for Visual Information Fusion

Juan Gómez-Romero, Jesús García, Miguel A. Patricio, José M. Molina
{jgromero, jgherrer, mpatrici}@inf.uc3m.es, molina@ia.uc3m.es

Applied Artificial Intelligence Group, Department of Computer Science, Univ. Carlos III of Madrid

Abstract—High-level information fusion requires the use of abstract knowledge representation formalisms, such as ontologies. In previous research works, we proposed an ontology-based framework for visual information fusion aimed at contextual recognition of activities and enhancement of basic tracking procedures. In this paper, we describe various aspects of the implementation of this framework, focusing on the development of the supporting rule base.

Index Terms—Visual Information Fusion, Ontologies, Reasoning, Context-Awareness

I. INTRODUCTION

IN the last years, the interest of the Fusion community is shifting from low-level data integration to high-level information assessment. High-level fusion poses considerable challenges: assessment entails achieving a certain degree of *understanding* of the knowledge present in heterogeneous sources, instead of only *combining* sensor data. Accordingly, it has been pointed out that high-level information fusion requires the use of semantic models to acquire, represent, and exploit knowledge [1]. Besides domain-specific knowledge, some authors have also highlighted the importance of formalizing the management of context knowledge, especially when visual inputs need to be interpreted [2].

Ontologies are a widespread formalism for context knowledge representation and reasoning in high-level information fusion. Among other advantages, ontologies promote reusability, interoperability, and standardization. Various authors have described innovative proposals that use ontologies for situation recognition from general inputs [3, 4] and visual data [5]. Nevertheless, practical implementations of ontology-based fusion systems are relatively scarce or undocumented.

In previous research works, we proposed a framework for visual information fusion that, relying on contextual information, builds a symbolic model of the scene from tracking data [6] [7]. The knowledge of the framework is represented with formal ontologies, and reasoning procedures are executed in order to:

- Obtain a high-level interpretation of the perceived situation.
- Provide feedback to the low-level tracking procedure to improve its accuracy and performance.

In this paper, we describe some experiences learned in the

development of this framework, focusing on the problems concerning the implementation of reasoning processes. Interestingly enough, the obtained results are general, and can be taken into account when implementing any other ontology-based fusion system.

The paper is organized as follows. First, we summarize the architecture of the framework and the structure of the knowledge model, and give an overview of their implementation. Next, we describe various types of reasoning tasks within the scene model: information retrieval, arithmetical calculus, spatial and temporal reasoning, deduction, and abduction. Finally, we depict an example that shows how reasoning is triggered within the framework. The paper finishes with some conclusions and directions for future work.

II. A FRAMEWORK FOR VISUAL INFORMATION FUSION

In [7], we described the architecture of our framework for visual information fusion. The framework encompasses two components, the General Tracking Layer (GTL) and the Context Layer (CL), which communicate through the GTL/CL interface. The GTL is a classical tracker executing image-processing algorithms to segment video frames, distinguishing moving objects from background and labeling them consistently [8]. The CL is an extension of the GTL that applies formal reasoning to interpret and correct the values provided by the tracker. The CL manages the CL model, an abstract ontology-based representation of the scene, and creates recommendations to be executed by the GTL in order to improve its functioning.

Architecture. Fig. 1 depicts the functional architecture of the framework and the layered structure of the context model. Briefly, the framework works as follows:

1. The GTL calls the GTL/CL interface when tracks are created, deleted, or updated, and sends the modified track data.
2. The GTL/CL interface transforms GTL numeric data to CL symbols, and updates the low-level representation of tracks in the model.
3. Updated track information triggers rule-based reasoning processes in the CL that, supported by context knowledge, revise the whole interpretation of the scene.

4. Recommendations for the GTL are created by means of rule-based reasoning with the current scene model.
5. The GTL calls the GTL/CL interface methods to consult recommendations generated by the CL, and converts them to concrete actions.

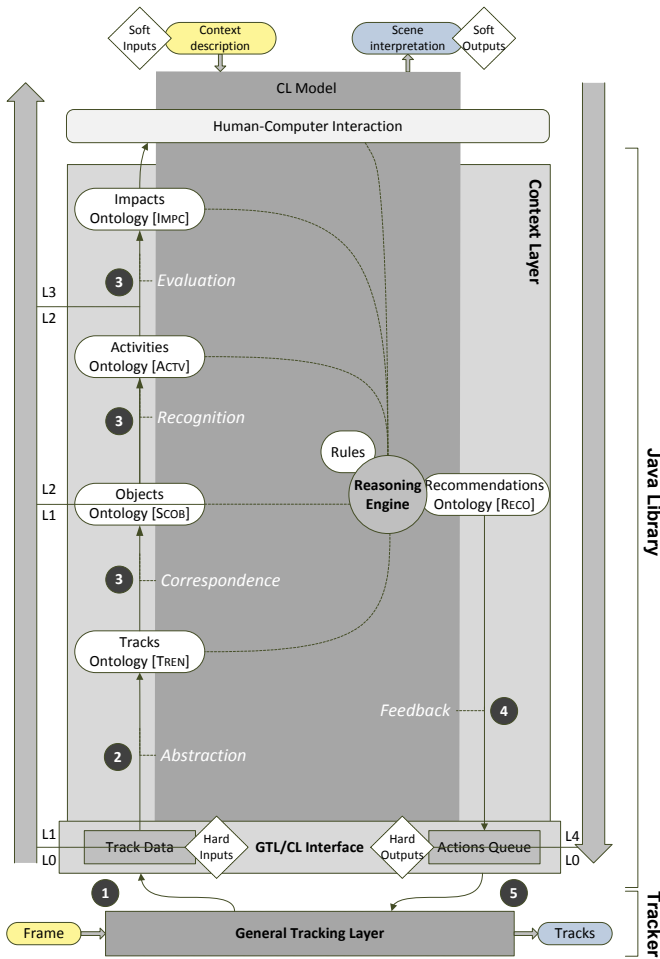


Fig. 1. Architecture of the Visual Information Fusion Framework

The knowledge model of the framework is structured in five different interrelated ontologies, which have been designed in compliance with the JDL model [9] from less to more abstract:

- Camera data (L0). This data is not managed directly in the CL, but in the GTL. Actually, the live or recorded sequence (in a suitable video format) is the input which is analyzed by the GTL.
- Tracking data (L1). The TREN ontology represents entity data, which corresponds to the output of the GTL expressed in ontological terms after the abstraction step. The main concept of this ontology is Track, which has associated track properties (color, position, velocity, etc.). The ontology represents the current and the previous track property values.
- Scene objects (L1-L1½). The SCOB ontology represents scene objects, which are the result of making a correspondence between existing tracks and possible scene entities. For instance, a Track can be inferred to

be associated to a Person. SceneObjects can also be static elements, which may be defined a priori.

- Activities (L2): The ACTV ontology represents the behaviors of the objects of the scene. Activities describe relations between SceneObjects that last in time, e.g. grouping or approaching. Activities are recognized by interpreting object properties in context.
- Impacts and threats (L3). The IMPC ontology represents associations between an Activity and Impact values. These assessments are obtained after evaluating the risk of the recognized activities.
- Feedback and process improvement (L4). The RECO ontology represents abstractly Actions to improve the accuracy of the GTL, and consequently, the quality of the measures of the fusion system.

The CL model distinguishes between general and specific knowledge. General knowledge is common to several applications, whereas specific knowledge is only useful in certain domains. Each one of the ontologies can be considered to have a general part and a specific part. This separation allows using the framework in different application areas, since it facilitates the specialization of the architecture and the knowledge model to particular requirements. An initial version of the general ontologies and an example of domain-specific model can be found in the authors' web page¹.

Ontological reasoning procedures are performed within the CL ontologies to infer implicit knowledge from explicitly asserted facts. Tasks such as classification and instance checking can be performed with a Description Logics reasoning engine [10], as well as safe rule-based reasoning [11]. These are intra-ontology reasoning procedures, since they *deduce* knowledge of an abstraction level from knowledge of the same level. In contrast, scene interpretation requires the *creation* of higher-level knowledge from lower-level information: scene objects from tracks, activities from objects, etc. These are inter-ontology reasoning procedures, and can be regarded as abductive reasoning. Abduction is out of the scope of canonical ontologies [12], but it can be simulated in some reasoners by defining non-standard inference rules. Abduction is also used to generate GTL recommendations from the current interpretation of the scene. Section III describes the formulation of abductive rules in our framework, and section IV depicts an example of rule-based reasoning.

Implementation. The framework is implemented as a Java library, which is invoked from the GTL C++ code through JNI (Java Native Interface). The library offers public methods for initializing the scene model, updating the representation, and retrieving suggested actions, composing the GTL/CL interface. The library also contains private methods to manage the ontologies of the CL model –based on the OWL API²–, and communicate with the inference engine –in our case the

¹ <http://www.giaa.inf.uc3m.es/miembros/jgomez/ontologies/ontologies.html>
² <http://owlapi.sourceforge.net/>

RACER reasoner³.

Before the GTL starts the processing of the video sequence, the CL must be started. This requires loading the CL model into the reasoning engine, including the general and the specialized ontologies. Additionally, the (specialized) model may include preliminary instances describing the context of the scenario, e.g. static object features. Inference rules must also be loaded into the inference engine. By default, the reasoner is configured to add the consequences of the rules into the ontology immediately after they are fired.

Once the CL has been initialized, the GTL can begin the processing of the video sequence. When analyzing a frame, it track data changes, the GTL invokes the public methods of the library to update the CL model. The interface methods transform the tracking data provided by the GTL in ontology instances, and insert them in the TREN ontology. If these instances match some of the (interpretation or feedback) rules of the model, they are fired and new instances may be created. This process can be repeated several times automatically. In this manner, the CL ontologies are instantiated, composing an abstract description of the current scene.

The eventual results of the update-and-reason process are two: the interpretation of the scene (in terms of SceneObject, Activity, and Impact instances) and the creation of GTL recommendations (in terms of Action instances). The interpretation may be consulted by a human analyst or reused in other fusion process, whereas the actions are placed in the recommendations queue. The GTL retrieves and decode the actions, and executes appropriate procedures to correct its behavior according to the recommendations. The queue can be consulted synchronously (after the CL has finished processing an update) or asynchronously (at any time), in which case it is necessary to add a timestamp to the recommendations and implement a more complex retrieval policy.

III. CONTEXTUAL RULE-BASED ONTOLOGICAL REASONING

As introduced in the previous section, reasoning in the framework has been implemented with the RACER reasoner. RACER supports transparently standard (deductive) and non-standard (abductive) ontology-based reasoning with nRQL (new RACER Query Language) [13]. nRQL is a modification and query language with Lisp-like syntax that supports instance querying and rule definition. In this section, we explain some nRQL features used in the implementation of the framework, which can be extended to other fusion applications. We present some examples developed for the CL example ontologies published on the web that can be directly introduced in RACER.

A. Model Query

The basic expressions in nRQL are instance queries. Instance queries contain a set of variables (e.g. $?x$), which are bound to the named instances of the ontology. The result of the query is the set of bindings of the variables in the head that satisfy the logical condition of the body. For example, the

following query finds all the instances of SceneObject that are being observed in the current scene (i.e., they have an unknown value in the end of validity property):

```

Query 1
(retrieve
 (?x)
 (and
  (?x scob4:SceneObject)
  (?x ?s scob:hasObjectSnapshot)
  (?s scob:SceneObjectSnapshot)
  (?s tren:unknown_frame tren:isValidInEnd)))

```

Matching between ontology instances and variables is shown in Fig. 2. We assume that *mirror1* has been asserted to be an instance of the Mirror concept, and *o* is a SceneObject. *a* and *b* are SceneObjectSnapshot instances related to *o* and *mirror1* with hasObjectSnapshot property, and to *f* and unknown_frame with isValidInEnd property. Since *a* is not related to *unknown_frame*, it does not match the query pattern, and the result of the consult is only the binding $(?x, mirror1)$.

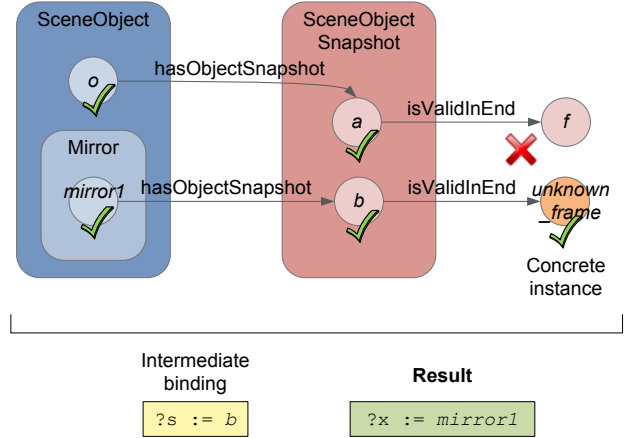


Fig. 2. Matching between variables and ontology instances in nRQL

It is also possible to create defined queries, which are stored queries that can be nested inside other queries. The following instruction defines a stored query that retrieves the current position of an object.

```

Query 2
(defquery current-position-of-object (?o ?op)
 (and
  (?o scob:SceneObject)
  (?o ?osn scob:hasObjectSnapshot)
  (?osn tren:unknown_frame tren:isValidInEnd)
  (?osn ?opr scob:hasObjectProperties)
  (?opr ?op scob:OhasPosition)
  (?op scob:OPosition)))

```

Defined queries can be used in subsequent consults. Analogous stored queries can also be created for convenience to perform other common consults (current position of a track, current area, etc.).

³ <http://www.sts.tu-harburg.de/~r.f.moeller/racer/>

⁴ For the sake of simplicity, the namespaces of the ontologies are shortened.

B. Arithmetical Calculus

More complex queries can be created to compute arithmetical functions on the values of the CL instances. nRQL admits the use of lambda calculus expressions to denote computations on the retrieved values, both in the head and the conditions of the queries. The next query calculates the Euclidean distance between the points $p5 = (445, 100)$ and $p6 = (365, 105)$, instances of 2DPoint. The query can easily be adapted to obtain the distance between any pair of 2D points of the scene, and more generally, to retrieve other object and track numerical properties:

Query 3

```
(retrieve
 (?p1 ?p2
  (
    (lambda
      (x1 x2 y1 y2)
      (expt
        (+
          (expt
            (- (first x1) (first x2)) 2)
            (expt
              (- (first y1) (first y2)) 2)
            )
          1/2)
        )
      )
    (datatype-fillers (tren:x ?p1))
    (datatype-fillers (tren:x ?p2))
    (datatype-fillers (tren:y ?p1))
    (datatype-fillers (tren:y ?p2))
  )
  )
  (and
    (?p1      tren:2DPoint)
    (?p2      tren:2DPoint)
    (same-as ?p1 surv:p5)
    (same-as ?p2 surv:p6)))
```

C. Spatial and Temporal Reasoning

Spatial reasoning is essential in a Computer Vision system. Topological reasoning can be implemented based on the native support of RACER for RCC-based predicates. RCC (Region Connection Calculus) is a logic-based formalism to symbolically represent and reason with topological properties of objects [14]. RCC semantics cannot be fully represented with ontologies [15], but RACER includes support for them through the definition of an additional layer (*substratum*) and the use of special query symbols.

The use of RCC predicates in RACER with nRQL is shown in the next query. First, the RCC substrate is activated. Then, the property *insideOf*, defined in the SCOB ontology, is stated as equivalent to the RCC predicates *ntpp* (inclusion of the first object in the second one without connection) and *tpp* (inclusion of the first object in the second one with tangential connection). Finally, all the instances of *Person* located inside a *Building* are retrieved. With the RCC substrate activated, if a person p is asserted to be inside a room r , and the room is inside of a building b , query 4 retrieves p , according to the semantics of *ntpp* and *tpp*.

Query 4

```
(enable-rcc-substrate-mirroring)

(rcc-synonym scob:insideOf (:ntpp :tpp))

(retrieve
 (?*x ?*y)
 (and
  (?*x ?*y scob:insideOf)
  (?*x     scob:Person)
  (?*y     scob:Building)))
```

RCC semantics can also be used with temporal properties, in such a way that a time interval can be asserted to be included in another one, consecutive, partially coincident, etc.

D. Deductive Rules

nRQL rules have a structure very similar to instance queries. The antecedent of the rule is a pattern equivalent to the body of a query. The consequent includes logical conditions that must be satisfied by the ontology, or expressions to create new instances. Deductive rules only include conditions of the first type, and do not contain in the consequent any individual not mentioned in the antecedent.

Deductive rules are used to maintain the consistency of the ontology. For example, the next command prepares a rule that states that the position values of a *TrackedObject* and its associated *Track* must be equal. If the rule is not satisfied, the ontology would be reported as inconsistent.

Rule 1

```
(prepare-rule
 (and
  (?o      scob:TrackedObject)
  (?t      tren:Track)
  (?o ?t   scob:hasAssociatedTrack)
  (?t ?tp  current-position-of-track)
  (?o ?op  current-position-of-object)
  (?tp ?tpt tren:TpositionValue)
  (?op ?opt scob:OpositionValue))
  (
    (instance ?opt
      (some tren:x
        (= racer-internal%has-real-value
          ( (lambda (x) (float (first x)) )
            (datatype-fillers (tren:x ?tpt))))))
    (instance ?opt
      (some tren:y
        (= racer-internal%has-real-value
          ( (lambda (y) (float (first y)) )
            (datatype-fillers (tren:y ?tpt))))))
```

E. Abductive Rules

Abductive rules allow the inclusion of additional individuals in the consequent, which are created as new instances of the ontology. Abductive rules are used in the framework to achieve its two main objectives: scene interpretation and creation of feedback for the GTL.

Scene interpretation. Abductive rules are defined in the framework to interpret what is happening in the scene from the basic tracking data. In that manner, symbolic scene

descriptions are built from visual measures through various inference steps (correspondence, recognition, evaluation) that calculate instances of an upper ontology with rules operating on instances of a lower ontology.

An example of an abductive rule to calculate correspondences between tracks and objects is the following. This rule creates a new `Person` instance when a track bigger than a predefined size (25×40) (and not associated to any object) is detected in the image. The created `Person` instance is associated with the identified track:

Rule 2

```
(prepare-rule
 (and
  (?t      scob:unknown_object
           scob:isAssociatedToObject)
  (?t ?ts current-size-of-track)
  (?ts ?d  tren:TsizeValue)
  (?d     tren:2DDimension)
  (?d     (>= tren:w 25.0))
  (?d     (>= tren:h 40.0)))
 (
  (instance
   (new-ind person-ins ?t) scob:Person)
  (forget-role-assertion
   ?t
   scob:unknown_object
   scob:isAssociatedToObject)
  (related
   (new-ind person-ins ?t)
   ?t
   scob:hasAssociatedTrack)))
```

Abduction rules for scene recognition can involve knowledge at different levels. Another example of abductive rule is the following one, which recognizes the situation when a person is reflected by a mirror. This rule is fired when a person is inside of the area of influence of a mirror and, at the same time, there is another detected person inside of the corresponding mirror. For the sake of clarity, the rule supposes that the value of the `insideOf` property has been previously calculated. As a result, the rule creates a new instance of the `ReflectionSituation` concept with suitable property values.

Rule 3

```
(prepare-rule
 (and
  (?real_person      scob:Person)
  (?real_person ?olp current-position-of-object)
  (?mirror_person    scob:Person)
  (?mirror_person ?o2p current-position-of-object)
  (?mirror           scob:Mirror)
  (?mirror ?ma      current-area-of-object)
  (?mirrored_area    scob:MirroredArea)
  (?mirror ?mirrored_area surv:mirrorsArea)
  (?mirrored_area ?maa current-area-of-object)
  (?olp ?ma scob:insideOf)
  (?o2p ?maa scob:insideOf))
 (
  (instance
   (new-ind ref-ins ?mirror_person)
   actv:ReflectionSituation)
  (related
   (new-ind ref-ins ?mirror_person)
   ?mirror
```

```
actv:mirror)
 (related
  (new-ind ref-ins ?mirror_person)
  ?mirror_person
  actv:objectInMirror)))
```

Feedback. In essence, process refinement is very similar to scene interpretation, since in both cases abductive reasoning is carried out to draw new knowledge—in this case, instances of the `Reco` ontology. From the previous and current interpretations of the scene, abductive rules are triggered to create instances of the `SuggestedAction` concept. Recommendations can be generated at different abstraction levels, i.e. they can involve `Tracks`, `SceneObjects`, `Situations`, `Impacts`, etc. or even combinations of them. In practice, that means that feedback rules have terms of the `TREN`, `SCOB`, `ACTV`, and `IMPC` ontologies in their antecedent; and terms of the `RECO` ontology in their consequent.

The following rule obtains a recommendation that suggests ignoring the track associated to a reflection in a mirror.

Rule 4

```
(prepare-rule
 (and
  (?s          actv:ReflectionSituation)
  (?s ?objectInMirror actv:objectInMirror)
  (?objectInMirror ?t scob:hasAssociatedTrack))
 (
  (instance
   (new-ind ign-instance ?s)
   reco:IgnoreTrack)
  (related
   (new-ind ign-instance ?s)
   ?t
   reco:trackToBeIgnored)))
```

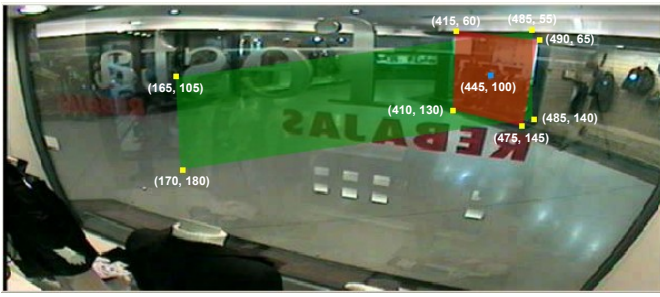
IV. EXAMPLE

The queries and rules of the previous section give an introduction on the implementation of reasoning processes within our framework. In this section, we exemplify how these rules are fired. We have selected two frames of a video sequence of `PETS2002` benchmark⁵. Fig. 3.a shows the initial state of the scene, where a mirror and its associated mirrored area are marked. Fig. 3.b, which we suppose is the next available frame, shows two new tracks detected by the `GTL`.

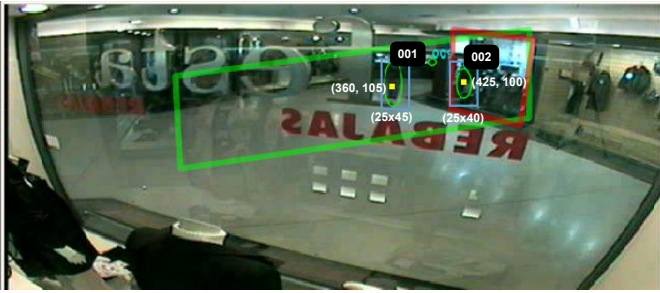
Before the `GTL` is started, the `CL` model is initialized with the previous rules and suitable instances describing the scene of Fig. 3.a. That is, instances of `Mirror` and `MirroredArea` are created, with the position and area properties depicted in this image. Next, the `GTL` begins processing frame 0. Since no moving objects are detected, no track data updates are communicated to the `CL`.

After that, the `GTL` processes frame 1. In this case, two new tracks are detected, track 1 and track 2, with the properties depicted in Fig. 3.b. The `GTL` invokes the `GTL/CL` interface, and notifies the `CL` that two new tracks have been created. The interface creates two `Track` instances, with associated position and area properties, and adds them to the `CL` model.

⁵ <http://www.cvg.cs.rdg.ac.uk/PETS2002/pets2002-db.html>



(a) Frame 0



(b) Frame 1

FIG. 2. BETS2003 example frames

This triggers the reasoning procedures explained in Table 1, which eventually finish with the creation of an IgnoreTrack recommended action for the GTL. Additions and removals to the CL model are represented with ‘+’ and ‘-’, respectively.

Initial state	(mirror1:Mirror) (mirroredArea1:MirroredArea) ((mirror1, mirroredArea1) : mirrorsArea) position & area of mirror1, mirroredArea1
Event	Track 1 and track 2 detected. The GTL invokes the interface
Changes	+ (track1: Track) [position, size, isAssociatedToObject object:unknown] + (track2: Track) [position, size, isAssociatedToObject object:unknown]
Rules fired	Rule 2: track1 matches the rule; new person instance created Rule 2: track2 matches the rule; new person instance created
Changes	+ (person1: Person) - ((track1, unknown_object): isAssociatedToObject) + ((person1, track1): hasAssociatedTrack) + (person2: Person) - ((track2, unknown_object): isAssociatedToObject) + ((person2, track2): hasAssociatedTrack)
Rules fired	Rule 1: person1 and track1 match the rule; position of person1 (which was unknown) is made equal to position of track1 Rule 1: person2 and track2 match the rule; position of person2 (which was unknown) is made equal to position of track2
Changes	+ position of person 1 = (360, 105) + position of person 2 = (425, 100)
Rules fired	Rule 3: person1, person2, mirror, mirroredArea1 match the rule; a reflection situation is detected
Changes	+ (reflection1:ReflectionSituation) + ((reflection1, mirror1): mirror)

	+ ((reflection1, person2): objectInMirror)
Rules fired	Rule 4: reflection1 matches the rule; a ignore reflection track recommendation is created
Changes	+ (ignoreTrack:IgnoreTrack) + ((ignoreTrack, track2): trackToBelgnored)
STOP	

Table 1. Rules triggered and changes in the CL model after frame 1 is processed by the GTL

V. CONCLUSIONS AND FUTURE WORK

In this paper, we have shown that ontology-based rules are a flexible formalism to achieve high-level scene understanding and to improve fusion processes that can be, interestingly, extended to different application domains. We plan to continue this research work in two main directions. First, we will finish the implementation of the visual fusion system, in order to demonstrate the functioning, evaluate and analyze the results, and quantify the benefits of our framework with respect to other approaches. This will require the development of further ontologies and rule bases. Second, we plan to extend our proposals to multi-camera environments and to introduce uncertainty in the representation and the reasoning.

REFERENCES

- [1] C. Nowak, “On ontologies for high-level information fusion,” *6th Int. Conf. on Information Fusion*, Cairns, Australia: 2003, pp. 657-664.
- [2] A.N. Steinberg and G. Rogova, “Situation and context in data fusion and natural language understanding,” *11th Int. Conf. on Information Fusion*, Cologne, Germany: 2008, pp. 1-8.
- [3] E.G. Little and G.L. Rogova, “Designing ontologies for higher level fusion,” *Information Fusion*, vol. 10, Jan. 2009, pp. 70-82.
- [4] M. Kokar, C.J. Matheus, and K. Baclawski, “Ontology-based situation awareness,” *Information Fusion*, vol. 10, Jan. 2009, pp. 83-98.
- [5] B. Neumann and R. Möller, “On Scene Interpretation with Description Logics,” *Image and Vision Computing*, vol. 26, 2008, pp. 82-101.
- [6] J. Gómez-Romero, M.A. Patricio, J. García, and J.M. Molina, “Ontological representation of context knowledge for visual data fusion,” *12th Int. Conf. on Information Fusion*, Seattle, WA, USA: 2009.
- [7] J. Gómez-Romero, M.A. Patricio, J. García, and J.M. Molina, “Context-based reasoning using ontologies to adapt visual tracking in surveillance,” *6th Int. Conf. on Advanced Video and Signal Based Surveillance*, Genoa, Italy: 2009.
- [8] M.A. Patricio, F. Castanedo, A. Berlanga, Ó. Pérez, J. García, and J.M. Molina, “Computational Intelligence in Visual Sensor Networks: Improving Video Processing Systems,” *Computational Intelligence in Multimedia Processing: Recent Advances*, 2008, pp. 351-377.
- [9] A.N. Steinberg and C.L. Bowman, “Rethinking the JDL data fusion levels,” *MSS National Symposium on Sensor and Data Fusion*, Columbia, SC, USA: 2004.
- [10] F. Baader and R. Küsters, “Nonstandard Inferences in Description Logics: The Story So Far,” *Mathematical Problems from Applied Logic I*, Springer New York, 2006, pp. 1-75.
- [11] B. Motik, U. Sattler, and R. Studer, “Query Answering for OWL-DL with rules,” *Journal of Web Semantics*, vol. 3, Jul. 2005, pp. 41-60.
- [12] C. Elsenbroich, O. Kutz, and U. Sattler, “A case for abductive reasoning over ontologies,” *OWLED 2006*, Athens, GA, USA: 2006.
- [13] M. Wessel and R. Möller, “A high performance semantic web query answering engine,” *Int. Workshop on Description Logics*, Edinburgh, Scotland: 2005.
- [14] B. Bennett, “Logics for Topological Reasoning,” *12th European Summer School in Logic, Language and Information*, Birmingham, UK: 2000, pp. 6-18.
- [15] R. Grütter, T. Scharrenbach, and B. Bauer-Messmer, “Improving an RCC-Derived Geospatial Approximation by OWL Axioms,” *7th Int. Semantic Web Conference*, 2008, pp. 293-306.