Diplomarbeit

Präferenzbasierte Szeneninterpretation

Diploma Thesis

Preference-based Scene Interpretation

Seyed Mohammad Reza Rasouli

supervised by:
Prof. Dr. Ralf Möller
Dr. Michael Wessel

January 4, 2011

TUHH
Technische Universität Hamburg-Harburg

Name: Seyedmohammad Reza Rasouli
Mtr.Nr.: 31106
Major: IIW
Birthday: 31.08.1982
Email: reza dot rasouli at tu-harburg dot de
Institute for Software Systems (*http://www.sts.tu-harburg.de*)
Supervised By: Prof. Dr. Ralf Möller,
Dr. Michael Wessel

Eidesstaatliche Erklärung :

Ich versichere hiermit, dass ich meine vorliegende Diplomarbeit mit dem Thema "Präferenzbasierte Szeneninterpretation" selbständig verfasst habe und keine anderen Quellen und Hilfsmittel als die, welche am Ende dieser Arbeit aufgeführt sind, benutzt habe.

Seyedmohamamd Reza Rasouli, Hamburg, den 04.01.2011

Declaration:

I hereby declare that I have written and prepared this report with the topic "Preference-based Scene Interpretation" all by myself and independently and did not use any other resources than those which are listed at the end of this work.

Seyedmohamamd Reza Rasouli, Hamburg, 04.01.2011

Acknowledgements:

I would like to thank Prof. Dr. Ralf Möller for his support and help during my entire work and diploma thesis.

I would like to thank Dr. Michael Wessel for helping me overcome the problems regarding my thesis and specially supporting me with the questions and problems regarding RacerPro.

I want to dedicate this work to my parents and brother who have always shown me their support and love.

Contents

Chapter 1

Introduction

Annotating the media documents plays an important role in order to work with these documents. The growth in the amount of the media documents and expansion of the fields, where they are used, makes the annotation process more important but harder.

The simplest but very inefficient way to annotate a media document is to comment these documents manually. A human user should watch, listen and/or read the entire media document to be able to comment this document. One of the disadvantages of this method is that each human individual has its own understanding of each document. For example, one can annotate an audio document (music document) with the words "Classic Guitar" since he is an expert in playing classic guitar and one could annotate the same document with "Modern Saxophone" if he is interested in this music instrument. A person who believes that a film is in the horror genre has a different way of thinking compared to a person who believes that the film should be categorized under the thriller group.

Another problem of manual annotation is the time needed for each media document to be processed. Each video clip or audio document should be studied by a person individually, which is a time consuming process. It is also not efficient, for example, to watch the entire video clip to annotate this movie only by a few keywords later. In other words, the amount of the data that is observed in a media document is usually very high but also redundant.

As discussed above, manual annotation of the media documents is not efficient at all. If the annotation process is done by machines instead, the speed of the process increases significantly. This is done due to several reasons such as filtering the redundant and/or non-relevant data, using image/audio/text processing tools and programs, applying standard modulation of the interpretation steps and, etc. . Since on one hand the image/audio/text processing tools are not fully accurate and on the other hand, the interpretation and annotation of the media documents are dependent on the basic and primitive data that are presented in the document and should be identified by these tools, this solution might not be the best solution at the first glance. Therefore, relying entirely on the machines is not effective at all and the results should be controlled from time to time by a human user. However, if the machines are able to learn and model reliable patterns, the need of manual control will decrease.

One of the projects that is conducted in the field of media annotation is the **CASAM**[1] [6] [7] [5] project. **CASAM** stands for **Computer-Aided Semantic Annotation of Multimedia**. The goal of this project is to implement a machine for annotating the multimedia documents. CASAM has an abduction engine, which is based on an engine that was used for another project, which is called **Boemie** [2]. In this work, for the abduction process, the **RacerPro** software is used.

In the next Chapter, the table cloth scenario, which this work is based on it, is discussed. In this Chapter, the **RacerPro**[3] software is introduced and the TBox regarding the scenario is designed. In this Chapter, the tracking data of a video clip provided by **The Cognitive Systems Laboratory of the University of Hamburg**[4] and studied in [23], will be illustrated and used for the presentation knowledge presentation of the scene. In the third Chapter, the bottom-up and top-down rules that are needed to interpret the scene, are studied and for each , two versions of rules are introduced. In addition, three different strategies are discussed that specify how and when the rules should be applied. In this Chapter, a scoring function that is implemented in this work to choose the best results of explanations is described. In Chapter four, it will be looked into the framework and its GUI that were implemented during this diploma thesis. In Chapter five, the results of interpretation using the framework and the reasoner software are illustrated. These results are studied in Chapter six in depth for twelve different modes resulted by the combination of four groups of rules and three different strategies. Finally, in Chapter seven, the conclusion of this work and some ideas as cases of study are described.

[1]http://www.casam-project.eu/
[2]http://www.boemie.org/
[3]http://www.racer-systems.com/
[4]http://kogs-www.informatik.uni-hamburg.de/

Chapter 2

Table Cloth Scenario

In this Chapter, the table cloth scenario upon which this work focuses, will be illustrated. As discussed in the previous Chapter, the goal of this work is to use the low-level information[1] contained in the frames of a video document to interpret this video. The result of the interpretation is high-level information about the scene shown in each frame and the entire video.

2.1 The Scenario

The primitive objects[2] (low-level data) in the scenario are the dishes (plate, saucer and cup) and the cutlery objects (spoon, knife, fork and teaspoon) that will be placed on a table during the video. The first frame of the clip contains an empty table. As the video clip proceeds the end, more objects will be put on the table by hand. At the same time, a camera located above the table tries to identify these objects. The objects may be removed or relocated during the video clip. If the primitive objects presented in a frame are immobile, this frame is called a **Keyframe**[3]. If an object is discovered on the table, but could not be categorized under any type of the primitive objects mentioned above, this object will be labeled as ***Unknown***. The camera captures and analyzes the scene and determines the position of each object and its bounding box. This feature helps us specify the size of a primitive object and its spatial relationships regarding other objects. The spatial relationships are such as ***near-right-of***, ***near-left-of***, ***near-above-of***, ***near-under-of*** and ***on-same-position-of***.

The aggregates[4] in this work are ***Meal_Cover***, ***Dessert_Cover***(lower aggregates) and ***Combined_Cover*** (higher aggregate)[5].

A plate and the cutlery objects related to it (knife, fork and/or), form a meal

[1]The presentation of the low-level information will be discussed in Section 2.2.

[2]The tracking data and content of the video clip studied in this work are provided by **The Cognitive Systems Laboratory of the University of Hamburg** and studied in [23].

[3]For brevity, from this point on whenever the word frame is mentioned, a keyframe is meant.

[4]An aggregate is a concept that is formatted by the primitive objects[17].

[5]The description of the aggregates and their relationships regarding the primitive objects are described in Section 2.3.

cover. A cup and the objects related to it (saucer and/or teaspoon) form a dessert cover. A single plate or cup can also form a cover. It is also possible that a plate or cup form an aggregate by itself. The cutlery objects and the saucers that are not related to a plate or a cup will not be categorized under any cover in this work, since at least one kind of dish (a plate or cup) should be present on the table.

A meal cover and a dessert cover could form a combined cover when they are integrated together[6]. A schema of the concepts discussed so far could be seen in figure 2.1.

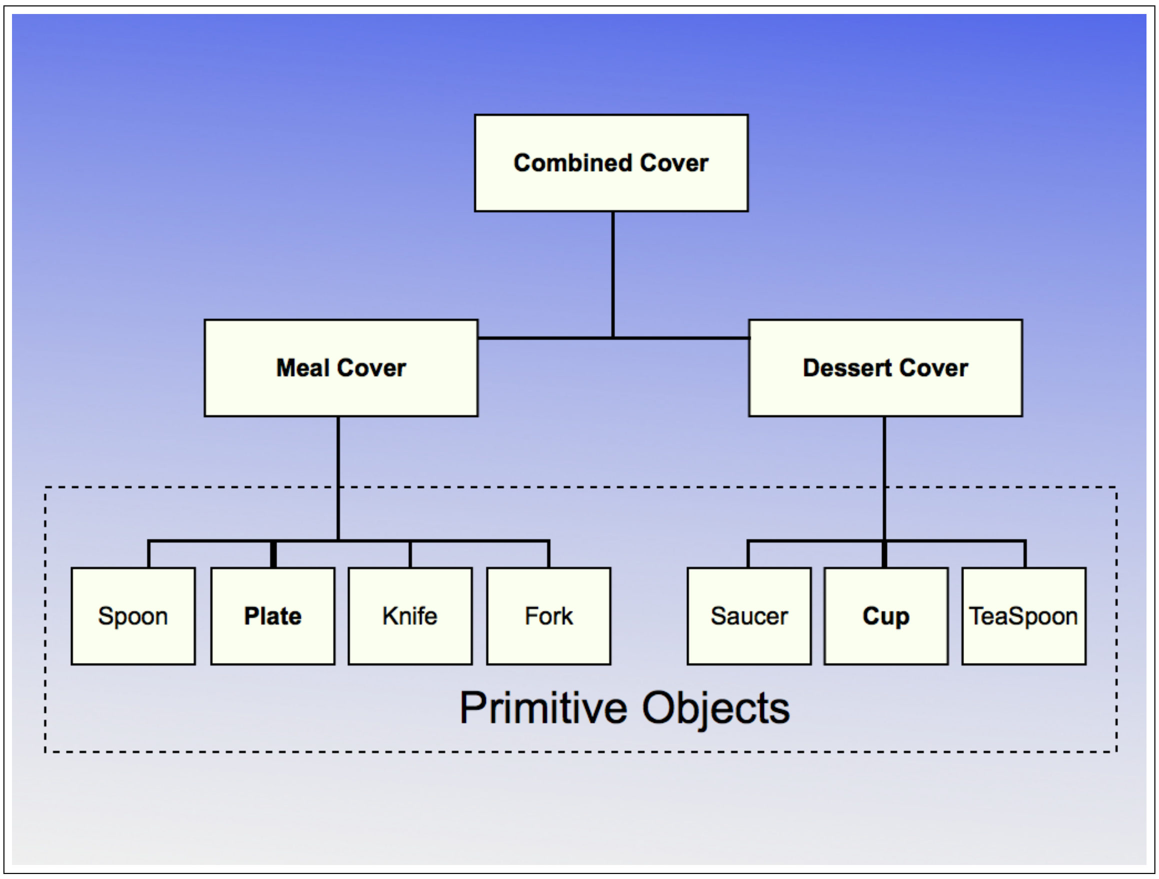


Figure 2.1: Basic Schema of the Primitive Objects and Aggregates

In general, all covers could have more specific meanings such as: ***Steak_Cover***, ***Tea_Cover***, ***Spaghetti_Cover***, ***Full_Dinner_Cover***, , ***Dinner_With_Bitter_Coffee_Cover***, ***Soup_Cover***, ***Bitter_Coffee_Cover***, ***Small_Breakfast _ Cover*** and ***Big_Breakfast_Cover***.

One of the main goals of this work is to interpret the type of the table and the number of the people who will be sitting at this table. Therefore, an instance of the ***Table*** concept could have more specific definitions regarding its type and number of seats.

[6]To be more precise, if the cup of a dessert cover is on the right side of the plate of a meal cover and there is no other cup between these two objects, the dessert cover and the meal cover that contain these objects form a combined cover.

The types in this work are ***Small_ Breakfast_Table***, ***Soup_Table***, ***Italian_Table English_ Tea_Table***, ***Big_Breakfast_Table***, ***Full_Dinner_With_Tea_Table*** and ***Full_ Dinner_With_Bitter_Coffee_Table.***

The types regarding the number of the people who will be sitting at a table are: ***Single_Table***, ***Double_Table***, ***Family_Table***, ***Business_Table*** and ***Big_Family_Table***[7].

2.2 Tracking Data

The tracking data is the information delivered by the camera for each frame. It contains the specifications of the objects that have been identified by the camera [8] . These specifications include an ID referring to each object, the position of the pivot of each object and their corresponding bounding boxes. It also denotes if an object is moving in the scene or not.

In general, the tracking data of an object look as follows:

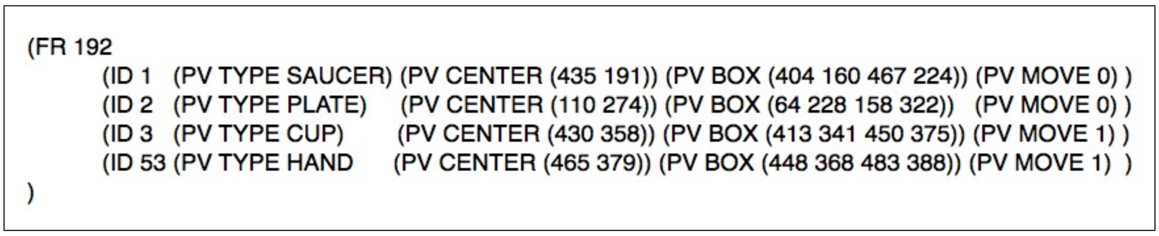
```
(ID #Num
            (PV TYPE     (Object's Type)
            (PV CENTER (X Y))
            (PV BOX       (X1, Y1, X2, Y2))
            (PV MOVE     (0 or 1))
)
```

The ID of each object is a unique number. If the object is moving in the scene, its move property will be set to 1 in the tracking data, and if it is not moving, it will be set to 0. The tracking data is saved in a text file.

An entity in the scene is either a primitive object, a hand or an object that is not categorized yet, which is marked as an ***Unknown*** object.

The objects of type ***Hand*** will be ignored in the entire work, since they have no effect on the outcome of this work.

For example, the tracking data regarding the frame 192 in the tracking data file looks as shown in figure 2.2:



```
(FR 192
      (ID 1   (PV TYPE SAUCER) (PV CENTER (435 191)) (PV BOX (404 160 467 224)) (PV MOVE 0) )
      (ID 2   (PV TYPE PLATE)    (PV CENTER (110 274)) (PV BOX (64 228 158 322))   (PV MOVE 0) )
      (ID 3   (PV TYPE CUP)       (PV CENTER (430 358)) (PV BOX (413 341 450 375)) (PV MOVE 1) )
      (ID 53 (PV TYPE HAND      (PV CENTER (465 379)) (PV BOX (448 368 483 388)) (PV MOVE 1)  )
)
```

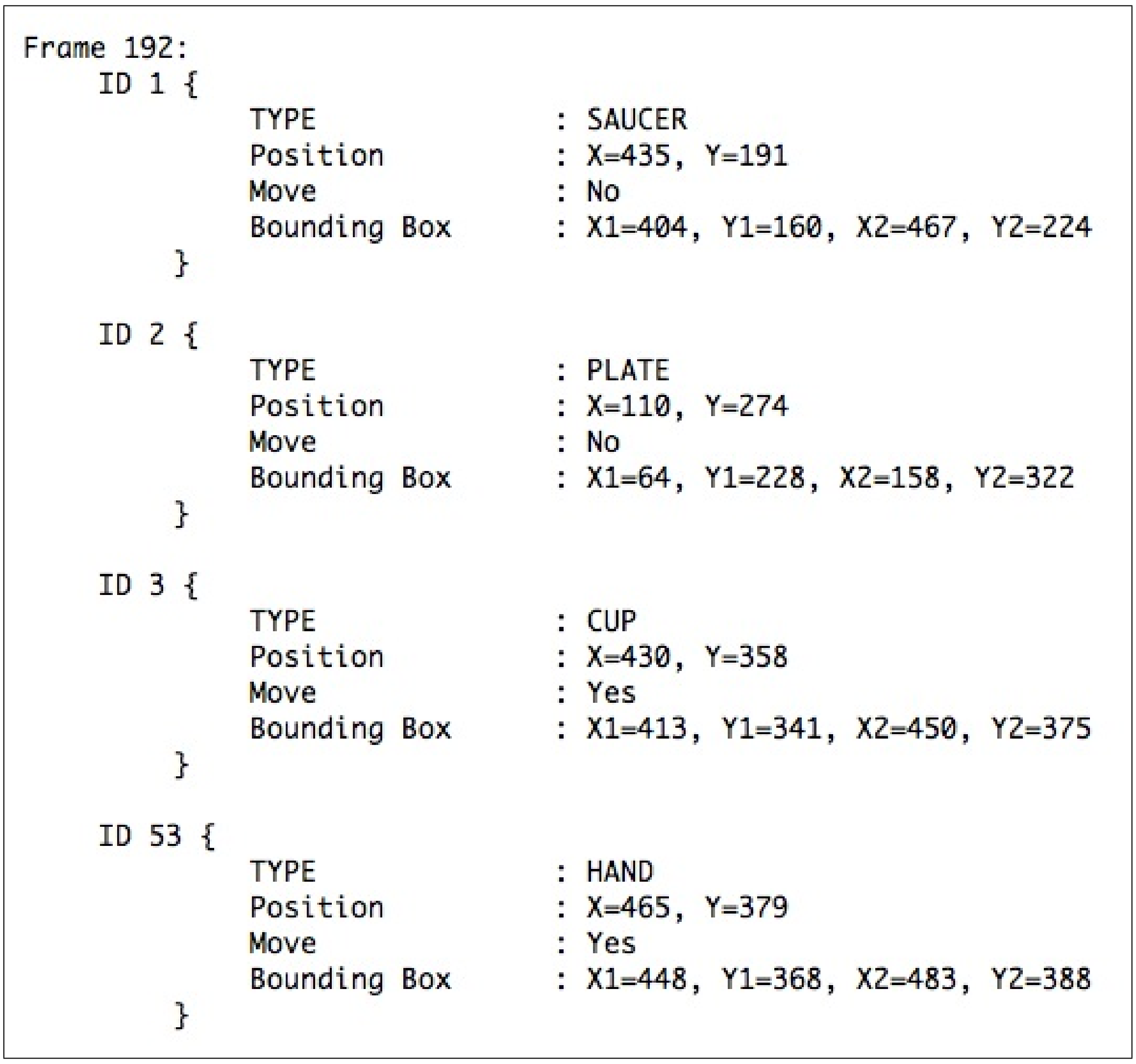
Figure 2.2: Tracking Data of Frame 192

The lines in figure 2.2 are modified and shown in a user-friendly mode in figure 2.3. The tracking data of frame 192 shows that in this frame four objects have been

[7]More details are discussed in Section 2.3.

[8]Not necessarily categorized.

identified. The object with ID 1 is of type ***SAUCER***, which is located in *(435, 292)*[9] and is **not moving**. The object with ID 2 is of type ***PLATE***, which is located in *(110, 274)* and is **not moving**. The Object with the ID 3 is of type ***CUP***, which is located in *(430, 358)*, but is **moving**. The object with ID 53 is a hand and will be ignored as previously mentioned.



```
Frame 192:
    ID 1 {
            TYPE                : SAUCER
            Position            : X=435, Y=191
            Move                : No
            Bounding Box        : X1=404, Y1=160, X2=467, Y2=224
        }

    ID 2 {
            TYPE                : PLATE
            Position            : X=110, Y=274
            Move                : No
            Bounding Box        : X1=64, Y1=228, X2=158, Y2=322
        }

    ID 3 {
            TYPE                : CUP
            Position            : X=430, Y=358
            Move                : Yes
            Bounding Box        : X1=413, Y1=341, X2=450, Y2=375
        }

    ID 53 {
            TYPE                : HAND
            Position            : X=465, Y=379
            Move                : Yes
            Bounding Box        : X1=448, Y1=368, X2=483, Y2=388
        }
```

Figure 2.3: Modified Tracking Data of Frame 192

The data in figure 2.3 is illustrated by the implemented framework in figure 2.4. Each object with its ID, pivot and its corresponding bounding box is shown in this figure.

[9]In X-Y coordinate system.

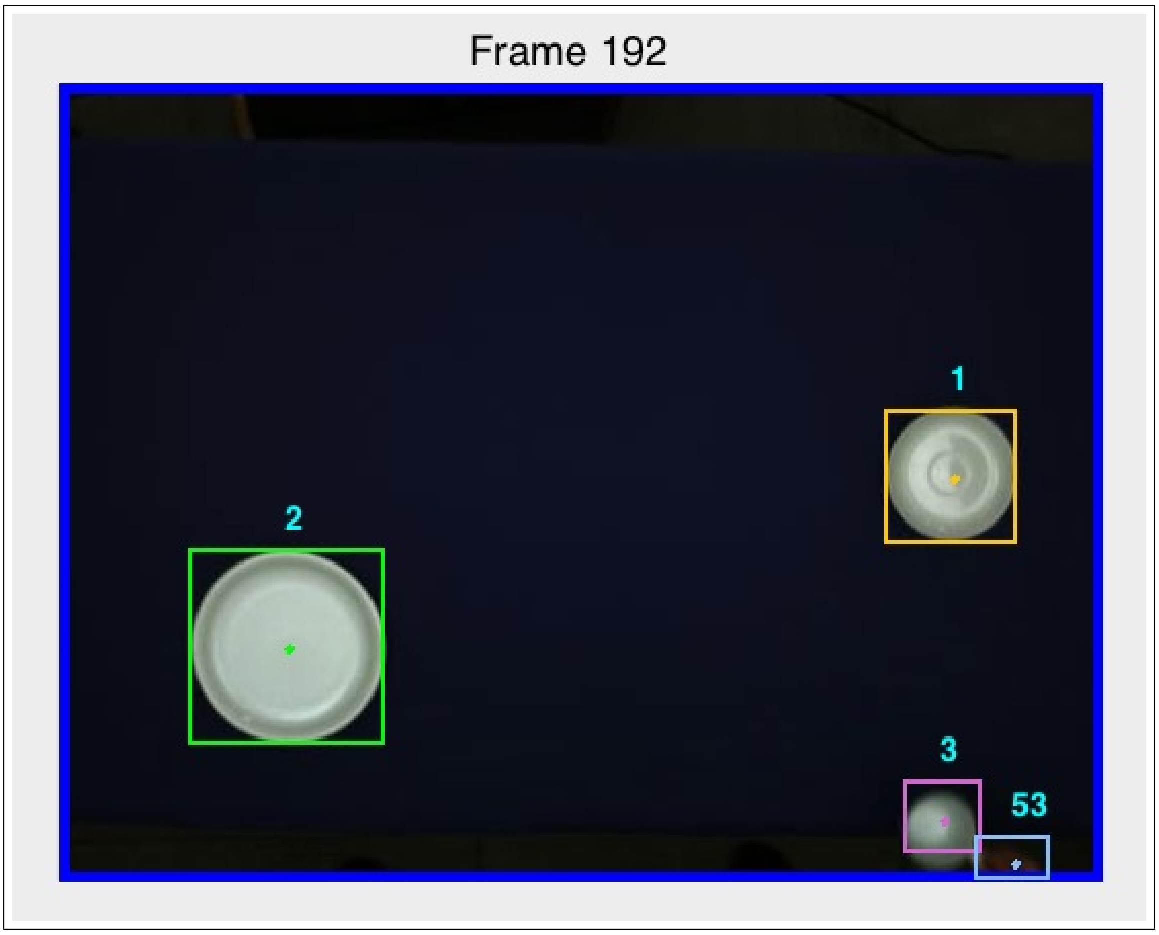


Figure 2.4: Frame 192

2.3 Reasoner

Since in this work, description logics are used to work with the knowledge base regarding the scenes and to define the corresponding concepts for the primitive objects and aggregates, their specification and the relations between the concepts and their instances, a reasoner software is needed.

Hence, the RacerPro Reasoner, which works with ***SHIQ*** description logics is used. ***SHIQ*** is ***ALC*** [1] [3] with inverse, transitive and hierarchical roles and number restrictions.

RacerPro can handle TBoxes and ABoxes; it also answers queries using the nRQL[10] language.

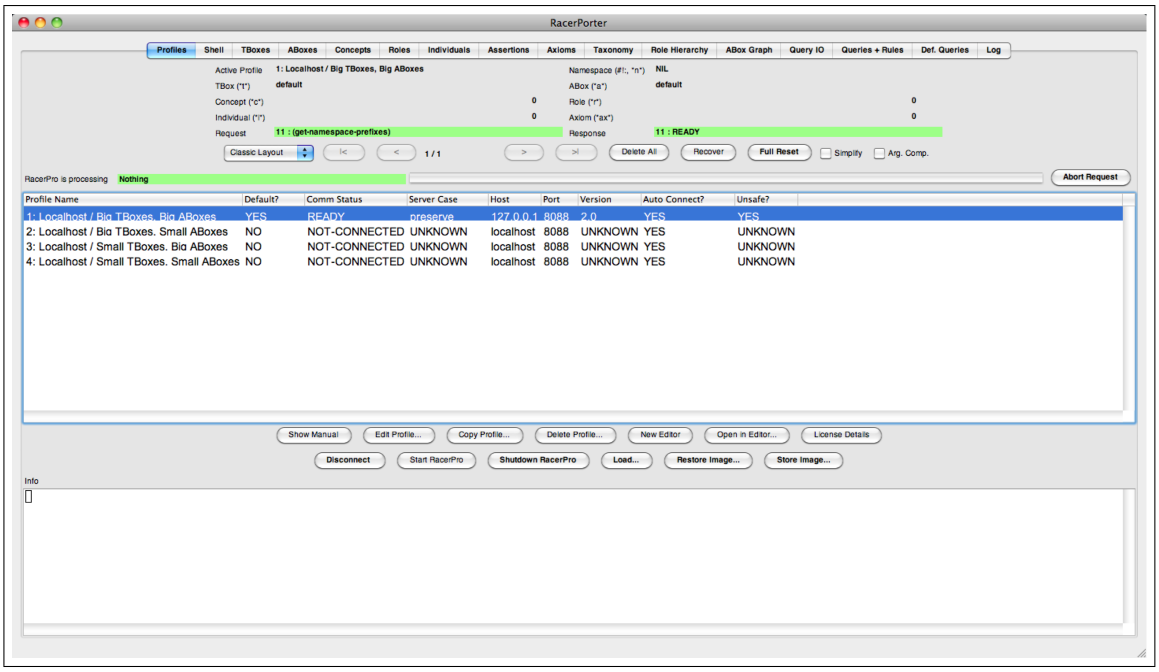
2.3.1 RacerPro

Racer stands for **Renamed ABox and Concept Expression Reasoner Professional**. RacerPro is a software used in the area of the description logics. RacerPro

[10] New Racer Query Language

is being developed by **Racer Systems GmbH & Co. KG**[11] using programming languages, which are available for multiple computer platforms.

While key algorithms are written in Lisp, other parts of the system such as tools are implemented in Java. RacerPro is offered on Windows, Linux and Mac OS X. In this thesis the Mac OS X(RacerPro 2.0 preview for Mac OS X, 32bit, x86 CPU) which was released on 01.11.2009, is used. RacerPro plays the role of a server, and it is the core of the software. RacerPorter (Figure 2.5) is the GUI of RacerPro[12]. In this diploma thesis, the **abduction reasoning feature** of **RacerPro** plays a major role, specially for interpreting and explaining the scene.



Active Profile	1: Localhost / Big TBoxes, Big ABoxes	Namespace (#!:, "n")	NIL
TBox ("t")	default	ABox ("a")	default
Concept ("c")	0	Role ("r")	0
Individual ("i")	0	Axiom ("ax")	0
Request	11 : (get-namespace-prefixes)	Response	11 : READY

RacerPro is processing Nothing

Profile Name	Default?	Comm Status	Server Case	Host	Port	Version	Auto Connect?	Unsafe?
1: Localhost / Big TBoxes, Big ABoxes	YES	READY	preserve	127.0.0.1	8088	2.0	YES	YES
2: Localhost / Big TBoxes, Small ABoxes	NO	NOT-CONNECTED	UNKNOWN	localhost	8088	UNKNOWN	YES	UNKNOWN
3: Localhost / Small TBoxes, Big ABoxes	NO	NOT-CONNECTED	UNKNOWN	localhost	8088	UNKNOWN	YES	UNKNOWN
4: Localhost / Small TBoxes, Small ABoxes	NO	NOT-CONNECTED	UNKNOWN	localhost	8088	UNKNOWN	YES	UNKNOWN

Figure 2.5: RacerPorter: GUI of RacerPro

2.4 TBox of the Scenario

For the scene interpretation, a TBox needed to define the concepts and their relations to each other for the objects and the aggregates. This TBox is saved in the

[11]**Racer Systems GmbH & Co. KG** is a company that was founded in September 2004 and is located in Hamburg. This company develops a reasoner software called RacerPro and consults in the area of semantic web for industrial projects based on the W3C standards RDF/OWL.
For more information, please visit *www.racer-systems.com*

[12]For more information, regarding RacerPro, please read the following documents:
"RacerPro Users Guide":
(*http://www.racer-systems.com/products/racerpro/users-guide-1-9-2-beta.pdf*)

"RacerPro Reference Manual":
(*http://www.racer-systems.com/products/racerpro/reference-manual-1-9-2-beta.pdf*)

file ***"TBox.racer"*** which is located in the *docs* folder of the program. This makes it possible to run the scene interpretation program[13] regardless of the media document that is to be interpreted. Therefore, the TBox is always the same for all media documents. Before the program starts, the TBox should be loaded in RacerPro. The TBox starts with the following lines:

```
1:   (full-reset)
2:   (init-tbox tracking)
3:   (in-knowledge-base tracking tracking-family)
4:   (signature:  ( :atomic-concepts {concept list} )
5:                 :roles ( {role list} )
```

The first three lines reset the RacerPro and create a TBox with the name *tracking.* In line (4) and in line (5), the concepts and the roles, which are not shown here for readability, are defined respectively.

```
6:   (implies PLATE  Meal_Objects)
7:   (implies KNIFE  Meal_Objects)
8:   (implies FORK   Meal_Objects)
9:   (implies Spoon  Meal_Objects)
```

$$\begin{array}{ll} PLATE & \sqsubseteq Meal\_Objects \\ KNIFE & \sqsubseteq Meal\_Objects \\ FORK & \sqsubseteq Meal\_Objects \\ Spoon & \sqsubseteq Meal\_Objects \end{array}$$

```
10:   (implies CUP       Dessert_Objects)
11:   (implies SAUCER    Dessert_Objects)
12:   (implies TeaSpoon  Dessert_Objects)
```

$$\begin{array}{ll} CUP & \sqsubseteq Dessert\_Objects \\ SAUCER & \sqsubseteq Dessert\_Objects \\ TeaSpoon & \sqsubseteq Dessert\_Objects \end{array}$$

```
13:  (implies Meal_Objects     Objects)
14:  (implies Dessert_Objects  Objects)
```

$$\begin{array}{ll} Meal\_Objects & \sqsubseteq Objects \\ Dessert\_Objects & \sqsubseteq Objects \end{array}$$

Lines (6-9) imply that the objects ***PLATE***, ***KNIFE***, ***FORK*** and ***Spoon*** are subconcepts of the ***Meal_Objects*** concept, and lines (10-12) imply that the objects ***CUP***, ***SAUCER*** and ***TeaSpoon*** are subconcepts of the concept ***Dessert_Objects***.

Finally, the lines (13) and (14) imply that all the objects, which are either a ***Meal_Objects*** or ***Dessert_Objects***, are subconcepts of the higher concept ***Objects***.

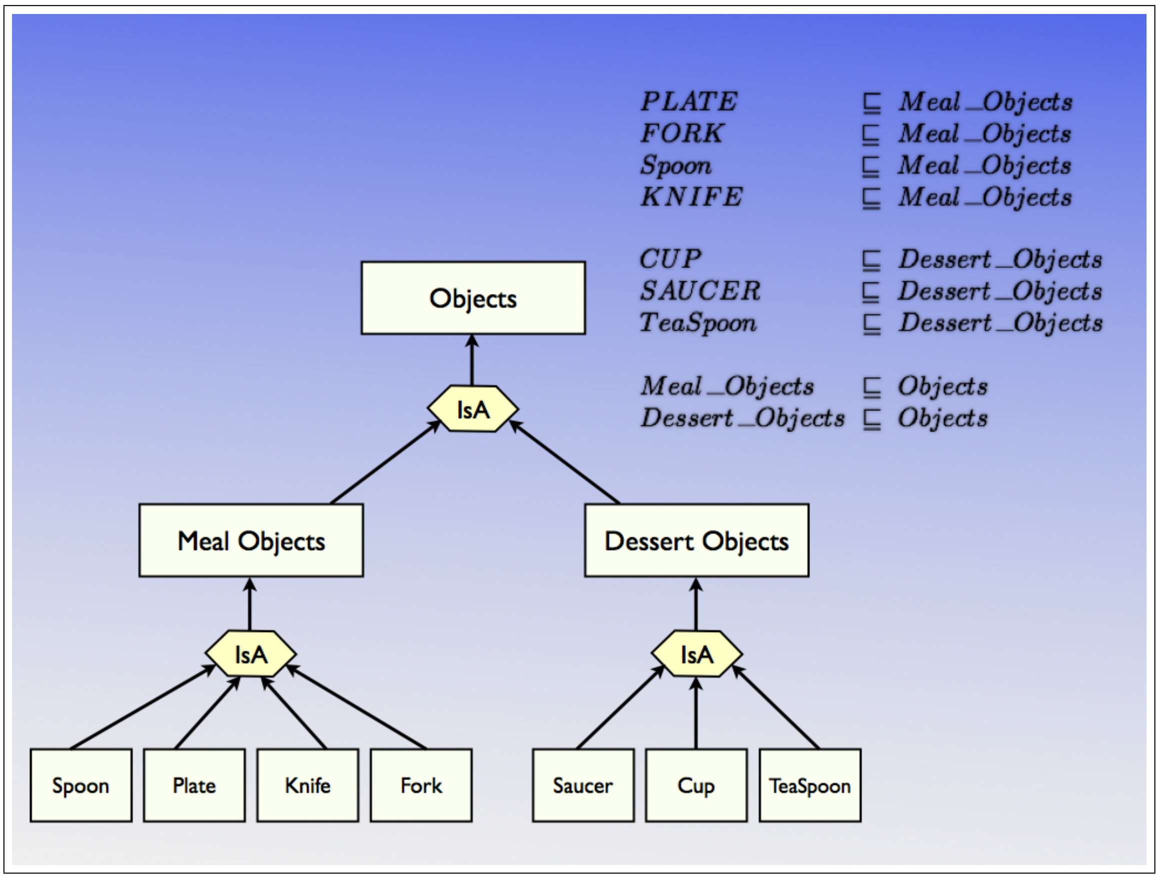
```
15:   (implies Meal_Cover      cover)
16:   (implies Dessert_Cover   cover)
17:   (implies Combined_Cover  cover)
```

$$\begin{array}{ll} Meal\_Cover & \sqsubseteq cover \\ Dessert\_Cover & \sqsubseteq cover \\ Combined\_Cover & \sqsubseteq cover \end{array}$$

After categorizing the primitive objects, the covers are categorized all under the super concept ***cover*** as stated in the lines (15-17).

One of the most important parts of defining the concepts is stating the disjointness of the concepts. Otherwise, an instance of the concept ***PLATE*** could be at the same time an instance of the concept ***SAUCER***. Therefore, the following lines are

[13]For more information regarding the scene interpretation program, please go to Chapter 3, Section 3.



$PLATE \sqsubseteq Meal\_Objects$
$FORK \sqsubseteq Meal\_Objects$
$Spoon \sqsubseteq Meal\_Objects$
$KNIFE \sqsubseteq Meal\_Objects$
$CUP \sqsubseteq Dessert\_Objects$
$SAUCER \sqsubseteq Dessert\_Objects$
$TeaSpoon \sqsubseteq Dessert\_Objects$
$Meal\_Objects \sqsubseteq Objects$
$Dessert\_Objects \sqsubseteq Objects$
Objects
IsA
Meal Objects
Dessert Objects
IsA
IsA
Spoon
Plate
Knife
Fork
Saucer
Cup
TeaSpoon

Figure 2.6: Hierarchy of the Primitive Objects

added to the TBox:

```
16:   (disjoint PLATE FORK Spoon KNIFE)
17:   (disjoint TeaSpoon CUP SAUCER )
18:   (disjoint Meal_Objects Dessert_Objects cover)
19:   (disjoint Meal_Cover Dessert_Cover Combined_Cover Table)
```

Next, the roles should be defined, so that later the relations between the concepts and the number restrictions could be given. First, the roles are all defined as primitive roles:

```
20:   (define-primitive-role near-right-of)
21:   (define-primitive-role near-left-of  )
22:   (define-primitive-role on-same-position-of      )
23:   (define-primitive-role has-part      )
24:   (define-primitive-role belongs       )
25:   (define-primitive-role under        )
26:   (define-primitive-role owns         )
27:   (define-primitive-role is_under     )
```

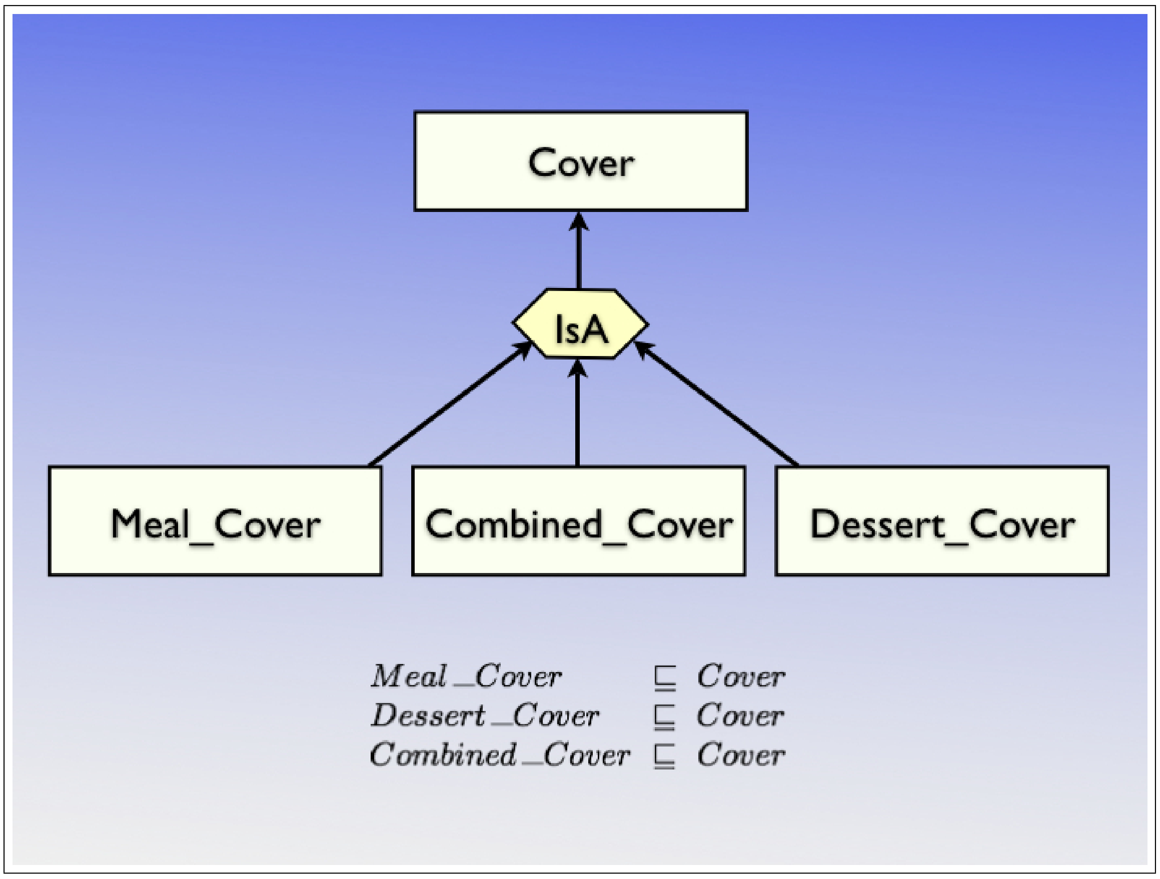

Cover
IsA
Meal_Cover
Combined_Cover
Dessert_Cover
$Meal\_Cover \sqsubseteq Cover$
$Dessert\_Cover \sqsubseteq Cover$
$Combined\_Cover \sqsubseteq Cover$

Figure 2.7: Hierarchy of the Covers

```
28:   (define-primitive-role contains     )
```

The role ***near-right-of*** means that for an assertion like:

*KNIFE(**K**), PLATE(**P**), near-right-of(**K**,**P**)*,

where ***P*** is an instance of ***PLATE***, ***K*** is an instance of ***KNIFE*** in the ABox, ***K*** is on the right side of ***P*** and also ***K*** is the nearest instance of its type (here ***KNIFE***) on the right side of ***P***. For the role ***near-left-of***, the same definitions are valid; however, here the first object in the relation is on the left side of the second object in the relation.

The role ***on-same-position-of*** indicates that two objects are located in the same X-Y position; therefore, one of them is on the top of the other one.

As shown below:

```
29:   (inverse has-part belongs  )
30:   (inverse is_under contains)
31:   (inverse under    owns    )
```

the roles defined in lines 23 and 24 are inverse of each other. The same thing is also valid for the roles defined in the lines 25-26 and 27-28. The role ***has-part***, which is the inverse of the role ***belongs*** describes the relation between an instance of the concept ***Meal_Cover*** or ***Dessert_Cover*** and their corresponding objects. In other words, for an assertion like:

*Meal_Cover(**M**), PLATE(**P**), has-part(**M**, **P**),*

where ***P*** is an instance of ***PLATE***, ***M*** is an instance of ***Meal_Cover*** in the ABox, the assertion *has-part(**M**, **P**)* indicates that ***M*** has the individual ***P*** at one its parts. This means that the relation *belongs(**P**, **M**)* is also true since the role ***belongs*** is the inverse of the role ***has-part***.

The role ***contains*** describes the relation between an instance of the concept ***Combined_Cover*** (higher aggregate) and the instances of the concepts ***Meal_Cover*** and ***Dessert_Cover***(lower aggregates)[14]. The role ***owns*** describes the relation between an instance of the concept ***Table*** and the instances of the concept ***cover***[15].

```
32:   (range   on-same-position-of       Objects)
33:   (domain on-same-position-of       Objects)
34:   (range   near-right-of (or Meal_Cover Objects))
35:   (domain near-right-of (or Dessert_Cover Objects))
36:   (range   near-left-of   (or Dessert_Cover Objects))
37:   (domain near-left-of   (or Meal_Cover Objects))

38:   (range   owns     cover )
39:   (domain owns     Table )
40:   (range   has-part Objects)
41:   (domain has-part (or Dessert_Cover Meal_Cover) )
42:   (range   contains (or Meal_Cover Dessert_Cover) )
43:   (domain contains Combined_Cover )
```

In lines (32-43), the domain and the range of each role have been defined. Lines (32-37) include the domain and the range for the spatial relations between the objects. All instances of the objects could be on either side of the relations. In addition, the concept ***Dessert_Cover*** is added to the domain of the ***near-right-of*** role and the concept ***Meal_Cover*** is added to the range of this role because to be able to create an instance of the concept ***Combined_Cover***, an instance of the ***Dessert_Cover***, which is on the right side of an instance of the concept ***Meal_Cover*** is needed. Accordingly, the opposite is true for the role ***near-left-of***.

In lines (38-43), the range and the domain of the roles between the higher concepts and aggregates have been defined. It should be noted that if a concept ***A*** is in the domain and concept ***B*** is in the range of role ***R***, and the role ***I*** is the inverse role of ***R***, the concept ***A*** is in the range and concept ***B*** is in the domain of the role ***I***. In other words:

[14]The opposite is valid for the role ***is_under***.
[15]The opposite is valid for the role ***under***.

$$\textbf{\textit{d(R)}} = \textbf{\textit{r(I)}} \text{ and } \textbf{\textit{r(R)}} = \textbf{\textit{d(I)}}$$

As a result, by defining the domain and the range of the roles ***has-part***, ***contains*** and ***owns***, there is no need to define the domain and the range of their corresponding inverse roles, which are the roles ***belongs***, ***is_under*** and ***under***.

In practice, there were some difficulties working with the roles in RacerPro although the domain and the range of the roles were defined in the TBox. This problem is solved by adding the following lines into the TBox. These lines are just restrictions for the concepts, indicating that they cannot be in the domain of the specific roles.

```
44:   (implies Objects (not (some contains *top* )))
```
$$Objects \sqsubseteq \nexists\ contains.\top$$

```
45:   (implies Objects (not (some has-part *top* )))
```
$$Objects \sqsubseteq \nexists\ has{-}part.\top$$

```
46:   (implies Meal_Cover (not (some contains *top* )))
```
$$Meal\ Cover \sqsubseteq \nexists\ contains.\top$$

```
47:   (implies Dessert_Cover (not (some contains *top* )))
```
$$Dessert\ Cover \sqsubseteq \nexists contains.\top$$

```
48:   (implies Table (not (some contains *top* )))
```
$$Table \sqsubseteq \nexists\ contains.\top$$

After the concepts and roles are defined, the number restrictions are added, which enables the concepts to be described more precisely and makes them consistent with the environment of the scene interpretation scenario that is the objective of this work.

```
49:   (implies Meal_Cover (exactly 1 has-part PLATE))
```
$$Meal\_Cover \sqsubseteq (\leq 1\ has{-}part.PLATE) \sqcap (\geq 1\ has{-}part.PLATE)$$

```
50:   (implies Meal_Cover (at-most 1 has-part KNIFE))
```
$$Meal\_Cover \sqsubseteq (\leq 1\ has{-}part.KNIFE)$$

```
51:   (implies Meal_Cover (at-most 1 has-part FORK))
```
$$Meal\_Cover \sqsubseteq (\leq 1\ has{-}part.FORK)$$

```
52:   (implies Meal_Cover (at-most 1 has-part Spoon))
```
$$Meal\_Cover \sqsubseteq (\leq 1\ has{-}part.Spoon)$$

```
53:   (implies Meal_Cover (all has-part Meal_Objects))
```
$$Meal\_Cover \sqsubseteq \forall\ has{-}part.Meal\_Objects)$$

```
54:   (implies Dessert_Cover (exactly 1 has-part CUP))
```
$$Dessert\_Cover \sqsubseteq (\leq 1\ has{-}part.CUP)$$

```
55:   (implies Dessert_Cover (at-most 1 has-part SAUCER))
```
$$Dessert\_Cover \sqsubseteq (\leq 1\ has{-}part.SAUCER)$$

```
56:   (implies Dessert_Cover (at-most 1 has-part TeaSpoon))
```
$$Dessert\_Cover \sqsubseteq (\leq 1\ has{-}part.TeaSpoon)$$

```
57:   (implies Dessert_Cover (all has-part Dessert_Objects))
```
$$Dessert\_Cover \sqsubseteq \forall\ has{-}part.Dessert\_Objects)$$

Line (49) indicates that for an instance of ***Meal_Cover***, there must be exactly (at least and at most) one instance of the concept ***PLATE*** as a filler of the role ***has-part***. Lines (50-52) indicate that an instance of ***Meal_Cover*** can have at most one instance of the concepts ***KNIFE***, ***FORK*** and/or ***Spoon*** as fillers of the role ***has-part***. Line (53) indicates that an instance of ***Meal_Cover*** can only have the instances of the concept ***Meal_Objects*** as fillers of the role ***has-part***. The predecessors of the role ***has-part*** in lines (49-53) should all be an instance of the concept ***Meal_Cover***.

For the concept ***Dessert_Meal***, line (54) demonstrates that for an instance of this concept, there must be exactly one instance of the concept ***CUP*** as a filler of the role ***has-part***. The next two lines illustrate that ***Dessert_Cover*** can have at most one instance of the concepts ***SAUCER*** and/or ***TeaSpoon*** as fillers of the role ***has-part***. Finally, line (57) indicates that an instance of ***Dessert_Cover*** can only have the instances of ***Dessert_Objects*** as fillers for the role ***has-part***. The predecessors of the role ***has-part*** in lines (54-57) should all be an instance of the concept ***Dessert_Cover***.

```
58:   (implies Combined_Cover (exactly 1 contains Meal_Cover ))
```
$$\begin{aligned} Combined\_Cover \sqsubseteq\ & (\leq 1\ contains.Meal\_Cover) \sqcap \\ & (\geq 1 contains.Meal\_Cover) \end{aligned}$$

```
59:   (implies Combined_Cover (exactly 1 contains Dessert_Cover))
```
$$\begin{aligned} Combined\_Cover \sqsubseteq\ & (\leq 1\ contains.Dessert\_Cover) \sqcap \\ & (\geq 1 contains.Dessert\_Cover) \end{aligned}$$

```
60:   (implies Combined_Cover (all contains cover))
```
$$Combined\_Cover \sqsubseteq \forall\ contains.cover)$$

```
61:   (implies Meal_Cover (at-most 1 is_under Combined_Cover))
```
$$Meal\_Cover \sqsubseteq (\leq 1\ isunder.CombinedCover))$$

```
62:   (implies Dessert_Cover (at-most 1 is_under Combined_Cover))
```
$$Dessert\_Cover \sqsubseteq (\leq 1\ isunder.CombinedCover))$$

The relations between different types of the covers could be described. From lines (58-60) demonstrate that an instance of ***Combined_Cover*** must have exactly one instance of ***Meal_Cover*** and ***Dessert_Cover*** as fillers for the role ***contains***.

The next two lines demonstrate that an instance of ***Meal_Cover*** and an instance of ***Dessert_Cover*** could belong at most to one ***Combined_Cover*** and that there could be instances of ***Meal_Cover*** and ***Dessert_Cover*** that do not belong to any instances of ***Combined_Cover***.

In other words, in order to build an instance of ***Combined_Cover***, an instance of ***Meal_Cover*** and an instance of ***Dessert_Cover***, which do not belong to any other instances of ***Combined_Cover***, are needed.

```
63:   (implies Meal_Objects (at-most 1 belongs Meal_Cover))
```

$$Meal\_Objects \sqsubseteq (\leq 1\ belongs.Meal\_Cover)$$

```
64:   (implies Dessert_Objects (at-most 1 belongs Dessert_Cover))
```

$$Dessert\_Objects \sqsubseteq (\leq 1\ belongs.Dessert\_Cover)$$

```
65:   (implies PLATE (exactly 1 belongs Meal_Cover))
```

$$PLATE \sqsubseteq (\leq 1\ belongs.Meal\_Cover) \sqcap (\geq 1\ belongs.Meal\_Cover)$$

```
66:   (implies KNIFE (at-most 1 belongs Meal_Cover))
```

$$KNIFE \sqsubseteq (\leq 1\ belongs.Meal\_Cover)$$

```
67:   (implies FORK (at-most 1 belongs Meal_Cover))
```

$$FORK \sqsubseteq (\leq 1\ belongs.Meal\_Cover)$$

```
68:   (implies Spoon (at-most 1 belongs Meal_Cover))
```

$$Spoon \sqsubseteq (\leq 1\ belongs.Meal\_Cover)$$

```
69:   (implies CUP (exactly 1 belongs Dessert_Cover))
```

$$CUP \sqsubseteq (\leq 1\ belongs.Dessert\_Cover) \sqcap (\geq 1\ belongs.Dessert\_Cover)$$

```
70:   (implies TeaSpoon (at-most 1 belongs Dessert_Cover))
```

$$TeaSpoon \sqsubseteq (\leq 1\ belongs.Dessert\_Cover)$$

```
71:   (implies SAUCER (at-most 1 belongs Dessert_Cover))
```

$$SAUCER \sqsubseteq (\leq 1\ belongs.Dessert\_Cover)$$

The combination of lines (63-71) indicates that all the objects could belong to at most one cover (either to an instance of ***Meal_Cover*** or to an instance of ***Dessert_Cover***, but not both). This is more restricted for the instances of ***PLATE*** and ***CUP***illustrated by lines (65) and (69).

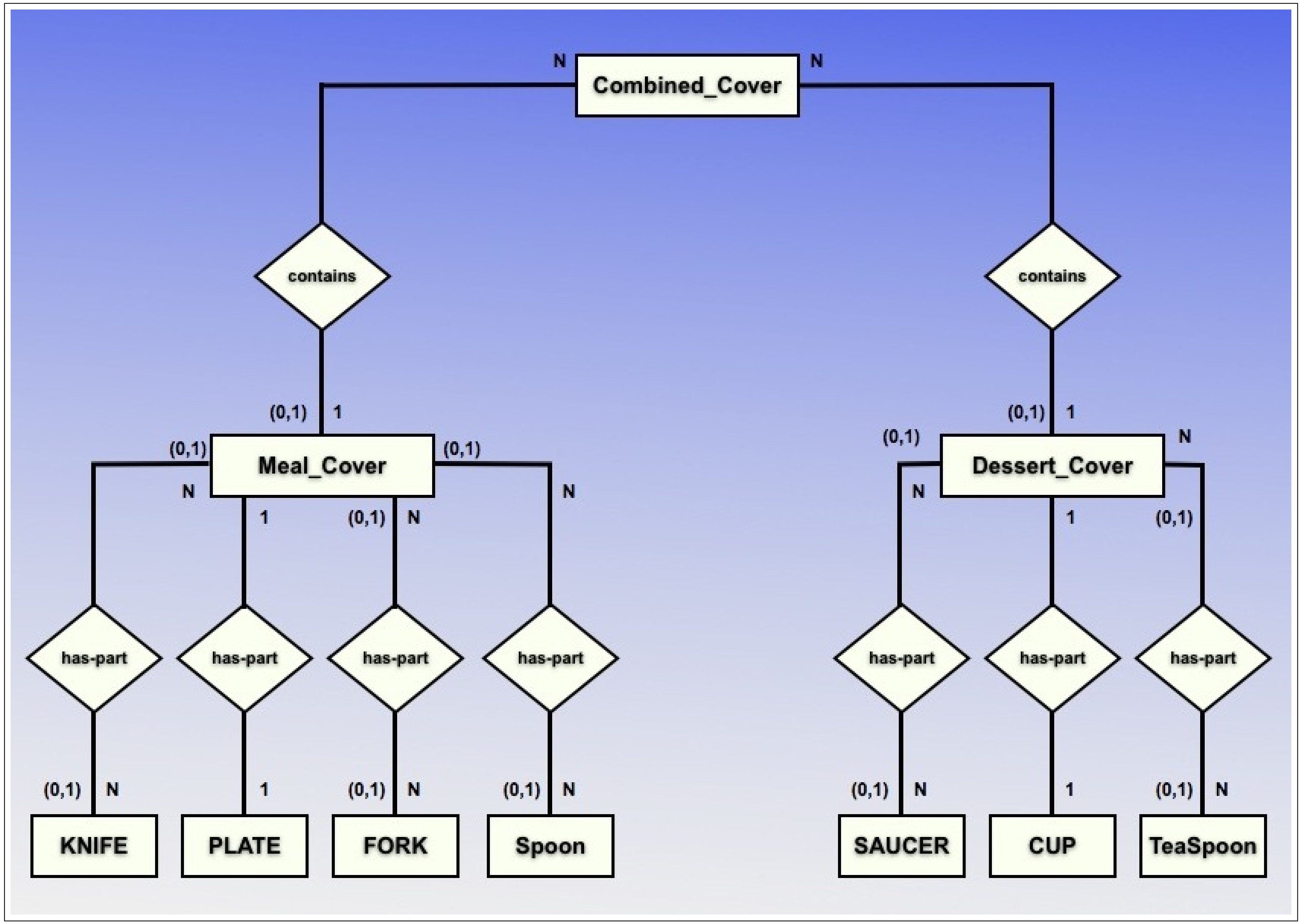


Figure 2.8: ER Diagram of Primitive Objects and Covers

In other words, if the data, which is processed by the implemented framework, contains an instance of ***PLATE***, this means that an instance of ***Meal_Cover*** could be created since the necessary and sufficient conditions for having an instance of ***Meal_Cover*** are fulfilled. The same story is true for the concepts ***CUP*** and ***Dessert_Cover***.

It should be noted that although the above definitions have been inserted in the TBox, it is not yet defined how RacerPro should find and create the aggregates. This will be discussed in the form of the rules in the next Chapter.

```
72:   (equivalent Lonely_Table (exactly 1 owns General_Cover))
73:   (equivalent Romantic_Table (exactly 2 owns General_Cover))
74:   (equivalent Family_Table (exactly 4 owns General_Cover))
75:   (equivalent Business_Table (exactly 6 owns General_Cover))
76:   (equivalent Big_Family_Table (exactly 12 owns General_Cover))
```

The description continues by defining the higher and more special aggregates. First, the types of the table are determined according to the number of the covers on the table or to be more precise, the number of persons who are going to be seated at the table.

As it is shown in the lines (72-76), it has been decided to relate the number of persons at a table to the environment of the table. For example, a table with two covers on it is a "Romantic Table" and a table with 12 covers is a "Big Family Table"[16].

The tables could be defined more specifically not only by the number of the covers on them but also by the type of these covers. Hence, it is important to identify the type of the different covers before identifying the type of the table[17].

```
77:   (equivalent SMALL_BREAKFAST_COVER (and cover
78:                                     (some has-part CUP)
79:                                     (some has-part SAUCER)
80:                                     (not (some has-part Spoon))
81:                                     (some has-part TeaSpoon)
82:                                     (some has-part PLATE)
83:                                     (some has-part KNIFE )
84:                                     (not (some has-part FORK)) ) )
85:   (equivalent SOUP_COVER (and cover
86:                                     (not (some has-part CUP))
87:                                     (not (some has-part SAUCER))
88:                                     (some has-part Spoon)
89:                                     (not (some has-part TeaSpoon))
```

[16]These definitions are based on personal choices and there could be some alternatives with different definitions in other contexts.

[17]Although a cover is either an instance of ***Meal_Cover***, ***Dessert_Cover*** or ***Combined_Cover***, it is not clear for which type of meal or dessert or, etc. this cover is used.Therefore, the above specifications are needed.

```
                                     (some has-part PLATE)
                                     (not (some has-part KNIFE ))
                                     (not (some has-part FORK)) ))
(equivalent SPAGHETTI_COVER (and cover
                                     (not (some has-part CUP))
                                     (not (some has-part SAUCER))
                                     (not (some has-part Spoon))
                                     (not (some has-part TeaSpoon))
                                     (some has-part PLATE)
                                     (not (some has-part KNIFE ))
                                     (some has-part FORK) ) )
(equivalent TEA_COVER (and cover
                                     (some has-part CUP)
                                     (some has-part SAUCER)
                                     (not (some has-part Spoon))
                                     (some has-part TeaSpoon)
                                     (not (some has-part PLATE))
                                     (not (some has-part KNIFE ))
                                     (not (some has-part FORK)) ) )
(equivalent BITTER_COFFEE_COVER (and cover
                                     (some has-part CUP)
                                     (some has-part SAUCER)
                                     (not (some has-part Spoon))
                                     (not (some has-part TeaSpoon))
                                     (not (some has-part PLATE))
                                     (not (some has-part KNIFE ))
                                     (not (some has-part FORK)) ))
(equivalent DINNER_COVER (and cover
                                     (not (some has-part CUP))
                                     (not (some has-part SAUCER))
                                     (some has-part Spoon)
                                     (not (some has-part TeaSpoon))
                                     (some has-part PLATE)
                                     (some has-part KNIFE )
                                     (some has-part FORK)) )
(equivalent STEAK_COVER (and cover
                                     (not (some has-part CUP))
                                     (not (some has-part SAUCER))
                                     (not (some has-part Spoon))
                                     (not (some has-part TeaSpoon))
                                     (some has-part PLATE)
                                     (some has-part KNIFE )
                                     (some has-part FORK) ) )
(equivalent DINNER_WITH_BITTER_COFFEE_COVER (and Combined_Cover
                                     (some contains DINNER_COVER)
```

```
134:                                      (some contains BITTER_COFFEE_COVER) ))
135:    (equivalent BIG_BREAKFAST_COVER (and Combined_Cover
136:                                      (some contains STEAK_COVER)
137:                                      (some contains TEA_COVER) ) )
138:    (equivalent FULL_DINNER_COVER (and Combined_Cover
139:                                      (some contains DINNER_COVER)
140:                                      (some contains TEA_COVER) ) )
```

The more specialized definitions of the covers are demonstrated in lines (77-140). For example, an instance of ***SMALL_BREAKFAST_COVER*** is a cover that has a cup, plate, teaspoon, knife and saucer and an instance of ***DINNER_WITH_BITTER_COFFEE_COVER*** is a combined cover containing a meal cover with the type ***DINNER_COVER*** and a dessert cover with the type ***BITTER_COFFEE_COVER*** (lines 132-134).

These lines directly state in the TBox what primitive objects each cover could have as its parts in a way that the covers become mutually exclusive. This feature prevents the situation where a cover has two different types, none of which is generally wrong.

Now that the different types of cover are defined, the tables could be specialized based on the type of the covers on them.

```
141:    (equivalent Big_Breakfast_Table (all owns BIG_BREAKFAST_COVER))
142:    (equivalent Small_Breakfast_Table (all owns SMALL_BREAKFAST_COVER))
143:    (equivalent Full_Dinner_With_Tea_Table       (all owns (and Combined_Cover
144:                                                          FULL_DINNER_COVER)))
145:    (equivalent Full_Dinner_With_Bitter_Coffee_Table (all owns (and Combined_Cover
146:                                                 DINNER_WITH_BITTER_COFFEE_COVER)))
147:    (equivalent Soup_Table                           (all owns SOUP_COVER))
148:    (equivalent Italian_Table                        (all owns SPAGHETTI_COVER))
149:    (equivalent English_Tea_Table                    (all owns TEA_COVER))
```

As shown above, seven different tables have been defined in the TBox. Since the definitions of the type of the covers make all of these types mutually exclusive, the types of the tables are also mutually exclusive.

As an example, the concepts ***Full_Dinner_With_Bitter_Coffee_Table*** and ***Small_Breakfast_Table*** are considered. The first table owns only the instances of ***SMALL_BREAKFAST_COVER*** and the second one owns instances of ***DINNER_WITH_BITTER_COFFEE_COVER*** (which are also instances of ***Combined_Cover***) because whenever a cup and a saucer are on the table without a teaspoon, it means that coffee is going to be served on the table with no sugar or milk. All the instances of cover on a table should be of the same concept regardless of their quantity.

Figure 2.9 shows the relationships between the instances of different entities for the tables and covers in the TBox as an ER diagram.

In addition to the concepts mentioned before, to make the scene interpreter a non-memoryless system, a new concept called ***Waitress*** is added to the ABox. If an instance of ***Waitress*** appears in the scene, it means that the scene has been taken in a restaurant; otherwise, it has been captured in the dining room of a house. This new concept has not been predicated in the tracking data; thus, such an instance should be instantiated by the framework. The framework decides randomly if an instance of ***Waitress*** is going to be seen during the clip. If the framework decides to have an instance of ***Waitress***, it randomly chooses a frame of a clip as a frame where the waitress supposedly appears in the scene; then, the interpreter should memorize that a waitress has become visible in a specific point of time in the scene, and it should consider this information for the rest of the interpretation.

At the beginning of the interpretation, the framework assumes that the scene is taken in a house; however, if an instance of ***Waitress*** appears in the scene, the framework changes its assumption.

Although the TBox is completed now, the methods describing how different primitive objects could be categorized under a cover and how they could build higher aggregates have not been declared yet. In other words, since the scene interpretation requires abduction rules, which have not been defined yet, it is impossible to form any aggregates[6]. It is also important how and in which direction the abduction rules are used and when a rule should be fired. All of these problems will be discussed in the next Chapter.

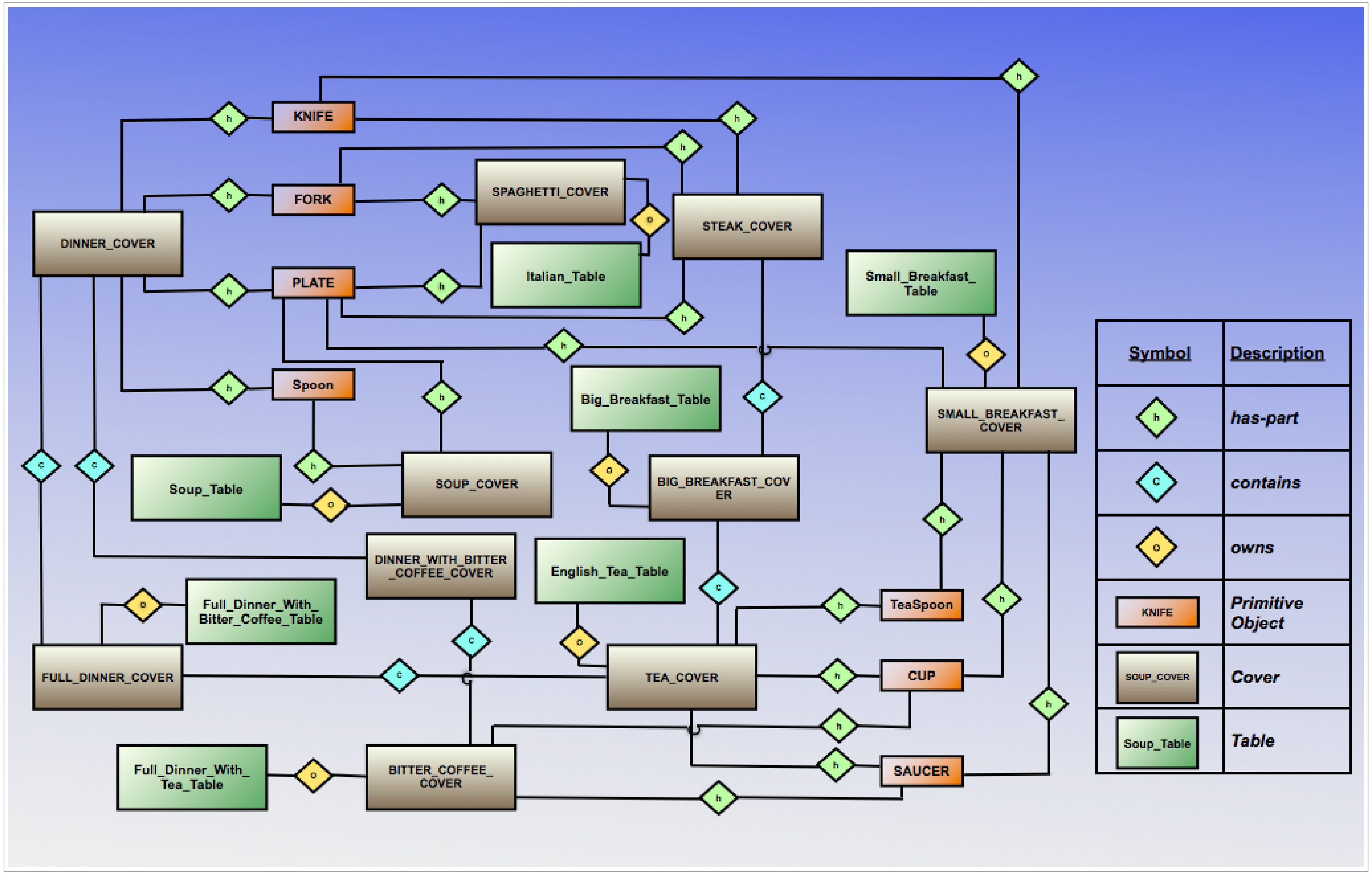


Figure 2.9: ER Diagram of the Table

Chapter 3

Rules and Strategies

Based on experience, beliefs and desires, each human being has his own interpretation when he is asked to interpret a scene from of a video clip. The differences might be between the specific points in time that a person starts hypothesizing and interpreting the scene or the methods he uses to make different hypotheses, search them and choose the best ones with a best explanation. In this Chapter, these issues are explained and answered in the terms of **Rules** and **Strategies**.

As discussed in the previous Chapters, for the interpretation of the ABox assertions regarding the scene, abductive reasoning and abduction rules are used in order to explain the effects and events in the scene(observations). By applying these rules [1] [2] some hypotheses (explanations) could be made. These hypotheses are either concept assertions or role assertions. In other words, if these hypotheses were added to the ABox, the observations that have been made in the scene could be explained.

The question that arises is **how** and **when** these rules should be used and applied in order to explain the observations. Section 3.1 explains how the rules should be defined and used. In this Section, different rule approaches called **bottom-up**, **top-down** and their variations are introduced and explained. Section 3.2 explains when the rules should be used and applied. Here three different strategies called the **Optimist**, **Pessimist** and **Realist** are introduced and discussed.

3.1 Rules

The abduction reasoning could be described as:

$$\Sigma \cup \Delta \models_{\mathcal{R}} \Gamma$$

where Σ is the background knowledge, Γ is the observations(effects), Δ is the explanations(causes) and $\mathcal{R}$ is the rules [8] [5].

It should be noted that there are different explanations for an observation; therefore, different hypotheses could be found.

An abduction rule has a head part and a body part. Generally, two approaches could be used by the interpretation. If the low-level knowledge (here the primitive

[1]The rules in this work have no time interval.

objects and the relations between them) is to be explained, higher aggregates should be hypothesized. This approach is called **bottom-up** and the rules are applied in this direction. The other approach is called **top-down** approach. In this approach, an instance of the aggregate is created in the form of a variable for each aggregate. By applying the rules in the backward chaining direction, it will be examined which assertions should be added to the ABox(should be hypothesized) so that the instance of the higher aggregate could be explained as an observation. In this work, both approaches have been studied. In addition, another approach called *Stepwise-top-down* is studied in which instead of beginning from the aggregates in the highest level, it starts with the aggregates in the lowest level. In the following sections, each of these approaches is discussed in depth, and the corresponding rules will be described.

It should be mentioned that RacerPro has a command called ***retrieve-with-explanation***, which is used to find hypotheses for the explanation of a role or a concept assertion. Furthermore, it is possible to use this command with variables in the head to find higher level explanations[2].

In addition to this command, another command called ***add-explanation- assertions*** is used for this work. This command makes it possible to add the hypothesized assertions that the ***retrieve-with-explanation*** command delivers according to the rules and queries. The combination of these two commands makes it possible to explain an observation, choose the best explanation for it and add the assertions based on the best result to the ABox.

3.1.1 Bottom-up Rules

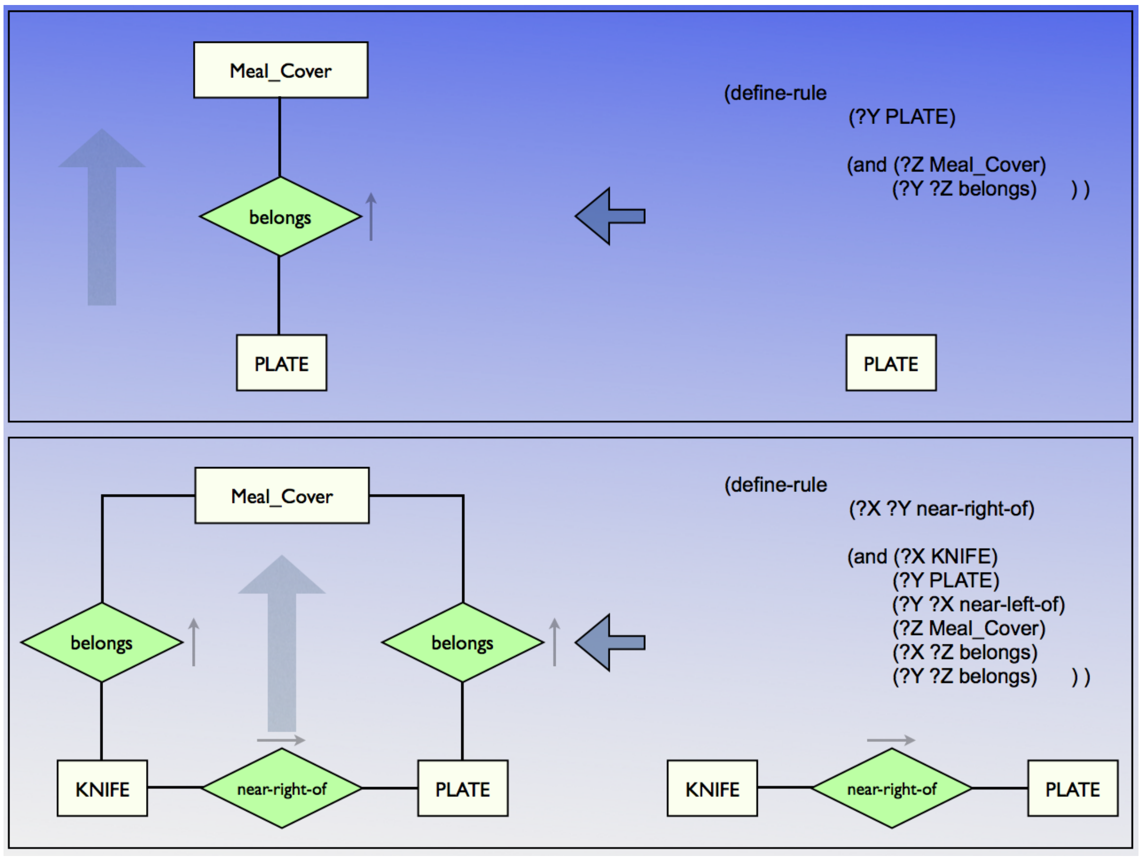
The tracking data from the image processing tools only delivers the primitive objects that are observed in the scene. The image processing tools are not able to instantiate the higher aggregates.

It is the task of the reasoner software to form and instantiate the aggregates using the primitive objects. This is done in a bottom-up approach way where the aggregates that are created are based on the primitive objects.Due to the fact that available evidences are used to form the upper concepts (aggregates), the word **bottom-up** is used for this approach.

As discussed in the previous Chapters, there are seven primitive objects for the table cloth scenario: ***PLATE, KNIFE, FORK, Spoon, TeaSpoon, CUP and SAUCER***. Their corresponding concepts are all located in the lowest level of the concept hierarchy.

In the level above the level of the primitive objects, the two aggregates ***Meal_Cover*** and ***Dessert_Cover*** are located. By combining the ***Meal_Cover*** and ***Dessert_Cover***

[2]The ***retrieve-with-explanation*** command uses a scoring function by default in order to specify the score of each result so that the best results with the highest score are chosen.This scoring function which, has been introduced by Paul R. Thagard, a Philosophy professor, suggests that the result, which uses a fewer number of hypotheses and more number of evidences found so far in the scene, is a better result. If $\mathbf{S_f}$ is the number of available evidences on the scene and $\mathbf{S_h}$ is the number of the hypothesized assertions, the scoring function is defined as follows: $\mathbf{S} = \mathbf{S_f} - \mathbf{S_h}$ [3]



```
(define-rule
        (?Y PLATE)

        (and (?Z Meal_Cover)
            (?Y ?Z belongs)      ) )
```

```
(define-rule
        (?X ?Y near-right-of)

        (and (?X KNIFE)
            (?Y PLATE)
            (?Y ?X near-left-of)
            (?Z Meal_Cover)
            (?X ?Z belongs)
            (?Y ?Z belongs)      ) )
```

Figure 3.1: bottom-up Rules for the Aggregate ***Meal_Cover*** using the Instances of ***PLATE*** and ***KNIFE***

under certain circumstances, an aggregate in a higher level called ***Combined_Cover*** could be instantiated. The most specific instantiators[3] of the instances of the aggregates could be specified before going to the next higher level, so that each aggregate could be described more precisely.

In the next level, the table aggregate is located, which could be specified more according to the cover instances that will be attached to it using the relation ***owns*** and the number of the covers that the table owns.

The approach described above shows briefly how the evidences available on the scene (primitive objects) are used to form the aggregates by the rules.

Before showing how the rules are defined and used, it is important to make some assumptions for the table cloth scenario. These assumptions are based on studies that were conducted by looking at the real life table cloth scenarios, specially at how a table should be covered using the primitive objects mentioned before in a standard way, regardless of where the table is located, whether in a public place like a restaurant or in a private place like the dining room of a house. In this work, these standards are applied.

For an instance of ***Meal_Cover*** having an instance of ***PLATE*** as one of its part

[3]By using the command ***most-specific-instantiators*** of RacerPro.

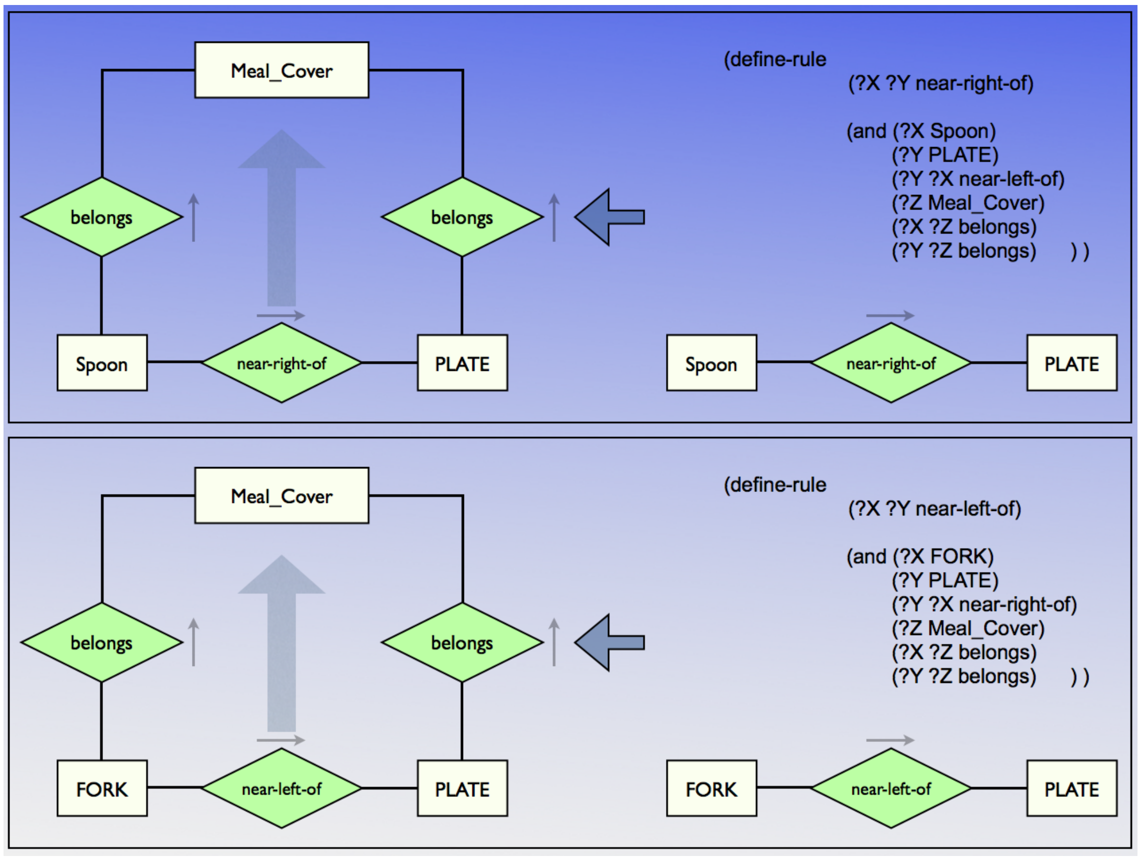


Figure 3.2: bottom-up Rules for the Aggregate ***Meal_Cover*** using the Instances of ***PLATE***, ***FORK*** and ***Spoon***

is a necessary condition (also sufficient to build a meal cover). A plate is usually the heart of the cover, and the cutlery objects are located next to the plate.

The standard location of the knives and spoons is usually on the right side of the plate and the standard location of the fork is on the left side of a plate.[4] Therefore, if an object is located on a non-standard position next to the plate, this object should be ignored; in other words, these rules should ignore this situation.

The model discussed above is shown in figures 3.1 and 3.2. The rules shown in these figures could also be described as follows:

$$\textit{1:}\quad \textit{PLATE(Y)} \longleftarrow \textit{Meal\_Cover(Z), belongs(Y,Z)}$$

$$\textit{2: near-right-of(X,Y)} \longleftarrow \textit{KNIFE(X), PLATE(Y),Meal\_Cover(Z),}$$
$$\textit{belongs(X,Z), belongs(Y,Z)}$$

$$\textit{3: near-right-of(X,Y)} \longleftarrow \textit{Spoon(X), PLATE(Y),Meal\_Cover(Z),}$$

[4]It should be mentioned that it is also possible to locate the cutlery objects on different positions next to a plate;however, since this is not standard and the scene interpretation of this scenario, should be real and standard, the standard definitions are applied.

belongs(X,Z), belongs(Y,Z)

4: ***near-left-of(X,Y)*** $\longleftarrow$ ***FORK(X), PLATE(Y),Meal_Cover(Z), belongs(X,Z), belongs(Y,Z)***

The first rule explains that an instance of ***PLATE*** could exist, because there is an instance of ***Meal_Cover*** and for this instance, the instance of ***PLATE*** is a filler of the role ***has-part***.

The second rule indicates that if an instance of ***KNIFE*** is the nearest instance of all the knives on the scene which are on the right side of an instance of ***PLATE***, this could be explained by an instance of ***Meal_Cover*** which has the instance of ***PLATE*** and ***KNIFE*** as fillers for the role ***has-part***. In other words, the instances of these two primitive objects belong to an instance of ***Meal_Cover***. This is also true for the third rule, but for an instance of ***Spoon*** instead of ***KNIFE***.

Finally, the forth rule indicates that if an instance of ***FORK*** is the nearest instance of all the forks on the scene which are on the left side of an instance of ***PLATE***, this could be explained as follows: There exists an instance of ***Meal_Cover*** which has the instances of ***PLATE*** and ***FORK*** as fillers for the role ***has-part***. In other words, the instances of these two primitive objects belong to an instance of ***Meal_Cover***.

The necessary condition (also sufficient) for an instance of ***Dessert_Cover*** is having an instance of ***CUP*** as its part. In other words, an instance of ***CUP*** must be a filler of the role ***has-part*** for each instance of ***Dessert_Cover*** and having an instance of ***CUP*** is sufficient for forming an instance of ***Dessert_Cover***.

The position of an instance of ***SAUCER*** (in case of existence) regarding the cup it belongs to is clearly under the cup. This means that both of these objects have the same X-Y position on the scene; hence, the role ***on-same-position-of*** is defined for this situation. The standard position of an instance of ***TeaSpoon*** is usually on the right side of the cup and saucer (in case of existence). As a result, other non-standard situations should be ignored.

The standard situations discussed above are illustrated in figures 3.3 and 3.3 and their corresponding rules could be shown as follows:

1: ***CUP(Y)*** $\longleftarrow$ ***Dessert_Cover(Z), belongs(Y,Z)***

2: ***near-right-of(X,Y)*** $\longleftarrow$ ***TeaSpoon(X), CUP(Y), Dessert_Cover(Z), belongs(X,Z), belongs(Y,Z)***

3: ***on-same-position-of(X,Y)*** $\longleftarrow$ ***SAUCER(X), CUP(Y), Dessert_Cover(Z), belongs(X,Z), belongs(Y,Z)***

The first rule indicates that if an instance of ***CUP*** exists, it could mean that

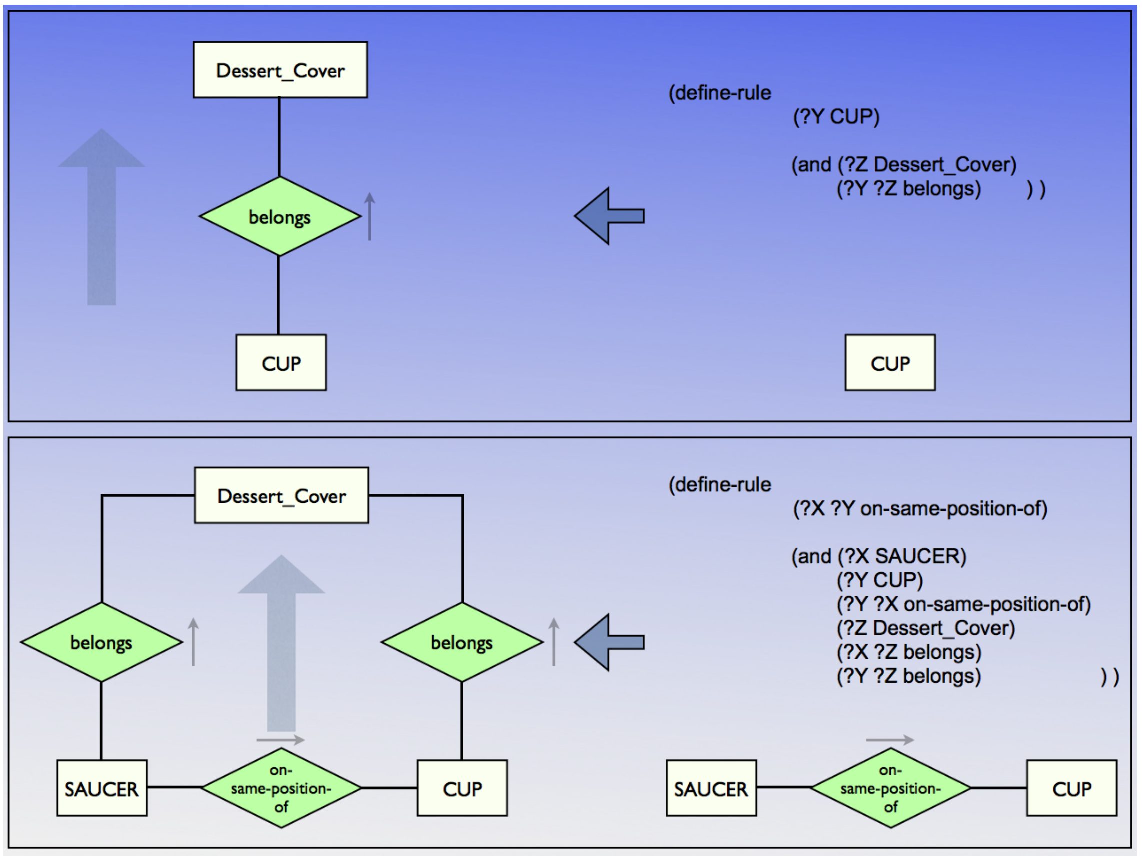


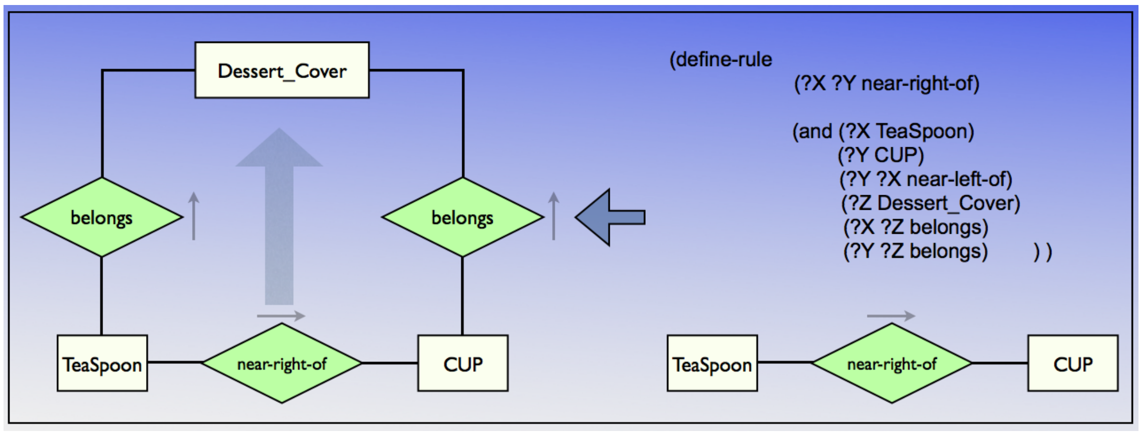
Figure 3.3: bottom-up Rules for the Aggregate ***Dessert_Cover*** using the Instances of ***CUP*** and ***SAUCER***

there is an instance of ***Dessert_Cover*** and for this instance, the instance of ***CUP*** is a filler of the role ***has-part***.

The second rule indicates that if an instance of ***SAUCER*** is located on the same position where an instance of ***CUP*** is located, it could mean that there is an instance of ***Dessert_Cover*** which has the instances of ***CUP*** and ***SAUCER*** as fillers for the role ***has-part***. In other words, the instances of these two primitive objects belong to an instance of ***Dessert_Cover***.

Finally, the third rule indicates that if an instance of ***TeaSpoon*** is the nearest instance of all the teaspoons on the scene which are on the right side of an instance of ***CUP***, this could be explained as follows: There is an instance of ***Dessert_Cover*** which has the instances of ***CUP*** and ***TeaSpoon*** as fillers for the role ***has-part***; therefore, the instances of these two primitive objects belong to the instance of ***Dessert_Cover***.

An instance of ***Combined_Cover*** could be formed by two different approaches: either the primitive objects are used to form such an instance or an instance of ***Meal_Cover*** and an instance ***Dessert_Cover*** are used. These two approaches are introduced as:



```
(define-rule
        (?X ?Y near-right-of)

        (and (?X TeaSpoon)
             (?Y CUP)
             (?Y ?X near-left-of)
             (?Z Dessert_Cover)
             (?X ?Z belongs)
             (?Y ?Z belongs)       ) )
```

Figure 3.4: bottom-up Rule for the Aggregate ***Dessert_Cover*** using the Instances of ***CUP*** and ***TeaSpoon***

3.1.1.1 Absolute-bottom-up

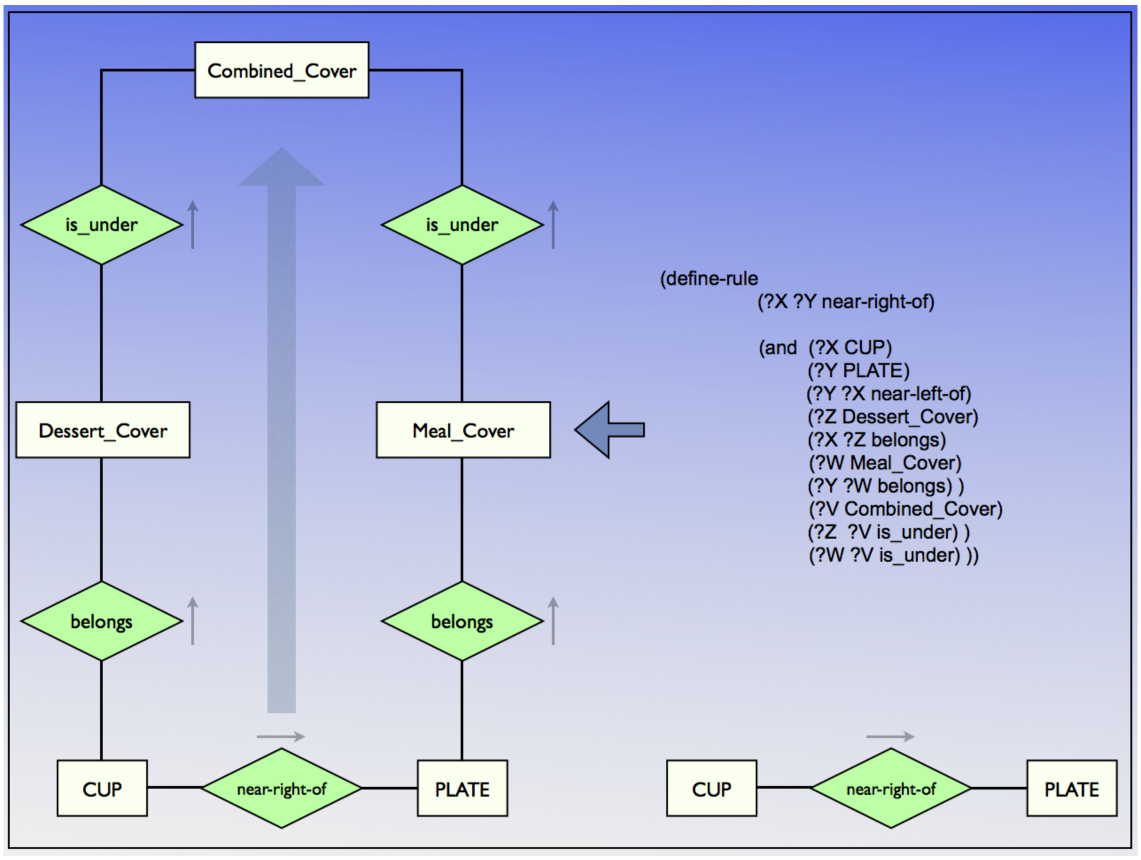
The first to form an instance of ***Combined_Cover*** is to use the primitive objects as shown in figure 3.5. According to this figure, an instance of ***CUP*** is the nearest cup on the right side of an instance of ***PLATE***, because the standard location of a cup is on the right side of plate. As it has been mentioned, an instance of ***CUP*** is sufficient to form an instance of ***Dessert_Cover*** and an instance of ***PLATE*** is sufficient to form an instance of ***Meal_Cover***. Thus, the instance of ***Dessert_Cover*** is also on the right side of the instance of ***Meal_Cover*** and also the nearest one. As a result, an instance of ***Combined_Cover*** is formed by using two primitive objects.

The corresponding rule is shown as follows:

$$\begin{aligned}\textit{near-right-of}(X,Y) \longleftarrow\ & \textit{CUP}(X),\ \textit{PLATE}(Y),\\ & \textit{Dessert\_Cover}(Z),\ \textit{belongs}(X,Z),\\ & \textit{Meal\_Cover}(W),\ \textit{belongs}(Y,W),\\ & \textit{Combined\_Cover}(V),\\ & \textit{is\_under}(Z,V),\ \textit{is\_under}(W,V)\end{aligned}$$

3.1.1.2 Stepwise-bottom-up

Another way to form an instance of ***Combined_Cover*** is to use the instances of ***Meal_Cover*** and ***Dessert_Cover*** under the condition that the spatial relationships between the instances of these two concepts are already given in the ABox. Therefore, if an instance of ***Dessert_Cover*** is the nearest cover among all the dessert covers, which are on the right side of an instance of ***Meal_Cover***, an instance of ***Combined_Cover*** could be formed using the meal and dessert covers as fillers of the role ***contains*** for the combined cover. This is shown on figure 3.6 with the following rule:



```
(define-rule
        (?X ?Y near-right-of)

        (and  (?X CUP)
              (?Y PLATE)
              (?Y ?X near-left-of)
              (?Z Dessert_Cover)
              (?X ?Z belongs)
              (?W Meal_Cover)
              (?Y ?W belongs) )
              (?V Combined_Cover)
              (?Z  ?V is_under) )
              (?W ?V is_under) ))
```

Figure 3.5: bottom-up Rule for the Aggregate ***Combined_Cover*** using the Instances of ***CUP*** and ***PLATE***

$$\begin{aligned} \textit{near-right-of}(X,Y) \longleftarrow\ & \textit{Dessert\_Cover}(X) \\ & \textit{Meal\_Cover}(Y), \\ & \textit{Combined\_Cover}(Z) \\ & \textit{is\_under}(X,Z),\ \textit{is\_under}(Y,Z) \end{aligned}$$

As it can be seen, the second option uses only the aggregates and not the primitive objects. Nevertheless, the relation ***near-right-of*** between these aggregates is determined using the primitive objects in the lower levels of the interpretation.

■

It is possible for the user of the framework of this work to choose which rule could be used to form the instances of ***Combined_Cover***. This will be described in the next Chapter.

3.1.2 Top-down Rules

Another approach in high-level scene interpretation is to generate hypotheses of high-level assertions and aggregates and to use the top-down rules [21] [16] in

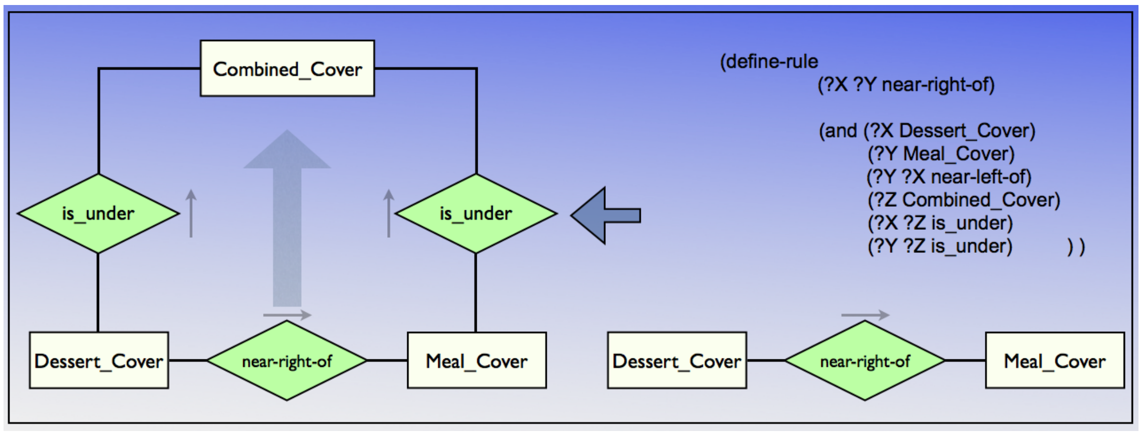


Figure 3.6: bottom-up Rule for the Aggregate ***Combined_Cover*** using the Instances of ***Meal_Cover*** and ***Dessert_Cover***

order to find the missing objects, evidences and low-level image analysis results that should be present in the scene to make these hypotheses become valid.

In other words, the hypotheses should be explained by using the evidences that are available in the scene. This approach is called then the **top-down** approach.

The ***retrieve-with-explanation*** command of RacerPro that was mentioned earlier, could be used again for this matter. This command creates a temporary instance of the aggregate that should be formed. The temporary instance is in fact a variable. The command tries to explain why this variable exists in the knowledge base using the evidences that are available in the scene. If such evidences are not found or are partly found, RacerPro creates temporary instances of the concept or role assertions that are missing and are needed to form an instance of the aggregate, which the command tries to find and explain.

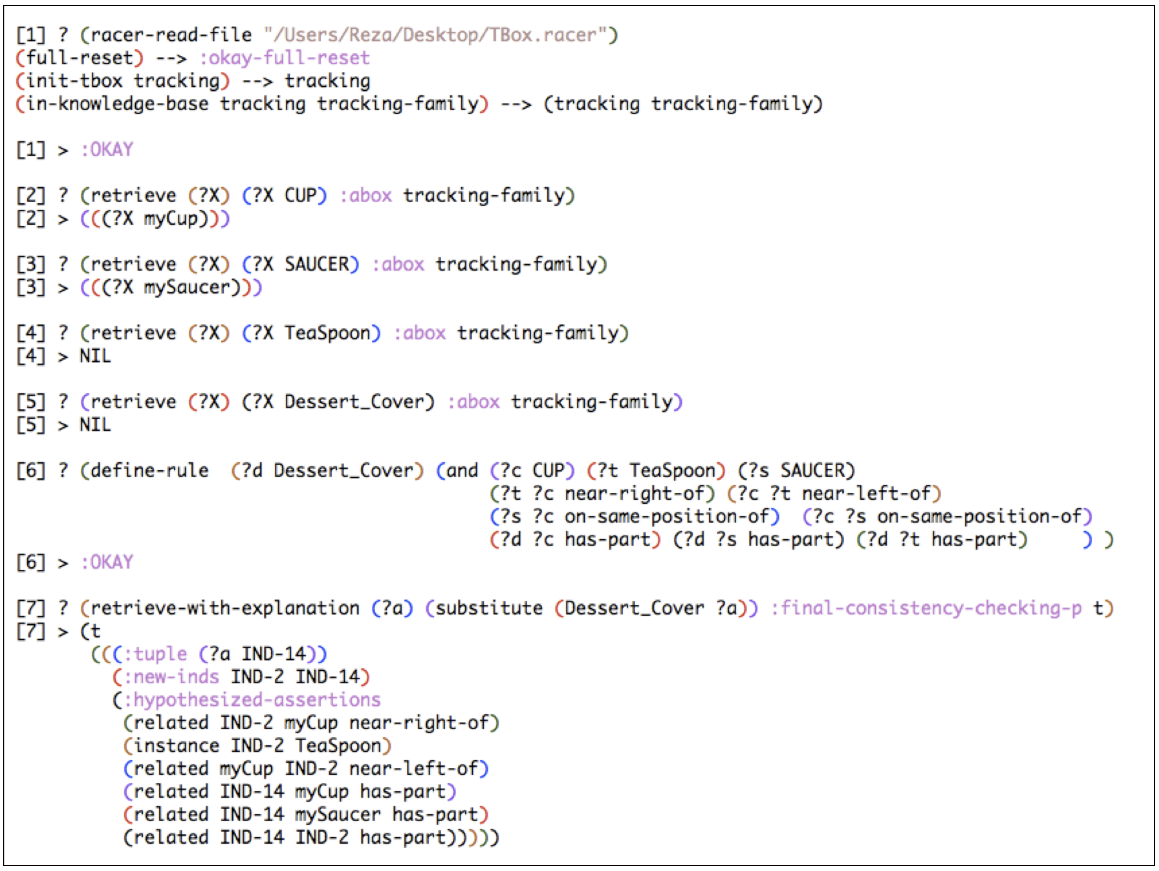
It should be noted that using the ***retrieve-with-explanation*** command in the top-down mode requires the aggregate, that should be found, to occur in the head of the rule. The rule body contains the assertions that explain why the aggregate (which will be addressed as a variable) in the head exists. These assertions may be already available fully or partly in the knowledge base or even not available at all. The missing parts will also be constructed to complete the rule.

The ***retrieve-with-explanation*** command should be called in the ***:final- consistency - checking t*** mode. Otherwise false assertions or explanations would be suggested.

A small example to show how this command works is illustrated in figure 3.7. The first line shows that the TBox[5] and the ABox are loaded. Lines (2-5) show that there is an instance of ***CUP*** (myCup) and an instance of ***SAUCER*** (mySaucer) and no instances of ***TeaSpoon*** and ***Dessert_Cover*** in the ABox.

In the sixth line, the rule that should be used to form an instance of ***Dessert_Cover*** in a top-down direction is shown. This rule indicates that an instance of ***Dessert_ Cover*** could exist because there are a cup, teaspoon and saucer in the scene, and

[5]The TBox is described in Chapter 2, Section 4.



```
[1] ? (racer-read-file "/Users/Reza/Desktop/TBox.racer")
(full-reset) --> :okay-full-reset
(init-tbox tracking) --> tracking
(in-knowledge-base tracking tracking-family) --> (tracking tracking-family)

[1] > :OKAY

[2] ? (retrieve (?X) (?X CUP) :abox tracking-family)
[2] > (((?X myCup)))

[3] ? (retrieve (?X) (?X SAUCER) :abox tracking-family)
[3] > (((?X mySaucer)))

[4] ? (retrieve (?X) (?X TeaSpoon) :abox tracking-family)
[4] > NIL

[5] ? (retrieve (?X) (?X Dessert_Cover) :abox tracking-family)
[5] > NIL

[6] ? (define-rule  (?d Dessert_Cover) (and (?c CUP) (?t TeaSpoon) (?s SAUCER)
                                            (?t ?c near-right-of) (?c ?t near-left-of)
                                            (?s ?c on-same-position-of)  (?c ?s on-same-position-of)
                                            (?d ?c has-part) (?d ?s has-part) (?d ?t has-part)     ) )
[6] > :OKAY

[7] ? (retrieve-with-explanation (?a) (substitute (Dessert_Cover ?a)) :final-consistency-checking-p t)
[7] > (t
       (((:tuple (?a IND-14))
         (:new-inds IND-2 IND-14)
         (:hypothesized-assertions
          (related IND-2 myCup near-right-of)
          (instance IND-2 TeaSpoon)
          (related myCup IND-2 near-left-of)
          (related IND-14 myCup has-part)
          (related IND-14 mySaucer has-part)
          (related IND-14 IND-2 has-part))))
```

Figure 3.7: The ***retrieve-with-explanation*** Command

the saucer is on the same position of the cup (the cup is on the saucer) and the cup is the nearest instance among all instances of ***CUP*** on the left side of the teaspoon. In addition, these three primitive objects belong to the instance of the ***Dessert_Cover***.

By creating an instance of ***Dessert_Cover*** indirectly in form of a variable and calling the ***retrieve-with-explanation*** command for this variable, all of the parts in the body of the rule should exist, so that the variable becomes valid as an instance of ***Dessert_Cover***. If there are missing parts, they should be instantiated.

If the command *retrieve-with-explanation* is called for this example, since there is an instance of ***CUP***(myCup) and *SAUCER*(mySaucer) on the same position(*on-same-position-of(myCup, mySaucer)* is entailed), the first, third and fifth tuples of the body of the rule are entailed. The rest of the tuples are not entailed and should be hypothesized first.

The seventh line shows the result of the *retrieve-with-explanation* command. ***IND-14*** is the instance of ***Dessert_Cover*** that has been created for the variable ***?x*** (in other words, it is hypothesized for the current situation in the knowledge base).

The other new individual is ***IND-2*** which is an instance of *TeaSpoon* and has been hypothesized because no instance of *TeaSpoon* has been yet discovered in the scene; hence, this missing evidence must be hypothesized and added first. The teaspoon should be positioned correctly according to the rule. This is done by hypothesizing the role assertion *near-left-of(myCup, IND-2)*[6] .

Finally, the only missing evidences are stating that the primitive objects belong to the instance of the ***Dessert_Cover*** which has been hypothesized earlier. This is done by the three role assertions for the role *has-part*.

The aggregates that are addressed in the head of the rules for the top-down mode are ***Combined_Cover***, ***Meal_Cover*** and ***Dessert_Cover***.

Figure 3.8 shows the rule defined for the concept ***Combined_Cover*** which could also be described as follows:

$$\begin{array}{rl} \textbf{\textit{Combined\_Cover(X)}} \longleftarrow & \textbf{\textit{Dessert\_Cover(Y)}} \\ & \textbf{\textit{Meal\_Cover(Z),}} \\ & \textbf{\textit{CUP(C), PLATE(P)}} \\ & \textbf{\textit{has-part(Y,C), has-part(Z,P)}} \\ & \textbf{\textit{near-right-of(C,P), near-left-of(P,C)}} \\ & \textbf{\textit{contains(X,Y), contains(X,Z)}} \end{array}$$

As it can be seen, an instance of a ***Combined_Cover*** could exist when there is an instance of ***Meal_Cover*** and an instance of ***Dessert_Cover*** available. Since these two instances should be hypothesized, an instance of *PLATE* and an instance of *CUP* are needed, because they are the necessary and sufficient condition for forming their corresponding covers. Additionally, the instance of *CUP* should be the nearest individual among the cups that are on the right side of the instance of *PLATE* so that the condition for the standard location of a cup regarding a plate is fulfilled.

If this rule results in assertions for ***Meal_Cover*** and ***Dessert_Cover***, the rest of the primitive objects that should be added to the cover groups will be added using the rules in the lower levels.

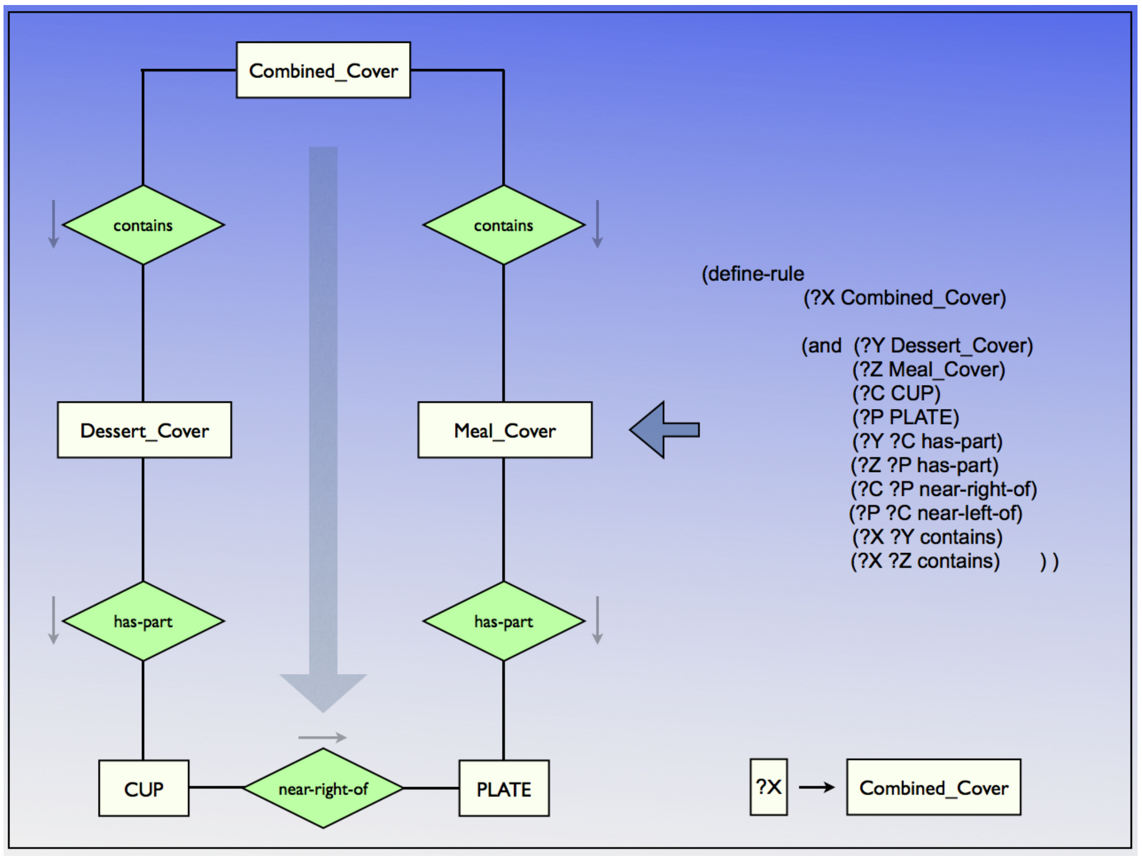
The next two aggregates which are addressed in the rules in the lower level are ***Meal_Cover*** and ***Dessert_Cover***.

There are different combinations of the primitive objects for these two aggregates; therefore, there are different explanations for why an instance of each of these aggregates could exist. It is also possible that more than one explanation is correct for the current situation in the scene. The decision which explanation should be chosen and which assertions should be added into the ABox is made as discussed before by using the scoring function that has been implemented in the *retrieve-with-explanation* command in RacerPro. This issue will be discussed later, after describing the rules for the two aggregates mentioned above.

The rules for the explanation of an instance of ***Meal_Cover*** are as follows:

$$\begin{array}{rl} \textbf{\textit{Meal\_Cover(X)}} \longleftarrow & \textbf{\textit{PLATE(Y), FORK(Z),}} \\ & \textbf{\textit{KNIFE(W), Spoon(V),}} \\ & \textbf{\textit{near-right-of(Y,Z),}} \\ & \textbf{\textit{near-left-of(Y,W),}} \end{array}$$

[6]This is equivalent to *near-right-of(IND-2, myCup)*, because the role *near-right-of* is the inverse role of *near-left-of*.



```
(define-rule
        (?X Combined_Cover)

        (and  (?Y Dessert_Cover)
              (?Z Meal_Cover)
              (?C CUP)
              (?P PLATE)
              (?Y ?C has-part)
              (?Z ?P has-part)
              (?C ?P near-right-of)
              (?P ?C near-left-of)
              (?X ?Y contains)
              (?X ?Z contains)      ) )
```

Figure 3.8: The top-down Rule for the *Combined_Cover* Aggregate

$$\begin{aligned}
& \quad \textit{near-left-of}(Y,V),\\
& \quad \textit{has-part}(X,Y),\ \textit{has-part}(X,Z),\\
& \quad \textit{has-part}(X,W),\ \textit{has-part}(X,V)\\[1em]
\textit{Meal\_Cover}(X) \longleftarrow\ & \textit{PLATE}(Y),\\
& \quad \textit{FORK}(Z),\ \textit{Spoon}(W),\\
& \quad \textit{near-right-of}(Y,Z),\\
& \quad \textit{near-left-of}(Y,W),\\
& \quad \textit{has-part}(X,Y),\\
& \quad \textit{has-part}(X,Z),\ \textit{has-part}(X,W)\\[1em]
\textit{Meal\_Cover}(X) \longleftarrow\ & \textit{PLATE}(Y),\\
& \quad \textit{FORK}(Z),\ \textit{KNIFE}(W),\\
& \quad \textit{near-right-of}(Y,Z),\\
& \quad \textit{near-left-of}(Y,W),\\
& \quad \textit{has-part}(X,Y),\\
& \quad \textit{has-part}(X,Z),\ \textit{has-part}(X,W)\\[1em]
\textit{Meal\_Cover}(X) \longleftarrow\ & \textit{PLATE}(Y),\\
& \quad \textit{Spoon}(Z),\ \textit{KNIFE}(W),\\
& \quad \textit{near-left-of}(Y,Z),\\
& \quad \textit{near-left-of}(Y,W),\\
& \quad \textit{has-part}(X,Y),\\
& \quad \textit{has-part}(X,Z),\ \textit{has-part}(X,W)
\end{aligned}$$

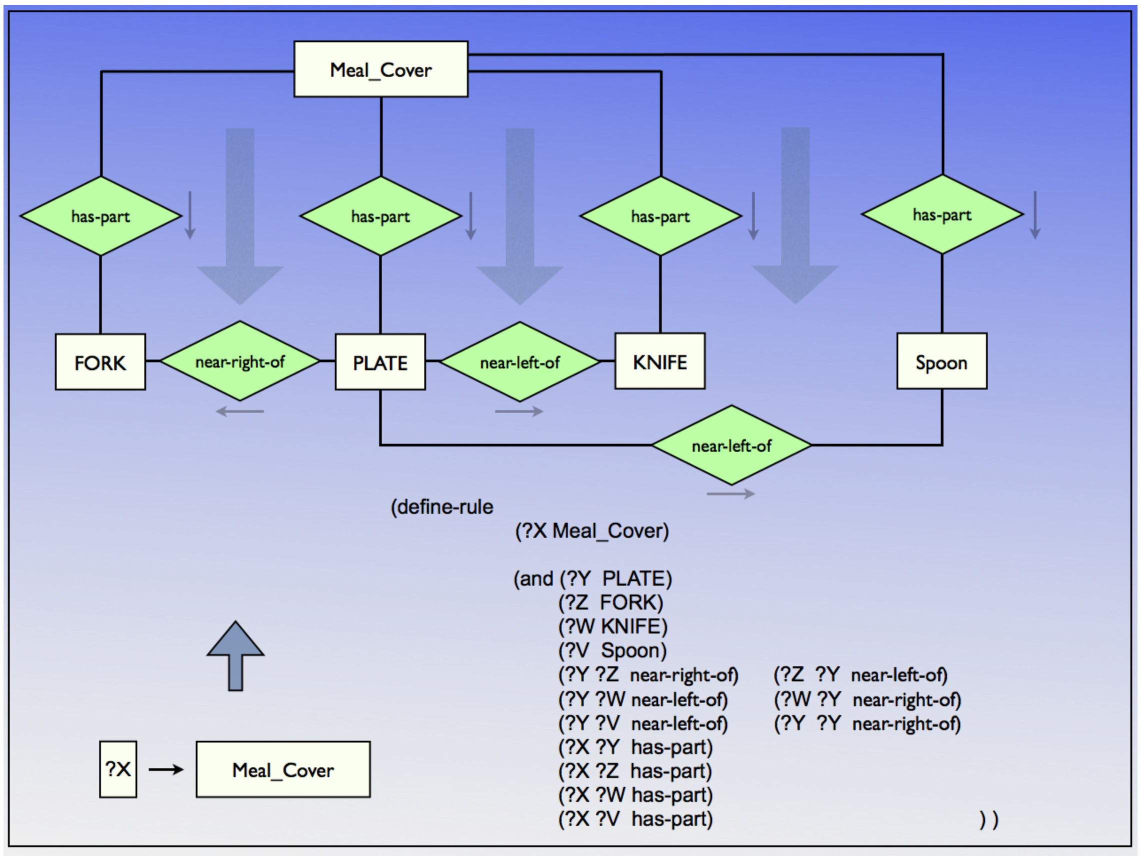
$$
\begin{array}{rl}
\textit{\textbf{Meal\_Cover(X)}} \longleftarrow & \textit{\textbf{PLATE(Y),}} \\
& \textit{\textbf{Spoon(Z)}} \\
& \textit{\textbf{near-left-of(Y,Z),}} \\
& \textit{\textbf{has-part(X,Y), has-part(X,Z)}} \\
\textit{\textbf{Meal\_Cover(X)}} \longleftarrow & \textit{\textbf{PLATE(Y),}} \\
& \textit{\textbf{FORK(Z)}} \\
& \textit{\textbf{near-right-of(Y,Z),}} \\
& \textit{\textbf{has-part(X,Y), has-part(X,Z)}} \\
\textit{\textbf{Meal\_Cover(X)}} \longleftarrow & \textit{\textbf{PLATE(Y),}} \\
& \textit{\textbf{KNIFE(Z)}} \\
& \textit{\textbf{near-left-of(Y,Z),}} \\
& \textit{\textbf{has-part(X,Y), has-part(X,Z)}} \\
\textit{\textbf{Meal\_Cover(X)}} \longleftarrow & \textit{\textbf{PLATE(Y), has-part(X,Y)}} \\
\textit{\textbf{Dessert\_Cover(X)}} \longleftarrow & \textit{\textbf{CUP(Y),}} \\
& \textit{\textbf{TeaSpoon(Z)}} \\
& \textit{\textbf{SAUCER(W)}} \\
& \textit{\textbf{near-left-of(Y,Z),}} \\
& \textit{\textbf{on-same-position-of(Y,W),}} \\
& \textit{\textbf{has-part(X,Y),}} \\
& \textit{\textbf{has-part(X,Z), has-part(X,W)}} \\
\textit{\textbf{Dessert\_Cover(X)}} \longleftarrow & \textit{\textbf{CUP(Y),}} \\
& \textit{\textbf{TeaSpoon(Z)}} \\
& \textit{\textbf{near-left-of(Y,Z),}} \\
& \textit{\textbf{has-part(X,Y), has-part(X,Z)}} \\
\textit{\textbf{Dessert\_Cover(X)}} \longleftarrow & \textit{\textbf{CUP(Y),}} \\
& \textit{\textbf{SAUCER(Z)}} \\
& \textit{\textbf{on-same-position-of(Y,Z),}} \\
& \textit{\textbf{has-part(X,Y), has-part(X,Z)}} \\
\textit{\textbf{Dessert\_Cover(X)}} \longleftarrow & \textit{\textbf{CUP(Y), has-part(X,Y)}}
\end{array}
$$

Figures 3.9, 3.10, 3.11 and 3.12 show different combinations of the primitive objects that could be the different explanations for an instance of ***Meal_Cover***. It should be mentioned again that in this work the instance of ***Meal_Cover*** will be created in terms of a variable for the top-down direction; the reasoner then should try to explain if this instance could exist or not; if yes, it determines which ABox assertions are needed for the corresponding explanation.

Figure 3.9 shows a situation where all four primitive objects that a meal cover could have are available in the scene and could be used to explain the instance of ***Meal_Cover***. In figure 3.10, those situations in which only three out of four primitive objects, which belong to an instance of ***Meal_Cover***, are available in the scene are illustrated. The same story is also shown in figure 3.11 where instead of three primitive objects, the instances of ***Meal_Cover*** could only have two primitive objects as their parts.

Finally, in figure 3.12, the situation where an instance of ***Meal_Cover*** could be explained due to the presence of only an instance of ***PLATE*** in the scene is presented.

As it is shown, in all of the situations above, an instance of ***PLATE*** should always



```
(define-rule
          (?X Meal_Cover)

          (and (?Y  PLATE)
               (?Z  FORK)
               (?W KNIFE)
               (?V  Spoon)
               (?Y ?Z  near-right-of)    (?Z  ?Y  near-left-of)
               (?Y ?W near-left-of)      (?W ?Y  near-right-of)
               (?Y ?V  near-left-of)     (?Y  ?Y  near-right-of)
               (?X ?Y  has-part)
               (?X ?Z  has-part)
               (?X ?W has-part)
               (?X ?V  has-part)                             ))
```

Figure 3.9: The top-down Rule for the ***Meal_Cover*** Aggregate

be present because this is a necessary condition for the existence of an instance of ***Meal_Cover***.

Figures 3.13 and 3.14 show the different rule definitions for the concept ***Dessert_Cover***.

The first model in figure 3.13 shows the situation where all the objects that an instance of ***Dessert_Cover*** could have as its parts are present.

In the second model of figure 3.13 and in the first model of figure 3.14, those situations in which only two (out of three) primitive objects, which belong to an instance of ***Dessert_Cover***, are available in the scene are illustrated.

Finally, the second model of figure 3.14 shows the initial necessary (and sufficient) condition for an instance of ***Dessert_Cover*** where only an instance of ***CUP*** is available in the scene. This condition is also fulfilled in other models of ***Dessert_Cover***.

As discussed earlier, before forming the higher aggregates by using the top-down rules, the ABox contains only the primitive objects that have been found in the scene by the image processing tools. There are different approaches for how the top-down rules for forming the higher aggregates are applied. In this work, two different approaches are introduced[7].

[7]The comparison between the results of each approach is studied and shown in Chapters 5 and 6.

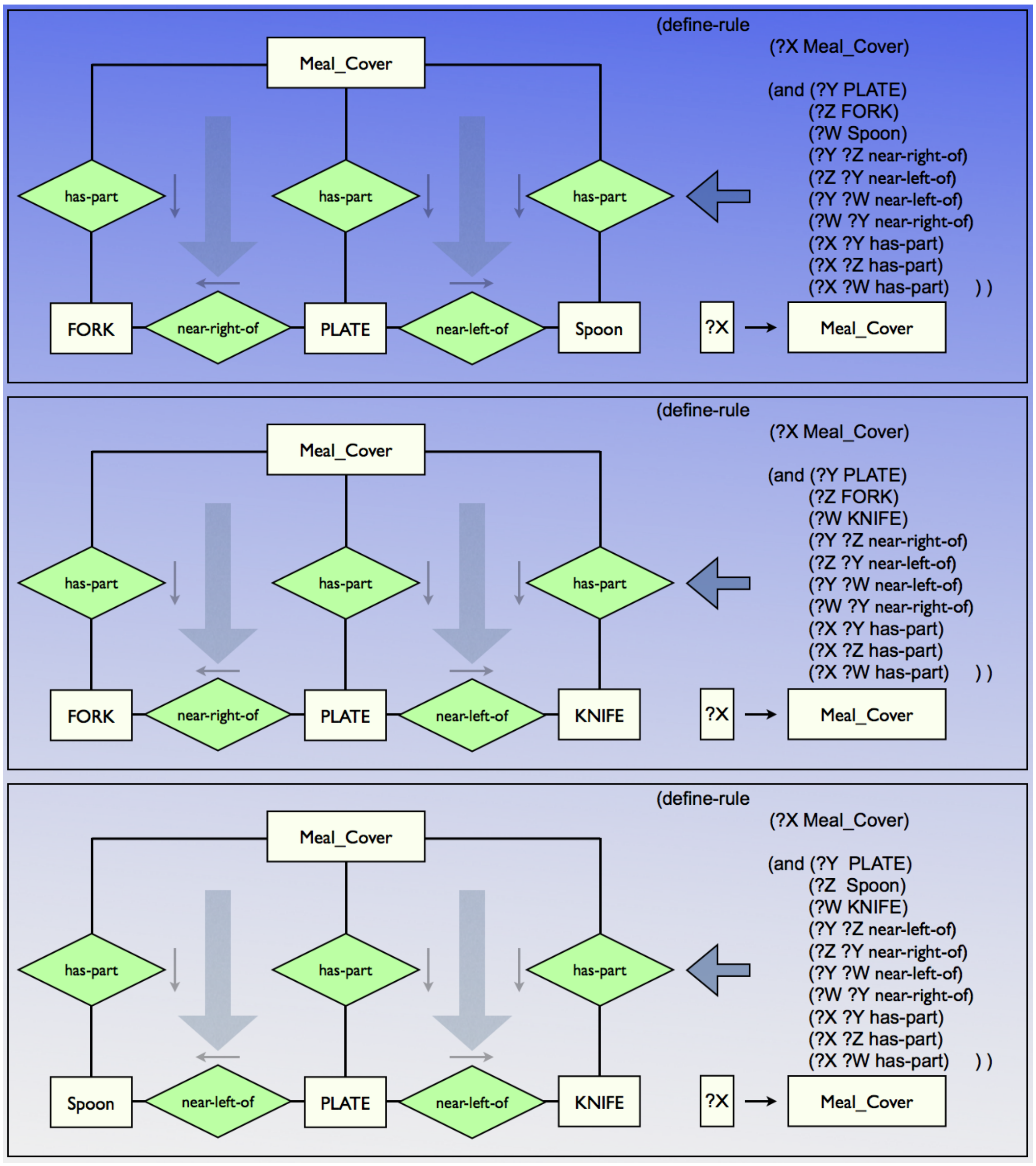


Figure 3.10: The top-down Rules for the ***Meal_Cover*** Aggregate using Three Primitive Objects

3.1.2.1 Absolute-top-down

In this approach, the instances of the aggregates in the highest level are formed first by applying the top-down rules. Each level has a higher priority than its lower level and a lower priority than its higher level.

By applying this approach, the aggregates are formed starting from the highest level and ending in the lowest level. If two concepts are in the same level, the order by which they are formed does not matter. In our example, the instances of the aggregate in the higher level, which is ***Combined_Cover***, are created first; then, the instances of the aggregates in the lower level, which are the instances of ***Meal_Cover*** and ***Dessert_Cover***, are created.

3.1.2.2 Stepwise-top-down

This approach uses the same top-down rules used before; however, the order by which these rules are called is different; namely, the instances of the aggregates in

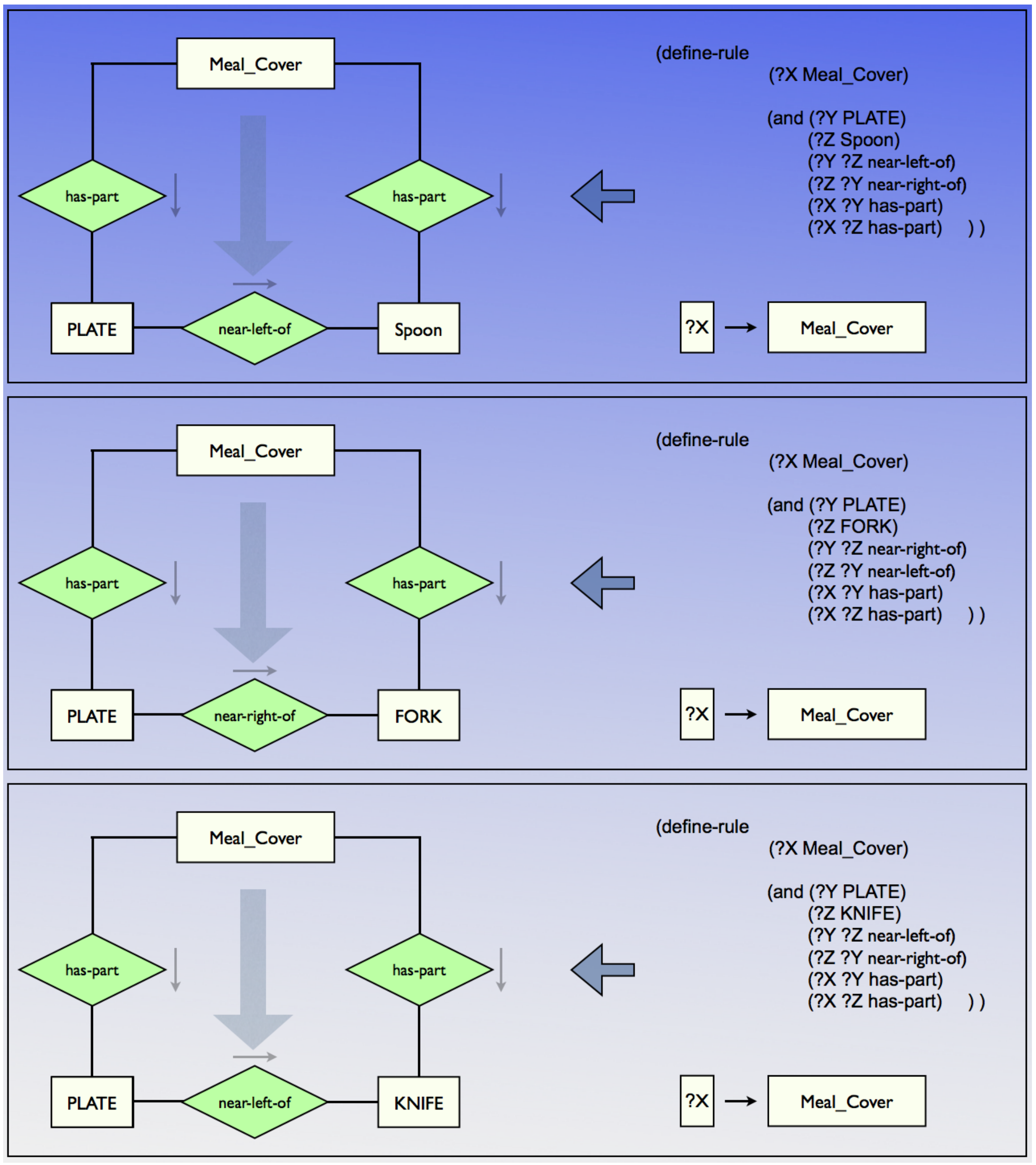


Figure 3.11: The top-down Rules for the ***Meal_Cover*** Aggregate using Two Primitive Object

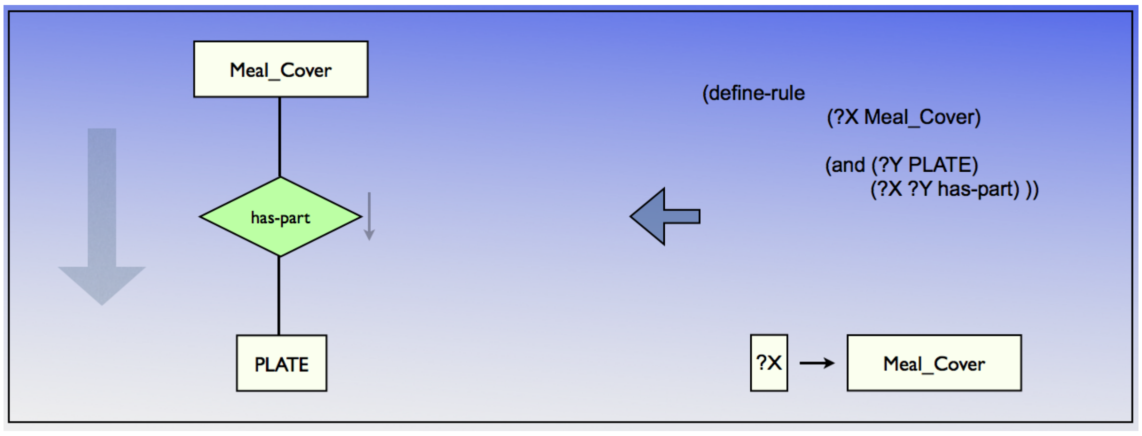


Figure 3.12: The top-down Rule for the ***Meal_Cover*** Aggregate using One Primitive Object

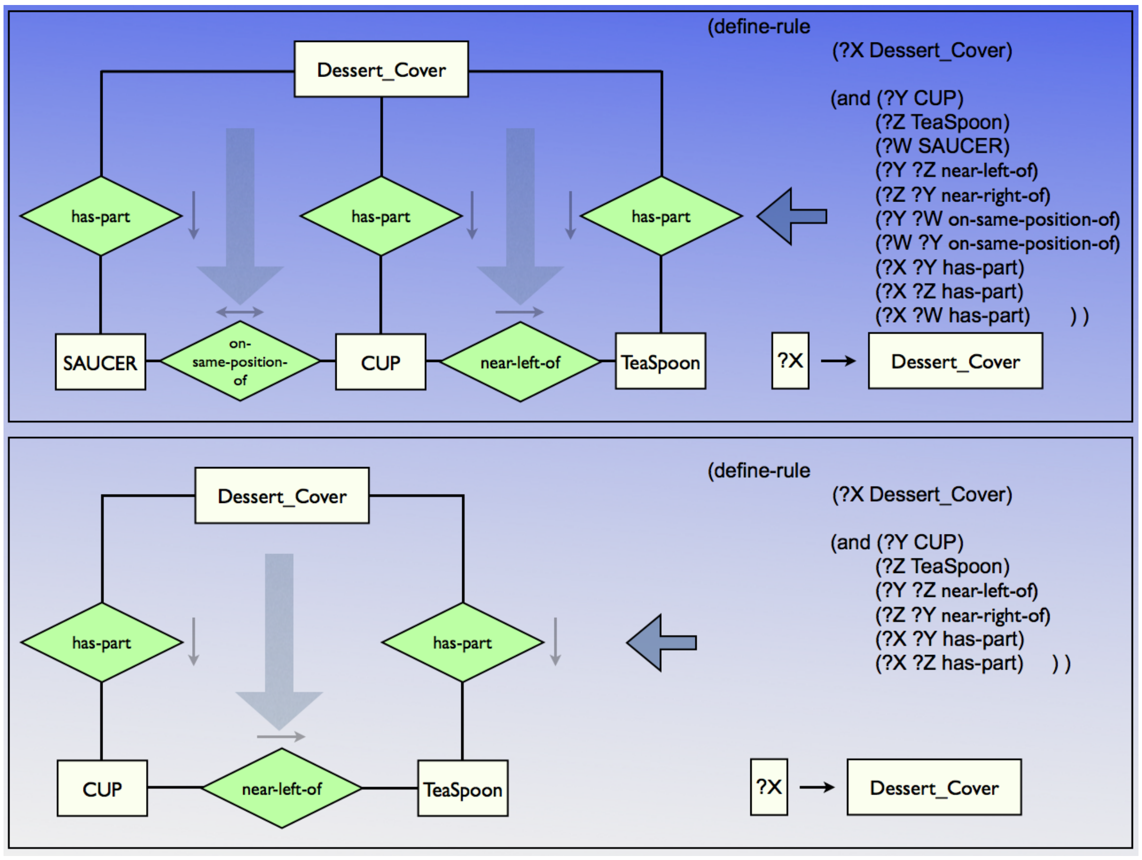
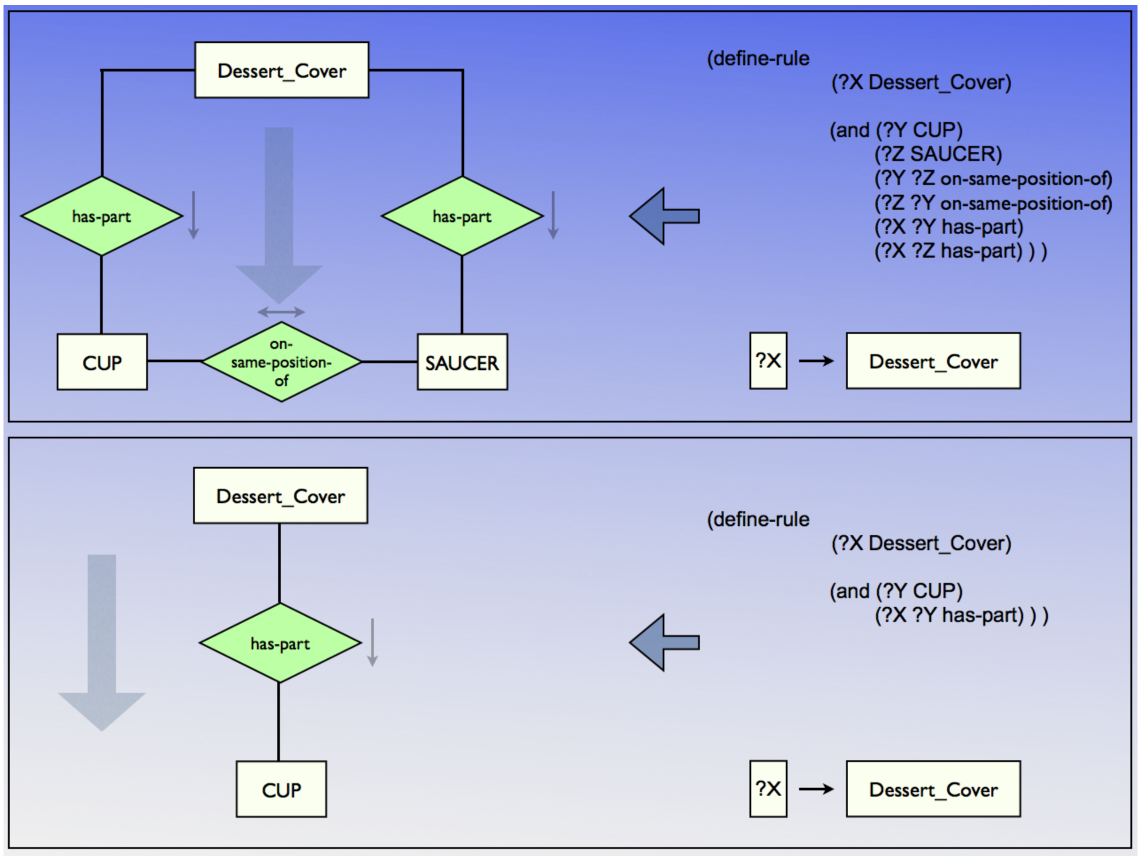


Figure 3.13: (top): The top-down Rule for the ***Dessert_Cover*** Aggregate using Three Primitive Objects – (bottom): The top-down Rule for the ***Dessert_Cover*** Aggregate using Two Primitive Objects

the lowest level are formed first. Each level has a higher priority than its higher level and a lower priority than its lower level. For the scenario of this work, this approach looks for the instances of ***Meal_Cover*** and ***Dessert_Cover*** that are in the same level; then, for the instances of ***Combined_Cover*** that are in the higher level.



```
(define-rule
        (?X Dessert_Cover)

        (and (?Y CUP)
             (?Z SAUCER)
             (?Y ?Z on-same-position-of)
             (?Z ?Y on-same-position-of)
             (?X ?Y has-part)
             (?X ?Z has-part) ) )
```

```
(define-rule
        (?X Dessert_Cover)

        (and (?Y CUP)
             (?X ?Y has-part) ) )
```

Figure 3.14: (top): The top-down Rule for the ***Dessert_Cover*** Aggregate using Two Primitive Objects – (bottom): The top-down Rule for the ***Dessert_Cover*** Aggregate using One Primitive Objects

This method forms the aggregates starting from the lowest level and ends in the highest level. If two aggregates are in the same level, the order by which they are formed is not important.

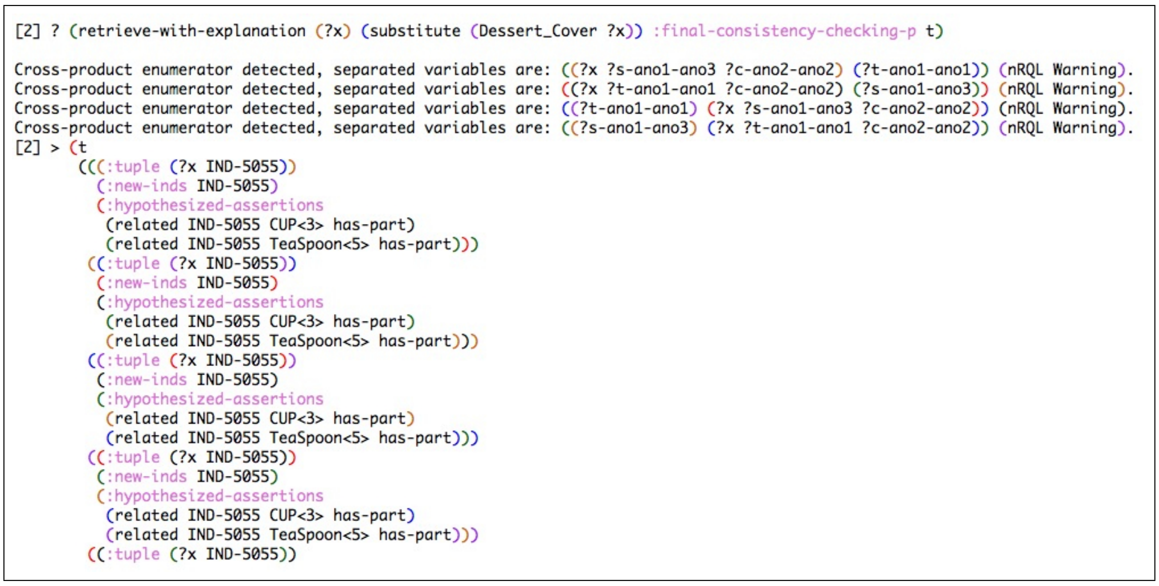
3.1.3 Difficulties of top-down Rules with RacerPro and Solutions

During the examination of the top-down rules in RacerPro, some errors and difficulties were observed. One problem was that the hypotheses, which were delivered by RacerPro according to the top-down rules, were not the ones which were expected. Another problem was that the same results were repeated several times as the outcome of the explanation command which were not always true.

Before discussing this issue with the RacerPro developers, the issues that caused this behavior of RacerPro were sought.

To find out these issues the rules were studied on paper in order to see if they match the scenario completely, and if they are able to deliver the desired results. In the next step, the focus was only on one of the aggregates and its corresponding top-down rules, for which the ***Dessert_Cover*** aggregate was chosen.

In figure 3.15, the result in the second line has been delivered after calling the ***retrieve-with-explanation*** in order to form the instances of ***Dessert_Cover***(in case of existence).



```
[2] ? (retrieve-with-explanation (?x) (substitute (Dessert_Cover ?x)) :final-consistency-checking-p t)

Cross-product enumerator detected, separated variables are: ((?x ?s-ano1-ano3 ?c-ano2-ano2) (?t-ano1-ano1)) (nRQL Warning).
Cross-product enumerator detected, separated variables are: ((?x ?t-ano1-ano1 ?c-ano2-ano2) (?s-ano1-ano3)) (nRQL Warning).
Cross-product enumerator detected, separated variables are: ((?t-ano1-ano1) (?x ?s-ano1-ano3 ?c-ano2-ano2)) (nRQL Warning).
Cross-product enumerator detected, separated variables are: ((?s-ano1-ano3) (?x ?t-ano1-ano1 ?c-ano2-ano2)) (nRQL Warning).
[2] > (t
       (((:tuple (?x IND-5055))
         (:new-inds IND-5055)
         (:hypothesized-assertions
          (related IND-5055 CUP<3> has-part)
          (related IND-5055 TeaSpoon<5> has-part)))
        ((:tuple (?x IND-5055))
         (:new-inds IND-5055)
         (:hypothesized-assertions
          (related IND-5055 CUP<3> has-part)
          (related IND-5055 TeaSpoon<5> has-part)))
        ((:tuple (?x IND-5055))
         (:new-inds IND-5055)
         (:hypothesized-assertions
          (related IND-5055 CUP<3> has-part)
          (related IND-5055 TeaSpoon<5> has-part)))
        ((:tuple (?x IND-5055))
         (:new-inds IND-5055)
         (:hypothesized-assertions
          (related IND-5055 CUP<3> has-part)
          (related IND-5055 TeaSpoon<5> has-part)))
        ((:tuple (?x IND-5055))
```

Figure 3.15: Problem with the Scoring Function of the ***retrieve-with-explanation*** Command

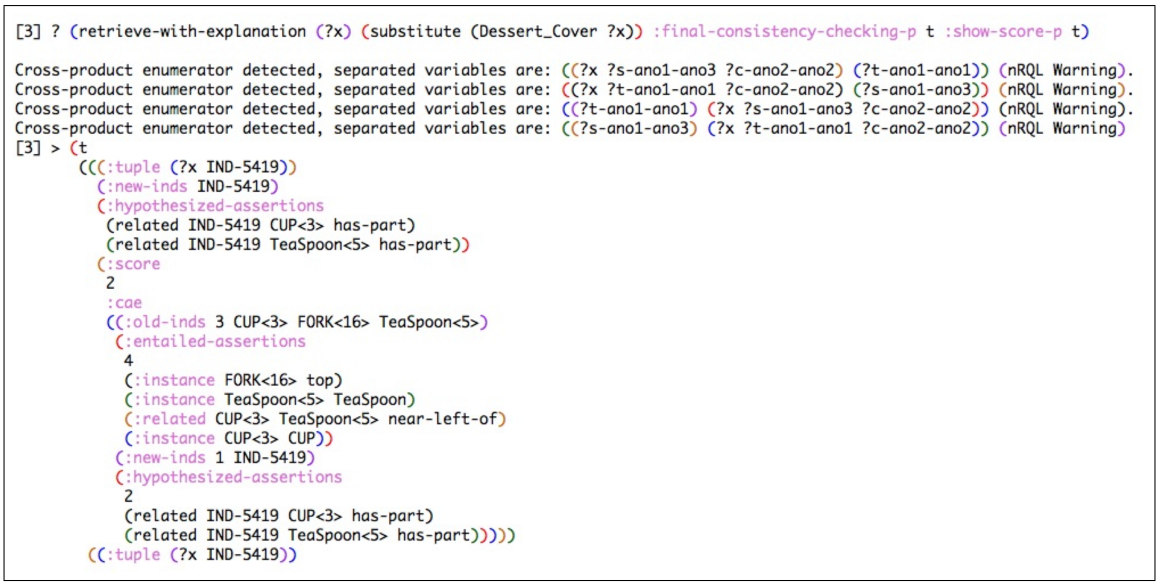
As it is shown, all the explanations look exactly the same and there is no difference between the new individuals as well as the hypothesized assertions that are suggested for the query in figure 3.15[8]

By enabling the ***:show-score-p*** feature of the ***retrieve-with-explanation*** command, it is possible to see the scoring of the results and also more details about how the explanations, such as the entailed assertions, their corresponding scores and the old individuals that are used in the explanation, are constructed. These terms are illustrated in figure 3.16.

The explanations shown in figure 3.16 suggest that an instance of ***CUP*** and ***TeaSpoon*** should be grouped under an instance of ***Dessert_Cover***. This explanation is correct according to the scene.

All of the assertions that are shown under entailed assertions are also correct; however, in the first line of these assertions for the first explanation, the individual ***FORK<16>***, which is an instance of ***FORK***, has appeared under the entailed assertions as ***<top>*** although, as it was shown earlier, the concept ***FORK*** has nothing to do with the concept ***Dessert_Cover***, and it was not mentioned in the

[8]The warning messages were ignored during the study of the cause of the issues at first. These warning messages are addressed later in this Section.



```
[3] ? (retrieve-with-explanation (?x) (substitute (Dessert_Cover ?x)) :final-consistency-checking-p t :show-score-p t)

Cross-product enumerator detected, separated variables are: ((?x ?s-ano1-ano3 ?c-ano2-ano2) (?t-ano1-ano1)) (nRQL Warning).
Cross-product enumerator detected, separated variables are: ((?x ?t-ano1-ano1 ?c-ano2-ano2) (?s-ano1-ano3)) (nRQL Warning).
Cross-product enumerator detected, separated variables are: ((?t-ano1-ano1) (?x ?s-ano1-ano3 ?c-ano2-ano2)) (nRQL Warning).
Cross-product enumerator detected, separated variables are: ((?s-ano1-ano3) (?x ?t-ano1-ano1 ?c-ano2-ano2)) (nRQL Warning)
[3] > (t
       (((:tuple (?x IND-5419))
         (:new-inds IND-5419)
         (:hypothesized-assertions
          (related IND-5419 CUP<3> has-part)
          (related IND-5419 TeaSpoon<5> has-part))
         (:score
          2
          :cae
          ((:old-inds 3 CUP<3> FORK<16> TeaSpoon<5>)
           (:entailed-assertions
            4
            (:instance FORK<16> top)
            (:instance TeaSpoon<5> TeaSpoon)
            (:related CUP<3> TeaSpoon<5> near-left-of)
            (:instance CUP<3> CUP))
           (:new-inds 1 IND-5419)
           (:hypothesized-assertions
            2
            (related IND-5419 CUP<3> has-part)
            (related IND-5419 TeaSpoon<5> has-part)))))
        ((:tuple (?x IND-5419))
```

Figure 3.16: Undesired Top Conjunctions in the Explanation

corresponding rules for dessert covers neither[9].

This issue is also observed in other explanations where an instance of ***CUP*** and an instance of ***SAUCER*** have been grouped correctly under an instance of ***Dessert_Cover*** as shown in figure 3.17.

In figure 3.17, the individual ***KNIFE<14>*** that is an instance of *KNIFE* has been appeared under the entailed assertions, although there is no relation between this individual (its corresponding concept) and the concept ***Dessert_Cover***.

The reason why the same results are suggested by RacerPro is the top conjunctions. Each result contains the same explanations but different top conjunctions under the entailed assertions.

It was attempted to eliminate these top conjunctions in the explanations by changing the rule definitions. To do so, ***not*** terms were added into the rules to avoid the unrelated objects to occur in the explanations. Unfortunately, this effort was not successful.

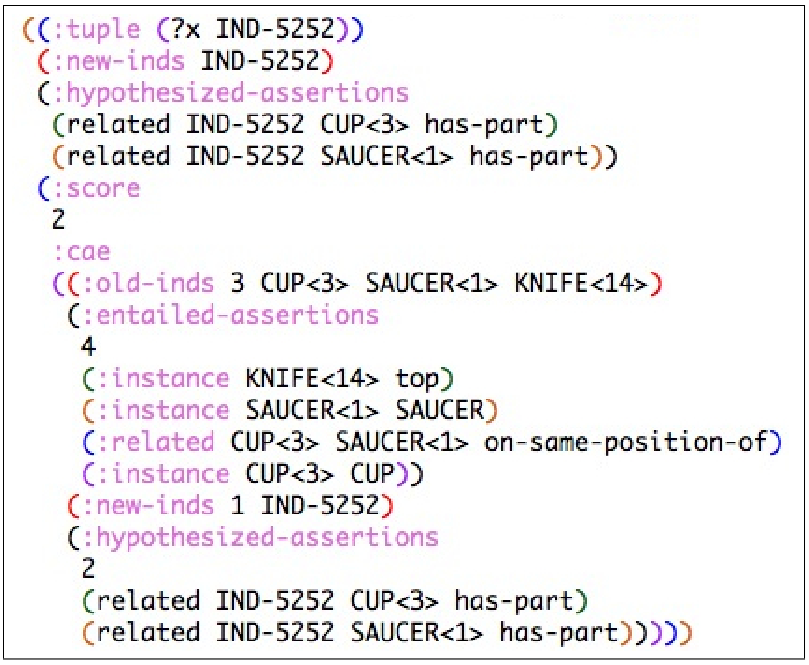
Enabling the ***only-best-p*** option of the ***retrieve-with-explanation*** command was not helpful because all the explanations that were repeated got the same score again.

Despite the efforts mentioned above and some further efforts, the issue remained unchanged, which resulted in some difficulties for the framework by preventing it to distinguish between the correct explanations and the ones that were suggested as a consequence of the bug in RacerPro. At this point, it was decided to inform the development team of RacerPro about this issue.

Their respond to this issue was that the abduction rules in RacerPro also consider the individuals that already exist in the ABox as possible bonds for the variables; therefore, the top conjunctions are caused by expansion of these rules.

To illustrate this incident, the following top-down rules for the ***Dessert_Cover***

[9]The line (:instance FORK<16> top) is true by itself.



```
((:tuple (?x IND-5252))
 (:new-inds IND-5252)
 (:hypothesized-assertions
  (related IND-5252 CUP<3> has-part)
  (related IND-5252 SAUCER<1> has-part))
 (:score
  2
  :cae
  ((:old-inds 3 CUP<3> SAUCER<1> KNIFE<14>)
   (:entailed-assertions
    4
    (:instance KNIFE<14> top)
    (:instance SAUCER<1> SAUCER)
    (:related CUP<3> SAUCER<1> on-same-position-of)
    (:instance CUP<3> CUP))
   (:new-inds 1 IND-5252)
   (:hypothesized-assertions
    2
    (related IND-5252 CUP<3> has-part)
    (related IND-5252 SAUCER<1> has-part)))))
```

Figure 3.17: Undesired Top Conjunctions in the Explanation

are considered:

```
(define-rule (?d Dessert_Cover)
             (and (?c CUP) (?t TeaSpoon) (?c ?t near-left-of)
                  (?d ?c has-part) (?d ?t has-part)))

(define-rule (?d Dessert_Cover)
             (and (?c CUP) (?s SAUCER) (?c ?s on-same-position-of)
                  (?d ?c has-part) (?d ?s has-part)))
```

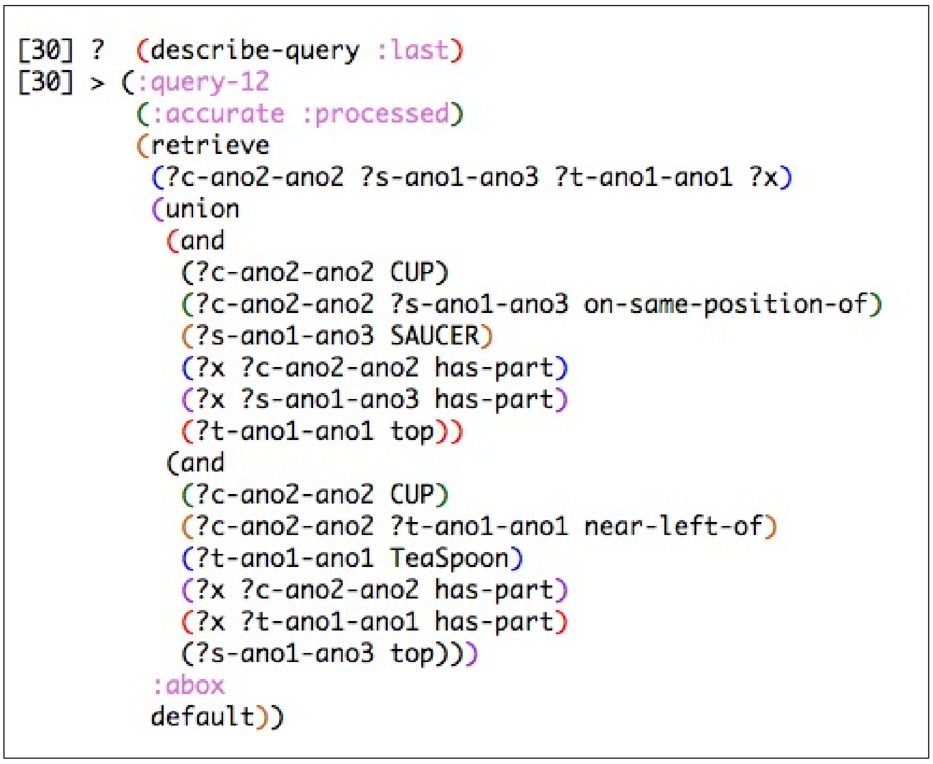
The description of these rules in RacerPro is shown in figure 3.18. Both rules are expanded and described in the form of a union query. For the first, rule the term **(?t-ano1-ano1 top)** and for the second, rule the term **(?t-ano1-ano3 top)** have been added to their descriptions.

The developers of RacerPro said that these terms are added in the descriptions so that the result of the union queries get the same arity although this is not necessary for the queries; therefore, this issue could be considered as a bug.

The first solution that they gave was trying to rewrite the rules, in the following form:

```
(define-rule (?d Dessert_Cover)
             (and (?c CUP) (?t TeaSpoon) (?c ?t near-left-of)
                  (?d ?c has-part) (?d ?t has-part)))

(define-rule (?d Dessert_Cover)
             (and (?c CUP) (?t SAUCER) (?c ?t on-same-position-of)
```



```
[30] ?  (describe-query :last)
[30] > (:query-12
        (:accurate :processed)
        (retrieve
         (?c-ano2-ano2 ?s-ano1-ano3 ?t-ano1-ano1 ?x)
         (union
          (and
           (?c-ano2-ano2 CUP)
           (?c-ano2-ano2 ?s-ano1-ano3 on-same-position-of)
           (?s-ano1-ano3 SAUCER)
           (?x ?c-ano2-ano2 has-part)
           (?x ?s-ano1-ano3 has-part)
           (?t-ano1-ano1 top))
          (and
           (?c-ano2-ano2 CUP)
           (?c-ano2-ano2 ?t-ano1-ano1 near-left-of)
           (?t-ano1-ano1 TeaSpoon)
           (?x ?c-ano2-ano2 has-part)
           (?x ?t-ano1-ano1 has-part)
           (?s-ano1-ano3 top)))
         :abox
         default))
```

Figure 3.18: Undesired Top Conjunctions in the Description of the Rules

```
                (?d ?c has-part) (?d ?t has-part)))
```

In the rewritten format of the rules the, same variable names are used for the instances of ***TeaSpoon*** and ***SAUCER***. The descriptions of these rules in the form of a union query are illustrated in figure 3.19.

As it is shown in figure 3.19, there are no terms of top conjunctions in the descriptions. The result of calling ***retrieve-with-explanation*** command for ***Dessert_Cover*** is shown in figure 3.20. As it can be seen, the redundant results do not occur anymore and the correct results are outputted.

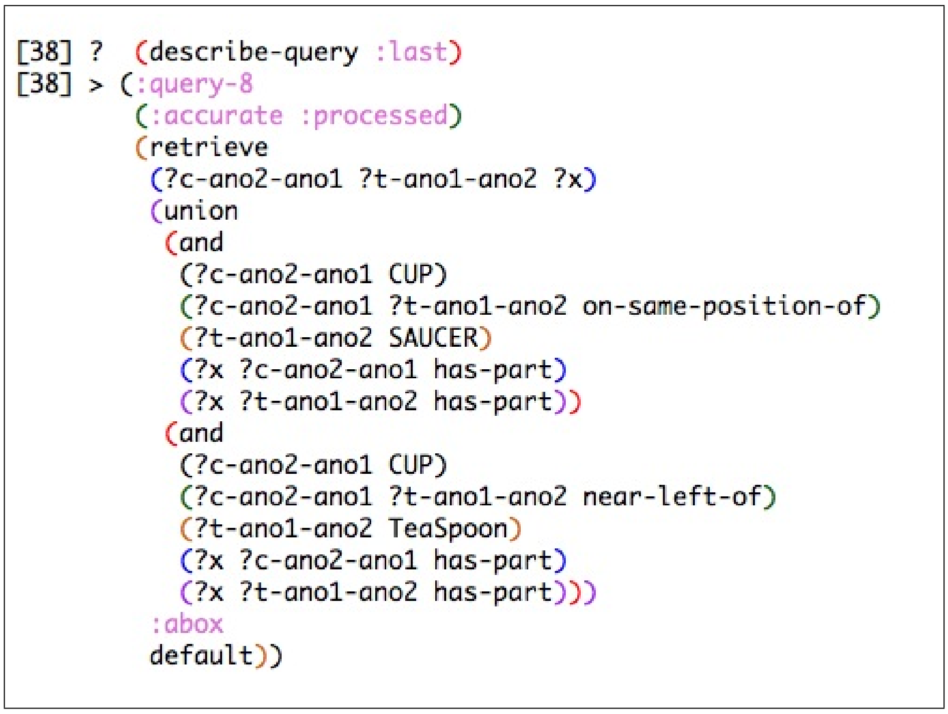
It should be noted that this bug was fixed quickly and the issue with the different variables was solved[10]. However, after adding the following rule, the same disturbing issue occurred again. Figure 3.21 shows the reoccurrence of the top conjunctions in the descriptions of the queries.

```
(define-rule (?d Dessert_Cover)
             (and (?c CUP) (?t TeaSpoon) (?s SAUCER)
                  (?c ?t near-left-of) (?c ?s on-same-position-of)
                  (?d ?c has-part) (?d ?s has-part) (?d ?t has-part) ))
```

The team of developers of RacerPro were informed again about this issue. Meanwhile, other solutions were tested to deal with this problem.

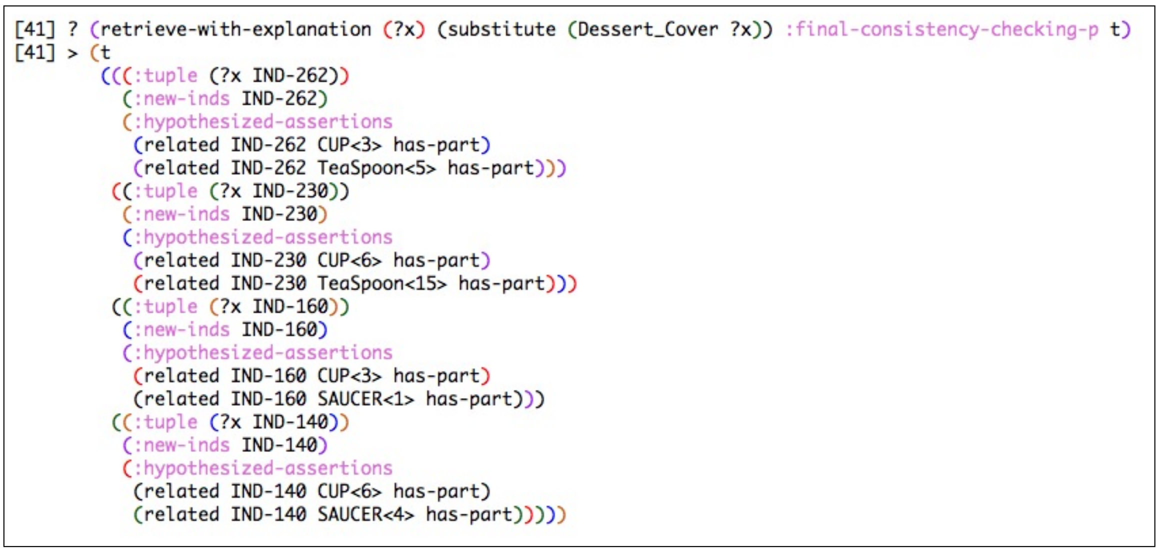
One of the ideas that was tested was rewriting the rules so that the variable

[10]The error messages were also eliminated at this point.



```
[38] ?  (describe-query :last)
[38] > (:query-8
        (:accurate :processed)
        (retrieve
         (?c-ano2-ano1 ?t-ano1-ano2 ?x)
         (union
          (and
           (?c-ano2-ano1 CUP)
           (?c-ano2-ano1 ?t-ano1-ano2 on-same-position-of)
           (?t-ano1-ano2 SAUCER)
           (?x ?c-ano2-ano1 has-part)
           (?x ?t-ano1-ano2 has-part))
          (and
           (?c-ano2-ano1 CUP)
           (?c-ano2-ano1 ?t-ano1-ano2 near-left-of)
           (?t-ano1-ano2 TeaSpoon)
           (?x ?c-ano2-ano1 has-part)
           (?x ?t-ano1-ano2 has-part)))
         :abox
         default))
```

Figure 3.19: Result of defining the Rules with the same Variable Names



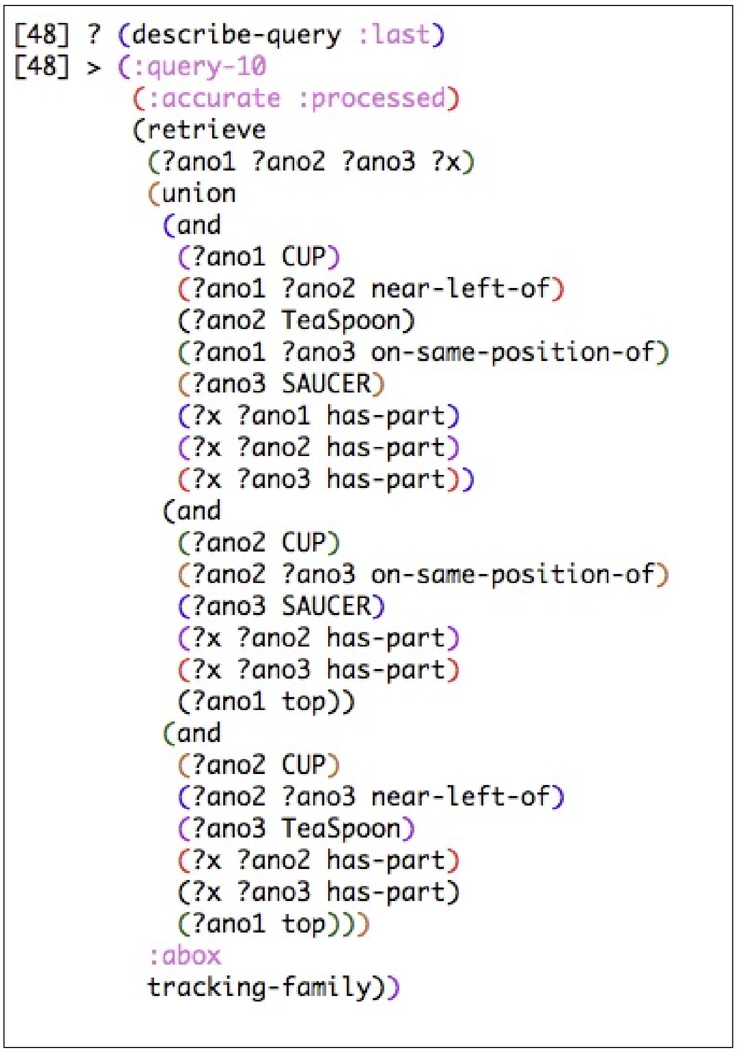
```
[41] ? (retrieve-with-explanation (?x) (substitute (Dessert_Cover ?x)) :final-consistency-checking-p t)
[41] > (t
        (((:tuple (?x IND-262))
          (:new-inds IND-262)
          (:hypothesized-assertions
           (related IND-262 CUP<3> has-part)
           (related IND-262 TeaSpoon<5> has-part)))
         ((:tuple (?x IND-230))
          (:new-inds IND-230)
          (:hypothesized-assertions
           (related IND-230 CUP<6> has-part)
           (related IND-230 TeaSpoon<15> has-part)))
         ((:tuple (?x IND-160))
          (:new-inds IND-160)
          (:hypothesized-assertions
           (related IND-160 CUP<3> has-part)
           (related IND-160 SAUCER<1> has-part)))
         ((:tuple (?x IND-140))
          (:new-inds IND-140)
          (:hypothesized-assertions
           (related IND-140 CUP<6> has-part)
           (related IND-140 SAUCER<4> has-part)))))
```

Figure 3.20: Correct Explanation for the Rules that have the same Number of Tuples

names in one rule were not present in other rules. This solution failed because the issue remained unchanged in the explanations.

The other idea was implementing a parser so that the results that contain a top conjunction were going to be eliminated by the parser. This idea could be an optimal solution, but it was not implemented due to time constraints.

Another solution was deactivating all the rules, then activating them one by one, calling the ***retrieve-with-explanation*** command according to the active rule, analyzing and asserting the explanations and finally deactivating the rule. This procedure should be applied to all the rules. The final result should be the same result as if the disturbing issue does not exist and all the rules are active. However, deactivation/activation of the rules should be first implemented in RacerPro because these options were not present in any versions of RacerPro.



```
[48] ? (describe-query :last)
[48] > (:query-10
        (:accurate :processed)
        (retrieve
         (?ano1 ?ano2 ?ano3 ?x)
         (union
          (and
           (?ano1 CUP)
           (?ano1 ?ano2 near-left-of)
           (?ano2 TeaSpoon)
           (?ano1 ?ano3 on-same-position-of)
           (?ano3 SAUCER)
           (?x ?ano1 has-part)
           (?x ?ano2 has-part)
           (?x ?ano3 has-part))
          (and
           (?ano2 CUP)
           (?ano2 ?ano3 on-same-position-of)
           (?ano3 SAUCER)
           (?x ?ano2 has-part)
           (?x ?ano3 has-part)
           (?ano1 top))
          (and
           (?ano2 CUP)
           (?ano2 ?ano3 near-left-of)
           (?ano3 TeaSpoon)
           (?x ?ano2 has-part)
           (?x ?ano3 has-part)
           (?ano1 top)))
         :abox
         tracking-family))
```

Figure 3.21: Reappearance of the Top Conjunctions in the Description of the Rules

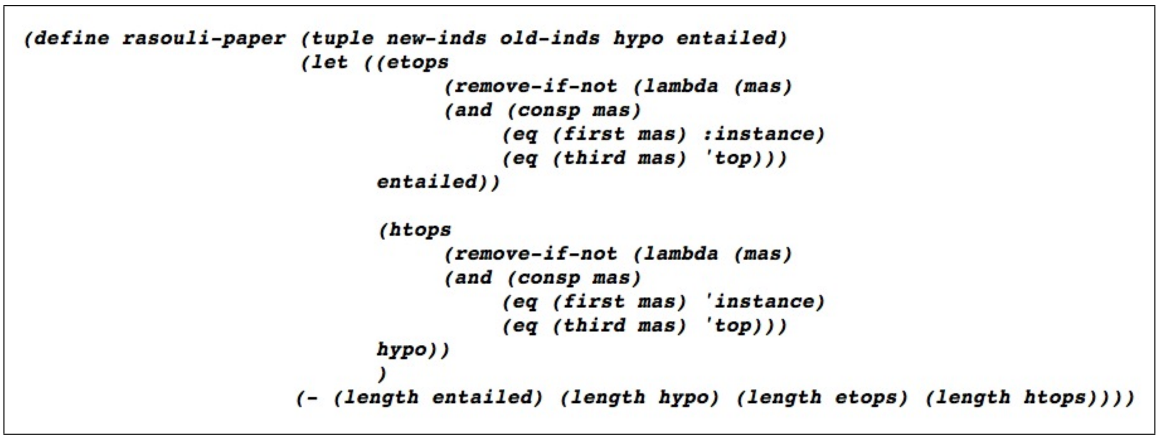
In the meantime, one of the supervisors of RacerPro suggested to redefine the scoring function so that the explanations that contain the top conjunctions under the entailed assertions and the hypothesized assertions get a lower score at the end; as a result, they would automatically be eliminated by the ***only-best-p*** feature of the ***retrieve-with-explanation*** command.

The suggested scoring function for this case is:

```
Score = (# entailed assertions) - (# hypothesized-assertions)
          - (# TOP-conjunctions under the entailed assertions)
          - (# TOP-conjunctions under the hypothesized assertions)
```

Each top conjunction in the result of an explanation has a negative effect on the score of this explanation. This is an advantage for the explanations that do not contain any top conjunctions.

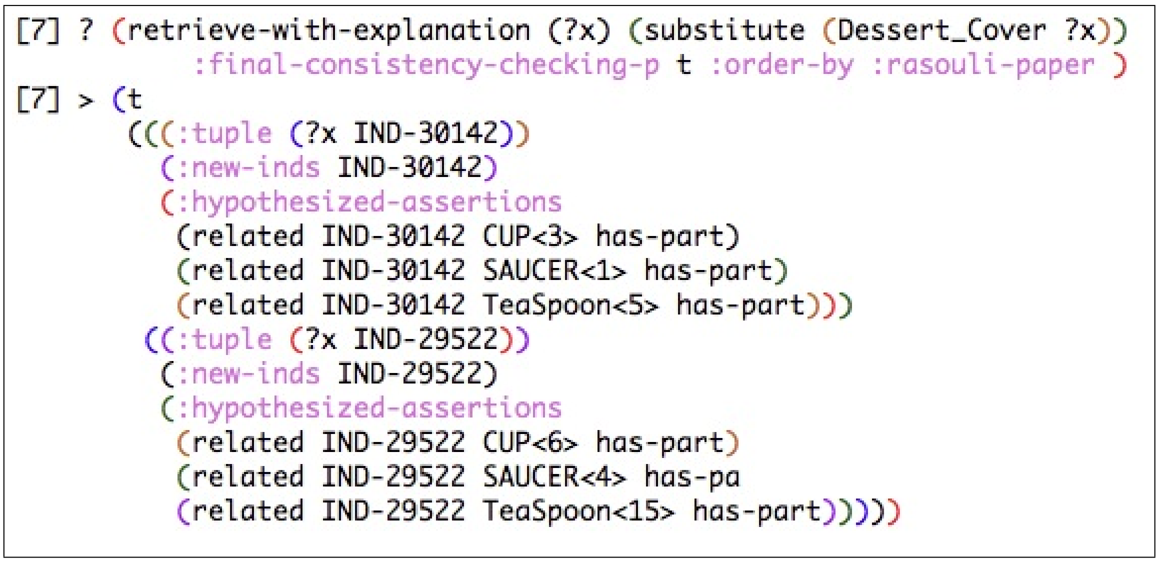
The new scoring function should be implemented in LISP. With the help of the supervisor, the scoring function called ***rasouli-paper-fn*** was implemented in LISP as follows:



```
(define rasouli-paper (tuple new-inds old-inds hypo entailed)
                      (let ((etops
                                 (remove-if-not (lambda (mas)
                                 (and (consp mas)
                                      (eq (first mas) :instance)
                                      (eq (third mas) 'top)))
                            entailed))

                            (htops
                                 (remove-if-not (lambda (mas)
                                 (and (consp mas)
                                      (eq (first mas) 'instance)
                                      (eq (third mas) 'top)))
                            hypo))
                            )
                     (- (length entailed) (length hypo) (length etops) (length htops))))
```

Figure 3.22: ***rasouli-paper*** Scoring Function in LISP



```
[7] ? (retrieve-with-explanation (?x) (substitute (Dessert_Cover ?x))
          :final-consistency-checking-p t :order-by :rasouli-paper )
[7] > (t
       (((:tuple (?x IND-30142))
         (:new-inds IND-30142)
         (:hypothesized-assertions
          (related IND-30142 CUP<3> has-part)
          (related IND-30142 SAUCER<1> has-part)
          (related IND-30142 TeaSpoon<5> has-part)))
        ((:tuple (?x IND-29522))
         (:new-inds IND-29522)
         (:hypothesized-assertions
          (related IND-29522 CUP<6> has-part)
          (related IND-29522 SAUCER<4> has-pa
          (related IND-29522 TeaSpoon<15> has-part)))))
```

Figure 3.23: Result of using the ***rasouli-paper*** Scoring Function

etops contains the list of top conjunctions under the entailed assertions, and ***htops*** contains the list of top conjunctions under the hypothesized assertions.

If the length of each list (number of the top conjunctions) is subtracted from the length of the list of the entailed assertions (number of entailed assertions) following by the length of the list of the hypothesized assertions (number of hypothesized assertions), the correct score for each explanation will be calculated.

By using the ***order-by*** feature followed by the name of the new scoring function (***rasouli-paper-fn***), it is possible to run the ***retrieve-with-explanation*** command and choose the explanations based on their scores regarding the new scoring function.

Figure 3.23 shows the correct behavior of the reasoner for the instances of ***Dessert_Cover*** in the scene.

3.2 Strategies

In the previous Section, the question **how** the aggregates are formed using the rules has been answered, but the question **when** these rules should be applied has been remained unanswered.

The initial solution that could be used is to fire the rules for each frame of the movie. This strategy may be the simplest solution, but it is not efficient at all because it takes a long time for the framework to reach the last scene of the clip and the user should wait for a long time to get the scene interpretation results.

Secondly, the information that each frame delivers is highly redundant compared to the previous and next frames; therefore, the amount of the data that is gained in each frame is mostly the same for the frames that are close to the current scene that is being interpreted.

This solution also matches the normal behavior of a human being when he is watching a movie and trying to analyze and make hypotheses for that movie. No one can interpret a whole movie by watching just a few frames of that movie (except detecting the primitive objects that are available in the frames which are not important initially and belong to the low-level data).

In the normal behavior, a human viewer can come to some conclusions about the movie in specific points in time, which depends surely on the person. The interpretation times may vary from one person to another.

The moment that a human tries to interpret a scene of a movie based on the observation that he has made so far, is called a **Synchronization Point**.

Accordingly, the frame that the user tries to interpret is named a **Keyframe**. There are different ways to categorize a frame under the list of the keyframes. The chosen approach is to specify if the primitive objects in the scene are moving or not. If no moving object has been found in the scene and all the objects that are observed are immobile, this frame could be added to the list of the keyframes. Since the situation where all the objects in the scene are constant may occur in many adjacent frames, only one of these frames should be selected and added to the collection of the keyframes. In this work, a keyframe is selected from a list of potential keyframes that are next to each other in the following manner:

If ***a*** is the number of the first frame among the potential keyframes which are all next to each other and ***b*** is the number of the last frame among them, the frame that will be chosen and added to the list of the keyframes has the number ***c*** and is calculated using this formula:

$$c = \lceil \frac{a+b}{2} \rceil^{11} \tag{3.1}$$

Each person uses its own method to interpret a keyframe. These methods could be implemented in the form of different **Strategies** in a machine.

Since scene interpretation of machines should work as correctly and efficiently as humans, the same terms such as *Synchronization Point*, *Keyframe* and *Strategies* that a person uses should be implemented in the machines.

For identifying the keyframes, the tracking data that the image processing tools deliver could be used. As shown in Section 2.2, to specify if a primitive object is moving in a frame or not, there is a term for each frame, which appears as in the tracking data: ***(PV MOVE (0 or 1))***. If ***MOVE*** is set to 1, it means that the object is moving, and if it is set to 0, it means that the object is standing still in the scene.

If the ***MOVE*** properties of the objects are considered as binary variables and if the result of disjunction of these binary variables (move properties) of the objects in a frame is 0, it means that none of the objects in the scene is moving. Otherwise, if the result of disjunction is 1, it means that at least one of the objects is moving.

In order to choose a keyframe, the frames are marked in the following:

When a frame with the disjunction result of 0 for the move properties of the existing objects is reached, the number of this frame is marked as the beginning frame a. The disjunction operation continues until the disjunction result of a frame is changed from 0 to 1. The number of this frame is marked as b. By using formula 3.1, a keyframe with the number c is specified and added to the list of the keyframes.

It should be noted that the first frame and the last frame of the clip will also be added to the list of the keyframes, even if moving objects are detected in the scene of these two frames.

After the keyframes are identified and added to the list of the keyframes, a strategy is needed to process them. In this work, three different strategies are introduced. While working with the framework, the user can change the strategy, whenever he wishes.

3.2.1 Optimist Strategy

An optimist person is someone who is confident and hopeful that the outcome and the result of an operation is successful. In other words, he thinks positive that what he sees and believes, remains true and unchanged. For this person, the observations, he has made so far are facts, and he expects that they will not change in the future.

If a machine is considered an optimist, the same behavior that an optimist human being has, should be implemented for this machine. In the context of this work, an optimist machine should consider the primitive objects, which have been identified in the scene of a keyframe, to be positioned in the same location until the last frame of the clip, and that they would not change their locations.

[11] If the result is not a natural number, the ceiling of the result will be taken, because it is closer to the end of the clip.

For example, if an instance of ***Spoon*** is the nearest spoon on the right side of an instance of ***PLATE*** in the current keyframe, the optimist strategy assumes that this spoon will remain on the right side of the plate until the last frame of the clip has been reached.

Furthermore, it will assume that no other instances of ***Spoon*** will be put on the right side of the instance of ***PLATE***, between the plate and the initial spoon; otherwise, the initial spoon is not the nearest spoon on the right side of the plate.

The result of this behavior is that the machine groups the instance of ***PLATE*** and the instance of ***Spoon*** under the aggregate ***Meal_Cover*** and adds the corresponding instances and role assertions into the ABox.

3.2.2 Pessimist Strategy

A pessimist person is someone who has the tendency that the worst aspect of a thing could happen. This person always considers the worst case scenario. This means that he is not confident in what he has seen and observed so far; thus, he remains restrained when dealing with an issue.

He may change his behavior when the end point of a story has been reached; then, he is ready to express his views and interpretation about the whole case.

A pessimist machine ignores the observations in the scene and would not try to explain the evidences that has been discovered so far until the last keyframe of the clip has been reached. This means that the machine assumes the position of the primitive objects that have been discovered and identified so far could be changed and that the scene of the last keyframe could differ completely from the current keyframe.

For example, if an instance of ***CUP*** is located on an instance of ***SAUCER***, the machine does not group them under the aggregate *Dessert_Cover* because it assumes that the instance of ***CUP*** could be removed from the instance of ***SAUCER*** or even from the scene or could be put on another instance of ***SAUCER*** that would arrive later in the scene.

This machine does not make any high-level interpretation until the last keyframe has been reached. In other words, no high-level information will be asserted until the end of the clip.

3.2.3 Realist Strategy

A person who thinks realistic is someone who observes everything as it is. He has the attitude to accept the current situation while at the same time he thinks that the current conditions and situations could change in the future; therefore, he is ready to deal with the current situation as well as the situations in the future that may differ from the current one.

The realist machine in the context of this work is a machine that explains the observations that seem accurate in the current situation and ignores the observations that are highly capable of change in the future.

As an example, the situation where an instance of ***KNIFE*** is the nearest knife among all instances of ***KNIFE*** in the scene, that are next to an instance of ***PLATE***, is considered. In this situation, there are two possibilities for a realist agent:

1- If there is space for an instance of ***KNIFE*** between the instance of ***PLATE*** and the nearest instance of ***KNIFE*** on the right side of the instance of ***PLATE***, the agent remains restrained and does not make any interpretation regarding the current situation.

2- If the gap between the instance of ***PLATE*** and the nearest instance of ***KNIFE*** on the right side of the instance of ***PLATE*** is so small that no other instances of ***KNIFE*** are able to be placed on the right side of the instance of ***PLATE*** closer to this instance, it means that the initial nearest instance of ***KNIFE*** on the right side of the instance ***PLATE*** should be removed first so that a new instance of ***KNIFE*** could be put next to the instance of ***PLATE***.

In other words, the current situation seems to stay unchanged; therefore, the realist agent tries to make high-level interpretation for the current situation by grouping the instance of ***KNIFE*** and the instance of ***PLATE*** under the ***Meal_Cover*** aggregate.

Chapter 4

Implementation

For showing and illustrating the result of the interpretation of the table cloth scenario, a framework using the JAVA programming language and jRacer is implemented.

The major task of this framework is showing the keyframes of the scene, marking the objects that have been identified by the image processing tools with their ID and outputting the high-level interpretations in different fields.

In this Chapter, an overview of the structure of the framework will be discussed first. The implementation of each package of this structure will be shown afterwards.

4.1 Overview

The framework has many different tasks, which could be summarized as:

1- Analyzing the tracking data
2- Choosing the keyframes
3- Providing the graphical interface
4- Communicating with RacerPro using jRacer
5- Updating the ABox of the scene
6- High-level interpretation of the scene based on the chosen rules
and strategies by the user
7-Showing the results of the interpretation on the graphical interface

Each of these tasks is performed by one or more packages of the software. For implementing the framework, Eclipse Galileo IDE, RacerPro and jRacer are used. For the graphical part, JSwing[1] is used. The main programming language used is Java.

In the next Sections, the structure of the GUI and the framework will be described in terms of software packages that are implemented.

[1]A Java GUI widget toolkit implemented by Sun Microsystems for Java developers as an API to be used for implementing GUIs.

4.2 GUI of the Interpretation Framework

Understanding the results of the interpretation of a frame in a text format is a hard task; specially, when the interpretation rules and methods should be tested first in order to verify their correctness. The time-consuming task of tracing the results by hand is another problem. Without a visual demonstration of the results, it would also hard to view the resulting groups and instances of the aggregates.

To overcome these problems, a graphical user interface(GUI) is designed and implemented. The images of the keyframes, the primitive objects and the resulting aggregates can be shown on the GUI. It is also possible to choose between different strategies and groups of rules for the interpretation steps. Finally, the time that the framework needs to explain a scene is shown on the GUI when the interpretation of a scene is finished.

Figure 4.1 shows the GUI of the framework. The most important parts of the interface are marked by letters. The description of each part is explained in the following lines:

A: Number of the frame that is being interpreted.

B: The image of the frame.

C: A primitive object with its ID. The **blue** color of the ID indicates that the object has been identified in the previous keyframes.

D: A primitive object with its ID. The **red** color of the ID indicates that the object has been identified in the current keyframes.

E: Instance of ***Meal_Cover*** with its bounding box drawn with **solid** lines. The objects inside the bounding box are the parts of the cover.

F: Instance of ***Dessert_Cover*** with its bounding box drawn with **solid** lines. The objects inside the bounding box are the parts of the cover.

G: Instance of ***Combined_Cover*** with its bounding box drawn with **dotted** lines. The objects inside the bounding box are the parts of the cover.

H: Slider showing the position of the keyframe in the clip.

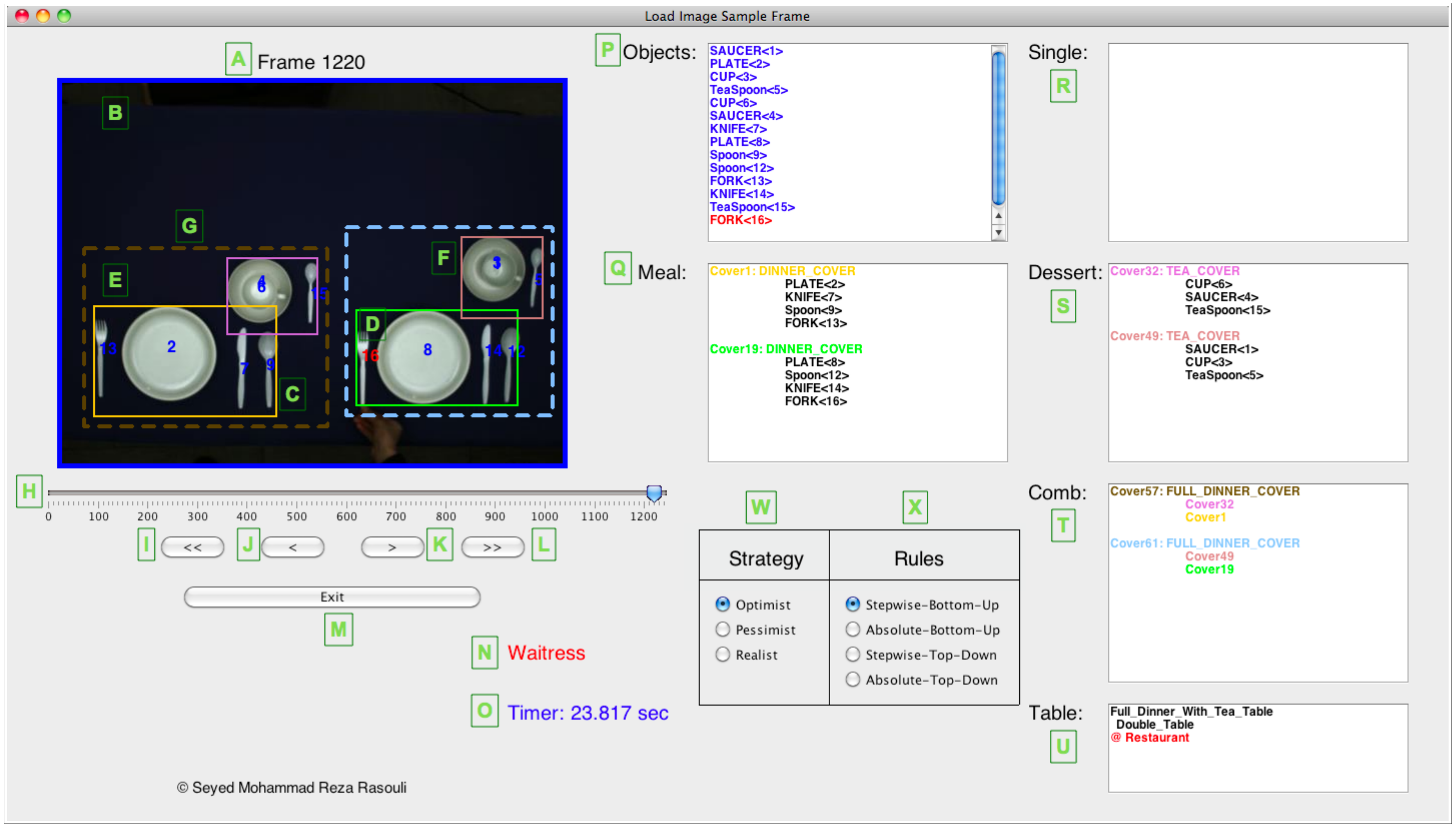
I: First Frame Button - Going to the **first** frame.

J: Rewind Button - Going to the previous keyframe.

K: Forward Button - Going to the next keyframe.

L: Last Frame Button - Going to the **last** frame.

M: Exit Button - Finishing the interpretation.



Frame 1220

Objects:
SAUCER<1>
PLATE<2>
CUP<3>
TeaSpoon<5>
CUP<6>
SAUCER<4>
KNIFE<7>
PLATE<8>
Spoon<9>
Spoon<12>
FORK<13>
KNIFE<14>
TeaSpoon<15>
FORK<16>

Single:

Meal:
Cover1: DINNER_COVER
PLATE<2>
KNIFE<7>
Spoon<9>
FORK<13>

Cover19: DINNER_COVER
PLATE<8>
Spoon<12>
KNIFE<14>
FORK<16>

Dessert:
Cover32: TEA_COVER
CUP<6>
SAUCER<4>
TeaSpoon<15>

Cover49: TEA_COVER
SAUCER<1>
CUP<3>
TeaSpoon<5>

Comb:
Cover57: FULL_DINNER_COVER
Cover32
Cover1

Cover61: FULL_DINNER_COVER
Cover49
Cover19

Table:
Full_Dinner_With_Tea_Table
Double_Table
@ Restaurant

Strategy: [✓ Optimist] Pessimist Realist

Rules: [✓ Stepwise-Bottom-Up] Absolute-Bottom-Up Stepwise-Top-Down Absolute-Top-Down

Waitress

Timer: 23.817 sec

© Seyed Mohammad Reza Rasouli

Figure 4.1: GUI of the Framework

N: Indicating if a waitress has been seen in the clip.

O: Interpretation time - This is the time that it takes for the framework to load the ABox, to interpret and to add the new assertions to the loaded ABox.

P: The objects that have been identified so far. The objects written in red are the objects discovered in the current keyframe and the objects written in blue are the objects found in the previous keyframes.

Q: Instances of ***Meal_Cover***. The objects belonging each cover are written in black.

R: Instances of single covers. These are the covers that do not belong to any instances of ***Combined_Cover***.

S: Instances of ***Dessert_Cover***. The objects belonging each cover are written in black.

T: Instances of ***Combined_Cover***. The covers that form a combined cover are written in their corresponding color.

U: Instance of the table. The type of the table and whether or not the table is in a restaurant or at home is shown here.

W: Strategies Column - To choose a strategy for the interpretation of the scene. This strategy could be changed by the user during the session.

X: Rules Column - To select a rule for the interpretation of the scene. The rule could be changed by the user during the session.

The GUI starts by showing and interpreting the first frame [2] ; then, it shows the results of the interpretation of this frame on the screen followed by the interpretation time.

A user can change the rules and strategies or go to the next or previous keyframes. The last frame [2] of the clip is the last frame that is going to be shown on the screen.

It should be noted that for each combination of the strategies and rules, a different timer is set. Therefore, the user can see which approach is more ideal for the interpretation of the frame, although this depends on the content of the media document and the position of the current keyframe in the document[3].

[2]Not necessarily a **keyframe**.

[3]More details regarding the verification and validation of the framework and GUI are discussed in Chapter 6.

4.3 Packages

The framework is consisted of six packages. The structure of the framework and the classes of each package and the relations between them are shown in figure 4.2.

trackingDataParser package was the first package that is implemented to parse the tracking data file and transform the tracking data into a structure that is easier for other packages to use.

For illustration of the data and the frames of the clip, the package ***graphics*** has been implemented, which was mostly based on Java Swing.

In the package ***rules***, the class for each group of rules is defined, which determines how the framework should work in different situations based on the rules that have been selected by the user.

The management of the frames is done by the ***frameManagement*** package. This package also updates the information that is needed for the ***graphics*** package to update the GUI of the framework.

To communicate with the RacerPro software the package ***racerInterface*** is used, which includes the classes in the jRacer package provided by the Racer Systems company.

Finally, the management of all the frames and the classes that include the necessary information about how to handle the objects and frames of the scenes is done by ***main*** package. This package also contains a class that is responsible for the timing of the scene interpretation and registering the results.

4.3.1 Package *trackingDataParser*

This package was implemented for representing the tracking data in a user friendly format in the first steps of this work.

Since it was a little hard to read the tracking data, for the initial phases of my research, a parser was implemented to parse the tracking data file and gather the information stored in the file and output them so that it becomes more readable.

This package contains the following four classes: ***PrimitiveObject***, ***BoundingBox***, ***NeighboringObjects*** and ***Parser***.

The ***PrimitiveObject*** class is responsible for the presentation of a primitive object in the framework. The presentation includes both the specification of the object and its spatial relations with other primitive objects.

Each primitive object is specified by an ID, name(type), its corresponding position, its bounding box[4] on the scene and whether or not the object is moving or is constant. This data is stored in the tracking data file. For the bounding boxes, an extra class with the name ***BoundingBox*** is implemented, which contains useful methods and functions specially for the graphical interface. The area of each object is calculated using the bounding box and is added to the specifications of the object by the framework.

The second part of the class contains the spatial relations between a given primitive object and other primitive objects in the scene. An object could be on the right

[4]The smallest rectangle surrounding an object. In this work 2-dimensional bounding boxes are used, which are also called also Minimum bounding rectangles.

or left side of another object if they do not have the same horizontal position. An object could be either above or under another object, if they do not have the same vertical position in the scene. If two objects have the same horizontal and vertical position, it means that they are located on the same position on the scene.

For alleviating the job of the reasoner while working with the spatial relations between the objects, these relations are specified for each object regarding other objects based on the nearest object for each type in the scene. For example, if the object *a1* of type **A** is on the right side of the objects *b1*, *b2* and *b3* of type **B** and the distance between *a1* and *b1* is smaller than the distance between *a1* and *b2* as well as the distance between *a1* and *b3*, it means that *a1* is located closer to *b1*; thus, only the object *b1* will be considered for the object *a1* as its right side relation regarding the objects of type **B**.

Although only the spatial relations between the nearest objects for different types are presented to the reasoner, the whole spatial relations of an object regarding other objects(regardless of their type) are presented in the data presentation of an object[5].

In order to distinguish between the information that is used for the reasoner and the information that is extracted from the tracking data file, the ***NeighboringObjects*** class was implemented. Each instance of this class contains an ID number, a type and a position. Any primitive object that is identified has an array of ***NeighboringObjects***.

Finally, the ***Parser*** class uses these three classes for parsing the tracking data and representing them in a way that is easy for the framework to work with. This class gets the path of the tracking data as input and opens the file. The main variable in this class is an array list of the instances of the ***ResultRegistry*** class. Each of the instances in this array is responsible for a frame of the clip.

The parser reads the tracking data frame by frame, identifies the objects and their corresponding specifications and constructs an instance of ***PrimitiveObject*** for each object, and updates it in the list of the objects of the frame. For each object, the spatial relations regarding other objects is also calculated and outputted[6].

When the parsing of the tracking data is finished, the keyframes are spotted; therefore, the contents of the variable *move* of the objects in each frame are logically disjuncted[7]. If the result of the conjunction is 0, it means all the objects of the frame are stable; as a result, this frame is a keyframe candidate.

If there is a series of candidate keyframes next to each other, the frame in the middle of the series will be chosen and added to the list of the keyframes. The first and the last frame of the clip are added to the keyframe list too.

Two keyframes that follow each other have to be at least 30 frames far from each other, otherwise the latter one will be discarded.

An instance of the ***Parser*** class is used in the ***General*** class. In the *main* package where the path of the file containing the tracking data of the clip is given to the class ***Parser*** and the resulting keyframes list[8] could be retrieved by calling

[5]This information is discarded, because it is not useful for the reasoner.

[6]Figure 2.3 shown in Chapter 2 is the resulting output of these steps.

[7]The objects of type *Hand* are all discarded when the parsing of the tracking data is finished; therefore, they have no effect on the rest of the work.

[8]An array of the instances of class ***ResultRegistry***.

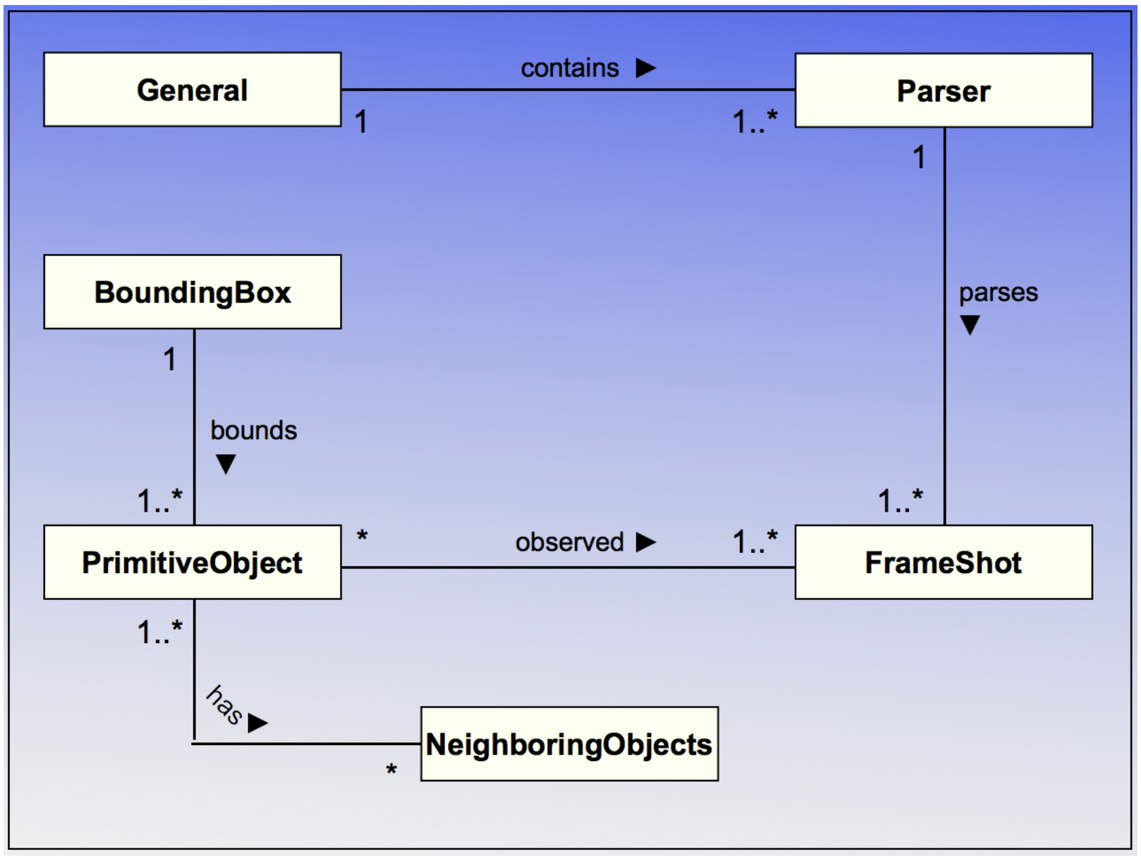


Figure 4.2: UML Class Diagram of the ***trackingDataParser*** Package

the function *getList()* of this class.

The UML class diagram of this package is shown in figure 4.2.

4.3.2 Package *graphics*

This package provides the required graphical tools for updating the data in graphical user interface (GUI) of the framework and is consisted of the four following classes: ***UpdateGUI***, ***DrawBox***, ***MyButton*** and ***GuiManager***. The UML class diagram of this package is shown in figure 4.3.

The ***MyButton*** class supports the GUI with the buttons that are placed on it and has three major tasks. Initializing the buttons is the first major task of this class. Its second assignment is recognizing if a button is pressed. The third job of this class is reacting to the action that is performed by pressing the button based on the type of the button and the event that is occurred.

The ***GuiManager*** class is the main class responsible for the establishment of the GUI. This class uses JSwing components. In the first step, the main window of the GUI and its size are initialized. The control buttons and the radio buttons for selecting the rules, the strategies and the labels will be added to the main window in the next step. A slider is configured and appended to the window, showing the position of the current keyframe in the clip. To show the content of the covers, separate scroll panes for the instances of *Objects*, ***Meal_Cover***, ***Dessert_Cover***, ***Single_Cover*** and ***Combined_Cover*** are created and embedded to the window. All the tasks regarding the update of the GUI contents such as the timer, the appearance of the waitress and, etc. (expect in the image area) are done by the methods that are implemented in this class. This class adjusts the size and area of the contents of the GUI using the finest size.

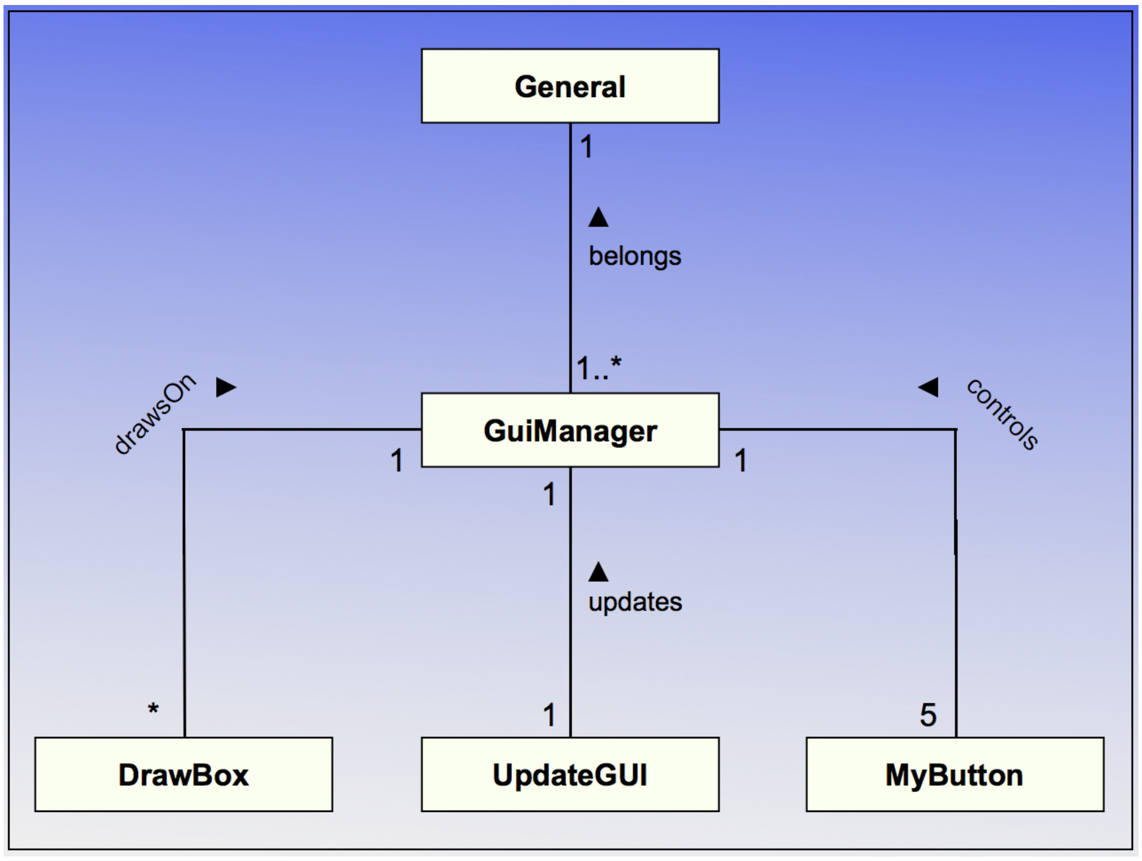


Figure 4.3: UML Class Diagram of the ***graphics*** Package

When the framework and RacerPro analyze the scene and the high-level interpretations are asserted into the ABox, the new updated ABox should be illustrated on the GUI. This is done in the ***UpdateGUI*** class. This class loads the picture of the frame on the GUI. The information that is presented in the ABox should be shown and marked on the picture.

Each object will be marked with its ID on the picture. If a new object has been identified, its ID will be shown in red and if the object has been identified in the previous keyframes, its ID will be marked in blue. This class also calculates the bounding boxes of the new high-level instances such as instances of ***Meal_Cover***, ***Dessert_Cover*** and ***Combined_Cover***. Each high-level instance and its bounding box will be shown by a color that is specified for them randomly. The objects that are grouped under a high-level instance are also included in the bounding boxes of the higher-level instances.

When the user asks the framework to analyze the next keyframe by pressing the *Next Button*, the GUI shows a message informing the user to wait until the interpretation of the keyframe is finished. When the interpretation is done, the package discussed above gets the new data from the ***KeyFrameManager***[9] and updates the contents of the GUI.

4.3.3 Package *frameManagement*

This package, which manages the operations between RacerPro and the framework, contains the following two classes: ***ClipManager*** and ***KeyFrameManager***.

In the ***ClipManager*** class, first, the list of the primitive objects which are discovered by the image processing tools and saved in the tracking data file is loaded. In the next step, the picture of the frame is placed in the image area of the GUI. This class also decides whether or not an instance of ***Waitress*** should be added to the scene. If it decides to add an instance of ***Waitress***, it should also determine the frame number in which this instance should be added. Based on the combination of the rules and the strategies, the corresponding instance of the ***KeyFrameManager*** class is initialized for the current frame. The timer of the framework starts when the ***KeyFrameManager*** class takes the control.

An instance of ***KeyFrameManager*** is responsible for a specific keyframe. In this class, for each combination of the rules, the strategies and the position of the keyframe in the clip (frame number), a separate ABox is created. After adding the individuals (primitive objects) and role assertions to the ABox, the methods of the corresponding interpretation class are called based on the rules and the strategies for the explanation of the ABox (scene).

If the **Optimist** strategy is selected, all the role assertions identified between the individuals by the parser are asserted. For the **Pessimist**, no role assertion is added (unless the last frame has been reached). Finally, for the **Realist** strategy, those roles which appear to remain until the end of the clip are asserted into the ABox. After finishing the interpretation of a scene and before starting the next level, the framework performs the closing operation mentioned in 3.3.

When the closing operation ends, the control goes back to the ***ClipManager*** class. This class stops the timer and prints the time between the loading of the ABox and the end of the interpretation on the GUI. Afterwards, the content of the GUI text boxes is updated. The new content is delivered by running the command ***"(retrieve (?x) (?x C***[10]***))"*** for each desired concept. RacerPro answers this query by returning the list of the individuals, which are the instances of ***C***.

After updating the content of the GUI, the framework waits for an event triggered by the user such as changing the rule or the strategy of interpretation of the frame (using the buttons). If the exit button is pressed, this class closes the connection with the RacerPro.

An instance of the ***ClipManager*** class should be available in the ***General*** class in the package ***main***. However, before that, the tracking data should be presented as discussed in Section 4.3.1.

[9]More details are available in the description of the *frameManagement* package.

[10]C stands for a concept.

4.3.4 Package *rules*

The task of this package is controlling the interpretation of the scene based on the rule that is selected by the user in the GUI. The different rule groups are ***Absolute-bottom-up***, ***Absolute-top-down***, ***Stepwise-bottom-up*** and ***Stepwise-top-down*** and their corresponding classes are ***AbsoluteBottomUpRules***, ***AbsoluteTopDownRules***, ***StepwiseBottomUpRules*** and ***StepwiseTopDownRules***. In all of these classes, the command ***retrieve-with-explanation*** is used for the interpretation.

In the classes responsible for the rules with bottom-up directions, the reasoner and framework try to explain the existing assertions. In other words, if the head of a rule matches an assertion (either a concept or a role assertion) the reasoner tries to find an explanation that matches the body of the rule.

It is possible that the assertion matches the head of different rules. Therefore, different explanations could be delivered by the reasoner for the same assertion. Only the explanations that do not cause the ABox to become inconsistent are accepted[11]. If more than one consistent explanation is found, those with the highest score will be selected and asserted into the ABoxes.

The main difference between the ***AbsoluteBottomUpRules*** and ***StepwiseBottomUpRules*** classes is that in the first class, the aggregates are created by the primitive objects; however, in the second class, the aggregates in each level are created using the primitive objects or aggregates in the lower level. For example, the instances of ***Meal_Cover*** or ***Dessert_Cover*** in this class are generated using the primitive objects and the instance of ***Combined_Cover*** are created using the instances of ***Meal_Cover*** or ***Dessert_Cover***.

In the classes containing the top-down rules, the framework and the reasoner try to find instances of a desired aggregate based on the observations that are available in the ABox. In other words, it will be attempted to explain the observation by creating aggregates that verify the occurrence of these observations and events. In this situation, all or some of the observations are used as entailed assertions to generate the aggregates by hypothesizing concept or role assertions. The less hypothesized assertions and the more entailed assertions are used, the more reliable the explanation is.

In the ***AbsoluteTopDownRules*** class, it will look for the aggregates in the highest level first, whereas in the ***StepwiseTopDownRules*** class, it will look for the aggregates in the lowest level. In the table cloth scenario, the first class tries to create an instance of ***Combined_Cover*** first and then the instances of ***Meal_Cover*** and ***Dessert_Cover***. The opposite happens in the second class.

It should be mentioned that these classes are **not implemented hard-coded**; therefore, if the pattern of the scenario in this work changes, these classes should not be re-implemented.

The rule patterns for each class are saved in separate files under the folder ***Rules***. Before the interpretation begins, the rules should be loaded in RacerPro. This is done in the ***LoadRules*** class. When an instance of this class is created, the previous rules, which are defined in the reasoner, will be deleted first and the new rules are

[11]This is possible by enabling the ***final-consistency-p*** feature of the explanation command.

read from the corresponding file and loaded into the reasoner. This option makes the definition of the rules independent of how the rules should be used and fired in the framework.

4.3.5 Package *racerInterface*

This package is based on the JRacer API[12] provided by the Racer Systems Company to access the server of RacerPro. JRacer uses a socket-based interface for the connection to RacerPro. This is implemented in the ***RacerClient*** class. One of the most important classes in this package, is the ***RacerStubs*** class in which for each Racer function, a Java function is represented with the same signature.

In addition to this package, the ***RacerCall*** class is implemented for establishing a connection with RacerPro using the arguments required for this work, whenever a connection is needed between the framework and RacerPro. The port *8088* and the *localhost* as the IP are used for opening the connection. After opening the connection, the RacerPro is fully reset if desired by the user, and the logging-on option is enabled. This makes it possible to see all the operations done in RacerPro while the framework is running; therefore, the debugging of the framework regarding RacerPro becomes easier. By closing the connection, the logging-on option is disabled.

4.3.6 Package *main*

This package contains the classes that start the framework and track the results of the interpretation and includes the following three classes: ***ResultRegistry***, ***InterpretationTimer*** and ***General***.

The ***ResultRegistry*** class registers the results of the interpretation. For each combination of frame numbers, rules and strategies, the duration of the interpretation is registered separately. Since there are three strategies and four different rule types in this work, for each frame twelve different interpretation times could be measured. Each of these twelve interpretation times is set once for a frame. Thus, if a frame is revisited in a situation[13] that has been visited before, the same interpretation time for that situation that has been measured before, will be outputted on the GUI. For measuring the time, the ***InterpretationTimer*** has been implemented in which the time of an event could be measured by using the *start* and *stop* functions.

Finally, the class ***General*** that contains the *static void main()* function is the class that starts the framework. First, an instance of the ***Parser*** class is created to read and parse the tracking data from the tracking data file and hand over these data in a framework-friendly presentation. In the next step, it establishes a connection with RacerPro and loads the TBox in the reasoner. In the end, the GUI of the framework is started and the interpretation of the keyframes begins. The results of running this class are shown in the next Chapter.

[12]The JRacer API can be downloaded from
http://www.racer-systems.com/products/download/nativelibraries.phtml .

[13]A combination of a rule and a strategy.

Chapter 5

Tests and Results

This Chapter focuses on the results of the interpretation done by the framework for the video document introduced in the previous Chapters.

The results of running the framework in two different states are illustrated in the first Section. The first situation is where a user has chosen the **Stepwise-bottom-up** rule and the **Optimist** strategy. In the second example, the rule is changed to **Absolute-top-down** and the strategy is switched to **Realist**.

In the second section, a comparison between the results of interpretation based on different combinations of rules and strategies is shown. Each combination is demonstrated graphically as well as in tables.

5.1 Results of Run

In this Section, the interpretation results for two different situations will be demonstrated and discussed. The first situation is where the **Stepwise-bottom-up** rule and **Optimist** strategy are selected. In the second situation, the **Absolute-top-down** rule and **Realist** strategy are chosen. For simplification, the first situation is called **SBUO** and the second situation is called **ATDR**[1].The interpretation results for each combination of rules and strategies are illustrated at the end of this work.

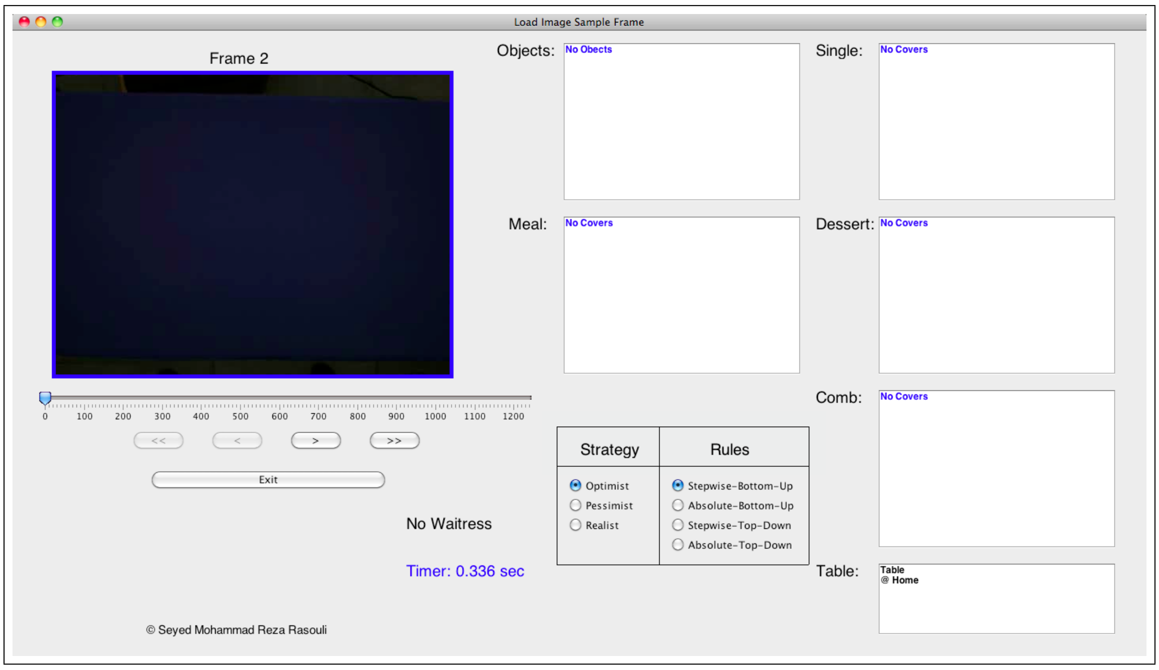
5.1.1 Rule: Stepwise-bottom-up, Strategy: Optimist

The framework starts with the interpretation of the first frame of the clip. As it can be seen, the first frame is an empty table, and no primitive objects are identified in this frame (figure 5.1). It should be mentioned that the first frame is not a keyframe.

In frame 86 (the first keyframe), a plate and a saucer have been identified by the image processing tool[2]. Since these objects have been appeared in this frame for

[1]The full description of the abbreviations used in this Chapter are illustrated in table B.1.

[2]The information in the tracking data file contains the results of the image processing tool used by KOGS[11].


Load Image Sample Frame

Frame 2

Objects: No Obects

Single: No Covers

Meal: No Covers

Dessert: No Covers

Comb: No Covers

Table: Table @ Home

Strategy: [✓ Optimist] Pessimist Realist

Rules: [✓ Stepwise-Bottom-Up] Absolute-Bottom-Up Stepwise-Top-Down Absolute-Top-Down

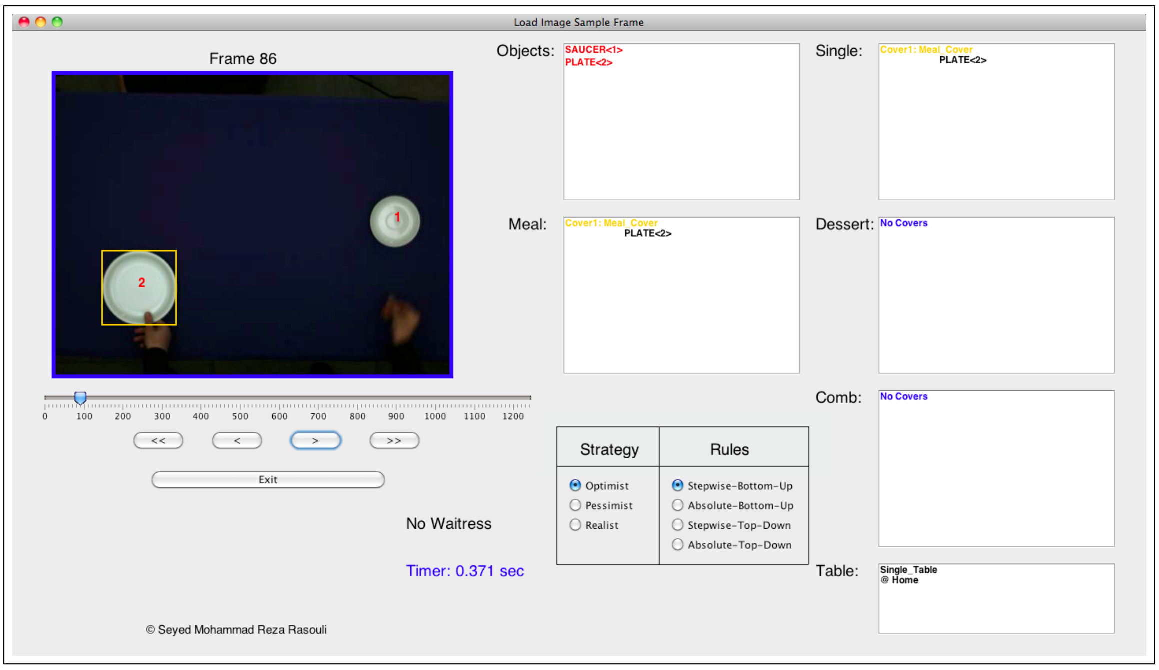
No Waitress

Timer: 0.336 sec

© Seyed Mohammad Reza Rasouli

Figure 5.1: Rule: Stepwise-bottom-up, Strategy: Optimist - Frame 2

the first time, they are written in red color in the image area and also listed in red under the list of the objects.


Load Image Sample Frame

Frame 86

Objects: SAUCER<1> PLATE<2>

Single: Cover1: Meal_Cover PLATE<2>

Meal: Cover1: Meal_Cover PLATE<2>

Dessert: No Covers

Comb: No Covers

Table: Single_Table @ Home

Strategy: [✓ Optimist] Pessimist Realist

Rules: [✓ Stepwise-Bottom-Up] Absolute-Bottom-Up Stepwise-Top-Down Absolute-Top-Down

No Waitress

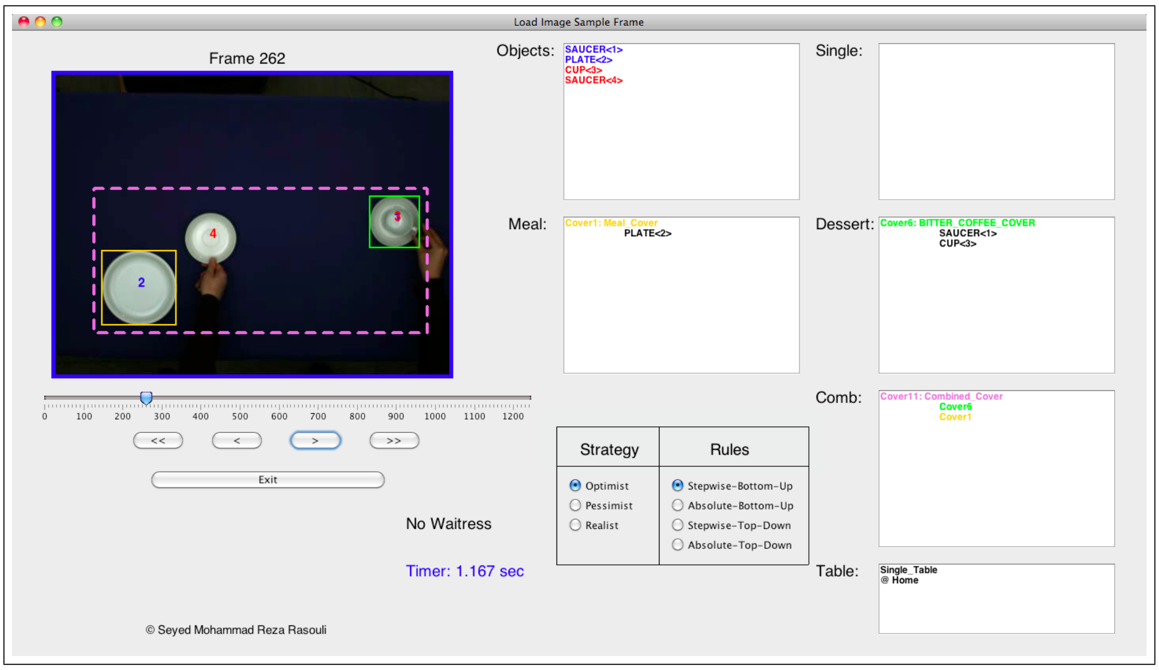
Timer: 0.371 sec

© Seyed Mohammad Reza Rasouli

Figure 5.2: Rule: Stepwise-bottom-up, Strategy: Optimist - Frame 86

The appearance of the plates points to the existence of an instance of ***Meal_Cover*** in this frame and as it is shown in figure 5.2, an instance of ***Meal_Cover*** with the name ***Cover1***[3] is instantiated in this frame that can be explained by the presence of the plate. Since the instance of ***Meal_Cover*** in this frame does not belong to an instance of ***Combined_Cover***, the cover is listed under the single covers. Because only one cover has been instantiated by the framework for this frame[4], the table has been designated as a single table.

The next keyframe is frame 262 that is illustrated in figure 5.3. This figure shows that a cup and a saucer have become apparent in this frame. The objects that have been discovered in the previous keyframe continue to exist in this frame; therefore, their IDs are marked in blue in the image area, and they are also listed in blue under the list of the objects. The cup and the saucer under it are grouped under an instance of ***Dessert_Cover*** with the name **Cover6**. In addition, the description of this cover matches the ***Dessert_Cover*** with the name ***Bitter_Coffee_Cover***; thus, this cover is listed as an instance of ***Bitter_Coffee_Cover*** under the list of ***Dessert_Covers***.


Load Image Sample Frame

Frame 262

Objects:
SAUCER<1>
PLATE<2>
CUP<3>
SAUCER<4>

Single:

Meal:
Cover1: Meal_Cover
PLATE<2>

Dessert:
Cover6: BITTER_COFFEE_COVER
SAUCER<1>
CUP<3>

Comb:
Cover11: Combined_Cover
Cover6
Cover1

Strategy: [✓ Optimist] Pessimist Realist

Rules: [✓ Stepwise-Bottom-Up] Absolute-Bottom-Up Stepwise-Top-Down Absolute-Top-Down

No Waitress

Timer: 1.167 sec

Table:
Single_Table
@ Home

© Seyed Mohammad Reza Rasouli

Figure 5.3: Rule: Stepwise-bottom-up, Strategy: Optimist - Frame 262

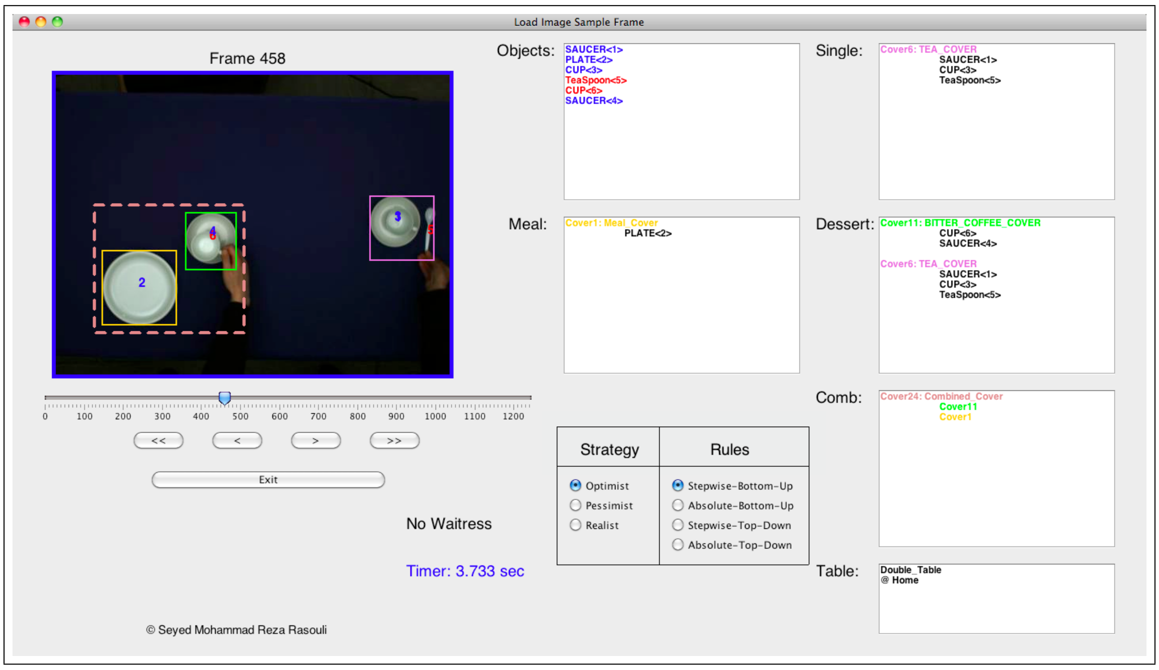
Since the instance of ***Dessert_Cover*** in this frame is on the right side of the instance of ***Meal_Cover*** and there are no instances of ***Dessert_Cover*** between them, these two covers build an instance of the higher ***Combined_Cover*** aggregate with the name **Cover11**. The placement of the primitive objects in this frame suggests that the instance of the ***Combined_Cover*** has been created as an explanation for the scene of this frame by mistake. This surmise is verified as the framework continues with the interpretation of the next keyframes.

[3]The names for the new individuals are generated by RacerPro automatically. The framework only defines the new prefix for the new individuals. The prefix chosen for this work is the word **Cover**.

[4]Independent of their types.

Although three covers have been constructed as explanations for the content of the table in this frame, this table has been characterized as a table for a single person because the ***Combined_Cover*** is a higher aggregate compared to the other two aggregates (***Meal_Cover*** and ***Dessert_Cover***). On the other hand, the instances of these two lower aggregates belong to an instance of ***Combined_Cover*** and since no one shares a cover with another person, only one person can sit at this table.

In figure 5.4, the interpretation for the frame 458 is demonstrated. In this frame, four covers have been created. The new instance with the name **CUP**<**6**> points to a new instance of ***Dessert_Cover***, which could be considered with the other instance of ***Dessert_Cover*** that was found in the previous cover. This new instance has the name **Cover11** and contains **CUP**<**6**> and **TeaSpoon**<**5**> individuals. Since **CUP**<**6**> that is on the right side of **PLATE**<**2**> is located closer to this plate than **CUP**<**3**>, the instance of ***Combined_Cover***, which has been found in the previous keyframe, is discarded and the new instance of ***Combined_Cover*** named **Cover24** has been created. This cover contains the instance of ***Meal_Cover*** that was created in the previous keyframe and the new instance of ***Dessert_Cover***.



Objects:
SAUCER<1>
PLATE<2>
CUP<3>
TeaSpoon<5>
CUP<6>
SAUCER<4>

Single:
Cover6: TEA_COVER
SAUCER<1>
CUP<3>
TeaSpoon<5>

Meal:
Cover1: Meal_Cover
PLATE<2>

Dessert:
Cover11: BITTER_COFFEE_COVER
CUP<6>
SAUCER<4>

Cover6: TEA_COVER
SAUCER<1>
CUP<3>
TeaSpoon<5>

Comb:
Cover24: Combined_Cover
Cover11
Cover1

Strategy	Rules
[✓ Optimist]	[✓ Stepwise-Bottom-Up]
Pessimist	Absolute-Bottom-Up
Realist	Stepwise-Top-Down
	Absolute-Top-Down

No Waitress

Timer: 3.733 sec

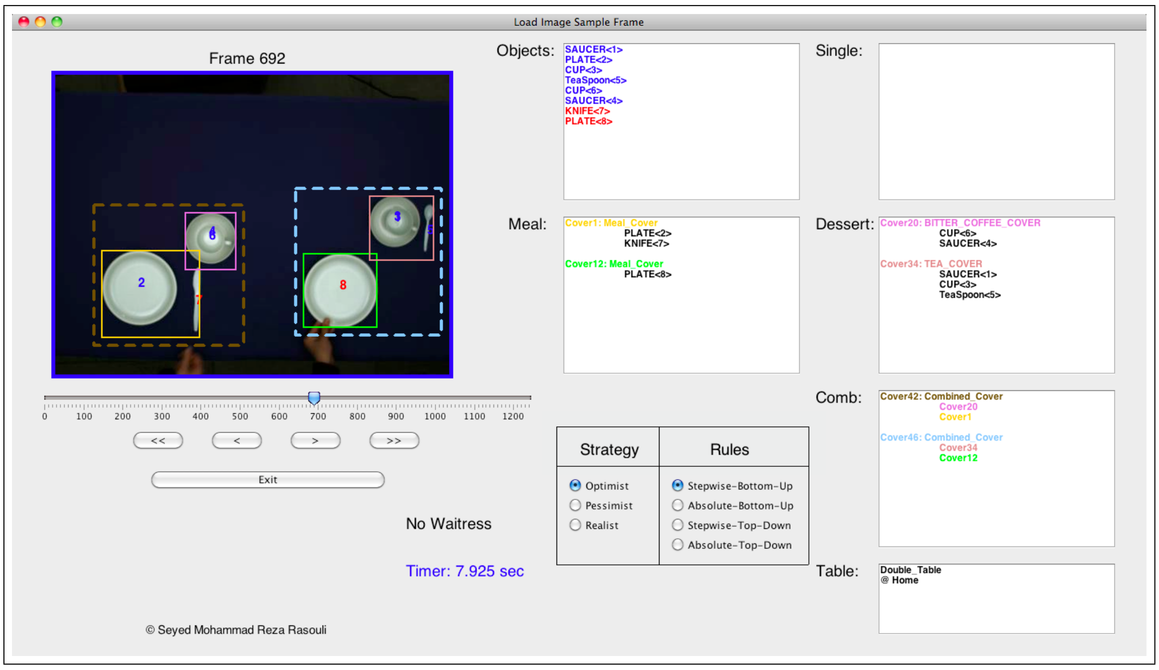
Table:
Double_Table
@ Home

© Seyed Mohammad Reza Rasouli

Figure 5.4: Rule: Stepwise-bottom-up, Strategy: Optimist - Frame 458

The absence of a teaspoon in **Cover11** has caused this cover to become an instance of ***BITTER_COFFEE_COVER***. However, the situation is different on the other side of the table. The **CUP**<**3**>, **SAUCER**<**1**> and **TeaSpoon**<**5**> individuals are grouped under an instance of ***Dessert_Cover*** with the name **Cover6**. This cover contains all the possible primitive objects that an instance of ***Dessert_Cover*** could have. **Cover6** has the type ***TEA_Cover*** and is listed under the single covers because it does not belong to an instance of ***Combined_Cover***.

In general, two independent covers are found in this frame, and two persons can now sit at this table. Therefore, the type of the table is ***Double_Table***.


Load Image Sample Frame

Frame 692

Objects:
SAUCER<1>
PLATE<2>
CUP<3>
TeaSpoon<5>
CUP<6>
SAUCER<4>
KNIFE<7>
PLATE<8>

Single:

Meal:
Cover1: Meal_Cover
PLATE<2>
KNIFE<7>

Cover12: Meal_Cover
PLATE<8>

Dessert:
Cover20: BITTER_COFFEE_COVER
CUP<6>
SAUCER<4>

Cover34: TEA_COVER
SAUCER<1>
CUP<3>
TeaSpoon<5>

Comb:
Cover42: Combined_Cover
Cover20
Cover1

Cover46: Combined_Cover
Cover34
Cover12

Strategy	Rules
[✓ Optimist]	[✓ Stepwise-Bottom-Up]
Pessimist	Absolute-Bottom-Up
Realist	Stepwise-Top-Down
	Absolute-Top-Down

Table:
Double_Table
@ Home

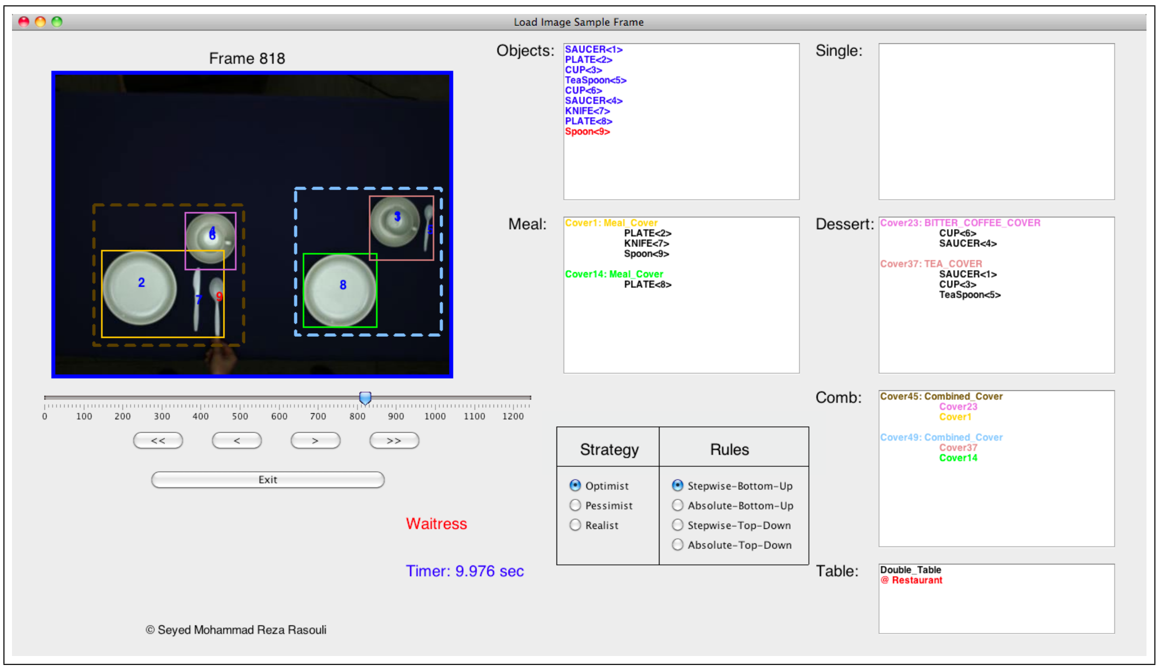
No Waitress

Timer: 7.925 sec

© Seyed Mohammad Reza Rasouli

Figure 5.5: Rule: Stepwise-bottom-up, Strategy: Optimist - Frame 692

A new situation is seen in the next keyframe. Figure 5.5 shows that in keyframe 692, six covers have been found, namely two instance ***Meal_Cover***, two instances of ***Dessert_Cover*** and two instances of ***Combined_Cover***.


Load Image Sample Frame

Frame 818

Objects:
SAUCER<1>
PLATE<2>
CUP<3>
TeaSpoon<5>
CUP<6>
SAUCER<4>
KNIFE<7>
PLATE<8>
Spoon<9>

Single:

Meal:
Cover1: Meal_Cover
PLATE<2>
KNIFE<7>
Spoon<9>

Cover14: Meal_Cover
PLATE<8>

Dessert:
Cover23: BITTER_COFFEE_COVER
CUP<6>
SAUCER<4>

Cover37: TEA_COVER
SAUCER<1>
CUP<3>
TeaSpoon<5>

Comb:
Cover45: Combined_Cover
Cover23
Cover1

Cover49: Combined_Cover
Cover37
Cover14

Strategy	Rules
[✓ Optimist]	[✓ Stepwise-Bottom-Up]
Pessimist	Absolute-Bottom-Up
Realist	Stepwise-Top-Down
	Absolute-Top-Down

Table:
Double_Table
@ Restaurant

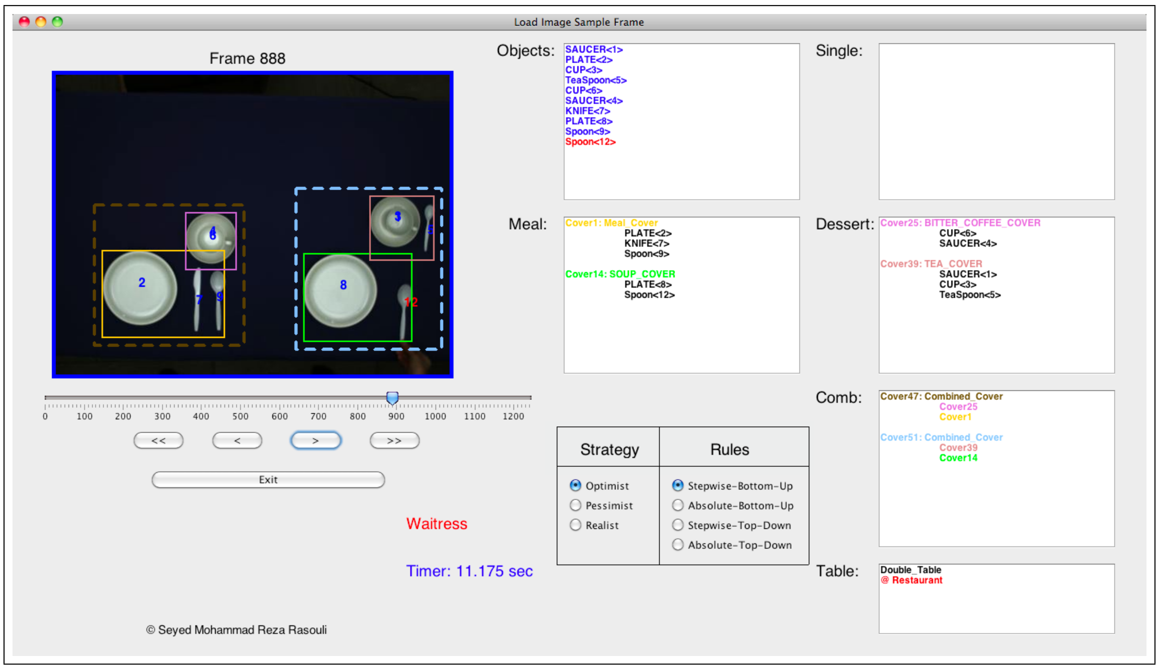
Waitress

Timer: 9.976 sec

© Seyed Mohammad Reza Rasouli

Figure 5.6: Rule: Stepwise-bottom-up, Strategy: Optimist - Frame 818

Since in this keyframe a new instance of ***PLATE***, named **PLATE<8>**, has been


Load Image Sample Frame

Frame 888

Objects:
SAUCER<1>
PLATE<2>
CUP<3>
TeaSpoon<5>
CUP<6>
SAUCER<4>
KNIFE<7>
PLATE<8>
Spoon<9>
Spoon<12>

Single:

Meal:
Cover1: Meal_Cover
PLATE<2>
KNIFE<7>
Spoon<9>

Cover14: SOUP_COVER
PLATE<8>
Spoon<12>

Dessert:
Cover25: BITTER_COFFEE_COVER
CUP<6>
SAUCER<4>

Cover39: TEA_COVER
SAUCER<1>
CUP<3>
TeaSpoon<5>

Comb:
Cover47: Combined_Cover
Cover25
Cover1

Cover51: Combined_Cover
Cover39
Cover14

Strategy	Rules
[✓ Optimist]	[✓ Stepwise-Bottom-Up]
Pessimist	Absolute-Bottom-Up
Realist	Stepwise-Top-Down
	Absolute-Top-Down

Table:
Double_Table
@ Restaurant

Waitress

Timer: 11.175 sec

© Seyed Mohammad Reza Rasouli

Figure 5.7: Rule: Stepwise-bottom-up, Strategy: Optimist - Frame 888

added to the scene, a new instance of ***Meal_Cover*** with the name **Cover12** is created. Together with the instance of ***Dessert_Cover*** that has been found in keyframe 458 and contains **SAUCER**$<$**1**$>$, **CUP**$<$**3**$>$ and **TeaSpoon**$<$**5**$>$, they build up a new instance of ***Combined_Cover***. This new instance of ***Combined_Cover*** has the name **Cover46**.

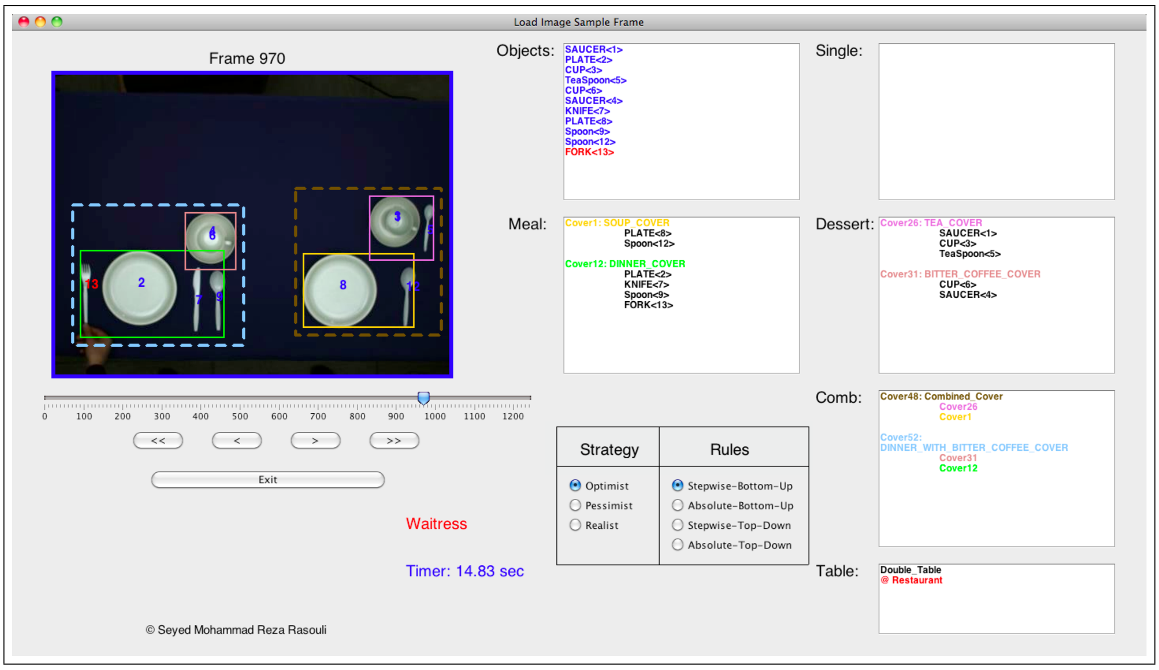
Another new individual that is identified in this frame is an instance of ***KNIFE*** that will be added to **Cover1** which is a meal cover. In this frame,**Cover1** and **Cover20** build an instance of ***Combined_Cover*** with the name **Cover42**, and **Cover12** and **Cover34** build an instance of ***Combined_Cover*** named as **Cover46**.

It should be noted that since no waitress has been seen so far, the framework assumes that the table is located at home. This assumption turns to be wrong as in the next keyframe a waitress will be seen (figure 5.6. From this point, the reasoner and framework know that the table is not located in a house but in a restaurant. This is also shown for frame 818 in the table specifications on the lower right corner of the GUI (Figure 5.6). The new individual **Spoon**$<$**7**$>$ is added to the instance of ***Meal_Cover*** that contains the individual**PLATE**$<$**2**$>$.

The new instance of ***Spoon*** in frame 888 causes the meal cover on the right side to get the type ***SOUP_Cover*** as figure 5.7 shows. According to the interpretation time, the closer it gets to the end of the clip, the longer it takes to interpret the scene.

Fork$<$**13**$>$ in the keyframe 970 is the first individual of type ***FORK*** that has been identified from the beginning of the clip. This individual is added to the **Cover12** and gives this cover the type ***Dinner_Cover***. The other covers keep the same **most specified types** that have been shown and discussed in the previous keyframes.

As illustrated in 5.8, **Cover12** contains all possible primitive objects that an in-


Load Image Sample Frame

Frame 970

Objects:
SAUCER<1>
PLATE<2>
CUP<3>
TeaSpoon<5>
CUP<6>
SAUCER<4>
KNIFE<7>
PLATE<8>
Spoon<9>
Spoon<12>
FORK<13>

Single:

Meal:
Cover1: SOUP_COVER
PLATE<8>
Spoon<12>

Cover12: DINNER_COVER
PLATE<2>
KNIFE<7>
Spoon<9>
FORK<13>

Dessert:
Cover26: TEA_COVER
SAUCER<1>
CUP<3>
TeaSpoon<5>

Cover31: BITTER_COFFEE_COVER
CUP<6>
SAUCER<4>

Comb:
Cover48: Combined_Cover
Cover26
Cover1

Cover52:
DINNER_WITH_BITTER_COFFEE_COVER
Cover31
Cover12

Table:
Double_Table
@ Restaurant

Strategy: [✓ Optimist] Pessimist Realist

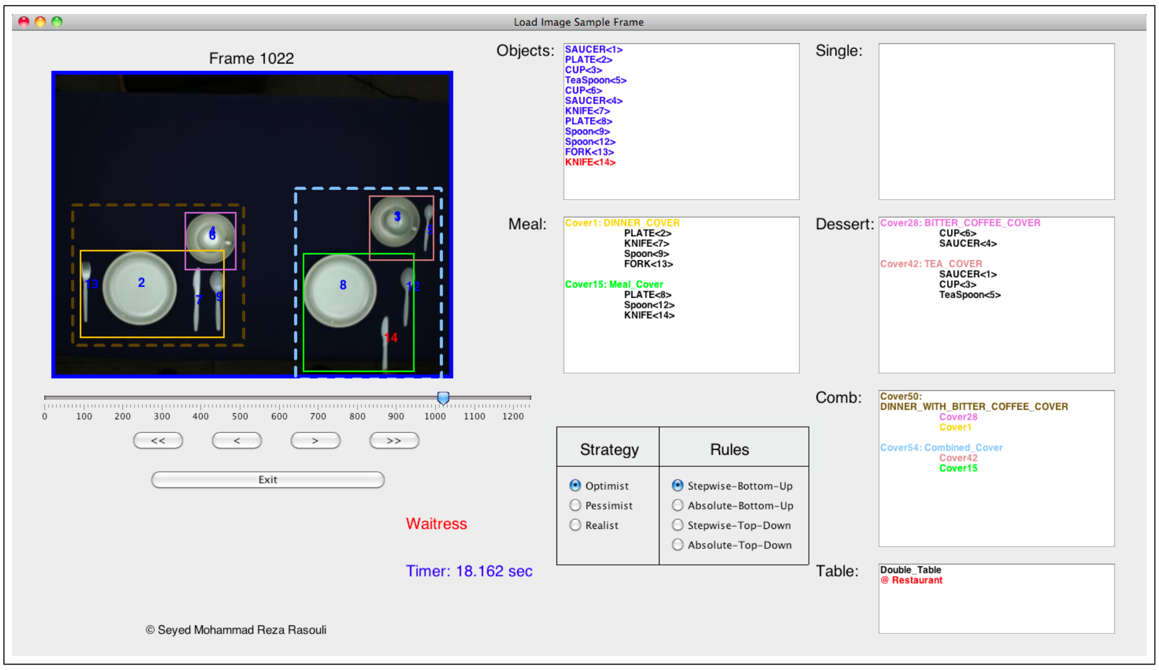
Rules: [✓ Stepwise-Bottom-Up] Absolute-Bottom-Up Stepwise-Top-Down Absolute-Top-Down

Waitress

Timer: 14.83 sec

© Seyed Mohammad Reza Rasouli

Figure 5.8: Rule: Stepwise-bottom-up, Strategy: Optimist - Frame 970


Load Image Sample Frame

Frame 1022

Objects:
SAUCER<1>
PLATE<2>
CUP<3>
TeaSpoon<5>
CUP<6>
SAUCER<4>
KNIFE<7>
PLATE<8>
Spoon<9>
Spoon<12>
FORK<13>
KNIFE<14>

Single:

Meal:
Cover1: DINNER_COVER
PLATE<2>
KNIFE<7>
Spoon<9>
FORK<13>

Cover15: Meal_Cover
PLATE<8>
Spoon<12>
KNIFE<14>

Dessert:
Cover28: BITTER_COFFEE_COVER
CUP<6>
SAUCER<4>

Cover42: TEA_COVER
SAUCER<1>
CUP<3>
TeaSpoon<5>

Comb:
Cover50:
DINNER_WITH_BITTER_COFFEE_COVER
Cover28
Cover1

Cover54: Combined_Cover
Cover42
Cover15

Table:
Double_Table
@ Restaurant

Strategy: [✓ Optimist] Pessimist Realist

Rules: [✓ Stepwise-Bottom-Up] Absolute-Bottom-Up Stepwise-Top-Down Absolute-Top-Down

Waitress

Timer: 18.162 sec

© Seyed Mohammad Reza Rasouli

Figure 5.9: Rule: Stepwise-bottom-up, Strategy: Optimist - Frame 1022

stance of ***Meal_Cover*** could have; hence, no additional primitive object of type ***Meal_Objects*** could be added to this cover anymore. In addition, the type of the instance of ***Combined_ Cover*** that contains **Cover12** and **Cover31** (shown in 5.8) has been updated to ***DINNER _WITH_ BITTER_COFFEE_COVER*** because

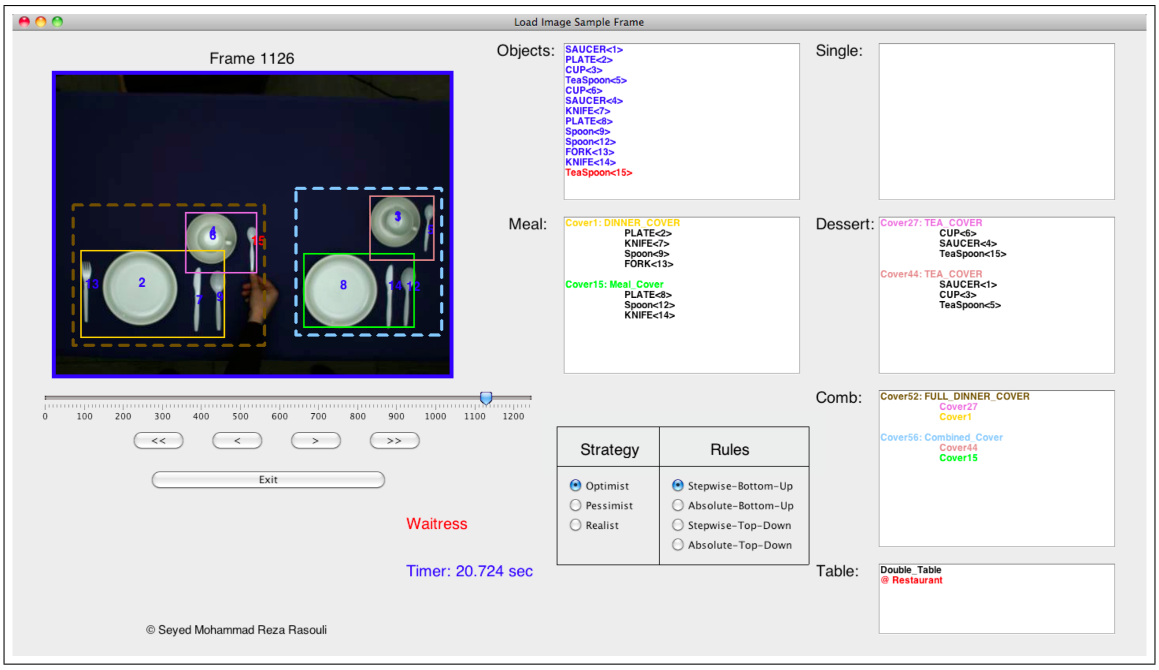

```
Load Image Sample Frame
Frame 1126                      Objects: SAUCER<1>
                                         PLATE<2>
                                         CUP<3>
                                         TeaSpoon<5>
                                         CUP<6>
                                         SAUCER<4>
                                         KNIFE<7>
                                         PLATE<8>
                                         Spoon<9>
                                         Spoon<12>
                                         FORK<13>
                                         KNIFE<14>
                                         TeaSpoon<15>
Single:
Meal:    Cover1: DINNER_COVER
                 PLATE<2>
                 KNIFE<7>
                 Spoon<9>
                 FORK<13>
         Cover15: Meal_Cover
                 PLATE<8>
                 Spoon<12>
                 KNIFE<14>
Dessert: Cover27: TEA_COVER
                 CUP<6>
                 SAUCER<4>
                 TeaSpoon<15>
         Cover44: TEA_COVER
                 SAUCER<1>
                 CUP<3>
                 TeaSpoon<5>
Comb:    Cover52: FULL_DINNER_COVER
                 Cover27
                 Cover1
         Cover56: Combined_Cover
                 Cover44
                 Cover15
Strategy: (●) Optimist  ( ) Pessimist  ( ) Realist
Rules:    (●) Stepwise-Bottom-Up  ( ) Absolute-Bottom-Up  ( ) Stepwise-Top-Down  ( ) Absolute-Top-Down
Waitress
Timer: 20.724 sec
Table:   Double_Table
         @ Restaurant
© Seyed Mohammad Reza Rasouli
```

Figure 5.10: Rule: Stepwise-bottom-up, Strategy: Optimist - Frame 1126

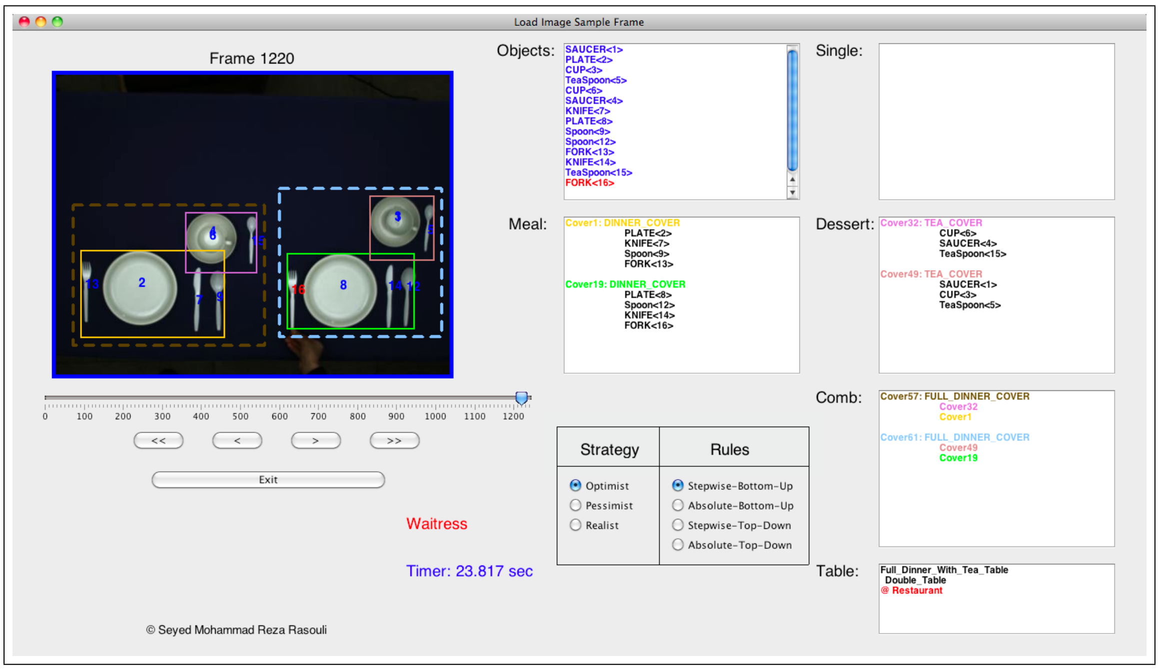
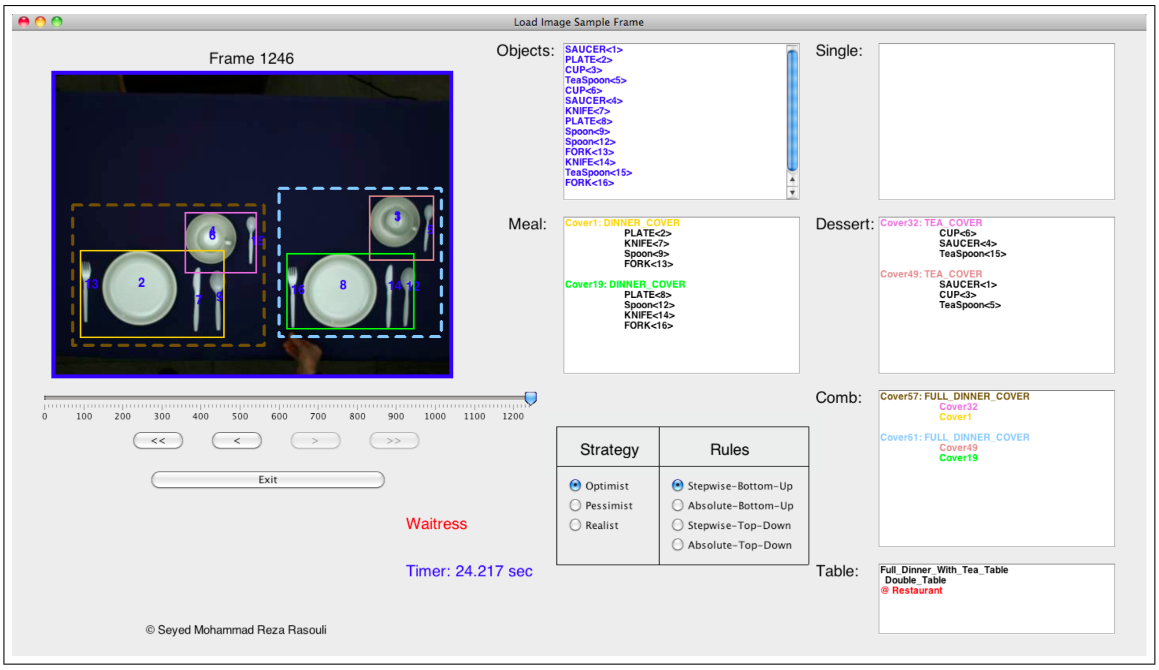

```
Load Image Sample Frame
Frame 1220                      Objects: SAUCER<1>
                                         PLATE<2>
                                         CUP<3>
                                         TeaSpoon<5>
                                         CUP<6>
                                         SAUCER<4>
                                         KNIFE<7>
                                         PLATE<8>
                                         Spoon<9>
                                         Spoon<12>
                                         FORK<13>
                                         KNIFE<14>
                                         TeaSpoon<15>
                                         FORK<16>
Single:
Meal:    Cover1: DINNER_COVER
                 PLATE<2>
                 KNIFE<7>
                 Spoon<9>
                 FORK<13>
         Cover19: DINNER_COVER
                 PLATE<8>
                 Spoon<12>
                 KNIFE<14>
                 FORK<16>
Dessert: Cover32: TEA_COVER
                 CUP<6>
                 SAUCER<4>
                 TeaSpoon<15>
         Cover49: TEA_COVER
                 SAUCER<1>
                 CUP<3>
                 TeaSpoon<5>
Comb:    Cover57: FULL_DINNER_COVER
                 Cover32
                 Cover1
         Cover61: FULL_DINNER_COVER
                 Cover49
                 Cover19
Strategy: (●) Optimist  ( ) Pessimist  ( ) Realist
Rules:    (●) Stepwise-Bottom-Up  ( ) Absolute-Bottom-Up  ( ) Stepwise-Top-Down  ( ) Absolute-Top-Down
Waitress
Timer: 23.817 sec
Table:   Full_Dinner_With_Tea_Table
          Double_Table
         @ Restaurant
© Seyed Mohammad Reza Rasouli
```

Figure 5.11: Rule: Stepwise-bottom-up, Strategy: Optimist - Frame 1220

the instance of ***Meal_Cover*** that belongs to this cover has the type ***DINNER_COVER*** and the instance of ***Dessert_ Cover*** that belongs to this cover has the type ***BITTER_COFFEE_COVER***.

In the next keyframe (figure 5.9), the only difference is the appearance of the indi-



Timer: 24.217 sec

Objects:
SAUCER<1>
PLATE<2>
CUP<3>
TeaSpoon<5>
CUP<6>
SAUCER<4>
KNIFE<7>
PLATE<8>
Spoon<9>
Spoon<12>
FORK<13>
KNIFE<14>
TeaSpoon<15>
FORK<16>

Single:

Meal:
Cover1: DINNER_COVER
PLATE<2>
KNIFE<7>
Spoon<9>
FORK<13>

Cover19: DINNER_COVER
PLATE<8>
Spoon<12>
KNIFE<14>
FORK<16>

Dessert:
Cover32: TEA_COVER
CUP<6>
SAUCER<4>
TeaSpoon<15>

Cover49: TEA_COVER
SAUCER<1>
CUP<3>
TeaSpoon<5>

Comb:
Cover57: FULL_DINNER_COVER
Cover32
Cover1

Cover61: FULL_DINNER_COVER
Cover49
Cover19

Strategy: [✓ Optimist] Pessimist Realist

Rules: [✓ Stepwise-Bottom-Up] Absolute-Bottom-Up Stepwise-Top-Down Absolute-Top-Down

Table:
Full_Dinner_With_Tea_Table
Double_Table
@ Restaurant

Figure 5.12: Rule: Stepwise-bottom-up, Strategy: Optimist - Frame 1246

vidual **KNIFE<14>**. As it is shown, this individual is added to the**Cover15**. Figure 5.10 shows that both instances of ***Dessert_Cover*** in the next keyframe have the type ***TEA_COVER***. As a result, the type of the instance of instance of ***Combined_Cover*** on the left side is changed to ***FULL_DINNER_COVER***. This cover contains all the different primitive objects that are introduced in this work.

Finally, the individual **FORK<16>** in frame 1220 causes all the covers to become full; all possible primitive objects that each cover could contain is listed under these covers. In other words, the two instances of ***Meal_Cover*** both have the type ***FULL_DINNER_COVER*** and the two instances of ***Dessert_Cover*** both have the type ***TEA_COVER***. The resulting instances of ***Combined_Cover*** both have the type ***FULL_DINNER _WITH_TEA_COVER***. Therefore, the type of the table is changed to ***FULL_DINNER_WITH_TEA*** as shown in figure 5.11.

Frame 1220 is the last keyframe in the clip. As mentioned before, the interpretation will also be done for the last frame of the clip. Figure 5.12 shows the last frame of the clip, frame 1246. No new individual is found in this frame and the individuals listed for this frame are the primitive objects that have been identified in the previous frames.

It takes **24.217 seconds** for the framework and the reasoner to interpret the last frame of the clip. The average interpretation time for the keyframes is **11.369 seconds**.

Since the clip has 1246 frames and the average interpretation time has almost been reached by frame 888, it could indicate that almost **71.26%** of the frames are interpreted below the average interpretation time of the frames. In Section 5.2, these time and percentage calculations will be studied more.

The graph in figure 5.13 shows that how the interpretation time changes as the frame number increases. The average time is shown with a green line. Using MatLab, the interpolation for the interpretation time has been calculated and drawn with a red line. The interpolation shows a linear behavior for the interpretation time of the scene regarding the frame number when the **Stepwise-bottom-up** rule and **Optimist** strategy are selected.

Frame Number	**Number of Objects**	**Interpretation Time (seconds)**
2	0	0.336
86	2	0.371
262	4	1.167
458	6	3.733
692	8	7.925
818	9	9.976
888	10	11.175
970	11	14.83
1022	12	18.162
1126	13	20.724
1220	14	23.817
1246	14	24.217
Average	**8.58**	**11.369**

Table 5.1: Results of Optimist - Stepwise-bottom-up Mode

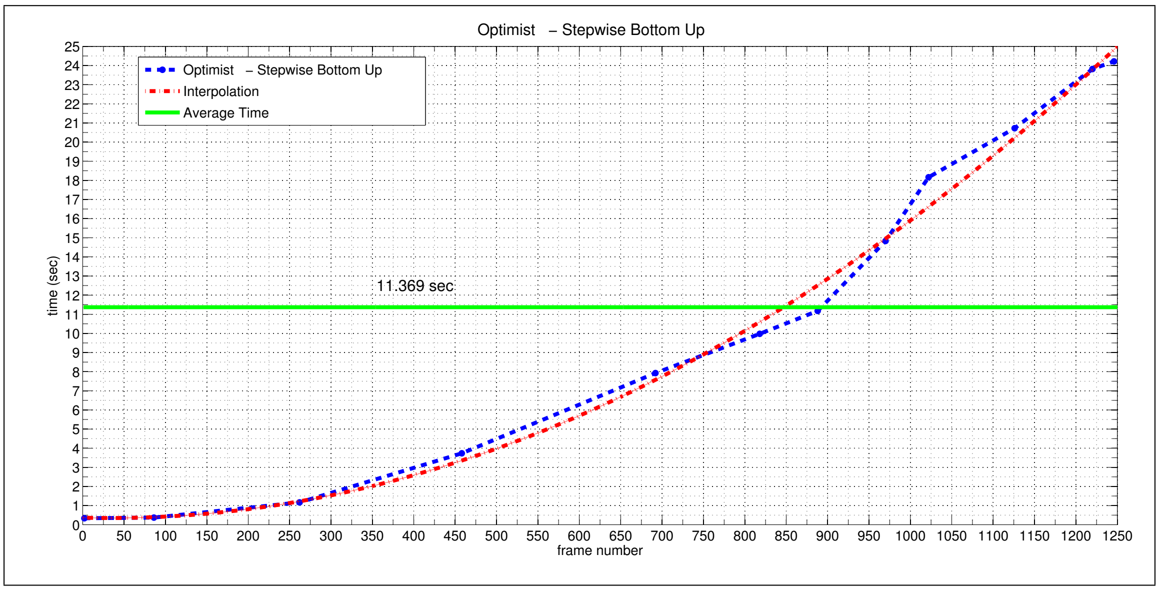
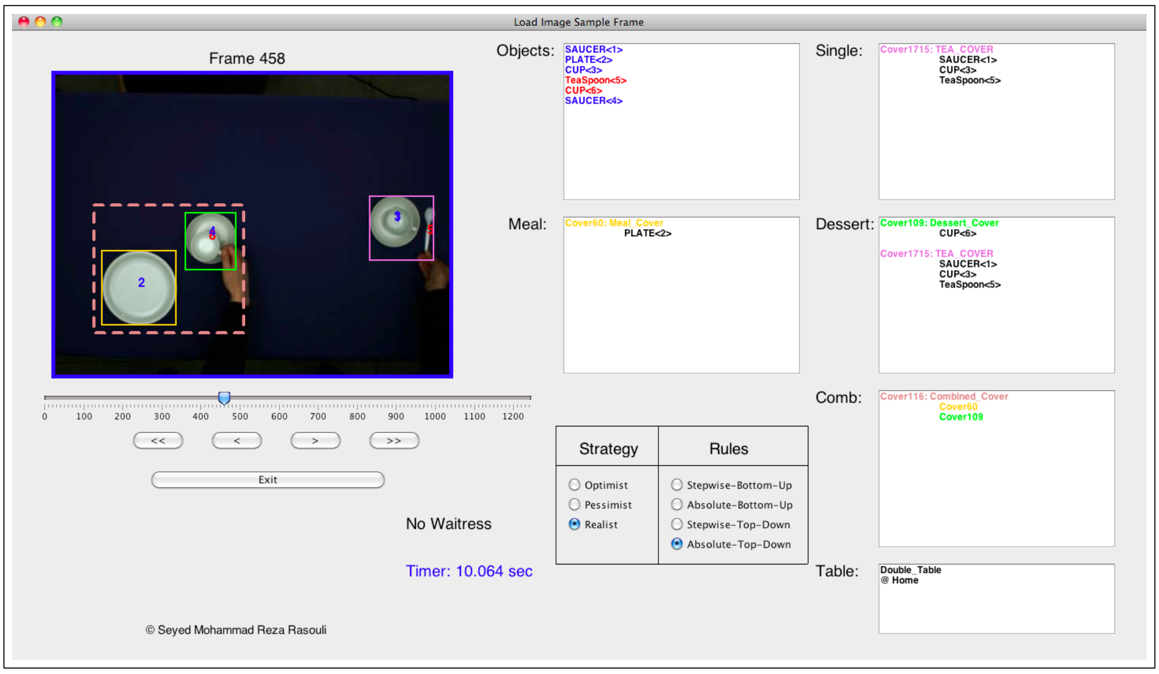


Figure 5.13: Graph of Optimist - Stepwise-bottom-up Mode



Objects:
SAUCER<1>
PLATE<2>
CUP<3>
TeaSpoon<5>
CUP<6>
SAUCER<4>

Single:
Cover1715: TEA_COVER
SAUCER<1>
CUP<3>
TeaSpoon<5>

Meal:
Cover60: Meal_Cover
PLATE<2>

Dessert:
Cover109: Dessert_Cover
CUP<6>

Cover1715: TEA_COVER
SAUCER<1>
CUP<3>
TeaSpoon<5>

Comb:
Cover116: Combined_Cover
Cover60
Cover109

Table:
Double_Table
@ Home

Strategy: Optimist [✓ Realist] Pessimist

Rules: Stepwise-Bottom-Up, Absolute-Bottom-Up, Stepwise-Top-Down [✓ Absolute-Top-Down]

No Waitress

Timer: 10.064 sec

© Seyed Mohammad Reza Rasouli

Figure 5.14: Rule: Absolute-top-down, Strategy: Realist - Frame 458

5.1.2 Rule: Absolute-top-down, Strategy: Realist

In this Section, the results of the framework in the situation where **Absolute-top-down** rule and **Realist** strategy[5] are selected, will be studied.

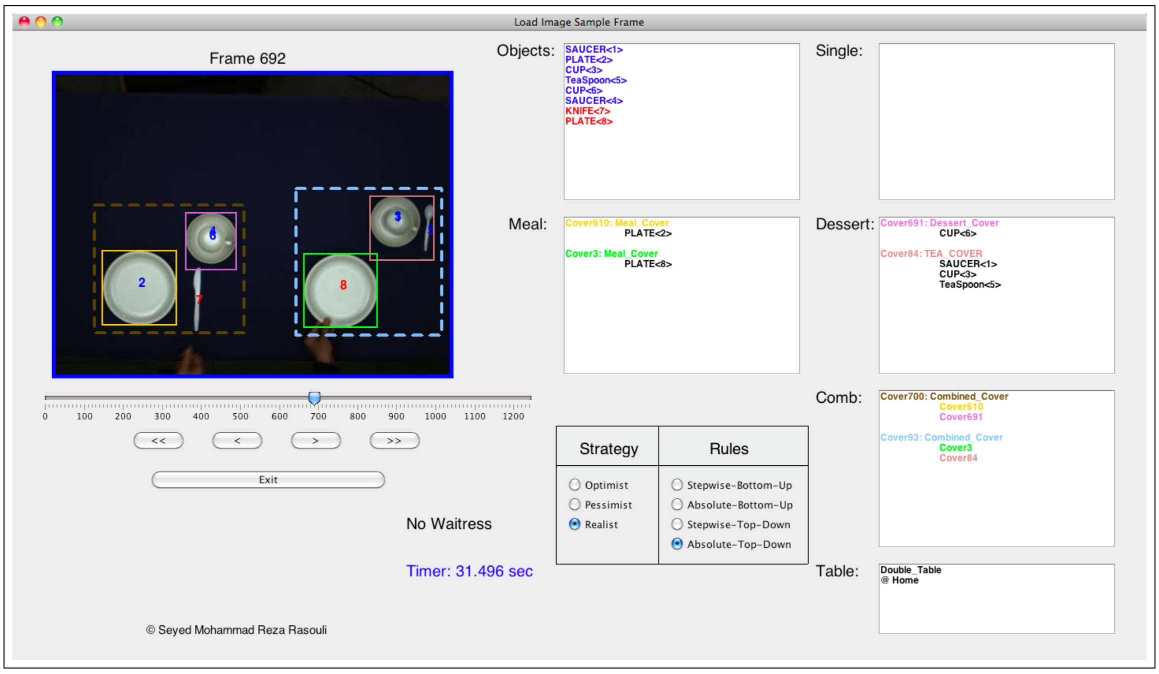
In the first three keyframes, there are no differences between the results of the interpretation compared to the previous Section, where the **Stepwise-bottom-up** rule and **Optimist** strategy have been selected.

The first difference occurs in frame 458 as shown in figure 5.14. In this frame, **SAUCER<4>** is not added to the instance of ***Dessert_Cover*** that contains the individual **CUP<6>**. As a result, this cover could not be specified more regarding its type. This is not the case in figure 5.4.

As it could be read from figures 5.14 and 5.4, the interpretation time in **ATDR** mode is almost 2.7 times longer than the interpretation time in **SBUO** mode. This is because the framework tries to explain a specific relation between two individuals in **SBUO** mode; there could be at most one explanation for the relation between these two individuals, which should also match the rules and should not make the ABox inconsistent. However, in **ATDR** mode, instead of explaining the scene by explaining the relations between specific primitive objects in the scene, the framework tries to build instances of higher aggregates using the primitive objects.

Since there could be a higher number of possible explanations in this mode compared to bottom-up mode, the framework asks the reasoner to deliver the best explanation if an instance of aggregate (in the form of a variable) is verifiable for the current scene; therefore, the reasoner must study all possible explanations, discard the inconsistent ones and choose the best explanation among the consistent

[5] **ATDR**


Load Image Sample Frame

Frame 692

Objects: SAUCER<1> PLATE<2> CUP<3> TeaSpoon<5> CUP<6> SAUCER<4> KNIFE<7> PLATE<8>

Single:

Meal:
Cover610: Meal_Cover
PLATE<2>
Cover3: Meal_Cover
PLATE<8>

Dessert:
Cover691: Dessert_Cover
CUP<6>
Cover84: TEA_COVER
SAUCER<1>
CUP<3>
TeaSpoon<5>

Comb:
Cover700: Combined_Cover
Cover610
Cover691
Cover93: Combined_Cover
Cover3
Cover84

Strategy: Optimist, Pessimist, [✓ Realist]

Rules: Stepwise-Bottom-Up, Absolute-Bottom-Up, Stepwise-Top-Down, [✓ Absolute-Top-Down]

Table: Double_Table @ Home

No Waitress

Timer: 31.496 sec

© Seyed Mohammad Reza Rasouli

Figure 5.15: Rule: Absolute-top-down, Strategy: Realist - Frame 692

explanations by using the scoring function introduced in Section 3.1.3. Thus, the reasoner and the framework need a longer time to analyze and process the results; therefore, the interpretation time in **ATDR** mode is longer although the **Realist** strategy is selected in this mode in which fewer numbers of assertions are present in the ABox compared to **SBUO** mode. In other words, the interpretation time is highly affected by the rule that is used for the interpretation.

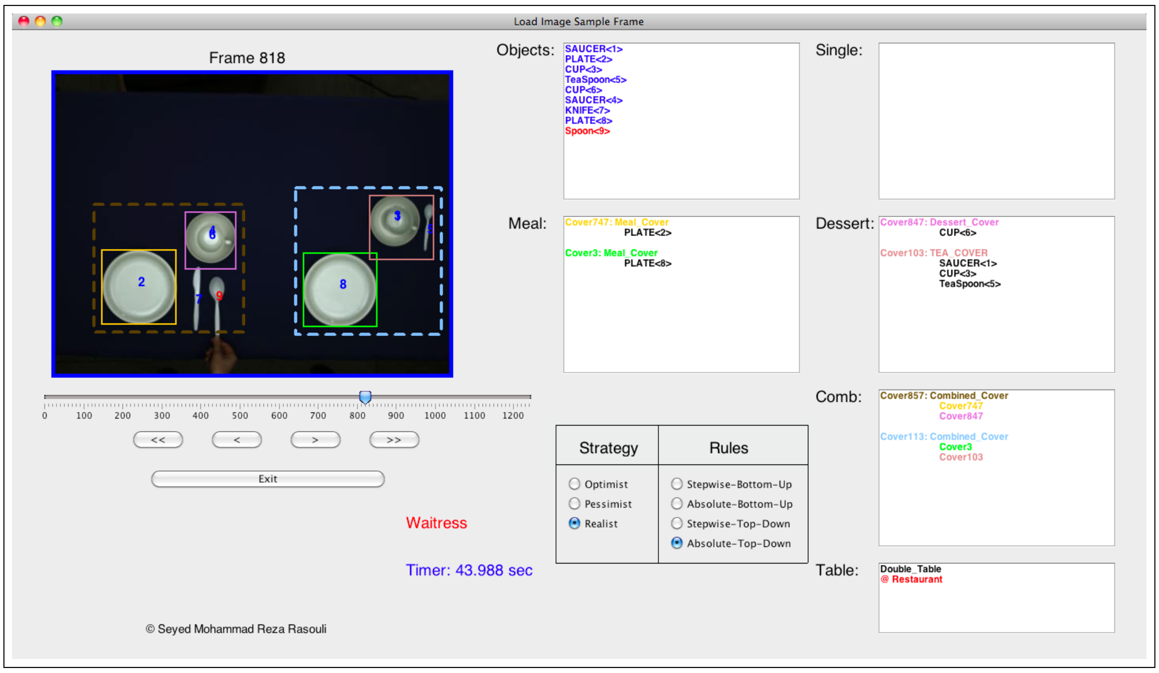
The effect of the **Realist** strategy could be seen in figure 5.15. **KNIFE<7>** has not been added to the instance of ***Meal_Cover*** that contains **PLATE<2>**. This is because there is still space for another instance of ***KNIFE*** between **PLATE<2>** and **KNIFE<7>**; therefore, the **Realist** strategy prevents the following role assertion to be added to the ABox: ***near-right-of(KNIFE<7>, PLATE<2>)***. As a result, the reasoner is not able to add **KNIFE<7>** to the instance of ***Meal_Cover*** because the mentioned relation is not present in the ABox[6].

In spite of the difference described above, the same number of instances of ***Combined_Cover*** are instantiated at the same locations in the scene under different modes.

Another effect of the **Realist** strategy is illustrated in figure 5.16 where **Spoon<9>** has been added to the scene but not to the instance of ***Meal_Cover*** on the left side of the scene because there is still room for another instance of ***Spoon*** next to **PLATE<2>**. In addition, the instance of ***Waitress***, which has been added to the scene, is considered; therefore, from this point, the framework knows that the table is located in a restaurant.

[6]It should be mentioned that ***near-right-of(KNIFE<7>, PLATE<2>)*** could be hypothesized for the current situation, but this results in a worse scoring result compared to the other explanations regarding the **Realist** strategy.

Although the bounding boxes of the instances of ***Meal_Cover*** and ***Dessert_Cover*** only contain the individuals that they own, the bounding boxes of the instances of ***Combined_Cover*** also contain the individuals that they do not own; however, regardless of the picture shown on the GUI, the framework and the reasoner both are aware of exact individuals under all instances of ***Cover***. This visual wrong effect is due to the fact that the bounding boxes of the instances of ***Combined_Cover*** are calculated using the tracking data. As a result, when the bounding boxes are drawn in the picture area, some individuals are under these bounding boxes but not under the instances of ***Combined_Cover***. This effect can be seen in figure 5.17 in which the old individual **Spoon<9>** has appeared in the bounding box of **Cover1219**, which is an instance of ***Combined_Cover***, although this instance does not own it.



Objects:
SAUCER<1>
PLATE<2>
CUP<3>
TeaSpoon<5>
CUP<6>
SAUCER<4>
KNIFE<7>
PLATE<8>
Spoon<9>

Single:

Meal:
Cover747: Meal_Cover
PLATE<2>
Cover3: Meal_Cover
PLATE<8>

Dessert:
Cover847: Dessert_Cover
CUP<6>
Cover103: TEA_COVER
SAUCER<1>
CUP<3>
TeaSpoon<5>

Comb:
Cover857: Combined_Cover
Cover747
Cover847
Cover113: Combined_Cover
Cover3
Cover103

Strategy: Optimist, Pessimist, [✓ Realist]

Rules: Stepwise-Bottom-Up, Absolute-Bottom-Up, Stepwise-Top-Down, [✓ Absolute-Top-Down]

Waitress

Timer: 43.988 sec

Table:
Double_Table
@ Restaurant

© Seyed Mohammad Reza Rasouli

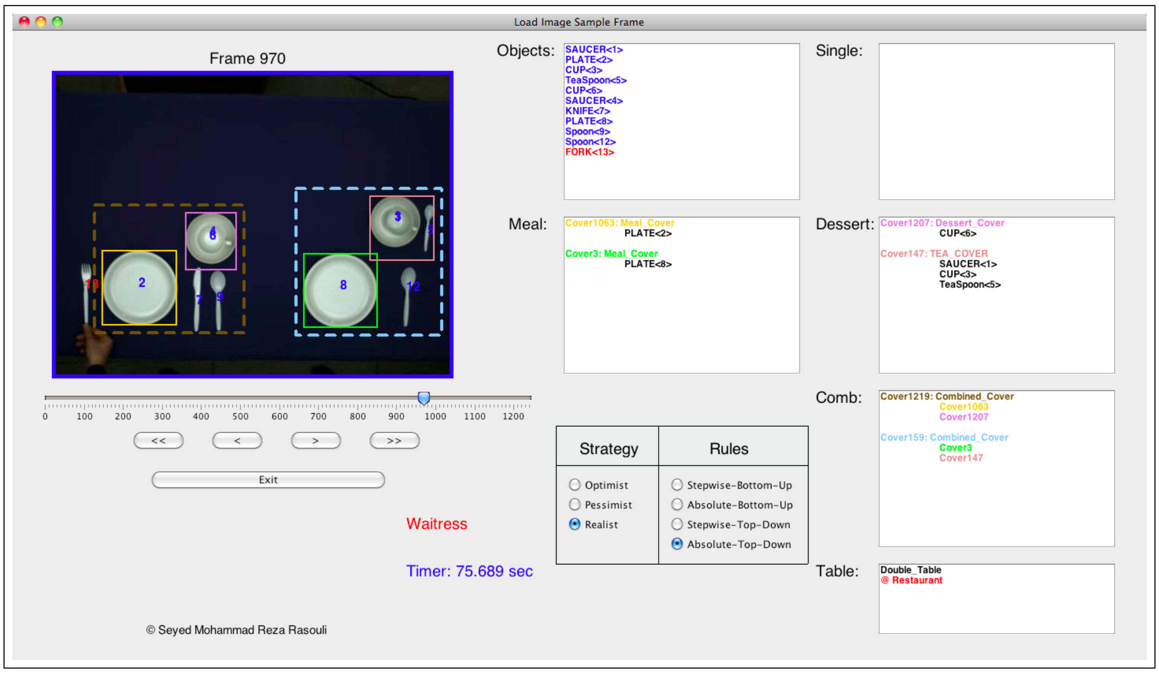
Figure 5.16: Rule: Absolute-top-down, Strategy: Realist - Frame 818

A main characteristic of the **Realist** strategy is shown in 5.18. The new instance of ***KNIFE***, which is called **KNIFE<14>**, is added to the scene. The **Realist** strategy perceives that there is no room[7] between **PLATE<8>** and **KNIFE<14>**; therefore, it assumes that **KNIFE<14>** is related to **PLATE<8>**. As a result, **KNIFE<14>** is added to **Cover1594**, an instance of ***Meal_Cover*** that contains **PLATE<8>**. If **KNIFE<14>** is relocated in the scene in a way that enough room for another instance of ***KNIFE*** to be placed closer next to **PLATE<8>** is made, **Cover1594** will be discarded.

The interpretation time in figure 5.18 for frame 1022 is **151.009 seconds**. This time is **8.31** times longer than the interpretation time of this frame in **SBUO** mode, which shows the difference between the **top-down** and **bottom-up** rules.

Finally, figure 5.19 shows the interpretation results for the last frame of the clip

[7]Horizontally



Objects:
SAUCER<1>
PLATE<2>
CUP<3>
TeaSpoon<5>
CUP<6>
SAUCER<4>
KNIFE<7>
PLATE<8>
Spoon<9>
Spoon<12>
FORK<13>

Single:

Meal:
Cover1063: Meal_Cover
PLATE<2>

Cover3: Meal_Cover
PLATE<8>

Dessert:
Cover1207: Dessert_Cover
CUP<6>

Cover147: TEA_COVER
SAUCER<1>
CUP<3>
TeaSpoon<5>

Comb:
Cover1219: Combined_Cover
Cover1063
Cover1207

Cover159: Combined_Cover
Cover3
Cover147

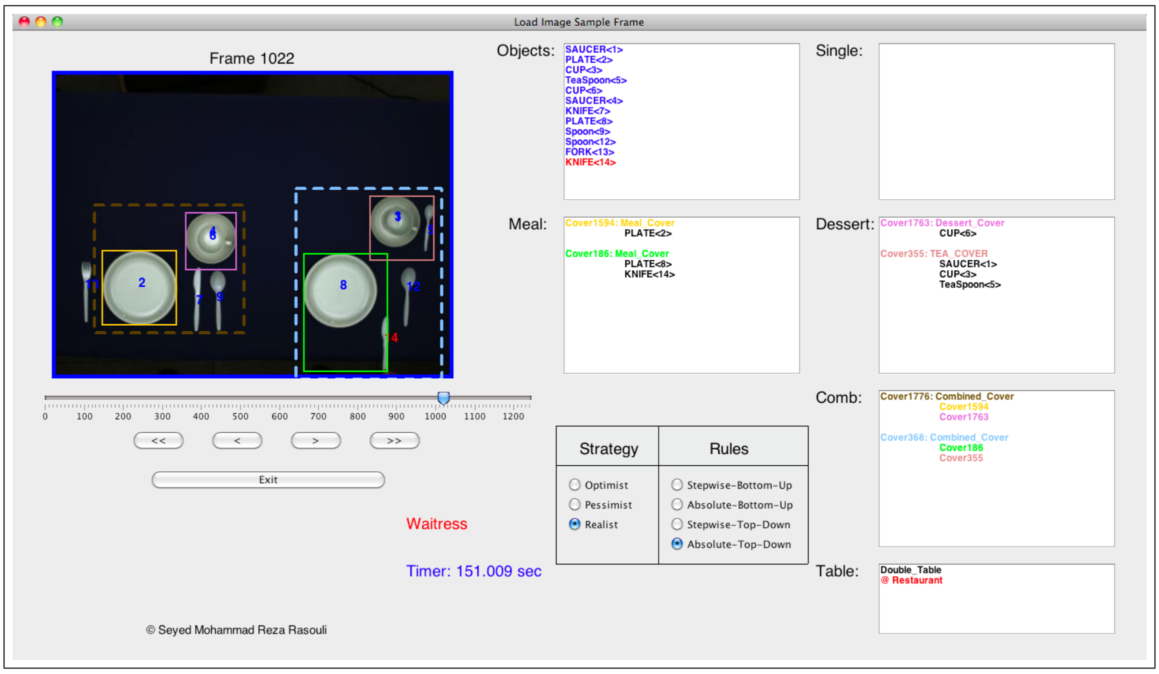
Table:
Double_Table
@ Restaurant

Strategy	Rules
Optimist	Stepwise-Bottom-Up
Pessimist	Absolute-Bottom-Up
[✓ Realist]	Stepwise-Top-Down
	[✓ Absolute-Top-Down]

Waitress

Timer: 75.689 sec

Figure 5.17: Rule: Absolute-top-down, Strategy: Realist - Frame 970



Objects:
SAUCER<1>
PLATE<2>
CUP<3>
TeaSpoon<5>
CUP<6>
SAUCER<4>
KNIFE<7>
PLATE<8>
Spoon<9>
Spoon<12>
FORK<13>
KNIFE<14>

Single:

Meal:
Cover1594: Meal_Cover
PLATE<2>

Cover186: Meal_Cover
PLATE<8>
KNIFE<14>

Dessert:
Cover1763: Dessert_Cover
CUP<6>

Cover355: TEA_COVER
SAUCER<1>
CUP<3>
TeaSpoon<5>

Comb:
Cover1776: Combined_Cover
Cover1594
Cover1763

Cover368: Combined_Cover
Cover186
Cover355

Table:
Double_Table
@ Restaurant

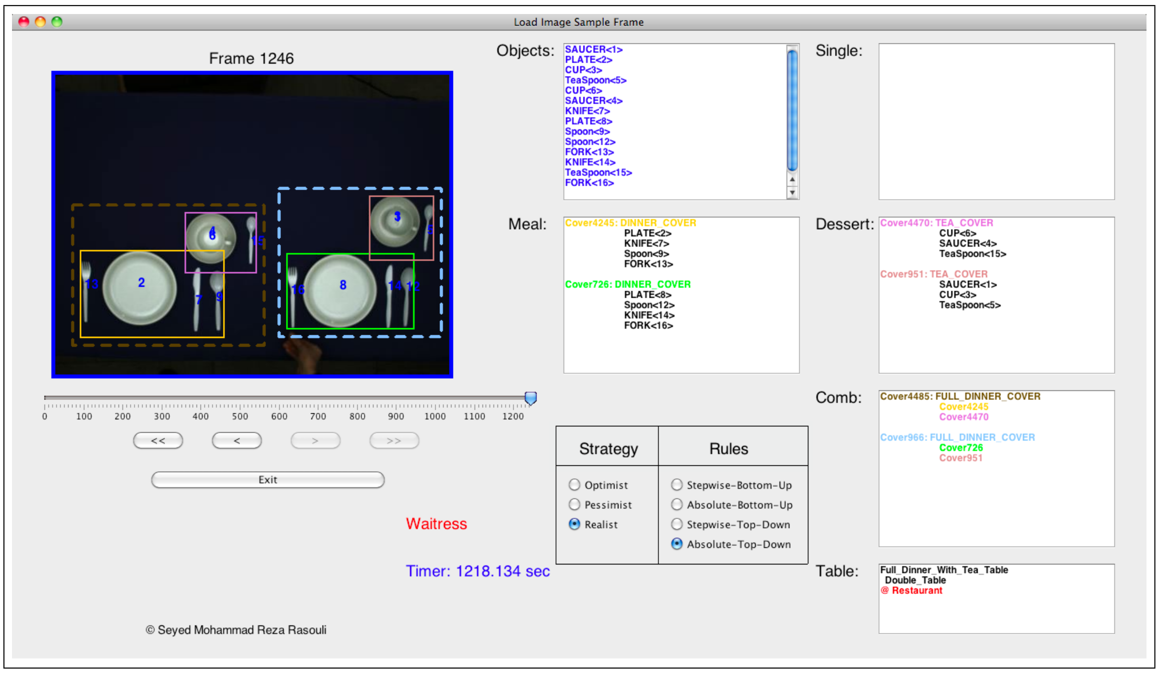
Strategy	Rules
Optimist	Stepwise-Bottom-Up
Pessimist	Absolute-Bottom-Up
[✓ Realist]	Stepwise-Top-Down
	[✓ Absolute-Top-Down]

Waitress

Timer: 151.009 sec

Figure 5.18: Rule: Absolute-top-down, Strategy: Realist - Frame 1022

in **ATDR** mode. As it is illustrated in this figure, the exact same results that the framework has delivered for this frame in **SBUO** mode have been reached for this frame in the **ATDR** mode. Since frame 1246 is the last frame of the clip, the **Realist** strategy considers all the relations in the scene as facts and interprets the



Timer: 1218.134 sec

Objects:
SAUCER<1>
PLATE<2>
CUP<3>
TeaSpoon<5>
CUP<6>
SAUCER<4>
KNIFE<7>
PLATE<8>
Spoon<9>
Spoon<12>
FORK<13>
KNIFE<14>
TeaSpoon<15>
FORK<16>

Single:

Meal:
Cover4245: DINNER_COVER
PLATE<2>
KNIFE<7>
Spoon<9>
FORK<13>

Cover726: DINNER_COVER
PLATE<8>
Spoon<12>
KNIFE<14>
FORK<16>

Dessert:
Cover4470: TEA_COVER
CUP<6>
SAUCER<4>
TeaSpoon<15>

Cover951: TEA_COVER
SAUCER<1>
CUP<3>
TeaSpoon<5>

Comb:
Cover4485: FULL_DINNER_COVER
Cover4245
Cover4470

Cover966: FULL_DINNER_COVER
Cover726
Cover951

Table:
Full_Dinner_With_Tea_Table
Double_Table
@ Restaurant

Strategy	Rules
Optimist	Stepwise-Bottom-Up
Pessimist	Absolute-Bottom-Up
[✓ Realist]	Stepwise-Top-Down
	[✓ Absolute-Top-Down]

Figure 5.19: Rule: Absolute-top-down, Strategy: Realist - Frame 1246

scene as if the **Optimist** strategy has been selected. However, the interpretation time for the scene in this frame in **ATDR** mode is **1218.134 seconds**, which is 50.3 longer than the interpretation time of this frame in **SBUO**. This is because of the rule that has been chosen and not the strategy.

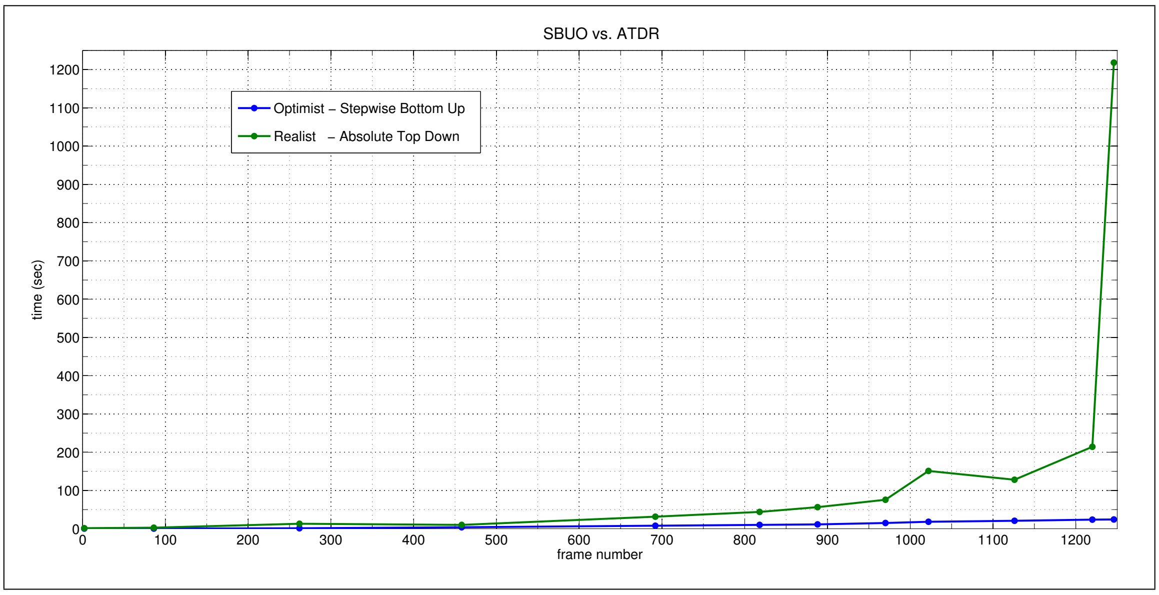


Figure 5.20: SBUO vs. ATDR

Table 5.2 shows the interpretation time of **ATDR** for all the keyframes. Furthermore, the graph in figure 5.20 shows the behavior of the interpretation time for **SBUO** and **ATDR** as the frame number increases. The interpretation time of **SBUO** is shown with a blue line and the interpretation time of **ATDR** is shown with a green line.

Frame Number	**Number of Objects**	**Interpretation Time (seconds)**
2	0	1.455
86	2	2.762
262	4	12.999
458	6	10.064
692	8	31.496
818	9	43.988
888	10	56.373
970	11	75.689
1022	12	151.009
1126	13	128.062
1220	14	213.984
1246	14	1218.134
Average	**8.58**	**162.168**

Table 5.2: Results of Realist - Absolute-top-down Mode

5.2 Comparison

In this Section, a comparison is made between the results of the test in different situations. As mentioned before, the three strategies and the four groups of rules result in twelve combinations. Two of these combinations have been studied in detail in the previous Sections. The results for each rule group and each strategy are shown and discussed in the following Sections.

5.2.1 Rules

As discussed before, each group of rules could be run in three different modes according to the strategies. The best interpretation time average is achieved when the **Stepwise-bottom-up** rule is chosen because the framework and the reasoner only try to explain a specific relation between two individuals in this situation. In other words, they do not have to consider other primitive objects that are present on the scene. The disadvantage of that many explanations for the current scene according to the primitive objects could be found, which many of them might contradict each other or become the same score. As a result, the framework must study each explanation to choose the best answer among them. The ***only-best-p*** feature of the ***retrieve-with-explanation*** command of RacerPro delivers the explanations that have become the highest score. Since it is possible that more than one explanation becomes the highest score, it is the job of the framework to study these explanations and chooses the best one of them. In other words, the probability that more than one explanation is achieved after interpreting the relations between each two individual that are present in the scene is higher than getting a single explanation.

The opposite situation is observed if the **top-down** rules are applied. Since in this work, different rules are defined for each aggregate, the reasoner needs a longer time to apply different rules for finding an instance of an aggregate. It is possible again that the reasoner finds more than one explanation for the current situation in the scene. However, since the rules have different number of tuples in their body, it is most likely that one explanation becomes the highest score. This is illustrated by the following example.

The following situation is considered:

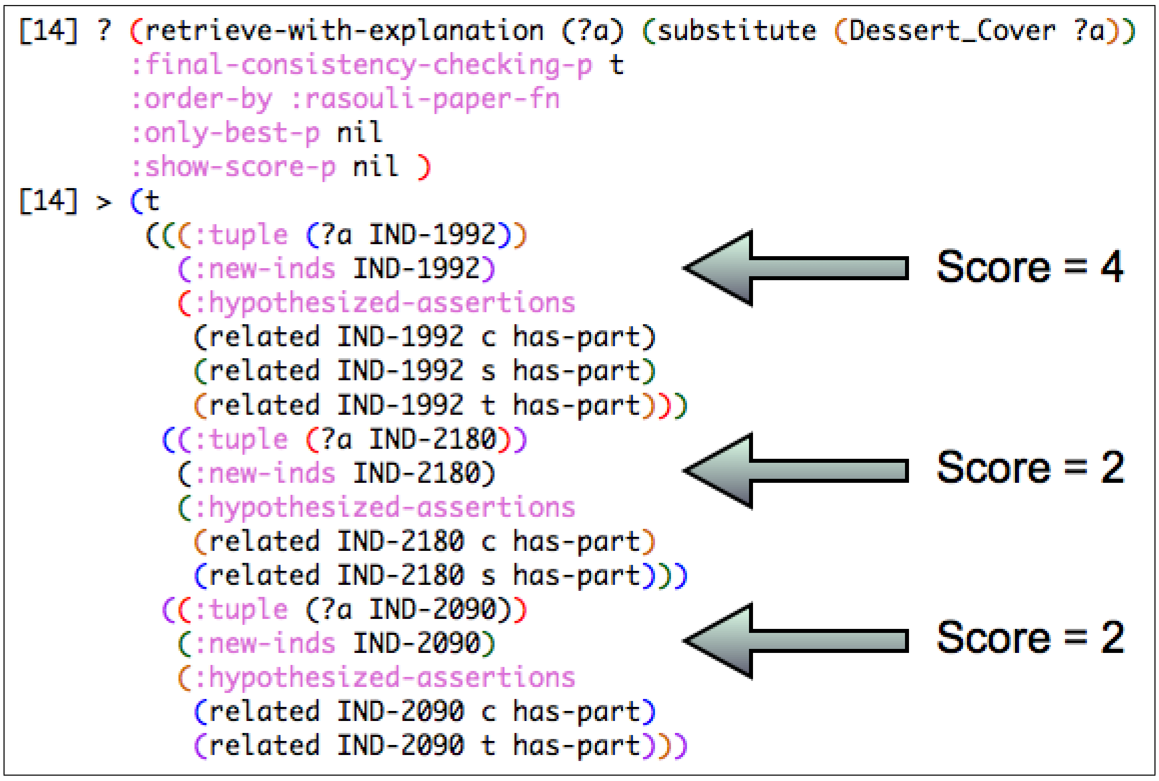
*CUP(**C**), SAUCER(**S**), TeaSpoon(**T**),*
*on-same-position-of(**C**,**S**), on-same-position-of(**T**,**S**),*
*near-right-of(**T**,**C**), near-left-of(**C**,**T**),*

The reasoner is asked to retrieve and explain the instances of ***Dessert_Cover*** with the help of the rules shown in 3.13 and 3.14. Figure 5.21 shows the result of the query where three different explanations could be found for this situation. As it could be seen, the first explanation has a higher score than the other explanations. If the ***only-best-p*** feature of the query was enabled, when the command was run, the first explanation would have only been delivered by the reasoner; however, all possible explanations that two of them are shown here additionally should be calculated in the reasoner; therefore, the reasoner needs a longer time to consider all the explanations and deliver the best results. If more than one explanation are delivered for the scene, the framework most likely can use all of them as explanations, since based on the definitions in the TBox, these explanations are independent of each other.

5.2.1.1 Stepwise-bottom-up

Figure 5.22 and table 5.3 show the results of applying the **Stepwise-bottom-up** rules for the interpretation of the clip. The **Optimist** strategy shows a linear behavior in this mode. The **Pessimist** and **Realist** strategies show almost the same behavior. The interesting point is that in frame 1220, the interpretation time of the **Realist** strategy is less than the interpretation time of the **Pessimist** strategy. This is because of the less numbers of assertions that the **Realist** strategy causes compared to **Pessimist** strategy.

The interpretation time average of the **Realist** strategy is almost 47.7% of the **Optimist** strategy and 127.3% of the **Pessimist** strategy. Since between there is a tradeoff between the amount of the data that the framework and reasoner explain and the time that they need for the interpretation, the **Realist** strategy shows a better **performance** if **Stepwise-bottom-up** rules are chosen.



```
[14] ? (retrieve-with-explanation (?a) (substitute (Dessert_Cover ?a))
       :final-consistency-checking-p t
       :order-by :rasouli-paper-fn
       :only-best-p nil
       :show-score-p nil )
[14] > (t
        (((:tuple (?a IND-1992))
          (:new-inds IND-1992)
          (:hypothesized-assertions
           (related IND-1992 c has-part)
           (related IND-1992 s has-part)
           (related IND-1992 t has-part)))
         ((:tuple (?a IND-2180))
          (:new-inds IND-2180)
          (:hypothesized-assertions
           (related IND-2180 c has-part)
           (related IND-2180 s has-part)))
         ((:tuple (?a IND-2090))
          (:new-inds IND-2090)
          (:hypothesized-assertions
           (related IND-2090 c has-part)
           (related IND-2090 t has-part)))
```

Figure 5.21: Different Instances of ***Dessert_Cover*** explaining the same situation

		Interpretation Time (seconds)		
		Strategy		
Frame	**# Objects**	**Optimist**	**Pessimist**	**Realist**
2	0	0.336	0.018	0.171
86	2	0.371	0.085	0.317
262	4	1.167	0.326	0.644
458	6	3.733	0.671	2.152
692	8	7.925	1.391	4.280
818	9	9.976	2.203	4.760
888	10	11.175	2.820	5.157
970	11	14.830	3.823	5.698
1022	12	18.162	4.871	6.249
1126	13	20.724	5.932	6.649
1220	14	23.817	7.615	7.163
1246	14	24.217	21.309	21.774
Average	8.58	11.369	4.255	5.418

Table 5.3: Results of Stepwise-bottom-up Rules

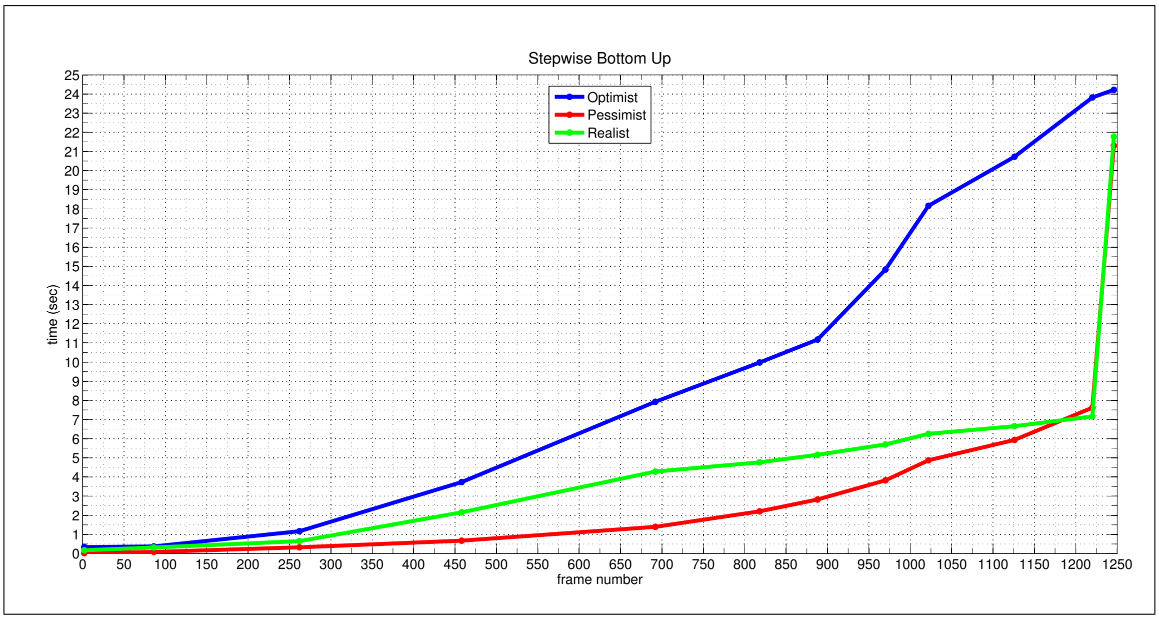


Figure 5.22: Graph of Stepwise-bottom-up Rules

5.2.1.2 Absolute-bottom-up

As it can be seen in figure 5.23 and table 5.4, the **Absolute-bottom-up** rules have a worse behavior compared to **Stepwise-bottom-up** rules regardless of the strategy that has been chosen.

Since by **Absolute-bottom-up** rules only the primitive objects are used to build the aggregates, no matter in which level these aggregate are located, the reasoner needs a longer time to achieve the aggregates (if possible) in the higher levels by only using the primitive objects compared to **Stepwise-bottom-up** rules where each aggregate is built by the aggregates in the below level an only the aggregates in the first level are built using the primitive objects.

By comparing the two variant of **bottom-up** rules, **Stepwise-bottom-up** rules show a better **performance** than **Absolute-bottom-up** rules.

5.2.1.3 Stepwise-top-down

The behavior of **Stepwise-top-down** rules is shown in figure 5.24 and 5.5. The curve of the **Realist** strategy shows that the interpretation time of this strategy has increased and moved towards the curve of **Optimist** strategy, when the **Stepwise-top-down** rules have been chosen.

5.2.1.4 Absolute-top-down

The results of the interpretation of the scene using **Absolute-top-down** are shown in figure 5.25 and 5.6. As it could be seen, the curve of the **Realist** strategy is almost near to the curve of the **Pessimist** strategy. Furthermore, the curve of the

		Interpretation Time (seconds)		
		Strategy		
Frame	**# Objects**	**Optimist**	**Pessimist**	**Realist**
2	0	0.204	0.606	0.138
86	2	0.34	0.9	0.285
262	4	2.764	0.342	0.56
458	6	39.686	0.843	13.758
692	8	158.912	1.658	80.517
818	9	239.896	2.365	100.602
888	10	367.308	3.094	125.429
970	11	558.989	4.276	155.359
1022	12	639.769	5.584	276.318
1126	13	868.087	6.717	231.817
1220	14	1193.036	8.735	294.838
1246	14	1175.135	1250.838	1339.154
Average	8.58	437.011	107.163	218.231

Table 5.4: Results of Absolute-bottom-up Rules

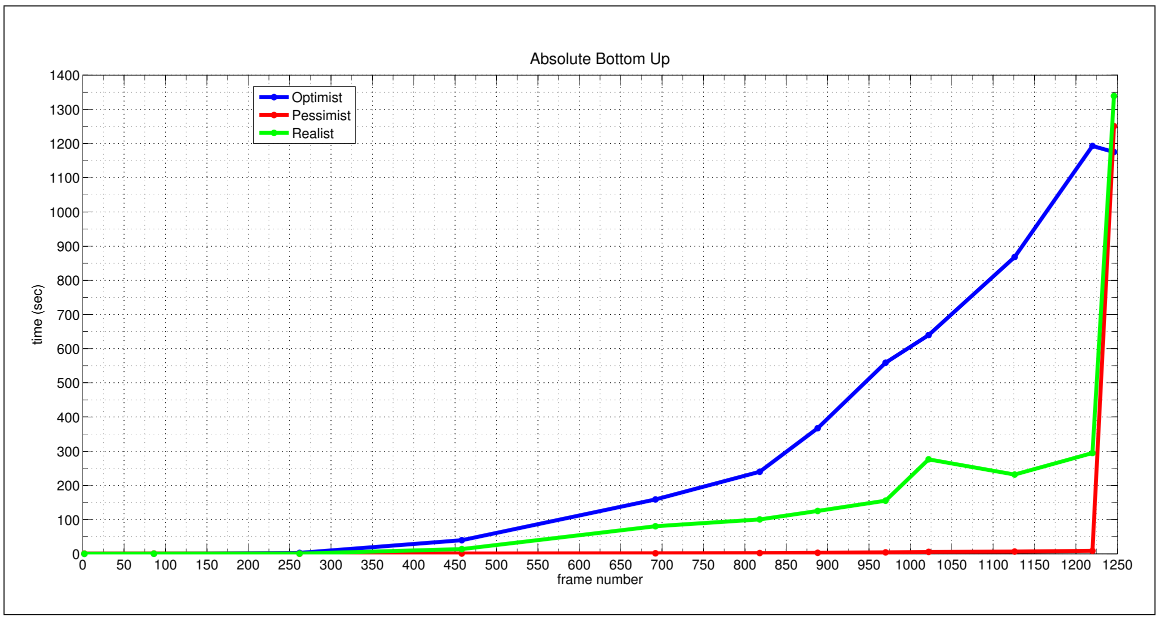


Figure 5.23: Graph of Absolute-bottom-up Rules

Optimist strategy has decreased compared to the **Optimist** curve in **Stepwise-top-down** and even compared to **Absolute-bottom-up**.

This happens because in **Absolute-bottom-up** mode, the framework and the reasoner look for the existence of instances of the aggregates in the highest level. If such an instance is found and built, the instances of the aggregates in the levels between will be instantiated automatically, whereas in the **Stepwise-bottom-up**

		Interpretation Time (seconds)		
		Strategy		
Frame	**# Objects**	**Optimist**	**Pessimist**	**Realist**
2	0	2.925	0.005	1.235
86	2	2.347	0.052	2.629
262	4	6.708	0.245	7.079
458	6	28.601	0.677	22.821
692	8	93.2	1.41	67.042
818	9	146.017	2.068	98.199
888	10	237.055	2.683	140.64
970	11	354.426	3.729	199.327
1022	12	529.205	4.786	293.224
1126	13	840.595	5.587	391.053
1220	14	1103.341	7.039	517.65
1246	14	1118.579	1185.311	1132.541
Average	8.58	371.917	101.133	239.453

Table 5.5: Results of Stepwise-top-down Rules

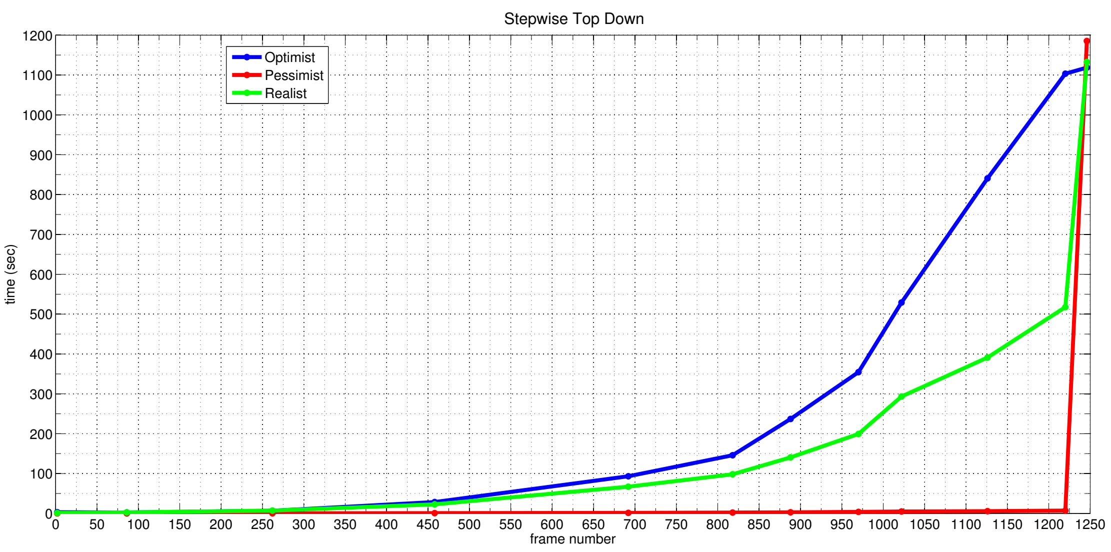


Figure 5.24: Graph of Stepwise-top-down Rules

mode, the aggregates in the lowest level are instantiated first (if existing) and by looking for the aggregates in the higher level, the current level will be revisited again; therefore, the interpretation time increases in the **Stepwise-bottom-up** mode. In other words, the interpretation time is lower for **Absolute-bottom-up** rules; thus, this group of rules shows a better **performance** by the interpretation.

		Interpretation Time (seconds)		
		Strategy		
Frame	**# Objects**	**Optimist**	**Pessimist**	**Realist**
2	0	1.451	0.01	1.455
86	2	2.418	0.071	2.762
262	4	4.01	0.258	12.999
458	6	14.277	0.611	10.064
692	8	63.258	1.33	31.496
818	9	101.942	1.963	43.988
888	10	171.292	2.495	56.373
970	11	258.96	3.692	75.689
1022	12	451.81	4.832	151.009
1126	13	609.607	5.659	128.062
1220	14	920.009	6.851	213.984
1246	14	916.667	1029.03	1218.134
Average	8.58	292.975	88.067	162.168

Table 5.6: Results of Absolute-top-down Rules

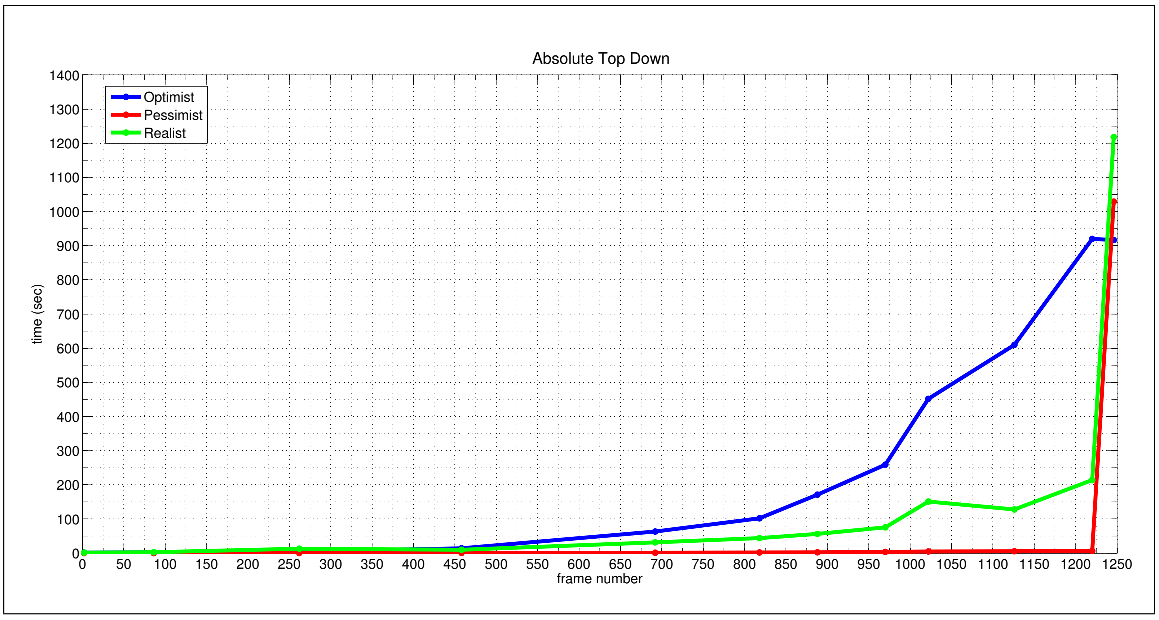


Figure 5.25: Graph of Absolute-top-down Rules

5.2.2 Strategies

In general, regardless of the rule that has been chosen, the **Pessimist** strategy has the lowest interpretation time average for each group of rules; however, as the name of this strategy suggests, the amount of the data that will be delivered by the framework during the clip is very low[8]. In other words, the user has to wait until the last frame of the movie to get the interpretation results of the framework.

The **Realist** strategy has a better interpretation time average than the **Optimist** strategy and a worse interpretation time average than the **Pessimist** strategy.

Finally, the **Optimist** strategy explains and generates the highest amount of data compared to the other two strategies, although it has a higher interpretation time average (almost 1.8 times longer than **Realist** strategy and 3.44 times longer than the **Pessimist** strategy).

Although the **Pessimist** strategy has the lowest interpretation time average regarding to the other two strategies, the amount of data and explanations that this strategy delivers is very low and the user has to wait until the end of the movie to become the explanations of the scene. As a result, the **Realist** strategy delivers better information for the user, even if its interpretation time is longer than the **Pessimist** strategy. Nevertheless, the user can rely on the explanations that the **Realist** strategy delivers. Finally, the textbfOptimist delivers the highest amount of data and explanations; however, not all the data that this strategy delivers is necessarily correct because it does not consider that the situation might change later in the clip. On the other hand, its interpretation time is longer than the other two strategies.

In the following Subsections, the behavior of each strategy according to the rules that have been chosen is shown and discussed.

5.2.2.1 Optimist

The graph in figure 5.26 shows the different behavior of the rules in case of choosing the **Optimist** strategy. As it can be seen, the interpretation time in

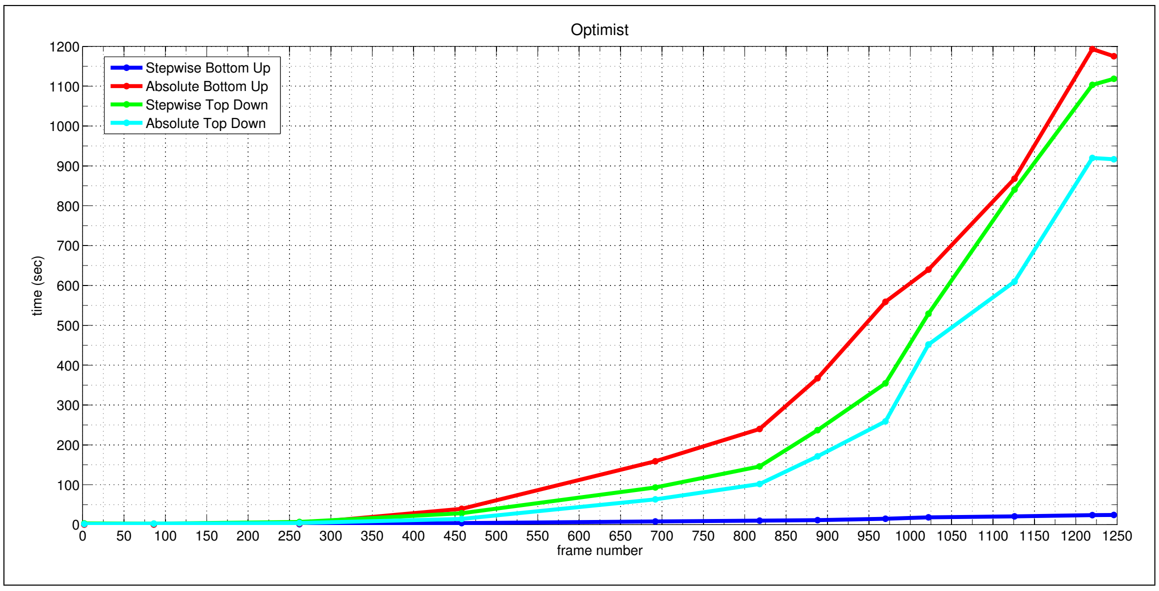


Figure 5.26: Graph of Optimist Strategy

[8]The data includes only the primitive objects and their low-level relations regarding one another.

SBUP mode is extremely lower than other modes. The curves of the other three modes have almost the same shape; however, as it is shown, the **Absolute-top-down** rule delivers the results in a shorter time compared to the red curve and the green curve.

Furthermore, it is observed that the sudden increase of the interpretation time for the **ABUO**, **ATDO** and **STDO** rules is almost by frame 800. This is the point where half of the objects that are supposed to be added to the scene until the end of the clip has already arrived at the scene.

		Interpretation Time (seconds)			
		Rule			
Frame	**# Objects**	**SBU**	**ABU**	**STD**	**ATD**
2	0	0.336	0.204	2.925	1.451
86	2	0.371	0.34	2.347	2.418
262	4	1.167	2.764	6.708	4.01
458	6	3.733	39.686	28.601	14.277
692	8	7.925	158.912	93.2	63.258
818	9	9.976	239.896	146.017	101.942
888	10	11.175	367.308	237.055	171.292
970	11	14.83	558.989	354.426	258.96
1022	12	18.162	639.769	529.205	451.81
1126	13	20.724	868.087	840.595	609.607
1220	14	23.817	1193.036	1103.341	920.009
1246	14	24.217	1175.135	1118.579	916.667
Average	8.58	11.369	437.011	371.917	292.975

Table 5.7: Result of Optimist Strategy - Interpretation Time (seconds)

5.2.2.2 Pessimist

If the **Pessimist** strategy is chosen, the **Stepwise-bottom-up** again shows a better interpretation time average; nevertheless, all the rules show the same behavior before reaching the last frame of the clip, as shown in figure 5.27. The pessimist strategy could be studied better in terms of the amount of time that the framework needs to start the interpretation from the first frame and finish it in the last frame.

For the **Stepwise-bottom-up** rule, the entire runtime of the clip is **51.064 seconds**. Compared to the interpretation time of the entire clip for **Absolute-top-down** rule, which is **1056.8 seconds**, the framework delivers the results of the entire interpretation in **ATDP** almost 96% lower than in **SBUP**.

5.2.2.3 Realist

The graph in figure 5.28 shows that except for the **Stepwise-bottom-up** rule, the corresponding curves for the other three rules have almost the same shape;

		Interpretation Time (seconds)			
		Rule			
Frame	**# Objects**	**SBU**	**ABU**	**STD**	**ATD**
2	0	0.018	0.606	0.005	0.01
86	2	0.085	0.9	0.052	0.071
262	4	0.326	0.342	0.245	0.258
458	6	0.671	0.843	0.677	0.611
692	8	1.391	1.658	1.41	1.33
818	9	2.203	2.365	2.068	1.963
888	10	2.82	3.094	2.683	2.495
970	11	3.823	4.276	3.729	3.692
1022	12	4.871	5.584	4.786	4.832
1126	13	5.932	6.717	5.587	5.659
1220	14	7.615	8.735	7.039	6.851
1246	14	21.309	1250.838	1185.311	1029.03
Average	8.58	4.255	107.163	101.133	88.067

Table 5.8: Results of Pessimist Strategy - Pessimist Interpretation Time (seconds)

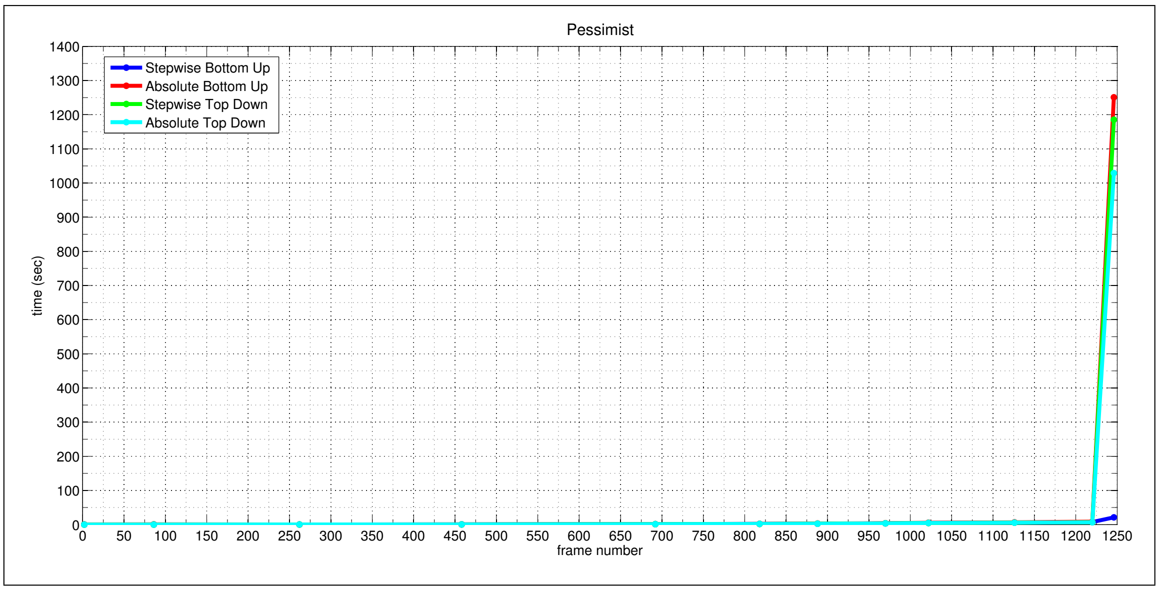


Figure 5.27: Graph of Pessimist Strategy

however, the **Absolute-top-down** rule has a better interpretation time average among these three rules.

A decrease in the interpretation time is observed for the **Absolute** rules between frame 1022 and frame 1126. Frame 1126 is the point where a missing evidence, (**TeaSpoon<15>**, arrives at the scene. The result of presence of this individual in the scene of this frame causes that both of instances of ***Dessert_Cover***, which now have all the individuals that they can own, get the same highest score; therefore, the reasoner and the framework should not process the situation more to interpret the scene, while in the previous keyframe (frame 1022), the absence of **TeaSpoon<15>** caused a longer interpretation time. In other words, by the arrival of the missing objects at the scene, the interpretation time could even become shorter[9].

		Interpretation Time (seconds)			
		Rule			
Frame	**# Objects**	**SBU**	**ABU**	**STD**	**ATD**
2	0	0.171	0.138	1.235	1.455
86	2	0.317	0.285	2.629	2.762
262	4	0.644	0.56	7.079	12.999
458	6	2.152	13.758	22.821	10.064
692	8	4.28	80.517	67.042	31.496
818	9	4.76	100.602	98.199	43.988
888	10	5.157	125.429	140.64	56.373
970	11	5.698	155.359	199.327	75.689
1022	12	6.249	276.318	293.224	151.009
1126	13	6.649	231.817	391.053	128.062
1220	14	7.163	294.838	517.65	213.984
1246	14	21.774	1339.154	1132.541	1218.134
Average	8.58	5.418	218.231	239.453	162.168

Table 5.9: Results of Realist Strategy - Interpretation Time (seconds)

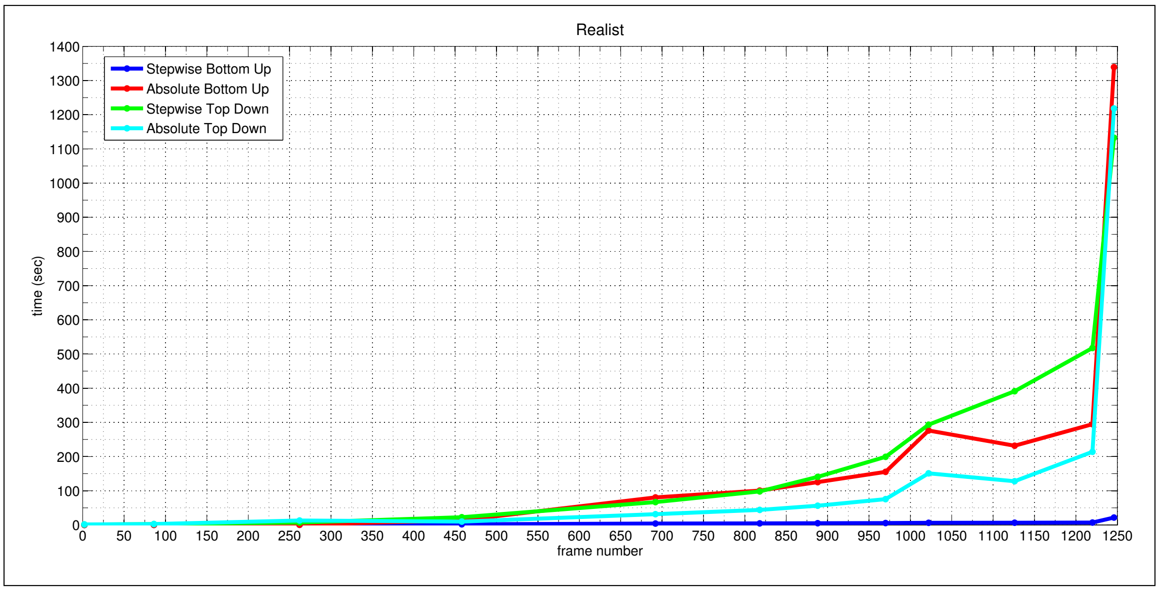


Figure 5.28: Graph of Realist Strategy

[9]Depending on the rule that has been chosen.

The graph in figure 5.29 shows all the curves of the twelve modes that have been introduced in this work. In the next Chapter, this graph is studied in terms of a scoring function determine the performance of each interpretation mode. The average of interpretation time of each mode is calculated and shown in table 5.10.

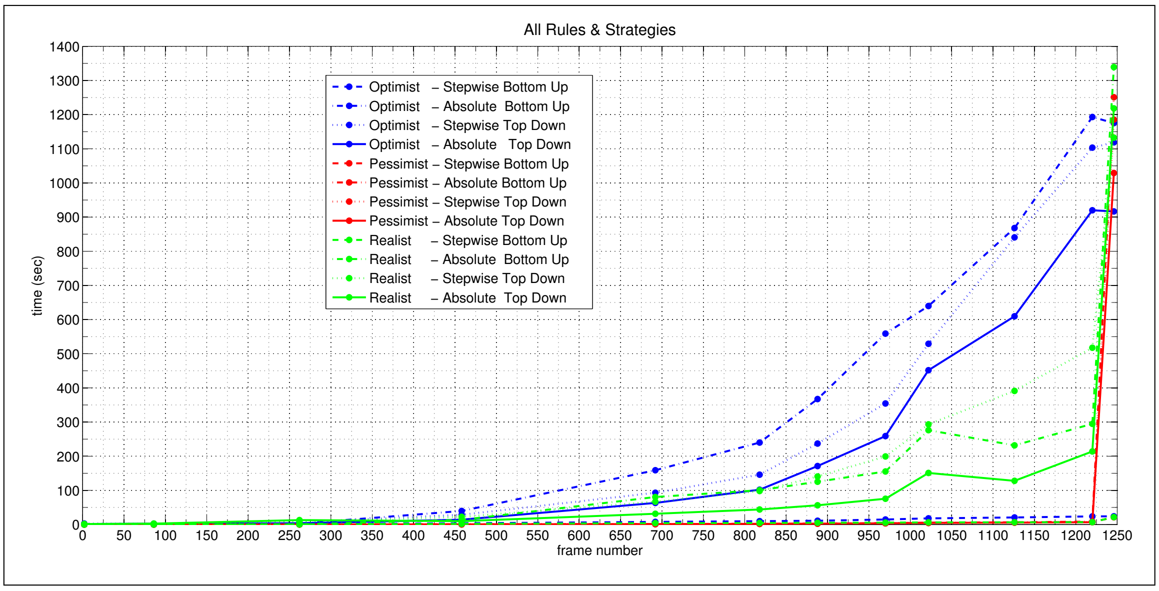


Figure 5.29: Graph of All Rules & Strategies

		Rule				
		SBU	**ABU**	**STD**	**ATD**	**Average**
Strategy	**Optimist**	11.369	437.011	371.917	292.975	278.318
	Pessimist	4.255	107.163	101.133	88.067	75.155
	Realist	5.418	218.231	239.453	162.168	156.318
	Average	7.014	254.135	237.501	181.070	

Table 5.10: Average Interpretation Time Summary (seconds)

Chapter 6

Validation and Verification

This Chapter focuses on the performance of the framework and the accuracy of the result that the framework delivers. As discussed in the previous Section, regardless of the strategy and the group of rule that have been selected by the user, there exist a tradeoff between the interpretation time and the amount of the data that could be explained; therefore, to present and compare the quality of the results, a scoring function is introduced in this Chapter. Before describing the characteristics of the scoring function, three measurements are described in the first Section. In the second Section, the scoring function is introduced and with its help, the results of the interpretation are evaluated.

6.1 Preliminaries

The following measurements are introduced in [22] and discussed and studied in [19] and [3].

precision: It shows the ratio of the documents that are correct and relevant in the set of the results and could be measured as follows:

$$\mathbf{P} = \frac{(\textbf{relevant documents}) \cap (\textbf{retrieved documents})}{(\textbf{retrieved documents})} \tag{6.1}$$

recall: Measures the ratio of the relevant documents in the result set to all the relevant documents that are available and could be calculated as follows:

$$\mathbf{R} = \frac{(\textbf{relevant documents}) \cap (\textbf{retrieved documents})}{(\textbf{relevant documents})} \tag{6.2}$$

F-Measure: Combines **precision** and **recall** to a single number that is the harmonic mean of these two measurements [22] and is calculated by using the following formula:

$$\mathbf{F_1} = \frac{\mathbf{2PR}}{(\mathbf{P}+\mathbf{R})} \tag{6.3}$$

F-Measure is a number between 0 and 1. The closer this number is to one, the better the quality of the system is.

To illustrate the meaning of the measurements introduced in 6.1, 6.2 and 6.3, the following example is considered:

There is a collection of 80 documents which 50 of them are relevant documents. If a system retrieves 40 documents, which only 25 of them are relevant, the precision, recall and F_1[1] are calculated as follows:

$$\mathbf{P = \frac{25}{40} = 0.625, \quad R = \frac{25}{50} = 0.5, \;\; F_1 = \frac{2*0.625*0.5}{(0.625+0.5)} \simeq 0.56}$$

6.2 Scoring Function

In this Section, a scoring function is introduced to show the trade-off between the interpretation time of a frame and the quality of the explanations that the framework delivers. The main characteristics of this scoring function are the interpretation time, the **recall** rate and the **precision** rate and F_1 rate of the results of the interpretation of each frame.

A short interpretation time of a frame could be a desire goal at first glance; however, the amount of the data that could be explained correctly differs based on the interpretation time. The lower the interpretation time is and the higher the amount of the data that are explained, the better is the result of the framework according to the strategy and rule that have been selected. The F_1 is a suitable rate to measure the quality of the result that the framework delivers as explanations of the scene.

In the context of table scenario, the **precision** rate indicates how relevant are the explanations that the framework and the reasoner deliver for the current scene and the **recall** rate indicates which proportion of the relevant and correct explanations in the scene are retrieved and found by the framework and the reasoner. F_1 combines these two rates to show the quality of the results. By calculating the F_1 rate and interpretation time of a frame, a scoring rate for the interpretation of **a frame** could be defined as follows:

$$\mathbf{S_k = \frac{F_1(k)}{t_k}} \tag{6.4}$$

where $\mathbf{k}$ is the frame number, $\mathbf{F_1(k)}$ is the F_1 rate of the explanations delivered by the framework and the reasoner for this frame, $\mathbf{t_k}$ is the interpretation time and finally, $\mathbf{S_k}$ is the interpretation score of the frame $\mathbf{k}$.

To evaluate the performance of the framework and the reasoner for the entire clip and summarize the scores of all keyframes, the following score function for the **entire** interpretation is defined as follows:

$$\mathbf{S = \frac{\sum_{i=1}^{n} S_i}{n}} \tag{6.5}$$

[1]The F-Measure is also called F_1 score.

where $\mathbf{S_i}$ is the score of frame **i**, **n** is the number of keyframes and **S** is the score of the entire interpretation. The scoring function **S** is the average of the scores of the keyframes. The closer the average is to 1, the better the result of interpretation for the strategy and rule, which have been selected, is.

The combinations of the rules and strategies could now be evaluated using the scoring function defined above6.5.

Table 6.1 and figures 6.1 and 6.2 show the score of each combination of rules and strategies.

		Rule				
		SBU	**ABU**	**STD**	**ATD**	**Average**
Strategy	**Optimist**	0.3991	0.3043	0.0578	0.0697	0.2077
	Pessimist	0.00427	0.00007	0.00007	0.00009	0.00113
	Realist	0.5518	0.4686	0.0537	0.0561	0.2826
	Average	0.3184	0.2577	0.0372	0.042	

Table 6.1: Score of All Rules and Strategies

As it can be seen, the best score is achieved by choosing the **Realist** strategy and **Stepwise-bottom-up** rules. Regardless of the rules that have been chosen, the **Pessimist** strategy becomes the worse score compared to the other two strategies. The **Realist** strategy shows the best performance among the strategies by achieving the score average **0.2826**. Finally, the best performance of the rules is achieved by **Stepwise-bottom-up** rule and the worse performance by the **Stepwise-top-down**. As as figure 6.3 shows, this behavior of the rules is also verifiable when the **Pessimist** strategy is chosen.

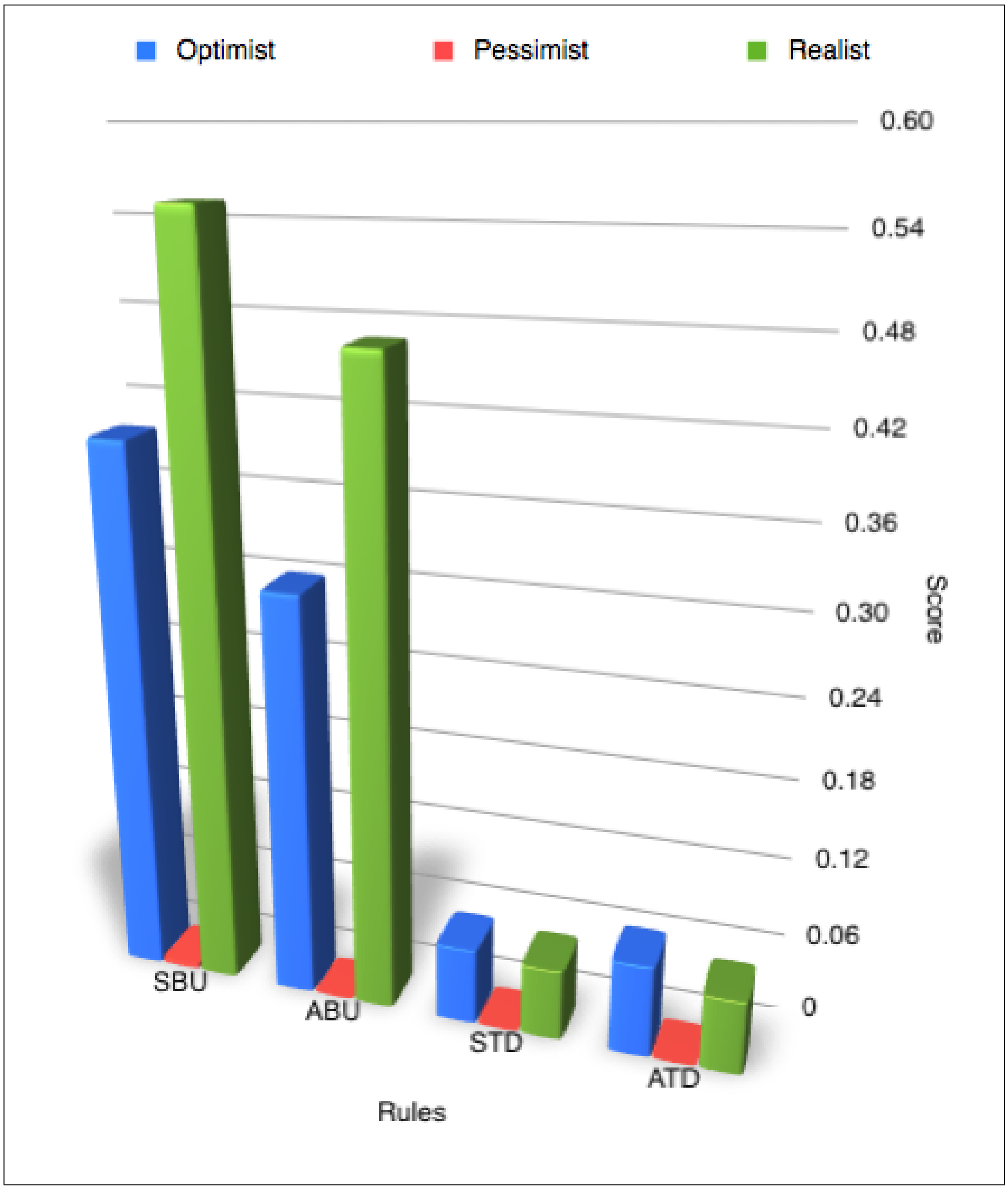


Figure 6.1: Chart of the Scores of All Rules and Strategies

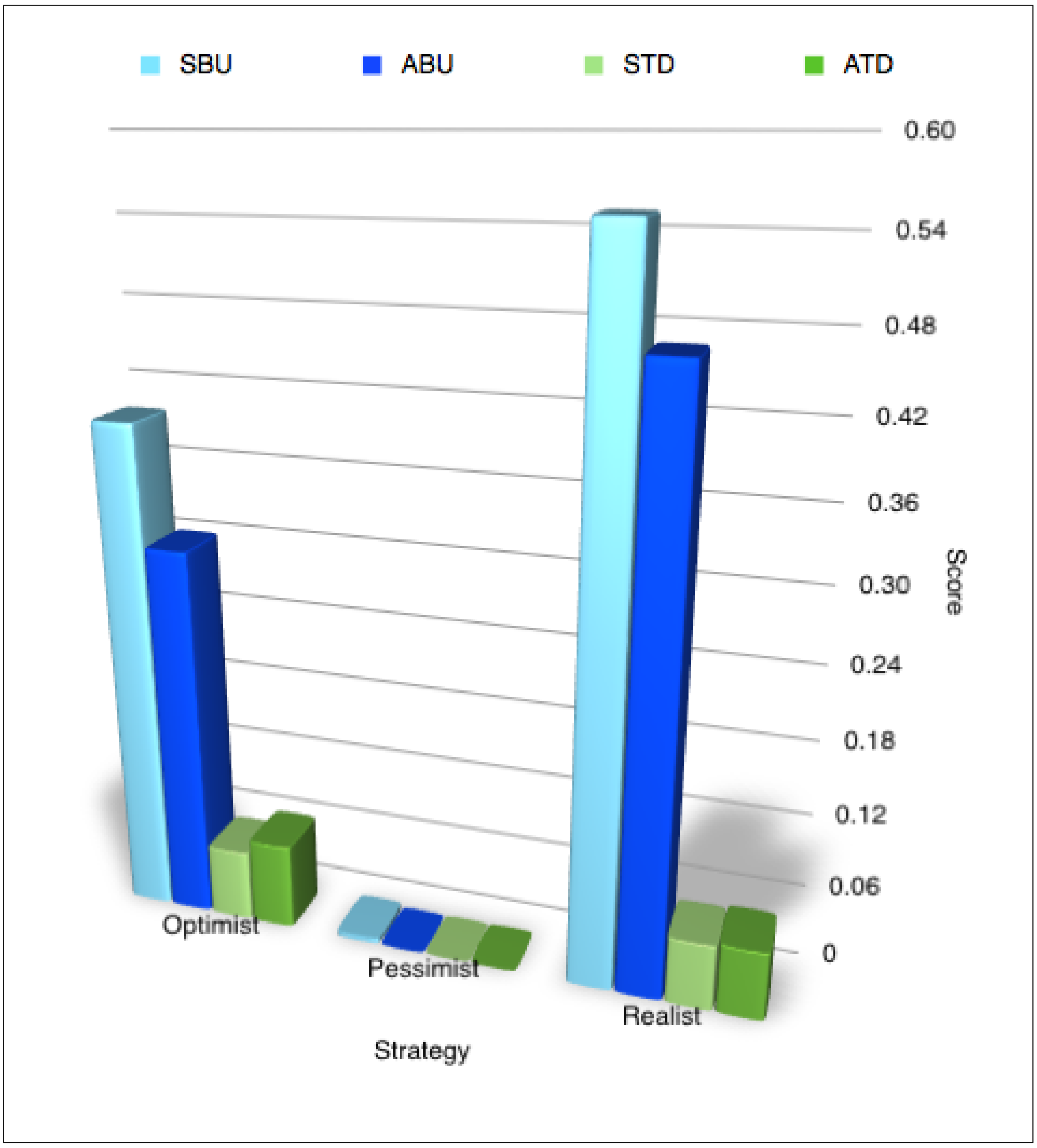


Figure 6.2: Chart of the Scores of All Rules and Strategies

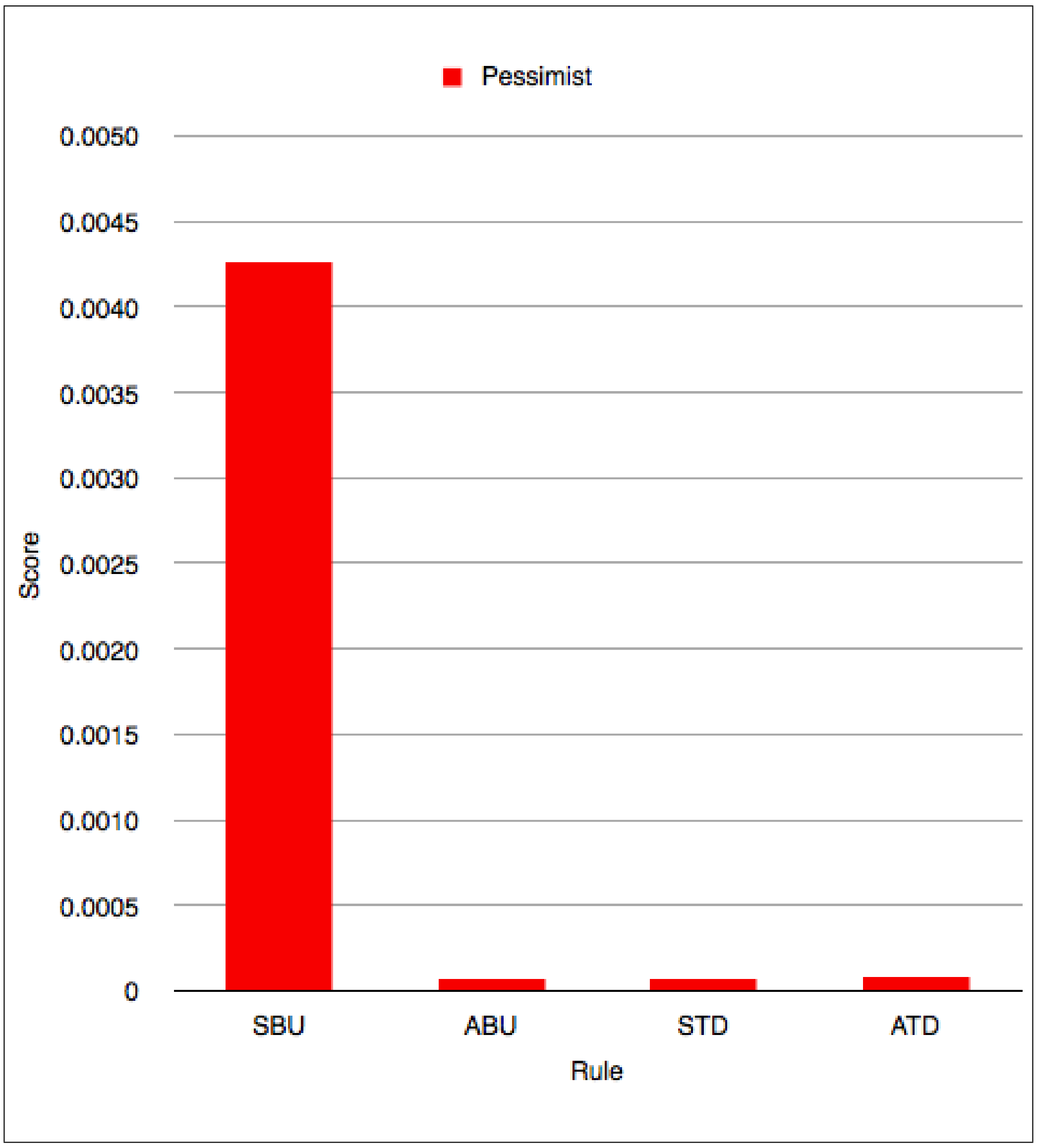


Figure 6.3: Chart of the Scores of Pessimist Strategy

Chapter 7

Conclusion and Future Works

This Chapter focuses on the results of this diploma thesis. During the work, a framework was implemented to interpret the video clips with the help of RacerPro and the tracking data of the frames of the clips. The performance of the interpretation using the frame is discussed in the first Section. In the second Section, the cases of study, which could improve the performance of the work introduced and studied in this work are mentioned.

7.1 Conclusion

Four groups of rules and three strategies for interpreting the scene were introduced in this work; therefore, the interpretation was done in twelve different modes. To evaluate these modes, a scoring function[1] was introduced in Chapter 6. By examining the rules and the strategies using the scoring function, the following results were achieved:

- The **Realist** strategy reached the best score compared to the **Optimist** and the **Pessimist** strategies. The weakness of the **Optimist** strategy is the high interpretation time. The average interpretation time of this strategy is **278.318 seconds**. The advantage of this strategy is the high amount of the data that is interpreted and explained. The weakness of the **Pessimist** strategy is the low amount of data that the framework delivers if this strategy is chosen; nevertheless, this strategy has the best interpretation time average, which is about **75.155 seconds**. The **Realist** strategy balances the advantages and disadvantages of the other two strategies by reaching the time interpretation average of **156.318 seconds**, which is 56.16% shorter than the interpretation time average of the **Optimist** strategy but 207.99% longer the interpretation time average of the **Realist** strategy. However, the **Realist** strategy delivers a high amount of data as explanations of the scene compared to the **Optimist** strategy.

  The scoring function shows better the balance between the **Optimist** and the **Pessimist** strategies, which has been achieved by the **Realist** strategy.

[1] Equation 6.5

The average score of each strategy is shown in table 7.1.

Strategy	Average Score
Optimist	**0.2077**
Pessimist	**0.00113**
Realist	**0.2826**

Table 7.1: Average Score of the Strategies

As it can be seen, the score of the **Realist** strategy is **0.2826**, which is 1.36 times better than the **Optimist** strategy and **250.8** times better than the **Pessimist** strategy.

- Each group of rules explains and delivers almost the same amount of data; therefore, the interpretation time has the major role if the performance of the rules is to be considered. Table 7.2 shows the scores of each group of rules.

Rule	Average Score
Stepwise-bottom-up	**0.3184**
Absolute-bottom-up	**0.2577**
Stepwise-top-down	**0.0372**
Absolute-top-down	**0.042**

Table 7.2: Average Score of the Rules

Between the groups of rules introduced in this work, the **Stepwise-bottom-up** showed the best results, mostly because of the low interpretation time that is needed when this rule is chosen; however, the bottom-up rules are only able to describe a relation between two primitive objects or the existence of a primitive object. On the one hand, this is an advantage for the bottom-up rules, since the interpretation of a single concept or role assertion is a short process, on the other hand, the quality of the explanation decreases, since the other primitive objects on the scene are not considered, and the explanation might not be an accurate regarding the general situation of the scene.

The top-down rules need a longer time for the interpretation of the scene, since they consider the aggregates in the higher levels first and try to explain the scene by instantiating the instances of these aggregates or the relations between them; Because the best explanation should be chosen by the reasoner, the reasoner needs a longer time to study all the results and since different explanations could be found for a specific aggregate and in general, for the scene, the number of possible explanations is a high number; even though, only a few or even a single one of them is the best explanation. The **Absolute** version of the top-down rules shows a better behavior than the **Stepwise** version because the levels in the hierarchy of the aggregates and the concepts in **Stepwise** version are visited and processed more than one time by the reasoner and therefore, the framework and the reasoner need a longer time to interpret a scene using this group of rules.

7.2 Future Works

As mentioned so far, the interpretation time plays a major role for the explanation of the scene, regardless of the rules and the strategies that are chosen. The performance of the interpretation will be improved if the interpretation time becomes shorter. To do so, many approaches could be studied.

One approach could be the idea mentioned in 3.1.3, which was deactivating all the rules with the same head, activating them one by one (based on their priority and completeness) and explaining the scene with the help of a single rule. If a rule with a higher priority could explain the scene, the rest of the rules should be ignored; otherwise, another rule should be activated and applied.

Another possibility is to switch between the rules and strategies regarding the position of the frame, which is to be interpreted, in the clip. In other words, each rule or strategy is active in specific point of time of the interpretation [8] [10] [7] [21]. For example, the interpretation could begin with a **Pessimist** strategy because the interpreting framework is not aware of the situation in the scene at the beginning of its work. The strategy could be changed to **Realist** when enough evidences are present in the scene to explain some of the observations. By reaching the end of the clip, the strategy could be changed to the **Optimist** strategy because the probability that the primitive objects change their positions or be replaced becomes lower as the clip reaches its end.

This could also be done for the rules. The interpretation can begin with the bottom-up rules in order to interpret the initial observations in the scene. The framework can specifies then which aggregates have the highest probability to be instantiated on the scene; as a result, the framework is able to fire the corresponding top-down rules of these aggregates. ■

List of Figures

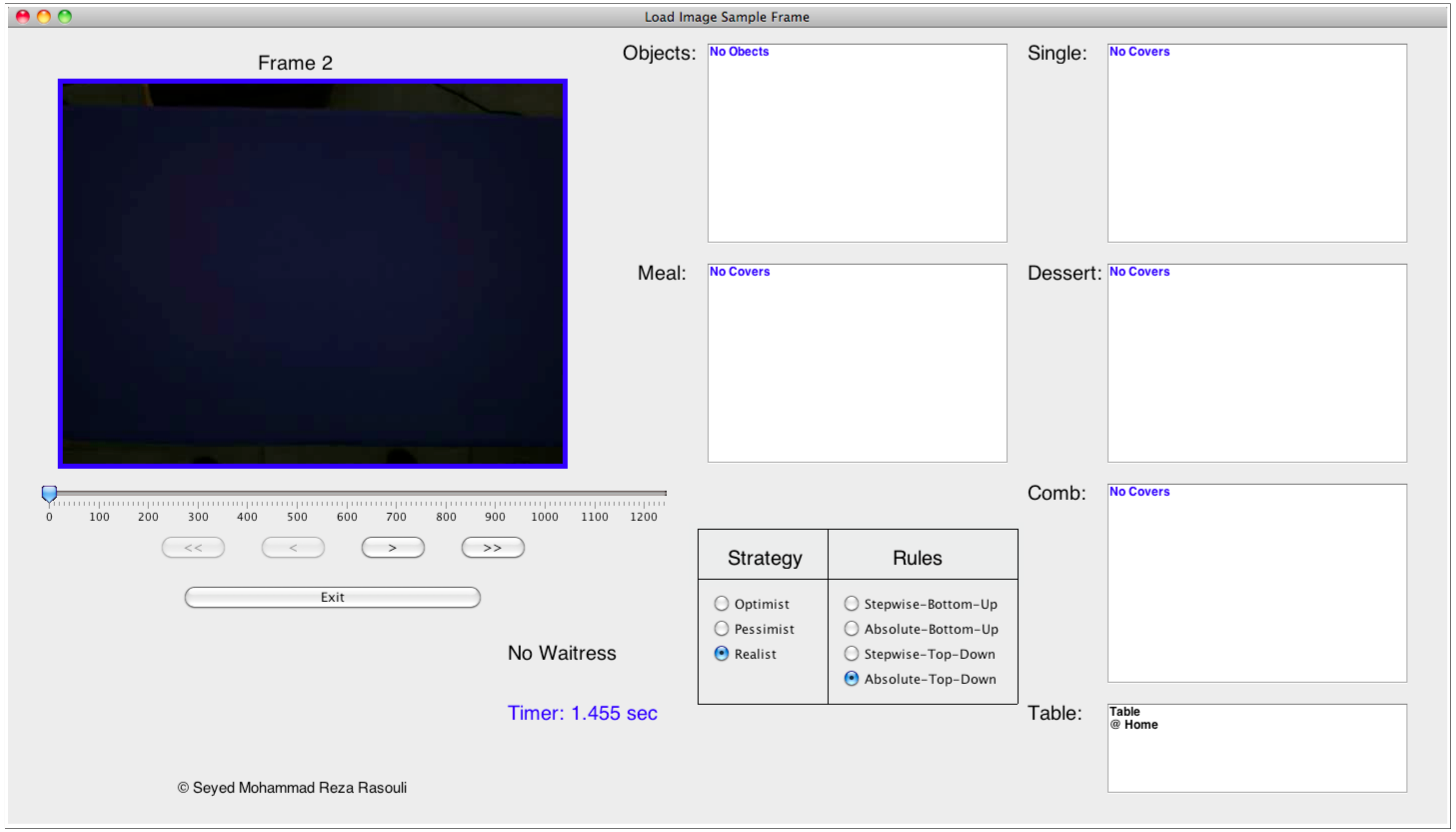
List of Tables

Appendix A

Result of Run for Rule: Absolute-top-down, Strategy: Realist

Frame Number	**Number of Objects**	**Interpretation Time (seconds)**
2	0	1.455
86	2	2.762
262	4	12.999
458	6	10.064
692	8	31.496
818	9	43.988
888	10	56.373
970	11	75.689
1022	12	151.009
1126	13	128.062
1220	14	213.984
1246	14	1218.134
Average	**8.58**	**162.168**

Table A.1: Result of Absolute-top-down Rule - Realist Strategy


Load Image Sample Frame

Frame 2

Objects: No Obects

Single: No Covers

Meal: No Covers

Dessert: No Covers

Comb: No Covers

Table: Table @ Home

Strategy: Optimist [✓ Realist] Pessimist

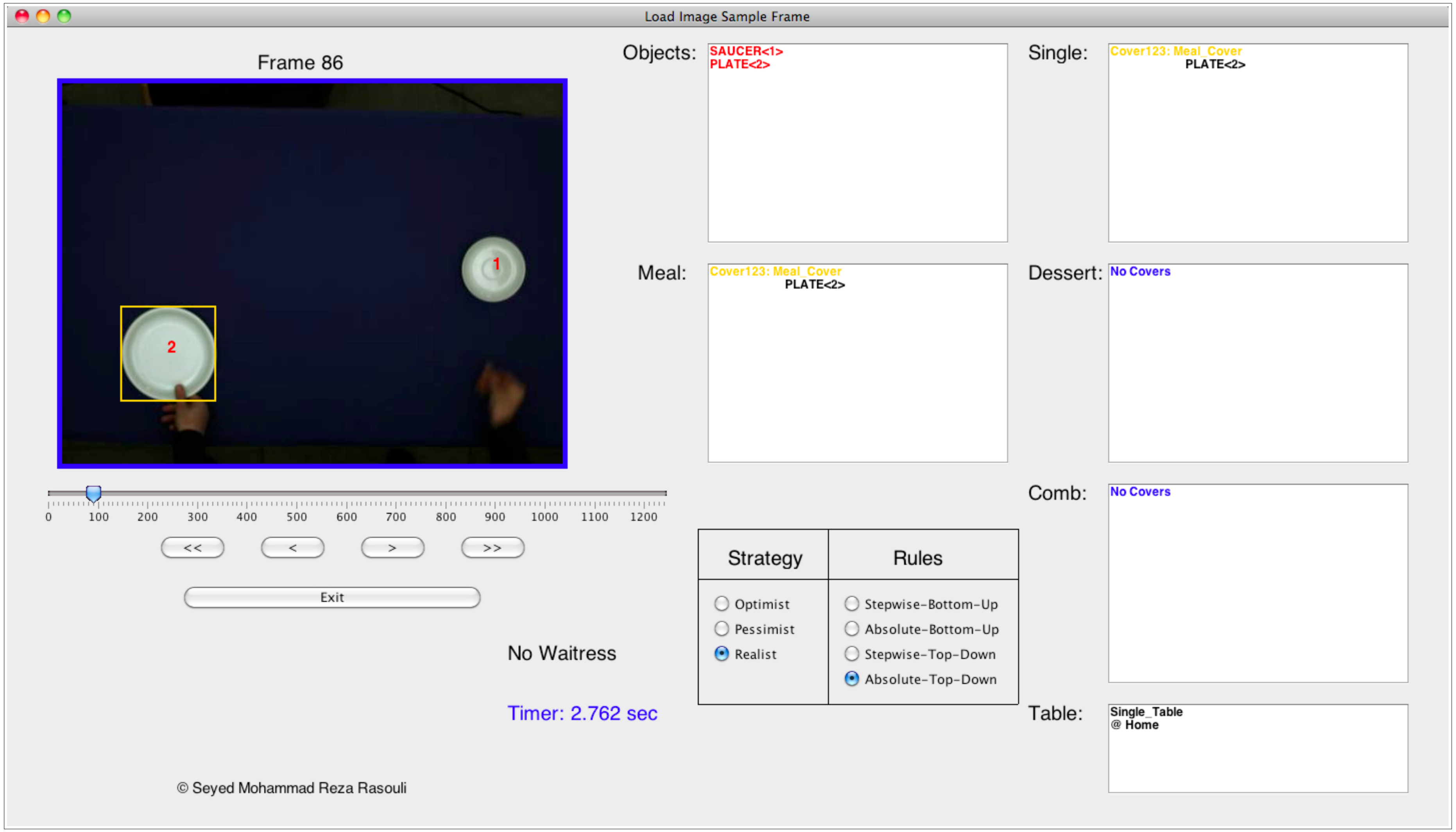
Rules: Stepwise-Bottom-Up, Absolute-Bottom-Up, Stepwise-Top-Down [✓ Absolute-Top-Down]

No Waitress

Timer: 1.455 sec

© Seyed Mohammad Reza Rasouli

Figure A.1: Rule: Absolute-top-down, Strategy: Realist - Frame 2



Load Image Sample Frame

Frame 86

Objects: SAUCER<1> PLATE<2>

Meal: Cover123: Meal_Cover PLATE<2>

Single: Cover123: Meal_Cover PLATE<2>

Dessert: No Covers

Comb: No Covers

Table: Single_Table @ Home

Strategy: Optimist [✓ Realist] Pessimist

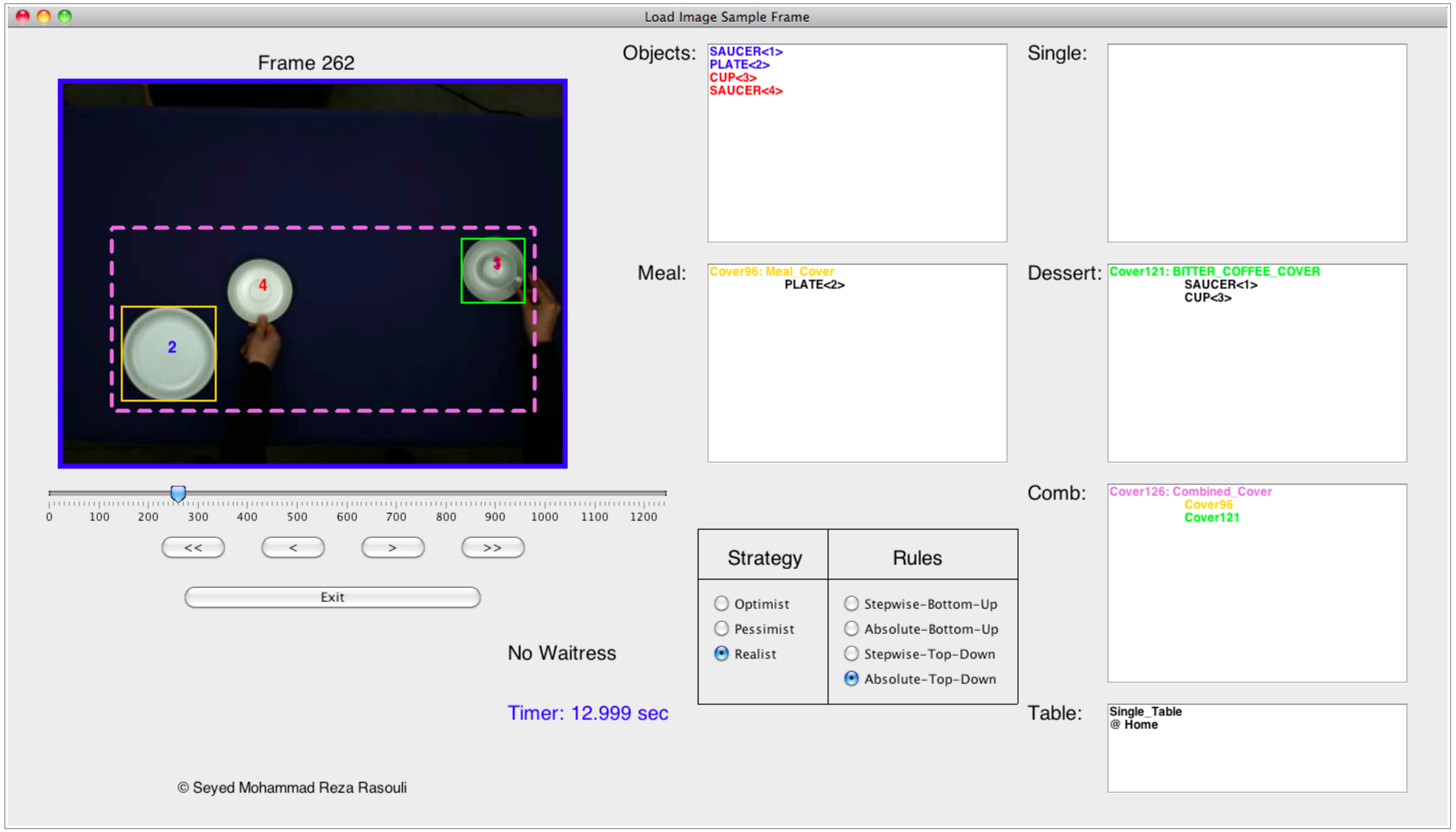
Rules: Stepwise-Bottom-Up, Absolute-Bottom-Up, Stepwise-Top-Down [✓ Absolute-Top-Down]

No Waitress

Timer: 2.762 sec

© Seyed Mohammad Reza Rasouli

Figure A.2: Rule: Absolute-top-down, Strategy: Realist - Frame 86



Objects:
SAUCER<1>
PLATE<2>
CUP<3>
SAUCER<4>

Single:

Meal:
Cover96: Meal_Cover
PLATE<2>

Dessert:
Cover121: BITTER_COFFEE_COVER
SAUCER<1>
CUP<3>

Comb:
Cover126: Combined_Cover
Cover96
Cover121

Table:
Single_Table
@ Home

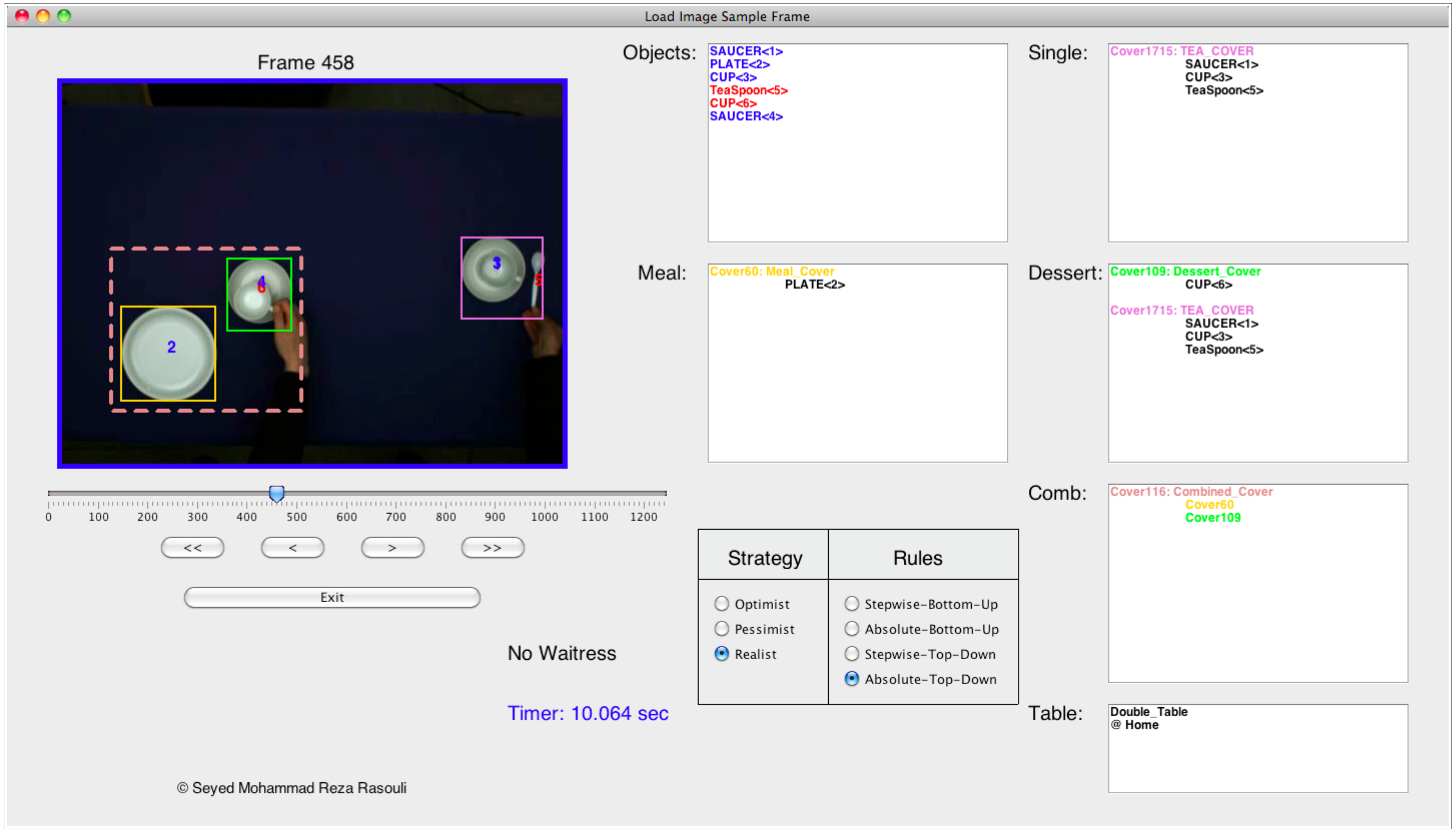
Strategy	Rules
Optimist Pessimist [✓ Realist]	Stepwise-Bottom-Up Absolute-Bottom-Up Stepwise-Top-Down [✓ Absolute-Top-Down]

No Waitress

Timer: 12.999 sec

© Seyed Mohammad Reza Rasouli

Figure A.3: Rule: Absolute-top-down, Strategy: Realist - Frame 262



Load Image Sample Frame

Frame 458

Objects:
SAUCER<1>
PLATE<2>
CUP<3>
TeaSpoon<5>
CUP<6>
SAUCER<4>

Single:
Cover1715: TEA_COVER
SAUCER<1>
CUP<3>
TeaSpoon<5>

Meal:
Cover60: Meal_Cover
PLATE<2>

Dessert:
Cover109: Dessert_Cover
CUP<6>

Cover1715: TEA_COVER
SAUCER<1>
CUP<3>
TeaSpoon<5>

Comb:
Cover116: Combined_Cover
Cover60
Cover109

Table:
Double_Table
@ Home

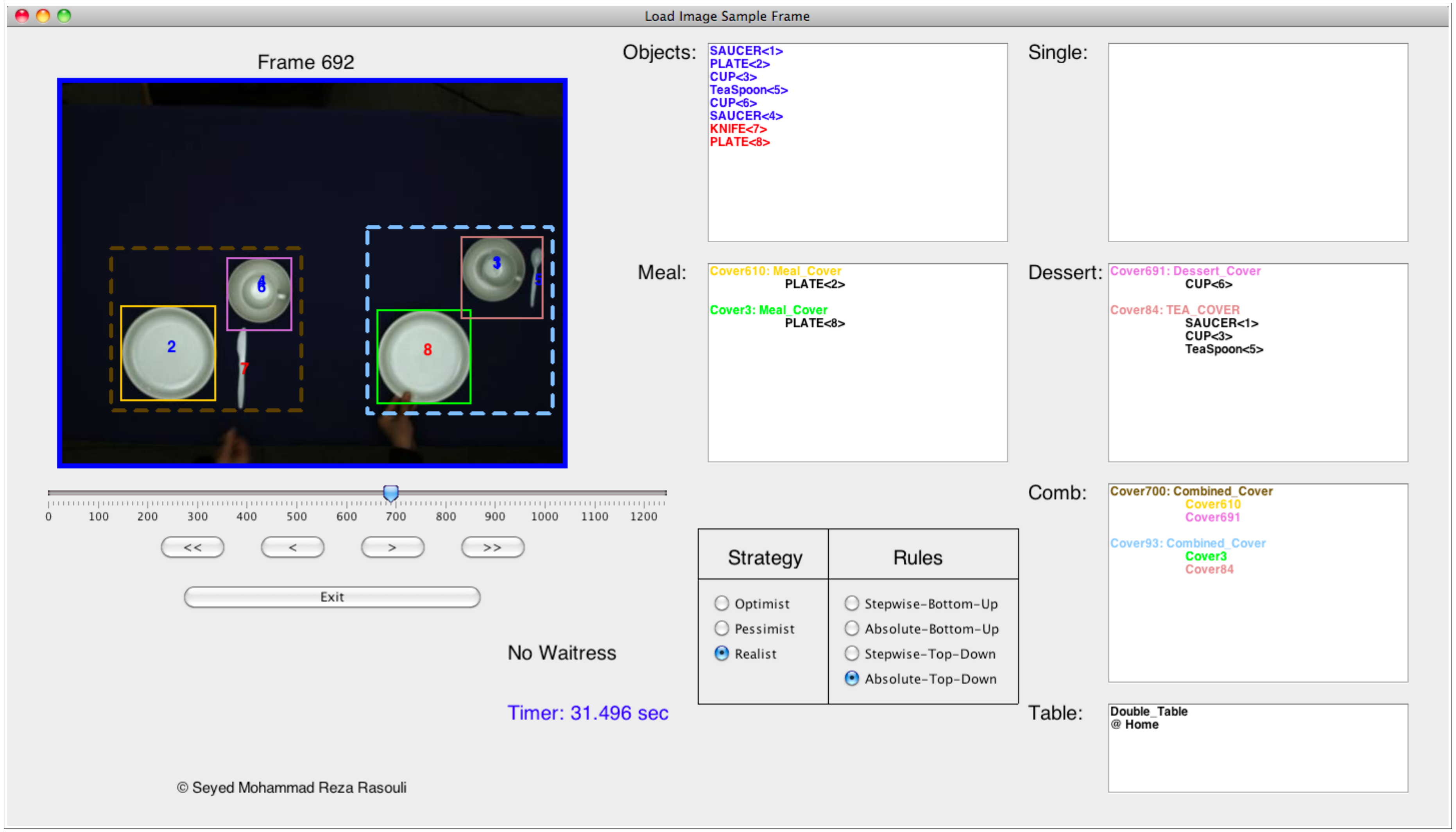
Strategy	Rules
Optimist	Stepwise-Bottom-Up
Pessimist	Absolute-Bottom-Up
[✓ Realist]	Stepwise-Top-Down
	[✓ Absolute-Top-Down]

No Waitress

Timer: 10.064 sec

© Seyed Mohammad Reza Rasouli

Figure A.4: Rule: Absolute-top-down, Strategy: Realist - Frame 458



Objects:
SAUCER<1>
PLATE<2>
CUP<3>
TeaSpoon<5>
CUP<6>
SAUCER<4>
KNIFE<7>
PLATE<8>

Single:

Meal:
Cover610: Meal_Cover
PLATE<2>

Cover3: Meal_Cover
PLATE<8>

Dessert:
Cover691: Dessert_Cover
CUP<6>

Cover84: TEA_COVER
SAUCER<1>
CUP<3>
TeaSpoon<5>

Comb:
Cover700: Combined_Cover
Cover610
Cover691

Cover93: Combined_Cover
Cover3
Cover84

Table:
Double_Table
@ Home

Strategy: Optimist [✓ Realist] Pessimist

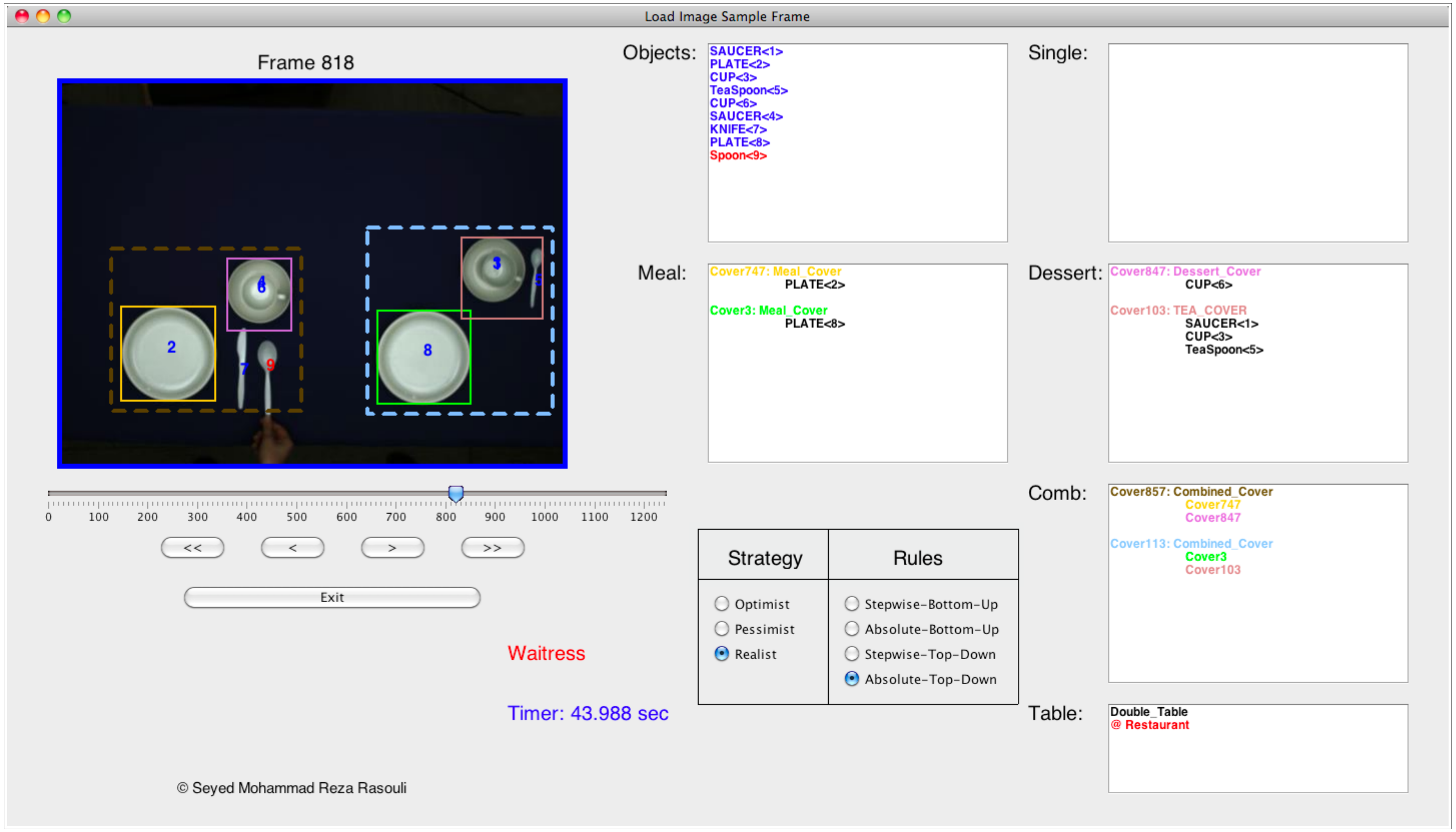
Rules: Stepwise-Bottom-Up, Absolute-Bottom-Up, Stepwise-Top-Down, [✓ Absolute-Top-Down]

No Waitress

Timer: 31.496 sec

© Seyed Mohammad Reza Rasouli

Figure A.5: Rule: Absolute-top-down, Strategy: Realist - Frame 692



Objects:
SAUCER<1>
PLATE<2>
CUP<3>
TeaSpoon<5>
CUP<6>
SAUCER<4>
KNIFE<7>
PLATE<8>
Spoon<9>

Single:

Meal:
Cover747: Meal_Cover
PLATE<2>

Cover3: Meal_Cover
PLATE<8>

Dessert:
Cover847: Dessert_Cover
CUP<6>

Cover103: TEA_COVER
SAUCER<1>
CUP<3>
TeaSpoon<5>

Comb:
Cover857: Combined_Cover
Cover747
Cover847

Cover113: Combined_Cover
Cover3
Cover103

Strategy	Rules
○ Optimist	○ Stepwise-Bottom-Up
○ Pessimist	○ Absolute-Bottom-Up
[✓ Realist]	○ Stepwise-Top-Down
	[✓ Absolute-Top-Down]

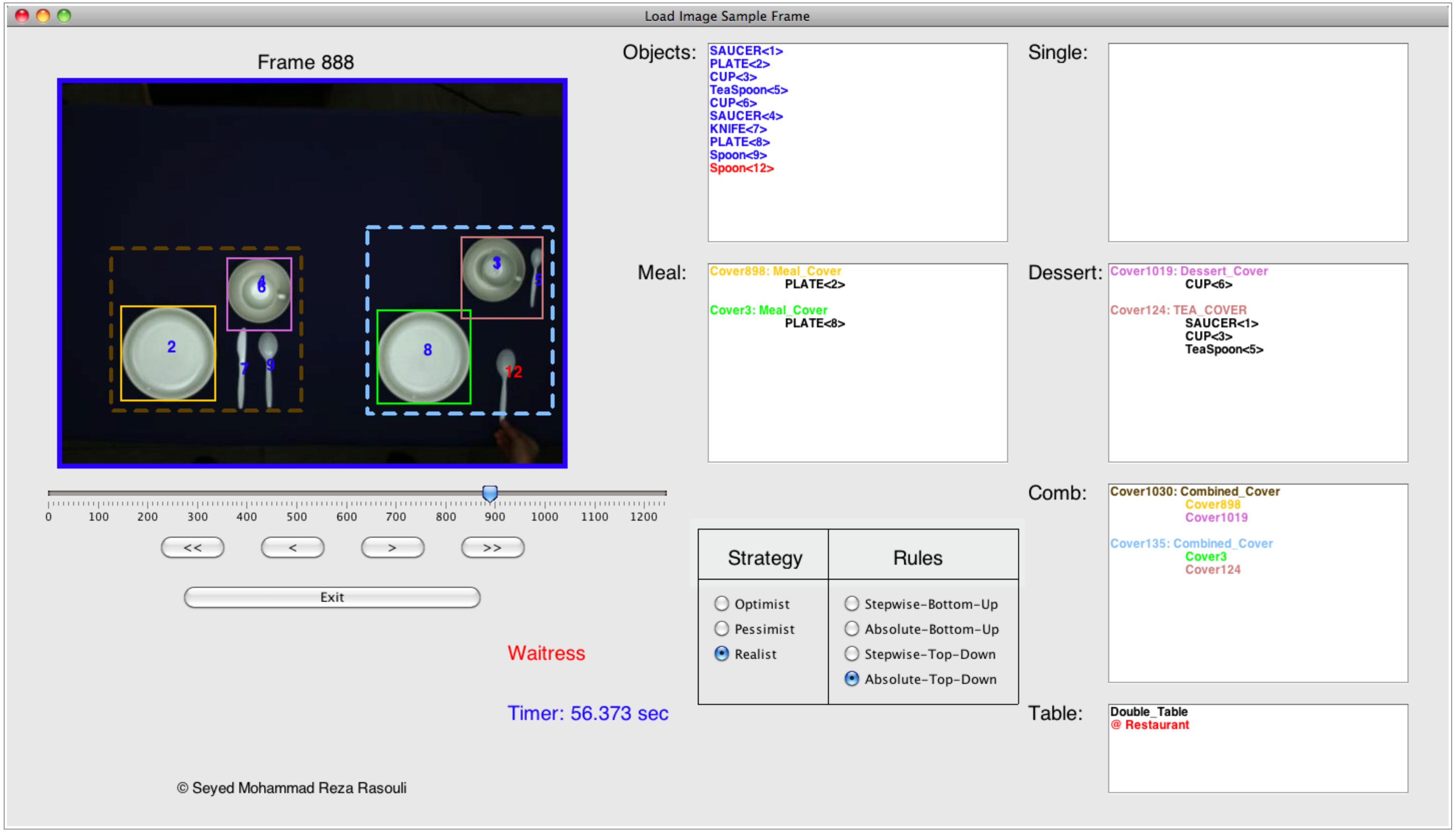
Table:
Double_Table
@ Restaurant

Waitress

Timer: 43.988 sec

© Seyed Mohammad Reza Rasouli

Figure A.6: Rule: Absolute-top-down, Strategy: Realist - Frame 818



Objects:
SAUCER<1>
PLATE<2>
CUP<3>
TeaSpoon<5>
CUP<6>
SAUCER<4>
KNIFE<7>
PLATE<8>
Spoon<9>
Spoon<12>

Single:

Meal:
Cover898: Meal_Cover
PLATE<2>

Cover3: Meal_Cover
PLATE<8>

Dessert:
Cover1019: Dessert_Cover
CUP<6>

Cover124: TEA_COVER
SAUCER<1>
CUP<3>
TeaSpoon<5>

Comb:
Cover1030: Combined_Cover
Cover898
Cover1019

Cover135: Combined_Cover
Cover3
Cover124

Table:
Double_Table
@ Restaurant

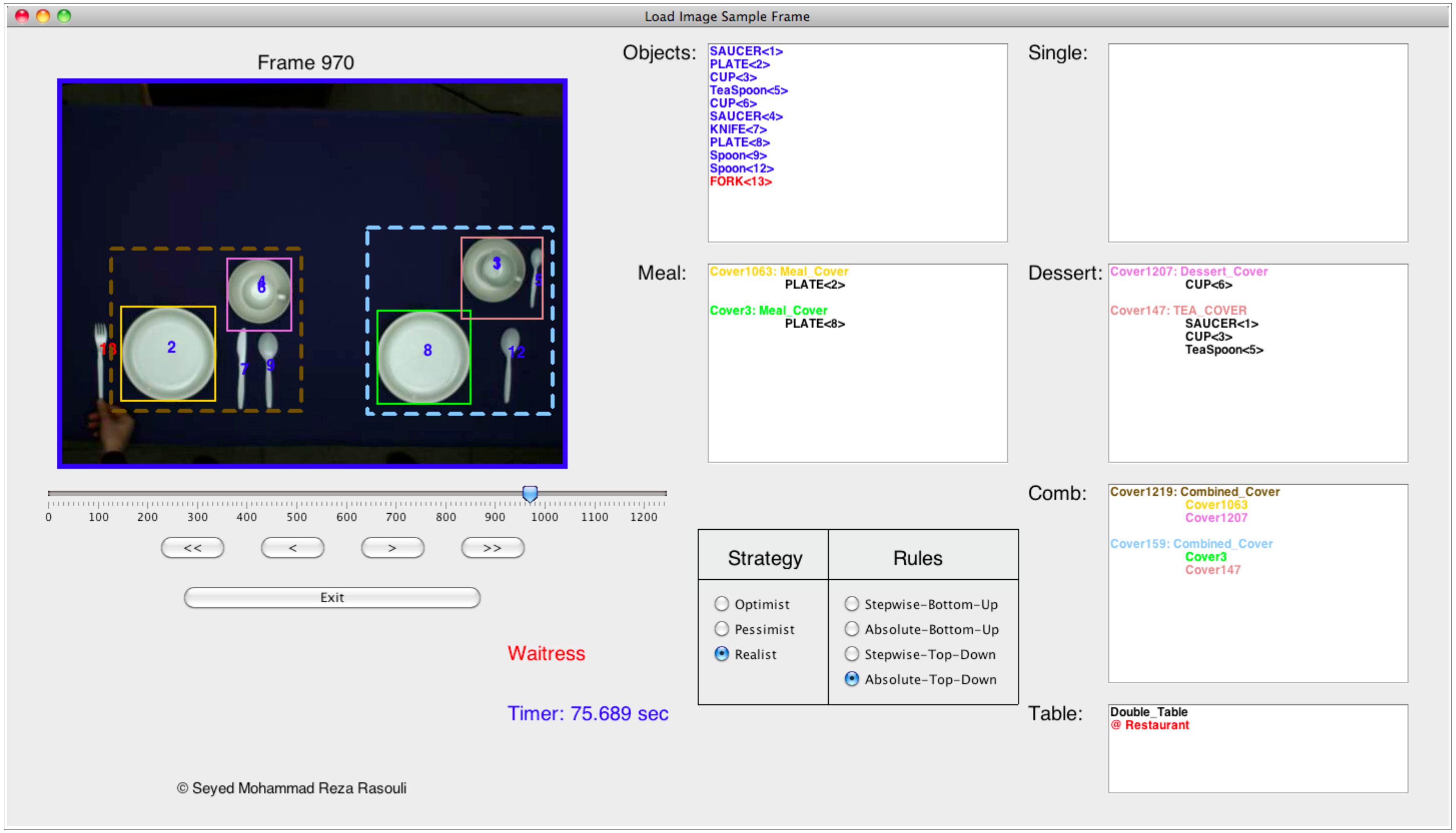
Strategy	Rules
Optimist	Stepwise-Bottom-Up
Pessimist	Absolute-Bottom-Up
[✓ Realist]	Stepwise-Top-Down
	[✓ Absolute-Top-Down]

Waitress

Timer: 56.373 sec

© Seyed Mohammad Reza Rasouli

Figure A.7: Rule: Absolute-top-down, Strategy: Realist - Frame 888



Waitress

Timer: 75.689 sec

© Seyed Mohammad Reza Rasouli

Objects:
SAUCER<1>
PLATE<2>
CUP<3>
TeaSpoon<5>
CUP<6>
SAUCER<4>
KNIFE<7>
PLATE<8>
Spoon<9>
Spoon<12>
FORK<13>

Single:

Meal:
Cover1063: Meal_Cover
PLATE<2>

Cover3: Meal_Cover
PLATE<8>

Dessert:
Cover1207: Dessert_Cover
CUP<6>

Cover147: TEA_COVER
SAUCER<1>
CUP<3>
TeaSpoon<5>

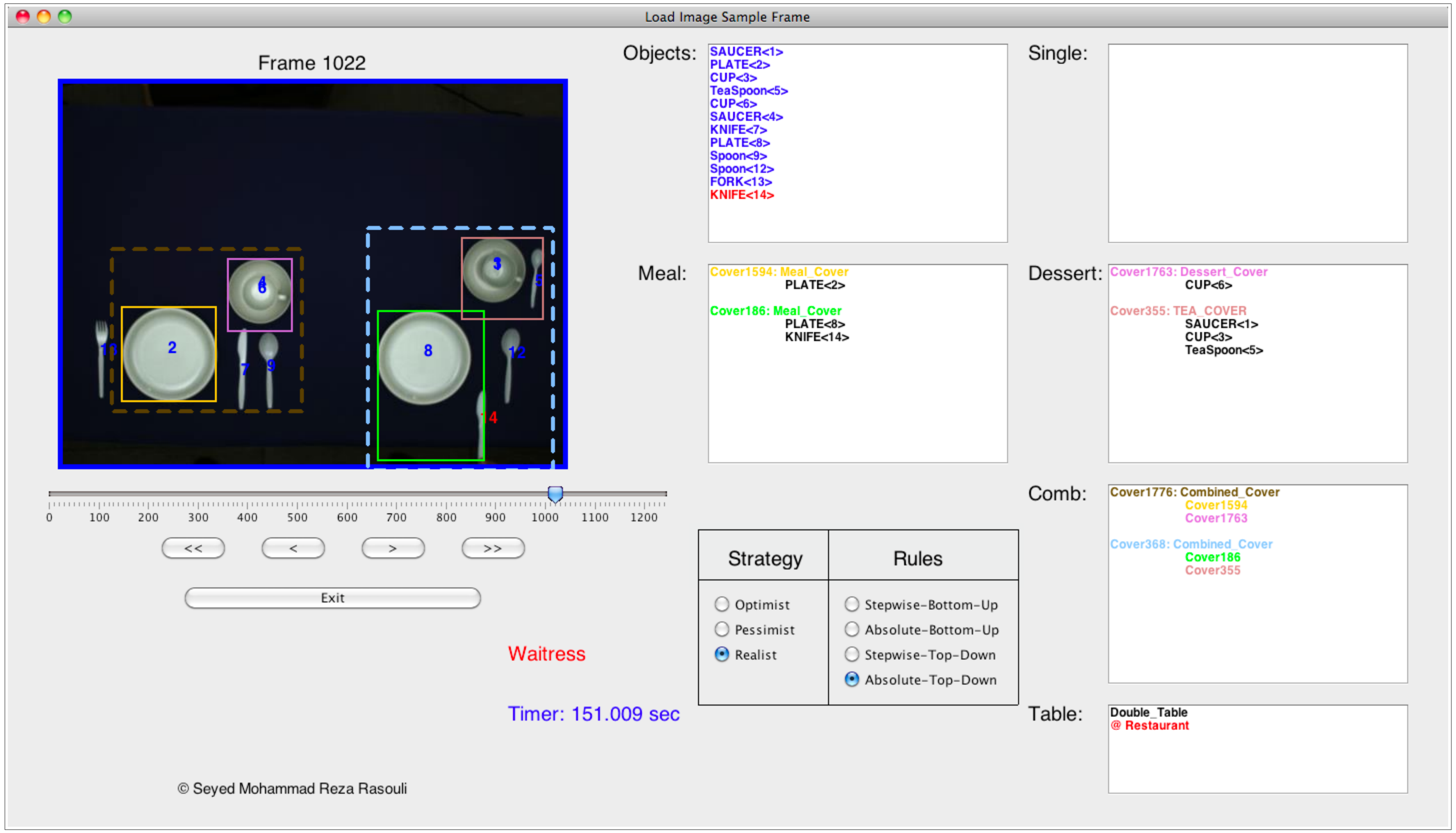
Comb:
Cover1219: Combined_Cover
Cover1063
Cover1207

Cover159: Combined_Cover
Cover3
Cover147

Table:
Double_Table
@ Restaurant

Strategy	Rules
Optimist	Stepwise-Bottom-Up
Pessimist	Absolute-Bottom-Up
[✓ Realist]	Stepwise-Top-Down
	[✓ Absolute-Top-Down]

Figure A.8: Rule: Absolute-top-down, Strategy: Realist - Frame 970



Timer: 151.009 sec

© Seyed Mohammad Reza Rasouli

Objects:
SAUCER<1>
PLATE<2>
CUP<3>
TeaSpoon<5>
CUP<6>
SAUCER<4>
KNIFE<7>
PLATE<8>
Spoon<9>
Spoon<12>
FORK<13>
KNIFE<14>

Single:

Meal:
Cover1594: Meal_Cover
PLATE<2>

Cover186: Meal_Cover
PLATE<8>
KNIFE<14>

Dessert:
Cover1763: Dessert_Cover
CUP<6>

Cover355: TEA_COVER
SAUCER<1>
CUP<3>
TeaSpoon<5>

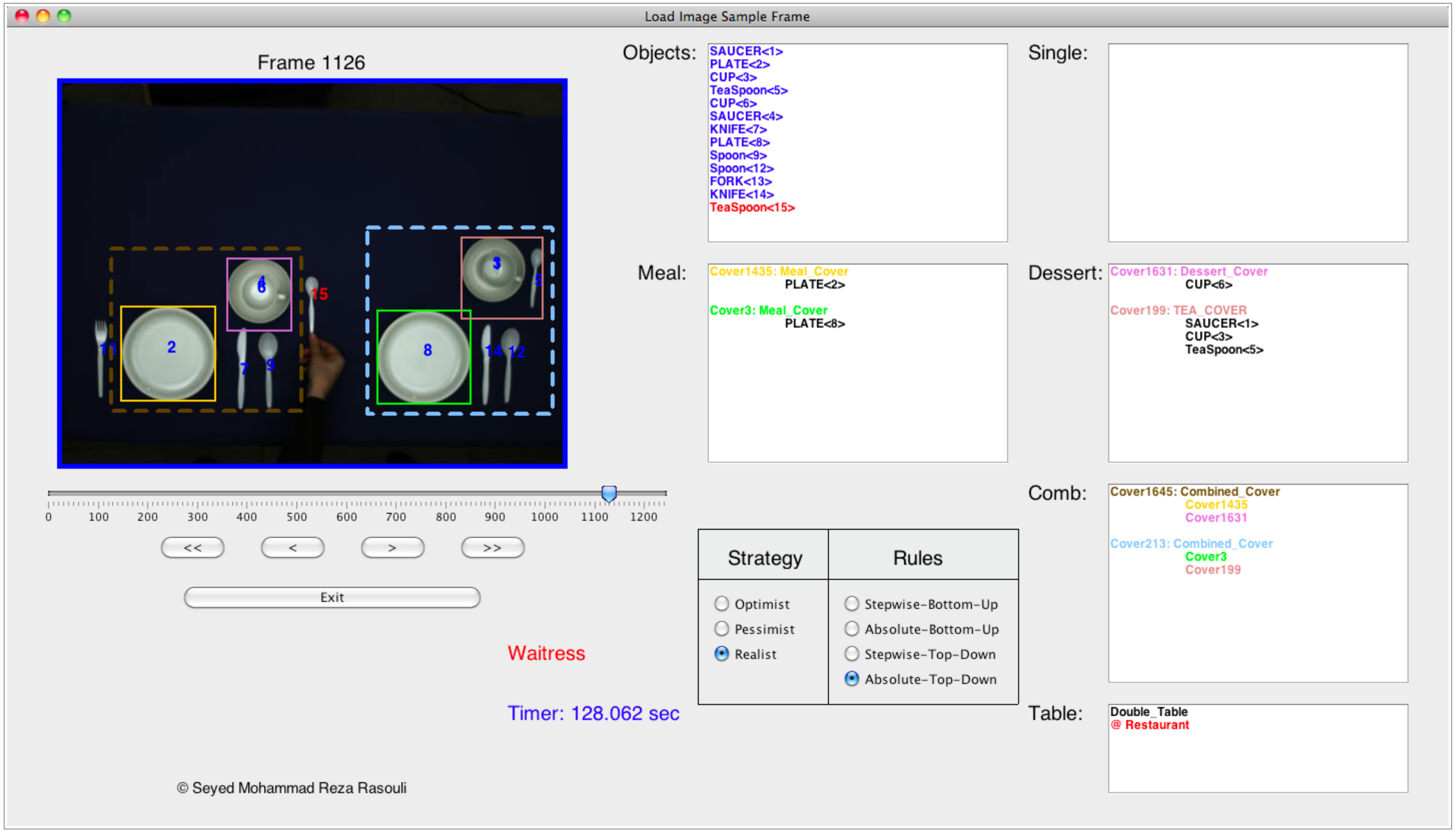
Comb:
Cover1776: Combined_Cover
Cover1594
Cover1763

Cover368: Combined_Cover
Cover186
Cover355

Table:
Double_Table
@ Restaurant

Strategy	Rules
Optimist	Stepwise-Bottom-Up
Pessimist	Absolute-Bottom-Up
[✓ Realist]	Stepwise-Top-Down
	[✓ Absolute-Top-Down]

Figure A.9: Rule: Absolute-top-down, Strategy: Realist - Frame 1022



Waitress

Timer: 128.062 sec

© Seyed Mohammad Reza Rasouli

Objects:
SAUCER<1>
PLATE<2>
CUP<3>
TeaSpoon<5>
CUP<6>
SAUCER<4>
KNIFE<7>
PLATE<8>
Spoon<9>
Spoon<12>
FORK<13>
KNIFE<14>
TeaSpoon<15>

Single:

Meal:
Cover1435: Meal_Cover
PLATE<2>

Cover3: Meal_Cover
PLATE<8>

Dessert:
Cover1631: Dessert_Cover
CUP<6>

Cover199: TEA_COVER
SAUCER<1>
CUP<3>
TeaSpoon<5>

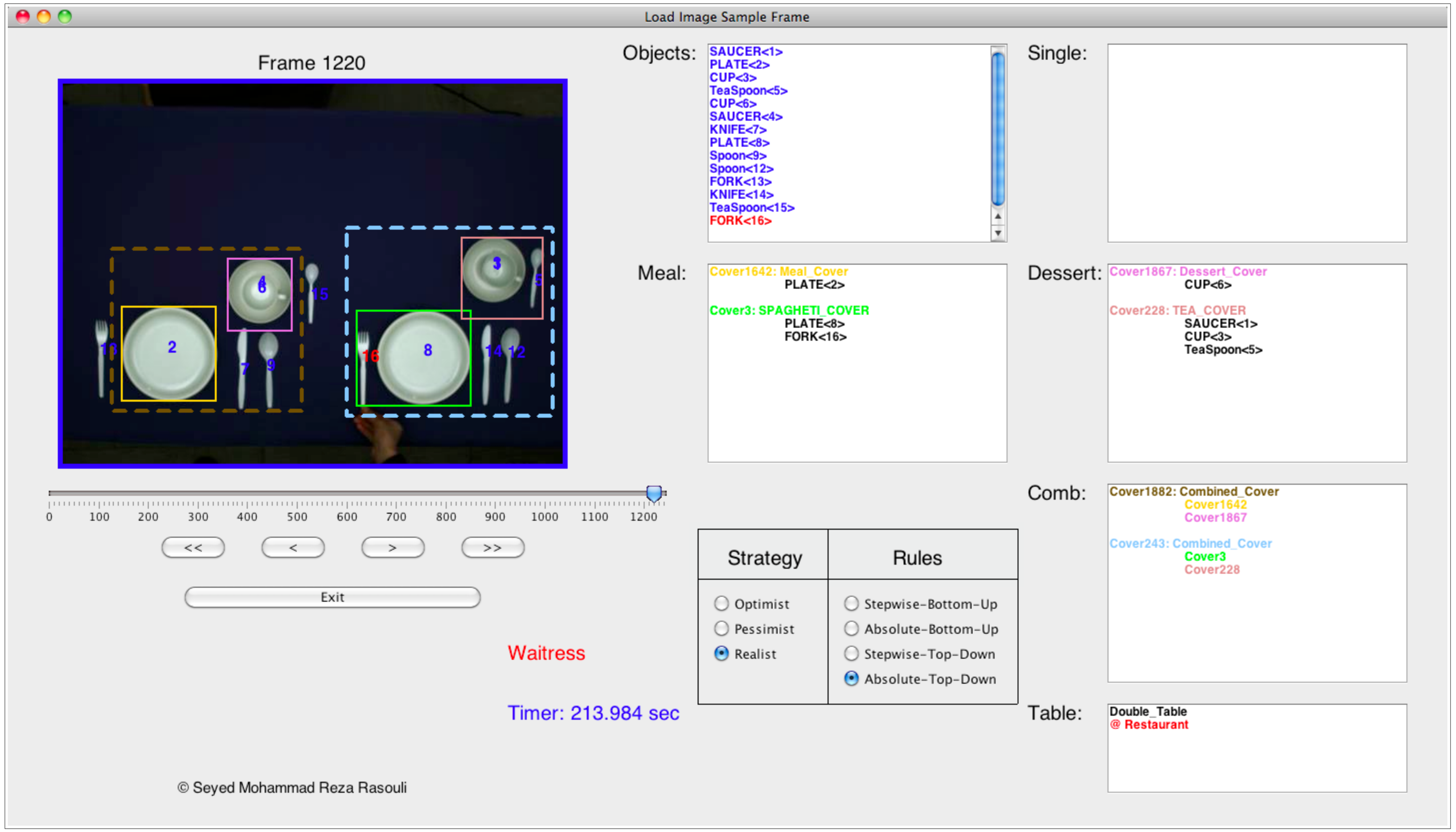
Comb:
Cover1645: Combined_Cover
Cover1435
Cover1631

Cover213: Combined_Cover
Cover3
Cover199

Table:
Double_Table
@ Restaurant

Strategy	Rules
Optimist	Stepwise-Bottom-Up
Pessimist	Absolute-Bottom-Up
[✓ Realist]	Stepwise-Top-Down
	[✓ Absolute-Top-Down]

Figure A.10: Rule: Absolute-top-down, Strategy: Realist - Frame 1126



Objects:
SAUCER<1>
PLATE<2>
CUP<3>
TeaSpoon<5>
CUP<6>
SAUCER<4>
KNIFE<7>
PLATE<8>
Spoon<9>
Spoon<12>
FORK<13>
KNIFE<14>
TeaSpoon<15>
FORK<16>

Single:

Meal:
Cover1642: Meal_Cover
PLATE<2>

Cover3: SPAGHETI_COVER
PLATE<8>
FORK<16>

Dessert:
Cover1867: Dessert_Cover
CUP<6>

Cover228: TEA_COVER
SAUCER<1>
CUP<3>
TeaSpoon<5>

Comb:
Cover1882: Combined_Cover
Cover1642
Cover1867

Cover243: Combined_Cover
Cover3
Cover228

Table:
Double_Table
@ Restaurant

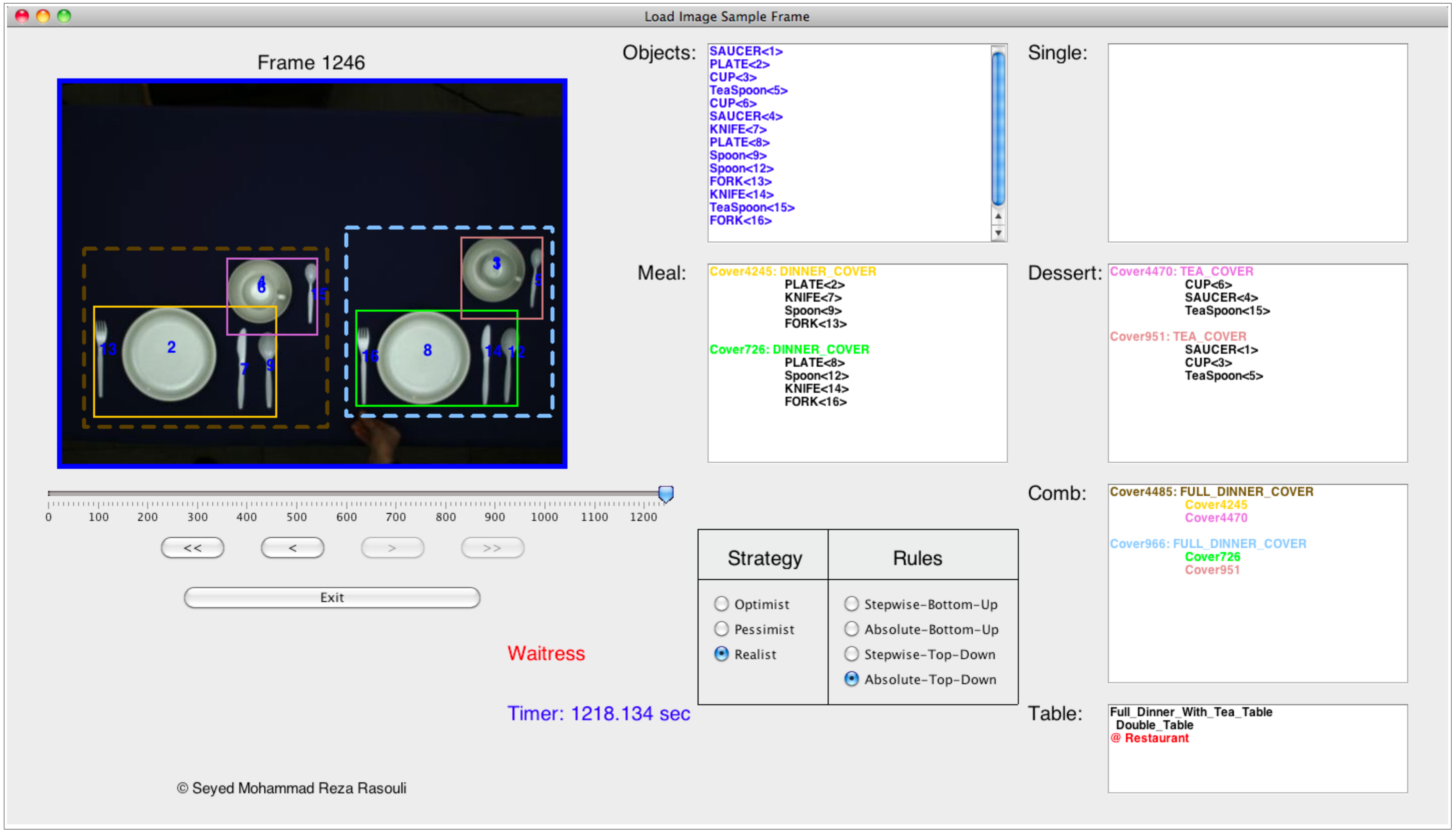
Strategy	Rules
Optimist	Stepwise-Bottom-Up
Pessimist	Absolute-Bottom-Up
[✓ Realist]	Stepwise-Top-Down
	[✓ Absolute-Top-Down]

Waitress

Timer: 213.984 sec

© Seyed Mohammad Reza Rasouli

Figure A.11: Rule: Absolute-top-down, Strategy: Realist - Frame 1220


Load Image Sample Frame

Frame 1246

Objects:
SAUCER<1>
PLATE<2>
CUP<3>
TeaSpoon<5>
CUP<6>
SAUCER<4>
KNIFE<7>
PLATE<8>
Spoon<9>
Spoon<12>
FORK<13>
KNIFE<14>
TeaSpoon<15>
FORK<16>

Single:

Meal:
Cover4245: DINNER_COVER
PLATE<2>
KNIFE<7>
Spoon<9>
FORK<13>

Cover726: DINNER_COVER
PLATE<8>
Spoon<12>
KNIFE<14>
FORK<16>

Dessert:
Cover4470: TEA_COVER
CUP<6>
SAUCER<4>
TeaSpoon<15>

Cover951: TEA_COVER
SAUCER<1>
CUP<3>
TeaSpoon<5>

Comb:
Cover4485: FULL_DINNER_COVER
Cover4245
Cover4470

Cover966: FULL_DINNER_COVER
Cover726
Cover951

Table:
Full_Dinner_With_Tea_Table
Double_Table
@ Restaurant

Strategy	Rules
Optimist	Stepwise-Bottom-Up
Pessimist	Absolute-Bottom-Up
[✓ Realist]	Stepwise-Top-Down
	[✓ Absolute-Top-Down]

Waitress

Timer: 1218.134 sec

© Seyed Mohammad Reza Rasouli

Figure A.12: Rule: Absolute-top-down, Strategy: Realist - Frame 1246

Appendix B

Abbreviations

Abbreviation	**Rule**	**Strategy**
SBUO	Stepwise-bottom-up	Optimist
SBUP	Stepwise-bottom-up	Pessimist
SBUR	Stepwise-bottom-up	Realist
ABUO	Absolute-bottom-up	Optimist
ABUP	Absolute-bottom-up	Pessimist
ABUR	Absolute-bottom-up	Realist
STDO	Stepwise-top-down	Optimist
STDP	Stepwise-top-down	Pessimist
STDR	Stepwise-top-down	Realist
ATDO	Absolute-top-down	Optimist
ATDP	Absolute-top-down	Pessimist
ATDR	Absolute-top-down	Realist

Table B.1: Abbreviation Table

Bibliography

[1] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[2] Wilfried Bohlken and Bernd Neumann. *Generation of Rules from Ontologies for High-Level Scene Interpretation.* 2009.

[3] S. Castano, S. Espinosa, A. Ferrara, V. Karkaletsis, A. Kaya, R. Möller, S. Montanelli, G. Petasis, and M. Wessel. *Multimedia Interpretation for Dynamic Ontology Evolution.* Oxford University Press.

[4] S. Espinosa, A. Kaya, and R. Möller. *Formalizing Multimedia Interpretation based on Abduction over Description Logic Aboxes.* 2009. CEUR Workshop Proceedings (Vol. 477).

[5] O. Gries, R. Möller, A. Nafissi, K. Sokolski, and M. Rosenfeld. Basic reasoning engine: Report on optimization techniques for first-order probabilistic reasoning. Technical report, Hamburg University of Technology, 2009.

[6] O. Gries, R. Möller, A. Nafissi, K. Sokolski, and M. Rosenfeld. Casam domain ontology. Technical report, Hamburg University of Technology, 2009.

[7] O. Gries, R. Möller, A. Nafissi, K. Sokolski, and M. Rosenfeld. Formalisms supporting first-order probabilistic structures. Technical report, Hamburg University of Technology, 2009.

[8] Oliver Gries, Ralf Möller, Anahita Nafissi, Maurice Rosenfeld, Kamil Sokolski, and Michael Wessel. A probabilistic abduction engine for media interpretation based on ontologies. In *Proceedings of the Fourth international conference on Web reasoning and rule systems*, 2010.

[9] Stefan W. Hamerich, Lars König, and Marcus E. Hennecke. Sprachdialogsysteme im kfz. *KI*, 19(3), 2005.

[10] Somboon Hongeng. Unsupervised learning of multi-object event classes. 2004.

[11] Lothar Hotz and Bernd Neumann. Scene interpretation as a configuration task. *KI*, 2005.

[12] Lothar Hotz and Bernd Neumann. Learning and recognizing structures in façade scenes (etrims) - a retrospective. *KI*, 24(1):63–68, 2010.

[13] Lothar Hotz, Bernd Neumann, and Kasim Terzic. *High-Level Expectations for Low-Level Image Processing.* 2008.

[14] Lothar Hotz, Bernd Neumann, Kasim Terzić, and Jan Sochmann. Feedback between low-level and high-level image processing. Technical Report Report FBI-HH-B-278/07, Universität Hamburg, Hamburg, 2007.

[15] Andreas Jungbluth. über die kopplung von bildverarbeitung und bildverstehen. Studienarbeit, TU Hamburg-Harburg, July 2009.

[16] Karsten Martiny. Query generation for high-level interpretation of multimedia documents. Technical report, Technische Universität Hamburg-Harburg Institute for Software Systems, 2010.

[17] R. Möller and B. Neumann. Ontology-based Reasoning Techniques for Multimedia Interpretation and Retrieval. In *Semantic Multimedia and Ontologies : Theory and Applications*, pages 55–98. Springer, 2008.

[18] R. Möller, C. Schröder, and R. Carsten Lutz. Analyzing configuration systems with description logics: A case study. Technical report, University of Hamburg, Computer Science Department, 1996.

[19] Irma Sofia Espinosa Peraldi, Atila Kaya, and Ralf Möller. *Formalizing Multimedia Interpretation based on Abduction over Description Logic Aboxes.* 2009.

[20] David Poole. Logic programming, abduction and probability. In *FGCS*, pages 530–538, 1992.

[21] David Poole. Logic programming, abduction and probability - a top-down anytime algorithm for estimating prior and posterior probabilities. *New Generation Comput.*, 11(3):377–400, 1993.

[22] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall, Upper Saddle River, NJ, 3. edition, 2009.

[23] Kasim Terzic, Lothar Hotz, and Bernd Neumann. Division of work during behaviour recognition - the scenic approach. 2007.

[24] Kasim Terzic, Lothar Hotz, and Jan Sochman. *Interpreting Structures in Man-made Scenes - Combining Low-Level and High-Level Structure Sources.* 2010.