

Ontology of Altarpieces

Paula Severi and David Ekserdjian

¹ School of Arts - dpme1@leicester.ac.uk

² Department of Mathematics and Computer Science - ps56@mcs.le.ac.uk
University of Leicester. University Road, Leicester, LE1 7RH, UK

Abstract. This report explains the ontology of altarpieces and how it was implemented in the Web Ontology Language (OWL). Altarpieces are paintings located on or behind the altar of churches. On the web, we can find information about paintings and usually we get a description in text format which is understandable by humans but not by machines. The main goal of this ontology is to describe the content of a picture (the figures, the landscape, the symbology, etc) using Description Logic (the Logic behind OWL).

1 Introduction

The ontology of altarpieces is a first step towards building a web database of altarpieces for art specialists. Users of this web database should be able to add and retrieve information about altarpieces.

The information on altarpieces in the Wikipedia is quite incomplete³. We can find information about numerous altarpieces in the websites of certain museums⁴ but many of them are still in the churches where were originally executed or in private collections or in museums without comprehensive websites. Our first goal is then to have a complete database of the most relevant altarpieces.

The second goal is to be able to perform deep and sophisticated queries on altarpieces which are not possible to do with conventional art search engines. A typical request for us would consist in finding out the name and painter of an altarpiece from its description. For instance, we want to get a list of those altarpieces that have the Virgin in the center holding Christ.

In the Web Gallery of Art⁵ we could search for those altarpieces whose text contains the words "Virgin", "center", "holding", "Christ". These gives us a long list of paintings and we have to browse one by one which ones have actually the Virgin in the centre holding Christ. Search engines like google, the one of the Wikipedia or the National Gallery in London cannot do these types of queries because the descriptions of the altarpieces are written in text format which is understandable by humans but it is not understandable by machines. For this, we have to express the text describing each painting in a machine processable

³ <http://www.wikipedia.org/>

⁴ <http://www.nationalgallery.org.uk/>

⁵ <http://www.wga.hu/>

way. This is exactly one of the ideas behind the Semantic Web. The Semantic Web is an evolving development of the World Wide Web in which the meaning (semantics) of information and services on the web is defined, making it possible for the web to understand and satisfy the requests of people and machines to use the web content [5, 2, 11].

As a first step towards the construction of a web database of altarpieces, we have written an ontology of altarpieces in OWL [8, 15, 17, 16, 13, 2] using the editor Protege 3.4 and 4.0 [18]. OWL is considered one of the fundamental technologies underpinning the Semantic Web. It is based on Description Logic [3, 4, 12] and is serialized using RDF/XML syntax [19, 10]. The query language used so far is SPARQL [20].

2 Ontology of Altarpieces

An ontology in OWL is a set of classes and a set of properties (roles, predicates) between those classes⁶. A complete diagram of (only) the classes of the ontology of altarpieces is pictured in Section 9.

2.1 Class of Altarpieces

The main class of this ontology is `Altarpieces` that contains the altarpieces themselves. An altarpiece is a picture representing a religious subject and suspended in a frame on or behind the altar of a church. An example of an altarpiece is shown in Figure 1⁷.

Example 1. We would like to express sentences such as:

1. Raphael has painted an altarpiece called "Sistine Madonna".
2. The altarpiece "Sistine Madonna" painted by Raphael has got the figure of Christ on it.

For the first sentence, we could have the role `hasPainted`. For the second sentence, however, we would need a predicate `hasFigure` with three arguments `SistineMadonna`, `Raphael` and `Christ` and this cannot be expressed directly in OWL. This is because only binary predicates (roles) are allowed in OWL. The solution is to do what is called reification in Description Logic [6] (some case patterns are explained in [14]). The reification of the binary relation `hasPainted` is a class representing the relation `hasPainted`. This is exactly what our class `Altarpieces` is. The class `Altarpieces` is the reification of the relation `hasPainted`. In that reification, we have to create an individual for each pair (`Raphael`, `SistineMadonna`) in `hasPainted`. As a convention, individuals belonging to `Altarpieces` are created by concatenating the name of the painter with the name of the picture as follows

⁶ In Logic, classes and properties correspond to unary and binary predicates respectively.

⁷ The photo was taken from the Wikipedia.



Fig. 1. Sistine Madonna by Raphael

`namepainter_namepicture.`

For example, `Raphael_Sistine_Madonna` is an individual of the class `Altarpieces` that represents the altarpiece named Sistine Madonna and painted by Raphael.

Now we can express the two sentences in Example 1 using only binary roles as follows.

1. `Raphael` `haspainted` `Raphael_Sistine_Madonna`.
2. `Raphael_Sistine_Madonna` `contains` `Christ`.

2.2 Basic Properties on Altarpieces

Individuals belonging to the class `Altarpieces` are identified by concatenating the name of painters with the name of the altarpiece. In other words, the name of the painter and the name of the altarpiece is coded inside each individual. From this codification, we can extract the name of the painter and the name of the altarpiece by means of two predicates `paintedy` and `hasPictureName`. For example,

1. `Raphael_Sistine_Madonna` `paintedy` `Raphael`.
2. `Raphael_Sistine_Madonna` `hasPictureName` "Sistine Madonna".

The two above sentences can be represented diagrammatically as shown in Figure 2. The predicate `paintedy` is called an *Object Property* because the

range is a class, i.e. the class `Painters`. While the predicate `hasPictureName` is called a *Datatype Property* because the range is a string, i.e. the name is in between apostrophes.

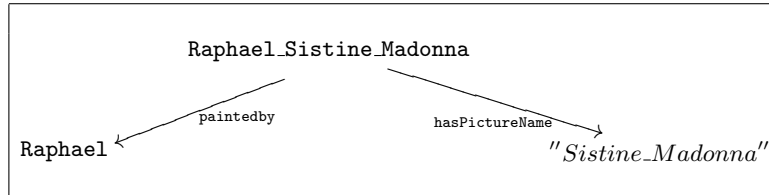


Fig. 2. Extracting the information coded in `Raphael_Sistine_Madonna`

We have defined more basic properties on altarpieces which are listed below. In the following two examples, we give some examples to show how to use these properties.

<u>Object Properties</u>	<u>Datatype Properties</u>
<code>isHousedin</code>	<code>wasPaintedin</code>
<code>wasOriginallyLocatedin</code>	<code>hasPictureHeight</code>
<code>wasPaintedon</code>	<code>hasPictureWidth</code>
<code>wasPaintedwith</code>	

Example 2. The following examples use the object properties shown above.

1. `Raphael_Sistine_Madonna isHousedin Dresden_Gemäldegalerie`
2. `Raphael_Sistine_Madonna wasOriginallyHousedin Piacenza_San_Sisto`
3. `Raphael_Sistine_Madonna wasPaintedon Canvas`
4. `Raphael_Sistine_Madonna wasPaintedwith Oil`

A diagrammatic representation of the above examples is shown in Figure 3.

Example 3. We give some examples showing how to use the above datatype properties.

1. `Raphael_Sistine_Madonna hasPictureWidth 196`
2. `Raphael_Sistine_Madonna hasPictureHeight 265`
3. `Raphael_Sistine_Madonna wasPaintedinDate 'circa 1514'`

These examples are represented in a diagram in Figure 4.

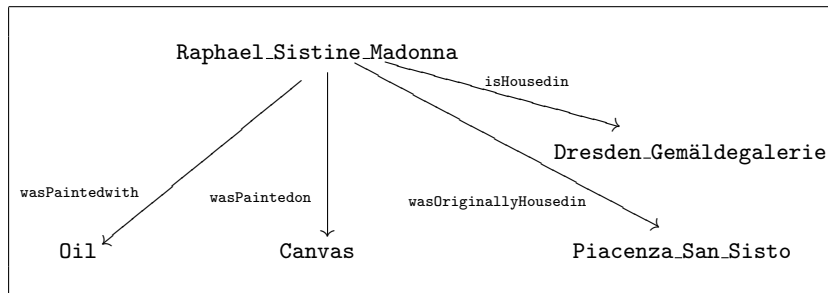


Fig. 3. Facts about Raphael_Sistine_Madonna using Object Properties

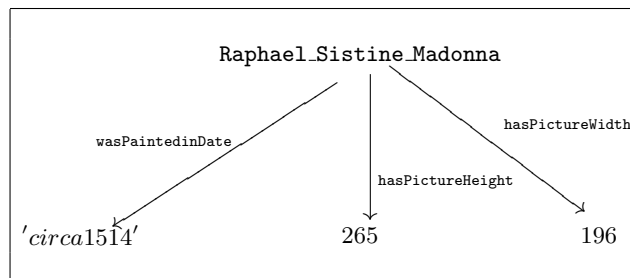


Fig. 4. Facts about Raphael_Sistine_Madonna using Datatype Predicates

2.3 The Image of an Altarpiece

We cannot currently publish the image or photo of an altarpiece. However, we can circumvent this problem by giving a reference. For this, we have a datatype property `hasImageReproduced` whose domain is `Altarpieces` and the range is a string. This predicate gives a reference where the photo of the altarpiece can be found.

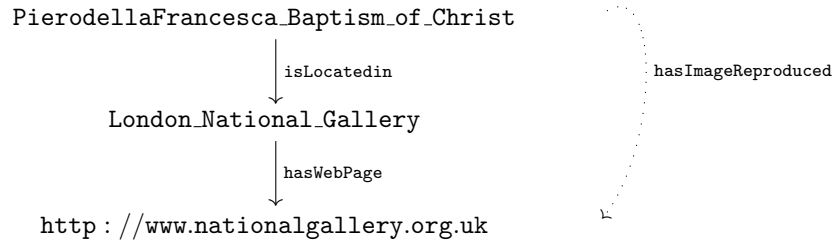
Example 4. In these examples, the strings are references to a book and a webpage respectively.

1. `Raphael_Sistine_Madonna` `hasImageReproduced`
André Chasel.*La Pala d'Altare nel Rinascimento*.1993.
2. `Raphael_Sistine_Madonna` `hasImageReproduced`
'http://en.wikipedia.org/wiki/Sistine_Madonna'

The system could infer a reference to the webpage of a museum where it is located if we state the following axiom:

$$\text{hasImageReproduced} = \text{hasWebPage} \circ \text{isLocatedin}$$

For example,



The system will infer that the website of the museum is the website where we can find a photo of the altarpiece. It is not the actual location of this altarpiece within the website of the museum. For example, the system cannot infer that a photo of the altarpiece can be found in

<http://www.nationalgallery.org.uk/paintings/piero-della-francesca-the-baptism-of-christ>.

2.4 Template of an Altarpiece

Instead of publishing the image of the altarpiece itself, we can publish a schema or a template as shown in [7], pages 257-287. The template is a picture showing the layout of the icons and it may be considered as the fingerprint of the altarpiece. For this, we have the predicate `hasTemplate` that relates an altarpiece to an URI. The URI is a reference to the template. For a onefield altarpiece, the template could show the distribution of the figures. For a manyfield altarpiece, it can show the distribution of the fields.

2.5 Necessary Conditions for Altarpieces

We now consider the following two sentences.

1. Altarpieces are pictures that have some religious figure.
2. The height of an altarpiece is bigger than 120.

The above two sentences are expressed in OWL 2 by saying that `Altarpieces` is a subclass of the following class:

```
Pictures
and contains some Religious
and hasPictureHeight only int[>= "120"^^integer]
```

Note that the last condition could not be expressed in OWL 1 unless we use an SWRL rule.

```
OneFieldAltarpieces(?x)  $\wedge$  hasPictureHeight(?x, ?y)
 $\rightarrow$  swrlb:greaterThanOrEqual(?y, 120)
```

2.6 Subclasses of Altarpieces

The class of altarpieces has two main subclasses (see Figure 5).

1. `OneFieldAltarpieces` are altarpieces in one frame or field.
2. `ManyFieldAltarpieces` contains altarpieces composed by several fields.

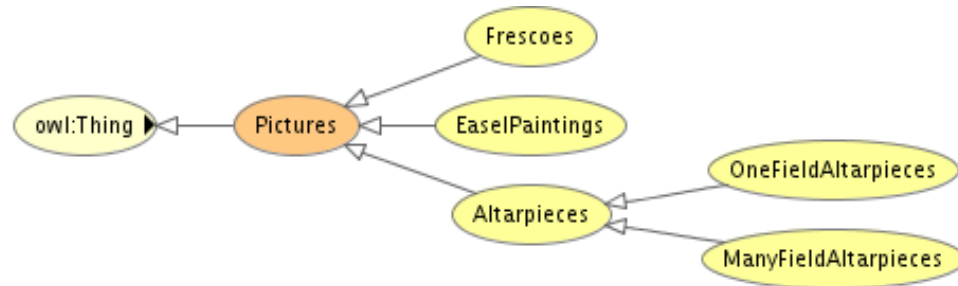


Fig. 5. The Class of `Pictures` and its subclasses

3 Class of Elements

The elements of an altarpiece are all things we can find on it such as figures, objects or attributes. Figure 6 shows the class `Elements` and its subclasses.

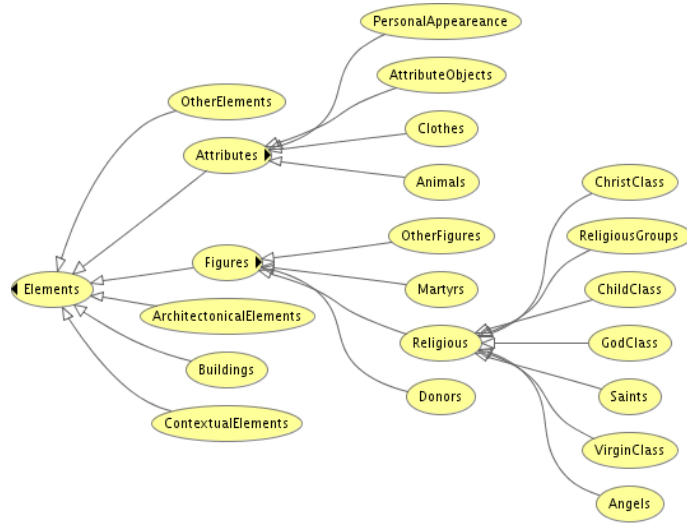


Fig. 6. The Class of Elements

3.1 Subclass of Attributes

Attributes are usually objects that identify a religious figure or a donor, e.g. trumpet, wings, palm, cross, etc. The attributes `cross`, `palm` and `trumpet` belong to the subclass `Objects`. The attribute `wings` belongs to the subclass `PersonalAppearance`.

The relation between attributes and figures is expressed by the role `identifies` or its inverse `isidentifiedby`. For example, we express that the sword is an attribute identifying Saint Michael as follows.

`sword identifies Saint Michael.`

3.2 Subclass of Figures

In the class `Figures`, we have individuals that represent the actual people of the altarpieces such as `Saint Michael` or `Saint John the Baptist`. The subclasses `GodClass`, `VirginClass`, `ChristClass` and `ChildClass` are all singletons.

3.3 Subclass of Martyrs

Martyrs are defined as those figures that are identified by the palm.

`isidentifiedby has palm`

Since we have introduced this definition, it is enough to state that `St Sebastian` `isidentifiedby palm` and the system will infer that he is a martyr.

3.4 Other Subclasses of Elements

The curtains at the top of the altarpiece "Sistine Madonna" is represented by the element `curtains` which belongs to `ContextualElements`. The papal tiara at the bottom could be represented by the element `Papal Tiara` which belongs to `OtherElements`.

A niche could be represented by the individual `niche` which belongs to the class `ArchitectonicalElements`.

The temple in the altarpiece "Marriage of the Virgin" (see Figure 9) could be represented by the element `temple` which belongs to the class `Buildings`.

4 Scene Parts for One Frame Altarpieces

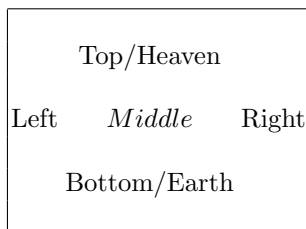
The elements of an altarpiece are described using the predicate `contains` or its inverse `appearson`. For instance, we can express that `Christ` is on the altarpiece `Raphael_Sistine_Madonna` as follows.

`Raphael_Sistine_Madonna contains Christ.`

If we want to be more specific and say that Christ appears in the middle of `Raphael_Sistine_Madonna`, we can say

`Raphael_Sistine_Madonna hasinMiddle Christ.`

We have several predicates to express where the element is placed on the scene. The scene of a one frame altarpiece is divided in parts and for each part we have a predicate. The parts of a scene are the following.



We also have Background/Foreground. The complete list of roles and their inverses that can be used to place elements on the scene is given next.

Role	Inverse
<code>hasatTop</code>	<code>isatTopof</code>
<code>hasatBottom</code>	<code>isatBottomof</code>
<code>hasonLeft</code>	<code>isonLeftof</code>
<code>hasonRight</code>	<code>isonRightof</code>
<code>hasinHeaven</code>	<code>isinHeavenof</code>
<code>hasinEarth</code>	<code>isinEarthof</code>
<code>hasinMiddle</code>	<code>isinMiddleof</code>
<code>hasinBackground</code>	<code>isinBackgroundof</code>
<code>hasinForeground</code>	<code>isinForegroundof</code>

All roles on the left column are subroles of `contains` (see Figure 7) and the roles on the right column are subroles of `appearson` (see Figure 8).



Fig. 7. The property `contains` and its subproperties

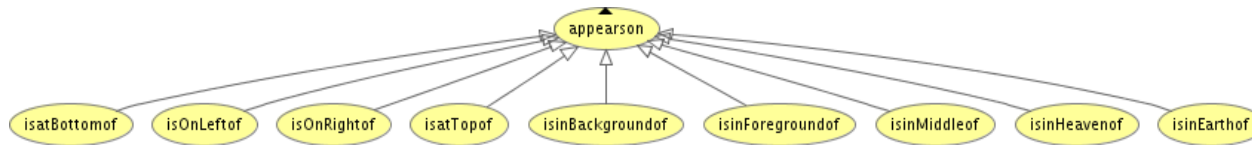
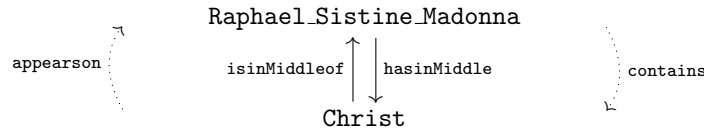


Fig. 8. The property `appearson` and its subproperties

Inverses and subproperties are useful because some information may not be necessary to be introduced as the system infers it. For example,



From the fact that `Raphael_Sistine_Madonna hasinMiddle Christ`, the system will infer that

`Raphael_Sistine_Madonna contains Christ`

because `hasinMiddle` is a subrol of `contains`. Since `appearson` is inverse of `contains`, the system will also infer that:

`Christ appearson Raphael_Sistine_Madonna .`

5 Relation between Elements

We would like to give more complex descriptions of an altarpiece.

Example 5. Suppose we want to express the following sentences over the altarpiece `Raphael_Marriage_of_Virgin` in Figure 9 ⁸.

⁸ The photo was taken from the Wikipedia [9].

1. **Joseph** is placing the ring on the **Virgin's finger**.
2. **Joseph** is holding the **flowering staff** in his left hand.
3. The other **suitors** are holding **dry staffs**.
4. Two of the **suitors**, disappointed, are breaking their **staffs**.

Note that the above sentences are all relations between three arguments since they involve a figure, an object and the altarpiece. This means that to express these sentences, we need trinary predicates such as **holds**, **placesringon**, **breaks**. The problem is that OWL does not allow predicates of three or more arguments.



Fig. 9. Marriage of Virgin by Raphael

5.1 First Solution: Multiple Reification

One solution is to do reification for each non-binary predicate [6, 14]. But this solution involves introducing many extra classes, individuals and predicates. For the first example above, we would have to introduce a class `Cholds` and an individual such as `Saint_Joseph-flowering_staff-Raphael_Marriage_of_Virgin` representing the triplet in the relation. We would also need to introduce the three predicates for the projections to get the components.

We want to avoid introducing so many classes and individuals. Not only because they imply more workload for the user but also because they are an interesting source of mistakes.

5.2 Second Solution: FigurewithAttributes

The second solution consists in introducing only one extra class. This class is called `FigurewithAttributes` and it is the reification of the relation `isidentifiedby`. In this class, we have individuals that represent the pair composed by a figure and an attribute. For instance, an element of the class `FigurewithAttributes` is `Saint Joseph_flowering staff` which represents the pair composed by `Saint Joseph` and `flowering staff`. The first sentence at the beginning of this section can be expressed using the predicate `appearson` as follows.

`St Joseph flowering staff aaf appearson Raphael_Marriage_of_Virgin.`

This solution is **unsatisfactory** because we cannot express relations between figures or between figures and other elements that are not attributes such as the second or fifth sentences in Example 5.

5.3 Third Solution: ElementswithinAltarpieces

Similarly to the second solution, the third solution consists in introducing only one extra class called `ElementswithinAltarpieces` that contains individuals of the form `nameofaltarpiece-nameofelement`. An individual of this class represents a pair composed by the altarpiece and the element that appears on it. This class is the reification of the predicate `appearson`. For example, individuals of this class are

```
Raphael_Marriage_of_Virgin-Joseph
Raphael_Marriage_of_Virgin-Virgin
Raphael_Sistine_Madonna-papal tiara
Raphael_Sistine_Madonna-balustrade
```

Figure 10 shows the properties that can be used to relate elements with other elements in an altarpiece. Note that `holdsonrighthand` is a subproperty of `holds` which is a subproperty of `hasGot`.

Fig. 10. The property `relatedtoElement` and its subproperties

Example 6. The sentences in Example 5 are expressed using the above defined class and properties as follows.

1. `Raphael_Marriage_of_Virgin-Saint_Joseph holds Raphael_Marriage_of_Virgin-flowering staff.`
2. `Raphael_Marriage_of_Virgin-Joseph putsrington Raphael_Marriage_of_Virgin-Virgin`
3. `Raphael_Marriage_of_Virgin-Joseph holdslefthand Raphael_Marriage_of_Virgin-flowering_staff`

4. Raphael_Marriage_of_Virgin-suitors hold
Raphael_Marriage_of_Virgin-dry_staff
5. Raphael_Marriage_of_Virgin-twosuitors break
Raphael_Marriage_of_Virgin-dry_staff

The sentences in the previous example can be simplified. The first argument of the predicate must belong to `ElementswithinAltarpieces` but the second argument can belong to `Elements`. Hence, what we can do is to delete the prefix `Raphael_Marriage_of_Virgin` of the second argument from all sentences of the previous example.

Example 7. Following this simplification, we get

1. Raphael_Marriage_of_Virgin-Saint_Joseph holds flowering_staff.
2. Raphael_Marriage_of_Virgin-Joseph putsringon Virgin
3. Raphael_Marriage_of_Virgin-Joseph holdslefthand
flowering_staff
4. Raphael_Marriage_of_Virgin-suitors hold
dry_staff
5. Raphael_Marriage_of_Virgin-twosuitors break
dry_staff

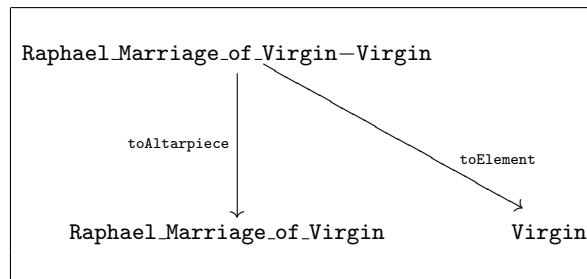


Fig. 11. Extracting the information coded in the individual `Raphael_Marriage_of_Virgin-Virgin`

As for any reification, we have to provide the projections for getting the components coded in each individual of `ElementswithinAltarpieces`. We call them `toAltarpiece` and `toElement`. An example is shown in Figure 11.

The predicate `toAltarpiece` is functional. However, the predicate `toElement` is not because an unknown figure may correspond to several figures if the information about it is incomplete or ambiguous.

The hierarchy of the class `ElementswithinAltarpieces` is a copy of the hierarchy of the class `Elements`. Compare Figure 12 with Figure 6.



Fig. 12. The class `ElementswithinAltarpieces` and its subclasses

5.4 Comparison of solutions

The third solution is obviously more expressive than the second one. Comparing the number of classes, the third solution is better than the first one since the third solution introduces only one extra class while the first solution introduces a class for each new predicate. However, comparing the number of individuals, the third solution is better than the first one only if we have to relate the same individuals in more than one way. If besides referring to putting the ring, we want to say more things about Joseph and the Virgin, the first solution obliges us to add more individuals while the number of individuals for the third solution remains constant.

The third solution has another important advantage over the previous ones. We can have the flexibility of having unknown figures on an altarpiece. The system could deduce the name of the unknown figures from their attributes. It could also give several options if it is not certain which figure it is being referred to. In that case, instead of giving explicit names, we would have to enumerate the figures in the altarpiece as follows.

Raphael_Marriage_of_Virgin-Figure1
 Raphael_Marriage_of_Virgin-Figure2

The system could deduce if Figure1 is the Virgin depending on the information used to describe her. The system could also give several options in case the information does not determine the name of the figure.

6 Class of AltarpiecesFields

Altarpieces belonging to `ManyFieldAltarpieces` can have many fields. The altarpiece `Pietro_Lorenzetti_Birth_of_Virgin` in Figure 13 has three fields ⁹.

⁹ The photo was taken from the Web Gallery of Art [1].



Fig. 13. Birth of Virgin by Pietro Lorenzetti

For these fields, we introduce the class `AltarpieceFields`. A field is identified by the name of the altarpiece and a number. For example, for the many field altarpiece `Pietro_Lorenzetti_Birth_of_Virgin` which has three fields we add the following three individuals to the class `AltarpieceFields`.

```
Pietro_Lorenzetti_Birth_of_Virgin_Field1  
Pietro_Lorenzetti_Birth_of_Virgin_Field2  
Pietro_Lorenzetti_Birth_of_Virgin_Field3
```

We express that the altarpiece `Pietro_Lorenzetti_Birth_of_Virgin` has the three fields mentioned above using the predicate `hasField` as follows.

```
Pietro_Lorenzetti_Birth_of_Virgin hasField  
Pietro_Lorenzetti_Birth_of_Virgin_Field1
```

```
Pietro_Lorenzetti_Birth_of_Virgin hasField  
Pietro_Lorenzetti_Birth_of_Virgin_Field2
```

```
Pietro_Lorenzetti_Birth_of_Virgin hasField  
Pietro_Lorenzetti_Birth_of_Virgin_Field3
```

We could have three classes: `LeftPanels`, `CentralPanels` and `RightPanels` for the first, second and third field.

Example 8. 1. It is signed and dated, the inscription on the frame below the central panel is: PETRUS LAURENTII DE SENIS ME PINXIT MCC-CXLII.

```
Pietro_Lorenzetti_Birth_of_Virgin hasInscriptiononFrame  
PETRUS LAURENTII DE SENIS ME PINXIT MCCCXLII.
```

2. The second field contains Saint Anne, the newborn Mary and midservants.

```
Pietro_Lorenzetti_Birth_of_Virgin_Field2 contains Saint_Anne  
Pietro_Lorenzetti_Birth_of_Virgin_Field2 contains midservants  
Pietro_Lorenzetti_Birth_of_Virgin_Field2 contains newbornMary
```

3. In the main room St Anne, who has just given birth, is lying in bed

```
Saint_Anne liesinbedon Pietro_Lorenzetti_Birth_of_Virgin_Field2  
Saint_Anne hasgivenbirthon Pietro_Lorenzetti_Birth_of_Virgin_Field2
```

4. The midservants wash the newborn Mary.

```
Pietro_Lorenzetti_Birth_of_Virgin_midservants wash newbornMary
```

5. The child's father Joachim is receiving the happy news from a boy in the anteroom.

```
Pietro_Lorenzetti_Birth_of_Virgin_Joachim receivesHappyNewsfrom Boy
```

```
Joachim appearson Pietro_Lorenzetti_Birth_of_Virgin_Field1  
Boy appearson Pietro_Lorenzetti_Birth_of_Virgin_Field1
```

The figures that are in a many field altarpiece are the figures that appear in its fields. This is expressed in OWL 2 using composition of roles.

```
hasField o contains = contains
```

This means that we only describe the figures for each field and the system will infer the figures that occur in the altarpiece.

If there is a religious figure in a field of an altarpiece, the system will infer that there is a religious figure on the altarpiece itself.

7 Some Queries implemented in SPARQL

We give a list of queries and express them in SPARQL [20].

1. The most interesting query of all is to find out the altarpiece(s) that correspond to certain description. For instance, list the altarpieces that have Christ on the left and Saint John the Baptist on the right.

```

PREFIX altar: <http://www.owl-ontologies.com/AltarPieces.owl#>
SELECT *
WHERE { OPTIONAL {?Picture altar:hasonLeft altar:Christ.}
        OPTIONAL {?Picture
                    altar:hasonRight altar:Saint John the Baptist.}
      }

```

2. The altarpieces that have no signature

```

PREFIX altar: <http://www.owl-ontologies.com/AltarPieces.owl#>
SELECT ?x
WHERE { ?x altar:hasSignature false}

```

3. The altarpieces on panels

```

PREFIX altar: <http://www.owl-ontologies.com/AltarPieces.owl#>
SELECT ?x
WHERE { ?x altar:wasPaintedon altar:Panel}

```

4. The panel altarpieces by Parmigianino

```

PREFIX altar: <http://www.owl-ontologies.com/AltarPieces.owl#>
SELECT ?x
WHERE { ?x altar:wasPaintedon altar:Panel.
        ?x altar:wasPaintedby altar:Parmigianino}

```

5. The altarpieces by Parmigianino with no signature

```

PREFIX altar: <http://www.owl-ontologies.com/AltarPieces.owl#>
SELECT ?x
WHERE { ?x altar:hasSignature false.
        ?x altar:wasPaintedby altar:Parmigianino}

```

6. The altarpieces whose name contains the word "Saint".

```

PREFIX altar: <http://www.owl-ontologies.com/AltarPieces.owl#>
SELECT ?x
WHERE { ?x altar:hasPictureName ?y
        FILTER regex(?y, "Saint")
      }

```

7. Altarpieces not wider than 120

```

PREFIX altar: <http://www.owl-ontologies.com/AltarPieces.owl#>
SELECT ?x ?y
WHERE { ?x altar:hasPictureWidth ?y
        FILTER (?y <=120)
      }

```

8. A table with the columns painter, name of picture, paintmedia, surface, height, width

```

PREFIX altar: <http://www.owl-ontologies.com/AltarPieces.owl#>
SELECT *
WHERE {?Painter altar:hasPainted ?Picture.
       ?Picture altar:wasPaintedwith ?PaintMedia.
       ?Picture altar:wasPaintedon ?Surface.
       ?Picture altar:hasPictureHeight ?height.
       ?Picture altar:hasPictureWidth ?width
      }

```

9. Same table as before but including pictures without surface and size.

```

PREFIX altar: <http://www.owl-ontologies.com/AltarPieces.owl#>
SELECT *
WHERE {?Painter altar:hasPainted ?Picture.
       ?Picture altar:wasPaintedwith ?PaintMedia.
       OPTIONAL{?Picture altar:wasPaintedon ?Surface.}
       OPTIONAL{?Picture altar:hasPictureHeight ?height.}
       OPTIONAL{?Picture altar:hasPictureWidth ?width}
      }

```

10. Describe the picture Parmigianino Christ and the Baptist.

```

PREFIX altar: <http://www.owl-ontologies.com/AltarPieces.owl#>
SELECT *
WHERE {OPTIONAL{Parmigianino_Christ_and_the_Baptist
              hasonLeft ?onLeft.}
       OPTIONAL{Parmigianino_Christ_and_the_Baptist
              hasinForeground ?onLeft.}
      }

```

There is one query that involves counting and cannot be expressed in SPARQL.

1. Percentage of altarpieces without signature.

8 Conclusions and Future Work

The query language SPARQL has some important limitations. The first important limitation is that we cannot do queries for doing statistics, i.e. queries that involve counting the number of tuples satisfying certain condition. The second limitation is that SPARQL does not do deductions and make use of the axioms added to the ontology. This is because SPARQL is a query language designed for RDF but not for OWL. For instance, suppose we declared the property `ancestor` as transitive. If we could make use of the reasoner, it will be sufficient to add to the ontology that

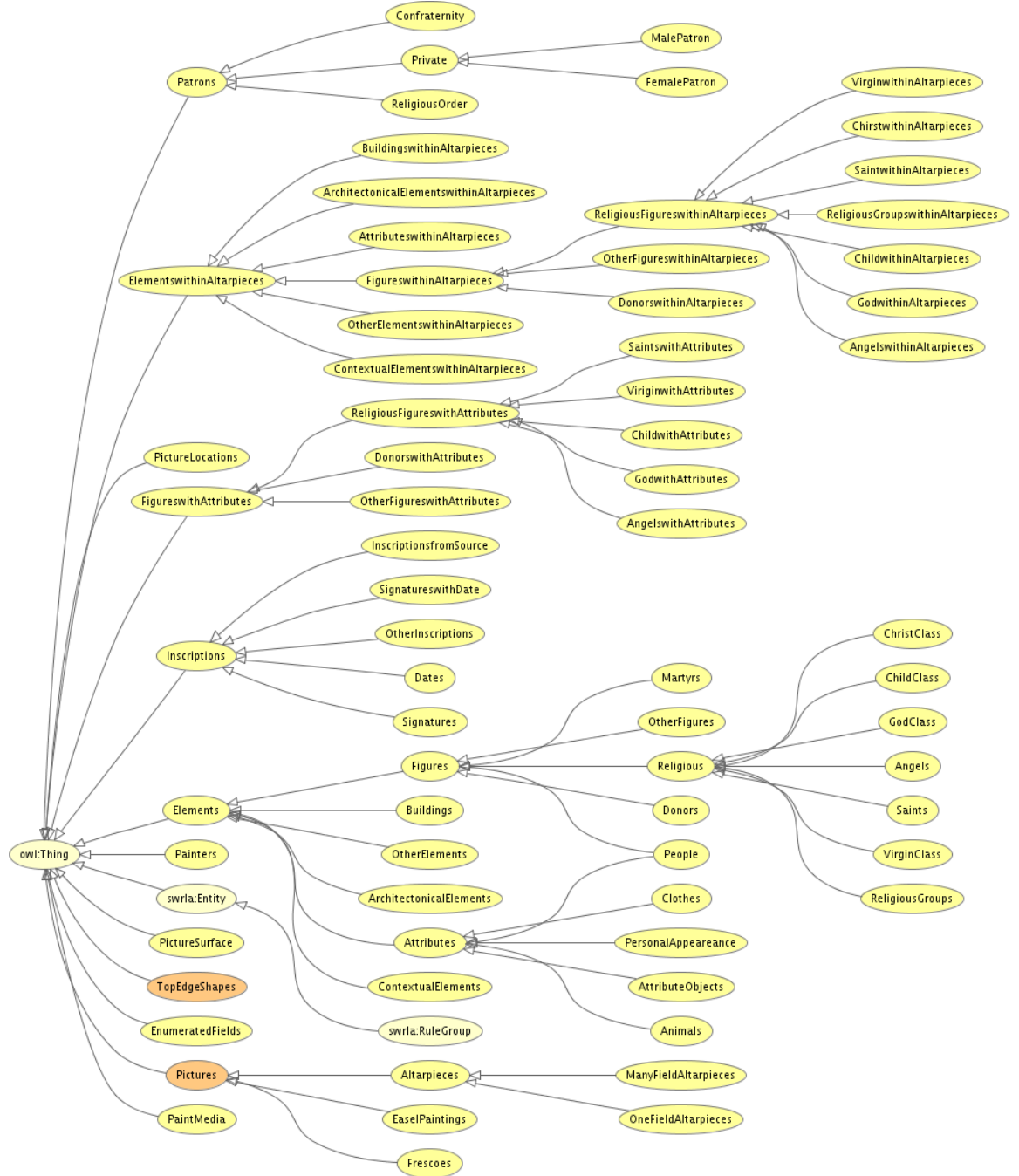
1. John ancestor Tom
2. Tom ancestor Mary.

Any OWL reasoner would deduce that `John ancestor Mary`. However, when we do a query in SPARQL to know all the ancestors of `John` it will give only one answer which is `Tom`. The third limitation of SPARQL is that it is not user-friendly.

It is necessary to build a web interface that allows us to do queries in a friendly environment like the Web Gallery of Art or the London National Gallery. The style of queries will be more clever and sophisticated than the ones on these sites. For implementing the queries, we will be using the Protege-API which gives the flexibility and capability of performing any queries (including the ones involving statistics). Through the Protege-API, we can also make use of the reasoner.

Acknowledgements We would like to thank Jose Fiadeiro for establishing the contact between the School of Arts and the Computer Science Department of the University of Leicester. We are in debt with Fer-Jan de Vries for suggesting to reify the class `ElementswithinAltarpieces` instead of `FigurewithAttributes`, a key idea in the design of this ontology. We would also like to thank Yi Hong and Monika Solanki for installing Protege in David's Macintosh.

9 Diagram of Asserted Ontology



10 Diagram of Inferred Ontology



References

1. Web Gallery of Art. <http://www.wga.hu/>.
2. G. Antoniou. *A Semantic Web Primer*. MIT, 2008.
3. F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, Applications*. CUP, 2003.
4. F. Baader, I. Horrocks, and U. Sattler. Description Logics. In F. van Harmelen, V. Lifschitz, and B. Porter, editors, *Handbook of Knowledge Representation*. Elsevier, 2007.
5. T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American Magazine*, 2001.
6. D. Calvanese and G. De Giacomo. Expressive Descriptions Logics. In *The Description Logic Handbook*, pages 193–236. CUP.
7. A. Chasel. *La Pala d'Altare nel Rinascimento*. Garzanti Editore, 1993.
8. B. Cuenca Grau, I. Horrocks, B. Motik, B. Parsia, P. Patel-Schneider, and U. Sattler. OWL 2: The next step for OWL. *J. of Web Semantics*, 6(4):309–322, November 2008.
9. Sanzio da Urbino R. Marriage of virgin. [http://en.wikipedia.org/wiki/The_Marriage_of_the_Virgin_\(Raphael\)](http://en.wikipedia.org/wiki/The_Marriage_of_the_Virgin_(Raphael)).
10. J. Hjelm. *Creating the semantic Web with RDF*. Wiley, 2001.
11. I. Horrocks. Ontologies and the semantic web. *Communications of the ACM*, 51(12):58–67, December 2008.
12. I. Horrocks, O. Kutz, and U. Sattler. The Even More Irresistible *SR_OI_Q*. In *Proc. of the 10th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2006)*, pages 57–67. AAAI Press, 2006.
13. L.W. Lacy. *OWL : representing information using the web ontology language*. Trafford, 2005.
14. N. Noy and A. Rector. N-ary relations. <http://www.w3.org/TR/swbp-n-aryRelations/>.
15. M.K. Smith, C. Welty, and D.L. (eds.) McGuinness. W3C: OWL Web Ontology Language Guide. <http://www.w3.org/TR/owl-guide/>, 2004. W3C Recommendation.
16. W3C. OWL 2 Overview. <http://www.w3.org/TR/owl2-overview/>.
17. W3C. OWL working group. http://www.w3.org/2007/OWL/wiki/OWL_Working_Group.
18. W3C. Protege-OWL.Editor. <http://protege.stanford.edu/overview/protege-owl.html>.
19. W3C. RDF. <http://www.w3.org/RDF/>.
20. W3C. SPARQL. Query Language. <http://www.w3.org/TR/rdf-sparql-query/>.