Guiding the Representation of $n$-ary Relations in Ontologies through Aggregation, Generalization and Participation

Paula Severi*,** José Fiadeiro*,** David Ekserdjian*

*Department of Computer Science, University of Leicester, Leicester, LE1 7RH, United Kingdom
**Department of History of Art and Film, University of Leicester, LE1 7RH, United Kingdom

Abstract

We put forward a methodological approach aimed at guiding ontologists in choosing which relations to reify. Our proposal is based on the notions of aggregation, generalization and participation as used in conceptual modelling approaches for database design in order to represent situations that, normally, would require non-binary relations or complex integrity constraints. In order to justify our approach, we provide mathematical definitions of the constructs that we propose and use them to analyse the extent to which they can be implemented in languages such as OWL. A number of results are also proved that attest to the soundness of the methodological guidelines that we propose. The feedback received from using the method in a real-world situation is that it offers a more controlled use of reification and a closer fit between the resulting ontology and the application domain as perceived by an expert.

Keywords: Aggregation, generalization, $n$-ary relations, ontology engineering, participation dependency, reification

1. Introduction

A well-known limitation of OWL 2 (Web Ontology Language) is that only binary relations between classes can be represented [1, 2, 3]. In practice, relations of arbitrary arity are quite common and they have to be represented in OWL in an indirect way by coding them as classes1. In the literature on Description Logic (DL) [4], the class codifying a relation $\rho$ is called the reification of $\rho$ 2. As any codification, reification requires extra work in addition to ‘simple’ modelling, which can make it quite impractical (and unintuitive), especially when performed by people who are not ‘experts’: extra classes, predicates, individuals and axioms [5] need to be introduced and, as the number of classes increases, ontologies can become very difficult to read and understand, mainly because this additional information often masks the concepts and structures that it encodes. That is, there is a mismatch between the layer of abstraction at which domain modellers work and that of the representation where information is encoded, which is particularly harmful when we want to extend and reuse ontologies. Ontologies that are simple and easy to read are also more likely to be reused.

In this paper we detail and expand on a method of Ontology Engineering that we introduced in [6] for simplifying ontologies. Our method is inspired by notions previously proposed in database modelling and construction for increasing the “understandability of relational models by the imposition of additional semantic structure” [7]: aggregation and generalization [8] and dependencies between relations [9]. Although, in ontologies, the technical problems that arise are not necessarily the same as those of relational databases, the methodological issues are similar in the sense that the solution to our problem lies first of all in helping modellers to conceptualize the real world in a way that can lead to a better representation, and then offering them a mechanism for implementing these semantic structures in ontologies. By ‘better’ we mean a more controlled use of reification and a closer fit between the resulting ontology and the real-world domain as perceived by an expert. Our method also makes ontologies easier to extend, in particular to reuse existing reifications when adding new relations to an ontology, which is essential for supporting an incremental process of ontology engineering.

Having this in mind, we start by motivating the problem using the case study that led us to investigate the representation of $n$-ary relationships — an ontology of 16th-century Italian altarpieces [10]. In Section 3, we discuss a formal, set-theoretical notion of aggregation and the way that it can be implemented in ontologies through reification. Then, in Section 4, we show how aggregation as
2. Motivation

In order to illustrate some of the problems that may arise from the limitations of having to encode n-ary relations through reification and the method that we propose to minimize them, we use the Ontology of Altarpieces [12] — a joint project between the Departments of Computer Science and History of Art and Film at the University of Leicester. This case study is a good example of a domain in which n-ary relations arise quite naturally and frequently. There are a fair number of 16th-century altarpieces in Italy [10], each of which with a rich set of variations in the way they depict the subject matters, which is precisely what motivated this project as the typical tool of the art expert — the spreadsheet — is not powerful enough to analyse their properties.

Suppose that we want to express the following knowledge as produced in natural language by an art expert:

Joseph is holding the flowering staff in the altarpiece called “The Marriage of the Virgin” by Raphael

The natural representation of this domain property is in terms of a relation holds of arity 3:

\[
\text{holds}(\text{raphael} \triangleright \text{marriageofvirgin}, \text{joseph}, \text{floweringstaff}) \in holds
\]

where raphael \triangleright marriageofvirgin is an identifier for the altarpiece “The Marriage of Virgin” painted by Raphael. In the ontology of altarpieces, we follow the traditional practice of art historians (the domain experts) and use identifiers of the form painter*title for altarpieces where painter is the name of the painter and title is the designation of the picture.

Our notion of participation differs from the notion of participation used in the ER model which relates an entity with a relation.

One of the queries we are interested in is in finding out the names and painters of the altarpieces that satisfy certain description. For example, what are the altarpieces that have someone holding a flowering staff?

If the painter is not known, we associate with it a special individual anonymous1, anonymous2, etc. If a painter had two paintings under the same title such as “Coronation of the Virgin” by Lorenzo Monaco in the National Gallery, we enumerate the titles such as coronationofvirgin1 and coronationofvirgin2.

\[
\text{holds} = \\
\{(\text{raphael} \triangleright \text{marriageofvirgin}, \text{joseph}, \text{floweringstaff}) \}
\]

However, reifying holds is not necessarily the right decision that a modeller should make. In order to understand why, consider that subsequent additions of triples leads to the following extension of the relationship holds.

\[
\text{holds} = \\
\{(\text{raphael} \triangleright \text{marriageofvirgin}, \text{joseph}, \text{floweringstaff}),\
(\text{corregio} \triangleright \text{foursaints}, \text{peter}, \text{keys}),
(\text{corregio} \triangleright \text{foursaints}, \text{peter}, \text{book}),
(\text{roselli} \triangleright \text{madonnaandsaints}, \text{catherine}, \text{palm}),
(\text{roselli} \triangleright \text{madonnaandsaints}, \text{catherine}, \text{book}),
(\text{dabrescia} \triangleright \text{madonnaandchild}, \text{catherine}, \text{sword}),
(\text{dabrescia} \triangleright \text{madonnaandchild}, \text{catherine}, \text{palm})
\}
Through reification, we end up with 8 individuals, each coding one of the triples, and 24 assertions for establishing the connections depicted in Figure 2. A simple inspection of the triples suggests that a simpler representation could be achieved by coding instead the pairs

\[(raphael \ast marriageofvirgin, joseph), \]
\[(corregio \ast foursaints, peter), \]
\[(roselli \ast madonnaand saints, catherine), \]
\[(dabrescia \ast madonnaandchild, catherine)\]

say with individuals \(f_1, f_2, f_3\) and \(f_4\) respectively, and then use a binary relation to represent \(holds\):

\[
\begin{align*}
\text{holds} &= \{(f_1, \text{floweringstaff}), \\
& (f_1, \text{ring}), \\
& (f_2, \text{keys}), \\
& (f_2, \text{book}), \\
& (f_3, \text{palm}), \\
& (f_3, \text{book}), \\
& (f_4, \text{sword}), \\
& (f_4, \text{palm})\}
\end{align*}
\]

The main simplification arises from the fact that we were able to transform the ternary relation \(holds\) into a binary relation \(\hat{\text{holds}}\), which does not need to be reified: it can be represented directly in OWL through a role.

In addition to saving on the number of individuals (4 instead of 8) and assertions (8 role assertions for the reification instead of 24) \(^6\), we argue that this simplification can be justified in methodological terms as it brings the representation closer to the domain. Indeed, a conceptual model of the whole domain would reveal a richer semantic structure that is not captured in the simple diagram given in Figure 1. More precisely, a wider conceptual model of the domain of altarpieces as depicted in Figure 3 shows that the entities Altarpieces, Figures and Objects are involved in more complex relationships. On the one hand, \(holds\) is actually a binary relationship between Objects and the ‘aggregation’ of a relationship \(hasFigure\) between Altarpieces and Figures (the aggregation is depicted by a box surrounding the relationship subdiagram, which is the notation usually adopted in conceptual modelling approaches). On the other hand, \(hasFigure\) has a ‘descriptive attribute’ (functional relationship) that returns the location of the figure in the altarpiece — one of right, left, center, top, bottom, heaven or earth.

In summary, the simplification discussed above corresponds, effectively, to reifying \(hasFigure\) and representing \(holds\) as a binary relation as depicted in Figure 3. Our purpose in this paper is, precisely, to investigate how far the reification of \(hasFigure\) can be taken to represent the aggregation of the relation as understood in conceptual modelling and how this and other constructions (such as descriptive attributes) that have been proposed by the database community 30 years ago can be used for developing simpler and reusable ontologies from conceptual models. To state the obvious, one should not take a blind approach to the representation of the domain and reify relations as they come: the complexity of the ontologies thus generated would be even beyond skilled computer scientists, let alone domain experts. As in database design, one should build a conceptual model of the domain before starting coding in OWL or any other language, and follow a sound methodology, as outlined in this paper, to generate or reuse code.

\(^6\)There are an additional 8 role assertions for representing the relation itself but these are ‘natural’ in the sense they do not arise from a reification. Still, this would mean a total of 16 assertions instead of 24.
3. Aggregation in Set Theory vs Reification in OWL

Aggregation, as defined in [7], refers to an abstraction in which a relationship between objects is regarded as a higher-level object. The intention, as stated therein, was to adapt cartesian product structures (as proposed by T. Hoare for record structures in programming languages [13]) to be used in the context of relational models. Although a formal definition was not given therein as a semantics for the abstraction, we found it useful to advance the notion and, on the other hand, we can relate it to the mechanism of reification.

3.1. Aggregation and Reification of Binary Relations

In this section, we propose a formalisation for the notions of aggregation and reification of binary relations. It may seem strange that we define these notions for binary relations when the motivation for the paper is the representation of non-binary ones. The reasons for doing so are twofold. On the one hand, the case of binary relations is simpler and easier to understand. On the other hand, and more importantly, the method we propose shows that, sometimes, it is convenient to reify binary relations as a means of simplifying the representation of non-binary ones, as illustrated in the previous section. Similar happening happens in [14] where, in some cases, a ternary relation is not directly reified but represented indirectly by reifying a binary relation.

We start by defining the concept of aggregation in the context of Set Theory and then that of reification in the context of OWL. We then analyse the extent to which aggregations can be implemented in OWL as reifications.

**Definition 1.** Let \( \Delta_1, \Delta_2 \subseteq \Delta \) be sets and \( \rho \subseteq \Delta_1 \times \Delta_2 \) be a binary relation. An aggregation of \( \rho \) over \( \Delta \) is a set \( \Delta_\rho \subseteq \Delta \) together with two (total) functions \( \sigma_1 \) and \( \sigma_2 \) (called attribute functions) from \( \Delta_\rho \) to \( \Delta_1 \) and \( \Delta_2 \), respectively, such that the following conditions hold:

1. For all \( r \in \Delta_\rho \), \( (\sigma_1(r), \sigma_2(r)) \in \rho \) — i.e., there is no ‘junk’ in \( \Delta_\rho \).
2. For all \( (x_1, x_2) \in \rho \), there exists \( r \in \Delta_\rho \) such that \( \sigma_1(r) = x_1 \) and \( \sigma_2(r) = x_2 \) — i.e., the aggregation covers the whole relation \( \rho \).
3. For all \( r_1, r_2 \in \Delta_\rho \), if \( \sigma_1(r_1) = \sigma_1(r_2) \) and \( \sigma_2(r_1) = \sigma_2(r_2) \) then \( r_1 = r_2 \) — i.e., there is no ‘confusion’: every tuple of the relation has a unique representation as an aggregate (see Figure 4).

Note that the aggregation is the set \( \Delta_\rho \) together with the attribute functions \( \sigma_1 \) and \( \sigma_2 \), i.e. the whole structure \( (\Delta_\rho, \sigma_1, \sigma_2) \).

Throughout the paper, we use the Greek alphabet for metavariables ranging on sets and relations in Set Theory. Capital letters \( \Delta, \Gamma, \ldots \) are used for sets and lower case letters \( \rho, \sigma, \ldots \) for relations.

It is easy to see that aggregations are unique up to isomorphism.

Example 1. Consider the following binary relation:

\[
\text{hasPainted} = \{(\text{raphael}\text{*marriageofvirgin}), \text{(corregio}\text{*foursaints})\}
\]

An aggregation of \( \text{hasPainted} \) is the set

\[
\text{Altarpieces} = \\
\{\text{raphael}\text{*marriageofvirgin}, \text{corregio}\text{*foursaints}, \text{roselli}\text{*madonnaandsaints}, \text{dabrescia}\text{*madonnaandchild}\}
\]

together with the attribute functions \text{painter} and \text{picturename} defined by:

\[
\text{painter}(\text{raphael}\text{*marriageofvirgin}) = \text{raphael}, \\
\text{picturename}(\text{raphael}\text{*marriageofvirgin}) = \text{marriageofvirgin}, \\
\vdots
\]

Note that we could have used \text{a1}, \text{a2}, \text{a3} and \text{a4} as an alternative notation for the four altarpieces shown in \text{Altarpieces} (or any another similar encoding), together with the corresponding attribute functions, because aggregations are unique up to isomorphism.

Example 2. Consider the following binary relation:

\[
\text{hasFigure} = \\
\{(\text{raphael}\text{*marriageofvirgin}\text{*joseph}), \text{(corregio}\text{*foursaints}\text{*peter}), \text{(roselli}\text{*madonnaandsaints}\text{*catherine}), \text{(dabrescia}\text{*madonnaandchild}\text{*catherine})\}
\]

An aggregation of \( \text{hasFigure} \) is the set

\[
\text{FiguresinAltarpieces} = \\
\{\text{raphael}\text{*marriageofvirgin}\text{*joseph}, \text{corregio}\text{*foursaints}\text{*peter}, \text{roselli}\text{*madonnaandsaints}\text{*catherine}, \text{dabrescia}\text{*madonnaandchild}\text{*catherine}\}
\]

together with the attribute functions \text{altarpiece} and \text{figure} defined by:

\[
\text{altarpiece}(\text{raphael}\text{*marriageofvirgin}\text{*joseph}) = \text{raphael}\text{*marriageofvirgin}, \\
\text{figure}(\text{raphael}\text{*marriageofvirgin}\text{*joseph}) = \text{joseph}, \\
\vdots
\]
Example 3. In order to represent polyptych altarpieces such as “The Birth of the Virgin” by Lorenzetti, we divide the set Altarpieces of altarpieces in two disjoint subsets, the set OAltarpieces of one-field altarpieces and the set MAltarpieces of polyptych (many-field) altarpieces that can have any number of panels (or fields). We follow the convention of [10] and we enumerate the fields clockwise from left to right, top to bottom. For example, the altarpiece by Lorenzetti has three fields: field1, field2 and field3. We associate fields with many-field altarpieces by means of a relation hasField ⊆ MAltarpieces × Fields which in the example would be:

\[
\text{hasField} = \\
\{ (\text{lorenzetti} \times \text{birthofvirgin}, \text{field1}), \\
(\text{lorenzetti} \times \text{birthofvirgin}, \text{field2}), \\
(\text{lorenzetti} \times \text{birthofvirgin}, \text{field3}) \}
\]

The set FieldsinAltarpieces defined by

\[
\{ (\text{lorenzetti} \times \text{birthofvirgin}, \text{field1}), \\
(\text{lorenzetti} \times \text{birthofvirgin}, \text{field2}), \\
(\text{lorenzetti} \times \text{birthofvirgin}, \text{field3}) \}
\]

is an aggregation of hasField together with the attributes maltarpiece and field defined in the obvious manner.

The following proposition says that an aggregation of a relation is as expressive as the relation: it stores the same information, which can be retrieved by the attribute functions.

**Proposition 1.** Let ρ be a binary relation. Every aggregation ∆₁ρ of ρ is isomorphic to ρ in the sense that there exists a unique function Ψ:∆₁ρ→∆₁×∆₂ such that πᵢ∘Ψ = σᵢ (i = 1, 2), where each πᵢ is the i-th-projection of the Cartesian product ∆₁×∆₂.

It is trivial to prove that Ψ is a bijection. Its inverse defines the encoding of the relation, i.e. it assigns to each pair in the relation ρ a unique element (aggregate) of the set ∆₁ρ.

Informally, the reification of a relation ρ in OWL is a class Cρ representing the tuples of ρ [4, 15]. In order to be able to analyze the relationship between reification and aggregation, it is useful to provide a concrete definition of how we use the notion of reification:

**Definition 2.** Let ∆₁, ∆₂ ⊆ Δ and ρ ⊆ ∆₁ × ∆₂ be a binary relation. A reification of ρ in OWL is a concept Cρ together with two roles S₁ and S₂ (called attribute roles), a role R, two concepts D₁ and D₂, and the following collection 𝒯₀ of axioms:

| (func) | ⊤ ⊑ ≤ 1S₁ ∩ ≤ 1S₂ |
| (domain) | ∃S₁, ⊤ ⊑ ∃S₂, ⊤ ⊑ Cρ |
| (range) | ⊤ ⊑ ∀S₁, D₁ ∩ ∀S₂, D₂ |
| (totality) | Cρ ⊑ ∃S₁, D₁ ∩ ∃S₂, D₂ |
| (contains) | S₂ o (S₁)⁻¹ ⊑ R |
| (unique rep) | Cρ hasKey(S₁, S₂) |

Please note that the hasKey constructor of OWL-2 [16] is used in the last axiom to ensure that any two named individuals r and r’ in Cρ are equal if they satisfy S₁(r, s₁), S₂(r, s₂), S₁(r’, s₁) and S₂(r’, s₂). The importance of this axiom is discussed after the following example.

**Example 4.** Consider the relation hasPainted introduced in Example 1. Since we identify altarpieces precisely through the name of the painter and the designation of the picture, the class Altarpieces correspond in a natural way to the reification of hasPainted. In order to declare the concept Altarpieces to be the reification of hasPainted in OWL, we need a role hasPainted, two attributes roles painter and picturename, the attribute role inversepainter inverse of painter, two concepts Painters and PictureNames, and the axioms of Definition 2. The axioms are written in a user’s friendly syntax (OWL Manchester Syntax) below. Note that we swapped the roles in the composition.

ObjectProperty: painter
Characteristics: Functional
Domain: Altarpieces
Range: Painters
InverseOf: inversepainter

ObjectProperty: picturename
Characteristics: Functional
Domain: Altarpieces
Range: PictureNames

ObjectProperty: hasPainted
Domain: Painters
Range: PictureNames
SubPropertyChain: inversepainter o picturename

Class: Painters
Class: PictureNames
Class: Altarpieces
SubClassOf: painter some Painters and
picturename some PictureNames
HasKey: (painter,picturename)

Figure 5 shows a diagram illustrating all the components involved in the reification of the relation hasPainted. The reification itself is the concept or class Altarpieces and the two attribute roles painter and picturename.
We first discuss the importance of the axiom (contains) which is expressed above as a property chain. Using this axiom, we only have to introduce in OWL that the individual raphael+marriageofvirgin belongs to the class Altarpieces and the values for the projections:

Individual: raphael+marriageofvirgin
Types: Altarpieces
Facts: painter raphael, picturename marriageofvirgin

Then, OWL will be able to infer that raphael painted marriageofvirgin, i.e.

Individual: raphael
Fact: hasPainted marriageofvirgin

This inference is denoted by dotted arrow in Figure 5.

It is worth discussing in more detail the importance of including the last axiom — unique rep. If we do not ensure uniqueness of representation, a modeller (or different people working over the same ontology) can perfectly well introduce two individuals representing the same tuple without the reasoner being able to infer that they are equal. In simple terms, this means that, in the absence of the axiom, a query to retrieve the original table may have repeated rows. This implies, in particular, that queries may return the wrong answers. For instance, if we want to query the number of altarpieces that have no signature, we may count the same tuple more than once.

To illustrate the problem, suppose that we want to express two relationships — paintmedia and height — on the class Altarpieces. As depicted in Figure 6, it is possible that two representations

raphael+marriageofvirgin and raphael+marriageofvirgin

of the same tuple of hasPainted have been accidentally introduced. The individual raphael+marriageofvirgin got connected by the predicate paintmedia to oil — representing the fact that, Raphael’s Marriage of Virgin has been painted in oil — and raphaelmarriageofvirgin by the predicate height to 170 — representing the fact that, the same painting is 170cm high. A query can go awfully wrong if it does not take into account the fact that the tuples can be duplicated. To clarify this point, consider the query “Is there an altarpiece painted in oil and 170cm high?”. The intuitive formulation of the query

\[ \text{Wrong} \{x_1, x_2\} \leftarrow \text{paintmedia}(r, \text{oil}) \cap \text{height}(r, 170) \cap \text{painter}(r, x_1) \cap \text{picturename}(r, x_2) \]

will incorrectly return NO answers because of the duplicated representation. The following alternative formulation would take into account that tuples can be duplicated:

\[ \text{Right} \{x_1, x_2\} \leftarrow \text{paintmedia}(r_1, \text{oil}) \cap \text{height}(r_2, 170) \cap \text{painter}(r_1, x_1) \cap \text{picturename}(r_1, x_2) \cap \text{painter}(r_2, x_1) \cap \text{picturename}(r_2, x_2) \]

And it will correctly return one answer which is \( x_1 = \text{raphael} \) and \( x_2 = \text{sistinemadonna} \).

However, this formulation is no longer very intuitive, i.e. it needs to anticipate the existence of multiple representations, which is a problem of the representation that does not arise from the domain of discourse. In summary, the inclusion of unique rep is essential to ensure that we have a faithful representation of the domain.

Example 5. Consider the relation hasFigure defined in Example 2. The corresponding representation in OWL is achieved through the reification of the relation hasFigure. We introduce the concept FiguresinAltarpieces to represent this reification and add all the corresponding axioms (see Definition 2). These axioms are written in OWL Manchester syntax as follows:

ObjectProperty: altarpiece
Characteristics: Functional
Domain: FiguresinAltarpieces
Range: Altarpieces
InverseOf: inversealtarpieces
Figure 7: FiguresinAltarpieces: the reification of hasFigure

ObjectProperty: figure
  Characteristics: Functional
  Domain: FiguresinAltarpieces
  Range: Figures

ObjectProperty: hasFigure
  Domain: Altarpieces
  Range: Figures
  SubPropertyChain: inversealtarpieces o figure

Class: Figures
Class: FiguresinAltarpieces
  SubClassOf: altarpieces some Altarpieces and figure some Figures
  HasKey: (altarpiece, figure)

Figure 7 shows a diagram illustrating all the components involved in the reification of hasFigure.

We follow the convention mentioned in Example 2 and the tuple (raphael*marriageofvirgin, joseph) is represented by the individual raphael*marriageofvirgin*joseph

We also have to associate values with the attribute roles altarpiece and figure:

Individual: raphael*marriageofvirgin*joseph
  Types: FiguresinAltarpieces
  Facts: altarpiece raphael*marriageofvirgin, figure joseph

Example 6. Consider the relation hasField introduced in Example 3. For the reification of the relation hasField, we introduce the concept FieldsinAltarpieces and add all the corresponding axioms (see Definition 2). These axioms are written in OWL Manchester syntax as follows:

ObjectProperty: maltarpiece
  Characteristics: Functional
  Domain: FieldsinAltarpieces
  Range: MFieldAltarpieces
  InverseOf: inversemaltarpieces

ObjectProperty: field

Characteristics: Functional
  Domain: FieldsinAltarpieces
  Range: Fields

ObjectProperty: hasField
  Domain: MFieldAltarpieces
  Range: Fields
  SubPropertyChain: inversemaltarpieces o field

Class: Fields
Class: FieldsinAltarpieces
  SubClassOf: maltarpiece some MFieldAltarpieces and field some Fields
  HasKey: (maltarpiece, field)

Figure 8 shows a diagram illustrating all the components involved in the reification of hasField.

We follow the same convention mentioned before and the tuple (lorenzetti * birthofvirgin, field2) is represented by the individual lorenzetti * birthofvirgin * field2. We also have to associate values with the attribute roles maltarpiece and field:

Individual: lorenzetti*birthofvirgin*field2
  Types: FieldsinAltarpieces
  Facts: maltarpiece lorenzetti*birthofvirgin, field field2

We can now define more precisely how a reification relates to the relation. Throughout the remainder of the paper, we use capital letters C, D... for metavariables ranging over concepts or classes and we use R, S,... for roles or properties.

Definition 3. Let \( \rho \subseteq D_1 \times D_2 \) be a binary relation. Given an interpretation \( I \), we say that the reification \( \langle C_\rho, R, D_1, D_2, S_1, S_2 \rangle \) is faithful to \( \rho \) in relation to \( I \) iff \( R^I = \rho, D_1^I = D_1, D_2^I = D_2 \), and \( \langle C_\rho, S_1, S_2 \rangle \) is an aggregation of \( \rho \).

Unfortunately, the axioms that are part of the reification (Definition 2) are not sufficient to guarantee that the reification is faithful to \( \rho \) in relation to every interpretation:
The first three axioms state that the role names $S_1$ and $S_2$ are total functions from $C_ρ$ to $D_1$ and $D_2$, respectively. However, a limitation of OWL is that the reasoner does not show any inconsistency if we forget to define $S_1$ or $S_2$ for some element of $C_ρ$ (see [17]).

The converse of $R$-contains, which would correspond to the second condition of Definition 1, is as follows:

$$(R\text{-inclusion}) \quad R \subseteq S_2 \circ (S_1)^{-1}$$

However, this axiom cannot be expressed in OWL in that way because the right-hand side of the inclusion is not a role name (see [3]).

The axiom unique rep is weaker than the third condition of Definition 1 in the sense that the unicity of the representation is not enforced for all individuals but only on those that are explicitly named in the ontology. This is because the hasKey constructor of OWL-2 is a weak form of key representation (so-called “EasyKey constraints”) that is valid only for individuals belonging to the Herbrand Universe [16].

Summarising, reification is not only hard work (in the sense that it requires the modeller to introduce a number of roles and axioms that are ‘technical’, i.e. more related to the limitations of the formalism and less specific to the domain of application) but also prone to errors. Essentially, errors may arise if the modeller forgets to enforce the properties that cannot be expressed in OWL.

### 3.2. Aggregation and Reification of $n$-ary Relations

Definition 1 can be generalised to relations of arbitrary arity and to relations with ‘key attributes’ as originally introduced in [7]:

**Definition 4.** Let $Δ_1, \ldots, Δ_n \subseteq Δ$ and $ρ \subseteq Δ_1 \times \ldots \times Δ_n$ be a relation. Let $i_1, \ldots, i_k \in \{1, \ldots, n\}$. An aggregation of $ρ$ with keys $i_1, \ldots, i_k$ over a universe $Δ$ is a set $Δ_ρ \subseteq Δ$ together with $n$ (total) functions $σ_1, \ldots, σ_n$ from $Δ_ρ$ to $Δ_1, \ldots, Δ_n$, respectively, such that:

1. For all $r \in Δ_ρ$, we have that $⟨σ_1(r), \ldots, σ_n(r)⟩ \in ρ$.
2. For all $⟨x_1, \ldots, x_n⟩ \in ρ$, there exists $r \in Δ_ρ$ such that $σ_1(r) = x_1, \ldots, σ_n(r) = x_n$.
3. For all $r_1, r_2 \in Δ_ρ$, if $σ_1(r_1) = σ_1(r_2), \ldots, σ_k(r_1) = σ_k(r_2)$ then $r_1 = r_2$ — i.e., every tuple of the relation is uniquely identified by its key attributes.

The functions $σ_1, \ldots, σ_n$ are called **key attributes functions** and the remaining ones are called **non-key attribute functions**.

Proposition 1 has a trivial generalisation to the $n$-ary case:

**Proposition 2.** Let $ρ \subseteq Δ_1 \times \ldots \times Δ_n$ be a relation and $i_1, \ldots, i_k \in \{1, \ldots, n\}$. Then, every aggregation $Δ_ρ$ of $ρ$ with keys $\{i_1, \ldots, i_k\}$ is isomorphic to $ρ$ in the sense that there exists a unique function $Ψ: Δ_ρ \to Δ_1 \times \ldots \times Δ_n$ such that $π_i \circ Ψ = σ_i (i = 1, \ldots, n)$.

The existence of an aggregation with given keys cannot always be guaranteed:

**Proposition 3.** An aggregation exists for a relation $ρ \subseteq Δ_1 \times \ldots \times Δ_n$ with keys $\{i_1, \ldots, i_k\}$ iff

- $ρ$ is a partial function from $Δ_{i_1} \times \ldots \times Δ_{i_k}$ into $Δ_{j_1} \times \ldots \times Δ_{j_{n-k}}$, where the set $\{j_1, \ldots, j_{n-k}\}$ is the complement of $\{i_1, \ldots, i_k\}$;
- there is an embedding (injective function) of the domain of $ρ$ in $Δ$, i.e. the universe is large enough to represent the relation.

Note that, in the conditions of this proposition, if the universe $Δ$ contains the Cartesian product $Δ_1 \times \ldots \times Δ_n$, then $ρ$ is an aggregation of itself where the attribute functions are the Cartesian projections $π_i$. The reason we define the concept of aggregation is that the Cartesian product is not a construct of OWL (or, indeed, DL) and, therefore, one needs to resort to mechanisms like reification to encode relations.

**Example 7.** We consider the example of holds given in Section 2. An aggregation of holds is the set

$$Δ_{\text{holds}} = \{h1, h2, h3, h4, h5, h6, h7, h8\}$$

together with the attribute functions $\text{altarpiece}$, $\text{figure}$ and $\text{object}$ defined by:

- $\text{altarpiece}(h1) = \text{raphael} + \text{marriageofvirgin}$,
- $\text{figure}(h1) = \text{joseph}$,
- $\text{object}(h1) = \text{floweringstaff}$

**Example 8.** As an example of a relation where only a subset of the attributes are key is $\text{isLocated}$ the key attributes being $\text{altarpiece}$ and $\text{figure}$. Consider the following definition of $\text{isLocated}$:

$$\text{isLocated} = \{(\text{raphael} + \text{marriageofvirgin}. \text{joseph}. \text{right}), \text{corregio} + \text{foursaints}. \text{peter}. \text{left}), \text{roselli} + \text{madonnaand saints}. \text{catherine}. \text{left}), \text{dabrescia} + \text{madonnaandchild}. \text{catherine}. \text{right})\}$$

Note that the third component is functional on the first two ones. The set $\text{FiguresinAltarpieces}$ defined in Example 2 is an aggregation of $\text{isLocated}$. The non-key attribute location is defined in the obvious manner.

We now generalize Definition 2 to relations of arbitrary arity. The main difference is that, for non-binary relations, we cannot use an atomic role $R$ for representing the relation. For this reason, the axiom $R$-contains has to be dropped.
Definition 5. Let $\Delta_1, \ldots, \Delta_n \subseteq \Delta$ and $\rho \subseteq \Delta_1 \times \cdots \times \Delta_n$ be a relation. Let $i_1, \ldots, i_k \in \{1, \ldots, n\}$. A reification of $\rho$ in OWL with keys $\{i_1, \ldots, i_k\}$ is a concept $C_\rho$ together with roles $S_1, \ldots, S_n$ (called attribute roles), domains $D_1, \ldots, D_n$, and the following collection $T_\rho$ of axioms:

- **(func)** \[ \top \sqsubseteq \lambda S_1 \cap \cdots \cap \lambda S_n \sqsubseteq 1 \]
- **(domain)** \[ \exists S_1. \top \sqcap \cdots \sqcap \exists S_n. \top \sqsubseteq C_\rho \]
- **(range)** \[ \top \sqsubseteq \forall S_1. D_1 \sqcap \cdots \sqcap \forall S_n. D_n \]
- **(totality)** \[ C_R \sqsubseteq \exists S_1. D_1 \sqcap \cdots \sqcap \exists S_n. D_n \]
- **(unique rep)** \[ C_\rho \text{ hasKey}(S_{i_1}, \ldots, S_{i_k}) \]

We call $S_{i_1}, \ldots, S_{i_k}$ the key attribute roles of the reification and the rest are the non-key attribute roles.

Example 9. For the reification of holds, we introduce the concept Reifholds, three roles altarpiece, figure and object and the axioms of Definition 5 (see also Figure 9). These axioms are written in OWL Manchester Syntax as follows:

- **ObjectProperty**: altarpiece
  - Characteristics: Functional
  - Domain: Reifholds
  - Range: Altarpieces

- **ObjectProperty**: figure
  - Characteristics: Functional
  - Domain: Reifholds
  - Range: Figures

- **ObjectProperty**: object
  - Characteristics: Functional
  - Domain: Reifholds
  - Range: Objects

- **Class**: Objects
- **Class**: Reifholds
  - **SubClassOf**: altarpiece some Altarpieces and figure some Figures and object some Objects
  - **HasKey**: (altarpiece, figure, object)

Example 10. For the reification of the relation isLocated from Example 8, we introduce the concept ReifisLocated, the key attribute roles altarpiece, figure, and the non-key attribute role location (see also Figure 10). The axioms of Definition 5 are written in OWL Manchester Syntax as follows:

- **ObjectProperty**: altarpiece
  - Characteristics: Functional
  - Domain: ReifisLocated
  - Range: Altarpieces

- **ObjectProperty**: figure
  - Characteristics: Functional
  - Domain: ReifisLocated
  - Range: Figures

- **DataProperty**: location
  - Characteristics: Functional
  - Domain: ReifisLocated
  - Range: Locations

- **Class**: Locations
- **Class**: ReifisLocated
  - **SubClassOf**: altarpiece some Altarpieces and figure some Figures and location some Locations
  - **HasKey**: (altarpiece,figure)

Notice that, in this example, we used the same role names altarpiece and figure as in Examples 5 and 9. Strictly speaking, this is an abuse of notation and we should use different role names if the concepts $C_{\text{hasFigure}}$, $C_{\text{holds}}$ and $C_{\text{isLocated}}$ are all different. We will discuss this example again in the next section.

4. Participation Dependency

In this section, we put forward a methodological approach aimed at guiding the modeller in the use of reification based on the concepts formalised in the previous section. The method is based on the usage of the semantic primitive of aggregation as used in conceptual modelling precisely for representing situations that, normally, would require non-binary relations or complex integrity constraints [11].

The notion of aggregation allows us to reduce the arity of a relation. This reduction can be performed without
We say that binary relation and $\rho$ (called *participation*) participates in another relation $\rho'$ the key concept of ‘participation’. Intuitively, a relation $\rho$ has the property that allows us to reduce the arity of a relation? The answer that we provide to this question is based on the notion of inclusion dependency, which is typical in databases [9], is too weak to ensure that arity reduction preserves information. Because of this, we introduce a new notion of dependency called *participation*. We illustrate the approach with some examples that are representative of the situations that we have encountered in the altarpieces project.

### 4.1. Relationships amongst Relationships

A recurrent situation in database modelling is the use of aggregation in order to reduce certain ternary relationships to binary ones [11]. Using ER diagrams, the method can be explained in terms of evolving situations such as the one depicted in Figure 1 to the one depicted in Figure 3. More specifically, the method consists in identifying a binary relationship — *hasFigure* in the case at hand — such that the ternary relationship — *holds* — can be expressed as a binary relationship between the aggregation of the former and the remaining domain — Objects (see Figure 3).

Following this methodological principle, instead of reifying *holds*, we reify *hasFigure*. Because *hasFigure* is a binary relation, we represent it by a role *hasFigure* and consider the reification of *hasFigure* as in Example 5, which we name *FiguresinAltarpieces*. The relation *holds* is represented as an object property whose domain is *FiguresinAltarpieces* and whose range is *Objects*. This can be expressed in OWL, Manchester syntax as follows:

**ObjectProperty:** *holds*
- **Domain:** *FiguresinAltarpieces*
- **Range:** Objects

The result is depicted through the following diagram:

$$
\text{FiguresinAltarpieces} \xrightarrow{\text{holds}} \text{Objects}
$$

At the level of individuals, we add assertions such as:

**Individual:** raphael\#marriageofvirgin\#joseph
**Facts:** holds floweringstaff

The methodological question is, then: What is the property that allows us to reduce the arity of a relation? The answer that we provide to this question is based on the key concept of ‘participation’. Intuitively, a relation $\rho'$ participates in another relation $\rho$ if the projection of $\rho$ on some of its components is included in $\rho'$.

**Definition 6.** Let $\Delta_1, \Delta_2, \Delta_3 \subseteq \Delta$, $\rho' \subseteq \Delta_1 \times \Delta_2$ be a binary relation and $\rho \subseteq \Delta_1 \times \Delta_2 \times \Delta_3$ a ternary relation. We say that $\rho'$ participates in $\rho$ if the following condition (called *participation constraint*) is satisfied:

- For all $x \in \Delta_1, y \in \Delta_2, z \in \Delta_3$, if $(x, y, z) \in \rho$ then $(x, y) \in \rho'$.

Similarly, we can define when $\rho' \subseteq \Delta_2 \times \Delta_3$ or $\subseteq \Delta_1 \times \Delta_3$ participates in $\rho$. The relation $\rho'$ is called the *participating relation*. We also say that there is a *participation dependency* between $\rho'$ and $\rho$ when $\rho'$ participates in $\rho$.

Notice that our notion of participation differs from the one used in the ER model [11], which refers to the participation of an entity set in a relation, not of a relation in another relation. Furthermore, in the ER model, the participation constraint usually refers to the ‘total participation’ of an entity $\Delta_1$ in a relation $\rho$, i.e. for all $x \in \Delta_1$ there exist $y, z$ such that $(x, y, z) \in \rho$.

**Example 11.** The relation *hasFigure* participates in the relation *holds* because the participation constraint holds:

$$
\text{if } (x, y, z) \in \text{holds} \text{ then } (x, y) \in \text{hasFigure} \quad (1)
$$

for all $x, y, z$.

From the point of view of the database relational model, the notion of participation is a restricted form of inclusion dependency between relations [9]. More precisely, if $\rho'$ participates in $\rho$ then there is an inclusion dependency between $\rho'$ and $\rho$. However, the converse is not true: not all inclusion dependencies are participation dependencies. The stronger concept of participation dependency is needed to ensure that the arity of a relation can be reduced without losing information as we show in the next proposition.

**Proposition 4.** Let $\rho'$ participate in $\rho$ and let $\Delta_{\rho'}$ be an aggregation of $\rho'$ with attributes $\sigma_1$ and $\sigma_2$. Then, the ternary relation $\rho$ is isomorphic to a binary relation between the aggregation $\Delta_{\rho'}$ and $\Delta_3$, which we call the reduction of $\rho$ by $\Delta_{\rho'}$.

**Proof.** The reduction of $\rho$ by $\Delta_{\rho'}$ is the relation $\tilde{\rho} \subseteq \Delta_{\rho'} \times \Delta_3$ defined by:

$$
\tilde{\rho} = \{(r, z) \mid (\sigma_1(r), \sigma_2(r), z) \in \rho\}
$$

It follows from the fact that $\rho'$ participates in $\rho$ and Proposition 1 that $\rho$ and $\tilde{\rho}$ are isomorphic.

As discussed in Section 3, aggregation can be (partially) implemented in OWL through the mechanism of reification. Taking this forward to participation, if $\rho'$ participates in $\rho$, we can reify $\rho'$ and represent the reduction $\tilde{\rho}$ as a role $R$ whose domain is $C_{\rho'}$.

$$
C_{\rho'} \xrightarrow{R} \Delta_3
$$

For example, in order to represent the relation *holds* in OWL, we represent its reduction *holds* using the reification of *hasFigure*.

In summary, the method that we propose for guiding reification consists in analysing which relations participate...
in other relations: if \( \rho' \) participates in \( \rho \) then, instead of reifying the whole relation \( \rho \), we should consider reifying the participating relation \( \rho' \) and represent \( \rho \) as a role whose domain is \( C_{\rho'} \). If \( \rho' \) participates in yet another relation, say \( \rho'' \), that relation does not need to be reified either and we can reuse instead the reification \( C_{\rho''} \) of \( \rho'' \).

For example, \textit{hasFigure} participates in many relations other than \textit{holds} — e.g. \textit{wears}. All the corresponding relations can be represented in OWL as object properties whose domain is \textit{FiguresinAltarpieces}. That is, \textit{wears} can be represented as a role \textit{wears} whose domain is \textit{FiguresinAltarpieces} and whose range is \textit{Objects}, which can be written in OWL Manchester syntax as follows:

**ObjectProperty:** wears

**Domain:** FiguresinAltarpieces

**Range:** Objects

The corresponding diagrammatic representation is:

\[
\text{FiguresinAltarpieces} \xrightarrow{\text{wears}} \text{Objects}
\]

The advantage of reusing reifications is clear. The axioms for expressing that \textit{FiguresinAltarpieces} is the reification of \textit{hasFigure} (see Example 5) have to be introduced only once, thus avoiding the replication that a blind approach to representation would entail. By choosing to reify the participating relation, we do not only reduce the number of reifications but we also reduce the number of codifications for individuals. This is because we are coding only the components that are shared by several tuples as we showed in Section 2. These components are not only shared in one relation but also amongst several relations. For example, in order to express that Catherine is wearing a crown in “The Madonna and Saints” by Roselli, we can reuse the individual \( \text{roselli} \star \text{madonnaandsaints} \) which was already introduced to express that Catherine is holding the palm and a book.

**Individual:** roselli*madonnaandsaints*catherine

**Facts:** wears crown

Another important aspect of this representation (which is another reason why it is better than the reified ternary relation) is that we now have the relation \textit{holds} represented as a property \textit{holds} and not as a class \textit{Reifholds} as in Example 9. Reifications represent properties but they cannot be used in the syntax as properties because they are actually classes. For instance, we cannot use constructors for roles (e.g. composition, quantification or transitive closure) on \( C_{\text{holds}} \), which may restrict the ability of the modeller to capture important aspects of the domain. Instead, the representation of \textit{holds} as a property allows us to use the role name \textit{holds} in quantifications or in compositions. Below, we will see an example where \textit{holds} is used in a composition.

Yet, one of the most important aspect of our method is that, by reifying the participating relations, we are reflecting and enforcing the participation constraint within the logic. Figure 11 illustrates the case for the relation \textit{holds}. The participation constraint shown in Example 11 can be deduced. In particular, OWL will be able to infer the following assertion:

**Individual:** raphael*marriageofvirgin

**Facts:** hasFigure joseph

using the domain of \textit{holds}, the attribute roles and the axiom (contains) of Example 5.

However, not all relevant participation constraints result in reification and have to be explicitly stated. For example, there is another participation constraint for \textit{holds} that has not been enforced yet:

\[
\text{if}\left( x, y, z \right) \in \text{holds} \text{ then } \left( x, z \right) \in \text{hasObject}
\]

for all \( x, y, z \).

An easy way to enforce this constraint without changing the representation in Figure 11 is to add the following axiom:

\[
\text{holds} \circ \text{altarpiece}^{-1} \subseteq \text{hasObject}
\]

which in OWL Manchester syntax is:

**ObjectProperty:** hasObject

**Domain:** Altarpieces

**Range:** Objects

**SubPropertyChain:** inversealtarpiece o holds

Figure 12 illustrates the case of overlapping constraints by making the two small triangles commute.

There are other ways of representing the relation \textit{holds} and reflect the overlapping constraints. We prefer the solution already presented because it looks more readable and intuitive.

1. We could have reified \textit{hasObject} and represented the relation \textit{holds} as a role \textit{holds} whose domain is \textit{Figures} and range is the reification of \textit{hasObject}. We can see that the pairs

\[
(\text{raphael} \star \text{marriageofvirgin} \star \text{joseph}, \text{book})
\]

and

\[
(\text{joseph}, \text{raphael} \star \text{marriageofvirgin} \star \text{book})
\]
codify the same triplet, i.e.

$$(\text{raphael} \ast \text{marriageofvirgin}, \text{joseph}, \text{book})$$.

There is no mathematical difference between these solutions since both reductions of $holds$ are isomorphic by Proposition 4. The diagram of the OWL representation obtained by reifying $hasObject$ will look symmetric to Figure 12.

2. We could also have reified both participating relations $hasFigure$ and $hasObject$. This solution has some intuition behind it. The individual $joseph$ is representing the figure of Joseph in an abstract way but the individual

$$(\text{raphael} \ast \text{marriageofvirgin} \ast \text{joseph})$$

is representing the particular figure of Joseph on the altarpiece $\text{raphael} \ast \text{marriageofvirgin}$. Hence, we could consider the pair

$$(\text{raphael} \ast \text{marriageofvirgin} \ast \text{joseph}, \\text{raphael} \ast \text{marriageofvirgin} \ast \text{book})$$

to represent the triplet. This solution looks as if it is storing redundant information since the altarpiece is coded twice. However, this solution is also mathematically equivalent to the first one and it is possible to use the axioms of the logic to state that both components of the above pair have the same altarpiece.

Example 12. We show an interesting example that involves polyptych altarpieces. Suppose that we want to express the following fact about a triptych altarpiece:

St Anne is a figure that appears in the central panel of the altarpiece called “The Birth of the Virgin” by Lorenzetti

The natural representation of this domain property is in terms of a relation $hasFigure$ of arity 3:

$$(\text{lorenzetti} \ast \text{birthofvirgin}, \text{field2}, \text{anne}) \in hasFigure$$

Figure 12: Capturing the overlapping constraints: $hasFigure$ and $hasObject$ participate in $holds$

Figure 13: ER diagram: $hasFigure$ for Many Field Altarpieces reduced to a binary relation through an aggregation.

The central panel is identified by $field2$. This case is interesting because while $hasFigure$ has arity 2 for one field altarpieces, it has double arity (arity 2 and arity 3) for many field altarpieces! Formally, this is possible in Set Theory using unions:

$$hasFigure \subseteq Altarpieces \times Figures \cup MAltarpieces \times Fields \times Figures$$

where

$$Altarpieces = OAltarpieces \cup MAltarpieces$$

In spite of the fact that this relation can have double arity and different domains, we will represent this relation in OWL using only one role name $hasFigure$. This will be possible through the use of the union construct and by choosing an appropriate participation relation for reifying. We have two participation dependencies:

1. $hasField$ participates in $hasFigure$, i.e.

   if $(x, y, z) \in hasFigure$ then $(x, y) \in hasField$

   for all $x \in MAltarpieces$, $y \in Fields$ and $z \in Figures$.

2. $hasFigure$ participates in itself, i.e.

   if $(x, y, z) \in hasFigure$ then $(x, z) \in hasFigure$

   for all $x \in MAltarpieces$, $y \in Fields$ and $z \in Figures$.

The natural and simplest choice for reifying is the first participation relation. Figure 13 shows the ER diagram obtained by aggregating the first participation relation. Following the same pattern as in the example of $holds$, we represent the ternary subrelation of $hasFigure$ as an object property whose domain is $FieldsinAltarpieces$ and range is $Figures$ (see Figure 14.)
ObjectProperty: hasFigure
  Domain: FieldsinAltarpieces
  Range: Figures

Since adding domains in OWL is equivalent to taking the union of them, the domain of hasFigure is the union of Altarpieces and FieldsinAltarpieces (see Example 5).

The first participation constraint of hasField is enforced because the domain of hasFigure is the reification of hasField. To enforce the second participation constraint, we follow the same pattern as we did for the overlapping constraints of holds and we add the following axiom:

\[ \text{hasFigure} \circ \text{maltarpiece}^{-1} \sqsubseteq \text{hasFigure} \]

which in OWL Manchester syntax is:

ObjectProperty: hasFigure
  SubPropertyChain: inverse\text{maltarpiece} \circ \text{hasFigure}

4.2. Descriptive Attributes

Another related methodological guideline for the use of reification arises from what in [11] are called descriptive attributes. Descriptive attributes are used to record information about a relationship rather than about one of the participating entities, again using an aggregation. From a conceptual modelling point of view, they allow us to capture typical situations in which a functional dependency exists in a ternary relation as an attribute of the aggregation of a binary relation. For example, it would be intuitive to represent location in Figure 15 as a descriptive attribute associated with the relationship hasFigure.

Definition 7. Let \( \rho \subseteq \Delta_1 \times \Delta_2 \) and \( \rho' \subseteq \Delta_1 \times \Delta_2 \times \Delta_3 \). We say that \( \rho' \) is a descriptive attribute of \( \rho \) if the following conditions hold:

1. \( \rho' \) is a function from \( \Delta_1 \times \Delta_2 \) to \( \Delta_3 \).
2. \( \rho \) participates in \( \rho' \).

It is evident from this definition that the notion of descriptive attribute is a particular case of the notion of participation as introduced in Definition 6. Given that descriptive attributes involve a participating relation, the methodological guidelines that we discussed in 4.1 suggest that descriptive attributes be represented as (functional) roles of the reification of the participating relation.

For example, location is a descriptive attribute of hasFigure because the following properties hold in the domain:

1. There is a functional dependency between the location and pair given by the altarpiece and the figure. In other words, the ternary relation isLocated is actually a function

   \[ \text{isLocated} \in \text{Altarpieces} \times \text{Figures} \rightarrow \text{Locations} \]

2. There exists a participation dependency between the relations hasFigure and isLocated. In other words, hasFigure participates in isLocated, i.e. for all \( x, y \) and \( z \), we have that:

   if \( (x, y, z) \in \text{isLocated} \) then \( (x, y) \in \text{hasFigure} \).

We can represent the descriptive attribute location in OWL as follows:

1. We reuse the class FiguresinAltarpieces as the reification of hasFigure (see Example 5).
2. We define a role isLocated representing the descriptive attribute as a function whose domain is FiguresinAltarpieces.

This is written in OWL Manchester syntax as follows:

DateProperty: isLocated
  Characteristics: Functional
  Domain: FiguresinAltarpieces
  Range: Locations

The OWL-representation of the descriptive attribute isLocated can be depicted as follows:

![Diagram](image-url)
Notice that descriptive attributes are a particular case of non-key attributes (see Definition 4) where we have the additional involvement of a participating relation. At first sight it might look as if reifying isLocated as a relation of arity 3 whose key attributes are the first and second components (Example 10) would be the same as first reifying hasFigure and later adding isLocated as a descriptive attribute. However, there are some important differences between these two processes. Example 10 does not take into account that hasFigure is a participating relation. From the point of view of the axioms, this is reflected in the fact that the axiom (contains) was not present in Example 10, whilst this axiom is an important component of the reification of hasFigure.

There is also a methodological difference between the two processes. On the one hand, we want to leave the system open to the addition of any number of descriptive attributes (such as size). On the other hand, we also want to add the right amount of axioms and have a systematic way for doing so. In order to achieve this, it should be clear which is the participating relation and how the relations interact with each other. That is, descriptive attributes play an important methodological role in the representation of the domain knowledge.

4.3. Relations of Arbitrary Arity

In this section, we generalise the notion of participation to relations of arbitrary arity and show how the notion of participation can be used to reduce any n-ary relation to a relation of a smaller arity. We also show an example of a relation of arity 4 with several participating relations and how the choice of a participating relation can affect our representation of the relation.

**Definition 8.** Let \( i_1, \ldots, i_k \in \{1, \ldots, n\} \) be all different, \( \rho \subseteq \Delta_1 \times \cdots \times \Delta_n \) and \( \rho' \subseteq \Delta_{i_1} \times \cdots \times \Delta_{i_k} \). We say that \( \rho' \) participates in \( \rho \) iff the following constraint (also called the participation constraint) is satisfied for all \( x_1 \in \Delta_{1}, \ldots, x_n \in \Delta_{n}: \)

\[
(x_1, \ldots, x_n) \in \rho \text{ implies } (x_1, \ldots, x_k) \in \rho'.
\]

The relation \( \rho' \) is called the participating relation. We also say that there is a participation dependency between \( \rho' \) and \( \rho \) when \( \rho' \) participates in \( \rho \).

The notion of participation allows us to reduce the arity of a relation:

**Proposition 5.** If \( \rho' \) participates in \( \rho \) then \( \rho \) is isomorphic to a relation of arity \( n - k + 1 \) whose domains are the aggregation \( \Delta_{\rho'} \) and the remaining sets \( \Delta_{j_1}, \ldots, \Delta_{j_{n-k+1}} \) where \( \{j_1, \ldots, j_{n-k+1}\} \) is \( \{1, \ldots, n\} - \{i_1, \ldots, i_k\} \).

**Proof.** For simplicity, and in order to use examples from our case study, we show how the corresponding reduction is performed on a relation \( \rho \subseteq \Delta_1 \times \Delta_2 \times \Delta_3 \times \Delta_4 \) of arity 4. Suppose that there is a ternary relation \( \rho' \subseteq \Delta_1 \times \Delta_2 \times \Delta_3 \) participating in \( \rho \), i.e. for all \( x_1 \in \Delta_{1}, \ldots, x_4 \in \Delta_{4} \),

\[
(x_1, x_2, x_3, x_4) \in \rho \text{ implies } (x_1, x_2, x_3) \in \rho'.
\]

Let \( \Delta_{\rho'} \) be an aggregation of \( \rho' \) with attributes \( \sigma_1, \sigma_2 \) and \( \sigma_3 \). We can reduce the relation \( \rho \) to a binary relation \( \hat{\rho} \) between the aggregations \( \Delta_{\rho'} \) and \( \Delta_{\rho} \) as follows:

\[
\hat{\rho} = \{(r, x_4) \mid (\sigma_1(r), \sigma_2(r), \sigma_3(r), x_4) \in \rho\}
\]

As a corollary of Proposition 1, the relations \( \rho \) and \( \hat{\rho} \) are isomorphic.

As an example, suppose that we want to express the following fact about a triptych altarpiece:

*St Anne is lying in bed in the central panel of the altarpiece called “The Birth of the Virgin” by Lorenzetti*

The natural representation of this domain property is in terms of a relation liesOn of arity 4:

\[
(lorenzetti * birthofvirgin, field2, anne, bed) \in \text{liesOn}
\]

The central panel is identified by field2. The altarpiece lorenzetti{birthofvirgin} belongs to the subclass MA{Altarpieces} of Altarpieces (see Examples 3, 6 and 12).

We have three participation dependencies: the three relations hasField, hasFigure and hasObject participate in liesOn. We will choose to reify hasFigure and reduce the relation liesOn to a binary relation between the aggregation of hasFigure and the set Objects. However, it is more advantageous to reuse instead the reduction hasFigure discussed in Example 12. The corresponding ER diagram is shown in Figure 16. The advantage is that, because hasFigure is a binary relation obtained from the aggregation of the binary relation hasField, we do not only enforce the dependency constant of hasFigure but also the one of of hasField. This is illustrated in Figure 16.
17 by the fact that we stack one representation on top of the other. The third dependency illustrated in Figure 18 will be enforced by adding an axiom similar to the one for hasObject and holds as follows:

\[ \text{liesOn} \circ \text{fieldinaltarpiece}^{-1} \sqsubseteq \text{hasObject} \]

The corresponding representation in OWL is achieved through the reification of the relation hasFigure. We introduce the concept FiguresinMAltarpieces to represent this reification and add all the corresponding axioms (see Definition 2). These axioms are written in OWL Manchester syntax as follows:

**ObjectProperty: fieldinaltarpiece**
- **Characteristics**: Functional
- **Domain**: FiguresinMAltarpieces
- **Range**: FieldsinAltarpieces
- **InverseOf**: inversefieldinaltarpiece

**ObjectProperty: figure**
- **Characteristics**: Functional
- **Domain**: FiguresinMAltarpieces
- **Range**: Figures

**ObjectProperty: hasFigure**
- **SubPropertyChain**: inversefieldinaltarpiece \( \circ \) figure

**Class: FiguresinMAltarpieces**
- **SubClassOf**: fieldinaltarpiece
  - some FieldinAltarpieces and
  - figure some Figures
- **HasKey**: (fieldinaltarpieces, figure)

Finally, the relation liesOn is represented in OWL as a role liesOn whose domain is FiguresinMAltarpieces and whose range is Objects. This is written in OWL Manchester syntax as follows (see also Figure 17):

**ObjectProperty: liesOn**

**5. The Generalization Construct**

In this section, we extend our methodological approach with a construction that is inspired by the mechanism of generalization introduced in [8] for database design. For simplicity, we consider only binary relations and define generalization over their ranges. Extending the definition to include domains (not just ranges) and to relations of arbitrary arity is straightforward.

In order to motivate this construct, consider again the relation hasFigure introduced in Section 4.1 whose range is Figures. Although figures are, understandably, of major interest in altarpieces, there are a number of other elements that play an important role and need to be represented in the ontology — the flowering shaft, tiaras, architectural elements, and so on. For example, we have...
introduced in Section 4.3 the domain Objects, which is used as the range of \textit{holds}. We can then consider a relation hasObject similar to hasFigure but with Objects as range to represent the fact that given objects are depicted in given altarpieces.

However, in certain circumstances, it is more convenient to establish relationships over a more general class \textit{Elements} that consists of Figures and Objects. For instance, we are interested in representing the fact that \textit{The Papal tiara rests on top of the balustrade in the altarpiece Sistine Madonna by Raphael}.

For this purpose, it is convenient to define the relation
\[ \text{restson} \subseteq \text{Altarpieces} \times \text{Elements} \times \text{Elements} \]
According to the method that we have been proposing, we should seek to establish a participating relation hasElement \( \subseteq \text{Altarpieces} \times \text{Elements} \) that we would reify. However, given that we already have hasFigure and hasObject, we should not introduce hasElement as an independent relation. Intuitively, hasElement is a ‘generalization’ of hasFigure and hasObject and, indeed, we would expect the same to hold between their reifications.

In [8], a generalization abstraction is introduced as “an adaptation of Hoare’s discriminated union structure [13]”. This kind of structure, which supports programming language constructs such as Pascal’s \textit{variants}, can be explained in terms of the notion of disjoint union. In Set Theory, the disjoint union of two sets \( \Delta_1 \) and \( \Delta_2 \) is a triple \( \langle \Delta_1 \cup \Delta_2, i_1, i_2 \rangle \) where each \( i_j \) is an injective function \( i_j : \Delta_j \rightarrow \Delta_1 \cup \Delta_2 \) such that, for any other triple \( \langle \Delta', i_1', i_2' \rangle \) with \( i_j' : \Delta_j \rightarrow \Delta' \), there is a unique function \( \phi : \Delta_1 \cup \Delta_2 \rightarrow \Delta' \) such that \( \phi \circ i_j = i_j' \) (\( i = 1, 2 \)). That is, \( \Delta_1 \cup \Delta_2 \) is the smallest set that ‘contains’ both \( \Delta_1 \) and \( \Delta_2 \) while distinguishing between the elements that they have in common. The functions \( i_j \) provide the ‘tag fields’ that, in discriminated unions in the sense of [13], indicate which of the particular constituent sets, \( \Delta_1 \) or \( \Delta_2 \), each element of \( \Delta_1 \cup \Delta_2 \) originates from.

Naturally, if \( \Delta_1 \) and \( \Delta_2 \) are disjoint, their union, together with the corresponding inclusions, is also a disjoint union. In [8], the concept of generalization applies precisely to disjoint classes to define a superclass. However, for generality, we work with the original definition, which also has the advantage of providing a mathematical structure closer to that of aggregation given in Definition 1.

\textbf{Definition 9.} Let \( p_1 \subseteq \Gamma \times \Delta_1 \) and \( p_2 \subseteq \Gamma \times \Delta_2 \) be binary relations. A generalization of \( p_1 \) and \( p_2 \) is a triple \( \langle p_3, i_1, i_2 \rangle \) where:

\begin{itemize}
  \item \( \langle \Delta_1 \cup \Delta_2, i_1, i_2 \rangle \) is a disjoint union of \( \Delta_1 \) and \( \Delta_2 \)
  \item \( p_3 \subseteq \Gamma \times (\Delta_1 \cup \Delta_2) \) is defined by
    \[ p_3 = \{(z, i_1(x)) \mid (z, x) \in p_1\} \cup \{(z, i_2(y)) \mid (z, y) \in p_2\} \]
\end{itemize}

We normally use the notation \( p_1 \uplus p_2 \) to refer to a generalization of \( p_1 \) and \( p_2 \).

Going back to our example, how can we represent the relation hasElement as a generalization of hasFigure and hasObject? Consider first the problem of representing Elements as a disjoint union of Figures and Objects. In OWL, the disjoint union of concepts is not available as a primitive construct: it is an abbreviation for a union of two classes with an extra axiom requiring that the classes are disjoint. For example, in the case at hand, we would define:

\textbf{Class: Elements}

\textbf{DisjointUnionOf: Figures, Objects}

The system generates an inconsistency whenever the extensions of Figures and Objects are not disjoint. Unfortunately, OWL does not extend this mechanism to roles (in fact, it does not support the union of roles). Therefore, we cannot express that hasElement is the disjoint union of hasFigure and hasObject.

However, our main interest is not so much hasElement, but its aggregation and, ultimately, its reification, as a relation participating in restson. Intuitively, given

\[ \text{hasElement} \subseteq \text{Altarpieces} \times \text{Elements}, \]
\[ \text{hasFigure} \subseteq \text{Altarpieces} \times \text{Figures}, \]
\[ \text{hasObject} \subseteq \text{Altarpieces} \times \text{Objects} \]

where Altarpieces = \( \Delta_{\text{hasPainted}} \), if
\[ \text{hasElement} = \text{hasFigure} \uplus \text{hasObject} \]
then we should also have that
\[ \Delta_{\text{hasElement}} = \Delta_{\text{hasFigure}} \uplus \Delta_{\text{hasObject}} \]

This is what we prove next:

\textbf{Proposition 6.} Let \( p_1 \subseteq \Gamma \times \Delta_1 \) and \( p_2 \subseteq \Gamma \times \Delta_2 \) be binary relations. Let \( \langle \Delta_{p_1}, \gamma_1, \sigma_1 \rangle \) and \( \langle \Delta_{p_2}, \gamma_2, \sigma_2 \rangle \) be aggregations of \( p_1 \) and \( p_2 \), respectively. Let \( \langle p_3, i_1, i_2 \rangle \) be a generalization of \( p_1 \) and \( p_2 \). Finally, let \( \langle \Delta_{p_1} \uplus \Delta_{p_2}, \delta_1, \delta_2 \rangle \) be a disjoint union. Then, \( \langle \Delta_{p_1} \uplus \Delta_{p_2}, \gamma, \sigma \rangle \) define an aggregation of \( p_3 \) where

\begin{itemize}
  \item \( \gamma \) is the unique function \( \Delta_{p_1} \uplus \Delta_{p_2} \rightarrow \Gamma \) s.t. \( \gamma \circ \delta_i = \gamma_i \) (\( i = 1, 2 \))
  \item \( \sigma \) is the unique function \( \Delta_{p_1} \uplus \Delta_{p_2} \rightarrow \Delta_1 \uplus \Delta_2 \) s.t. \( \sigma \circ \delta_i = i_1 \circ \sigma_1 \) (\( i = 1, 2 \))
\end{itemize}

The above conditions are depicted in Figure 19.

\textbf{Proof.} The existence of the functions \( \sigma \) and \( \gamma \) results from the universal properties of \( \langle \Delta_{p_1} \uplus \Delta_{p_2}, \delta_1, \delta_2 \rangle \) as a disjoint union. The three conditions of Definition 1 are also easy to prove.
That is to say, the disjoint union of the aggregations of two relations is an aggregation of the generalization of the relations.

In OWL, taking the concept FiguresinAltarpieces for the reification of hasFigure as defined in Example 5, and similarly ObjectsinAltarpieces for the reification of hasObject, we would introduce a concept

\textbf{ElementsinAltarpieces}

for the reification of hasElement as follows:

\textbf{ObjectProperty: \textit{altarpiece}}
\begin{itemize}
  \item \textbf{Characteristics: Functional}
  \item \textbf{Domain: ElementsinAltarpieces}
  \item \textbf{Range: Altarpieces}
  \item \textbf{InverseOf: inversealtarpieces}
\end{itemize}

\textbf{ObjectProperty: \textit{element}}
\begin{itemize}
  \item \textbf{Characteristics: Functional}
  \item \textbf{Domain: ElementsinAltarpieces}
  \item \textbf{Range: Elements}
\end{itemize}

\textbf{ObjectProperty: hasElement}
\begin{itemize}
  \item \textbf{Domain: Altarpieces}
  \item \textbf{SubPropertyChain: inversealtarpieces o element}
\end{itemize}

\textbf{Class: ElementsinAltarpieces}
\begin{itemize}
  \item \textbf{SubClassOf: altarpieces some Altarpieces and element some Elements}
  \item \textbf{HasKey: (altarpiece, element)}
\end{itemize}

\textbf{Class: ElementswithAltarpieces}
\begin{itemize}
  \item \textbf{DisjointUnionOf: FiguresinAltarpieces, ObjectsinAltarpieces}
\end{itemize}

Notice that, relative to Definition 2, we add that the reification of hasElement is the disjoint union of the reifications of hasFigure and hasObject. A diagram connecting the reifications of the three relations hasElement, hasFigure and hasObject is shown in Figure 20.

Note that the attribute \textit{altarpiece} is shared by the three classes. It is enough to declare that the domain of the attribute \textit{altarpiece} is the biggest class which is the reification of hasElement.⁷ Because the attributes figure, object and element are actually functions, we can deduce in OWL that the non-common attributes are related by \textit{figure, object \sqsubseteq element}

Furthermore, given that the domains and ranges of figure and object are disjoint, OWL can infer that these properties are disjoint too. However, note that, as already mentioned, we cannot express in OWL that the property element is the union of figure and object.

Finally, we investigate how generalization can work together with the notion of participation and discuss the representation of restson.

\textbf{Definition 10.} Let \((\Delta_1 \sqcup \Delta_2, \iota_1, \iota_2)\) be a disjoint union of \(\Delta_1\) and \(\Delta_2\), and \(\rho \subseteq \Gamma \times (\Delta_1 \sqcup \Delta_2) \times \Theta\). We define the restrictions \(\rho \mid_{\iota_1} \subseteq \Gamma \times \Delta_1 \times \Theta\) and \(\rho \mid_{\iota_2} \subseteq \Gamma \times \Delta_2 \times \Theta\) as follows:

\[
\rho \mid_{\iota_1} = \{(x, y_1, z) \mid (x, \iota_1(y_1), z) \in \rho \land y_1 \in \Delta_1\}
\]
\[
\rho \mid_{\iota_2} = \{(x, y_2, z) \mid (x, \iota_2(y_2), z) \in \rho \land y_2 \in \Delta_2\}
\]

That is, a restriction extracts from a relation involving a disjoint union those triples that involve only the elements of one of the sets. Notice that, in the case in which the sets are disjoint, the functions \(\iota_i\) are inclusions, which leads to a simpler formulation of the restrictions. As already explained, this is case that interests us in OWL.

It is easy to see that if a relation participates in another relation \(\rho\), it also participates in any relation that contains \(\rho\). The following proposition is a refinement of this observation for the case of generalizations:

\textbf{Proposition 7.} Let \(\rho_1 \subseteq \Gamma \times \Delta_1\), \(\rho_2 \subseteq \Gamma \times \Delta_2\) and \(\langle \rho_3, \iota_1, \iota_2 \rangle\) be a generalization of \(\rho_1\) and \(\rho_2\). Let \(\rho \subseteq \Gamma \times (\Delta_1 \sqcup \Delta_2) \times \Theta\). The relation \(\rho_3\) participates in \(\rho\) iff \(\rho_1\) participates in \(\rho \mid_{\iota_1}\) and \(\rho_2\) participates in \(\rho \mid_{\iota_2}\).

That is, the participation of a generalization in another relation can be reduced to the participation of the components of the generalization in the corresponding restrictions. For example, hasElement participates in restson iff

⁷Actually, we should also remove it from the axioms in the reification of hasFigure (see Section 4.3) otherwise OWL takes the intersection.
hasFigure and hasObject participate in the corresponding restrictions of restson.

Following the methodology that we have introduced in Section 4, we can represent the relation restson in OWL as a role restson whose range is Elements and whose domain is ElementsinAltarpieces(= C\text{hasElement}).

ElementsinAltarpieces \xrightarrow{\text{restson}} \text{Elements}

ObjectProperty: restson
Domain: ElementsinAltarpieces
Range: Elements

The generalization hasElement is the right conceptualization because most of the relations used for describing altarpieces are between elements. Using this generalization, the ontology can be easily extended to include new properties with the same characteristics in the sense that the representation of those properties does not require any further reification. More precisely, we can re-use the reification \( C\text{hasElement} \) and represent any new relation involving elements of an altarpiece as a role \( R \) whose domain is ElementsinAltarpieces(= C\text{hasElement}) and whose range is Elements.

ElementsinAltarpieces \xrightarrow{R} \text{Elements}

The reification \( C\text{hasElement} \) can be used to represent even more relations, indeed any relation in which hasElement participates. For example, if we take the inscription of a book on a certain altarpiece to be represented by a string, we can define a relation hasInscription in which hasElement participates but whose range is the set of strings.

ElementsinAltarpieces \xrightarrow{\text{hasInscription}} \text{String}

DataProperty: hasInscription
Domain: ElementsinAltarpieces
Range: String

6. Related Work and Concluding Remarks

In this paper, we proposed a methodological approach for Ontology Engineering aimed at guiding the use of reification as a way of representing n-ary relations. Our method simplifies ontologies in the sense that it not only reduces the number of codifications but, more importantly, rationalizes the choice of relations to reify based on dependencies between relations that can be derived from a conceptual analysis of the application domain. This approach promotes reuse through the sharing of reifications of relations that participate in several other relations.

In a nutshell, we advocate that:

- Domain experts should start by building a conceptual model in which they can identify relationships between relations and descriptive attributes.
- Participation dependencies should be identified in those models with a view to reifying the participating relation (as in Section 4.1 and Section 4.2).
- A common domain that generalizes the domains of several participating relations should also be identified. This would induce a common participating relation generalizing all of them that should be reified (as in Section 5).

In order to justify our approach, we provided mathematical definitions of the aggregation and generalization constructs as used for database design [11, 7, 8], which we used to analyse the extent to which they can be implemented in languages such as OWL. A number of results were proved that attest to the soundness of the methodological guidelines that we put forward.

The use of conceptual modelling primitives in the context of ontologies is not new. For instance, [18] and [19] show how to transform ER diagrams into Description Logic. However, this transformation does not include relationships involving relationships or descriptive attributes as illustrated in Section 4, nor does it address aggregation as a modelling abstraction.

A paper that focuses specifically on aggregation is [20]. However, the author represents aggregations using union of classes, which does not correspond in any way to their original meaning [7]. Our use of aggregation (based on cartesian products) adheres to its use in databases and explores its methodological advantages for conceptual modelling [11].

Other proposals can be found in the literature that, like [14], put forward patterns for representing relations \( \rho \subseteq A \times B \times C \). The third case of Pattern 1 in that note reifies the whole relation and, in the remaining cases, reifies \( B \times C \) and represent \( \rho \) as a property whose range is the reification \( C_{\rho \subseteq A \times C} \). Our method is based on semantic abstractions and, therefore, goes beyond simple patterns. In fact, it adds depth and mathematical rigour to the study of these patterns in the sense that it guides the application of reification by the identification of relations that, like hasPainted, participate in other relations.

Extensions of description logics with n-ary relations can also be found in the literature [4, 15, 21]. However, the Web Ontology Language (OWL 2), which is based on the Description Logic of [3], does not provide this capability. OWL 2 provides the possibility of defining n-ary datatype predicates \( F \), albeit in a restricted way [22]. We can use an n-ary predicate \( F \) in expressions of the form \( \forall P_1 \ldots P_n, F \) or \( \exists P_1 \ldots P_n, F \) where \( P_1 \ldots P_n \) are binary data type predicates. The n-ary predicate \( F \) is actually a functional proposition defined implicitly by a formula of the form \( \lambda(x_1 \ldots x_n).\text{comp}(p, q) \) where \( \text{comp} \in \{\leq, =, \neq, >, <\} \) and \( p \) and \( q \) are linear polynomials on \( x_1, \ldots, x_n \). However, OWL does not support the definition of n-ary predicates by listing the tuples as for object and datatype properties.
The feedback received from using the method in the construction of the Ontology of Altarpieces is that it offers a more controlled use of reification and a closer fit between the resulting ontology and the application domain as perceived by an expert. Our plans for future work include developing tools for helping ontologist follow the proposed methodology and assist them in the representation of relations of arity \( n \).

References
