Learning and Nonlinear Models - Journal of the Brazilian Society on Computational Intelligence (SBIC), Vol. 12, Iss. 1, pp. 4-16, 2014
© Brazilian Society on Computational Intelligence

HOW CAN ONTOLOGY DESIGN PATTERNS HELP ONTOLOGY REFINEMENT?

Wander dos Santos Vasconcellos, Kate Revoredo, Fernanda Baião
Department of Applied Informatics, Unirio Av. Pasteur, 458, Rio de Janeiro, RJ, Brazil
{wander.vasconcellos,katerevoredo,fernanda.baio}@uniriotec.br

Abstract – Over time, an Ontology may become obsolete, thus raising the need for it to be refined towards correctly reflecting the domain of discourse. Usually, when this refinement is performed manually or semiautomatically, it is ad hoc, time consuming and error-prone. Therefore, automatic mechanisms are demanding to improve Ontology refinement processes. On the other hand, Ontology Design Patterns are modeling solutions with recognized good quality that solve recurrent ontology modeling problems, thus improving the quality of the Ontology. In this paper, we propose to automatically refine ontologies by applying Theory Revision techniques using Ontology Design Patterns (ODPs) to reduce the search space of possible decision points to be considered, thus ensuring the good quality of an ontology.

Keywords – Ontology, Ontology Refinement, Ontology Design Patterns, Theory Revision.

1 Introduction

Ontologies are important artifacts for knowledge representation, being useful for knowledge based systems. They have their own life cycle: design, implementation, evaluation, production, maintenance, reuse, etc [1]. When designing, implementing and refining an ontology, one can benefit from reusing an existing one. Cost and time reduction for ontology construction and duplicated effort avoidance are among the advantages of ontology reuse. Ontology reuse is based on the realization that many terms that one wants to model is already defined in some ontology, henceforth called *source ontology*. A source ontology is composed by Modules, which are small fragments containing a complete meaning of a set of terms. Reusing a source ontology means reusing some of its modules [2] [3], then a module can be seen as an *ontology design pattern* (ODPs).

Ontology maintenance is an important activity, since over time, an ontology can become obsolete, needing evolution. Usually, when the Ontology evolution is performed manually or semiautomatically, it is ad hoc, time consuming and error-prone, specially on large-scale domains. Automatic mechanisms are thus needed to improve Ontology evolution processes achieving efficiency and speed, both requirements for the success of knowledge based systems.

On the other hand, Theory Refinement [4] is an Inductive Logic Programming (ILP) [5] field focused on modifying or restructuring an existing theory. The modification of an ontology is known as *theory revision* and aims at finding a revised theory that suits a set of examples.

In this paper, we investigate how ODPs can help the process of ontology refinement. An approach for automatically refine an ontology encompassed by ODPs considering theory revision techniques is proposed. In this approach, we rely on the reuse of ODPs to reduce the search space of revision points during the ontology refinement process, while ensuring the quality of the ontology. A real scenario is used to evaluated our proposal.

Besides this introduction, this paper consists of the following sections. In section 2, we review some important concepts to the understanding of our proposal, which is presented in Section 3. In section 4, we show how our proposal should behave, by simulating some expected scenarios. In section 5, related works are presented. And in section 6, we make some considerations and present future work.

2 Background Knowledge

In this section, Theory Revision and Ontology Design Patterns are reviewed.

2.1 Theory Revision

The acquisition of knowledge is a difficult task, time consuming and error prone. The process of improving automatically a knowledge base using learning methods can be achieved through theory revision [4]. Given a knowledge base composed by a set of first-order Horn clauses[1] (theory), which can be incomplete or incorrect, theory revision is a process that minimally modifies this theory aiming to make it consistent with a dataset. This dataset is composed of positive and negative examples, usually represented by ground facts. The former are instances of objects or relationships deemed correct and, therefore, must be confirmed by the theory, while the later are considered incorrect instances and, therefore, must be refuted by the theory.

[1]A first-order Horn clause is a clause (disjunction of literals) with at most one positive literal.

A theory revision system starts from the evaluation of an initial theory through the dataset, where this initial theory can be divided in two parts: background knowledge (henceforth called FDT), which is assumed to be correct, and another part that can be modified by the revision (henceforth called THY). From the evaluation, parts of the THY, responsible for misclassification of examples, are collected. These parts are called *revision points* and can be of two types: *generalization* and *specialization* revision points. The first represents failures in the evaluation of positive examples and the second represents failures in the evaluation of negative examples. In order to correct these failures, *revision operators* are used. Generalization revision points are corrected through generalization revision operators, such as add-clauses and delete-antecedents, while specialization revision points are corrected through specialization revision operators, such as add-antecedents and delete-clauses [4].

An example of theory revision system is FORTE (First-Order Revision Theory from Examples) [6,7]. The top-level algorithm is depicted in Algorithm 1.

```
Require: Initial Theory composed by FDT and THY ; Dataset
Ensure: Final Theory composed by FDT and a revised THY (THY')
 1: repeat
 2:   generate revision points;
 3:   sort revision points by potential (high to low);
 4:   for all revision point do
 5:     if potential of current revision point is less than the score of the best revision to date then
 6:       break;
 7:     end if
 8:     generate revisions;
 9:     update best revision found;
10:   end for
11:   if best revision improves the theory then
12:     implement best revision;
13:   end if
14: until no revision improves the theory
```

Algorithm 1: FORTE's top-level refinement algorithm (adapted from [6]

For each iteration and each example the algorithm annotates the revision points. Each revision point has a potential, which is the number of examples correctly classified which could result from a revision of that point. After finding the revision points, revision operators are used accordingly in an attempt to make the theory consistent. From the application of a revision operator to a revision point a revised theory is found and evaluated receiving a score, which is the actual increase in theory accuracy it achieves. The best revised theory, i.e. the revised theory with the highest score, will be the one implemented. It is a hill-climbing algorithm and it stops when no improvement in theory accuracy can be achieved.

2.2 Ontology Design Patterns

A domain Ontology is an artifact which encodes some knowledge about the domain. It describes the relevant domain entities and their relationships through the definition of concepts and roles respectively. The Ontology shown in Figure 1 represents some concepts and relationships of the Family law domain. It is defined by the Brazilian Civil Code and determines rights and duties within the family and between it and society. For example, it stipulates that marriage is a relationship between a man and a woman. Thus, in Figure 1 the "married"relationship between a father and a mother necessarily indicates a relationship between a man and a woman.

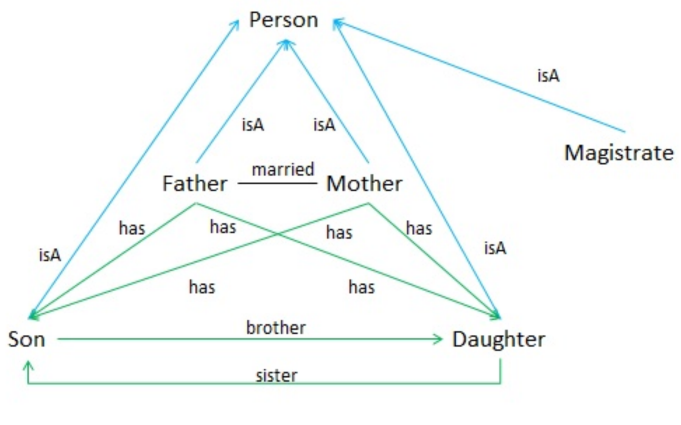

Figura 1: Concepts and Relationships of the Family Law Domain

Moreover, to improve the quality of an Ontology, Ontology Design Patterns (ODPs) [1] may be applied. ODPs are modeling solutions with recognized good quality that solve recurrent ontology modeling problems, such as lack of expressivity of logical formalism used and become a model using any paradigm (UML, ER, etc) in an ontological model. They can be classified into six groups [8]: Structural, Correspondence, Reasoning, Presentation, Lexico-syntactic and Content. Since our proposal is based on Content ODPs, we describe them briefly as follows.

Content ODPs focus on specific content problems involving domain classes and properties belonging to an ontology [1]. For instance, define which objects participate in the same event and which events an object participates. Among the content ODPs, we can mention: Co-participation, Participation and Communities ODPs.

The Co-participation Content ODP represents the participation of two objects in an event. Figure 2 is a diagram representation depicting this ODP. With this pattern it is possible to identify an event in which two objects participates. The event is a specialization of something that has participants and contains information about its participants. Furthermore, each co-participant knows each other.

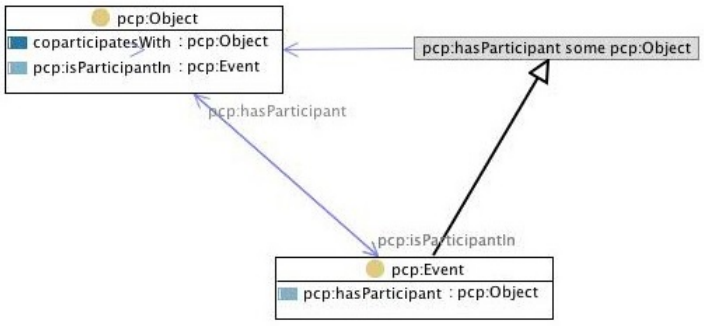
pcp:Object
coparticipatesWith : pcp:Object
pcp:isParticipantIn : pcp:Event
pcp:hasParticipant some pcp:Object
pcp:hasParticipant
pcp:isParticipantIn
pcp:Event
pcp:hasParticipant : pcp:Object

Figura 2: Co-participation Content ODP

The Participation content ODP represents one object that participates in one event. Figure 3 is a diagram representation depicting this ODP. With this pattern it is possible to identify which events an object participates. In this case, the object is a specialization of participant in some event and contains the information of the events that it participates regardless of any additional information. Moreover, the event contains information about its participants.

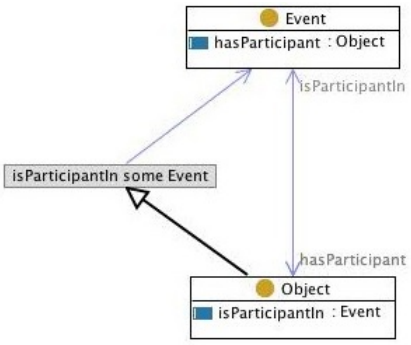

Figura 3: Participation Content ODP

The Communities content ODP represents a community whose members are agents. An agent is any executor of an action, whether physical or social. For example, a community of magistrates where these agents are responsible for celebrating civil marriages. Further, a community may be one element of another community.

3 Ontology Refinement Based on Theory Revision and ODP

In this section, we present a method to represent ODPs in First-Order Logic (FOL) and our proposal for ontology refinement through Theory Revision and ODPs.

3.1 Modeling Ontology Design Patterns in First-Order Logic

An ODP, described as in Figures 3 and 2, is composed of classes and relations generic enough to be able to express how modeled domain concepts interact. Therefore, the method used to model ODPs in FOL consists of 2steps:

1. Define unary predicates from the classes.
2. Define n-ary predicates (relations) from the existing relationships between classes.

Classes are defined by unary predicates, such as:

- object(O).
- event(E).

The relations are defined by n-ary predicates, where each term corresponds to a concept involved in the relationship. For example, the concepts Object and Event in the Participation ODP in Figure 3 have a relationship. Therefore, binary predicate is created in order to associate these two concepts, as follows:

- isParticipantIn(O, E).
- hasParticipant(E, O).

Thus, in a generic way, the ODPs are fully defined as shown in Table 1, for example.

```
/* Communities ODP */
member(M) :- ?(M).
community :- ?(C).
isMemberOf(M, C) :- member(M), community(C), ?(M, C).
hasMember(C, M) :- isMemberOf(M, C).
/* Co-participation ODP */
event(E) :- ?(O).
object(O) :- ?(O).
isParticipantIn(O, E) :- object(O), event(E), ?(O, E).
hasParticipant(E, O) :- isParticipantIn(O, E).
coParticipatesWith(O1, O2, E) :- isParticipantIn(O1, E), isParticipantIn(O2, E).
/* Participation ODP */
object(O) :- ?(O).
event(E) :- ?(E).
isParticipantIn(O, E) :- object(O), event(E), ?(O, E).
hasParticipant(E, O) :- isParticipantIn(O, E).
```

Tabela 1: Communities, Co-Participation and Participation ODPs broadly defined

The general structures shown in Table 1 represent the content ODPs called communities, co-participation and participation, expressed in first-order logic. The question marks represent domain dependent information. Thus, the instantiated ODP would be the following manner in Table 2:

```
object(O) :- person(O).
event(E) :- marriage(E).
isParticipantIn(O, E) :- object(O), event(E), participatesIn(O, E).
hasParticipant(E, O) :- isParticipantIn(O, E).
```

Tabela 2: Instantiated Participation ODP

A part of instantiation of an ODP is made by binding the objects defined therein with modeled domain concepts. For instance, in a domain where a person is an object and a marriage is an event, the representation in FOL is:

- object(O) :- person(O).
- event(E) :- marriage(E).

In another part of the instantiation of an ODP, relations can be defined from the validation of the participating objects and the connection to a modeled domain relationship, or from another relation. According the examples below:

- isParticipantIn(O, E) :- object(O), event(E), participatesIn(O, E).
- hasParticipant(E, O) :- isParticipantIn(O, E).

The definition of relation *isParticipantIn* described above, tell us that it is necessarily defined by an object, an event and the domain relationship called *participatesIn*. And the relation *hasParticipant* is defined from the relation *isParticipantIn*.

3.2 Ontology Refinement Proposal

In this section, we describe our approach for ontology refinement through theory revision and ODPs. We discuss how the application of theory revision techniques to automatically revise an ontology, built according to ODPs can benefit from the use of these patterns. Figure 4 describes our proposal.

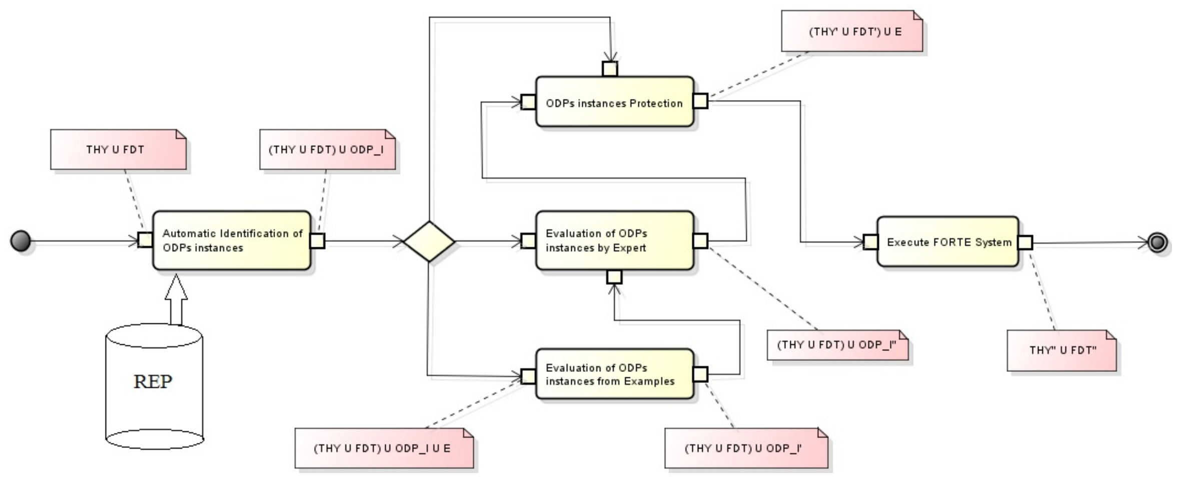
(THY' U FDT') U E
ODPs instances Protection
THY U FDT
(THY U FDT) U ODP_I
Automatic Identification of ODPs instances
Evaluation of ODPs instances by Expert
Execute FORTE System
REP
(THY U FDT) U ODP_I''
THY'' U FDT''
Evaluation of ODPs instances from Examples
(THY U FDT) U ODP_I U E
(THY U FDT) U ODP_I'

Figura 4: Proposal's Schema

In this work, we assume that the ontology axioms are described in first-order clauses, allowing the application of a theory revision system, such as FORTE.

Since ODPs are modeling solutions with recognized good quality, that solve recurrent ontology modeling problems and have been tested and evaluated, they can be treated as background knowledge. In this condition, they should be protected from the revision process, because they are considered correct.

To start the pre-revision process begins with the identification step. This step uses a repository of general structures for each registered ODPs, as shown in Table 1. It receives as its inputs the theory ($THY \cup FDT$) and the repository of ODPs (REP), as shown in Figure 4. In FDT, the ODPs are found instantiated. This means that the question marks were replaced by predicates of the domain. In THY, clauses adhering to an ODP contain a predicate belonging to the ODP. The pre-revision process then tries to find the registered ODPs in the REP in the FDT. If any ODP is found, we try to find the clauses in the THY that reference these ODPs. We call this subset of THY's clauses identified ODP_I.

After we obtain the ODP_I and according to information previously known, the pre-revision process can take one of the following decisions:

1. Consider all ODP_I as background knowledge and protects them, i.e. include ODP_I in FDT.
2. Submit ODP_I to a specialist who will decide which clauses should be part of the background knowledge.
3. The ODP_I is submitted to the specialist, who will decide which clauses should be part of the background knowledge, after analysis using the positive and negative examples.

In options (2) and (3), the THY's clauses can also be selected for protection. So, consider that the THY contains, among others, the following clauses:

1. allowedAdoption(J, E) :- isParticipantIn(J, E), isMemberOf(J, tj).
2. adopt(P, C, E) :- coParticipatesWith(P, C, E).
3. adoptLegally(P, C, J, E) :- allowsAdoption(J, E), adopt(P, C, E).

The ODP_I set generated during the pre-revision process contains the clauses (1) and (2). After generating this set, the process continues by one of options mentioned above. In option (1) all set ODP_I is forwarded to protection step. In option (2), the specialist evaluates the ODP_I and THY according to their expertise and the set of clauses selected by him (ODP_I") is sent to protection step. And the option (3), we try to use the set of examples (E) to evaluate the ODP_I and THY. The specialist receives the set of clauses evaluated (ODP_I') by examples. It then evaluates the received set and then generates the set of clauses evaluated by it (ODP_I"). So this set is sent to the protection step.

The last step of the pre-revision process is the protection of clauses selected. This step receives a set of clauses (ODP_I or ODP_I") to be inserted in the FDT to generate FDT'. The THY is also updated to generate THY'. So, FDT' and THY' are forwarded to the revision process that will generate THY".

The theory revision process transforms an initial incorrect or incomplete theory in a revised theory that correctly represents domain of knowledge. In our proposal, it is conducted through the FORTE system as described in Algorithm 1. The revised theory can represent the transformation of the initial theory. This transformation always occurs when the initial theory is improved by the process.

4 Scenario of use

In our proposal, we work with the Family Law Domain, in accord with Brazilian Civil Code, available in PDF at Library of the Senate of the Brazilian National Congress. On 05.05.2011, the Federal Supreme Court (STF) defined are valid civil marriages of same-sex couples. And the National Council of Justice(CNJ) reinforced this decision by Resolution nÂº 175 of 05.14.2013. This decision causes a change of the Brazilian Civil Code, which defines civil marriage as an union between a man and a woman, so would need a logic theory to represent this are were revised.

Based on the new understanding given by STF, two generalization scenarios were defined: in clauses without ODP and in clauses with ODP. The Table 3 shows the THY used to evaluate the scenario of generalization in clauses without ODP.

```
1 siblings(X, Y) :- fdt:progenitor(Z, X), fdt:progenitor(Z, Y), X \== Y.
2 uncles(X, Y) :- siblings(X, Z), fdt:progenitor(Z, Y).
3 uncles(X, Y) :- fdt:married(X, Z), siblings(Z, K), progenitor(K, Y).
4 cousins(X, Y) :- fdt:progenitor(Z, X), fdt:progenitor(K, Y), siblings(Z, K).
5 grandParents(X, Y) :- fdt:progenitor(X, Z), fdt:progenitor(Z, Y).
6 grandFather(X, Y) :- fdt:gender(X, male), grandParents(X, Y).
7 grandMother(X, Y) :- fdt:gender(X, female), grandParents(X, Y).
8 father(X, Y) :- fdt:gender(x, male), fdt:progenitor(X, Y).
9 mother(X, Y) :- fdt:gender(X, female), fdt:progenitor(X, Y).
10 son(X, Y) :- fdt:gender(X, male), fdt:progenitor(Y, X).
11 daughter(X, Y) :- fdt:gender(X, female), fdt:progenitor(Y, X).
12 brother(X, Y) :- fdt:gender(X, male), siblings(X, Y).
13 sister(X, Y) :- fdt:gender(X, female), siblings(X, Y).
14 uncle(X, Y) :- fdt:gender(X, male), uncles(X, Y).
15 aunt(X, Y) :- fdt:gender(X, female), uncles(X, Y).
16 nephew(X, Y) :- fdt:gender(X, male), uncles(Y, X).
17 niece(X, Y) :- fdt:gender(X, female), uncles(Y, X).
18 cousin(X, Y) :- fdt:gender(X, male), cousins(X, Y).
19 press(X, Y) :- fdt:gender(X, female), cousins(X, Y).
20 ancestral(X, Y) :- fdt:progenitor(X, Y).
21 ancestral(X, Y) :- fdt:progenitor(Z, Y), ancestral(X, Z).
22 descendent(X, Y) :- fdt:progenitor(Y, X).
23 descendent(X, Y) :- fdt:progenitor(Y, Z), descendent(X, Z).
24 isMarried(X) :- fdt:married(X, Y), not fdt:dead(Y).
25 ableToCivilMarriage(H, W) :- fdt:gender(H, male), fdt:gender(W, female).
26 ableToCivilMarriage(H, W) :- fdt:gender(H, female), fdt:gender(W, male).
27 isUnableToCivilMarriage(H, W) :- ancestral(H, W).
28 isUnableToCivilMarriage(H, W) :- descendent(H, W).
29 isUnableToCivilMarriage(H, W) :- ancestral(W, H).
30 isUnableToCivilMarriage(H, W) :- descendent(W, H).
31 isUnableToCivilMarriage(H, W) :- fdt:civilKinship(H, W).
32 isUnableToCivilMarriage(H, W) :- fdt:killed(H, S), fdt:married(S, W).
33 isUnableToCivilMarriage(H, W) :- fdt:killed(W, S), fdt:married(H, S).
34 isUnableToCivilMarriage(H, W) :- isMarried(H), fdt:married(H, W2), W \== W2.
35 isUnableToCivilMarriage(H, W) :- isMarried(W), fdt:married(W, H2), H \== H2.
36 isUnableToCivilMarriage(H, W) :- uncles(H, W).
37 isUnableToCivilMarriage(H, W) :- uncles(W, H).
38 isUnableToCivilMarriage(H, W) :- cousins(H, W).
39 isUnableToCivilMarriage(H, W) :- fdt:adopted(P, H), fdt:progenitor(P, W).
40 isUnableToCivilMarriage(H, W) :- fdt:adopted(P, W), fdt:progenitor(P, H).
41 isUnableToCivilMarriage(H, W) :- fdt:adopted(H, C), fdt:married(C, W), fdt:dead(C).
42 isUnableToCivilMarriage(H, W) :- fdt:adopted(W, C), fdt:married(H, C), fdt:dead(C).
43 not_UnableToCivilMarriage(H, W) :- not isUnableToCivilMarriage(H, W).
44 allowedMarriage(J, E) :- fdt:isParticipantIn(J, E), fdt:isMemberOf(J, tj).
45 canMarry(H, W, E) :- fdt:coParticipatesWith(H, W, E), not_UnableToCivilMarriage(H, W).
46 civilMarriage(H, W, J, E) :- allowedMarriage(J, E), canMarry(H, W, E), ableToCivilMarriage(H, W).
```

Tabela 3: THY for the Scenario of Generalization of Clauses without ODP.

Table 3 contains forty six clauses of THY. Of these, clause(43), clause(44), clause(45) are related to the ODPs (communities, participation and co-participation). And clause(45) is the clause with the ODPs instances. The co-participation ODP instance is formed by variables **H**, **W** and **E**. They represent the couple (H and W) and the marriage (E). The antecedent **canMarry** (clause(44)) ensures that co-participation pattern is being followed. The participation ODP instance is formed by variables **J** and **E**, participant and event respectively. The antecedent **allowedMarriage** (clause(43)) ensures that participation pattern is being followed. Furthermore, this antecedent ensures compliance with the communities pattern.

The Tables 4 and 5 show the background knowledge used in scenarios. The first eleven rows form the communities, co-participation and participation content ODPs. The remaining lines are background knowledge of the domain.

```
/* Communities ODP */
member(M) :- magistrate(M).
community :- group(C).
isMemberOf(M, C) :- member(M), community(C), belongs(M, C).
hasMember(C, M) :- isMemberOf(M, C).
/* Co-participation and Participation ODPs */
event(E) :- marriage(E).
object(O) :- person(O).
isParticipantIn(O, E) :- object(O), event(E), participatesIn(O, E).
hasCoParticipant(E, O) :- isParticipantIn(O, E).
coParticipatesWith(O1, O2, E) :- isParticipantIn(O1, E), isParticipantIn(O2, E).
belongs(luiz, tj), belongs(marcio, tj), belongs(leticia, tj), group(tj), matrimony(m1).
matrimony(m2), matrimony(m3), matrimony(m4), matrimony(m5), matrimony(m6), matrimony(m7).
matrimony(m8), matrimony(m9), matrimony(m10), matrimony(m11), matrimony(m12).
matrimony(m13), matrimony(m14), matrimony(m15), matrimony(m16), matrimony(m17).
matrimony(m18), matrimony(m19), matrimony(m20), matrimony(m21), matrimony(m22).
matrimony(m23), matrimony(m24), matrimony(m25), matrimony(m26), matrimony(m27).
matrimony(m28), matrimony(m29), matrimony(m30), matrimony(m31), matrimony(m32).
matrimony(m33), matrimony(m34), matrimony(m35), matrimony(m36), matrimony(m37).
matrimony(m38), matrimony(m39), matrimony(m40), matrimony(m41), matrimony(m42).
participatesIn(carla, m1), participatesIn(luiza, m1), participatesIn(luiz, m1).
participatesIn(amanda, m2), participatesIn(gabriel, m2), participatesIn(marcio, m2).
participatesIn(marcelo, m3), participatesIn(sonia, m3), participatesIn(leticia, m3).
participatesIn(diego, m4), participatesIn(jorge, m4), participatesIn(luiz, m4).
participatesIn(guilherme, m5), participatesIn(sofia, m5), participatesIn(marcio, m5).
participatesIn(thiago, m6), participatesIn(alberto, m6), participatesIn(leticia, m6).
participatesIn(renata, m7), participatesIn(alfredo, m7), participatesIn(luiz, m7).
participatesIn(lara, m8), participatesIn(yasmin, m8), participatesIn(marcio, m8).
participatesIn(amanda, m9), participatesIn(sofia, m9), participatesIn(leticia, m9).
participatesIn(renata, m10), participatesIn(sonia, m4), participatesIn(alberto, m11).
participatesIn(sonia, m10), participatesIn(luiz, m10), participatesIn(marcelo, m11).
participatesIn(gabriel, m11), participatesIn(marcio, m11), participatesIn(guilherme, m12).
participatesIn(alfredo, m12), participatesIn(leticia, m12), participatesIn(carla, m13).
participatesIn(jorge, m13), participatesIn(luiz, m13), participatesIn(diego, m14).
participatesIn(luiza, m14), participatesIn(marcio, m14), participatesIn(thiago, m15).
participatesIn(yasmin, m15), participatesIn(leticia, m15), participatesIn(lara, m16).
participatesIn(carla, m17), participatesIn(augusto, m17), participatesIn(marcio, m17).
participatesIn(amanda, m18), participatesIn(joaquim, m18), participatesIn(leticia, m18).
participatesIn(maria, m19), participatesIn(diego, m19), participatesIn(luiz, m19).
participatesIn(marcos, m20), participatesIn(sonia, m20), participatesIn(marcio, m20).
participatesIn(marcelo, m21), participatesIn(renata, m21), participatesIn(leticia, m21).
participatesIn(marcos, m22), participatesIn(yasmin, m22), participatesIn(luiz, m22).
participatesIn(jorge, m23), participatesIn(luiza, m23), participatesIn(marcio, m23).
participatesIn(manuel, m24), participatesIn(sofia, m24), participatesIn(leticia, m24).
participatesIn(thiago, m25), participatesIn(julia, m25), participatesIn(luiz, m25).
participatesIn(maria, m26), participatesIn(diego, m26), participatesIn(marcio, m26).
participatesIn(bruno, m27), participatesIn(roberta, m27), participatesIn(leticia, m27).
```

Tabela 4: Background Knowledge

participatesIn(bruna, m28), participatesIn(roberto, m28), participatesIn(luiz, m28).
participatesIn(alberto, m16), participatesIn(luiz, m16), participatesIn(anderson, m29).
participatesIn(solange, m29), participatesIn(marcio, m29), participatesIn(caio, m30).
participatesIn(heloisa, m30), participatesIn(leticia, m30), participatesIn(roberto, m31).
participatesIn(roberta, m31), participatesIn(luiz, m31), participatesIn(bruno, m32).
participatesIn(bruna, m32), participatesIn(marcio, m32), participatesIn(augusto, m33).
participatesIn(julieta, m33), participatesIn(leticia, m33), participatesIn(joaquim, m34).
participatesIn(maria, m34), participatesIn(luiz, m34), participatesIn(manuel, m35).
participatesIn(eduarda, m35), participatesIn(marcio, m35), participatesIn(julio, m36).
participatesIn(lucia, m36), participatesIn(leticia, m36), participatesIn(alberto, m37).
participatesIn(carla, m37), participatesIn(luiz, m37), participatesIn(amanda, m38).
participatesIn(luiza, m38), participatesIn(marcio, m38), participatesIn(marcelo, m39).
participatesIn(sofia, m39), participatesIn(leticia, m39), participatesIn(diego, m40).
participatesIn(gabriel, m40), participatesIn(luiz, m40), participatesIn(guilherme, m41).
participatesIn(sonia, m41), participatesIn(marcio, m41), participatesIn(thiago, m42).
participatesIn(sonia, m42), participatesIn(leticia, m42), married(anderson, solange).
married(caio, heloisa),married(roberto, roberta), married(bruno, bruna), married(augusto, julieta).
married(joaquim, maria), married(manuel, eduarda), married(julio, lucia).
progenitor(anderson, augusto), progenitor(anderson, maria), progenitor(solange,augusto).
progenitor(solange, maria), progenitor(caio, julieta), progenitor(caio, joaquim).
progenitor(heloisa, julieta), progenitor(heloisa, joaquim), progenitor(roberto, manuel).
progenitor(roberto, lucia), progenitor(roberta, manuel), progenitor(roberta, lucia).
progenitor(bruno, eduarda), progenitor(bruno, julio), progenitor(bruna, eduarda).
progenitor(bruna, julio), progenitor(augusto, marcelo), progenitor(julieta, marcelo).
progenitor(manuel, sonia), progenitor(eduarda, sonia), progenitor(joaquim, thiago).
progenitor(maria, thiago), progenitor(julio, alberto), progenitor(lucia, alberto).
progenitor(augusto, carla), progenitor(julieta, carla), progenitor(julio, luiza).
progenitor(lucia, luiza), progenitor(joaquim, renata), progenitor(maria, renata).
progenitor(manuel, alfredo), progenitor(eduarda, alfredo), progenitor(augusto, diego).
progenitor(julieta, diego), progenitor(manuel, jorge), progenitor(eduarda, jorge).
progenitor(joaquim, guilherme), progenitor(maria, guilherme), progenitor(julio, sofia).
progenitor(lucia, sofia), progenitor(augusto, amanda), progenitor(julieta, amanda).
progenitor(julio, gabriel), progenitor(lucia, gabriel), progenitor(joaquim, lara).
progenitor(maria, lara), progenitor(manuel, yasmin), progenitor(eduarda, yasmin).
civilkinship(anderson, caio), civilkinship(anderson, heloisa), civilkinship(solange, caio).
civilKinship(solange, heloisa), civilKinship(roberto, bruno), civilKinship(roberto, bruna).
civilKinship(roberta, bruno), civilKinship(roberta, bruna), civilKinship(augusto, joaquim).
civilKinship(julieta, maria), civilKinship(manuel, julio), civilKinship(eduarda, lucia).
killed(tulio, helena), dead(helena), adopted(manuel, marcos), adopted(eduarda, marcos).
magistrate(luiz), magistrate(marcio), magistrate(leticia), magistracy(luiz, chiefJudge).
magistracy(marcio, judge), magistracy(leticia, chiefJudge), person(marcelo), person(sonia).
person(luiz), person(thiago), person(alberto), person(marcio), person(carla), person(luiza).
person(leticia), person(renata), person(alfredo), person(guilherme), person(sofia), person(diego).
person(jorge), person(lara), person(yasmin), person(amanda), person(gabriel), person(anderson).
person(solange), person(caio), person(heloisa), person(roberto), person(roberta), person(bruno).
person(bruna), person(augusto), person(julieta), person(joaquim), person(maria), person(manuel).
person(eduarda), person(julio), person(lucia), person(tulio), person(helena), person(marcos).
gender(marcelo, male), gender(sonia, female), gender(luiz, male), gender(thiago, male).
gender(alberto, male), gender(marcio, male), gender(carla, female), gender(luiza, female).
gender(leticia, female), gender(renata, female), gender(alfredo, male), gender(guilherme, male).
gender(sofia, female), gender(diego, male), gender(jorge, male), gender(lara, female).
gender(yasmin, female), gender(amanda, female), gender(gabriel, male), gender(anderson, male).
gender(solange, female), gender(caio, male), gender(heloisa, female), gender(roberto, male).
gender(roberta, female), gender(bruno, male), gender(bruna, female), gender(augusto, male).
gender(julieta, female), gender(joaquim, male), gender(maria, female), gender(manuel, male).
gender(eduarda, female), gender(julio, male), gender(lucia, female), gender(tulio, male).
gender(helena, female), gender(marcos, male).

Tabela 5: Background Knowledge (Continuation)

The Table 6 shows the positive and negative examples, according to this decision.

Positive Examples:

civilMarriage(carla, luiza, luiz, m1), civilMarriage(amanda, gabriel, marcio, m2).
civilMarriage(marcelo, sonia, leticia, m3), civilMarriage(diego, jorge, luiz, m4).
civilMarriage(guilherme, sofia, marcio, m5), civilMarriage(thiago, alberto, leticia, m6).
civilMarriage(renata, alfredo, luiz, m7), civilMarriage(lara, yasmin, marcio, m8).
civilMarriage(amanda, sofia, leticia, m9), civilMarriage(renata, sonia, luiz, m10).
civilMarriage(marcelo, gabriel, marcio, m11), civilMarriage(guilherme, alfredo, leticia, m12).
civilMarriage(carla, jorge, luiz, m13), civilMarriage(diego, luiza, marcio, m14).
civilMarriage(thiago, yasmin, leticia, m15), civilMarriage(lara, alberto, luiz, m16).
civilMarriage(anderson, solange, marcio, m29), civilMarriage(caio, heloisa, leticia, m30).
civilMarriage(roberto, roberta, luiz, m31), civilMarriage(bruno, bruna, marcio, m32).
civilMarriage(augusto, julieta, leticia, m33), civilMarriage(joaquim, maria, luiz, m34).
civilMarriage(manuel, eduarda, marcio, m35), civilMarriage(julio, lucia, leticia, m36).
civilMarriage(alberto, carla, luiz, m37), civilMarriage(amanda, luiza, marcio, m38).
civilMarriage(marcelo, sofia, leticia, m39), civilMarriage(diego, gabriel, luiz, m40).
civilMarriage(guilherme, sonia, marcio, m41), civilMarriage(thiago, sonia, leticia, m42).

Negative Examples:

civilMarriage(gabriel, carla, luiz, m1), civilMarriage(amanda, luiza, marcio, m2).
civilMarriage(marcelo, sonia, luiz, m3), civilMarriage(diego, jorge, sonia, m4).
civilMarriage(diego, jorge, luiz, m5), civilMarriage(alfredo, thiago, leticia, m6).
civilMarriage(renata, alberto, luiz, m7), civilMarriage(lara, yasmin, leticia, m8).
civilMarriage(guilherme, alfredo, leticia, m9), civilMarriage(amanda, sofia, leticia, m10).
civilMarriage(marcelo, gabriel, alberto, m11), civilMarriage(guilherme, alfredo, marcio, m12).
civilMarriage(yasmin, carla, luiz, m13), civilMarriage(diego, yasmin, marcio, m14).
civilMarriage(lara, yasmin, leticia, m15), civilMarriage(lara, alberto, marcio, m1).
civilMarriage(lara, gabriel, luiz, m16), civilMarriage(carla, augusto, marcio, m17).
civilMarriage(augusto, carla, marcio, m17), civilMarriage(amanda, joaquim, leticia, m18).
civilMarriage(maria, diego, luiz, m19), civilMarriage(marcos, sonia, marcio, m20).
civilMarriage(marcelo, renata, leticia, m21), civilMarriage(marcos, yasmin, luiz, m22).
civilMarriage(jorge, luiza, marcio, m23), civilMarriage(manuel, sofia, leticia, m24).
civilMarriage(thiago, julia, luiz, m25), civilMarriage(maria, diego, marcio, m26).
civilMarriage(bruno, roberta, leticia, m27), civilMarriage(bruna, roberto, luiz, m28).

Tabela 6: Examples

The pre-revision process proposed by us use one repository of ODPs to identify clauses with ODPs instances. Thus, considering this repository, the clauses (43) and (44) are selected and moved to the FDT. Why are referenced directly or indirectly by clause (44), clauses (1) to (4), clauses (20) to (24) and clauses (26) to (42), will also be moved to the FDT. The Table 7 shows the results obtained with the revision of logical theory modified with these changes.

Metrics	Value
Number of Generalization Points Found	6
Number of Generalization Operators Executed	6
Number of Compaction Operators Executed	1
Initial THY Size	135
Modified THY Size	55
Revised THY Size	54
Initial Test Accuracy	85.00%
Final Test Accuracy	100.00%
Revision RunTime	1,185 ms

Tabela 7: Results of Generalization of Clauses without ODPs.

The Table 7 is composed of metrics and their value. These were collected from the execution of the FORTE system, except the metric of the Initial THY size. The first three metrics (number of generalization points found, number of generalization operators executed and number of compaction operators executed) are explained in subsection 2.1 and their values inform respectively: the number of revision points (generalization points) found in the theory, the number of revision operators (generalization) and the number of compression operators performed. The Initial THY Size metric is obtained from the count of the number of predicates in the THY shown in Table 3, which represents the THY before the implementation of the pre-revision stage. The Modified THY Size metric is obtained from the count of the number of predicates in the THY modified by pre-revision stage and will be submitted to the revision stage. The Revised THY Size metric is obtained from the count of the number of predicates in the THY revised by revision process. The Initial Test Accuracy metric represents the percentage of examples correctly classified by the modified theory. The Final Test Accuracy metric represents the percentage of examples correctly classified by the revised theory. And the Revision RunTime metric is the time spent by the revision stage in milliseconds. The Table 8 shows the revised theory obtained through the revision process mentioned above.

```
civilMarriage(A,B,C,D):-fdt:allowedMarriage(C,D),fdt:canMarry(A,B,D),ableToCivilMarriage(A,B).
ableToCivilMarriage(A,B).
ableToCivilMarriage(A,B):-fdt:gender(A,male),fdt:gender(B,female).
press(A,B):-fdt:gender(A,female),fdt:cousins(A,B).
cousin(A,B):-fdt:gender(A,male),fdt:cousins(A,B).
niece(A,B):-fdt:gender(A,female),fdt:uncles(B,A).
nephew(A,B):-fdt:gender(A,male),fdt:uncles(B,A).
aunt(A,B):-fdt:gender(A,female),fdt:uncles(A,B).
uncle(A,B):-fdt:gender(A,male),fdt:uncles(A,B).
sister(A,B):-fdt:gender(A,female),fdt:siblings(A,B).
brother(A,B):-fdt:gender(A,male),fdt:siblings(A,B).
daughter(A,B):-fdt:gender(A,female),fdt:progenitor(B,A).
son(A,B):-fdt:gender(A,male),fdt:progenitor(B,A).
mother(A,B):-fdt:gender(A,female),fdt:progenitor(A,B).
father(A,B):-fdt:gender(x,male),fdt:progenitor(A,B).
grandMother(A,B):-fdt:gender(A,female),grandParents(A,B).
grandFather(A,B):-fdt:gender(A,male),grandParents(A,B).
grandParents(A,B):-fdt:progenitor(A,C),fdt:progenitor(C,B).
```

Tabela 8: Revised Theory obtained from generalization of clauses without ODP.

Now, for the demonstration of the scenario of generalization in clauses with ODP, let's consider the Background Knowledge shown in Tables 4 and 5, and the THY shown in Tables 9 and 10 below.

```
1 siblings(X, Y) :- fdt:progenitor(Z, X), fdt:progenitor(Z, Y), X \== Y.
2 uncles(X, Y) :- siblings(X, Z), fdt:progenitor(Z, Y).
3 uncles(X, Y) :- fdt:married(X, Z), siblings(Z, K), progenitor(K, Y).
4 cousins(X, Y) :- fdt:progenitor(Z, X), fdt:progenitor(K, Y), siblings(Z, K).
5 grandParents(X, Y) :- fdt:progenitor(X, Z), fdt:progenitor(Z, Y).
6 grandFather(X, Y) :- fdt:gender(X, male), grandParents(X, Y).
7 grandMother(X, Y) :- fdt:gender(X, female), grandParents(X, Y).
8 father(X, Y) :- fdt:gender(x, male), fdt:progenitor(X, Y).
9 mother(X, Y) :- fdt:gender(X, female), fdt:progenitor(X, Y).
10 son(X, Y) :- fdt:gender(X, male), fdt:progenitor(Y, X).
11 daughter(X, Y) :- fdt:gender(X, female), fdt:progenitor(Y, X).
12 brother(X, Y) :- fdt:gender(X, male), siblings(X, Y).
13 sister(X, Y) :- fdt:gender(X, female), siblings(X, Y).
14 uncle(X, Y) :- fdt:gender(X, male), uncles(X, Y).
15 aunt(X, Y) :- fdt:gender(X, female), uncles(X, Y).
16 nephew(X, Y) :- fdt:gender(X, male), uncles(Y, X).
17 niece(X, Y) :- fdt:gender(X, female), uncles(Y, X).
18 cousin(X, Y) :- fdt:gender(X, male), cousins(X, Y).
19 press(X, Y) :- fdt:gender(X, female), cousins(X, Y).
20 ancestral(X, Y) :- fdt:progenitor(X, Y).
21 ancestral(X, Y) :- fdt:progenitor(Z, Y), ancestral(X, Z).
22 descendent(X, Y) :- fdt:progenitor(Y, X).
23 descendent(X, Y) :- fdt:progenitor(Y, Z), descendent(X, Z).
24 isMarried(X) :- fdt:married(X, Y), not fdt:dead(Y).
25 ableToCivilMarriage(H, W) :- fdt:gender(H, male), fdt:gender(W, female).
```

Tabela 9: THY for the Scenario of Generalization of Clauses with ODP.

26 ableToCivilMarriage(H, W) :- fdt:gender(H, female), fdt:gender(W, male).
27 isUnableToCivilMarriage(H, W) :- ancestral(H, W).
28 isUnableToCivilMarriage(H, W) :- descendent(H, W).
29 isUnableToCivilMarriage(H, W) :- ancestral(W, H).
30 isUnableToCivilMarriage(H, W) :- descendent(W, H).
31 isUnableToCivilMarriage(H, W) :- fdt:civilKinship(H, W).
32 isUnableToCivilMarriage(H, W) :- fdt:killed(H, S), fdt:married(S, W).
33 isUnableToCivilMarriage(H, W) :- fdt:killed(W, S), fdt:married(H, S).
34 isUnableToCivilMarriage(H, W) :- isMarried(H), fdt:married(H, W2), W \== W2.
35 isUnableToCivilMarriage(H, W) :- isMarried(W), fdt:married(W, H2), H \== H2.
36 isUnableToCivilMarriage(H, W) :- uncles(H, W).
37 isUnableToCivilMarriage(H, W) :- uncles(W, H).
38 isUnableToCivilMarriage(H, W) :- cousins(H, W).
39 isUnableToCivilMarriage(H, W) :- fdt:adopted(P, H), fdt:progenitor(P, W).
40 isUnableToCivilMarriage(H, W) :- fdt:adopted(P, W), fdt:progenitor(P, H).
41 isUnableToCivilMarriage(H, W) :- fdt:adopted(H, C), fdt:married(C, W), fdt:dead(C).
42 isUnableToCivilMarriage(H, W) :- fdt:adopted(W, C), fdt:married(H, C), fdt:dead(C).
43 not_UnableToCivilMarriage(H, W) :- not isUnableToCivilMarriage(H, W).
44 allowedMarriage(J, E) :- fdt:isParticipantIn(J, E), fdt:isMemberOf(J, tj).
45 canMarry(H, W, E) :- not_UnableToCivilMarriage(H, W), ableToCivilMarriage(H, W), fdt:coParticipatesWith(H, W, E).
46 civilMarriage(H, W, J, E) :- allowedMarriage(J, E), canMarry(H, W, E).

Tabela 10: THY for the Scenario of Generalization of Clauses with ODP (Continuation).

According to Table 9 and 10, clauses are protected, are the same as the first scenario (43, 44, 1 to 4, 20 to 24 and 26 to 42). The results of the revision of this modified theory is shown in Table 11.

Metrics	Value
Number of Generalization Points Found	3
Number of Generalization Operators Executed	2
Number of Compaction Operators Executed	0
Initial THY Size	135
Modified THY Size	48
Revised THY Size	51
Initial Test Accuracy	75.00%
Final Test Accuracy	100.00%
Revision RunTime	1,700 ms

Tabela 11: Results of Generalization of Clauses with ODPs.

The Table 12 shows the revised theory obtained through the revision process mentioned above.

civilMarriage(A,B,C,D):- allowedMarriage(C,D), not_UnableToCivilMarriage(B, A), coParticipatesWith(A,B,D).
civilMarriage(A,B,C,D):-fdt:allowedMarriage(C,D),fdt:canMarry(A,B,D).
press(A,B):-fdt:gender(A,female),fdt:cousins(A,B).
cousin(A,B):-fdt:gender(A,male),fdt:cousins(A,B).
niece(A,B):-fdt:gender(A,female),fdt:uncles(B,A).
nephew(A,B):-fdt:gender(A,male),fdt:uncles(B,A).
aunt(A,B):-fdt:gender(A,female),fdt:uncles(A,B).
uncle(A,B):-fdt:gender(A,male),fdt:uncles(A,B).
sister(A,B):-fdt:gender(A,female),fdt:siblings(A,B).
brother(A,B):-fdt:gender(A,male),fdt:siblings(A,B).
daughter(A,B):-fdt:gender(A,female),fdt:progenitor(B,A).
son(A,B):-fdt:gender(A,male),fdt:progenitor(B,A).
mother(A,B):-fdt:gender(A,female),fdt:progenitor(A,B).
father(A,B):-fdt:gender(x,male),fdt:progenitor(A,B).
grandMother(A,B):-fdt:gender(A,female),grandParents(A,B).
grandFather(A,B):-fdt:gender(A,male),grandParents(A,B).
grandParents(A,B):-fdt:progenitor(A,C),fdt:progenitor(C,B).

Tabela 12: Revised Theory obtained from generalization of clauses with ODP

5 Related Works

In [9], we find a proposal for a semiautomatic refinement of ontologies in Description Language (DL). This proposal is based on a process RAA (Role Assertion Analysis) and consists of four steps:

- generate a set of graphs from instances of a concept.
- transform the graphs in a formal context.
- analyse the formal context using FCA (Formal Context Analysis).
- transform the FCA results on new DL concepts and relations.

The graph mentioned above is directed and cyclic, its root is the instance of the concept, the vertices are instances of attributes and edges instances of relations. These graphs are explored by a depth-first search algorithm with deep limiter to find relationships between individuals of the ontology. Formal context considers the association of the set of instances of a concept with a set of attributes through a relationship. The FCA abstract conceptual descriptions from a set of objects described by attributes. The fact that the refinement is limited to the discovery of new concepts and relationships, and the limitation of maximum depth used in the depth-first search algorithm, which can prevent better performance, are the weaknesses of this proposal.

A proposal for refinement of ontologies, independent of the representation language, performed by complementing partial instances of ontology design patterns, performed semiautomatic way, is proposed in [10]. The central idea is to look for components within the ontology partially instantiate some ODP. Instantiations of ODPs are identified by their structure and meaning of its elements, expressed by the axioms and lexical features of each element. Once identified a partial instantiation of an ODP, can identify missing elements and generate a list of suggestions for refinement.

The proposal outlined above doesn't really makes the refinement of ontology. As mentioned, it generates a list of suggestions for refinement to a specialist to make the decision whether to apply or not for each item of the list. Therefore, the possibility of ontology contain errors at the end of the process, even exists.

A method for automatic revision of ontologies in OWL (Ontology Web Language) is the proposal for [11]. In the proposal, the ontology in OWL is converted into a logical theory in first-order logic (FOL) to then be revised by the FORTE system. The outcome of the revision, the revised theory, is then converted back into an refined ontology in OWL. However, this work doesn't take into account the design patterns that can be part of the initial ontology.

6 Conclusion

The use of ontology design patterns is an option to improve the ontology refinement. They are modeling solutions with recognized good quality applicable to recurring ontology modeling problems. Can therefore act as background knowledge in order to improve theory revision process.

The theory revision is a way to automate the process of refinement of ontology that can take advantage of the characteristics of ODPs to reduce the search space of revision points. So if an ontology is constructed using ODPs, it is possible to improve the refinement process by reducing the search space of revision points.

But for this, we need information that will help us during the process execution. This information is organized into a data repository and allow us to identify the used ODPs. Thus, we will be able in dealing with existing ODPs in the ontology and improve logical theory accuracy.

The fact that our proposal offer multiple options to realize the protection of ODPs (automatic, expert evaluation and use of positive and negative examples for further evaluation by experts) is positive because it allows us to overcome existing deficiencies in each. In automatic option, there is no possibility of errors caused by human intervention and is also the option to spend less time, but it would be unable to generate an appropriate solution in scenarios of generalization of clauses with ODPs, as shown in section 4. The option of expert evaluation is able to overcome the problem mentioned for automatic option, but the disadvantage is subject to errors due to human intervention, besides being time consuming. The option that uses examples to a previous analysis before evaluation of the expert is an intermediate between the first two. It tries to reduce the possibility of errors in a process with human intervention to present a recommendation list of ODPs that should be protected to the specialist, while it takes to overcome the disadvantage mentioned for automatic option. However, the evaluation performed from examples can be as costly as the revision of the theory itself.

However, the protection of clauses with ODPs may result in a revision process unable to correct logical theory or generate solutions of questionable quality. Another problem encountered was the need to protect clauses unrelated to an ODP because it is subject to failure and therefore can not be considered as background knowledge. And yet, it is an antecedent of a clause where another antecedent is an ODP. As an example, we can mention the revised theory of generalization in the scenario of clauses with ODPs.

As suggestions for future work derived from the evaluation of the proposed scenarios, we can mention the realization of a more complete evaluation with scenarios of generalization and specialization between the current way of doing revision theory and proposed in this paper. And the resolution of the problem of inclusion in background knowledge unrelated clauses to an ODP. The observation that the realization of the option to perform the evaluation from the examples, the ODPs to be protected is expensive, we suggest as future work the making of the process that performs this task less costly way.

Our work is focused on ODPs expressed in a single clause. A future work may address multiple clauses ODPs, removing a barrier to adoption of the theory revision to the ontology refinement. The development of an expert system to replace the human specialist in the pre-revision process, is also an option for future work. Another work that can be done is to develop a revision process that attempts to preserve the ODPs instances, or replace incorrect ODPs instances in another correct.

REFERENCES

[1] A. Gangemi and V. Presutti. “Ontology Design Patterns”. In *Handbook of Ontologies*, edited by S. Staab and R. Studer, pp. 221–243. Springer-Verlag, 2009.

[2] B. C. Grau, I. Horrocks, Y. Kazakov and U. Sattler. “Just the Right Amount: Extracting Modules from Ontologies”. In *Proceedings of the 16th International Conference on World Wide Web*, pp. 717–726, 2007.

[3] J. Hartmann, R. Palma and A. Gómez-Pérez. “Ontology Repositories”. In *Handbook of Ontologies*, edited by S. Staab and R. Studer, pp. 551–571. Springer-Verlag, 2009.

[4] S. Wrobel. “First-order Theory Refinement”. In *Advances in Inductive Logic Programming*, edited by L. D. Raedt, pp. 14–33. IOS Press, 1996.

[5] S. Muggleton. *Inductive logic programming*. Academic Press, New York, 1992.

[6] B. L. Richards and R. J. Mooney. “Automated Refinement of First-Order Horn-Clause Domain Theories”. *Machine Learning*, vol. 19, pp. 95–131, 1995.

[7] A. Duboc, A. Paes and G. Zaverucha. “Using the Bottom Clause and Mode Declarations in FOL Theory Revision from Examples”. *Machine Learning*, vol. 76, pp. 73–109, 2009.

[8] V. Pressuti, A. Gangemi, S. David, G. de Cea, M. Suárez-Figueroa, E. Montiel-Ponsoda and M. Poveda. *A Library of Ontology Design Patterns: reusable solutions for collaborative design of networked ontologies*. Neon-project, 2008.

[9] A. Coulet, M. Smail-Tabbone, A. Napoli and M. Devignes. “Ontology Refinement through Role Assertion Analysis: example in pharmacogenomics”. *21st International Workshop on Description Logics*, vol. 353, 2008.

[10] N. Nikitina, S. Rudolph and S. Blohm. “Refining Ontologies by Pattern-Based Completion”. *International Semantic Web Conference.*, 2009.

[11] F. Leão, K. Revoredo and F. Baião. “Um Framework para Refinamento de Ontologias Através de Técnicas de Revisão de Teorias”. *Sociedade Brasileira de Computação - Encontro Nacional de Inteligência Artificial*, 2011.