POOE
A Plug-in to Oversee Ontology Evolution

Emile Tawamba, ENSP-Yaoundé I, Roger Nkambou, UQAM, Montréal, Bernabé Batchakui, ENSP-Yaoundé I, Claude Tangha, ENSP-Yaoundé I, UPAC

Abstract—Ontologies are becoming nowadays a recurrent topic in many research areas of computer science. Among other things, they are used as best model for representing knowledge of a given field of expertise. More specifically, the evolution of Semantic Web mostly relies on the use of ontologies in describing various web resources. However, the world is changing, so do resources and knowledge. Consequently, ontologies on which are based knowledge and resources are also called to evolve. One of the solutions helping to ensure this evolution is to manage the development process in order to guarantee supported services as well as some quality requirements. Many tools have been designed to ensure designing/editing of ontologies, but none of them has enough functions to ensure their evolution. Evolution of ontology considered as its adaptation to changes in the field is a key issue in the ontology engineering field. Several studies have been focused on this problem, but proposed solutions did not manage all the side effects that may derive from the evolution. This article proposes an extension of MS-ONTO [1] called POOE. POOE not only supports the evolution effects on a double plan, externally and internally (verification of integrity constraints, quality analysis of the evolved ontology) but it constitutes a true validation tool of a change log.

Index Term—Ontology Evolution, Integrity Constraints, Semantic Referencing, Knowledge-Based Systems.

I. INTRODUCTION
Ontology is defined as “an explicit specification of a conceptualization” [2]. It is also considered as the result of a complete formulation and a rigorous conceptualization (hierarchical organization of pertinent concepts, relationships between these concepts, rules and axioms binding them) [3].

Ontology evolution refer to its capability to adapt from changes that are brought during its life-cycle and to manage the propagation of these changes at the level of the dependent artifact, i.e. objects referred by the ontology as well as other related ontologies and their related applications [4] [5]. Changes are classified into three levels: changes of the modeled area, changes in conceptualization and specification changes [6]. The domain is part of the real world, so it is dynamic and evolves with time. Conceptualization may change due to a new observation or knowledge restructuring. Specification of the changes relate to the formal description of ontology (the representation language) [7].

Changes can be classified into two categories namely the normal evolution and revolution. The first is used to enrich the ontology with new knowledge without jeopardizing existing ones; it is the principle of “ontological continuity”. The revolution on the other hand helps to disprove the true axioms.

The change must be considered as integral part of the life cycle of the ontology so that it can maintain its importance in relation to applications for which it was built. As a matter of fact, the evolution of knowledge of ontology influences the development and maintenance of systems using this ontology. In addition, the quality and the validity of knowledge modeled by the ontology directly affect the quality and the validity of these systems.

The management of ontology changes impels us to take into account a range of change activities. The distinction between these different activities is somewhat ambiguous. When we consider the role of evolution in the life cycle of an ontology and when we adopt terminology databases of the community, four major evolution activities can be distinguished: ontology review, ontology versions management also called versioning, ontology adaptation and ontology evolution.

Ontology review consists in changing the state of an ontology representation to adapt its characteristics to the modeled area. It corresponds to logic manipulations that extend the level of abstraction of ontology. For example, add or remove concepts or refactoring properties [8].

Versioning consists in creating and managing different versions of an ontology, as well as incompatibilities between versions, instances and applications that depend on ontology.

Adaptation can be seen as a practical review in which an older version is first revised and then adapted to a specific application. Changes of adaptation are mostly generated by identification approaches directed by the user.

Ontology Evolution is taking into account "persistent" changes to meet new needs, and to produce new versions. The change of ontology is managed by maintaining the consistency of ontology, ensuring the traceability of changes while preserving their history in order to provide correspondences (mappings) between versions, and controlling the use of forums.

Ensuring the evolution of ontology is a costly operation which requires the services of a competent expert in this domain [9]. However, the significant role of ontologies makes it essential that they should be kept up to date so as to reflect the changes which affect the life cycle of the systems and the applications for which they were conceived [10].

This paper presents a solution we have implemented which integrates the management system of ontologies evolution (MS-ONTO) extended with specific operations of partial deleting and partial reclassification. The paper is organized as follows: the second section presents some related works on the management of ontology evolution. Our solution is then introduced in the third section where a formal description of the operators of changes is given with a general presentation of our global approach towards ontology evolution.
management. The fourth section is the implementation and evaluation of this solution.

II. RELATED WORK

Many research teams and projects have worked on ontology evolution and brought out interesting findings. According to their methods, two broad categories stand out: ontology versioning and management of changes [11] [12]

2.1 Management of Versions

This first class [8] uses the term "versioning". Ontology evolution is defined as the ability to manage changes in the ontology and their effect in creating and maintaining different versions of an ontology. This ability is to identify, differentiate and change versions to specify the relationships that explain the changes made between versions and use mechanisms of access for dependent artifacts. The IMSE methodology, KAON, CONCORDIA and SHOE are classical examples of tools and methods that implement the view.

IMSE’s methodology to support versioning Ontology: ontology evolution is defined as "the ability to manage changes in the ontology and their effects in creating and maintaining different versions of the ontology. This ability is to identify, differentiate and modify the versions while specifying the relationships that explain the changes made between them [13] [14]. Unlike another definition which regards evolution as the passage of a state of ontology to another, by not preserving the latter, this definition rather considers the presence of several states of the same ontology (versions) and the possibility to differentiate them.

This solution has limits. In fact, it does not provide an approach to support the process of evolution of ontology, but rather to support versioning of an ontology after its evolution. Therefore, an analysis model of the relationship between versions of ontology is provided, but without taking into account the management of access to dependent artifacts (referred resources, other ontologies or applications) using versions of ontology. Moreover, no functional framework is offered for helping the integration of all the methodological elements that are suggested.

KAON [5] offers a range of tools for ontologies and the semantic web. Presently, it is one of the rare ontology management systems which has a function dedicated to the recording of changes. During the evolution, KAON saves all the changes carried out to move from one ontology version to another in a folder as an ordered sequence of RDF/XML declarations. The main drawback in KAON approach is that, it deals only with elementary changes.

CONCORDIA [15] on its part defines a conceptual model for the management of changes of a medical terminology. This model adds to each class a unique identifier and these classes could only be subsequently and logically withdrawn but not physically erased. As such, for each class, the Concordia model is capable of tracing all the parent-classes or withdrawing children through its identifiers.

HEFLIN proposes SHOE [16] as a language for representing knowledge on the web. It is a language based on FOL (First order logic). SHOE is based on HTML, which offers primary terms for the management of multiple versions while allowing the association to each version of ontology, a unique identifier and a code stating the compatibility with former versions. It brings out the relevance of each revision/operation on data and requests.

As a whole, ontology evolution solutions of this group do not really treat ontology evolution itself. They rather provide a model for analyzing links between ontology versions, but do not pay attention neither to the management of the dependent artifacts (referred resources, related ontologies, programs in which the evolved ontology is used) nor to the impact of changes on the ontology internal structure and/or semantic. Moreover, the authors do not provide any functional framework to integrate the totality of methodological elements that they propose.

2.2 Management of Changes

The proponents of this approach think that it is necessary to have a follow-up of changes: the request for change has to be interpreted, analyzed and executed under control. In fact, the system should be able to notify a change that might lead to an inconsistent ontology, or better still, it should at least help the knowledge engineer in evolution operations if it cannot completely automate them.

Ontology must remain consistent while evolving. Ontology is consistent in relation to a model if and only if it respects all the constraints of the model. These constraints are either invariables or user-defined. The invariables constraints are those related to the ontology structure; for instance, removing a root concept should not be allowed without a correct reorganization of the ontology. User-defined constraints are those defined by the user such as the maximum number of instance related to a given concept. Semantically speaking, a change maintains consistency only when the resulting ontology is consistent. Hence, change is defined as a function ch(args, prec, postc) with args representing the arguments, prec representing all the pre-conditions and postc representing all post-conditions. Some operations could lead to several options of possible ontologies; it is therefore necessary to choose the most consistent one that responds to user needs. However, several iterations are necessary if we want to obtain an acceptable resultant ontology [17].

This approach has many disadvantages; it consists of a solution which has a high algorithm complexity due to multiple possible iterations without any specific switch-off conditions. The reason being that at each stage, choices have to be made using relevant heuristics.

Another approach in this category is that of Delia [18]. It consists of a semi-automatic system where the user is involved in the ontology evolution process. It proposes versioning of changes which then serve to resolve eventual semantic referencing (based on the ontology) failures. The system records the changes during the evolution phase of the ontology and provides the user with a list of possible semantic references that are impacted. The management of changes is hereby concentrated on the control of the consistency of semantic referencing (meaning the usage of the target ontology – external integrity).

As a whole, the fore-mentioned approaches do not solve the problem of evolution on all its scope. Some are founded on structural aspects (internal integrity) while others focus on the external implications (external integrity). In addition, they lack
a formal framework which entirely integrates the ontology evolution engineering.

III. SPECIFICATION OF THE POOE SOLUTION
Despite the fact that several works have been carried out on the evolution of ontologies, the tools which should make it possible to ensure a guided evolution which guarantees the consistency of final ontology are still to come. The solution that we propose comprises four points: we start by taking into account the semantics behind the various operations of changes which can affect ontology during its life cycle. We have particularly defined new operations to facilitate the execution of an operation or of a set of operations of change. Thirdly, we control the execution of operations according to some used case proposed, and we end by giving a detailed report of the execution of the changes with the new version of ontology.

3.1 Formal Description of Operators of Change
These operators of change are formally defined using Description Logic (DL) as the semantic specification language. Evolution operations are usually done on ontology elements including T-Box axioms (concepts, roles, restrictions, attributes etc.) as well as A-Box axioms (instances).

There are two groups of operators: Basic DL operators such as negation, generalization, specialization, etc., and the other operators which we defined using DL primitives [17] [19] [20]. These operators include adding, removing, merging or grouping ontology elements (concept, role, restriction, etc.) Each operator is semantically defined as illustrated in the following for 3instances of operators. Our contribution here is based on the fact that, actions related to each operator are explicitly represented in the semantic.

The following examples illustrate some of these operators and their formal semantic descriptions. Let us consider (O, C, I, instcon, H_T) where O is an ontology, C is a set of concepts, I is a set of instances, instcon is a function that relates a concept and its formal semantic descriptions. Let us consider (O, C, I, instcon, H_T) where O is an ontology, C is a set of concepts, I is a set of instances, instcon is a function that relates a concept and its formal semantic descriptions. Let us consider (O, C, I, instcon, H_T) where O is an ontology, C is a set of concepts, I is a set of instances, instcon is a function that relates a concept and its formal semantic descriptions. Let us consider (O, C, I, instcon, H_T) where O is an ontology, C is a set of concepts, I is a set of instances, instcon is a function that relates a concept and its formal semantic descriptions. Let us consider (O, C, I, instcon, H_T) where O is an ontology, C is a set of concepts, I is a set of instances, instcon is a function that relates a concept and its formal semantic descriptions.

Table I
Semantic of the operation RemoveConcept

<table>
<thead>
<tr>
<th>Description</th>
<th>RemoveConcept</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preconditions</td>
<td>C ∈ T {root} ∨ ∃D_1,D_2 ∈ T ((C,D_1) ∈ H_T ∧ (D_2,D_1) ∈ H_T)</td>
</tr>
<tr>
<td>Post-conditions</td>
<td>(C,B_i) ∈ H_T</td>
</tr>
<tr>
<td>Actions</td>
<td>1. T (TBox containing C_1,...,C_n, C and D)</td>
</tr>
<tr>
<td></td>
<td>2. (∃1 ≤ i ≤ n) C_i ⊑ C → C_i ⊑ D</td>
</tr>
<tr>
<td></td>
<td>3. C ⊑ D (C is subsume by D)</td>
</tr>
<tr>
<td></td>
<td>4. C ⊑ ⊥ (Remove C)</td>
</tr>
</tbody>
</table>

3.2 Constraints of Semantic Integrity
We defined a set of constraints of semantic integrity which must be taken into account throughout the evolution process. Each single operation of change should lead to the checking of these constraints before the system can approve or reject the change. In all cases, a report must be drawn up. An example of constraints of semantic integrity could consist in setting a maximal DL expressiveness that should be preserved after the evolution process. In this way, it should be possible to preserve the logic of the initial ontology if needed. As such, taking into account the expressiveness includes three main steps:
- Identifying the structures allowed for each sub-language in a «database of structures of sub-languages». Several levels of language could be obtained: For example OWL-LITE, OWL-DL, OWL-FULL [21] if we refer to OWL1.
- Creating a structure checker, a program that is able to verify the ontology structures and to categorize them, in order to determine the ontology expressiveness.
- Ensuring that the degree of expressiveness of the ontology is maintained.

The expressiveness management model described above is presented in the next section. The initial ontology undergoes an audit which determines its expressiveness. Each operation of change which is applied to the ontology must absolutely respect the expressiveness. After the ontology expressiveness is verified, a report is produced.

3.3 Execution of the Operations of Change
The execution of an operation of change or a range of changes goes beyond the control of the expressiveness of ontology. In fact, in addition to the constraints of semantic integrity, each operation semantics is verified using its set of preconditions that should hold before execution and its post-conditions that should hold after execution.

Our overall model of ontology evolution is presented in Figure 01.

The list of changes is a set of operations recorded after the evolution of the ontology model or a set of operations which one must apply to make the ontology evolve. Constraints represent a set of constraints of semantic integrity which must
be verified to guarantee the consistency of the ontology along its evolution. The list of the operation of changes is drawn up by the operators of DL on the one hand, and the operations which we defined on the other hand as mentioned in the previous sections. We have developed a typical scenario where our solution can be used: an ontology evolution parser which goal is to validate an ontology evolution set of operations given an initial ontology and some evolution parameters such as the expressiveness constraints, the list of changes, etc. The parser carries out the operations of change while respecting constraints. A report is produced at the end of the execution as a feedback of the parsing process (Figure 01).

Algorithm of change log validation

ProcessChange (L, O)
//This programme allows for change log validation
Require: O= ontology being changed
L= List of changes
1: For each operation in L
2: verify the pre-condition
3: ExpressivityControl(O, val)
4: Run the operation
5: Verify the constraints
6: Verify the post-condition
7: End for
8: provide the report

ExpressivityControl(O, c, bool)
//This procedure ensures that we have not violated the expressiveness of the ontology following an operation.
Require: O= Ontology being changed
c= A change to process
9: Expressivity(O, val1)
10: Perform the operation c
11: Expressivity(O, val2)
12: If val2>val1
   The expressiveness is violated
   End if
13: Provide report

Expressivity (O, Exp)
//This procedure calculates the expressiveness of an ontology
Require: O= Ontology
14: Find all constructors
15: For each constructor, determine the correspond description logic
16: Exp = the expressivity of the larger description logic

IV. IMPLEMENTATION AND VALIDATION

4.1 Implementation
POOE is implemented and current use-case helps to control the ontology evolution giving as inputs, a list of changes and the initial ontology on which these changes are applied. The constraints of semantics integrity guaranteeing the consistence of the ontology along the evolution and their procedure of verification are also known to the system.

A parser executes each change operation while respecting predefined constraints. A report is produced at the end of the execution.

The operation class models all the operations which can be applied to the ontology. The diary of changes contains the operations of change which the ontology will undergo. The list of changes (changes log) can be provided directly to the system or edited by an Expert-user using an interface. The initial ontology which may eventually evolve is contained in an OWL file.

The operations contained in the log are then applied to the ontology. The integrity constraints are verified and a detailed report is produced.

Illustration through some screenshots
The example above illustrates how our solution is implemented: Given an initial ontology O1. A sequence of operations is proposed to be run (partial delete, reclassification ...). The initial ontology is shown in Figure 02. After execution of the «partial delete» operation the result is obtained in Figure 04. Figure 03 is an intermediate stage where the type of delete (total / partial delete) is supposed to be chosen. Figure 04 shows the results of the operation.
Validation
The validation of POOE has been carried out in two phases. We will start first with the internal validation and then move to the external validation.

On the internal side, the goal is to demonstrate that POOE allows for the supervision of the application of an ordered sequence of change operations to ontology. The supervision aims at allowing just the running of operations permitted while rejecting the others. On the external side, the goal is to demonstrate that the system helps to take into account the effects of the ontological change on its external artifacts. Specifically, we demonstrate that the system updates the semantic referencing of resources following an evolution of ontology. We have chosen the comparative method to validate our system internally. Externally, we carry out a formal demonstration of the validity of POOE.

Internally, we have chosen a sample test of two ontologies ($O_1$, $O_2$) and a change log. The choice of operations is deliberately done so that the evolution leads to abnormalities in some cases. Each change log is applied successively on the two ontologies to give evolved ontologies ($O'_1$, $O'_2$). The idea here is to apply the log by first using the POOE system and then applying the log in the editor Protégé. The results are then compared. We found out that in the first case, the expected results are obtained (valid transactions are executed and others are rejected with justifications / reasons). In the second case, the running log in Protege sometimes leads to abnormal situations for some.

Illustration
Given two ontologies $O_1$ and $O_2$ with $O_1$ previously defined (Figure 02) and $O_2$ defined as follows:
Change log Ch (op1, op2, op3) is made up of three operations defined as follows:

**op1(element)**: partial delete
Precondition: The element to be deleted should exist and be different from the root class.
Post-condition: the element exists and all its sub-classes are conserved.

**op2(element)**: addition
Precondition: The element should not exist in the ontology
Post-condition: the element is part of the ontology

**op3(element)**: total delete
Precondition: The deleted element should be present in the ontology
Post-condition: The element and its sub-classes no longer exist.

Operation op1(element) has been applied on the ontology O₁ (see Figure 02 and Figure 04) and on ontology O₂ (see Figure 06 and Figure 07) using our system. The same actions are repeated with Protege.

We have then used POOE and Protege to carry out operation op2 while adding the Postdoc content already present in ontology O₁. Later on, operation op2 has been used to add new concepts to O₁ with POOE and Protege respectively.

Finally operation op3(element) has been carried out to delete concepts such as MScStudent in O₁ and Microchip in O₂ with POOE and Protégé respectively.

The following table presents a recapitulation of those operations.

<table>
<thead>
<tr>
<th></th>
<th>O₁</th>
<th>O₂</th>
<th>POOE</th>
<th>Protege</th>
<th>POOE</th>
</tr>
</thead>
<tbody>
<tr>
<td>op₁</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>op₂</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Op₃</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table III shows that the results obtained with the POOE system correspond very well to what was forecast (which is represented by 1) whereas with Protege, we obtain false results in some case (which is represented by 0).

Externally, we have proven that our system helps to preserve the consistency of artifacts external to ontology after evolution. More specifically, we demonstrate that POOE helps to ensure the maintenance of semantic referencing of objects after ontology evolution. Given an ontology O and a set of resources R referenced in part or fully by O. We want to show that if O’ is the advanced ontology obtained after evolution of O under the supervision of POOE, then R can be referenced by O’ (partially or totally).

Demonstration
O = U {ei}, i = 1..n; R = U {rj}, j = 1..m
Given the function f such that f (ei, rj) = 1 if ei is included in the referencing of rj = 0 if otherwise.

The matrix M is obtained as:

\[
\begin{align*}
e_1 & \ 1 & \ 0 & \ 0 & \ \cdots & \ e_k & \ e_k & + & 1 & \ \cdots & \ e_n \\
r_1 & \ 1 & \ 0 & \ 0 & \ \cdots & \ 1 & \ 0 & \ \cdots & \ 1 \\
r_2 & \ 0 & \ 0 & \ 1 & \ \cdots & \ 1 & \ 1 & \ \cdots & \ 1 \\
\vdots & \ \vdots & \ \vdots & \ \vdots & \ \cdots & \ \vdots & \ \vdots & \ \cdots & \ \vdots \\
r_j & \ 1 & \ 1 & \ 1 & \ \cdots & \ 0 & \ 1 & \ \cdots & \ 1 \\
\vdots & \ \vdots & \ \vdots & \ \vdots & \ \cdots & \ \vdots & \ \vdots & \ \cdots & \ \vdots \\
r_n & \ 0 & \ 1 & \ 1 & \ \cdots & \ 0 & \ 0 & \ \cdots & \ 0
\end{align*}
\]

We are looking for a matrix M’ such that:

\[
\begin{align*}
e'_1 & \ e'_2 & \ \cdots & \ e'_k & \ \cdots & \ e'_n \\
r_1 & \\
r_2 & \\
\vdots & \ M' & \\
r_k & \\
\vdots & \\
r_n
\end{align*}
\]
Let’s prove that for every $rj$, $\exists e'i (i = 1..m2)$ and $h (e'i, rj) = 1$
In any $rj$, $\exists a set \{Ue'i \} i = 1..k$ to reference $rj$ (Assumption)
Given that the ontology $O'$ is obtained after evolution, let’s consider the various possible cases:
Case 1: Change due to operations of addition. The referencing is not affected and it is not necessary to upgrade.
Case 2: Change due to operations of modification. The referencing is not affected.
Case 3: Change due to operations of deleting. For each $e'i$ deleted, we look for the closest concept/element (semantic distance) of the ontology $O'$ [22]. This new element replaces the concept deleted in the references of the resource.
Case 4: Change due to Group, union or merge operations. The concepts deleted after these operations are replaced in the referencing by concepts resulting from these different operations.
Case 5: Change due to "set" operations.
Operations with "set" do not bring about the semantic referencing problems.
Illustration (on the example above):
Consider the $O_1$ ontology and semantic SEO "R". After execution of the removal operation, we get a new determination $R'$.
We should take note that the semantic referencing a resource consists of one or more UKI that each refer to a unique class belonging to an ontology (or version of ontology) clearly identified. In turn, each UKI has two components: a URI of an ontology (or version) and the name of a class belonging to this ontology.
$O_1$ being the initial ontology (see Figure 02). Let $R$ be a semantic SEO defined as follows:
$R = \text{ontology_example.org/\#Student}$
Let’s determine the associated matrix $M$.
The elements of ontology are:
$e_1 = \text{intersh}$ $e_2 = \text{PhDstudent}$ $e_3 = \text{Student}$ $e_4 = \text{MSCstudent}$
$e_5 = \text{AcademicStaff}$ $e_6 = \text{Postdoc}$ $e_7 = \text{Professor}$
$e_8 = \text{Teguia-Bertrand}$ $e_9 = \text{Ndonna-Yacynth}$ $e_{10} = \text{Mr}$
$\text{Tawamba}$ $e_{11} = \text{Dr-Batchakui}$
The following matrix $M$ is obtained,
\[
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\end{bmatrix}
\]
After the partial removal of the concept of "student", it no longer exists. The interpretation of SEO also becomes impossible. The new template to consider the following:
\[
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\end{bmatrix}
\]
Which leads to the determination of a new semantic SEO. We get the new semantic SEO:
$R' = \text{ontology_example.org/\#intersh}$
V. CONCLUSION AND FURTHER RESEARCH
We have developed an application module which makes it possible to follow and control the evolution of ontologies. Our solution is founded on the formal description of the semantic associated to each operation of change based on the Description Logics. We associate to each operation constraints to verify (for some, before, for others, after) and to ensure the consistency and the integrity of the final ontology. This application still integrates specific important operations for the creation of ontologies but particularly helps in the effective update of semantic referencing after the evolution of ontologies.
This work should be extended in order to allow our system to measure the quality of the final ontology. In the same manner, externally, our system should be able to support the effects of evolution on applications and ontologies that refer to the initial ontology. Also, much still has to be done in order to cope with the external validation of the evolution which is usage-context dependent. The key issue here is to look for a possible generic core that can ease the specification of context-dependant constraints of semantic integrity related to the ontology use in a specific domain. However, we have provided a framework where both internal and external validation of the evolution process can be managed. This is a contribution as such integrated solution does not exist. Our next step here will be to use the system in other use-cases where it could act as an active coach during a live ontology evolution (even building) process. More importantly, upgrading of POOE system to a real ontological engineering platform.

REFERENCES


