Modeling Changes for $\mathcal{SHOIN}(\mathcal{D})$ Ontologies

An Exhaustive Structural Model

Perrine Pittet, Christophe Cruz, Christophe Nicolle
Le2i, UMR CNRS 6306
University of Burgundy
Dijon, France
{perrine.pittet, christophe.cruz, cnicolle}@u-bourgogne.fr

Abstract— **Ontology development starts with a rigorous ontological analysis that provides a conceptualization of the domain to model agreed by the community. An ontology, specified in a formal language, approximates the intended models of this conceptualization. It needs then to be revised and refined until an ontological commitment is found. Also ulterior updates, responding to changes in the domain and/or the conceptualization, are expected to occur throughout the ontology life cycle. To handle a consistent application of changes, a couple of ontology evolution methodologies have been proposed. Maintaining the structural consistency is one of the ontology evolution criteria. It implies modelling changes with respect to how the constructs of the ontology language are used. However there is no ontology model, among those proposed, that allows to exhaustively describe changes and their impact for languages based on SHOIN(D) description logic. To bridge this gap, this paper presents a complete structural ontology model suited for change modelling on SHOIN(D) ontologies. The application of this model is illustrated along the paper through the description of an ontology example inspired by the UOBM ontology benchmark and its evolution.**

Keywords- SHOIN(D) Description Logic; Change Modelling, Ontology Evolution; Ontology Model; Structural Consistency; OWL DL.

I. INTRODUCTION

In recent years, building ontologies are gaining ground to provide to the Semantic Web clear semantics in an agreed, consistent and shared encodings. Actually, ontologies make possible to application, enterprise, and community boundaries of any domain to bridge the gap of semantic heterogeneity. Ontologies development, to be correctly achieved, requires a dynamic and incremental process [1]. It starts with a rigorous ontological analysis [2] that provides a conceptualization of the domain to model agreed by the community. The ontology, specified in a formal language, approximates the intended models of the conceptualization [3]: the closer it is the better it is. The ontology needs to be revised and refined until an ontological commitment is found. Ulterior updates of the ontology, addressed by ontology evolution, aim at responding to changes in the domain and/or the conceptualization [4]. Changes are consequently inherent in the ontology life cycle.

Reference [5] defines an ontology change as an action on an ontology resulting in an ontology that is different from the original version. To manage the lifecycle of ontologies and to ensure structural and logical consistent updates with regards to changes, a couple of ontology evolution methodologies have been proposed like [[6], [7], [8], [9], [10]]. Among them, the AIFB methodology [8], which is one of the most popular, identifies 6 phases to ensure the quality of the ontology evolution process: detection, representation, semantics, implementation, propagation and validation. Among those phases, two are of utmost importance to correctly model changes and their impact: the change representation phase, which consists in the translation of these changes into formal ontological operations, and the change semantics phase, which clearly defines their impact on the ontology by decomposing each operation into additions and/or deletions of atomic elements of the ontology. These two phases aim at ensuring a non-ambiguous application of changes to clearly envision their consequences on the ontology consistency.

According to [11], a consistent ontology is one that satisfies all invariants of the ontology model. Invariants are constraints that must hold in every quiescent state of an ontology. Structure consistency is one of these constraints. It ensures that the ontology obeys the constraints of the ontology language with respect to how the constructs of the ontology language are used. Modelling changes then implies having an exhaustive and non-ambiguous definition of the ontology model according to its language, so that each element of the ontology impacted by changes can be formally described.

This paper focuses on the $\mathcal{SHOIN}(\mathcal{D})$ level of expressivity, on which the ontological language OWL DL is based [13]. It first presents a model that exhaustively describes the structural constraints of a $\mathcal{SHOIN}(\mathcal{D})$ ontology defined by the constructors, axioms and facts of the description logic. It then describes a list of basic changes, constrained by this structural model to avoid performing structural inconsistent updates on the ontology. It subsequently explains how to model complex changes, composed of basic changes of this list, which are safe for the structure consistency of the ontology. Additionally each application of a change is semantically defined as an addition or a deletion of a basic or complex change that corresponds to additions or deletions of identified elements of the ontology model. This improves the evaluation of the impact of the application on a $\mathcal{SHOIN}(\mathcal{D})$ ontology. The application of this model is illustrated along the paper through the description of an ontology example, inspired by the

UOBM Ontology Benchmark for OWL DL ontologies (Ma, 2006), and its evolution.

II. $\mathcal{SHOIN}(\mathcal{D})$ Ontology Model

A. A Structural Model

In order to formalize our framework the Karlsruhe Ontology Model [14] is used and extended to cover the whole $\mathcal{SHOIN}(\mathcal{D})$ constructors.

From a mathematical point of view, an ontology can be defined as a structure. Formally, a structure is a triple $A=(S, \sigma, F)$ consisting of an underlying set S, a signature σ, and an interpretation function F that indicates how the signature is to be interpreted on S.

Definition 1: $\mathcal{SHOIN}(\mathcal{D})$ Ontology Model.

A $\mathcal{SHOIN}(\mathcal{D})$ ontology is a structure $O=(SO ,\sigma O ,FO)$ consisting of:

- The underlying set SO containing:
  - Six disjoint sets sC, sT, sR, sA, sI, sV, sK_R and sK_A called concepts, datatypes, relations, attributes, instances, data values, relation characteristics (among Symmetric, Functional, Inverse Functional, Transitive) and attribute characteristics (Functional),
  - Four partial orders $\leq_C$, $\leq_T$, $\leq_R$ and $\leq_A$, respectively on sC called concept hierarchy or taxonomy, on sT called type hierarchy, on sR called relation hierarchy and on sA called attribute hierarchy,

such that $SO :=\{(sC, \leq_C),(sT, \leq_T),(sR, \leq_R),(sA, \leq_A), sI, sV, sK_R, sK_A,\}$,

- The signature σO containing two functions $\sigma_R:sR\rightarrow sC^2$ called relation signature and $\sigma_A:sA\rightarrow sC \times sT$ called attribute signature, such that $\sigma O :=\{\sigma_R, \sigma_A\}$,
- The interpretation function FO containing:
  - A function $\iota_C:sC\rightarrow 2^{sI}$ called concept instantiation,
  - A function $\iota_T:sA\rightarrow 2^{sV}$ called data type instantiation,
  - A function $\iota_R:sC\rightarrow 2^{sI\times sI}$ called relation instantiation,
  - A function $\iota_A:sC\rightarrow 2^{sI\times sV}$ called attribute instantiation,
  - A function $\kappa_R:sR\rightarrow 2^{sKR}$ called relation characterization,
  - A function $\kappa_A:sA\rightarrow 2^{sKA}$ called attribute characterization,
  - A function $\varepsilon_C:sC\rightarrow 2^{sC}$ called concept equivalence,
  - A function $\varepsilon_R:sR\rightarrow 2^{sR}$ called relation equivalence,
  - A function $\varepsilon_A:sA\rightarrow 2^{sA}$ called attribute equivalence,
  - A function $\varepsilon_I:sI\rightarrow 2^{sI}$ called instance equivalence,
  - A function $\delta_C:sC\rightarrow 2^{sC}$ called concept disjunction,
  - A function $\delta_I:sI\rightarrow 2^{sI}$ called instance differentiation,
  - A function $\text{-}_C:sC\rightarrow 2^{sC}$ called concept complement specification,
  - A function $\text{-}_R:sR\rightarrow 2^{sR}$ called relation inverse specification,
  - A function $maxCardR:sR\rightarrow N$ called relation maximal cardinality restriction,
  - A function $minCardR:sR\rightarrow N$ called relation minimal cardinality restriction,
  - A function $\sqcap_C:sC\rightarrow 2^{sC}$ called concept intersection,
  - A function $\sqcup_C:sC\rightarrow 2^{sC}$ called concept union,
  - A function $\sqcup_{iC}:sI\rightarrow 2^{sC}$ called concept union enumeration,
  - A function $\sqcup_V:sV\rightarrow 2^{sC}$ called data value union,
  - A function $\sqcap_{iC}:sC\rightarrow 2^{sI}$ called concept enumeration,
  - A function $\rho_{\exists R} :sR\rightarrow 2^{sC}$ called relation existential restriction
  - A function $\rho_{\forall R} :sR\rightarrow 2^{sC}$ called relation universal restriction,
  - A function $\rho_R :sR\rightarrow 2^{sI}$ called relation value restriction,
  - A function $\rho_{\exists A} :sA\rightarrow 2^{sT}$ called attribute existential restriction
  - A function $\rho_{\forall A} :sA\rightarrow 2^{sT}$ called attribute universal restriction,
  - A function $\rho_A :sA\rightarrow 2^{sV}$ called attribute value restriction,

such that $FO:=\{\iota_C, \iota_T, \iota_R,\iota_A, \kappa_R, \kappa_A, \varepsilon_C, \varepsilon_R, \varepsilon_A, \varepsilon_I, \delta_C, \delta_I, \text{-}_C, \text{-}_R, maxCard_R, minCard_R, \sqcap_C, \sqcup, \sqcup_{iC}, \sqcup_V, \sqcap_{iC}, \rho_{\exists R}, \rho_{\forall R}, \rho_R, \rho_{\exists A}, \rho_{\forall A}, \rho_A)$.

Table 1 shows the correspondence of our model with $\mathcal{SHOIN}(\mathcal{D})$ constructors, axioms and facts. The first column gives the description logic syntax for the construction of axioms and facts of a knowledge base in $\mathcal{SHOIN}(\mathcal{D})$ The letters C, T, R, A, I and V represent, respectively, names for classes (concepts), data types, relations, attributes, instances and data values. The second column gives the correspondence with the $\mathcal{SHOIN}(\mathcal{D})$ Ontology Model defined upper. The third column is described further.

	$\mathcal{SHOIN}(\mathcal{D})$ DL Syntax	$\mathcal{SHOIN}(\mathcal{D})$ Ontology Model	$\mathcal{SHOIN}(\mathcal{D})$-based Changes Abstract Syntax
Descriptions	C	sC	Class(Class)
	$\bot$	*bConcept*	
	$\top$	*tConcept*	
	$C_1 \sqcap \ldots \sqcap C_n$	$\sqcap_C$	IntersectionOf($\text{Class}_1,\ldots,\text{Class}_n$)
	$C_1 \sqcup \ldots \sqcup C_n$	$\sqcup_C$	**UnionOf($\text{Class}_1,\ldots,\text{Class}_n$)**
	$\neg C$	$-_C$	**ComplementOf(Class)**
	$\{I_1\} \sqcup \ldots \sqcup \{I_n\}$	$\sqcup_{IC}$	**OneOf(Class, $\text{Instance}_1,\ldots,\text{Instance}_n$)**
	$\exists R.C$	$\rho_{\exists R}$	SomeValuesFrom(ObjectProperty, Class)
	$\forall R.C$	$\rho_{\forall R}$	AllValuesFrom(ObjectProperty, Class)
	$R : I$	$\rho_{R:}$	HasValue(ObjectProperty, Instance)
	$\geq n\ R$	$maxCard_R$	MinCardinalityProperty(ObjectProperty, n)
	$\leq n\ R$	$minCard_R$	MaxCardinalityProperty(ObjectProperty, n)
	$\exists A.T$	$\rho_{\exists A}$	SomeValuesFrom(DatatypeProperty, Datatype)
	$\forall A.T$	$\rho_{\forall A}$	AllValuesFrom(DatatypeProperty, Datatype)
	$A : V$	$\rho_{A:}$	HasValue(DatatypeProperty, Datavalue)
	T	sT	Datatype(Datatype)
	$\{V_1\} \sqcup \ldots \sqcup \{V_n\}$	$\sqcup_V$	**OneOf($\text{Datavalue}_1,\ldots,\text{Datavalue}_n$)**
	R	sR	ObjectProperty(ObjectProperty)
	R^-		
	A	sA	DatatypeProperty(DatatypeProperty)
	I	sI	Instance(Instance)
	V	sV	Datavalue(Datavalue)
Axioms and Facts	$C \sqsubseteq C_1 \sqcap \ldots \sqcap C_n$	$C \sqsubseteq \sqcap_{Ci}$	**IntersectionClass(Class, ($\text{Class}_1,\ldots\text{Class}_n$))**
	$C \equiv C_1 \sqcap \ldots \sqcap C_n$	$\sqcap_C$	**IntersectionClass(Class, ($\text{Class}_1,\ldots\text{Class}_n$))**
	$C \equiv \{I_1\} \sqcap \ldots \sqcap \{I_n\}$	$\sqcap_{iC}$	**EnumeratedClass(Class, ($\text{Instance}_1,\ldots\text{Instance}_n$))**
	$C_1 \sqsubseteq C_2$	$\leq_C$	SubClassOf(Class_1, Class_2)
	$C_1 \equiv \ldots \equiv C_n$	ε_C	**EquivalentClass($\text{Class}_1,\ldots\text{Class}_n$)**
	$\bot \equiv C_1 \sqcap C_2$	δ_C	DisjointClass(Class_1, Class_2)
	$T_1 \sqsubseteq T_2$	$\leq_T$	SubDatatypeOf(Datatype_1, Datatype_2)
	$V \in T_i$	ι_T	InstancesOfDatatype(Datavalue, Datatype)
	$R \sqsubseteq R_i$	R	ObjectProperty(ObjectProperty)
	$\geq 1R \sqsubseteq C_i$	σ_R	DomainProperty(ObjectProperty, Class)
	$\top \sqsubseteq \forall R.C_i$		RangeProperty(ObjectProperty, Class)
	$R \equiv R_0^-$	$-_R$	InverseOf(ObjectProperty_1 ObjectProperty_2)
	$R \equiv R^-$	κ_R	SymmetricProperty(ObjectProperty)
	$\top \sqsubseteq \leq 1R$		FunctionalProperty(ObjectProperty)
	$\top \sqsubseteq \leq 1R^-$		InverseFunctionalProperty(ObjectProperty)
	$Tr(R)$		TransitiveProperty(ObjectProperty)
	$R_1 \sqsubseteq R_2$	$\leq_R$	InheritanceObjectPropertyLink(ObjectProperty_1, ObjectProperty_2)
	$R_1 \equiv \ldots \equiv R_n$	ε_R	**EquivalentProperty($\text{ObjectProperty}_1,\ldots,\text{ObjectProperty}_n$)**
	$A \sqsubseteq A_i$	A	DatatypeProperty(DatatypeProperty)
	$\geq 1A \sqsubseteq C_i$	σ_A	DomainProperty(DatatypeProperty Class)
	$\top \sqsubseteq A.T_i$		RangeProperty(DatatypeProperty, Datatype)
	$\top \sqsubseteq \leq 1A$	κ_A	FunctionalProperty(DatatypeProperty)
		sKA	
	$A_1 \sqsubseteq A_2$	$\leq_A$	InheritanceDatatypePropertyLink($\text{DatatypeProperty}_1$, $\text{DatatypeProperty}_2$)
	$A_1 \equiv \ldots \equiv A_n$	ε_A	**EquivalentProperty($\text{DatatypeProperty}_1,\ldots,\text{DatatypeProperty}_n$)**
	$I \in C_i$	ι_C	InstancesOf(Instance, Class)
	$\{I, I_i\} \in R_i$	ι_R	InstancesOfObjectProperty(Instance, Instance_1, ObjectProperty)
	$\{I, V_i\} \in A_i$	ι_A	InstanceOfDatatypeProperty(Instance, Datavalue, DatatypeProperty)
	$\{I_1\} \equiv \ldots \equiv \{I_n\}$	ε_I	**SameAs($\text{Instance}_1,\ldots,\text{Instance}_n$)**
	$\{I_i\} \sqsubseteq \neg\{I_j\}$, $i \neq j$	δ_I	**DifferentFrom($\text{Instance}_1,\ldots,\text{Instance}_n$)**

Table 1. Correspondence between $\mathcal{SHOIN}(\mathcal{D})$ Descriptions, Axioms and Facts, the $\mathcal{SHOIN}(\mathcal{D})$ Ontology Model and the $\mathcal{SHOIN}(\mathcal{D})$-based List of Basic Changes.

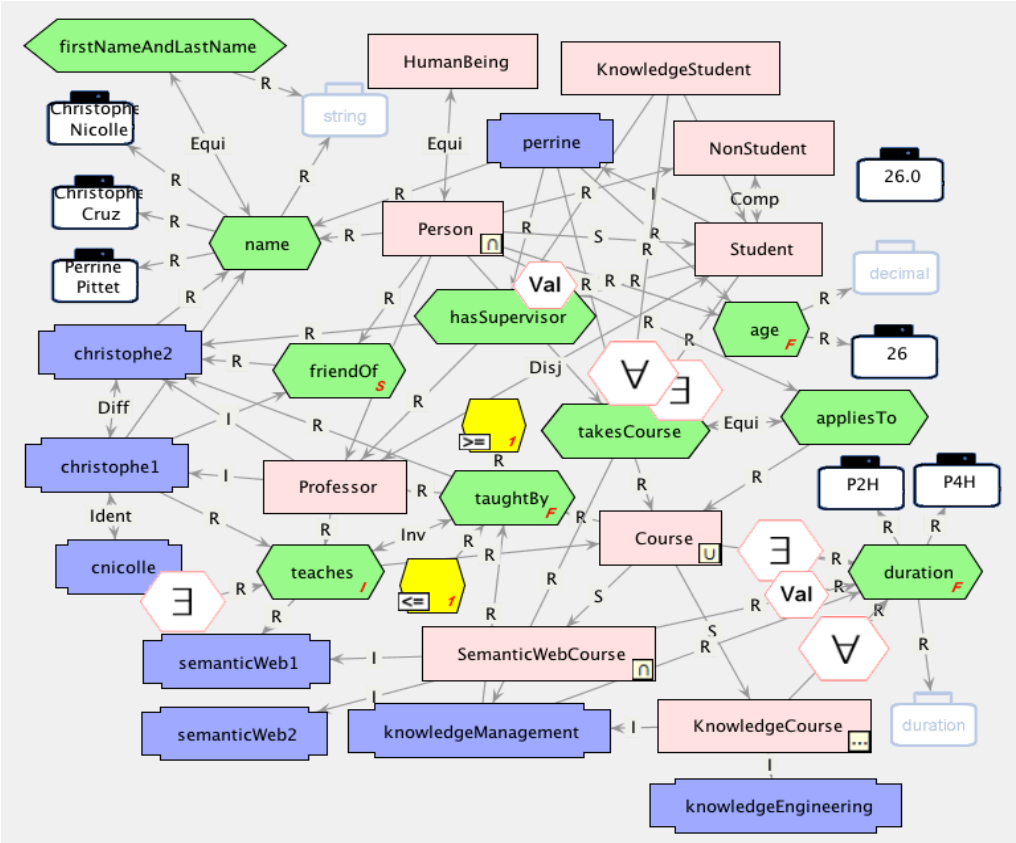

Fig. 1. Graphical Representation of the Ontology *O* with G-MOT.

We illustrate our model definition through an example inspired by the UOBM Ontology Benchmark [15]. The ontology O describes the relations between students taking courses, supervised by professors teaching courses. We have added instances and datavalues in order to show a complete illustration of our model:

- sC={TConcept, Person, HumanBeing, Student, Professor, Course, KnowledgeCourse, SemanticWebCourse, KnowledgeStudent, NonStudent},
- sT={tType, xs:decimal, xs:string, xs:duration},
- $\leq_C$ ={(TConcept, Person), (TConcept, HumanBeing), (Person, Student), (Person, Professor), (TConcept, Course), (Course, KnowledgeCourse), (Course, SemanticWebCourse), (Student, KnowledgeStudent), (Person, NonStudent)}
- $\leq_T$ ={(tType, xs:decimal), (tType, xs:string), (tType, xs:duration)},
- sR={tRelation, friendOf, taughtBy, teaches, takesCourse, appliesTo, hasSupervisor},
- sA={ name, firstNameAndLastName, age, duration},
- σ_R={(takesCourse, (Person, Course)), (friendOf, (Person, Person)), (taughtBy, (Course, Professor)), (teaches, (Professor, Course)), (hasSupervisor, (Student, Professor))}
- σ_A={(name, (Person, xs:string)), (age, (Person, xs:decimal)), (duration, (Course, xs:duration))}
- $\leq_R$={(tRelation, friendOf), (tRelation, taughtBy), (tRelation, takesCourse), (tRelation, appliesTo), (tRelation, hasSupervisor)}
- $\leq_A$={(tAttribute, name), (tAttribute, age), (tAttribute, duration)},
- sI={christophe1, cnicolle, christophe2, perrine, knowledgeManagement, knowledgeEngineering, semanticWeb1, semanticWeb2},
- sV={“Christophe Nicolle”, “Christophe Cruz”, “Perrine Pittet”, 26, 26.0, P2H, P4H},
- ι_C={(Professor,{christophe1, christophe2}), (Student,{perrine}), (Course, {knowledgeManagement, knowledgeEngineering, semanticWeb1, semanticWeb2})},
- ι_T={(xs:decimal,{26}), (xs:string, {“Christophe Nicolle”, “Christophe Cruz”, “Perrine Pittet”}), (xs:duration,{P2H, P4H})},
- ι_R={(friendOf,(christophe1, christophe2)), (taughtBy(knowledgeManagement, christophe2)), (teaches, (christophe1, semanticWeb1)), (takesCourse, (perrine, knowledgeManagement)), hasSupervisor(perrine, christophe2)}}
- ι_A={(age,(perrine, 26)), (name,(christophe1, “Christophe Nicolle”)), (name,(christophe2, “Christophe Cruz)), (name,(perrine, “Perrine Pittet”)), (duration, (knowledgeManagement, P2H))},

- sK_R={Symmetric, Functional, InverseFunctional},
- sK_A={Functional},
- κ_R={(friendOf, Symmetric), (taughtBy,{Functional}), (teaches,{InverseFunctional)},
- κ_A={(age, {Functional}), (duration,{Functional})},
- ε_C={(Person, {HumanBeing})},
- ε_R ={(takesCourse, {appliesTo })},
- ε_A={(hasName, {hasFirstNameAndLastName})},
- ε_I={(christophe1, {cnicolle})},
- δ_C={(Student,{Professor})},
- δ_I={(christophe1,{christophe2})},
- $-_C$={(Student,{NonStudent })},
- $-_R$={(teaches,{isTaughtBy})},
- maxCardR={(isTaughtBy, 1)},
- minCardR={(isTaughBy, 1)},
- $\sqcap_C$={(Person, {Student, NonStudent})},
- $\sqcup_C$={(Course, {KnowledgeCourse, SemanticWebCourse})},
- $\sqcup_{iC}$={(KnowledgeCourse, {knowledgeManagement, knowledgeEngineering})},
- $\sqcup_V$={(26, {26.0, 26.00})},
- $\sqcap_{iC}$={(SemanticWebCourse,{semanticWeb1, semanticWeb2})},
- $\rho_{\exists R}$= {(Professor, teaches, Course), (Student, takesCourse, Course)},
- $\rho_{\forall R}$ ={(KnowledgeStudent, takesCourse, KnowledgeCourse)}
- ρ_R ={(KnowledgeStudent, hasSupervisor, christophe2)},
- $\rho_{\exists A}$ ={(Course, duration, xsd:duration)},
- $\rho_{\forall A}$={(KnowledgeCourse, duration, {P2H, P4H})},
- ρ_A={(SemanticWebCourse, duration, P2H)},

Figure 1 shows a graphical representation of the ontology O realized with the G-MOT Ontology Editor [15].

B. $\mathcal{SHOIN}(\mathcal{D})$ Change Modeling

In order to model changes, we give the five definitions below.

Definition 2: Change. A change ω is the application of a modification on an ontology O, that potentially affects one or more elements of its structure as defined by the $\mathcal{SHOIN}(\mathcal{D})$ Ontology Model.

Definition 3: Log of Changes. Given an ontology O a log of changes, noted log_i, is defined by an ordered set of changes (simple and complex) $<\omega_1, ..., \omega_n>$ that applied to O results in O.

Like in [5], 2 change types are distinguished: basic and complex.

Definition 4: Basic Change. A basic change on an ontology O is a function $\omega_B: sK \rightarrow 2^O$ with $sK:=\{sC \cup sI \cup sR \cup sA\}$ that corresponds to an addition, a removal of a modification of one element $\in O$.

Definition 5: Complex Change. A complex change on an ontology O is a disjoint union of basic changes. It is a function $\omega_C: nsK \rightarrow 2^O$ such that $\omega_C := \omega_{B1} + ... + \omega_{Bn}$.

The application of a change on an ontology, basic or complex, can be an addition or a deletion. It is traced as such in the log of changes.

Definition 6: Addition of a Change. The addition of a change ω_i traced in the log of changes log_i, noted $log_i + \{\omega_i\}$, is defined by the disjoint union between the two disjoint sets log_i and $\{\omega_i\}$.

Definition 7: Deletion of a Change. The deletion of a change ω_i traced in the log of changes log_i, noted $log_i - \{\omega_i\}$, is defined by the set-theoretic complement such that $log_i - \{\omega_i\} = \{x \in log_i \mid x \notin \{\omega_i\}\}$.

C. Basic Changes Modeling

To produce the list of basic change operations on $\mathcal{SHOIN}(\mathcal{D})$ ontologies, the $\mathcal{SHOIN}(\mathcal{D})$ Ontology Model is exploited as described in Table 1. The third column lists the 47 operators representing basic changes, which, if applied on the ontology, affect the corresponding $\mathcal{SHOIN}(\mathcal{D})$ model element. According to our model, every basic change can be declined as an addition or a deletion of an element of the underlying set, the signature or the interpretation function.

Table 2 below represents the impact corresponding to the change entitled *InstancesOfObjectProperty* in terms of addition or deletion on the ontology model. If applied as an addition, this change corresponds to the addition of an element ι_{Ri} to the set of relation instantiations ι_R as described in Table 1. Inversely, if applied as a deletion, this change corresponds to the deletion of an element ι_{Ri}.

Example: Addition of the basic change InstancesOfObjectProperty.

Given the previous example ontology O, the evolution of O into O_{new} with the addition of the relation instantiation ι_{Ri} = *hasSupervisor(perrine christophe1)* w.r.t. our model, represented by the change ω_1^B =*InstancesOfObjectProperty(perrine, christophe1, hasSupervisor)* can be formalized:

$$\iota_{R\,new} = \{ \iota_R + \{(hasSupervisor(perrine, christophe1)\}\}$$

13 of the 47 changes (appearing in bold on Table 1) can potentially turn into complex changes if applied on more than 2 elements of the ontology. For instance, the change *addIntersectionOf*, which syntax is *addIntersectionOf*($Class_1,...,Class_n$) is considered basic if used to add the intersection of 2 classes (ex :*addIntersectionOf*($Class_1$, $Class_2$) but composed if adding the intersection of more than 2 classes (ex :*addIntersectionOf*($Class_1$, $Class_2$, $Class_3$) . Actually, in the second case, it is composed of several basic changes (ex: *addIntersectionOf*(*addIntersectionOf*($Class_1$, $Class_2$), $Class_3$)).

Identify applicable sponsor/s here. *(sponsors)*

$\mathcal{SHOIN}(\mathcal{D})$-based Change Abstract Syntax	$\mathcal{SHOIN}(\mathcal{D})$ Ontology Impact	
	Addition	Deletion
InstancesOfObjectProperty(Instance, Instance1, ObjectProperty)	$\iota_R + \iota_{Ri}$	$\iota_R - \iota_{Ri}$

Table 2. Modelling of the Impact of a Basic Change Addition or Deletion on a $\mathcal{SHOIN}(\mathcal{D})$ Ontology.

D. Complex Changes Modeling

More generally, an infinite set of complex changes can be generated from the aggregation of basic changes [16]. Their pertinence depends on the need of particular changes implied by particular uses. For example, the renaming of a concept is often used in collaborative development of an ontology to reach a consensus but can be unused in other contexts. For this reason, our model natively provides the limited set of 47 basic changes but, depending on change modelling needs, gives the opportunity to build complex changes from these basic changes.

Below is an example of the "renaming concept" complex change definition. In this example is considered the set-theory renaming not the lexical one.

Example: renameClass Complex Change definition.

Renaming a concept C in a concept C_{new} is a complex change called here *renameClass*, which implies the creation of a new concept C_{new}, the copy of the concept descriptions of C (from its related ontology sets, signatures and interpretations) to C_{new}, then the deletion these descriptions of C followed by the deletion of C itself. Below is the abstract syntax of such complex change:

renameClass(Class2, Class1)=<
Class(Class2)
+IntersectionOf(Class2, Class1.getIntersectionOf())
+UnionOf(Class2, Class1.getUnionOf())
+ComplementOf(Class2, Class1.getComplementOf())
+SomeValuesFrom(Class1.getSomeValuesFromObjectProperty(), Class2)
+AllValuesFrom(Class1.getAllValuesFromObjectProperty(), Class2)
+EquivalentClass(Class2, Class1.getEquivalentClass())
+DisjointClass(Class2, Class1.getDisjointClass())
+IntersectionClass(Class2, Class1.getIntersectionClass())
+EnumerationClass(Class2, Class1.getIntersectionClass())
+OneOf (Class2, Class1.getOneOf())
+SubClassOf (Class2, Class1.getSubClassOf())
+SuperClassOf (Class2, Class1.getSuperClassOf())
+DomainProperty(Class1.getObjectPropertyDomainOf(), Class2)
+RangeProperty(Class1.getObjectPropertyRangeOf(), Class2)
+DomainProperty(Class1.getDatatypePropertyDomainOf(), Class2)
+InstancesOf(Class2, Class1.getInstancesOf())
−IntersectionOf(Class1, Class1.getIntersectionOf())
−UnionOf(Class1, Class1.getUnionOf())
−ComplementOf(Class1, Class1.getComplementOf())
−SomeValuesFrom(Class1.getSomeValuesFromObjectProperty(), Class1)
−AllValuesFrom(Class1.getAllValuesFromObjectProperty(), Class1)
−EquivalentClass(Class1, Class1.getEquivalentClass())
−DisjointClass(Class1, Class1.getDisjointClass())
−IntersectionClass(Class1, Class1.getIntersectionClass())
−EnumerationClass(Class1, Class1.getIntersectionClass())
−OneOf(Class1, Class1.getOneOf())
−SubClassOf (Class1, Class1.getSubClassOf())
−SuperClassOf (Class1, Class1.getSuperClassOf())
−DomainProperty(Class1.getObjectPropertyDomainOf(), Class1)
−RangeProperty(Class1.getObjectPropertyRangeOf(), Class1)
−DomainProperty(Class1.getDatatypePropertyDomainOf(), Class1)
−InstancesOf(Class1, Class1.getInstancesOf())
−Class(Class1)>

Like any basic change, a complex change has a corresponding impact on the ontology definition in terms of additions and deletions of elements of the underlying sets, signatures and interpretations of the ontology definition. As a complex change is de-fined as a disjoint union of basic changes, then its impact on the ontology definition is the set of the additions and deletions corresponding to each basic change implied.

III. Discussion and Conclusion

It has long been realized that the web could benefit from having its content understandable and available in a machine processable form [17]. This can be achieved if the ontology is specified in a language having a formal logic based-semantics equipped with decision procedures designed for automated reasoning. That is why description logics have been introduced as a development basis of a number of ontological languages. Among them, OWL was heavily influenced by Description Logic research. The creation of the OWL DL sub-language (derived from the DL $\mathcal{SHOIN}(\mathcal{D})$) was motivated by the need to unambiguously represent information in a strongly expressive language, able to retain computational completeness, decidability and the availability of practical reasoning algorithms. Many works on ontology evolution consider the language OWL DL [[10]; [4]; [16]]. However they do not provide an OWL DL ontology model suited for their purposes. Reference [8] derives a set of ontology changes for the KAON1 ontology language. The author specifies fine-grained changes according to the KAON1 model that can be performed during ontology evolution. Similarly we have proposed a structural ontology model for change management dedicated to $\mathcal{SHOIN}(\mathcal{D})$. Our model aims at facilitating the modelling of basic and complex changes. It aims at contributing to the maintenance of the ontology structural consistency by clearly defining each change impact on the structure of the ontology. This model is the structural basis of a change management methodology called OntoVersionGraph [18]. To ensure a complete consistent evolution of the ontology before its validation, it is used in conjunction with a priori logical inconsistency identification methodology called CLOCk [19], based on ontology design patterns and model-checking.

REFERENCES

[1] Djedidi, R., & Aufaure, M. A. (2008.). - Change Management Patternsfor Ontology Evolution Process –. *IWOD at ISWC 2008*. Karlsruhe.

[2] Guarino, N. &. (1995). Formal ontology, conceptual analysis and knowledge representation. *International Journal of Human Computer Studies , 43(5)*, 625-640.

[3] Gruber, T. R. (1993). A translation approach to portable ontology specifications. *Knowledge acquisition , 5(2)*, 199-220.

[4] Flouris, G. M. (2008). Ontology Change: Classification & Survey. (C. U. Press, Ed.) *he Knowledge Engineering Review, , 23(2)*, pp. 117-152.

[5] Klein, M. C. (2004). Change management for distributed ontologies

[6] Noy, N. F. (2004). *Ontology Evolution: Not the Same as Schema Evolution* . Stanford Medical Informatics, Stanford University.

[7] Pinto, H. S. (2004). DILIGENT: Towards a fine-grained methodology for Distributed, Loosely-controlled and evolving Engineering of oNTologies. *ECAI , 16*, 393.

[8] Stojanovic, L. (2004). Methods and tools for ontology evolution.

[9] Djedidi, R. (2009). Approche d'évolution d'ontologie guidée par des patrons de gestion de changement.

[10] Jaziri, W. (2009). A methodology for ontology evolution and versioning. In IEEE. (Ed.), *Advances in Semantic Processing, SEMAPRO'09*, (pp. 15-21).

[11] Stojanovic, L. &. (2002). Ontology evolution within ontology editors. OntoWeb-SIG3 Workshop at the 13th International Conference on Knowledge Engineering and Knowledge Management (pp. 53-62). Siguenza: CEUR-WS.org.

[12] Smith, M. K., W. C. (n.d.). *OWL Web Ontology Language Guide*. Retrieved from w3.org: http://www.w3.org/TR/owl-guide/

[13] Horrocks, I. &.-S. (2003). Reducing OWL entailment to description logic satisfiability. *The Semantic Web - ISWC 2003* (pp. 17-29). Springer Berlin Heidelberg.

[14] Ehrig, M. H. (2004). Similarity for ontologies-a comprehensive framework. Workshop Enterprise Modelling and Ontology: Ingredients for Interoperability, at PAKM.

[15] Paquette, G. & Magnan, F. (2008). *An executable Model for Virtual Campus Environments.* Handbook on Information Technologies for Education and Training (pp. 363-403).

[16] Plessers, P. D. (2005). Ontology Change Detection using a Version Log. In Springer-Verlag (Ed.), *4th International Semantic Web Conference* (pp. 578-592). Galway, Ireland: Yolanda Gil, Enrico Motta, V.Richard Benjamins, Mark A. Musen.

[17] Horrocks, I. (2005). Owl: A description logic based ontology language. *Logic Programming.* , 1-4.

[18] Pittet, P. N. (2012). Guidelines for a Dynamic Ontology-Integrating Tools of Evolution and Versioning in Ontology. . arXiv.

[19] Gueffaz, M. P. (2012). Inconsistency Identification In Dynamic Ontologies Based On Model Checking. *INSTICC, ACM SIGMIS*, (pp. 418-421.)