SEMANTICS BASED RECONCILIATON FOR COLLABORATIVE ONTOLOGY EVOLUTION

Georgios M. Santipantakis and George A. Vouros
Dept. of Information and Communication Systems Eng, University of the Aegean, Karlovassi, 83200, Samos, Greece

Keywords: Ontology evolution, Collaborative ontology engineering, Reconciliation, Validity rules.

Abstract: The objective of this paper is to present the SR-COE ("Semantics-based Reconciliation in Collaborative Ontology Development and Evolution") system for supporting knowledge workers/engineers to the synchronous and asynchronous collaborative ontology development and evolution: This system provides the necessary generic infrastructure for enhancing the deployment of any ontology development tool in distributed settings with concurrent workers, and for applying any collaborative ontology engineering method effectively. SR-COE exploits the *semantics of the modification actions* performed from the different, *concurrent* contributing parties so as to *actively* support them to reach mutual agreed ontologies.

1 INTRODUCTION

Exploitation of knowledge represented by means of ontologies aims to facilitate the knowledge management processes within and across organizations. To obtain a global view of a domain, often people need to express their perspectives, in a collaborative manner. However, it is likely to happen that different persons provide conflicting specifications for ontology elements. It is important to locate the implied conflicts and provide support for these persons to resolve them towards reaching an agreement. Aiming to *actively support* knowledge workers and engineers to shape a commonly accepted conceptualization of a domain, this paper presents the "Semantics-based Reconciliation for Collaborative Ontology Development and Evolution" (SR-COE) system: SR-COE exploits the semantics of the modification actions performed by knowledge engineers and workers, who act concurrently on a single ontology. The system detects actions with conflicting effects and reconciles them, proposing alternative and conflicts-free ontology versions. Semantics are being captured by means of constraints/dependencies between the effects of modification actions.

2 RELATED WORK AND MOTIVATION

Methodologies towards the collaborative development of ontologies propose concrete methodological steps for facilitating collaboration and reaching consensus. A wide range of approaches is available, featuring techniques for locking ontologies' versions (Farquhar et al, 1996), centrally accessible shared spaces (Kotis, Vouros, 2006), and/or require collaborative parties to provide arguments on the developed versions via participating in argumentation dialogues (Tempich et al 2005).

Several tools have been presented in the collaborative ontology development domain (Sure et al, 2002), (Arprez et al, 2001), (Bozsak et al, 2002), (Domingue, 1998), (Ceusters et al, 2001), (Hotho et al, 2006), (Tummarello et al 2006), (Sunagawa et al. 2003), (Auer et al, 2006) (Tudorache et al, 2008). However, to the best of our knowledge, there is not any tool that allows collaborative parties to simultaneously and jointly modify the same ontology version *without any restriction* (as far as the creation and access to any version is concerned), *actively* supporting people to *locate and reconcile conflicting* specifications and views.

To facilitate collaborators' awareness on the status of the evolving ontology, some of the methodologies/tools focus on providing a central repository. Different versions of the same ontology can be stored in this repository, possibly escorted

M. Santipantakis G. and A. Vouros G. (2009).
SEMANTICS BASED RECONCILIATON FOR COLLABORATIVE ONTOLOGY EVOLUTION .
In *Proceedings of the International Conference on Knowledge Engineering and Ontology Development*, pages 153-158
DOI: 10.5220/0002290501530158

with arguments concerning changes made. Locking and shared-space methods have inherent limitations: (a) They do not conserve against conflicts that may appear between versions, (b) they confine *real* collaboration, as each user modifying an ontology is the only one connected to it, and (c) they do not actively support collaborative parties to inspect all possible conflicts and reconcile different perspectives.

To a greater extent than current tools/environments for ontology development/evolution we require a system that shall facilitate collaboration by:

- Imposing the minimum possible restrictions on the collaboration and multi-party development/evolution process: The evolution, creation and access to different versions of ontologies needs to be made seamlessly, for all participants, without directly affecting each other.
- Actively supporting the detection of conflicts for each ontology version, and facilitating the reconciliation of conflicts by exploiting the semantics of the actions performed.
- Being methodology-independent, therefore generically applicable.
- Facilitating the deployment of ontology engineering tools in distributed settings, so as to support collaboration.
- Being seamlessly combined with any argumentation framework.

We need to emphasize that the implementation of such a generic system, acting as an infrastructure for collaborative ontology development and evolution, will allow the transfer of ontology engineering tools from their "classic" standalone mode, to their deployment in distributed settings.

To address these issues we have implemented SR-COE. It actively supports people to collaborate while being either connected or disconnected: Each user holds a replica of an ontology, which is being developed/evolved by using any preferred tool. Modifications performed by remote and concurrent collaborators on their local replicas need to be merged: As local replicas may diverge, possible conflicts between them must be detected and reconciled. As a result of this reconciliation, all users are aware of (a) the modifications made by others, (b) of the possible conflicts that arise, and (c) of the possible, alternative conflicts-free versions of the ontology. We have to point out that the system has to be able to support any state of the art methodology that accentuate the active participation of knowledge workers in the ontology engineering lifecycle, (e.g. HCOME and DILIGENT).

3 BACKGROUND KNOWLEDGE

3.1 Semantics based Reconciliation via Telex

The work presented in (Shapiro et al, 2004) introduced a formalism for maintaining consistency in distributed systems for data sharing. Based on this formalism, work done in Telex (Benmouffok et al., 2009) enables collaborative and distributed development and evolution of "documents". A document may be of any form and format, containing formal or informal specifications. In the context of our work, *a document is an evolving ontology*.

An application is a "middleware" between the user and Telex (Shapiro et al, 2004). Telex is instantiated in each collaborating site. Users by means of their applications can modify their local replicas of documents, either being online or offline, while Telex takes the hard responsibility to merge replicas. To do this, applications need to update Telex with fragments of actions and constraints. Then, Telex exploits the semantics of modification actions which are being captured by constraints. Constraints are application-specific invariants that parameterize Telex. The types of constraints are presented in the paragraphs that follow.

Telex takes care of replication, consistency, storage and access control; collecting, transmitting and persisting actions; detecting conflicts and computing high-quality schedules.

Telex in each site maintains an Action-Constraint Graph (ACG), where nodes are actions, and arcs constraints between them. The actions represent operations that users apply to replicas of a document via their applications, and arcs represent constraints between these operations. Constraints enable the computation of possible, alternative schedules. The ACG is formed by the union of all operations performed in *all* sites: When a local update occurs, the system replicates every action to the other collaborating sites.

Table 1 shows the set of available types of constraints (Benmouffok et al., 2009). The constraint `NotAfter` indicates that an ordering between the actions `A` and `B` must be maintained, such that no schedule that commits to the execution of both actions executes `B` before `A`. The constraint `Enables` is an implication between the corresponding actions. If a schedule commits to execute `B`, then it must also commit to execute `A` without considering any ordering between the actions.

Table 1: Constraint types.

Name	Notation	Meaning for scheduling
NotAfter	$A \rightarrow B$	A is never after B in any schedule
Enables	$A \lhd B$	B in a schedule implies also A in the same schedule
NonCommuting	$A \nparallel B$	N/A
Atomic	$A \substack{\lhd \\ \rhd} B$	Either both A and B execute or nothing
Casual Dependence	$A \underrightarrow{\lhd} B$	B executes after A and if only A succeeds
Antagonism	$A \leftrightarrows B$	Conflict: A and B never in the same schedule

In case that A is aborted in a schedule this constraint forces B to be aborted as well. Combining these primitive constraints, the Atomic, Casual Dependence and Antagonism constraints are generated. The Casual Dependence constraint defines that action B depends on A, and a schedule commits to it iff it commits to A. The Antagonism constraint identifies a conflict between actions A and B. This constraint defines that when a schedule commits to execute action A, it aborts B and vice-versa.

A set of actions is said to be in conflict, if they form a "→" (NotAfter) cycle. This means that no sound schedule can commit to the execution of all these actions. The Antagonism constraint is a special case of such as cycle between the actions A and B.

As shown in Figure 1, Telex comprises the Scheduler, the Agreement module, the Logger and the Transmitter. Each detected conflict triggers the generation of schedules from Scheduler, which "marks" each action to be either "aborted" or "committed" in each schedule. For instance, the detection of a pair of antagonistic actions drives Telex to construct two alternative schedules, where each of the antagonistic actions is either aborted or committed.

A schedule may be selected by the local application and be provided to the agreement module as a proposal. Because of the asynchronous communication, proposals may be different. The agreement module is responsible though to help sites reach an agreement. It is possible that some applications require the selection of proposals based on application specific criteria and/or policies. Telex provides the freedom to developers to implement the algorithm they need. The basic algorithm relies on voting, so that user preferences are to be taken into account. It is important to be mentioned that each Telex module is instantiated per each locally "open" document. We refer the interested reader to (Shapiro et al, 2004) for further details.

3.2 Ontology Evolution via Validity Rules

Considering the problem of ontology evolution in the face of modification actions, authors in (Konstandinidis et al., 2008) devise a framework and algorithm for determining the effects and side-effects of any such action. Although the framework proposed is general, authors focus in a fragment of RDF/S. To abstract from the underlying language and develop a uniform framework, RDF is mapped to First-Order Logic (FOL) expressions. The representation of ontological facts by means of First Order Logic predicates is shown in table 2 (Konstandinidis et al., 2008).

Table 2: Ontological facts as First Order Logic predicates.

RDF triple	Intuitive Meaning	Predicate
C rdf: type rdfs: Class	C is a class	CS(C)
P rdf: type rdf: Property	P is a property	PS(P)
x rdf: type rdfs: Resource	x is a class instance	CI(X)
P rdfs: domain C	domain of property	Domain(P,C)
P rdfs: range C	range of property	Range(P,C)
C_1 rdfs: subClassOf C_2	IsA between classes	$C\_IsA(C_1,C_2)$
P_1 rdfs: subPropertyOf P_2	IsA between properties	$P\_IsA(P_1,P_2)$
x rdf: type C	class instantiation	C_Inst(x,C)
x P y	propperty instantiation	PI(x,y,P)

The semantics of specifications are captured via validity rules, which introduce a validity model. Validity rules are encoded in the form:

$$\forall \vec{u} P(\vec{u}) \rightarrow \vee_{i=1,\dots,n} \exists v_i Q_i(\vec{u},\vec{v}_i)$$

where $\vec{u},\vec{v}_i$ are tuples of variables, P and Q_i are conjunctions of relational atoms of the form $R(w_1,\dots,w_n)$, and equality atoms of the form $(w_k=w_m)$, where $w_1,\dots,w_n$, w_k,w_m, are variables or constants. An example of a validity rule is the following:

$$PI\,(x,y,z) \wedge Domain\ \ (z,w) \rightarrow C\_Inst\,(x,w)$$

This rule states that given an instance $(x\ z\ y)$ of a property z whose domain is w, then x must be an instance of w. All validity rules are specified in (Konstandinidis et al., 2008).

The FOL specifications are equipped with closed semantics, i.e., CWA (*closed world assumption*). This means that, for two formulas p, q, if $p \nvdash q$, then $p \vdash \neg q$. Abusing notation, for two sets of ground facts U, V, we will say that U implies V ($U \vdash V$) to denote that $U \vdash p$ for all p in V. Any expression of the form $P(x_1,\dots,x_k)$ is called a *positive ground fact* where P is a predicate of arity k and $x_1,\dots,x_k$ are constant symbols. Any expression of the form $\neg P(x_1,\dots,x_k)$ is called a *negative ground fact* iff $P(x_1,\dots,x_k)$ is a positive ground fact. L denotes the set of all well-formed formulae that can be formed in this FOL. We denote by $L+$ the set of positive ground facts, and $L-$

the set of negative ground facts. An evolving *ontology* is a set $K \subseteq L+$. In simple words, an ontology is any set of positive ground facts. An *update* is any set of positive or negative ground facts. The application of any modification action to an evolving ontology should result in a set of *effects*. Each effect is a ground fact.

By definition, ontologies have two properties: (a) they are always consistent (in the purely logical sense) and (b) they imply only the positive ground facts that are already in the ontology. The above two properties together with the CWA semantics, imply that (a) $P(x) \in K \Leftrightarrow K \vdash P(x) \Leftrightarrow K \nvdash \neg P(x)$ and

(b) $P(x) \notin K \Leftrightarrow K \vdash \neg P(x) \Leftrightarrow K \nvdash P(x)$

An application of these properties is that, applying a modification action with effect $\neg P(x)$ to the ontology K corresponds to removing $P(x)$ from K. On the other hand, updating an ontology with positive ground facts corresponds to adding facts in K.

With the aim to specify and compute conflicts using validity rules, following (Konstandinidis et al., 2008), we use component sets. The component set that corresponds to a validity rule c with respect to some tuple of constants $\vec{x}$ is defined as

$$Comp(c, \vec{x}) = \{\{\neg P_j(\vec{x}) \mid 0 < j \le k\} \cup \{Q_{i1}(\vec{x}, z) \wedge Q_{i2}(\vec{x}, z) \wedge \ldots \wedge Q_{im}(\vec{x}, z) \mid 1 \le i \le n, z : constant\}\}$$

For example, for the rule

$$PI(x, y, z) \wedge Domain(z, w) \rightarrow C\_Inst(x, w)$$

the derived component set would be

$$\{\{\neg PI(x,y,z)\}, \{\neg Domain(z,w)\}, \{C\_Inst(x,w)\}\}$$

As proposed in (Konstandinidis G. et al., 2008), a ground fact $P(x)$ which is added in an ontology K, would *violate* a rule c, iff there is some set V and tuple of constants $\vec{u}$ for which $\neg P(x) \in V$ and $V \in Comp(c, \vec{u})$ and *for all* $V' \in Comp(c, \vec{u})$, $V \neq V'$, it holds that $K \nvdash V'$.

Let us for instance consider the following scenario that exemplifies the above and reveals our considerations for collaborative ontology evolution: Let us consider George who performs an action that adds the fact $PI(x,y,z)$ in his ontology replica, and Peggy who performs an action with effect $Domain(z,w)$, in her own replica. Then, none of these replicas violate the validity rule $PI(x, y, z) \wedge Domain(z, w) \rightarrow C\_Inst(x, w)$, according to the above mentioned definition (given that $C\_Inst(x,w)$ does not hold in any of the replicas). This is indeed so, since, given the component set of this rule, the fact $\neg Domain(z,w)$ (respectively, $\neg PI(x,y,z)$) validates the rule for George (respectively Peggy). However, when the two replicas are combined, the rule is violated, given that in the merged version $PI(x,y,z)$ and $Domain(z,w)$ hold but $C\_Inst(x,w)$ does not hold. Notice that if $C\_Inst(x,w)$ holds, then both local views are consistent, as well as the global view. However, it is obvious that users cannot be "forced" to insert the fact $C\_Inst(x,w)$ in their replicas to assure the validity of the merged version. In the merged version this must be done to maintain consistency, or else, alternative schedules must be constructed where either $PI(x,y,z)$ or $Domain(z,w)$ is aborted.

4 COLLABORATIVE ONTOLOGY EVOLUTION

In order to apply the semantic-based reconciliation method to the domain of ontology evolution, possible modification actions and their constraints must be defined.

Table 3 shows the constraints that derive from the validity rules. The first column provides a reference name, the second one contains the component sets derived from the validity rules, and the third one provides the corresponding constraints. At this point it is important to notice two things:

(a) Constraints concern the effects of modification actions, rather than the actions themselves: Therefore, for SR-COE, nodes in ACG are labelled with predicates from Table 2 and conjunctions of them. *Subsequently, when we refer to actions we actually refer to their effects*. Given a schedule as described above, provided by Telex, we can create a new ontology version from the schedule itself. The generated version incorporates the effects of actions performed by the different collaborating parties and it provides a "landmark" for ontology developers to proceed towards an agreed conceptualization. This landmark is called a "*checkpoint*". Checkpoints give the ability to the users to "roll back", i.e. to undo modifications.

(b) Some of the constraints in Table 3 have a compound side, something which is not inherently supported by Telex. For this purpose we introduce the notion of the *compound* (*complex*) *action*. A compound action is formed by a conjunction of actions. Its status, as it is determined by the Telex scheduler, depends on the status of the actions it consists of: If any atomic action contained in some compound action is aborted, the compound is aborted as well. A compound action is also represented as an action in the ACG, along with the constraints that bound it to the atomic actions it consists of. For instance, given a compound action "*x_and_y*", the constraints $"x" \overset{\triangleleft}{\rightarrow} "x\_and\_y"$ and

$"y" \overset{\lhd}{\rightarrow} "x\_and\_y"$ are added to ACG.

Table 3: Derived constraints from component sets.

Rule name	Rule Component Set	Constraint(s)
Domain applicability	**R1.1:** $\forall x,y \in \Sigma$: Comp(R1.1,(x,y))= {{¬Domain(x,y)},{PS(x)}} **R1.2:** $\forall x,y \in \Sigma$: Comp(R1.2,(x,y))= {{¬Domain(x, y)},{CS(y)}}	$PS(x) \wedge CS(y) \overset{\leq}{\rightarrow} Domain(x,y)$
C_IsA Applicability	**R2.1:** $\forall x,y \in \Sigma$: Comp(R2.1,(x,y))= {{¬C_IsA(x,y)},{CS(x)} **R2.2:** $\forall x y \in \Sigma$: Comp(R2.2,(x,y))= {{¬C_IsA(x,y)},{CS(y)}}	$CS(x) \wedge CS(y) \overset{\leq}{\rightarrow} C\_IsA(x,y)$
C_Inst Applicability	**R3.1:** $\forall x,y \in \Sigma$: Comp(R3.1,(x,y))= {{¬C_Inst(x,y)},{CI(x)}} **R3.2:** $\forall x,y \in \Sigma$: Comp(R3.2,(x,y))= {{¬C_Inst(x,y)},{CS(y)}}	$CI(x) \wedge CS(y) \overset{\leq}{\rightarrow} C\_Inst(x,y)$
Domain is unique	**R4:** $\forall x,y,z \in \Sigma$: Comp(R4,(x,y,z))= {{¬Domain(x,y)}, {¬Domain(x, z)},{(y=z)}}	$(y = z) \overset{\leq}{\rightarrow} Domain(x,z) \wedge Domain(x,y)$
Domain and range exists	**R5.1:** $\forall x \in \Sigma, \exists z \in \Sigma$: Comp(R5.1,(x,z))= {{¬PS(x)},{Domain(x,z)}} **R5.2:** $\forall x \in \Sigma, \exists y \in \Sigma$: Comp(R5.2,(x,y))= {{¬PS(x)}, {Range(x,y)}}	$Domain(x,z) \wedge Range(x,y) \overset{\leq}{\rightarrow} PS(x)$
C_IsA Transitivity	**R6:** $\forall x,y,z \in \Sigma$: Comp(R6,(x,y,z))= {{¬C_IsA(x,y)}, {¬C_IsA(y, z)}, {C_IsA(x, z)}}	$C\_IsA(x,z) \overset{\leq}{\rightarrow} C\_IsA(x,y) \wedge C\_IsA(y,z)$
C_IsA Irreflexivity	**R7:** $\forall x,y \in \Sigma$: Comp(R7,(x,y))= {{¬C_IsA(x,y)}, {¬C_IsA(y, x)}}	$C\_IsA(x,y) \overset{\leq}{\rightarrow} \neg C\_IsA(y,x)$
Determining C_Inst	**R8:** $\forall x,y,z \in \Sigma$: Comp(R8,(x,y,z))= {{¬C_Inst(x,y)}, {¬C_IsA(y, z)}, {C_Inst(x, z)}}	$C\_Inst(x,z) \overset{\leq}{\rightarrow} C\_Inst(x,y) \wedge C\_IsA(y,z)$
Property Instance_of and Domain	**R9:** $\forall x, y, z, w \in \Sigma$: Comp(R9,(x,y, z, w))= {{¬PI(x, y, z)}, {¬Domain(z, w)}, {C_Inst(x, w)}}	$C\_Inst(x,w) \overset{\leq}{\rightarrow} PI(x,y,z) \wedge Domain(z,w)$
P_IsA Irreflexivity	**R10:** $\forall x,y \in \Sigma$: Comp(R10,(x,y))= {{¬P_IsA(x, y)}, {¬P_IsA(y, x)}}	$P\_IsA(x,y) \overset{\leq}{\rightarrow} \neg P\_IsA(y,x)$

Considering again our example, the constraint corresponding to the "Property instance and Domain" rule will not be violated in any of the replicas where *PI(x,y,z)* or *Domain(z,w)* hold, but a conflict will be detected in the merged version, where both *PI(x,y,z)* and *Domain(z,w)* hold and *C_Inst(x,w)* does not hold.

Each of the collaborators modifies a local replica of an ontology: The effects of modifications label the nodes in ACG. Telex replicates and schedules these effects to all sites. Any user may create a checkpoint from a generated schedule and continue evolving it. A checkpoint is published to remote users as a new ontology version, by the time the user who created it, starts modifying it. The set of shared checkpoints defines the *versioning tree*: Users may select to evolve any checkpoint created by any of the collaborating parties, creating a new set of checkpoints, and so on. The depth and breadth of this tree is subject to the rules and policies that the collaborating community sets.

5 SR-COE ARCHITECTURE

An overview of the SR-COE architecture is presented in figure 1.
Assuming that the user modifies a locally stored ontology version, then, the following events occur:

- The ontology parser detects new modifications (the *diff*) and calls the validity checker. These effects will not directly affect other replicas. Doing so, it allows users to modify the same ontology with no restriction, concurrently.
- The validity checker generates a new fragment that is sent to Telex.

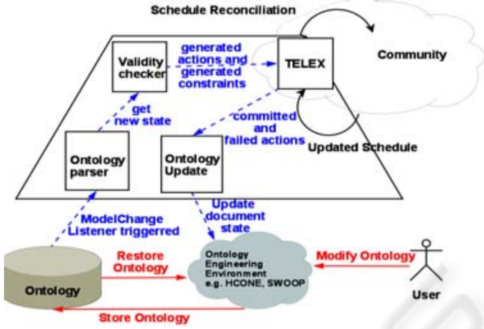

Figure 1: The overall architecture of SR-COE.

- Telex takes control, to handle new fragments appropriately. Modifications that do not raise conflicts in schedules are appended to these. Otherwise, alternative schedules are provided, if possible.
- The ontology updater module informs the user for any new schedule and updates the locally stored ontology model, on user's request.

The paragraphs that follow present each SR-COE module in detail.

The Ontology Parser module is responsible to monitor the selected ontology version and retrieve the effects of modification actions performed. When a change occurs, the module identifies the elements added or removed and provides them to the validity checker module. This module is also responsible to creating a checkpoint upon user's request.

Given the new facts retrieved by the ontology parser, the validity checker module is triggered to infer new facts (i.e. inherited properties and transitivity of subsumption relations), to produce constraints and to validate the changes in the local replica.

In case that removed elements are detected, the validity checker constructs antagonism constraints between these facts and their negations. Constraints generated by the validity rules are initially exploited locally, to check for local conflicts. If no conflicts are found, the validity checker proceeds to the generation of a fragment containing new facts and corresponding constraints to be sent to Telex. Otherwise, the validity checker generates antagonism constraints between the conflicting facts, driving the computation of alternative schedules. Doing so, users can continue modifying the locally stored file, even if a conflict has occurred.

The Scheduler and Agreement modules of Telex provide global consistency (given sound constraints), and provide assistance towards reaching an agreed conceptualization (Benmouffok et al., 2009). Each user "runs" an instance of SR-COE, and thus of Telex, that operates locally. Telex is responsible to maintain the ACG, given the fragments generated from validity checker module. When the graph is updated by new nodes and the corresponding constraints, the scheduler computes new schedules, which are provided to the ontology updater module.

The ontology updater connects the GUI of the corresponding ontology engineering tool and SR-COE. In our prototype implementation, it provides information on the available schedules, as well as information about modification actions (issuer, action, effects) performed in the concurrent collaborating sites. It notifies users about conflicts detected and schedules built, without requiring users to take any particular immediate action. However, users may request to view a schedule and create a new checkpoint which is added to the versioning tree. Subsequently, the user can access this new file via SR-COE, making this version available to the rest of the community. Users proceed by choosing the version to modify, creating new checkpoints, and so on, until they reach an agreed conceptualization.

It is important to be mentioned that ontologies comprise only positive facts that correspond to committed actions in a schedule. Negative facts are incorporated as nodes in ACG so as conflicts between replicas to be detected.

6 CONCLUSIONS

To a greater extent than current tools/environments for ontology development we have built a system that facilitates collaborative ontology development and evolution by: (a) Imposing the minimum possible restrictions on the collaboration and multi-party development/evolution process, allowing the creation of different versions of ontologies and the seamless access to these versions. (b) Actively supporting the detection and reconciliation of conflicts, by exploiting the semantics of the actions performed. (c) Being methodology-independent, therefore generically applicable. (d) Facilitating the deployment of current ontology engineering tools in distributed settings, supporting collaboration.

As described above, the validity rules presented support the development and evolution of lightweight ontologies using a rather simple model. However, SR-COE can be easily customised by incorporating any set of validity rules, to deal with more expressive ontology languages.

An important step to be made concerns the study of using SR-COE in conjunction with different ontology engineering tools, maybe in the context of different methodologies.

REFERENCES

Farquhar, A. et al, 1997. The Ontolingua Server: A tool for collaborative ontology construction. Intl. J. Human-Computer Studies 46, pp. 707–728 .

Kotis, K., Vouros, G., 2006. Human-centered ontology engineering: The HCOME methodology, In KAIS, 10, pp. 109-131.

Tempich, C. et al, 2005. An Argumentation Ontology for Distributed, Loosely-controlled and evolvInG Engineering processes of oNTologies (DILIGENT). ESWC 2005, LNCS 3532, pp. 241–256 .

Sure, Y., et al 2002. OntoEdit: Collaborative Ontology Development for the Semantic Web. ISWC 2002, LNCS 2342, pp. 221–235 .

Arprez, J.C., at al, 2001. WebODE: a scalable workbench for ontological engineering. K-CAP'01 .

Bozsak, E., et al, 2002. KAON - Towards a Large Scale Semantic Web. 3rd Intl. Conf. on e-Commerce and Web Technologies, LNCS 2455, pp. 304–313.

Domingue, J., 1998. Tadzebao and WebOnto: Discussing, Browsing, and Editing Ontologies on the Web. KAW'98, Banff, Canada.

Ceusters, W., et al, 2001. LinkFactory: an Advanced Formal Ontology Management System. KCAP-2001, Victoria, Canada.

Tudorache, T., et al, 2008. Collaborative Protege: Enabling Community-based Authoring of Ontologies, In Proc. Of the Workshop on Social and Collaborative Construction of Structured Knowledge, ISWC 2008, Banff, Canada.

Seidenberg, J., Rector, A., 2007. A Methodology for Asynchronous Multi-User Editing of Semantic Web Ontologies. K-CAP'07, pp. 127–134.

Hotho, A., et al, G., 2006. BibSonomy: A social bookmark and publication sharing system. Conceptual Structures Tool Interoperability Workshop at the 14th Int. Conf. on Conceptual Structures, pp. 87-102.

Tummarello, G., et al., 2006. Enabling semantic web communities with DBin: an overview. ISWC 06, LNCS 4273, pp. 943–950.

Sunagawa, E., et al, 2003. An environment for distributed ontology development based on dependency management. ISWC'03, LNCS 2870, pp. 453-468.

Auer, S., et al, 2006. OntoWiki a tool for social, semantic collaboration. ISWC, LNCS 4273, pp. 736-749.

Shapiro, M., et al, 2004. A constraint-based formalism for consistency in Replicated Systems. OPODIS 04, LNCS 3544, pp. 331-345.

Benmouffok, L., et al, 2009. Telex: A Platform for Decentralised Sharing. In EuroSys 2009.

Konstandinidis G., et al, 2008. A Formal Approach for RDF/S Ontology Evolution. ECAI-08, pp. 70-74.