Managing Ontology Change and Evolution via a Hybrid Matching Algorithm

Saravanan Muthaiyah
George Mason University
Department of Computer Science
4400 University Dr
Fairfax, VA 22030, USA
smuthaiy@gmu.edu

Marcel Barbulescu
George Mason University
Department of Computer Science
4400 University Dr
Fairfax, VA 22030, USA
mbarbulescu@gmu.edu

Larry Kerschberg
George Mason University
Department of Computer Science
4400 University Dr
Fairfax, VA 22030, USA
kersch@gmu.edu

Abstract: - In this paper, we present the problem of ontology evolution and change management. We provide a systematic approach to solve the problem by adopting a multi-agent system (MAS). The core of our solution is the Semantic Relatedness Score (SRS) which is an aggregate score of five well-tested semantic as well as syntactic algorithms. The focus of this paper is to resolve current problems related to ontology upgrade and managing evolution amongst shared ontologies. This paper highlights issues pertaining to ontological changes in a shared ontology environment which includes creating, renaming, deletion and modification of existing classes. These changes will definitely impact shared concepts and users would have to update their local ontologies to be consistent with changes in the commonly shared ontology. We propose a less laborious method to achieve this by using a semi-automated approach where a bulk of the processing is carried out by matching agents that would eliminate extraneous data and hence would only recommend to the ontologist data that can actually be upgraded. We have also designed and built a prototype in the Java Agent DEvelopment Framework (JADE) for proof-of-concept.

Key-Words: - Hierarchical Repository, Semantic Matching, Syntactic Matching, Agent, Ontology

1 Introduction

Ontologies specify the conceptualization of a domain of knowledge and provide a shared understanding of that domain of knowledge [2]. It formally describes data concepts and their relationships to make them machine processable and understandable. This is determined via data interchange formats such as N-Triples, RDF (Resource Description Framework), Turtle (Terse RDF Triple Language) and OWL (Web Ontology Language). Agents are defined as software programs. They are autonomous entities, sometimes referred to as software robots (i.e. softbots). Multi-agent systems (MAS) are systems in which many agents or softbots interact with each other to achieve a personal goal or a common goal. Ontologies provide necessary vocabulary for agents to run queries and make assertions for data that needs to be exchanged among them. The Semantic Web is a web of data. Several components are necessary for the creation of the Semantic Web, mainly ontologies, OWL, RDF, logic and reasoning capabilities for software agents. It is a distributed and collaborative environment where ontologies can naturally evolve and co-evolve. This is because evolution of knowledge is something inevitable. When new knowledge is discovered existing knowledge is updated to maintain consistency.

Update involves inclusion of new data, renaming of existing data, adding annotations and removal of erroneous data [1][3]. Research in the area of ontology evolution is critical because these changes are unavoidable. Our literature survey shows that work done in managing ontology evolution so far is somewhat limited to single ontologies. Very little has been done for shared ontologies [3][7][14][15]. Our paper fills this void by leveraging agent systems that has been implemented in JADE [17].

2 Background

Ontologies are hand crafted by domain experts and as such it is impossible to find a perfect ontology that covers all aspects of a shared domain of knowledge. This is also in line with the evolving knowledge issue mentioned earlier. To facilitate knowledge reuse and at the same time allow experts to express their knowledge, even when they don't completely agree on it, a shared hierarchical ontology becomes necessary.

This allows an ontologist to create their own definitions which would be domain specific and at the same time share common concepts from the hierarchical structure. In a shared hierarchical ontology, knowledge is organized in different levels, each inheriting knowledge from parent ontologies completely without partial inheritance.

For multiple inheritance relationships it is important for knowledge inherited to be consistent with no naming clashes. Figure 1 below, shows a shared hierarchical knowledge repository. It starts with knowledgebase (KB-O) on the top and expands down to KB-X on the left and KB-Y on the right. KB-X expands to KB-D1 and KB-D2 and KB-Y expands to KB-D3 and KB-D4. There are two agents AD-1 and AD-4 which use parts of the shared knowledge. They also have their own locally developed knowledge i.e. KB-A1 for agent AD-1 and KB-A4 for agent AD-4.

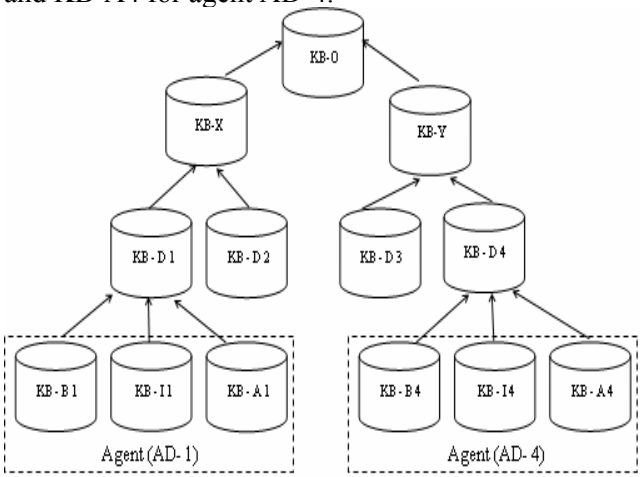

Fig.1 Shared Hierarchical Knowledge Repository

Agent A-D1 manages three local ontologies KB-B1, KB-I1 and KB-A1. Agent AD-4 manages ontologies KB-B4, KB-I4 and KB-A4. The local ontology for agent AD-1 inherits knowledge from KB-D1, KB-X and KB-O. The local ontology for agent AD-4, inherits knowledge from KB-D4, KB-Y and KB-O. Agent AD-1 and AD-4 are aware that their inherited knowledge is common to KB-O (see figure 2), when they collaborate. Knowledge reuse becomes easier in this way. Particularly when creating a new ontology, we can determine which parent ontology to inherit from and start adding the new knowledge to what already exists. Also, contradictory pieces of knowledge can be encoded in different ontologies which are not in an inheriting relation (e.g. one is not a parent of the other).

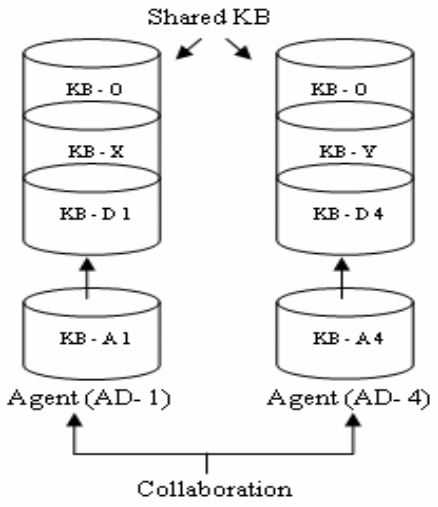

Fig.2 Collaboration between AD-1 and AD-4

At the same time, the communication between agents becomes easier with this way as they share some common knowledge. The hierarchical structure helps knowledge reuse. However, it is not realistic to assume that all developed ontologies will be under one central control and available at all times. As such, a distributed model of a hierarchical repository is more appropriate (see figure 3).There are three servers which contain the knowledgebase and they are distributed. Ontologies in these knowledgebases have their own creator and inherit knowledge from other ontologies. To solve the availability problem, when a new ontology is inherits knowledge from another ontology, a copy of the inherited knowledge will be made available locally (e.g. KB-O). The reason for this is that the parent ontology does not have to be available online at all times in order to have all their children ontologies functioning properly. In the next section we discuss problems related to heterogeneity, ontology versioning and

evolution that is caused by these shared ontology structures.

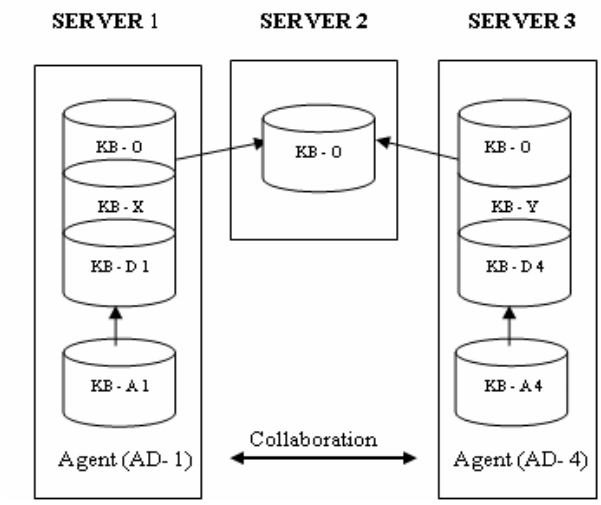

Fig.3 Distributed environment and collaboration

3 The Heterogeneity Problem

In a static environment where there are no updates made to an existing source ontology (SO) e.g. KB-O (see figure 3), heterogeneity is caused by mainly four problems. They are differences in ontological structures (structural heterogeneity), differences in data representation (semantic data heterogeneity), subjective mapping and atomic data storage [4]. Please refer to our work in [9][10] for detailed explanations. In a dynamic environment where changes are made to source ontology e.g. KB-O by its creator, such updates will create disparity to the shared concepts that are inherited by other knowledgebases e.g. (KB-D1, KB-X, KB-D4 and KB-Y) in servers 1 and 3 respectively. This is referred to as knowledge evolution problem and ontology versioning problem. This paper focuses on both static as well as dynamic environment problems. A definition is provided in the next section. Strategies to mitigate these problems are also discussed as part of our mediation framework.

3.1 Knowledge Evolution

Knowledge evolution is a situation where updates are continuously made to ontologies and as a result of that SO evolves to a new state. The critical thing here is to device a strategy to manage this evolution. We introduce a versioning strategy to enable sharing of static ontology versions. Let's call KB-O as the upper ontology and the first time it's published in the public domain, it is labelled as **first version** (v1). This version doesn't change until its maintainers update it and release a new version. The new version is developed locally and released only when they are ready. Let's call it KB-O (v2), where v2 is **version two**. The knowledgebases that inherited from v1 i.e. **(**KB-D1, KB-X, KB-D4 and KB-Y) now need to be upgraded to reflect changes of v2 of their parent ontology, KB-O (v2).

3.2 Ontology Versioning

Figure 3, shows a hierarchical ontology repository. Let's consider that we have multiple versions available for the following ontologies (see Table 1). Suppose the direct parents of KB-D1 (v1) are KB-X (v1) and KB-Y (v1) and if the maintainer of the KB-D1 decides to upgrade one of his parent ontologies, (i.e. KB-X (v1) or KB-Y (v1)) he has to upgrade it to inherit from other newer parent ontologies as well, due to version dependencies.

The maintainer will update to inherit knowledge from KB-X (v2) as it still uses KB-O (v1) but cannot update to inherit knowledge from KB-Y (v2), as it inherits from KB-0 (v2). Also there are no versions of KB-X that inherit knowledge from KB-0 (v2) (see table 1). Given that only one version of the ontology can exist at any one time, inherited knowledge (i.e. KB-0 (v1) and (v2)) cannot be inherited at the same time, directly or indirectly, by the same ontology. This also applies to KB-D4 as it uses KB-O (v2) but KB-X update will be based on KB-O (v1).

Table 1 Ontology Versions

Ontology	Versions (Inherit from)
KB-0	v1, v2
KB-X	v1 (KB-0 v1), v2 (KB-0 v1)
KB-Y	v1 (KB-0 v1), v2 (KB-0 v2)
KB-D1	v1 (KB-X v1, KB-Y v1)
KB-D4	v1 (KB-Y v1, KB-X v1)

After selecting the new versions of ontologies to inherit from, we put the inherited knowledge together. When attempting to upgrade to newer versions of inherited ontologies, naming clashes can happen as more than one version could have defined the same ontological concept. This would make both

ontologies "incompatible" and impossible to inherit from both versions at the same time.

If the operation of putting together the inherited knowledge succeeds, the next step is updating the locally defined ontology to accommodate the newly inherited knowledge. This also calls for an evolution log to be maintained. It would explicitly record changes made to the ontology so that changes will be easier to track in the ontology upgrading process. Changes to be logged are for all **new** elements, **modified** elements and **deleted** elements. In the next section we provide a summary of work that has been done to address these heterogeneity problems.

4 Literature Review

The change management approach examines ontology change management by addressing four major change environments such as consistent ontology evolution, repairing inconsistencies, reasoning with inconsistent ontologies and ontology versioning [16]. The authors emphasize on syntactic vs. semantic discrepancies, language dependent vs. language independent and functional or non-functional change. Consistency occurs when the developer constantly updates changes in definitions affecting his shared ontology. Inconsistencies are dealt via a reasoning system e.g. Processing Inconsistent ONtologies (PION) which is a system implemented in XDIG, an extended description logic interface [16]. It verifies the relevance of a returned query and extends consistent sub-ontology for further reasoning. In the end the ontology is transformed to an updated version where previous data is not lost. The authors however, did not provide a methodology of how the hierarchical ontology classes were actually matched. This is where we make a significant contribution.

A multi-agent system (MAS) architecture and algorithm for multiple agents is proposed for managing and deploying ontologies in a dynamic environment [6][8]. Only [8] proposes implementing MAS for managing ontology evolution. The authors propose a three layered architecture which includes agents and other functionalities such as learning and rule generation to refine ontologies and map between multiple ontologies. However, how agents were specified in JADE was still unclear. Our implementation overcomes the limitations of this paper.

Change and Annotation Ontology (CHAO) was developed by the creators of Protégé to keep track of the changes that happens in the ontology [12]. Changes are represented as subclasses of *Change* class and *Annotation* class. Changes made by users are represented as instances of corresponding subclass of *Change* and contain information describing the change and the class, where the property or individual to which the change is applied. Change ontologies are populated either by ontology tools during the editing episodes or generated by specialized tools that compare two ontologies and extract structural changes (e.g. Prompt) [12][13]. This paper provides a good framework that we use for building our MAS.

Another approach focused on "is-a" relations between classes in a hierarchy [5]. The authors propose to derive the similarity between classes from the similarity of the associated instances. The instance-based matching technique used was based on "Minimum Similarity" (MS) of hierarchical ontology structures developed at the University of Leipzip. However, it still does not quite address the upgrading problems in the ontology. Also, it is only useful in ontology structures that do not have to be upgraded.

We provide a hybrid approach for mapping concepts and instances to resolve this. Four algorithms out of thirteen linguistic and non linguistic matching algorithms were selected such as Lin, Gloss Vector, WordNet Vector and LSA (Latent Semantic Analysis) to determine our similarity scores (i.e. SRS). Our experiments have proven that the combination of these four measures provide highest reliability and precision compared to plain syntactic matching or any other combination of the thirteen algorithms [12]. In the next section we present our MAS architecture for the upgrade process.

5 MAS and the UPGRADE operation

We propose a MAS architecture (see figure 4) for the upgrade process (i.e. upgrade the local ontology to work with a new version of a shared ontology). Several agents are introduced for our architecture:

- Ontology Agent (OA): Used by other agents to get access to the ontology. It uses OWL

ontologies and stores the ontology modifications in an evolution log. It responds to queries like "what is the root class for wine?", "what are the subclasses of red wine?" and performs ontology modification to create an instance as a child of white wine.

- External Communication Ontology Agent (ECA): Acts as a proxy to talk to agents in other platforms and handles inter-platform communication via agent communication language (ACL). The protocols used for communication are Common Object Request Broker Architecture (CORBA) and Remote Method Invocation (RMI). CORBA is used for inter-platform communication for example between platform 1 and 2 (see figure 4). Where as RMI is used for intra-platform communication for agents to communication locally in each platform.

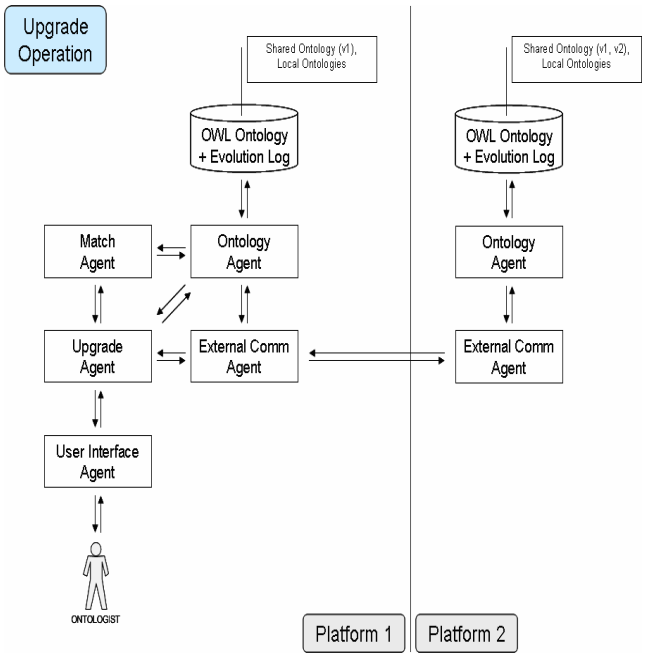

Fig.4. MAS architecture for the upgrade process

- Match Agent (MA): Computes similarity between two given concepts or instances. It the core of the semantic mediation process and it's based on the algorithms described in section 6.
- Upgrade Agent (UA): The business logic behind the upgrade process is encapsulated in this agent.
- User Interface Agent (UIA): Provides interface to the ontologist and handles actual reconfiguration of existing local ontologies during the upgrade process. It also services UA.
- The OWL ontology and evolution log is maintained in a repository as shown in figure 4. Platform 1 shows the (v1) of a shared ontology and the local ontology maintained in it. Platform 2 shows (v1) and (v2) of the shared ontology and its local ontology.

5.1 Upgrade Operation

The user or ontologist initiates the upgrading process through the UIA which communicates with the UA expressing the user's intention.

Then UA communicates with Ontology Agent (OA2) of Platform 2 via ECA (i.e. ECA1 and ECA2) and receives the differences from the current version of the shared ontology and the latest available version. The differences consist of atomic changes saved by the OA2 during previous ontology modification operations. They can be saved in an evolution log or can be used. The UA will show the user the differences through the UIA and will receive the permission to go on with the upgrade process or to abort it.

During the upgrade process UA will process the shared ontology differences. If the differences do not affect elements used in the definitions in the local ontology, they are performed directly without any modification of the local knowledge. Or else the differences need to be applied and the old version of the shared ontology is to be updated exactly to reflect the new version. Modifications if any, is done to the local ontology. We believe that if modifications need to be performed, semantic and syntactic mediation would play a major role in finding the best way of reorganizing the local knowledge. We provide a detailed example in section 5.2.

The goal of this paper is to give a concrete example where semantic and syntactic mediation can help in this process. Let's consider a class from the shared ontology that was used in a local ontology as a super-class. If the shared class is deleted, one possibility is to redefine the deleted concept locally and this will keep the hierarchy intact. Another possibility is to use semantic mediation and look for a new parent class for all local orphaned classes.

Our matching algorithm does exactly that. It helps us to find classes that are the most similar to

the deleted classes. In the past ontologists would have to painstakingly find similar matches manually. Lately syntactic matching based on string, prefix and suffix matching have reduced their workload but did not give reliable matches. Our hybrid matching algorithm that is based on both syntactic and semantic matching overcomes this limitation.

The ontologist is presented with more accurate matches and he chooses one of these classes as the new parent class based on the highest similarity score (SRS). The benefit of this is that the algorithm presents matches that are above the threshold (see appendix 1) and filters extraneous data that would otherwise increase the workload of the ontologist. The symbiotic relationship between the ontologist and the matching system makes this a semi-automated system. The algorithm is explained in section 6.

5.2 Upgrade Example

In this section we present an example to illustrate the upgrade process mentioned in section 5.1 using a wine ontology example. A local wine distributor, develops a wine ontology for his store. He develops this ontology by sharing general ontological concepts from the shared wine ontology and creates his own white wine definitions for his local ontology which is domain specific as he specializes only in retailing white wine.

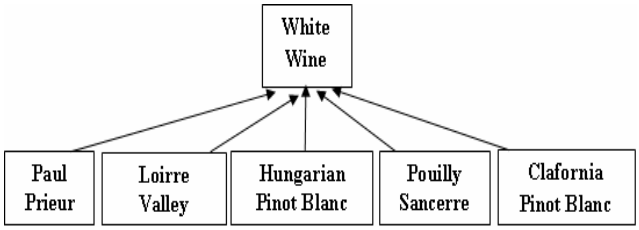

Fig.5. Local ontology classes for white wine

He begins creating five new classes under white wine which is a shared upper class (see figure 5). The subclasses that he created were Paul Prieur, Loirre Valley Sancerre, Hungarian Pinot Blanc, Pouilly Sancerre and Clafornia Pinot Blanc. Figure 6, shows classes in the local white wine ontology together with shared classes from the shared ontology highlighted by the dotted lines. Figure 7, shows a hierarchy where the local white wine ontology shares classes from the upper ontology.

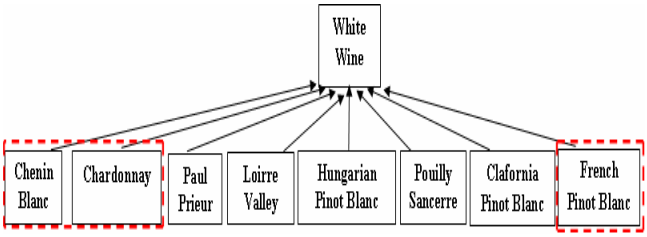

Fig.6. Shared ontology classes

Figure 8 shows a hierarchy of shared and local ontology classes for the white wine ontology. The red dotted line indicates the shared ontology and the blue dotted line indicates all local white wine classes that are domain specific. Let's assume that the shared or upper ontology was updated with new knowledge. For example if new concepts or classes were added on to the existing domain of white wine.

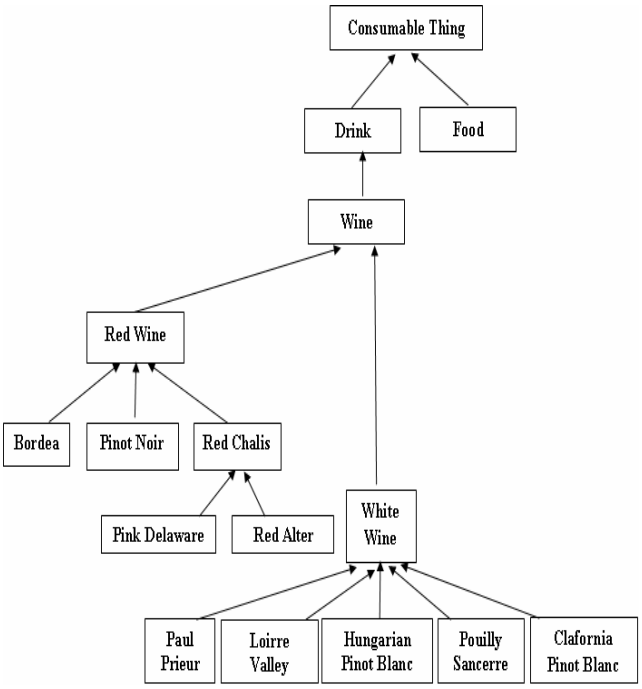

Fig.7. Concepts shared from upper ontology

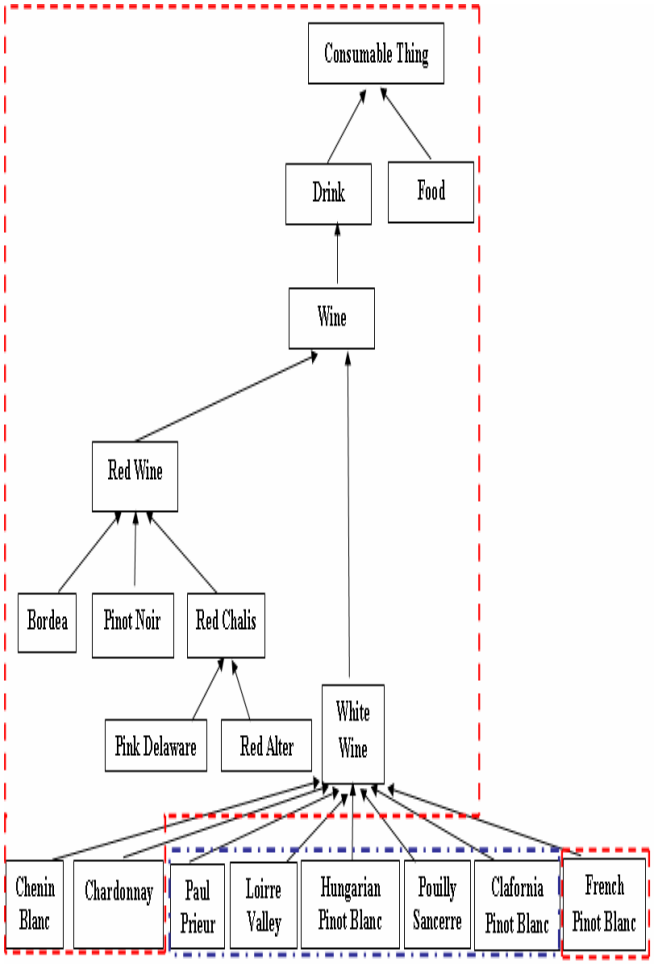

Fig.8. Shared and local ontology classes

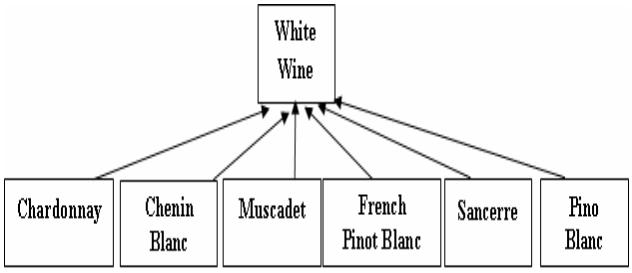

Fig.9. Newly updated classes in shared ontology

Figure 9 above shows the newly added classes in the shared ontology. Chenin Blanc, Chardonnay and French Pinot Blanc are the classes that existed before the shared ontology was updated. The newly updated classes are Sancerre, Pino Blanc and Muscadet.

6 Matching Algorithm and Similarity Definition

In this section we provide a formal definition for similarity and define our hybrid matching algorithm that was discussed earlier [11].

6.1 Similarity Definition

Let's first analyze the symbols used in our definition. (C) denotes concept or class, (c) denotes attributes or slots and (O) for ontology. Similarity (s) is a function of equality (E), inclusiveness (IC), syntactic similarity (SYN), semantic similarity (SEM) and consistency (CN) [11]. Thus producing the following function:

$$(s)\, f_x = \{ \text{E, IC ,CN, SYN and SEM} \} \quad (1)$$

The similarity function negates all disjoint (D) concepts (C) and attributes (c). As such, the new function is:

$$(s)\, f_x = \{ \text{E, IC,D,CN,SYN and SEM} \} \quad (2)$$

Mappings are produced only after (s) is determined. Each component of the (s) function is described in the following sections.

6.1.1 Equality (E)

Concepts (C) are **equal** if, they: 1) have semantically equivalent data labels, 2) are synonyms or 3) have the same slots or attribute names.

6.1.2 Inclusiveness (E)

Concepts (C) are **inclusive** if, the attribute (c) of one is inclusive in the other. For example if c_i = selling price and c_j =price, then c_i is a type of c_j. In other words *selling price* is inclusive in *price*. This is applicable to *hyponyms*.

6.1.3 Disjoint (D)

Concepts (C) are **disjoint** if, their attributes (c), c_i and c_j have nothing in common s.t. $c_1 \cap c_2 = \{\}$, results in an empty set.

6.1.4 Consistency (CN)

Two classes are **consistent** if, their attributes for a given class, C_1 (O_1) i.e. *name, location, rank* and *price* have nothing in common (c_1~*name* ≠ c_2~*location* ≠ c_3~*rank* ≠ c_4 ~*price*) s.t. $c_1 \cap c_2 = \{\}$. All slots (c_1~*name*, c_2~*location*, c_3~*rank*, c_4 ~*price*) must be subsets of C_1 (O_1).This is configured with RacerPro [19].

6.1.5 Syntactic Matching (SYN)

Syntactic matching uses approximate string matching to integrate data labels. Based on a number of deletions, insertions and substitutions a source string is matched with a target string [20].In the next section we define semantic matching.

6.1.6 Semantic Matching (SEM)

Semantics uses representation of meaning to measure similarity between words. We measure semantics via linguistic and cognitive measures. Four out of thirteen linguistic and non linguistic matching algorithms such as Lin, Gloss Vector, WordNet Vector and LSA (Latent Semantic Analysis) was used to determine SRS scores. Our experiments have proven that the combination of these four measures provides higher reliability and precision [12].

6.2 Matching Algorithm

The following are the steps involved for the matching algorithm (see appendix 1):

- Step 1 – Read loaded SO and TO taxonomies: Semantic engine reads taxonomies of the SO and TO. Prepare for detailed matching tests of data labels, go to step 2.
- Step 2 – Equivalence Test: Test for the **equivalence** of source and target classes: Test 1) do they have semantically equivalent data labels, Test 2) are they synonyms or Test 3) do they have the same slots or attribute names. Equivalence also implies adjacent neighbours are equal. If equivalent, proceed to step 3, 4 and 5. Else go to step 1.
- Step 3 – Inclusive Test: Source and target classes or concepts (C) are **inclusive** if, the attribute (c) of one is inclusive in the other. In other words *selling price* (c_i) is inclusive in *price* (c_j), this is applicable to *hyponyms*. If inclusive, proceed to step 6.
- Step 4 – Disjoint Test: Source and target classes or concepts (C) are **disjoint** if, the intersection of their two attribute sets (c), c_i and c_j results in an empty set {} or ø. If match test is not disjoint, proceed to step 6.
- Step 5 – Consistency Test: Source and target classes or concepts (C) are **consistent** if, all the attributes or slots (i.e. c_1 and c_2) in the class, have nothing in common s.t. $c_1 \cap c_2 = \{\}$. All slots must belong to class that is being tested. This can be configured with RacerPro. If consistent, proceed to step 6.
- Step 6 – Syntactic Match: Syntactic match similarity scores based on class prefix, suffix, substring matches are calculated. This calculation is performed for every class in the source and target ontology. Go to step 7.
- Step 7 – Semantic Match: Semantic match similarity scores based on cognitive measures such as LSA, Lin, Gloss Vector and WordNet Vector are used. This calculation is done for every class in the source and target ontology. Go to step 8.
- Step 8 – Aggregate both similarity scores: Similarity inputs from step 6 and 7 are aggregated, to produce SRS. Go to step 9.
- Step 9 –Populate similarity matrix: The aggregated values (SRS) from step 8 of candidate labels are populated into the similarity matrix. Multiple matches are carried out. Values are to be verified against the threshold. Go to step 10.
- Step 10 – Set threshold: Threshold value (t) is set based on scale used. For a scale between, 0 and 1 the threshold value is usually 0.5 (t >0.5). Those below threshold are logged in file in step 12. If greater than the threshold value, go to step 11.

- Step 11 – Domain Expert Selection: At this stage, candidates from step 10 are presented to domain expert by the system. Input from step 12 is accepted at the discretion of the domain expert.
- Step 12 – Manual Log: Selection is made manually only for those values below threshold. The domain expert uses his own cognitive judgment. Go to step 13.
- Step 13– Mapping/Alignment/Merge: All the candidates for mapping, alignment or merge (i.e. integration) chosen from step 11 and 12 are processed. End.

7 JADE for Upgrade Operation

In this section we discuss the prototype that we have implemented in JADE for the upgrade operation of the local white wine ontology to reflect the new classes that have been added using our hybrid matching algorithm and SRS scores. The agents discussed in section 5 are loaded in the JADE platform. Figure 10 shows agents have been successfully loaded. This is also called the JADE agent management GUI. When all agents have been invoked and are ready to receive input, the upgrade process is ready to be executed.

The UA will execute to find the differences in the shared ontology with input from the ontologist. This is done by clicking the differences icon (see figure 11). UA communicates with OA (OA2) of Platform 2 via ECA1 and ECA2 and receives the differences between the current version of the shared ontology and the latest available version. In this case the newly updated classes are those shown in figure 9 earlier i.e. Sancerre, Pino Blanc and Muscadet. UA will show the ontologist the differences through the UIA (see figure 11). The ontologist would then proceed with the upgrade process. Changes are saved in an evolution log.

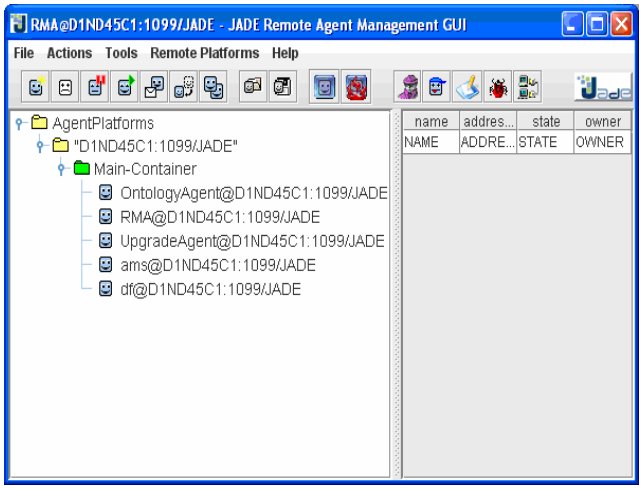

Fig.10. Agents loading in JADE

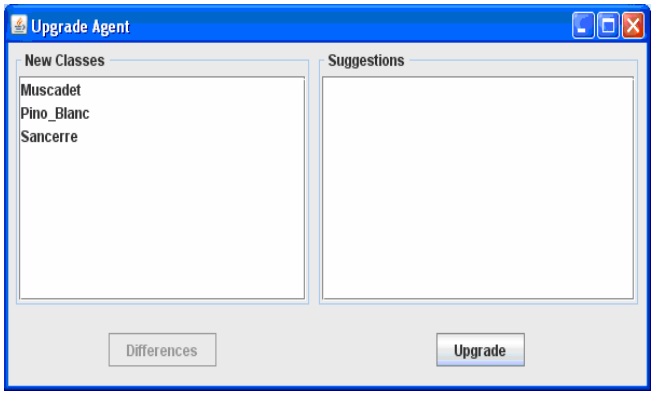

Fig.11. UIA showing all differences

8 Experiments and Results

We carried out an experiment to validate our approach. 50 questionnaires were distributed to domain experts and 38 responses were received giving this study a 76% response rate. The goal of the experiment was to test SRS against human cognitive responses (HCR) for ranking the similarity of word-pairs.

The idea for our experiment was conceived by works of Miller and Charles [21]. The results we obtained clearly shows that SRS had a higher correlation compared to purely SYN matching which is currently being used by the industry. Figure 12 shows the results that we had obtained and we had a significant positive correlation between the SRS and HCR i.e. r = + 0.919 (i.e. 91.9%). Table 2 shows significant correlation at 0.01, level (2-tailed) with

value (p) <0.05 thus (r =0.919, p <0.05) supports our null hypothesis (H_0) below.

(H_0): Combined scores match expert responses (HCR Rank)
(H_1): Combined scores don't match expert responses (HCR Rank)

Table 2 Pearson Product Moment Correlation

		Combined Rank	**HCR Rank**
Combined Rank	Pearson Correlation	1	.919(**)
	Sig. (2-tailed)		.000
	N	30	30

8.1 Precision and relevance of SRS

For further validation SRS was measured for precision and relevance compared to SYN scores. SRS obtained a 40% score for precision whereas SYN obtained a 16.67% score. In terms of relevance, SRS obtained a 96.67% score whereas SYN obtained only a 73.33% score. In summary, SRS provided higher precision and relevance scores compared to SYN.

9 Conclusion

In this paper we have introduced a multi-agent system and matching algorithm where multiple agents collaborate and deploy the management of changes that take place in a shared hierarchical knowledge environment. We demonstrate this by providing an example of a wine ontology that has been updated in the shared repository. This was mainly to see if it were more appropriate for the new concepts i.e. Sancerre, Pino Blanc and Muscadet to be added as subclass of white wine at the same level of the other local white wines (i.e. Paul Prieur, Loirre Valley Sancerre, Hungarian Pinot Blanc, Pouilly Sancerre, Clafornia Pinot Blanc) or to include Pino Blanc as a subclass of Hungarian Pinot Blanc or Claifornia Pinot Blanc. The match agent provides exact matches and presents this for the consideration of the ontologist and our experiments validates that our approach has a higher precision and relevance compared to current matching algorithms. Thus we are convinced that our approach will reduce the workload of the ontologist who at this moment have to carry out all the matches manually.

10 Acknowledgements

Our matching algorithm expands the work carried out by researchers, Li Li, Baolin Wu and Yun Yang of the Faculty of Information and Communication Technologies, Swinburne University of Technology, Melbourne, Australia. In particular, steps 2 to 5 of our matching algorithm which is explained in section 6.2, incorporates the definitions provided in their work. We would like to thank Li Li, Baolin Wu and Yun Yang for their prototyping work inspired us to build our own prototype using the JADE platform which is described in section 7.

11 References

[1] Giunchiglia, F. and Zaihrayeu, I. "Making peer databases interact - a vision for an architecture supporting data coordination." Proceedings of the Conference on Information Agents, Madrid, September 2002.

[2] Gruber, T.R., Toward Principles of Design Ontologies Used for Knoweledge Sharing, 1993.

[3] H. Wache, T. Voegele, U. Visser, H. Stuckenschmidt, G. Schuster, H. Neumann, and S. Huebner. "Ontology-based integration of information - a survey of existing approaches". Proceedings. of IJCAI, August 2001.

[4] H. Stuckenschmidt, U. Visser and H. Wache, ISWC-Tutorial, Sundial Resort, Sanibel Island, Florida, USA, October 20, 2003.

[5] Jun-ichi Akahani, Kaoru Hiramatsu, and Tetsuji Satoh "Approximate Query Reformation based on Hierarchical Ontology Mapping", pp 1-4.

[6] Jerry, F.,, Brad, P., Marian, N., and Bruce, B., Agent-Based Semantic Interoperability in InfoSleuth, SIGMOD Record 28:1, March, 1999, pp. 60-67.

[7] Kurgan, L, Swiercz, W and Cios, K "Semantic Mapping of XML Tags using Inductive Machine Learning", Department of Computer Science and Engineering, University of Colorado at Denver.

[8] Li Li, L.W., B.; Yang, Y. Semantic Mapping via Multi-Agent Systems in International

Conference on e-Technology, e-Commerce and e-Service, IEEE, April 2005: p. 54-57.

[9] Muthaiyah, S and Kerschberg, L "Dynamic Integration and Semantic Security Policy Ontology Mapping for Semantic Web Services (SWS)", IEEE, First International Conference on Digital Information Management ($ICDIM'_{06}$), Bangalore, India, pp. 116-120

[10] Muthaiyah, S and Kerschberg, L "Virtual Organization Security Policies: An Ontology-based Mapping and Integration Approach", Information Systems Frontiers (ISF), A Journal of Research and Innovation, Springer, USA, Special Issue on Secure Knowledge Management, 2007, pp. 505-515

[11] Muthaiyah, S and Kerschberg, L "A Hybrid Ontology Mediation Approach for the Semantic Web", Special Issue on Decision Technologies in E-Business, International Journal of E-Business Research (IJEBR), Idea Group Publishing Inc, U.S.A. ISSN: 1548-1131, EISSN: 1548-114X. – forthcoming.

[12] Natalya Fridman Noy, A.C., William Liu, Mark A. Musen. A Framework for Ontology Evolution in Collaborative Environments. in International Semantic Web Conference, 2006: p. 544 -558.

[13] Noy, N.F.a.M.A.M. PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment. in Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence. 2000. Austin,Texas: The MIT Press.

[14] Obrst, L., "Meditation and Data Sharing: Ontologies for Semantically Interoperable Systems", Proceedings of the Twelfth International Conference on Information and Knowledge Management, November, 2003, pp. 366-369

[15] Park, J. and Ram, S., "Information Systems Interoperability: What Lies Beneath?" ACM Transactions on Information Systems, Vol. 22, No. 4, October 2004, pp. 595-632

[16] Peter Haase, F.H., Zhisheng Huang, Heiner Stuckernschmidt and York Sure, A Framework for Handling Inconsistency in Changing Ontologies Springer-Verlang Berlin Heidelberg, 2005: p. 353-367.

[17] http://jade.tilab.com/

[18] http://jena.sourceforge.net/

[19] http://www.racer-systems.com/

[20] http://www.nist.gov/dads/HTML/Levenshtein.html

[21] G.A. Miller and W.G. Charles. Contextual correlates of semantic similarity. Language and Cognitive Processes, 6(1), pp.1–28, 1991

[22] Adrian Sergiu Darabant et al, The Similarity Measures and their Impact on OODB Fragmentation Using Hierarchical Clustering Algorithms, WSEAS Transactions on Computers, 5(9), September 2006, ISSN 1109-2750.

[23] W.Y.Zhang, F.R. Zhu and S.Zhang, A Service-Oriented Multi-Agent System Architecture for Multidisciplinary Collaborative Design in the Semantic Grid, WSEAS Transactions on Computers, 5(9), September 2006, ISSN 1109-2750.

[24] David B.Bracewell, Fuji Ren and Shingo Kuroiwa, Building Frames of Knowledge for Causal Agents in WordNet, WSEAS Transactions on Computers, 5(9), September 2006, ISSN 1109-2750.