Issues for Robust Consensus Building in P2P Networks

A-R. Mawlood-Yunis, M. Weiss, and N. Santoro

School of Computer Science, Carleton University
Ottawa, Ontario, K1S 5B6, CANADA.
{armyunis, santoro, weiss}@scs.carleton.ca

Abstract. The need for semantic interoperability between ontologies in a peer-to-peer (P2P) environment is imperative. This is because, by definition participants in P2P environment are equal, autonomous and distributed. For example, the synthesis of concepts developed independently by different academic researchers, different research labs, various emergency service departments and, hospitals and pharmacies, just to mention a few, are an assertive request for cooperation and collaboration among these independent peers. In this work we are looking at issues that enable us to build a robust semantic consensus to solve the interoperability problem among heterogeneous ontologies in P2P networks. To achieve a robust semantic consensus we focus on three key issues: i. semantic mapping faults, ii. consensus construction iii. fault-tolerance. All these three issues will be further elaborated in this paper, initial steps to address theses issues will be described and fault-tolerant semantic mapping research directions will be further identified.

1 Introduction

There has been considerable work on semantic interoperability, i.e. the mapping between different concepts from different ontologies. Some of this work suggests achieving interoperability through a global ontology mediator [6], while others suggest building consensus incrementally from local mapping [1, 7, 8]. We favor the latter approach and it is the focus of our research. To build a robust semantic consensus system we focus on three key issues: i. semantic mapping faults, i.e. semantic mapping conflicts, ii. consensus construction iii. fault-tolerance. The existing works on building semantic consensus among distributed ontologies do not distinguish between permanent and temporary **semantic mapping faults**. The failure to distinguish between permanent and temporary mapping faults could result in the erroneous labeling of peers with incompatible knowledge representation. Incompatible knowledge representation could have the further consequence of preventing labeled peers from teaming up with other knowledge-comparable peers. Our hypothesis is that to

be able to extract the most consensus possible among related peers we should focus not only on the cooperative peers, which most of the existing works do, but also on uncooperative peers as well. Observing the fact that **consensus building** technique used in semantic mapping is similar to the *majority voting* technique used in the designing fault-tolerance hardware and software systems opens up a new avenue for consensus construction research. We believe that there are opportunities to build a more robust semantic consensus systems using other applied majority voting and fault-tolerance techniques. The need for **fault-tolerance** capability of software have been determined by the fact that the real-world applications require a highly reliable, continuously available, safe and accurate softwares. Therefore, we believe that semantic mapping systems should be constructed with the built-in capability to tolerate faults.

The rest of this paper is organized as follows: in section 2, we look at the temporary fault issue in consensus building. In section 3, we describe the similarity between majority voting and consensus building and its effect on future research. In section 4, we discuss the construction of the consensus-based systems with fault-tolerance capabilities and finally in section 5, the conclusion of the paper with some future research directions are presented.

2 Semantic Mapping Faults

Consensus formation in a P2P network is identified by the greatest(lowest) possible common knowledge (GCK) among all the peers of the network. Current consensus building procedure obeys the following steps: Everytime a peer P encounters another peer $\bar{P}$ that could handle its request (i.e. a peer with similar semantic knowledge representation), that peer $\bar{P}$ will be added to the list of related peers to peer P. This knowledge will be used for the subsequent cooperation and encounters, for example, when answering a query. However, if a peer P meets another peer $\hat{P}$ with a different semantic knowledge representation, that peer $\hat{P}$ will not be considered for subsequent tasks [1, 7, 4]. Two key elements in the described consensus formation are semantic *mapping operation* and *peers participation*. In other words, correct mapping relates peers with semantic comparable concepts. Most of the existing works on concept mapping are concern with the *precision* of mapping. It is implemented as threshold variable δ and the user of the system decides on its value at a run time. The δ and GCK are inversely related, i.e. the higher the δ the lower the result of GCK and vise-versa. The following represents this relation.

$$f(GCK) : 1/\delta \qquad \text{Eq.1}$$

Others such as [8, 7] tried to improve the result of GCK by increasing the number of peers ρ who participate in consensus formation. This is done by accepting partial results. Hence Eq.1 could be rewritten as follows:

$$'f(GCK) : 1/\delta, P(\rho) \qquad \text{Eq.2}$$

where $'f(GCK)$ represents the improved $f(GCK)$ and $P(\rho)$ is the probability of extra peers participating in consensus formation because of their ability to provide partial results to the query. In the described consensus formation, peers' past collaborations used for future decisions on further collaborations. We will rewrite Eq.2 to reflect this reality where χ represents cooperative peers.

$$'f(GCK) : (1/\delta, P(\rho))[\chi] \qquad \text{Eq.3}$$

One shortcoming with the above described method is that, once a peer is unable to fulfill a particular request, for example answering a query, it will not be considered for the subsequent tasks. In other words, the described method sees the peers' inability to answer a query as a permanent fault - permanent non-cooperation. We see this as a deficiency because peers' inability to answer a query could be a result of temporary dis-connection, noise or incompetency to answer a particular request. Therefore, writing the Eq.3 to account for temporary uncooperative peers ϵ yields the following relation.

$$''f(GCK) : (1/\delta, P(\rho))[\chi, \epsilon] \qquad \text{Eq.4}$$

To be able to include temporary uncooperative peers for future tasks, we have to distinguish between permanent and temporary uncooperative peers. Classification of different types of faults along the temporal dimension: transient, intermittent and permanent is the enabling mechanism which facilitates the differentiation between permanent and temporary uncooperative peers. A detailed description of fault types, fault source and fault classification will be considered in future work.

3 Constructing Semantic consensus

In this section, we highlight some similarities between concepts and techniques used in two different fields and those used in the semantic mapping

process, i.e., consensus formation. These fields are: i. theory of cooperation and evolution and ii. fault-tolerant computing systems. The possibility of misunderstanding or misimplementation between players, i.e. noise, has been heavily studied in **Cooperation and Evolution** fields [2, 13, 11]. Strategies applied to bring autonomous selfish peers to a *consensus* with existing noise in the system is an attractive proposition for solving semantic mapping problem with existing faults in the process. We believe that there are similarities between handling faults in consensus building and coping with noise in autonomous agent cooperation. We see noise as fault, more specifically, as a transient fault. Therefore, strategies for coping with noise in agent cooperation could be adapted to tolerate faults in consensus formation.

Majority voting is used to design **Fault-Tolerant computing systems** and it has similarity with techniques used by [1, 7, 8, 5, 12] to perform mapping and to eliminate the disambiguation between concepts, i.e., the *consensus* formation technique. Lets consider both the majority voting, consensus and their similarities in more details.

In the **consensus based system** concepts are translated along the query translation or query propagation path. Hence, if the semantic of the concepts are preserved along the query propagation path the query yields a correct (consensus) answer. We could restate this as follow:

For every query to yield an acceptable answer, the translation of the query element semantics have to be approved by multiple peers. This could be considered as a form of a voting.

Please note that, not every consensus answer is a correct answer. There is a possibility that even when several peers reach a consensus about a particular query answer their conclusion might not to be the correct one when compared to some predefined or known facts.

Figure 1 represents the consensus based system where each node represents a peer P and each directed edge $M_{i,j}$ represents a query mapping from source peer P_i to target query P_j. Each cycle in the graph, $P_{i1}, P_{i2}.....P_j, P_{i1}$ represents the query translation path. The selection (deselection) of a query result among multiple results returned from different translation paths by *query initiator* is an approve (disapprove) to the voting decision made by different peers on the translation path. A dished circle in Fig. 1 represents one circle, i.e. translation query path, which peers on the path might reach a consensus.

There are other types of systems such as those described in [5, 12] that do not use *translation* to achieve consensus. Here, consensus is achieved through *counting* the number of times a *concept* or the *relation* between two concepts appears among different ontologies. We see that both described methods, translation and reinforcement, uses the notion of voting to reach consensus.

The **majority voting** techniques is a well-known technique used to determine a consensus result from the results delivered by multiple computation sources. We will concentrate on the TMR (Triple Modular Redundancy) majority voting technique for its simplicity. The TMR system is based on using extra hardware components. More specifically, the TMR system uses three components in place of one component. The TMR system also has an extra component called *Voter*. The Voter is the place where the voting on the different results takes place, i.e., consensus made. The main idea of this technique is that the system tries to build a consensus result from three results [10]. This technique is used to prevent the computation process from relying on a single result. Figure 2 is an illustration for this technique where three peers (components) produce data and a voter combine their output. From figure 2 we can notice that the role of the Voter component becomes an essential role and the reliability of entire system now depends upon the reliability of the Voter. We can notice that the same drawback does exist in the consensus technique as well. In Figure 1, the P8 plays this critical role.

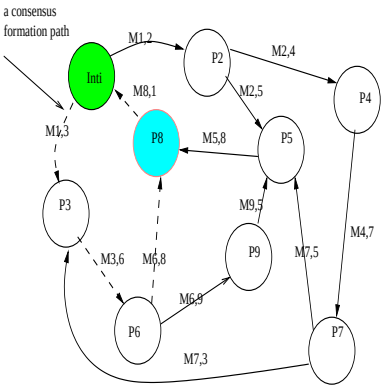

Fig. 1. Query Translation Along Query Paths, P8 plays similar role of voter in Fig. 2

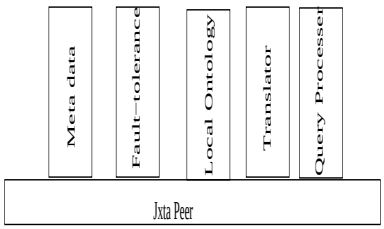

Fig. 2. Triple Modular Redundancy

We could replace T which stands for the Triplicate in the TMR technique by N where $N > 2$. This leads to a system with N components redundancy, the NMR system, instead of the TMR system.

Other voting methods such as plurality voting, threshold voting and weighted k-out-of-n are also used to reach consensus. There are tradeoffs involved in using each of these method. Some methods are more suitable than others for certain applications. For example, the plurality voting method is usually used to determine a winner in a given election. In the plurality method two parameters are important: i. the number of voters that voted for the consensus, i.e. L voters agree and ii. the number of voters which vote for consensus is grater than number of voters which do not vote for consensus, i.e. $M < L$. In other words, the winner does not need to have n/2+1 votes to win, where n is the number of total participants in the voting. The winner needs only L votes where L is the number of participants who voted for the winner and it exceeds the number M were voted against the winner.

It worth to re-emphasis that what we trying to convey here is that the *voting is a form of consensus reaching*. We believe that both the *voting technique* which is used by fault-tolerance systems and a *consensus reaching* used by semantic mapping process have a lot in common. This leads us to the next issues: the feasibility of adapting other forms of voting and fault-tolerance techniques to build consensus and to measure the certainty [1] and the confidences [2] in the consensus reaching. Examples of such techniques include a weighted majority voting, plurality voting and time and information redundancy techniques. We believe that the equivalences between consensus and majority voting will open up new avenues for research. Currently we are researching this issues further.

4 Fault-tolerance

Software components are human made products and since humans are subject to make mistakes, real-world software components cannot guaranteed to be error free. Hence, we should strive to achieve highly reliable, continuously available and safe software [3]. We scrutinized several promising ontology mapping systems and methods for fault-tolerance capability. The examination covered Chatty Web, OBSERVER, Piazza, MAFRA and H-Match. We find out that all of these approaches lack the fault-tolerance capability.

[1] Weight of peers participated in consensus formation

[2] Number of peers participated in consensus formation [14]

We are considering the construction of a consensus-based system with a fault-tolerance capability, i.e. building a system which tolerates faults that remain in the system after its development. A software fault-tolerance capability could be accomplished through various methods including information, component and time redundancy.

The choice of information and time redundancy are more applicable than the component redundancy (N-version programming) in P2P ontology mapping context. This is because P2P network is dynamic environment in which peers enter and leave the network on the fly. Performing multiple computations in such a dynamic environment is difficult and subject to termination, thus depriving peers from opportunities to produce responses. A reasonable alternative would be the duplication of critical variables and/or blocks of code and comparing the output of these code blocks and variables at different stages of the execution of the same program.

The time-redundancy technique could be used to add fault-tolerance capabilities to the consensus formation methods in at least two ways including: i. querying the peer service provider more than once at different times and comparing the obtained results, and ii. preparing a test query for which a querying peer knows the answer. In both of the above cases a querier could directly verify whether the related peers execute correctly [9]. Similarly, information redundancy technique could be used for building consensus formation with fault-tolerance abilities. This is could be done by incorporating extra information about the query and performing checking on the query response for the query added information.

We strongly believe that fault-tolerance capability should be used as a criterion to determine the quality of consensus based systems. The fault tolerant capability is particularly important in critical applications such as security and business applications. This particularity arises from the fact that excluding a useful source of information or a valuable business partner just for a transient type error will have severe consequences on the level of accuracy of the collected information and could jeopardize financial gain for the peers.

5 Conclusion and Future Work

We started by observing that there are several shortcoming of the incremental building semantic consensus among distributed ontologies. We proposed to solve the problem by focusing on three key issues: i. considering cooperative and temporary uncooperative peers in building seman-

tic consensus, ii. adapting other applied voting techniques to semantic mapping reaching and iii. building semantic mapping systems with fault-tolerance capability. Some first steps of these key issues were described. Future works include: i. implementing a bottom-up semantic consensus system with ability to tolerate the non-permanent faults. ii. exploring and adapting some new techniques to build a robust semantic consensus.

References

1. K. Aberer and P. Cudre-Mauroux and M. Hauswirth. Start making sense: The Chatty Web approach for global semantic agreements. In *Journal of Web Semantics*, 1(1): 89-114, 2003.
2. R. Axelrod. The Complexity of Cooperation. Princeton University Press, 1997.
3. M. R. Lyu. Software Fault Tolerance. *Wiley publishing*, 1995.
4. S. Castano, A. Ferrara, S. Montanelli. H-Match: an Algorithm for Dynamically Matching Ontologies in Peer-based Systems. In *Proc. of the 1st VLDB Int. Workshop on Semantic Web and Databases (SWDB)*, P: 231-250, 2003.
5. P. Fergus, A. Mingkhwan, M. Merabti,and M. Hanneghan . Distributed emergent semantics in P2P networks. In *Proc. of the Second IASTED International Conference on Information and Knowledge Sharing*, P: 75-82, 2003.
6. A. Gomez-Perez, M. Fernandez-Lopez and and O. Corcho. Ontological Engineering. Springer publishing, 2003.
7. A. Halevy, Z. Ives, P. Mork, and I. Tatarinov. Piazza: Mediation and integration infrastructure for semantic web data. In *proceedings of the International World-Wide Web Conference WWW-03*, 2003.
8. E. Mena, A. Illarramendi, V. Kashyap and A. Sheth. OBSERVER: an approach for query processing in global information systems based on interpretation across pre-existing ontologies. In*Distributed and Parallel Databases*, 8(2):223-71, 2000.
9. E. Papalilo, T. Friese, M. Smith, B. Freisleben. Trust Shaping: Adapting Trust Establishment and Management to Application Requirements in a Service-Oriented Grid Environment In *Proceedings of the 4th International Conference on Grid and Cooperative Computing (GCC)*, pp. 47-58, LNCS 3795, 2005.
10. D. K. Paradhan. Fault-Tolerant Computing System Design. *Prentice-Hall PTR* publication, 1996.
11. B. Sainty. Achieving greater cooperation in a noisy prisoner's dilemma: an experimental investigation In *Journal of Economic Behavior and Organization*, 39(4):421-435, 1999.
12. L.M. Stephens, M.N. Huhns. Consensus ontologies. Reconciling the semantics of Web pages and agents. In *IEEE Internet Computing*, 5(5): 92-95, 2001.
13. J. Wu and R. Axelrod. How to Cope with Noise in the Iterated Prisoner's Dilemma. In *Journal of Conflict Resolution*, 39(1): 183-189, 1995.
14. S. Yacoub, X. Lin, S. Simske, and J. Burns. Automating the analysis of voting systems. In *14th International Symposium on Software Reliability Engineering*, p 203-214, 2003.