Collaborative Understanding of Distributed Ontologies in a Multiagent Framework: Design and Experiments

Leen-Kiat Soh
Computer Science and Engineering
University of Nebraska
115 Ferguson Hall
Lincoln, NE
(402) 472-6738

lksoh@cse.unl.edu

ABSTRACT

In this paper, we describe our work-in-progress with collaborative understanding of distributed ontologies in a multiagent framework. As reported earlier, the objective of this framework is to improve communication and understanding among the agents while preserving agent autonomy. Each agent maintains a dictionary for its own ontology and a translation table. Our current work has focused on how neighborhood profiling, the translation tables, and query experience influence the collaborative activities among the agents. We have built an infrastructure prototype and conducted a series of comprehensive experiments from the viewpoint of agent executions, query scenarios, and translation credibility values. The specific goals of our analyses are to investigate (a) the learning of useful neighbors for sharing queries, (b) the efficiency of query handling in different real-time scenarios and with different resource constraints (such as the number of threads and available translations), and (c) the effects of different concepts and query demands on collaborative understanding. This paper reports on the results that we have collected so far.

Keywords

Multiagent systems, distributed ontology learning, dynamic profiling

1. INTRODUCTION

In our previous work, we described a distributed ontology learning framework in a multiagent environment [1]. There are two ways that an agent can learn to improve its ontology. First, users can teach them—by supplying a list of words and what the classifying concepts are for that list of words. Second, an agent can learn through its interactions with its neighbors. As a result, each agent learns its own concepts based on its experiences and specialties. When a new concept arrives, the agent needs to incorporate it into its dictionary and its translation table. This is supported by three important components: conceptual learning, translation, and interpretation, with a Dempster-Shafer belief system [2].

Our research focuses on developing and analyzing the operational components of our framework, applied to a document retrieval problem. Each agent interacts with a user who submits queries based on keywords. These keywords are known as concepts in the agents. The goal of this problem is satisfy as many queries as possible and as well as possible. An agent may turn to its neighbors for help. Thus, this collaboration facilitates the distributed ontology learning.

To improve communication, these agents must be able to understand each other. Thus, our research goals are to (1) promote understanding among agents of a community, thus reducing communication costs and inter-agent traffic, (2) improve cooperation among neighbors of a community, thus enhancing the strength (productivity, effectiveness, efficiency) of a neighborhood and supporting the distributed effort of the community, (3) encourage pluralism and decentralization within a multi-agent community—specialization of agents of a community since each agent can rely on its neighbors for tasks not covered by its capabilities, and (4) enable collaborative learning to improve the throughput of the community, the intelligence in communication and task allocation, the self-organization within the community, and integrity of the community.

At the current phase of our research, the objective is to understand how collaborative understanding of distributed ontologies is impacted by operational issues such as queries, the number of communication threads, the variability within the translation tables and so on. Therefore, in this paper, we focus on the operational design of our infrastructure and the investigations on how neighborhood profiling, translation tables, and query experience influence the relationship among collaborative agents. Our experiments are aimed at studying (a) the learning of useful neighbors for sharing queries, (b) the efficiency of query handling in different real-time scenarios and with different resource constraints, and (c) the effects of different ontological concepts and query demands on collaborative understanding. In Section 2, we briefly outline the methodology of our framework. Then we describe our implementation. Subsequently, we discuss our experiments and results. Finally, we conclude.

2. FRAMEWORK

In our framework, the multiagent system is one in which agents can exchange queries and messages to learn about each other's ontology. To improve the communication and collaboration efficiency, agents determine whether some translation is worth learning, which neighbors to communicate to, how to handle and distribute queries, and how to plan for agent activities. The framework consists of two sets of components. The operational components allow the agents to work together in a multiagent system. The ontological components allow the agents to communicate and understand each other. In this paper, we focus on the operational components of our framework.

When an agent receives a query, it checks the query against its ontology knowledge base. A query comes with a concept name and the number of documents or links desired. If the agent cannot satisfy the query, it will contact its neighbors. If the agent recognizes the concept name but does not have enough documents or links to fulfill the requirement, then it will approach its neighbors to obtain more links. If the agent does not recognize the concept name, then it passes the query to its neighbors. Every agent is equipped with *N* number of *negotiation* threads. For each contact, an agent has to activate one of these threads. So, if an agent does not have available inactive negotiation threads, it will not be able to collaborate with other agents. Hence, even if the agents do understand each other's ontologies, it is possible that due to the query frequency and the resource constraints, the agents may not be able to utilize that understanding to help solve a query problem. Since this collaborative activity requires Please refer to [3] and [1] for details on our original design of operational and ontological components, respectively.

2.1. Operational Components

There are three important operational components: query processing, action planning, and query composition. Note that in our framework, an agent sends out a query to its neighbor when it needs to find some additional links for that some classifying concepts. Agents are required to compose queries as well as they also need to relay or distribute queries to other agents by modifying the queries in their own words. Finally, for the system to be effective, the query distribution and the ontology learning behavior are supported by an action planning component that makes decision based on the agent's environment such as message traffic and neighborhood profile.

2.2. Ontological Components

There are three important ontological components in our framework: conceptual learning, translation, and interpretation. We represent an ontology item as a vector. Each vector consists of the classifying concept and then a list of words describing that concept. A concept may have many different supporting documents. Different concepts may be used to classify the same list of words, resulting in different supporting items. Moreover, for each concept, the agent also learns the description vector, combining all relevant experience cases together. This allows the system to incrementally learn and evolve existing ontologies. Currently, we are still implementing the ontological components and thus will not report further on them in this paper.

3. METHODOLOGY & DESIGN

An agent performs two types of learning. It learns incrementally, refining its concepts whenever there is a new submission. It also learns collaboratively, refining its translation table whenever there is a query that prompts the agent to ask for help from its neighbors. Figure 1 depicts the current status of operational components of an agent in our framework.

As shown in Figure 1, there are nine important modules:

(1) Interface: This module interacts with the user to obtain queries and to provide queried results. Currently, we have (simulated) software users that automatically generate timed queries for our experiments. Each software user submits its queries through a socket connection with the interface.

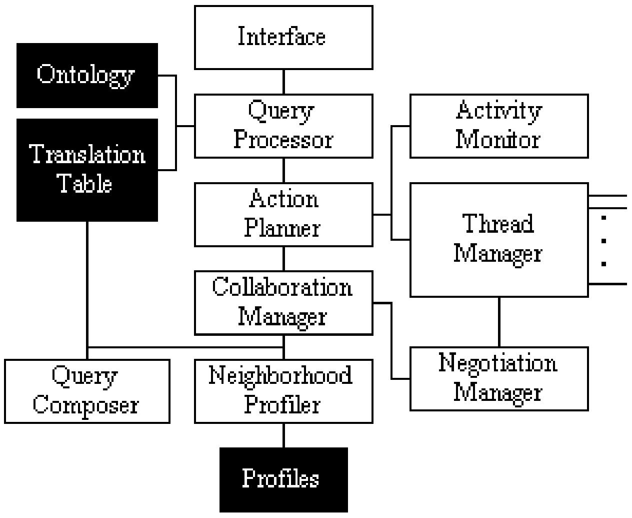

Figure 1 The current design of the operational components of an agent in our framework.

(2) Query Processor: This module receives a query from the Interface module and processes it. It first checks the agent's ontology base. If the query matches one of the concepts in the ontology, the module retrieves the number of links available. If the query does not find a match in the ontology, the module examines its translation table. If there are available translations, that means a collaboration is possible.

(3) Action Planner: This module serves as the main reasoning component of the agent: (a) If the number of internal links satisfies the query, then the action planner simply provides those links to through the Interface module to the user; (b) otherwise, if the agent understands the query and finds available translations, it initiates its collaborative activities; (c) if the agent does not understand the query, it will relay the query to another agent; and (d) finally, if there are no available translations, the link retrieval process stops and the agent reports back to the user. Whether a collaboration is feasible depends on the current status of the agent, as recorded by the Activity Monitor and Thread Manager modules. If the agent does not have enough resources for a collaboration, the link retrieval process terminates.

(4) Collaboration Manager: When the action planner calls for a collaboration, this module takes over. The objective of this module is to form an appropriate group of neighboring agents to approach and distribute the query demands (link allocations) accordingly among them. To design such a collaboration plan, this module relies on the Neighborhood Profiler module, and the translation table. Each neighbor is given a *utility* measure based on the translation credibility value, the past relationship and the current relationship. Note that in our original thesis [1], each translation has a credibility value: two concepts are similar to only a certain degree. The past relationship is the viewpoint of the agent of its neighbor of their interactions (or negotiations in our framework) monitored and stored by the Profiler module. The current relationship is captured by the Activity Monitor module to indicate whether the agent is currently engaged in any negotiations with the particular neighbor. A neighbor has a high utility if the translation credibility of the query in question is high, if the

lation credibility of the query in question is high, if the past relationship is strong, and if there is not any current interaction. The collaboration manager ranks these neighbors based on the utility measure and then assigns the query demands accordingly, with the help of the Query Composer. The manager assigns more links to neighbors with higher utility proportionally to maximize the chance of retrieval success. It also collects the negotiation results, sorts the received links based on the credibility, and filters out low-credibility links when it has more links than desired.

(5) Query Composer: Based on the allocation of query demands, this modules composes a specific query for each neighbor to be approached. As previously mentioned, each query is associated with a link requirement that specifies the number of links desired. A query will also include the name of the originator and a time stamp when it is first generated. If the query is based on a translation, then the translated concept name is used. If the agent does not recognize a concept and needs to relay a query it has received to a neighbor, it simply uses the queried concept directly.

(6) Neighborhood Profiler: The design of this module is based on our work in coalition formation. As we will later in Section 4, we keep track of the past relationship between the agent and each neighbor. The relationship is a composition of four basic numbers: *_numHelp* (the number of times the agent provides help to the neighbor), *_numSuccess* (the number of times the agent successfully solicits help from the neighbor), *_numRequestFrom* (the number of times the agent receives a request from the neighbor), and *_numRequestTo* (the number of times the agent initiates a request to the neighbor) [4]. Based on these numbers, we can derive helpfulness, usefulness, importance, and reliance of each neighbor, from the viewpoint of the agent.

(7) Activity Monitor: This modules keeps track of the activities in a *job vector*—whether the agent is processing a query on its own, or negotiating with other neighbors for more links, or entertaining a request by a neighbor. Each *job* is described with a list of attributes such as the originator, the executor, the task description, the current status, and so on.

(8) Thread Manager: This module housekeeps the threads of the agent. It is a low-level module that activates the threads, updates and monitors the thread activity.

(9) Negotiation Manager: This module manages the negotiation tasks. In our current design, the interaction between two agents does not involve negotiations as the two simply exchange information. However, our long-term plan views negotiation as an important part of ontology interpretation in a distributed environment. Negotiations that are too time consuming, stagnant, or no longer useful will be modified or aborted; negotiations that are successful will be learned; and so on. We will adapt our previous work in reflective negotiations [5] to distributed ontology in this framework.

Together with these nine operational components are three dynamic knowledge or data bases: ontology, translation table, and profiles. The profiles keep track of the relationships between the agent and its neighbors, updating the neighborhood parameters. The ontology is a *dictionary* listing the concepts that the agent knows. Each concept has a list of supporting documents or links. The translation table consists of translations between each concept that the agent knows and its neighbors. Each translation is accompanied with a credibility value. Some neighbors may not have concepts that are similar to a concept that the agent knows and the credibility value for those entries in the translation table is NIL. Table 1 shows an example of a translation table for agent A1. In the example, A1 has four neighbors. It knows of concepts such as "basketball" and "car". For "basketball", it is similar to N1's "NBA" with a credibility of 2.1, N2's "Bball" with a credibility of 1.0, and N4's "Basketball" with a credibility of 3.4. However, it does not have a translation for "basketball" between itself and N3.

Concepts	**N1**	**N2**	**N3**	**N4**
basketball	*NBA* 2.1	*Bball* 1.0	*NIL*	*Basketball* 3.4
car	*NIL*	*Auto* 2.1	*Car* 1.0	*Move* 1.0
…				

Table 1 A translation table example.

4. IMPLEMENTATION

We have implemented all the nine modules of our agent as depicted in Figure 1 in C++. Each agent receives its user queries from a software user through a socket connection, and communicates with other agents through a central relay server module through socket connections as well. Each agent generates and maintains its neighborhood profile during runtime dynamically.

For our experiments, each agent is equipped with a translation table right from the start. Note that in our original distributed ontology framework [1], the entries in a translation table is learned over time based on the experience of each agent. In this paper, we focus on the operational design of collaborative understanding of distributed ontologies and assume that each agent has a translation table to begin with.

In addition, each agent is equipped with an ontology database. This database lists all the concept terms that an agent knows. For each concept, there is a list of links (or documents) that are examples that illustrate the concept. Indeed, when interpreting two concepts, we simply compare the similarities of the two lists of links supporting the two concepts. Currently, we are building this interpretation module.

5. DISCUSSION OF RESULTS

We have performed a comprehensive set of experiments. In this Section, we will describe our experimental setup and then discuss the results.

5.1. Experimental Setup

Here is the setup of our experiments:

There are five agents supporting a software user each. All agents are neighbors and can communicate among themselves. All five agents and their threads are run on the same CPU.

Every agent has a unique set of nine concepts in its ontology. Each concept has five supporting links.

Each agent has a translation table where each cell of the table indicates the translation between a local concept and a foreign concept in a neighbor and the translation's credibility value. If a translation is not available, we use the symbol NIL.

Each software user has a query configuration file. Thus, instead of manually submitting these queries, the software user simply reads them from the file and sends them to the corresponding agent. For each query in a configuration file there are (a) a cycle number, (b) the queried concept name, and (c) the number of link desired. The cycle number indicates when the query will be submitted to the agent. (A cycle's time varies as this measures a loop of activities of an agent.) Each configuration file has about 300 cycles, and two batches of exactly the same query scenarios. We want to investigate whether the agents are able to improve in their response time in the second batch after learning how to form collaborations better through neighborhood profiling.

In the first batch of query scenarios,

(1) Cycles 0-10: Every user queries about all different concepts its agent has in the ontology. Each agent is also able to satisfy the query demand on its own. During this segment, each agent does not need to collaborate. All queries across the users are submitted at the same cycles.

(2) Cycles 11-40: Every user queries about all different concepts its agent has in the ontology. However, each agent is not able to fulfill all queries on its own. During this segment, each agent needs to collaborate. All queries across the users are submitted in a staggered manner. User 1 submits all its nine queries first; user 2 submits its queries after 3 cycles; and so on.

(3) Cycles 41-70: Every user queries about all different concepts its agent has in the ontology and each agent is not able to satisfy the queries on its own. Also, the number of links desired for every query is twice that in the second segment. Extensive collaborations are needed. Queries are also staggered in this segment.

(4) Cycles 71-80: Every user queries about different concepts its agent does *not* have in its ontology. This forces the agent to relay the queries to other neighboring agents. Queries are packed and not staggered in this segment.

(5) Cycles 81-110: The setup of this segment is similar to that during cycles 11-40, but with concepts that each agent does not have in its ontology. Queries are staggered.

(6) Cycles 111-120: During this segment, two users query about concepts that their agents do not have in their respective ontologies, two users query about only some concepts that their agents do not have in their respective ontologies, and one user queries about concepts that its agent has in its ontology. The queried number of links is small and no negotiations are needed.

The second batch starts around Cycle 150, and repeats the above query scenarios. Figure 2 gives a brief overview of our query scenarios.

Our query scenarios are staggered and packed to investigate the response behaviors of the agents. Since the number of negotiation threads is limited for each agent, packed queries with high link demands may lead to only partial link retrievals. Our query scenarios also come with low and high link demands. Low link demands do not require or require fewer collaborations, while high link demands prompt the agents to plan for collaborative actions. Finally, an agent may or may not know some of the queried concepts. The agent's ontology specifies this knowledge. When an agent knows the queried concept, it has more options, approaching different neighbors for help. When it does not know the queried concept, then it shifts the responsibility to one of the neighbors, essentially making itself a relay station.

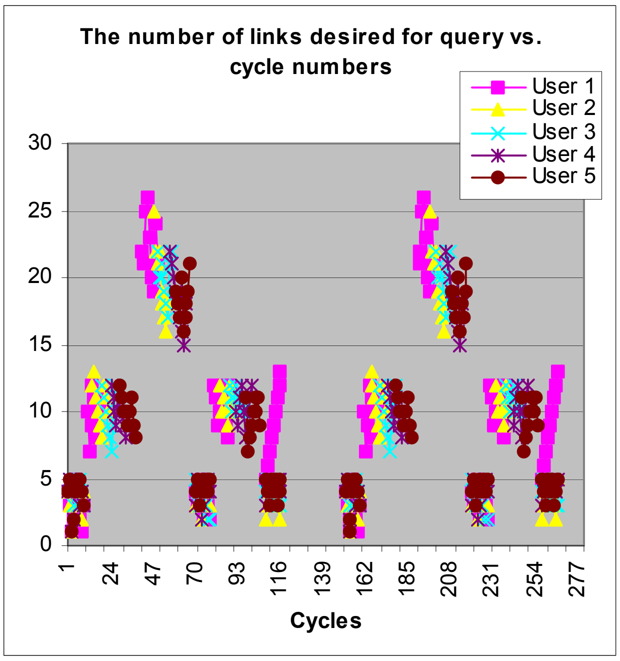

Figure 2 The number of links for the queries submitted by the software users to the agents for each cycle.

Given the above query scenarios, we further vary two sets of parameters: the number of negotiation threads and the credibility values in the translation tables. We vary the number of negotiation threads between 0 and 5. When the number is 0, the agents do not have collaborative capabilities since they can contact other agents. When the number is 5, an agent can simultaneously conduct 5 negotiations. Thus, this number directly impacts the resources that the agents have to collaborate to satisfy queries. This is relevant to *operational* constraints. There are also six sets of translation tables. In the first set, all credibility values of all translations are above zero. In this situation, every concept that one agent knows has four translations. In the second set, one of the agents has what term as a "narrow ontology". That is, its translation table contains many NIL translations, above 50%. (See Table 1.) In the third set, two agents have narrow ontologies. In the fourth set, three agents do; in the fifth set, four agents do; finally, all agents do. With these sets, we want to see how successful the agents are in satisfying high-demand queries. This is relevant to *ontological* constraints.

Given the six different numbers of negotiation threads and six sets of translation tables, we carry out a total of 36 runs using the same set of query scenarios.

5.2. Parameters Collected

Our long term effort is to study the learning of distributed ontologies, including the self-modification of the translation credibility values, and the exchange of ontological knowledge among the agents. However, at the time of the writing of this paper, we have not conducted a focused analysis on that.

Instead, our current experiments concentrate on two sets of parameters:

(1) Neighborhood Profile Parameters: For each neighbor, an agent collects parameters documenting the outcomes of their past

interactions. These parameters are also used in the computation of a neighbor's utility measure, as described in Section 3. Table 2 documents the definitions of these parameters.

(2) Query Result Parameters: For each query, an agent collects parameters documenting the characteristics of the query and the query outcome. Table 3 documents the definitions of these parameters.

Parameters	**Definitions**
_numSuccess	The number of successful negotiations that the agent has initiated to neighbor *i*
_numHelp	The number of successful negotiations that the agent has received from the neighbor *i*
_numRequestTo	The total number of negotiations that the agent has initiated to the neighbor *i*
_numRequestFrom	The total number of negotiation requests that the agent has received from neighbor *i*
_successRate	*_numSuccess/ _numRequestTo*
_helpRate	*_numHelp/ _numRequestFrom*
_requestToRate	*_numRequestTo/_totalRequestTo* where *_totalRequestTo* is the sum of all negotiations that the agent has initiated
_requestFromRate	Presently this number is not updated, as our negotiation design does not incorporate the argumentative reasoning in [5]. However, we plan to re-visit this number in the future once the interpretation module is completed. This number tells the agent how much neighbor *i* relies on the agent

Table 2 Neighborhood profile parameters.

Parameters	**Definitions**
_originator	The originator of the query, either from a software user (ID) or another agent
_cycle	The cycle ID when the query is first generated
_numLinksDesired	The number of links desired by the query
_numLinksRetrieved	The number of links retrieved at the end of the retrieval process and presented to the user, always smaller than *_numLinksDesired*
_conceptName	The query keyword
_successQuality	*numLinksRetrieved/numLinksDesired*
_duration	The actual elapsed time between the receipt of a query and the presentation of the query results to the user
_listLinks	The list of links retrieved and presented to the user at the end of the retrieval process

Table 3 Query result parameters.

5.3. Results

Our overall, longterm plan of analysis aims at analyzing the results at eight different levels. At level 0, we derive an overview of the correctness and assessment of the results. At level 1, we want to analyze the agents' retrieval quality in the two similar batches of queries. At level 2, we aim to compare across the agents and see whether there are significant patterns. At level 3, we want to look into the retrieval results of each segment. Note that each segment has its unique set of characteristics (Section 5.1). At level 4, we want to investigate the role of the concepts. Some concepts may have few supporting links and some have many. At level 5, we will analyze the impact of different queries on the quality of the retrieved results. A query with a high-link demand may not necessary result in poorer results than one with a low-link demand. At level 6, we plan to examine closely the impact of the translation tables with narrow and wide ontologies, and how distributed ontology learning may help improve the tables for better query effectiveness. Finally, at level 7, we will study the operational impact of the threads as a constrained resource.

In this paper, we report on some preliminary level-0 analyses. Figures 3-7 show the graphs of *_successQuality* vs. the number of threads for each software user. Here are some observations:

(1) The average *_successQuality* of a user's queries increases as expected when the number of threads increases. This is because for high-demand queries that call for collaborations, the agent has more resources (i.e., negotiation threads) to use.

(2) The average *_successQuality* of a user's queries drops significantly whenever the corresponding agent has a narrow ontology. However, the drops are more significant when the number of threads is smaller. This indicates that link retrieval, in our application, benefits from the collaborative distributed ontology design. When agents are able to collaborate more often, the *_successQuality* of a query is higher.

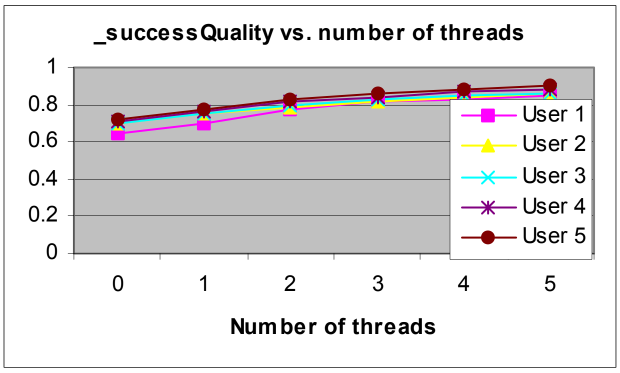

Figure 3 The average *_successQuality* value of each user's queries vs. the number of threads where no agents have narrow ontologies.

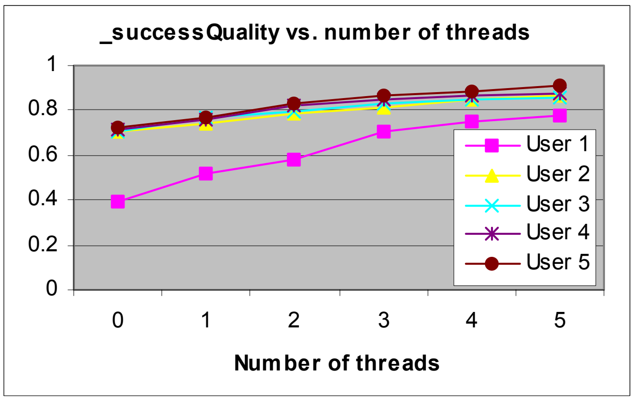

Figure 4 The average *_successQuality* value of each user's queries vs. the number of threads where agent 1 has narrow ontology.

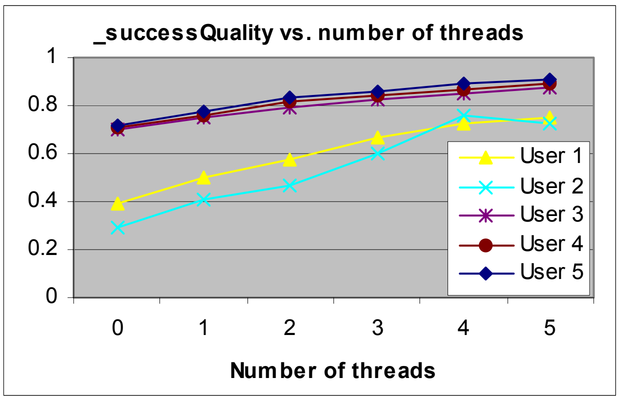

Figure 5 The average *_successQuality* value of each user's queries vs. the number of threads where agents 1 and 2 have narrow ontologies.

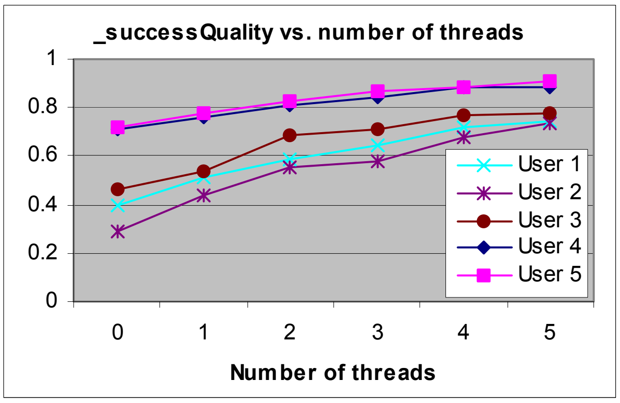

Figure 6 The average *_successQuality* value of each user's queries vs. the number of threads where agents 1, 2, and 3 have narrow ontologies.

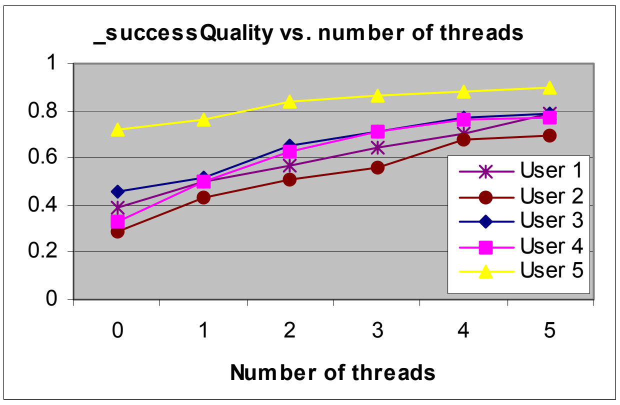

Figure 7 The average *_successQuality* value of each user's queries vs. the number of threads where agents 1, 2, and 3 have narrow ontologies.

(3) Figure 8 shows the average *_successQuality* and standard deviation of all queries for each number of threads. As we can see, with a higher number of negotiation threads, queries are satisfied more successfully (high average values), and also more consistently (low standard deviation values).

(4) Figure 9 shows the average *_successQuality* for agents with narrow ontologies and those with non-narrow ones. Note that if agent *A1* does not have a translation for mapping its concept name *C1* to any of agent *A2*'s, that does not necessarily mean that *A2* does not have a translation mapping one of its concepts to *A1*'s concept name *C1*. This is by design as we ultimately aim to show how collaborative agents can learn new translations or refine old ones as they help each other in satisfying queries. As observed, the number of narrow ontologies does not impact the success quality. From the operational point of view, this is unexpected. When the number of narrow ontologies within the multiagent system increases, we expect that more agents would *relay* queries to their neighbor, and that would cause the negotiation threads to be used more frequently.

(5) Figure 10 shows the average *_duration* (in seconds) for each query to be processed and presented back to software user 1 (by only agent 1), for different numbers of negotiation threads. As observed, when the number of threads increases, it takes longer for a query to be responded to. This observation was not anticipated. However, upon further analysis, we realize the following. When an agent has more threads, not only it can approach more neighbors for help, but it also receives more requests for help from other agents. As a result, the agent manages more tasks and slows down its processes for retrieving and supplying results to the software users. This indicates an oversight in our design with regards to the efficiency of our implementation. We are currently reviewing our program code to pinpoint the places where we could optimize the multi-threaded programming portion. We will also perform the same analysis on all other software users and agents to see whether the same patterns are observed as well.

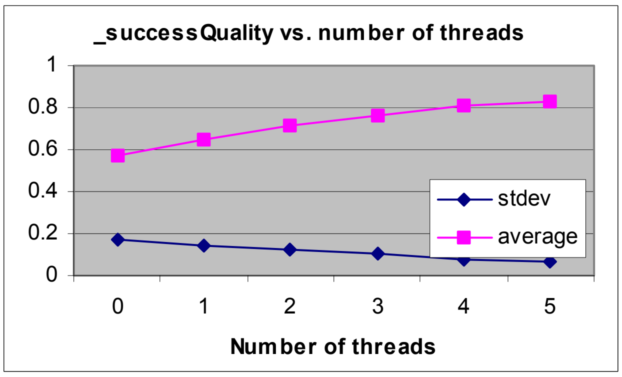

Figure 8 The average and standard deviation of the *_successQuality* for all users vs. the number of threads.

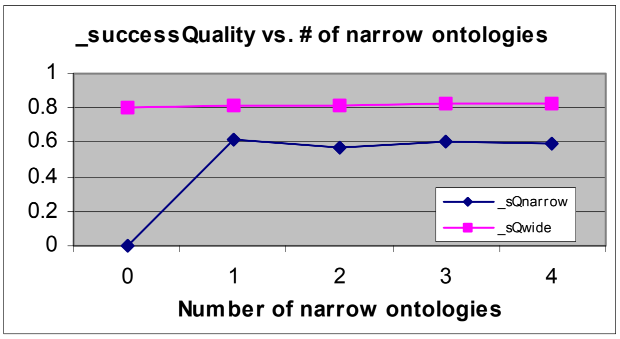

Figure 9 The average *_successQuality* for agents with narrow ontologies and agents with non-narrow ontologies. The *_sQnarrow* value for the 0 narrow ontologies is not applicable.

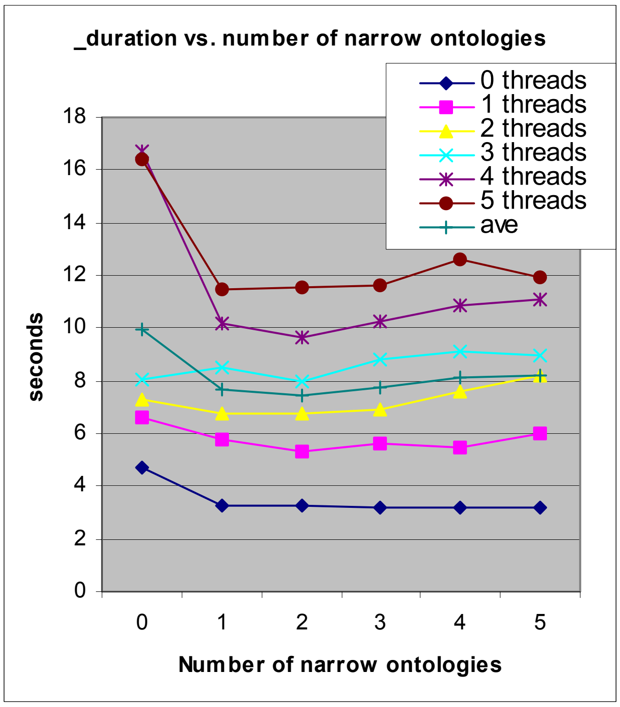

Figure 10 The average *_duration* for agent 1, for different numbers of threads, vs. the number of narrow ontologies.

(6) From Figure 10, the average *_duration* values for the different numbers of narrow ontologies are 9.96, 7.66, 7.41, 7.73, 8.15, and 8.24 seconds, respectively. The multiagent system where the agents do not narrow ontologies, unexpectedly, have the highest average *_duration* value. This value drops, has a minimum when the number of narrow ontologies is two, and then climbs up consistently for the next three sets. We are currently investigating the reasons behind this curve, to at least explain the data of the 0-narrow ontology case. Coupling the above observation with that in from Figure 9, we see that when the number of narrow ontologies increases (starting from number = 2), even though the *_successQuality* value remains mostly the same, the *_duration* value starts to dip. This clarifies somewhat our study.

(7) Figure 11 shows the average neighbor profile of agent 1 of its neighbors: *_numSuccess*, *_numHelp*, *_numRequestTo*, and *_numRequestFrom*. The values of *_numHelp* and *_numRequestFrom* are the same; that is, the *_helpRate* is 100%. For this agent 1, the number of times it has requested for help is smaller than the number of times it has entertained other agents' requests. This indicates that the query scenarios tend to invoke collaborations, causing the originating agents to ask for help from many different neighbors. From the graph, we see that the agent approaches more neighbors for help as it has more negotiation threads. However, when the number of threads is 5, the rate levels off just a little, indicating that a convergence may occur when the number of threads is larger than 5. This means that in our current experimental setup, our link demand is still more than what the agents can handle.

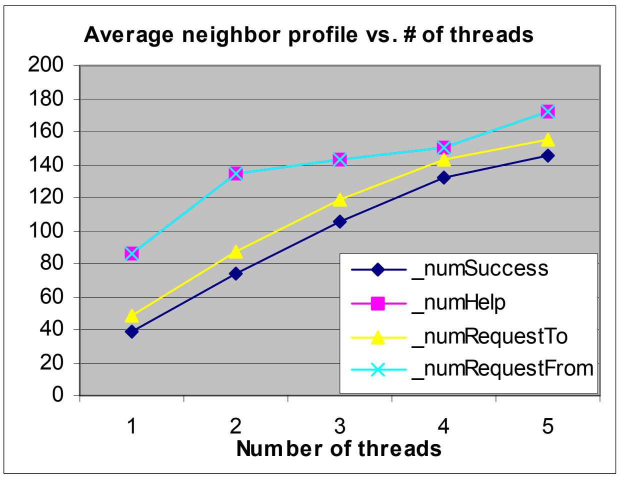

Figure 11 The average neighbor profile for agent 1 of its neighbors vs. the number of threads

(8) Figure 12 shows the average *_successRate* vs. the number of threads available. As observed, the agent is able to negotiate more successfully when the number of threads increases. This is expected since with more threads available, an agent is able to entertain more requests. Coupling this with Figure 11, we see that agent 1 is able to conduct *more* negotiations *more* successfully when the number of threads increases—more effectively and more efficiently. This is a good indicator that would help guide the design of distributed ontology learning in our work.

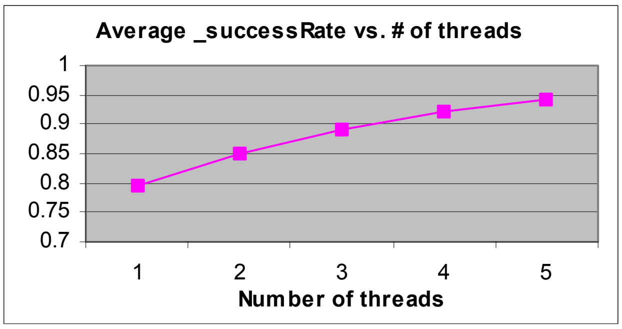

Figure 12 The average neighbor profile for agent 1 of its neighbors vs. the number of threads

(9) Figure 13 shows the *_requestToRate* vs. the number of threads available. As observed, when the number of threads is 1, agent 1 relies on agent 2 (or N1) almost heavily. This is due to the fact that in the beginning of an agent, all neighbors are weighted very

similarly; as a result, the agent will approach the first neighbor that it knows. However, as the number of threads increases, the agent is able to collaborate more with other neighbors. As a result, the reliance on N1 greatly decreases. Meanwhile, the reliance on the other three neighbors steadily increases. This is a good lesson, as we now know that in order for the system to exhibit un-intended bias favoring one neighbor over next, we need to have enough number of threads, laying the groundwork for the distributed ontology learning design of our work.

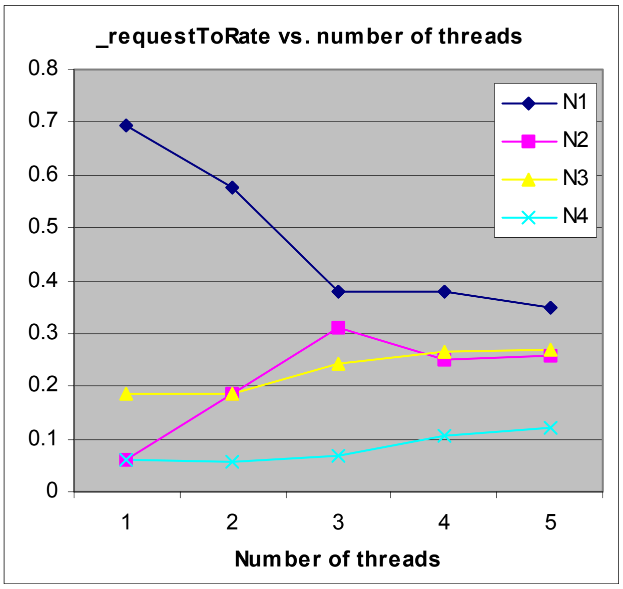

Figure 13 The *_requestToRate* from agent 1 to its neighbors, N1 (agent 2), N2 (agent 3), N3 (agent 4), and N4 (agent 5) vs. the number of threads.

6. CONCLUSIONS

In this paper, we have described our work-in-progress with collaborative understanding of distributed ontologies in a multiagent framework, focusing on the operational components. We have outlined the methodology and design of our framework. The methodology involves building agents with key operational components to support ontological functions such as query processing, query composition, negotiation, and collaboration. We have also briefly discussed our implementation. We have focused mainly on our on-going experiments. We have described our query scenarios, translation tables, and ontologies, as well as two key sets of parameters colleted from our experiments: neighborhood profile and query result parameters. Our experiments have generated a lot of data that we are currently reviewing and investigating. We have reported on some preliminary, low-level analyses to give an overall assessment of our system's feasibility and correctness. In general, we see that the number of negotiation threads available to each agent in the system has a key role in determining the *_successQuality* of a query task, the average *_successRate* of a negotiation, and the degree of collaboration among agents. We also see that the number of "narrow" ontologies influences the agents' behaviors negligibly. We plan to look into this finding further.

Our immediate future work includes (1) completing the 7 levels of analyses identified in this paper to analyze our infrastructure, (2) finishing the interpretation module to add complexity into the negotiation protocols, (3) activating the learning mechanism so that the translation credibility values can be revised dynamically, and (4) investigating the usefulness of the utility measure and its impact on the accuracy of translation. For the last item, remember that the utility measure of a neighbor is based on the credibility of the particular translation as well as the agents' relationships. That means, even if a neighbor is very knowledgeable (with high credibility), an agent may not approach that neighbor for help if the *_successRate* is low. As a result, our distributed ontology learning may be biased towards how close two agents have collaborated, and factor in less importantly the actual accuracy of the translation. Thus, in a way, we are addressing a type of *operational* distributed ontology: agents learn ontologies that are useful and credible to them, instead of only learning ontologies that are highly credible to them.

7. ACKNOWLEDGMENT

The author would like to thank JingFei Xu for her programming and running the experiments for this project.

8. REFERENCES

[1] Soh, L.-K. 2002. Multiagent, Distributed Ontology Learning, *Working Notes of the 2nd AAMAS OAS Workshop,* July, Bologna, Italy.

[2] Shafer, G. 1976. *A Mathematical Theory of Evidence*, Princeton, NJ: Princeton University Press.

[3] Soh, L.-K. 2002. A Mutliagent Framework for Collaborative Conceptual Learning Using a Dempster-Shafer Belief System, *Working Notes of AAAI Spring Symposium on Collaborative Learning Agents*, Stanford, CA, Mar 25-27, pp. 9-16.

[4] Soh, L.-K. and Tsatsoulis, C. 2002. Satisficing Coalition Formation among Agents, *Proceedings of AAMAAS'02*, July, Bologna, Italy.

[5] Soh, L.-K. and Tsatsoulis, C. 2002. Reflective Negotiating Agents for Real-Time Multisensor Target Tracking, in *Proceedings of IJCAI'01*, Seattle, WA, Aug 6-11, pp. 1121-1127.