A Cyc-based Multi-agent System

Jingwen HE, Hokyin LAI, Huaiqing WANG
Department of Information Systems, City University of Hong Kong, Hong Kong SAR, China
{jingwenhe2, isjlai}@student.cityu.edu.hk iswang@cityu.edu.hk

Abstract

Multi-agent Systems (MAS) are systems in which many intelligent agents interact with each other to accomplish a common goal, e.g. solving a complicated problem in a distributed environment. Domain specific knowledge can no longer solely support reasoning in MAS, whereas common sense knowledge becomes more critical to the reasoning quality, especially when e-Commerce has become more popular and more merchants are getting excited to the worldwide market. Incorporating common sense knowledge to the MAS is on edge. However, the use of common sense knowledge induces implementation dilemmas. For example, OpenCyc is not only a common sense knowledge base, but also has its own inference engine. Developers have to determine whether to use the existing inference engine or to use the one that OpenCyc provided. In this paper, four approaches to incorporate common sense knowledge to MAS are proposed and evaluated, and we finally advocate our favorite.

1. Introduction

Multi-agent Systems (MAS) are widely used to solve complicated problems. The social capabilities of intelligent agents enable themselves to work together to solve a common problem that is beyond the individual capabilities. Most intelligent agents have a basic reasoning capability. This capability can be supported in a form of a rule engine, e.g. Jess rule engine. However, the facts are only limited to be domain specific at the moment.

Different people typically have different answers to the same question, and have different comments on the same statement [5]. However, most of the current MAS can only solve problems in an objective manner by following the limited number of rules and facts that the users have provided. These rules and facts are designed to be domain-specific. In the past ten years, e-Commerce has become more popular and a lot of merchants took this chance to extend their businesses to other corners of the world. However, most of them are not aware of the importance of "local" common knowledge in different countries. Most current MAS cannot deal with heterogeneous types of knowledge generically at the same time.

In order to enforce the reasoning capability of the MAS, a lot of research [1, 5] addressed the need to introduce common knowledge to the existing MAS. However, there exists a challenge that most common knowledge bases are extremely huge in size and are difficult to be manipulated.

The objective of this paper is to find out the best way to use the common sense knowledge in multi-agent system architectures. Four approaches to integrate the multi-agent technology with common knowledge bases have been developed for comparison.

OpenCyc is one of the popular common knowledge bases with their own inference engines. It is popular not only because of its well-organized assertions, but also due to its design. The design of OpenCyc allows itself to "learn" new knowledge continuously and it provides various interfaces to external applications. OpenCyc is very flexible. Therefore, the proposed approaches have adopted it as a demonstration.

In the next section, background research on multi-agent architecture, common sense knowledge bases and Cyc, and Cyc-based multi-agent system are discussed. In section 3, four approaches to integrate the multi-agent technology with the OpenCyc knowledge base are discussed. In section 4, the performance evaluations on the four approaches are discussed. The final section considers future directions for the ideas in this paper.

2. Background research

2.1 Multi-agent architecture

1530-1605/08 $25.00 © 2008 IEEE

Multi-agent System (MAS) is defined as a loosely coupled network of problem solvers (i.e. intelligent agents) which work together to solve problems that are beyond the individual capabilities or knowledge.

Most MAS are developed based on a standardized framework called Foundation for Intelligent Physical Agent (FIPA). The FIPA supports interoperability between agents and agent-based applications. Each individual agent has to register its presence in the Agent Directory. Message Transport Service supports the sending and receiving of transport messages between agents. Agent Communication Language (ACL) is the message structure that FIPA-compliant MAS use to support the collaboration between agents.

A lot of Multi-agent system development tools are available in the market, e.g. SAGE, OSLO-Software, AgentScape, Aglets, JADE etc. The most recently used one is JADE. JADE (Java Agent Development Framework) is a FIPA-compliant multi-agent platform. Each JADE run-time environment is called a container. A group of containers makes up a platform. Multiple containers can run simultaneously. Agents from different containers can still communicate with each other by using a method called Agent Migration. Before the agents can communicate with each other on the same platform or remote platforms, they have to register in the platform directory. Each agent belongs to a container with the FIPA specific Agent Management Service (AMS), the Directory Facilitator (DF), and the Agent Communication Channel (ACC) respectively.

To enable reasoning functions in MAS, most developers have incorporated Jess into the MAS. Jess is a rule engine that is written in Java and is compatible with JADE.

2.2 Common sense knowledge bases and Cyc

Common sense knowledge is what people in common would agree. Commonsense reasoning is a branch of Artificial Intelligence concerned with incorporating human thinking. For some cases, the domain knowledge may not be enough for human beings or computer applications to improve the intelligence in their tasks [6, 8]. Solely relying on the domain knowledge is insufficient to solve complicated problems.

Common knowledge base is a formalized representation of a vast quantity of fundamental human knowledge (i.e. general knowledge): facts, rules of thumb, and heuristics for reasoning about the objects and events of everyday life. The general knowledge can be used to supplement the domain knowledge in order to find out the most optimal solution to the problem.

The concept of common sense knowledge base can be used to further nourish the mechanisms in knowledge retrieval, and knowledge reuse. Several common knowledge bases are available including ConceptNet [7], WordNet [3], and Cyc [2].

Cycorp's OpenCyc is a huge multi-contextual knowledge base with its own inference engine. OpenCyc has been developed since 1984. The knowledge base contains over 1,000,000 assertions as of today and keeps growing. These assertions are designed to capture the notion of "common sense" facts about the world [1].

OpenCyc, as a huge knowledge base, is written in an expressive formal language like 2nd order logic. The knowledge is organized into different microtheories according to its domain. In addition, it utilizes an inference engine that computes through the same kinds of logical deductions that are similar to the human thinking mechanism [1]. That is why OpenCyc can give a unique answer to a particular query which is domain dependent. OpenCyc has the ability to answer some mildly different questions in a right way.

The OpenCyc reasoning system is primarily built on top of seven components. They are (1) Cyc Ontology & Knowledge Base, (2) Knowledge Entry Tools, (3) Reasoning Components, (4) Knowledge Entry Guidelines, (5) Interfacing to External Data sources, (6) Knowledge Formation and Dialog, and (7) Java-based API. With these components, OpenCyc is able to learn new knowledge from the external environment and let external parties use its knowledge. Java-based API is a desirable component that all Java-based applications are able to use the power of OpenCyc.

Due to interoperability of Java-based applications and OpenCyc, the OpenCyc (version 1.0.2) [11] is utilized as the common knowledge base to support a Java-based multi-agent system [9]. OpenCyc (version 1.0.2) is the latest free version of the Cyc ontology. It contains both the repository developed to capture and represent common sense, and the common sense inference engine.

2.3 Cyc-based multi-agent system

In a group environment, individuals interpret the same piece of information differently according to the roles they are playing and the tasks they are performing [5]. Businesses from different parts of the world are closely interconnected, so localized domain-specific knowledge may not be sufficient to

support decision making anymore. For example, the typical salary range for a programmer who is employed in Hong Kong is far different from that in India, so the CEO of a company in Hong Kong may think of outsourcing the programming jobs to India to save the operating costs. Common knowledge like the salary range in different countries has a significant contribution. However, the CEO may consider other issues before he can make the final decision. Likewise, software agents in multi-agent systems increasingly need to deal with heterogeneous sources of "intelligence", not only the domain knowledge, but also the common knowledge.

Previous research [1, 5] on Cyc-based multi-agent systems are mainly focused on discussing the features of Cyc and the expected contribution of Cyc to the existing multi-agent system, but none of them has discussed the implementation details. Cyc is such a huge multi-contextual knowledge base that it contains over 1,000,000 assertions to date. It has a prominent need to investigate on the approaches to better manipulate the knowledge in Cyc.

In this paper, the objective is to find out the best way to use the common sense knowledge in the multi-agent system architecture. Four approaches to integrate the OpenCyc knowledge base with multi-agent technology have been developed for comparison. The ultimate contribution is to take advantage of the flexibility, inter-operability and extensibility of the OpenCyc knowledge base to enrich the intelligence of multi-agent systems.

3. Architecture design

The analysis and design of a novel Cyc-based multi-agent system are described in this section.

As mentioned in section 2, the knowledge used in this system can be categorized into 2 types: the Cyc knowledge (common knowledge), and the Non-Cyc knowledge (domain knowledge). The knowledge representation scheme in a system could be a Cyc knowledge scheme or a Non-Cyc scheme. The knowledge can be transformed from one scheme to the other one (see table 1).

Table 1. Subset of representation knowledge

Represen-tation / Knowledge	Cyc Knowledge format	Non-Cyc Knowledge format
Cyc Knowledge	{I}	{II}
Non-Cyc Knowledge	{III}	{IV}

According to Table 1, four approaches of this prototype architecture can be deduced.

3.1 Approach A

The simplest approach is that each agent has an extended Cyc knowledge base. The agent transforms and inserts domain knowledge into the Cyc knowledge base, and later uses the inference engine of Cyc to reason the results (see Figure 1). Given a knowledge *k*, if the knowledge set of Approach A K_A contains *k*: $k \in K_A$, then $k \in \{I\} \cup \{III\}$. The formats of knowledge in Approach A are Cyc knowledge format, therefore, they are homogeneous.

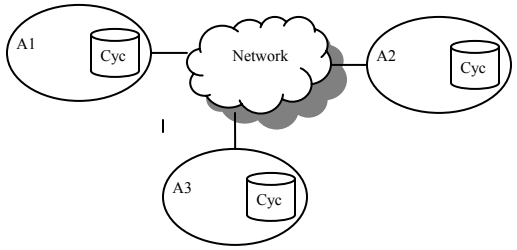

Figure 1. Approach A

3.2 Approach B

Approach B is originated from Approach A, but in a Client/Server mode. One Cyc acts as a knowledge base and an inference engine to provide facilities for each client agent in the server (see Figure 2). We create several micro-theory bases at server for each client agent to store its domain knowledge. The syntax of knowledge in Approach B is the same as that in Approach A. Given a knowledge *k*, if the knowledge set of Approach B K_B contains *k*: $k \in K_B$, then $k \in \{I\} \cup \{III\}$. Therefore, the formats of knowledge in Approach B are also homogeneous. The whole multi-agent system in client part will be much lighter than in Approach A.

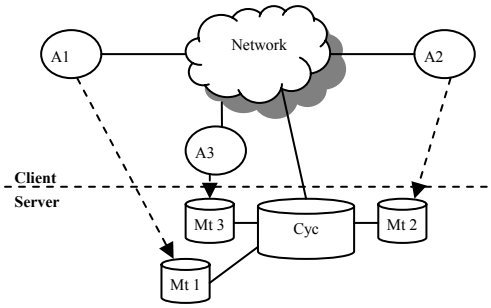

Figure 2. Approach B

3.3 Approach C

In this approach, all the processes will be finished by each agent. Cyc, as well as it inference engine, was utilized not only to extended the knowledge base, but also to provide the reasoning results for Jess inference engine; therefore, the results provided by Cyc will be transformed to Jess language format. Domain knowledge will be inserted into Jess directly (see Fig. 3). In this approach, given a knowledge *k*, if the knowledge set of Approach C K_C contains *k*: $k \in K_C$, then $k \in \{I\} \cup \{IV\}$. Obviously, the formats of knowledge are heterogeneous in the Cyc and Jess.

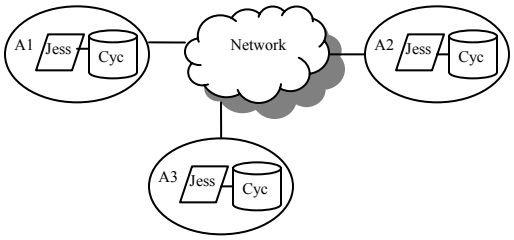

Figure 3. Approach C

3.4 Approach D

In this approach, the common knowledge from Cyc to Jess is loaded and transformed. Domain knowledge will be inserted into Jess directly. Therefore, given the knowledge *k*, if the knowledge set of Approach D K_D contains *k*: $k \in K_D$, then $k \in \{II\} \cup \{IV\}$. All formats of knowledge are homogeneous in this approach, as all of them are in the Jess language format.

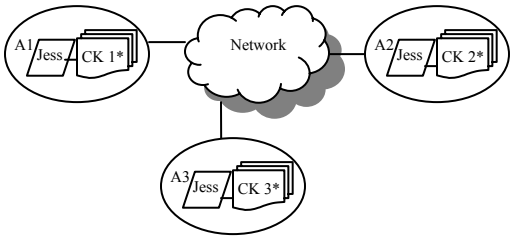

Figure 4. Approach D

CK *: Common Knowledge extended from Cyc

The knowledge and facts in Jess language and Cyc language are compared in Table 2.

Table 2. Knowledge and fact in Jess language and Cyc language

	Jess Language Format	Cyc Language Format
Domain facts	(deftemplate Level (slot level)) (assert Level (level 1)	(#$Level1)
Domain rules	(defrule Degree-Level1 (Degree (degree ?d (eq ?d PhDDegree))) => (assert (Level (level 1))))	(#$genls #$PhDDegree #$Level1)
Common facts	(deftemplate Location (slot location)) (assert Location (location Germany))	(#$Germany)
Common rules	(defrule Location-Area1 (Location (location ?l&: (eq ?l Germany))) => (assert (Area (area EuropeanCountry))))	(#$isa #$Germany #$EuropeanCountry)

4. Evaluation

This section evaluates the performance of the aforementioned approaches. Factors that are used to evaluate the performance of each approach, including storage allocation, runtime memory utilization, agent startup latency, inter-agent communication time, and end-to-end processing time, are discussed [10].

4.1 Experimental environment

The experiments used in comparing the performance are conducted on a client and server approach under Windows XP. The client is connected to the server by a 100Mbps Ethernet-based LAN. No other application is running on the testing workstations and the server throughout the tests.

A case scenario about online education is used to demonstrate the use of the four approaches. An online English writing class is opened to students from all over the world. Based on individual student's education background (i.e. highest academic award and its issuing country), the multi-agent systems are able to assign the student to a class in the suitable level. The four approaches are tested using the same case.

For this case, three intelligent agents (Agent 1, Agent 2, and Agent 3) are used in each setting. Assume that the award issuing countries are categorized into three main areas (Area 1 - European countries, Area 2 - African countries, and Area 3 - Middle Eastern countries). Agent 1 is responsible for determining the area that the award issuing country belongs to and then passing it to Agent 2. The geographic knowledge used by Agent 1 is common knowledge. Similar to the former classification, the academic awards are also classified into three levels (Level 1 – high school diploma or associates degrees, Level 2 – bachelor degree, and Level 3 – master degree or doctoral degree). Agent 2 is responsible for determining the level of the academic award and then reasoning a suitable class for the student by associating the location area from Agent 1 and the education award level from itself. The knowledge and facts used by Agent 2 are domain knowledge which is defined by the administrator. Agent 2 would pass the reasoning result (i.e. the assigned class) to Agent 3. Agent 3 then updates the student records in the database, and sends a notification email to both the instructors and the students.

For Approach A, both the common knowledge and the domain knowledge are represented in OpenCyc format and are loaded into the corresponding agents locally. This approach uses the local OpenCyc inference engine for reasoning. For Approach B, both the common knowledge and the domain knowledge are also represented in OpenCyc format, but are reposited centrally in the server. It also uses the remote OpenCyc inference engine for reasoning. For Approach C, the common knowledge is represented in OpenCyc format and the domain knowledge is represented in Jess format. Jess is a rule engine and a scripting environment written in Java language. Traditionally, most existing Multi-agent systems use Jess to support reasoning. The approach supports heterogeneous knowledge while the former two approaches only support homogeneous knowledge. This approach uses the local Jess inference engine for reasoning. For Approach D, the common knowledge is transformed into Jess format. Meanwhile, the domain knowledge is represented in Jess format, too. Both types of knowledge are loaded into the corresponding agents locally.

4.2 Performance evaluation

In this section, factors that would affect the runtime performance and some runtime figures are discussed.

4.2.1 Storage allocation

Storage management is one critical factor that can affect the processing time of an application. As the size of the agent increases, its response time and its processing performance would be lowered [12]. The sizes of the agents in each approach are summarized in Table 3. Since the programming logics for Agent 3 in all approaches are exactly the same, so it is omitted in the table.

Table 3. Storage allocation

Approach	Storage Requirements	Total
A	Local OpenCyc KB and inference engine for Agent 1 (887MB) + Local OpenCyc KB and inference engine for Agent 2 (887MB)	1,774MB
B	Remote OpenCyc KB and inference engine for Agent 1 and 2 (887MB)	887MB
C	Local Jess rules and facts (2.76KB), Jess reasoning engine (473KB) and OpenCyc KB (887MB) for Agent 1 + Local Jess rules and facts (2.23KB), Jess reasoning engine (473KB), and OpenCyc KB (887MB) for Agent 2	1,774.95MB
D	Local Jess rules and facts (0.6KB), Jess reasoning engine (473KB) and EK 1 (2.19KB) for Agent 1 + Local Jess rules and facts (2.23KB), Jess reasoning engine (473KB) and EK 2 (0KB) for Agent 2	951.11KB

Obviously, Approach D requires the least storage space and Approach C requires the largest amount of storage.

4.2.2 Runtime memory utilization

Memory utilization limits the scalability and efficiency of an application. In experiment, the runtime memory utilization of each approach has been recorded. The figures are summarized in Table 4.

Table 4. Runtime memory utilization

Approach	Runtime Memory Utilization	Total
A	20,772K (Entire logic in Agent 1, 2 and 3) + 126,088K (OpenCyc in Agent 1) + 95,604K (OpenCyc in Agent 2)	242,464K
B	20,720K (Entire logic in Agent 1, 2 and 3) + 126,156K (OpenCyc in Server)	146,876K
C	22,840K (Entire logic in Agent 1, 2, and 3) + 103,956K (OpenCyc in Agent 1) + 95,604K (OpenCyc in Agent 2)	222,400K
D	14,600K (Entire logic in Agent 1, 2, and 3)	14,600K

In terms of runtime memory utilization, Approach D requires the least amount of memory and Approach A requires the most.

4.2.3 Agent startup latency

Agent 1 and Agent 2 in the Approach A, C, and D are required to preload the OpenCyc knowledge, OpenCyc inference engine, Jess rules and facts, or Jess inference engine before they can reason. Agent 1 and Agent 2 in the Approach B are required to establish a connection with the OpenCyc Agent in the server side before they can start reasoning.

For any client and server approach, a client needs to establish a connection to the server before it can evoke the required method. The time to make such a connection is likely to have a strong impact on the overall performance if the client requires communication with the server frequently. There are a few factors that have a prominent impact on the agent startup latency. They are (1) the state of the server, (2) the distribution of the clients and the server, and (3) the collocation of the clients and the server respectively [13]. Since the second and the third factors affect Approach B most and have similar effects to the other three approaches, the latter two factors are not considered. The only way that can affect the startup latency in Approach B is the state of the server. The state of the server is either active or non-active. The time for activating a server is counted as an extra overhead. It takes around 5 to 10 minutes to restart the testing server.

All of the four approaches are running at the risk of incurring the extra overhead, as three of them require the readiness of the OpenCyc knowledge base and the inference engine. In the experiment, the OpenCyc infrastructure requires around 15 to 20 minutes to startup at the first time. After loading once, it takes a shorter time (i.e. around 5 to 10 seconds) to initiate then. The impact is not significant.

The loading time for Jess rules and facts totally depends on the number of Jess rules and facts incurred. However, the number of Jess facts is task-dependent. Take the testing cases as an example, less than 10 rules and facts are used. However, the loading time is still relatively higher than that of the OpenCyc setting.

For the testing case, approach B is highly dependent on the OpenCyc setting in the server side, so it bears the highest risk of incurring this overhead than others. Furthermore, if the client and the server can be located in the same address space, the latency can further diminish. However, in practice, most clients and servers are compiled into separate processes.

4.2.4 Inter-agent communication time

Most multi-agent systems are designed to solve a common problem. Each agent has ability to solve a part of the problem. Due to limited local knowledge, an agent cannot solve the whole problem alone and has to cooperate with each other in a highly distributed environment in most cases [4]. After one agent has solved its own part with its local knowledge, it has to further communicate with other related agents in order to accomplish a common goal. An overhead of inter-agent communications is inevitable. A lot of researchers focus on how to reduce the communication cost and hop count at the design time.

The inter-agent communication time increases if the agent has a lot of tasks to process or needs to deal with a lot of message routing tasks.

Another factor that has significant impact on communication performance is the competition for bandwidth. When more than one agents need to send data over the network using the same wire at the same time, only one message can be transmitted and other messages will be delayed.

In the experiment, twenty identical cases are evaluated continuously using the four approaches. Table 5 contains some of the inter-agent communication time records obtained from the tests, with mean time and standard derivation.

Table 5. Inter-agent communication time

Approach	Agent 1 to Agent 2	Agent 2 to Agent 3	Total
A	403ms +/- 153ms	0.4ms +/- 2ms	403.4ms +/- 155ms
B	9043ms +/- 967ms	0ms +/- 0ms	9043ms +/- 967ms
C	331ms +/- 165ms	5ms +/- 14ms	336ms +/- 159ms
D	205ms +/- 115ms	0.8ms +/- 3ms	206ms +/- 116ms

The communication time for Approach B > Approach A > Approach C > Approach D.

4.2.5 End-to-end processing time

The end-to-end times for one to twenty class allocation requests were measured. The communication time has been incorporated. Table 6 contains some of the timing results obtained from the tests. Each set of measurements is the average of twenty experimental runs, with standard derivation indicated.

Table 6. End-to-end processing

Approach	Agent 1	Agent 2	Agent 3	Total
A	219ms +/- 46ms	15ms +/- 9ms	7ms +/- 5ms	293ms +/- 164ms
B	194ms +/- 98ms	13ms +/- 9ms	4ms +/- 7ms	211ms +/- 93ms
C	261ms +/- 148ms	157ms +/- 298ms	9ms +/- 15ms	427ms +/- 417ms
D	123ms +/- 80ms	120ms +/- 11ms	5ms +/- 8ms	248ms +/- 85ms

For Approach A, Agent 1 and Agent 2 interact with the local OpenCyc KB and use the OpenCyc inference engine to reason. The processing time of Agent 1 is relatively slow. For Approach B, the OpenCyc KB with all microtheories is reposited centrally at the remote server. Theoretically, the runtime performance should be worse than Approach A due to remote connections. However, the performance is still similar to that of the Approach A. One exception is that the standard derivation is

higher due to unstable network effect. For Approach C, Agent 1 and Agent 2 have to load both Jess and OpenCyc settings in Agent 1 and Agent 2 respectively. The processing time is inevitably the highest among the four approaches. For Approach D, related microtheories from the OpenCyc KB have been transformed into Jess rules and facts whereas the OpenCyc inference engine is replaced by the Jess inference engine. Therefore, the reasoning time is shortened in Agent 1 while it is longer in Agent 2. It reveals that all cases can be supported by Jess setting with similar amount of CPU time. One more fact is that using Cyc KB with inference engine takes more CPU time than using Jess Facts with rules for the cases with similar scale.

4.3 Summary of performance evaluation

As a summary, Table 7 is shown. In the table, each approach is ranked from 0 to 3, where '3' means that this approach is the best among the four and '0' means that this approach is the worse among the four.

Table 7. Summary of evaluation result

	Storage Allocation	Runtime Memory Utilization	Agent Startup Latency	Inter-agent Communication time	Processing time	Total Score
Approach A	1	0	1*	1	1	4
Approach B	2	2	0	0	3	7
Approach C	0	1	1*	2	0	4
Approach D	3	3	3	3	2	14

*: Startup Latency of Approach A and C are almost the same

According to the total score calculated based on the five factors, Approach D is the best approach while Approach B is the second best approach. Approach A and C are the least efficient ones.

Approach D is the best approach to include common knowledge into existing MAS. This approach uses heterogeneous types of data by extracting the required common knowledge (i.e. microtheories), and then transforming them into the storage format of the domain knowledge (i.e. Jess rules and facts). It is advised to use the inference engine for domain knowledge to process both types of data.

One of the limitations of Approach D is that it requires extra effort to keep track of the data consistency of the OpenCyc knowledge base and the transformed common knowledge.

5. Conclusion

Domain knowledge can no longer support the reasoning ability in MAS alone. Common knowledge is required to strengthen the reasoning capability. The aims of this paper are to study the compatibility of heterogeneous knowledge in MAS and to determine the most appropriate way to implement this setting.

This research contributes to the amelioration of multi-agent system architectures in various aspects. First, heterogeneous types of knowledge (common knowledge and domain knowledge) can be successfully integrated to further improve the quality of reasoning result in MAS. Second, an ideal approach to use common sense knowledge has been deduced. Four possible approaches to implement the Cyc-based multi-agent system are developed and evaluated. The experiment results illustrates that the most efficient way to use the common sense knowledge in MAS is transforming the knowledge into the storage format of the domain knowledge (i.e. Jess rules and facts) and using the domain inference engine. However, this approach has one limitation. As the requirement of common knowledge increases, the converted knowledge may be insufficient to complete the reasoning task sometime in the future.

Future research in justifying the reliability of the reasoning result warrant to study. For example, approach D can be further investigated in a way that a Cyc knowledge base can be connected as a supplement to common sense knowledge.

6. References

[1] Craig Anken, Nathaniel Gemelli, Petter LaMonica, Robert Mineo, and John Spina, "Intelligent Systems Technology for Higher Level Fusion", *ISIF*, (2002).

[2] Cyc: http://www.cyc.com

[3] Fellbaum, V, "Introduction, in Introduction: Wordnet: An Electronic Lexical DataBase", *the MIT Press*, (1998).

[4] Jiaying Shen, Victor Lesser, and Norman Carver, "Minimizing Communication Cost in a Distributed Bayesian Network Using a Decentralized MDP", *Proceedings of Second International Joint Conference on Autonomous Agents and MultiAgent Systems*, Vol: AAMAS03, (2003), pp. 678 – 685.

[5] Kendall Lister, and Leon Sterling, "Tasks as Context for Intelligent Agents", *Proceedings of the IEEE/WIC International Conference on Intelligent Agent Technology*, (2003).

[6] Kidd, A., "Knowledge acquisition for expert systems: a practical handbook", *Plenum Press*, New York, (1987).

[7] Liu, H. and P. Singh, "ConceptNet: A Practical Commonsense Reasoning Toolkit", *BT Technology Journal*, 22(2004).

[8] M. E. Yahia, R. Mahmod, et al, "Rough neural expert systems", *Expert Systems with Applications*, 18(2000), pp. 87-99.

[9] Matuszek, C., et al, "Searching for common sense: populating Cyc from the web", *in proceedings of the Twentieth National Conference on Artificial Intelligence*, (2005).

[10] Michelle Casagni, and Margaret Lyell, "Comparison of Two Component Frameworks: The FIPA-Compliant Multi-Agent System and The Web-centric J2EE Platform", *Proceedings of the 25th International Conference on Software Engineering ICSE*, (2003).

[11] OpenCyc: http://www.opencyc.com

[12] Rubinstein, M.G., Duarte, O.C.M.B., and Pujolle, G, "Using mobile agent strategies for reducing the response time innetwork management", *Communication Technology Proceedings*, (2000).

[13] Weili Tao, and Shikharesh Majumdar, "Middleware performance analysis: Application level performance optimizations for CORBA-based systems", *Proceedings of the 3rd international workshop on Software and performance WOSP*, (2002).