MASAQ: A Multi-Agent System for Answering Questions Based on an Encyclopedic Knowledge Base[1]

Qiangze Feng, Cungen Cao, Yuefei Sui, Yufei Zheng and Qianfu Qin

Key Laboratory of Intelligent Information Processing, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100080, China
{qzfeng, cgcao,yfsui,yfzheng}@ict.ac.cn

Abstract. In this paper, we present a multi-agent system, called MASAQ, for answering users' queries based on an encyclopedic knowledge base. MASAQ has three major components: (1) a natural language interface; (2) an executable specification language (EASL) for developing multi-agent systems for answering or reasoning about users' queries; (3) an encyclopedic knowledge base covering twenty-one domains. In addition to those features, another novel feature of MASAQ is that the agents can run on the Internet as a distributed system, on a supercomputer as a parallel system, or on a desktop PC as a centralized system.

1 Introduction

A long-term research project was initiated in 1999 to develop a shareable encyclopedic knowledge base from various knowledge sources [6, 7, 10], such as WWW, domain handbooks and encyclopedias. So far, the knowledge base covers 21 domains, including traditional Chinese medicine [10], western medicine [3], history [4], geography [22], biology [14], military [16], music [13], ethnics [20], and archaeology [21].

In this paper, we present a multi-agent system, called MASAQ, for answering users' questions based on the knowledge base. The system consists of four major components. First, it has a natural language user interface [11]. Second, MASAQ has an encyclopedic knowledge base covering 21 domains [e.g. 3, 4, 5, 6, 7, 8, 10, 13, 14, 16, 20, 21, 22]. Third, it uses a communication protocol based XML and KQML [12, 15]. Finally, MASAQ provides an executable agent specification language (EASL) for developing domain-specific multi-agent systems for answering or reasoning about users' questions, and the target systems can be deployed on the Internet as a distributed system, on a parallel supercomputer as a parallel system, or on a desktop PC as a centralized system.

[1] This work is supported by the Natural Science Foundation (#60073017 and #60273019), the Ministry of Science and Technology (#2001CCA03000 and #2002DEA30036), and Institute of Computing Technology of Chinese Academy of Sciences (#2003-12-11).

In the following, we will mainly focus on the knowledge base and multi-agent components of the MASAQ system.

When receiving a complex query, inference is often necessary since there may be no direct answers retrievable from the knowledge base. For example, when a user asks "Is New York located in North America", the MASAQ needs to infer as follows: Since USA is a country and New York is a city of USA, New York is a part of USA. Furthermore, because USA is located in North America, it can be concluded that New York is located in North America. This line of reasoning uses the following facts of country(USA), city-of(New York City, USA), and located(USA, North America), and the rules of city-of(x, y) $\rightarrow$ geo-part-of(x, y), country(x) $\rightarrow$ geo-entity(x), geo-part-of(x, y) $\rightarrow$ part-of(x, y), and geo-entity(x) & part-of(y, x) & located(x, z) $\rightarrow$ located(y, z).

Reasoning in a huge knowledge base is a great challenge. The key problem is efficiency. In the MASAQ, every agent has a rule base for making inferences, and a meta-rule base for controlling inferences. In addition, each agent shares a huge fact base, i.e. the EKB (encyclopedic knowledge base). MASAQ have a number of advantages over a single reasoning machine:

1. EKB has 21 domains and every domain has some individual categories. Every category has its own actions and plans, and it can be well described by a single agent.
2. Modularity can aid efficiency. Knowledge can be located more quickly and fewer rules need to be considered for firing at once.
3. We distribute the agents on multiple computers or/and on a parallel machine for parallel inference.

The rest of the paper is organized as follows. Section 2 presents the general architecture of MASAQ. Section 3 introduces knowledge representation in MASAQ. Section 4 presents an executable agent specification language (EASL). Section 5 discusses agent communication. Section 6 presents algorithms and implementation details of MASAQ. Section 7 gives an experiment, and section 8 summarizes the work.

2 Architecture of MASAQ

The MASAQ is comprised of a number of agents, which represent distinct entities or subjects (such as people or mathematics) that are capable of making decisions and interacting with each other.

We use multiple agents to reason about a query (also called goal or task if no confusion is caused). At the knowledge level, each agent is generally composed of four major components: a rule base (RB) which is further divided into private and public rules, a meta-rule base (MRB), a common encyclopedic knowledge base (EKB), and a belief base (BB).

As shown in Fig. 1, agents are organized in a hierarchical manner, and they perform tasks simultaneously. Higher-level agents can call lower-level agents, and lower-level agents can inherit public rules from their super-agents. There is one top-level agent called the root agent. The function of the root agent is to receive queries

from users and to invoke relevant lower-level agents using its control rules. The final result of the query is assembled at the root agent, and then delivered to the user

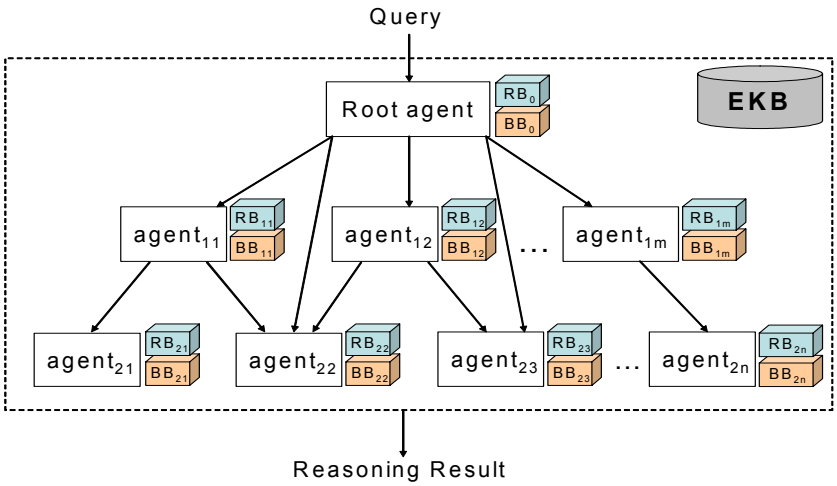

Fig. 1. Architecture of MASAQ

3 Knowledge Representation and Organization

Each agent has a private rule base, public rule base and a meta-rule base. Each rule has an identifier, a number of antecedents and a number of consequents. An antecedent or consequent is a conjunction of predicates. Formally, a rule is represented as a Horn clause of the format [9, 17]:

$$\text{id: } P_1 \,\&\, P_2 \,\&\, \ldots \,\&\, P_k \rightarrow Q_1 \,\&\, Q_2 \,\&\, \ldots \,\&\, Q_m$$

In the rule, $P_i(i=1\ldots k)$ are antecedents, and $Q_j(j=1\ldots m)$ are consequents. As illustration, let us consider the fourth rule in the introduction, i.e. geo-entity(x) & part-of(y, x) & located(x, z) → located(y, z). This rule is actually a Horn clause, where geo-entity(x), part-of(y, x) and located(x, z) are antecedents, and located(y, z) is the only consequent.

In rules, predicates can be user-defined and built-in predicates. User-defined predicates may include, say, geo-part-of(X,Y) and located(X,Y), depending on the concrete application under development. Built-in predicates are provided by the system, and further classified into two categories:

1. Common predicates: eq(X, Y), leq(X, Y), geq(X, Y), gt(X, Y), subset(X, Y), psubset(X, Y), diff(X, Y), in(X,Y), nin(X,Y), isa(X, Y), subcategory(X, Y), supercategory(X, Y), has-stages(X, Y), part-of(t1, t2)
2. Goal predicates: subtask(called-agent, G, direction, strategy) which calls a particular agent to assign a goal G to it, and may possibly recommend a

search direction and strategy to the called agent, and ifask(G) which indicates the current goal is G.

Arguments of a predicate are called terms. A term is a variable, a constant, or a function. Functions are classified into two categories:

1. Standard functions. We have designed a long list of standard functions, including arithmetic functions, such as sum(X, Y), sub(X, Y), div(X, Y), times(X, Y), sqrt(X), root(n, X), power(n, X), log(n, X), ln(X), abs(X), ceil(X), floor(X), factorial(X); trigonometric functions, such as sin(x), cos(x), and tan(x); and other functions, such as card(X), gcd(X, Y), lcm(X, Y), and reciprocal(X).
2. KAPI functions. The EKB provides a knowledge application programming interface (KAPI) for application developers. In table 1, we present a number of functions that have been already defined and implemented in the EKB.

Table 1. KAPI Functions

Operations or Predicates	Meaning
getValue(C, A)	Retrieve the value of attribute A of concept C.
A(C)	Another form of getValue(C, A).
isValue?(C, A, V)	True if V is the value of attribute A of concept C.
A(C, V)	Another form of isValue?(C, A, V).
equal?(getValue(C, A), V)	True if the value of attribute A of concept C is V.
insert(K)	Adds clause K to a BB if it is not present.
remove(K)	Removes the clause K from a BB.
getConcept(A, V)	Retrieve the concepts whose value of A is V
getAttributes(C)	Retrieve all the attributes of concept C

A meta-rule base is a collection of meta-rules. Meta rules use the same representation as (object-level) ones except that the former contain special built-in predicates and terms, such as subtask() and ifask().

In the past years, we have developed several methods for extracting knowledge from domain texts [3, 4, 5, 6, 7, 8, 10, 13, 14, 16, 20, 21, 22]. So far, we have constructed an encyclopedic knowledge base containing more than 3,000,000 assertions covering 21 domains. The EKB is encapsulated with a knowledge application programming interface (or KAPI) [8]. For details of KAPI, see Table 1.

The EKB is classified into two levels. The first level consists of domain categories (e.g. COUNTRY), which are organized into a hierarchical structure. The second level consists of instances with their category labels. Fig. 2 depicts an instance of COUNTRY (i.e. Cambodia). In the figure, the value of the attribute 'formal-English-names' of Cambodia is 'the Kingdom of Cambodia'.

```
definstance COUNTRY Cambodia
{
      formal-English-names: the Kingdom of Cambodia
      previous-formal-English-names: the Khmer Republic
      informal-Chinese-names: 柬埔寨
      formal-Chinese-names: 柬埔寨王国
      date-of-independence: November 9, 1953
      be-in-the-southeast-of: Asia
}
```

Fig. 2. An Instance of COUNTRY

4 Executable Agent Specification Language (EASL)

At the knowledge level, an agent consists of five components: super agents, subordinate agents, a meta-rule base, a public rule base and a private rule base. The first four components are optional. The public rule base of an agent consists of (object-level) problem-solving rules, which can be inherited by its subordinate agents, but private rules are not shareable to others. The meta-rule base consists of meta-rules for controlling the inference.

In MASAQ, agents are specified in a frame-like specification language called EASL. The overall syntax is depicted in Fig. 3.

```
defagent <agent>
'{'
        [Super-agents: <super agents>]
        [Subordinate-agents: <subordinate agents>]
        [<meta-rule base>]
        [<public rule base>]
        <private rule base>
'}'
```

Fig. 3. Knowledge-level Model of Agents

In our MASAQ, meta-rules are used in two situations. First, meta-rules are used for subtasking. A subtasking meta-rule is in the format:

$$\text{ifask}(G)\ \&\ P\ \&\ \ldots\ \&\ Q \rightarrow \text{subtask}(\text{called-agent}, G, \text{direction}, \text{strategy})$$

This meta-rule works as follows. Assume that the meta-rule belongs to agent A. When agent A accepts a query G, and the antecedents P & … & Q are true, then agent A calls an agent (i.e. the called agent) and sends G to it. In addition to sending

G, agent A may also recommend a search direction (either forward or backward) and a search strategy (either depth-first or breadth-first) to the called agent.

```
defagent geo-entity-agent
{
        super-agents: root-agent
        subordinate-agents: country-agent
        meta-rule1: ifask(geo-entity(x)) → subtask(country, geo-entity(x), backward,
                depth-first)
        meta-rule2: ifask(geo-part-of(x, y)) → subtask(city, geo-part-of(x, y), backward,
                depth-first)
        private-rule1: geo-part-of(x, y) → part-of(x, y)
        private-rule2: part-of(y, x) & geo-entity(x) & located(x, z) → located(y, z)
        private-rule3: geo-entity(x) & geo-entity(y) & east(x,y) → west(y,x)
}
defagent country-agent
{
        super-agents: geo-entity-agent
        subordinate-agents: city-agent
        public-rule1: eq(y, div(population(x), acreage(x))) → population-density(x, y)
        private-rule1: country(x) → geo-entity(x)
        private-rule2: eq(official-language(x),official-language(y))→equal-language(x, y)
}
defagent city-agent
{
        super-agents: country-agent
        subordinate-agents: NULL
        private-rule1: city-of(x, y) → geo-part-of(x, y)
        private-rule2: city-of(x, y) → leq(population(x), population(y))
}
```

Fig. 4. Three Agents in EASL

The second situation to use meta-rules is when several rules are invoked in an agent. In this case, meta-rules in the agent are invoked to select a best rule to continue with the inference. Such rules are called conflict-resolving rules, and they have the format:

<rule statistics> & P & … & Q → better-than(rule1, rule2)

The <rule statistics> part is a conjunction of a number of predicates about rules.

1. more-successful(rule1, rule2). It is true if rule1 is more often invoked to answer queries than rule2 does.
2. fewer-antecedents(rule1, rule2). It is true if rule1 has few antecedents than rule2.
3. cheaper(rule1, rule2).

Fig. 4 illustrates three geographical agents. 'country-agent' inherits from 'geo-entity-agent', and 'city-agent' inherits from 'country-agent'. The first meta-rule in the 'geo-entity-agent' indicates that it needs to call the agent 'country-agent' if the goal is 'geo-entity(x)'.

5 Agent Communication

In a multi-agent system, agents need to communicate with each other for many purposes. In MASAQ, there are three situations where agents communicate.

1. During reasoning, when an agent find a task (or goal) cannot be evaluated by itself, the task needs to be assigned to a particular agent that can evaluate it by corresponding meta-rule.
2. In task allocation and load balance, if there are multiple agents to fit, the current agent needs to ask the loads of these agents and choose the one with minimal cost.
3. When an agent has processed a (sub)query, it returns the results to the calling agent or to the user through the user interface.

At run time, each agent is assigned with three queues for communication: Inbox, Outbox, and Supbox. The Inbox keeps the messages that have been already received from other agents, and the Outbox contains all the messages that have been sent out to other agents. The Supbox is used during collaborative problem solving. When an agent sends a (sub)query Q to another agent, it records the relevant information of Q in its Supbox until it receives results from the other agent.

Messages in the Inbox and Outbox are expressed in a subset of KQML [12, 15]. The syntax of messages is given in Fig. 5, and the tags are explained in table 2.

```
<message>::=
        '<'action=<string>'>'
                '<'sender'>' <string> '<'/sender'>'
                '<'receiver'>' <string> '<'/receiver'>'
                '<'reply-with'>' <string> '<'/reply-with'>'
                '<'in-reply-to'>' <label> '<'/in-reply-to'>'
                '<'content'>' <string> '<'/content'>'
                '<'ontology'>' <string> '<'/ontology'>'
        '<'/action'>'
```

Fig. 5. Syntax of Agent Communication Language

In MASAQ, we have defined a list of actions, and some are shown in table 3. The most important action is ask-if, and it asks the receiving agent to perform a query. An ask-if statement contains the query, relevant data, search direction and search strategy that are proposed by the sending agent.

Table 2. Typical Tags

TAG	Meaning
action	Type of communication
sender	The agent who sends the message
receiver	The agent who receives the message
reply-with	The expected label in response to the current message
in-reply-to	The expected label in response to a previous message (same as the reply-with value of the previous message)
content	The message content communicated between agents
ontology	The ontology of the message content

Table 3. Actions in Messages

Action	Meaning
ask-if	An agent wants another agent to answer a query
tell	An agent tells another agent the result of a query
stop	An agent asks another agent to stop a QA task
ask-load	An agent asks another agent of its current task load
tell-load	An agent tells another agent of its current task load
join	An agent joins another agent as a subordinate agent
withdraw	An agent withdraws from its master agent

As illustration, Fig. 6 depicts a message that $agent_1$ sends to $agent_2$. The message is to ask $agent_2$ to answer which countries use German as the official language. $agent_1$ also advises $agent_2$ to reason using the depth-first strategy in a backward direction.

```
<action=ask-if>
      <sender> agent1 </sender>
      <receiver> agent2 </receiver>
      <reply-with> msg001 </reply-with>
      <content> official-language(x, German), backward, depth-first </content>
      <in-reply-to> msg000 </in-reply-to>
      <ontology> NKI Ontology </ontology>
</action>
```

Fig. 6. A Message that $agent_1$ Sends to $agent_2$

After receiving the message from $agent_1$, $agent_2$ processes it, and replies with the message shown in Fig. 7, where the answer is Austria, Germany, and Switzerland.

```
<action=tell>
      <sender> agent2 </sender>
      <receiver> agent1 </receiver>
      <reply-with> msg002 </reply-with>
      <in-reply-to> msg001 </in-reply-to>
      <content>  Austria,  Germany,  Switzerland
</content>
      <ontology> NKI Ontology </ontology>
</action>
```

Fig. 7. A Message in Reply to msg001 from $agent_1$

Now, we turn to the Supbox of an agent. The Supbox is used when an agent asks another agent to answer a sub-query. In this situation, the sending agent needs to suspend its inference of the current query to wait for the answer to the sub-query, and continues with other queries to achieve parallelism. In order to resume the suspended query, the sending agent preserves the relevant data into its Supbox.

Table 4. Data in Supbox

Field	**Meaning**
current query	Including data, content of query (including search strategy and sender agent)
serial_no	Serial number used to identify the suspended query according to the returned message from the other agent later
message	The message sent to the other agent
time of sending	The time of sending the message
stack	Containing information of every inference step of the query

In MASAQ, every agent is an autonomous process. Communication between agents is implemented by MPI. MPI is a message-passing interface standard, and it is the standard for multicomputer and cluster message passing introduced by the Message-Passing Interface Forum in April 1994. The goal of MPI is to develop a widely used standard for writing message-passing programs [1, 2].

Fig. 8 depicts the initialization of MPI. 'MPI_Init' initializes MPI and starts a MPI program. 'MPI'_Comm_rank' gets the identifier of the current process. 'MPI_Comm_size' gets the number of agents. Fig. 9 describes a message communicating course. The sender agent sends a message to the receiver agent by 'MPI_Send' and the receiver agent receives the message by 'MPI_Recv'.

```
MPI_Init (&argc, &argv)
MPI_Comm_rank (MPI_COMM_WORLD, &rank)
MPI_Comm_size (MPI_COMM_WORLD, &size)
```

Fig. 8. Initialization of MPI

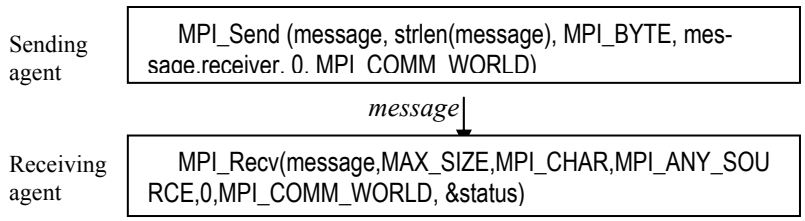

Fig. 9. Flow Chart of Message Communication

When an agent asks another agent to perform a goal, it needs to suspend the current task, preserve the current state, and then perform the next task. When the result of the task is returned, the agent needs to resume the corresponding reasoning scene and continue the task.

6 Algorithms and Implementation Details

6.1 Agent Compilation

Since an EASL program consists of a number of defagent statements, the compiler compiles the program statement by statement. For each such defagent statement, the EASL compiler, the agent compiler performs the following steps:

Step 1: Checking Well-Definedness of Super-agents and Subordinate-agents
The values of super-agents and subordinate-agents are agent names. The super-agents and subordinate-agents slots are well-defined if all the super-agents and subordinate-agents are defined in the program, or registered in the MASAQ registry.

Step 2: Checking Well-Definedness of Private Rules, Public Rules and Meta-Rules
Each predicate in a private, public or meta rule is defined in the Predicate Definition Table (PDT). The PDT consists of a tuple for each predicate in an EASL program. The fields of the table are: 1) predicate name: the name of the predicate. 2) user-defined?: true if the predicate is user-defined; otherwise built-in. 3) arity: The arity of the predicate. 4) type of arg_1, ..., type of arg_n. Each type of argument is kept in the PDT.

In addition to the PDT, MASAQ also has a separate Function Definition Table (FDT). The FDT consists of a tuple for each function in an EASL program. The FDT fields are: 1) function name: the name of the function. 2) user-defined: true if the function is user-defined; otherwise built-in. 3) function type: the type of the function. It can be one of basic types, such as Integer, Real, Boolean, and String; it can also be compound types, Integers, Reals, Booleans, and Strings, representing sets of integers,

real numbers, booleans, strings, and respectively. 4) arity: The arity of the function. 5) type of arg_1, ..., type of arg_n. Each type of argument is kept in the FDT.

A rule is well-defined if it obeys the rule syntax shown in section 3.1, and each predicate in the rule is well-defined according to the PDT.

Step 3: Compiling Rules into Internal Representation

First, the compiler uses type information supplied by predicate definitions to optimize situations: Knowledge base manipulations are compiled into simple KAPI to retrieve and calculate clauses, and some standard functions (E.g. sum and equal) are compiled into particular code. Then the rule base of an agent is represented as a network, where each rule is represented as a tree. For example, the rule "part-of(y, x) & geo-entity(x) & located(x, z) → located(y, z)" can be represented as Fig. 10.

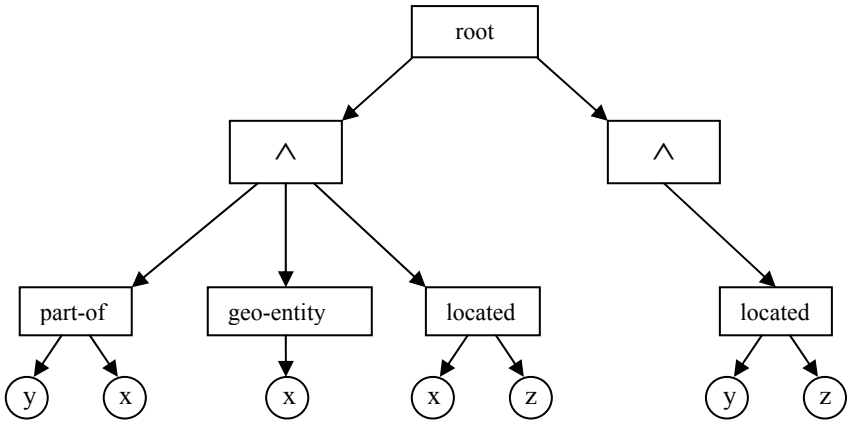

Fig. 10. Internal Representation of a Rule

Another important internal representation is an index table of the consequents of the rules in an agent. The index table is used during the backward reasoning in an agent to speed up search. Assume the current agent is A_k, and it has m rules. The fields of the index table are: 1) predicate Pt, and 2) {<i,j>|1≤i≤m, rule[i].consequent[j] has the same name as Pt}. In order to implement fast searching in the forward reasoning, we also produce an antecedent index table for each agent. The fields of the table are: 1) predicate Pt, and 2) {<i,j>|1≤i≤m, rule[i].antecedent[j] has the same name as Pt }

6.2 Evaluation of Predicate

Evaluation of predicate can be classified as predicate evaluation and argument evaluation. KAPI's are evaluated by retrieving knowledge bases, and standard functions are evaluated by executing their code, and others are evaluated by reasoning. How to evaluate a predicate (i.e. whether to make inference, directly retrieve from a BB, or directly retrieve from the EKB) - is determined according to the PDT.

Evaluating a predicate determines the conditions under which it is true, i.e. the bindings for variables such that the predicate is true. The predicate may succeed many times, with different combinations of bindings, until all solutions are found.

Predicate matching is a consistent matching of two predicates in most general unification. If there is a most general unification between two homonymous predicates, then the predicates matching is succeed, and we'll replace all relevant variables in the predicate and rule with the value of the variables in the general unification. Predicate matching is the most frequent function during reasoning. Predicate P_1 matches predicate P_2 in the following condition: 1) They are homonymous, and 2) their arguments are matched with each other.

6.3 Inference Engines

Each agent owns an inference engine – a rule interpreter, and all the inference engines are exactly the same. The inference engine has two search directions, i.e. *forward* and *backward*, and two search strategies, i.e. *depth-first* and *breadth-first*. The default search direction is backward, and the default search strategy is depth-first.

As shown in Fig. 11, subtasking rules of an agent can determine which reasoning machine to choose for a goal. Now we have developed two inference engines: backward depth-first and forward breadth-first. According to the type of goal (predicate or rule), every engine can be further divided into predicate inference engine and rule inference engine. Predicate inference engine can be further divided into predicate-evaluator and argument-evaluator. So we have six inference engines actually.

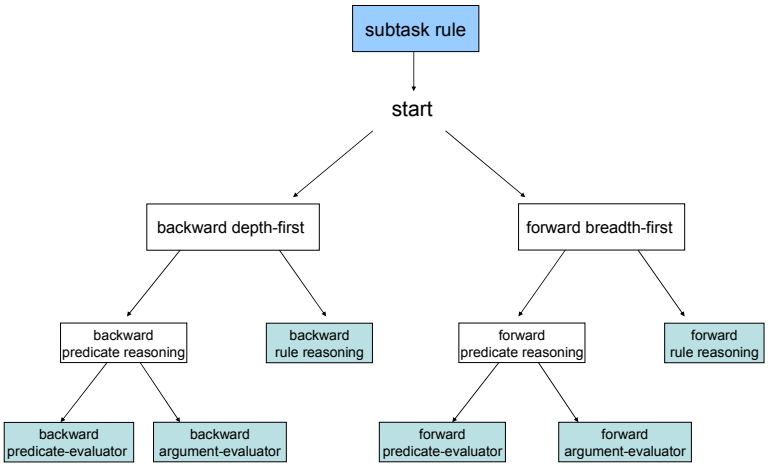

Fig. 11. Multiple Inference Modes

The predicate-evaluator determines if a goal is true, e.g. official-language(British, English). The argument-evaluator finds the values of the variables in the goal, e.g. official-language(x, English). In two engines above, the goal is a predicate. But the

goal is a rule, e.g. official-language(x, English)→located(x, Europe) in the rule inference engine, which determines if the rule is true.

6.4 Query Answering

To use the multi-agent system, the user can raise a query in the form of a predicate or a rule. In the first case, the system evaluates the predicate or reason backward or forward according to defined strategies.

When the query is a rule, e.g. city-of(x, y)→part-of(x, y), we say the query represents a verification task. It is transformed into predicate reasoning by adding the antecedents of the rule to the data base, and its consequents as the goal. For above example, we can transform it into "data=city-of(x, y), goal=part-of(x, y)", and then perform predicate evaluation. And we have to find the bindings of variables such that the antecedents are true, and deduce that the consequents are true when the bindings are applied. If the consequents are true for all bindings, then the goal is true.

6.5 Load Balance

A query can be performed by multiple agents, so load balance is necessary. However, it is difficult to balance load when the agents are selecting subtasks in a distributed manner [18, 19]. Load balance in MASAQ includes load measurement, transmission strategy and placement strategy.

Information measurement determines the load of an agent. The load of an agent is determined by the following factors:

1. The number of task on the agent
2. Average processing time of task on the agent
3. Difficulty of task
4. Difficulty of task can be valued by the following method. Diff(P, agenti) means difficulty of task P on $agent_i$

$$Diff(P, agent_i) = \begin{cases} 1, \text{ if the task can be valued distinctly (KAPI or standard function)} \\ Max\ \{Diff1(P, agent_i, rule_p[0]), Diff1(P, agent_i, rule_p[1]), \ldots\}\ , \text{else} \end{cases}$$

$$Diff1(P, agent_i, rule_p[i]) = \begin{cases} Diff(P, agent_k), \text{ if the consequent of } rule_p[i] \text{ is subtask}(agent_k, \ldots) \\ \sum_k \{Diff(rule_p[i].prem[k], agent_i)\}\ , \text{else} \end{cases}$$

$rule_p$: the collection of rules whose one consequent is P in the current agent

prem: antecedents of the current rule

Fig. 12. Calculating Difficulty of Task for Load Balance

A transmission strategy judges if we transmit a task on one agent. If the load of the agent is greater than α and the task waiting time is greater than t, then we need transmitting.

Placement strategy chooses the task transmitted and its destination. The task transmitted must be a large task and have little transmission cost. We use the polling method to determine the receiving agent. Firstly, select an arbitrary agent from those ones that have not been checked. Secondly, check if the load of the agent exceeds the limit when the task reaches. If the load exceeds the limit, then the agent is the destination of the task; otherwise we continue to select another agent to check until we find the destination or detecting time exceeds a limit.

6.6 Implementation

We have implemented a multi-agent parallel system in ANSI C on Windows2000/Linux/Unix. The MASAQ agents can be deployed on the Internet as a distributed system, on a parallel supercomputer as a parallel system, or on a desktop PC as a centralized system.

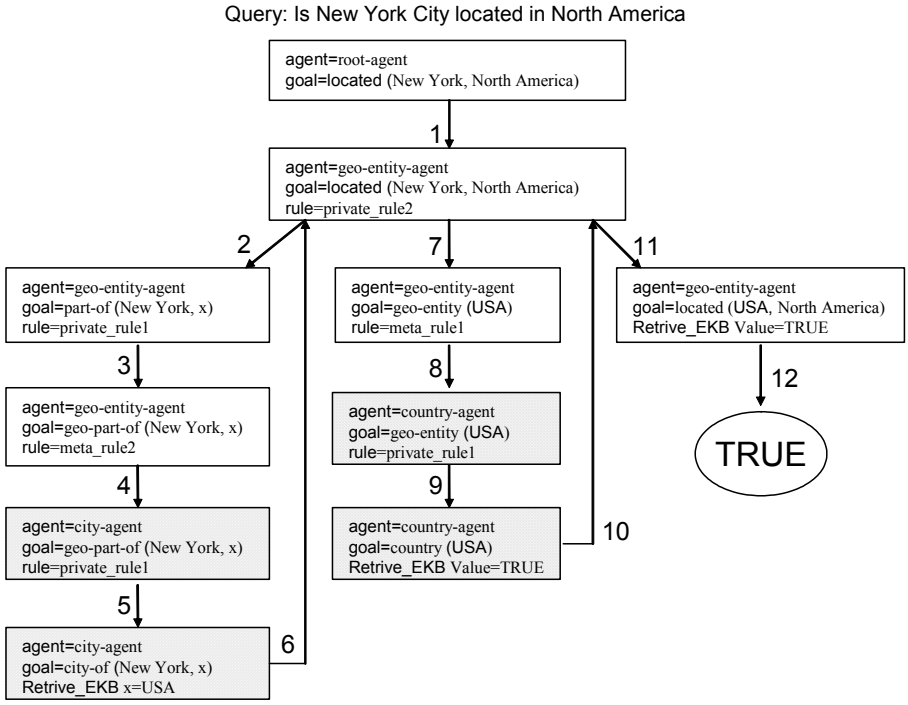

Fig. 13. An Instance of Multi-agent Reasoning

7 Experiment

We suppose that a user asks "Is New York City located in North America", the system receives the goal 'located (New York, North America)', and the root agent calls the 'geo-entity-agent', as shown in Fig. 4. The line of reasoning is depicted in Fig. 13.

The goal matches the second private-rule 'part-of(y, x) & geo-entity(x) & located(x, z) → located(y, z)', and then we instantiate the antecedents of the rule and get three sub-goals: 'part-of(New York, x)', 'geo-entity(x)' and 'located(x, North America)'.

The first sub-goal 'part-of(New York, x)' matches the first private-rule and gets its sub-goal 'geo-part-of(New York, x)' which matches the second meta-level. At the same time, the agent needs to call its subordinate 'country-agent'. When the value of the argument x is calculated by the 'country-agent', 'geo-entity-agent' continues with inference.

We performed four experiments on geography which contains 9 agents and 520 rules. 1000 tasks were sent to the root agent, and detailed data is collected in Table 5.

Table 5. Four Experiments of MASAQ

No.	Agent Deployment	Number of Agent Communication	Execution Time (ms)
1	On 1 PC	2646	4364
2	On 4 PCs	2887	2218
3	On 8 PCs	3013	1192
4	On 8 PCs and 1 supercomputer	3182	527

8 Conclusion

Answering questions based on large-scale domain knowledge is a challenging task. The key problem is efficiency. We developed a multi-agent system based on an executable specification language, and the system can run on the Internet as a distributed system, on a parallel machine as a parallel system, or on a desktop PC as a standalone system. In the multiple agents, there are a number of complicated tasks such as multi-agent communication and load balance.

MASAQ is encoded in ANSI C, and it can runs on Windows2000/Linux/Unix. Dozens of agents are distributed to eight PC computers and a supercomputer. The number of PCs and parallel machines can be initiated in advance. Repeated tests and experiments in the last two months have demonstrated our MASAQ can reason all the knowledge of 21 domains efficiently.

At present, the format of rules we considered is standard horn. We find that horn rules are not sufficient for representing the world depicted by domain knowledge. Extension of horn rules is our future work. We will also enhance the capabilities of MASAQ in abnormality handling.

References

1. Adamo, J.M.: Multi-Threaded Object-Oriented MPI-Based Message Passing Interface: the ARCH library. Kluwer Academic. 1998.
2. Alexandrov, V., Dongarra, J.: Recent Advances in Parallel Virtual Machine and Message Passing Interface. Proc. the 5th European PVM/MPI User's Group Meeting. 1998.
3. Cao, C.G: Medical Knowledge Acquisition from Encyclopedic Texts. Lectures in Computer Science. vol.2101, 268-271. 2001.
4. Cao, C.G., Shi, Q.Y: Acquiring Chinese Historical Knowledge from Encyclopedic Texts. In Proceedings of the International Conference for Young Computer Scientists. 1194-1198, 2001.
5. Cao, C.G., Sui, Y.F.: Constructing Ontology and Knowledge Bases from Text. ICCPOL'03. 34-42, 2003.
6. Cao, C.G., Wang, H.T., Sui, Y.F.: Acquiring Knowledge of Herbal Drugs and Formulae from Text. To appear in International Journal of Artificial Intelligence in Medicine. 2004.
7. Cao, C.G., Wang, H.T.: An Ontology-Mediated Knowledge Programming Framework for Rapidly Acquiring Domain Knowledge from Semi-Structured Text. Proceedings of the Pacific Rim Knowledge Acquisition Workshop. 2002.
8. Cao C.G., Zheng Y.F.: Knowledge Application Programming Interface 1.1 (KAPI 1.1). Technical Report. Institute of Computing Technology. Chinese Academy of Sciences. 2000.
9. Chandra, A., Harel, D.: Horn clause queries and generalizations. Journal of Logic Programming. 1-5. 1985.
10. EOC Publisher: The Encyclopedia of China. China EOC Publisher. 1997.
11. Feng, Q., Cao, C.G.: A Uniform Human Knowledge Interface to the Multi-Domain Knowledge Bases in the National Knowledge Infrastructure. ES2002. 163-176. 2002.
12. Finin, T., Weber,J.: Specification of the KQML: Agent Communication Language. DRAFT. 1993.
13. Gao, Y., Cao, C.G.: Acquiring Musical Knowledge from Encyclopedic Text. Journal of Coputer Science. 2003.
14. Gu, F., Cao, C.G.: Biological Knowledge Acquisition from the Electronic Encyclopedia of China. Proceedings of the 16th International Conference for Young Computer Scientists. 1199-1203, 2001.
15. Labrou, Y., Finin T.: A Proposal for a New KQML Specification. TR CS-97-03. 1997.
16. Lei, Y.X., Cao, C.G.: Acquiring Military Knowledge from the Encyclopedia of China. In the Proceedings of the Sixth International Conferences for Young Computer Scientists. 368-372, 2001.
17. Lloyd, J.: Foundations of Logic Programming. Springer-Verlag. 1987.
18. Krothapalli, N., Deshmukh, A.: Distributed Task Allocation in Multi-Agent Systems. Proc. the 11th Industrial Engineering Research Conference. 2002.
19. Wallis, S., Moss, S.: Efficient Forward Chaining for Declarative Rules in a Multi-Agent Modelling Language. CPM Report: 004. 1994.
20. Wang, L.L., Cao, C.G.: Acquiring Ethnic Knowledge from Encyclopedic Text. Journal of Computer Science. 2003.
21. Zhang, C.X., Cao, C.G., Gu, F., Si, J.X.: A Domain-Specific Formal Ontology for Archaeological Knowledge Sharing and Reusing. The 4th International Conference on Practical Aspects of Knowledge Management. Vol.2569, 213-225, 2002.
22. Zhang, D.H., Cao, C.G.: Acquiring Knowledge of Chinese Cities from the Encyclopedia of China. In the Proceedings of the Sixth International Conferences for Young Computer Scientists. 1111-1193, 2001.