# Design and Implementation of P2P Reasoning System based on Description Logic

Hai WAN*, Yang YU†, Jian-tian ZHENG‡
*Software School*
*Sun Yat-Sen University*
*Guangzhou, People's Republic of China*
**Corresponding author: wanhai@mail.sysu.edu.cn*
*†yuy@mail.sysu.edu.cn*
*‡zsu_tim@163.com*

***Abstract*—P2P reasoning system can answer queries from not only each peer's local theory but also some other peers related with sharing part of its vocabulary. This paper concentrates on P2P reasoning technology based on description logic, trying to provide an intelligent solution for searching network resources. A P2P reasoning algorithm suitable for description logic called DL-DeCA is proposed in this paper, as well as its communication protocol and system implementation of DL-P2PRS (description logic P2P reasoning system). Experiment results show that DL-P2PRS can work correctly and effectively.**

***Keywords*-P2P reasoning; Description logic; DL-DeCA; DL-P2PRS**

## I. Introduction

With the rapid development of Internet, the issue how to help users obtain information and resources quickly and accurately is an important research topic. Featured as load balance, high autonomy and scalability, P2P communication mode attracts more and more attentions. However, traditional P2P systems retrieve resources based on keyword-matching approach without semantic description.

This paper concentrates on P2P Reasoning technology based on description logic (DL), trying to provide an intelligent solution for searching network resources. Different from keyword-based matching approach, P2P reasoning based on DL reasons queries on a set of DL knowledge base to get richer and hidden consequences, achieving intelligent searching semantically.

The contributions of this paper are given as follows:

1. A P2P reasoning algorithm suitable for DL called DL-DeCA is proposed, by the means of modifying widely applied propositional P2P reasoning algorithm DeCA.

2. DL-P2PRS (description logic P2P reasoning system) is presented, with application layer communication protocol and system maintenance mechanisms, etc.

3. DL-P2PRS system prototype based on OWL is implemented. Experiment results show that the system can work correctly and effectively.

This paper is organized as follows. In the next section, we discuss related works, including P2P technique, consequence finding algorithm, etc. Section 3 introduces description logic basic concept and reasoning technique. Section 4 presents DL-P2PRS framework and depicts an example about how to apply P2P reasoning method. P2P reasoning algorithm DL-DeCA is proposed in section 5, as well as algorithm analysis. Based on DL-DeCA, reasoning message and control message design and system maintenance mechanisms of DL-P2PRS are given in section 6.

## II. Related Works

In P2P reasoning system, each peer maintains a knowledge base. When asked for query, peer should reason on local knowledge base and propagate through mappings relationship between neighbor peers in entire P2P network. Some P2P reasoning systems present this feature, such as Cycorp's Cyc [1] and HPKB [2].

Because logical reasoning in P2P reasoning system is processed in collaboration with each node, the first issue is how to split tasks and combine reasoning results. Amir investigates the problem of reasoning with partitions of related logical axioms, concerning on how to reason effectively with multiple knowledge bases which have overlap in content [3] [4]. Based on [3] [4], Goasdoue' approach is characterized by a simple class-based language for defining peer schemas as hierarchies of atomic classes [5]. Adjiman provides the first consequence finding algorithm in a peer-to-peer setting - DeCA, which can compute sequences gradually from the solicited peer to more and more distant peers [6] [7].

The second problem in P2P reasoning is how to eliminate conflict and get reasonable conclusion between peers. Benferhat's algorithms can combine answers from different peers so as to compute implicates involving target variables [8]. Arenas shows a method computing the query T(Q) for a given first order query Q, which can maintain consistent query answers in inconsistent databases [9]. Bertossi presents the answers of how to maintain consistent between local semantic constraints and other peers, by which semantics for peer consistent answers under exchange constraints and trust relationships is introduced [10] [11]. Calvanese studies query answering in new nonmonotonic logic, which can establish its decidability and computational complexity [12]. Bertossi describes different approaches to the issue of computing

978-0-7695-4612-4/11 $26.00 © 2011 IEEE
DOI 10.1109/DASC.2011.57

consistent query answers [13]. Adjiman' algorithm can split clauses if they involve vocabularies of several peers, in which each piece of a splitted clause can be transmitted to corresponding theory to find its consequences [6] [7]. Chatalic exhibits a distributed algorithm detecting inconsistencies in a fully decentralized propositional P2P inference systems [14]. Binas provides a formal characterization of a prioritized peer-to-peer query answering framework that exploits a priority ordering over the peers, as well as an ordering heuristic that exploits the peers' priority ordering and empirically evaluate its effectiveness [15].

## III. Description Logic and OWL Reasoning

### A. An Overview of Description Logic

Description Logic (DL) describes domain through concepts (classes), roles (relationships) and individuals, which is a logic based knowledge representation formalisms [16]. A typical description logic knowledge base can normally be separated into two parts - a TBox (Terminology box) which is a set of axioms in the form of terminology describing the structure of domain (i.e. schema) and an ABox (Assertional knowledge box) which is a set of axioms describing a concrete situation (i.e. data). Terminology in TBox is also known as vocabulary which contains concepts denoting sets of individuals and roles presenting binary relationships between concepts. Intentional knowledge is stored in TBox in the form of the declarations that describe general properties of concepts. A DL based knowledge representation system provides the ability to set up a knowledge base and to reason about its content as well as manipulate it. Elementary descriptions are atomic concepts and atomic roles. Complex descriptions can be inductively built on them by concept constructors. Description languages can be distinguished by the constructors they provide. The basic description language is AL (Attributive language). The other languages of this family are all extensions of AL. For instance, ALEN is the extension of AL by adding full existential quantification and number restrictions. Table I shows typical DL grammar and semantics.

**Table Ⅰ.** *DL grammar and semantics*

| Symbol | Semantics | Example | |
|---|---|---|---|
| T | $\Delta^I$ | $human \cup \neg human$ | $ALC$ |
| $\perp$ | $\varnothing$ | $human \cap \neg human$ | |
| $A$ | $A^I \subseteq \Delta^I$ | *Male* | |
| $R$ | $R^I \subseteq \Delta^I \times \Delta^I$ | *hasChild(Jo,Ann)* | |
| $C \cap D$ | $C^I \cap D^I$ | $human \cap male$ | |
| $C \cup D$ | $C^I \cup D^I$ | $female \cup male$ | |
| $\exists R.C$ | $\{a \in \Delta^I \mid \exists b.(a,b) \in R^I \wedge b \in C^I\}$ | $\exists hasChild.male$ | |
| $\forall R.C$ | $\{a \in \Delta^I \mid \forall b.(a,b) \in R^I \rightarrow b \in C^I\}$ | $\forall hasChild.male$ | |
| $\neg A$ | $\Delta^I \setminus A^I$ | $\neg male$ | |
| $\geq nR$<br>$\leq nR$ | $\{a \in \Delta^I \Vert \{b \mid (a,b) \in R^I\} \rvert \geq n\}$<br>$\{a \in \Delta^I \Vert \{b \mid (a,b) \in R^I\} \rvert \leq n\}$ | $\geq 2hasChild$<br>$\leq 1hasSon$ | $N$ |
| $\geq nR.C$<br>$\leq nR.C$ | $\{a \in \Delta^I \Vert \{b \mid (a,b) \in R^I \wedge b \in C^I\} \rvert \geq n\}$<br>$\{a \in \Delta^I \Vert \{b \mid (a,b) \in R^I \wedge b \in C^I\} \rvert \leq n\}$ | $\geq 2hasChild.Son$ | $Q$ |
| $R^+ \sqsubseteq R$ | $R^I = (R^I)+$ | *biggerThan* | $R^+$ |
| $R^-$ | $\{(a,b) \mid (b,a) \in R^I\}$ | *equals* | $I$ |
| $R \sqsubseteq S$ | $R^I \subseteq S^I$ | $hasSon \sqsubseteq hasChild$ | $H$ |
| $\circ$ | $\circ^I \in \Delta^I, \lvert \{a \mid a \in \circ^I \wedge \lvert a \rvert = 1\}$ | *{Red,Green,Blue}* | $O$ |
| $d$<br>$F$<br>$\exists F.d$<br>$\forall F.d$ | $d^D \in \Delta^D$<br>$F^I \in \Delta^I \times \Delta^D$<br>$\{a \in \Delta^I \mid \exists b.(a,b) \in F^I \wedge b \in d^D\}$<br>$\{a \in \Delta^I \mid \forall b.(a,b) \in F^I \rightarrow b \in d^D\}$ | | $D$ |
| $R : a$ | $\{b \in \Delta^I \mid (b,a^I) \in R^I\}$ | $hasChild : Jim$ | F |

Current DL systems focus on the need for complete algorithms with strong expressive power support. With the extensions of tableau-based techniques and the introduction of several optimization techniques, more advanced current DL systems have been developed. FaCT [17] is the most significant example.

### B. OWL Reasoning

Web Ontology Language (OWL) is a language designed to define and instantiate web ontology with which the meaning of terms and their relationships can be represented explicitly. OWL provides a reasoning service to help knowledge engineers and users build and use ontology by checking logical consistency of classes and computing implicit class hierarchy. The OWL reasoning service is especially important for designing and maintaining large global ontology, integrating and sharing ontology, checking consistency and implying implicit relationships. For most DLs, the basic inference problems are decidable, solving the problems in a finite number of steps. OWL also provides the following reasoning which can facilitate knowledge based approach:

- Subsumption reasoning: (a) infer when one class A is a subclass of another class B, (b) infer that B is a subclass of A if it is necessarily the case that all instances of B must be instances of A, (c) build concept hierarchies representing the taxonomy - the classification of classes.
- Satisfiability reasoning: (a) check when a concept is unsatisfiable, (b) check whether the model is consistent.

In OWL, reasoning classes can be described in terms of necessary and sufficient conditions. The necessary conditions indicate that the condition must hold for checking the instance of the class, while the sufficient conditions indicate that the object must have the properties recognized as a member of the class. In addition, the OWL reasoning service provides a way to perform automatic classification [18].

## IV. DL-P2PRS Framework

DL-based P2P reasoning system is a P2P network based on DL, in which each node maintains local DL-based knowledge base, and can communicate with each other by the means of mapping relationship. When one node is asked for query, peer will reason in local knowledge base and propagate query through mappings relationship between neighbor peers in entire P2P network, finally return reasoning result back to the initial node.

## A. DL-P2PRS Definition

**Definition 1.** ***Peer.*** *A peer node $P_i$ is a tuple $(KB_i, Map_i)$, in which $KB_i = (TBox_i, ABox_i)$ is the local knowledge base based on DL in node $P_i$ ; $Map_i = \bigcup_{j=1 \cdots k} mapping_{i,j}$ is the mapping relation set between other nodes.*

**Definition 2.** ***Mapping.*** *A $mapping_{i,j}$ is a specification like $C_j \sqsubseteq C_i$ or $C_j \equiv C_i$, in which $C_i$ and $C_j$ are legal concept expression based on DL. The mapping relationship between $C_i$ and $C_j$ reflects their concepts inclusion or equal relationship, stored only in the initiative node.*

**Definition 3.** ***Share variable.*** *In a mapping $C_j \sqsubseteq C_i$ or $C_j \equiv C_i$ , if this mapping is built by node $P_i$ , $C_i$ is regarded as the share variable between $P_i$ and $P_j$; share variable can only be stored in the initiative node.*

**Definition 4.** ***DL-P2PRS(DL-based P2P reasoning system).*** *DL-P2PRS is a tuple $(P, G)$ , in which $P = \bigcup_{i=1 \cdots n} P_i$ is the set of peer nodes, $G = (P, E)$ depicts the system topology graph. $E = \bigcup_{i=1 \cdots k} e_k$ , in which each $e_k$ is a undirected arc, corresponding to a $mapping_{i,j}$ .*

**Definition 5.** ***DL-P2PRS Semantics.***

- *The interpretation of a DL-P2PRS $P = \bigcup_{i=1 \cdots n} P_i$ is the interpretation set of all nodes, i.e.$I = \bigcup_{i=1 \cdots n} I_i$;*
- *The schema definition of $P$ is $Schema(p) = \bigcup_{i=1 \cdots n} TBox_i \cup ABox_i \cup Map_i$;*
- *If an interpretation of $P$ satisfies $Schema(p)$, regarded $I$ as the model of $P$;*
- *$P$ is satisfiable iff $P$ has at least one model.*

**Definition 6.** ***Implication concept.*** *Assuming $P$ is a DL-P2PRS, $C$ is a legal DL concept, if for every model $I$ of $P$ , there is $D^I \subseteq C^I$ or $D^I = C^I$ , then regard $D$ as implication concept of $C$. i.e. if concept $D$ is a implication concept of $C$, then there exits a chain in $P$ , in which the number of $D$ and $C$ is more than or equal to 0, any $E \sqsubseteq F$ (or $E \equiv F$) in chain is regarded as the containing relationship in one node $TBox$ or mapping relationship between two nodes.*

**Definition 7.** ***DL-P2PRS reasoning problem.*** *Assuming $\Gamma = (P, G)$ as a DL-P2PRS, in which $P = \bigcup_{i=1 \cdots n} P_i$ is a set of tuple $(KB_i, Map_i)$. Define reasoning problem in $\Gamma$ as: given a legal DL concept $C$, find all implication concept $D$ as well as instance set of $D$ in $\Gamma$.*

## B. Typical Example

The following is a DL-P2PRS example with 6 peers, showing how to apply P2P reasoning system. 6 peers are Jim, Lucy, Andy, Bily, Gigi, and Windy, all of them are movie lovers, collecting movie information and saving URL in each peer node (shown in Figure 1):

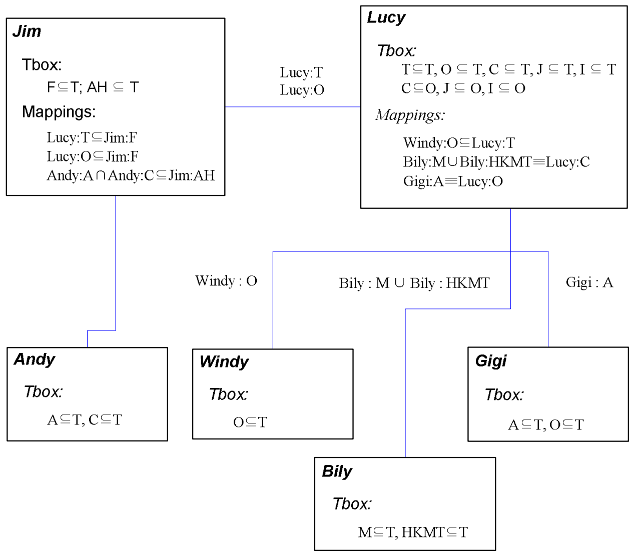


Figure 1. *DL-P2PRS network: an example*

(1) Jim collects movie information as two categories:

- F(Favorite): Jim's favorite movies;
- AH(Action Humorous): Jim like action and humorous movies. In TBox, Jim claims action and Humorous movies is part of Favorite, i.e. $AH \sqsubseteq F$.

(2) Lucy focuses on high level movies, and likes the oriental culture. she organizes as following:

- T(Top): The best movies;
- O(Oriental): Oriental movies;
- C(Chinese): Chinese movies;
- J(Japanese): Japanese movies;
- I(Indian): Indian movies.

, in which $C \sqsubseteq O$, $J \sqsubseteq O$, $I \sqsubseteq O$.

(3) Andy organizes his movies according to the styles:

- A(Action) : Action movies;
- C(Comedy): Comedy movies;
- Others movie.

(4) Bily likes Chinese movies, including:

- M(Mainland): Mainland movies;
- HKMT(HongKong,Macro,Taiwan): Movies produced in HongKong, Macro, or Taiwan.

(5) Gigi organizes her movies as two categories:

- A(Asian): Asian movies;
- O(Occident): Occident movies.

(6) Windy only focuses Oscar movies:

- O(Oscar): Oscar movies.

Jim agrees with the viewpoint of Lucy and he considers he like all the movies Lucy collected. Therefore he claims 2 mappings between Jim and Lucy, i.e. $Lucy : T \sqsubseteq Jim : F$,

$Lucy : O \sqsubseteq Jim : F$. In addition, Jim regards the intersection of Action and Comedy defined by Andy and Action and Humorous defined by himself, so he claims: $Andy : A \cap Andy : C \sqsubseteq Jim : AH$.

Lucy believes Oscar movies Windy Defined is part of the best movies, therefore she claims $Windy : O \sqsubseteq Lucy : T$. Lucy thinks Bily's ontology between Mainland and HKMT is the same concept of Chinese she defines, so she claims $Lucy : C \equiv Bily : M \cup Bily : HKMT$. In addition, Lucy believes Asian defined by Gigi is the same concept as Orient she defines, so she claims $Lucy : O \equiv Gigi : A$.

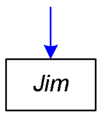


Figure 2. *User sends a reasoning request in node Jim*

We can show how to reason in this DL-P2PRS. Assuming user sends a query request in node Jim, shown in Figure 2. DL-P2PRS can reason as following steps:

**Step1:** Node Jim reasons in local, the result is $\{Jim : F, Jim : AH, Lucy : T, Lucy : O, Andy : A \cap Andy : C\}$. In which $Lucy : T$ and $Lucy : O$ are share variables between Jim and Lucy, $Andy : A \cap Andy : C$ is share variable between Jim and Andy. Node Jim can return result $Jim : F$ and $Jim : AH$ back to user, and send 2 reasoning request to node Lucy, ask Lucy to reason on $Lucy : T$ and $Lucy : O$. Furthermore, sending reason request message to node Andy, asking Andy to reason $Andy : A \cap Andy : C$. Figure 3 shows reasoning in the first level.

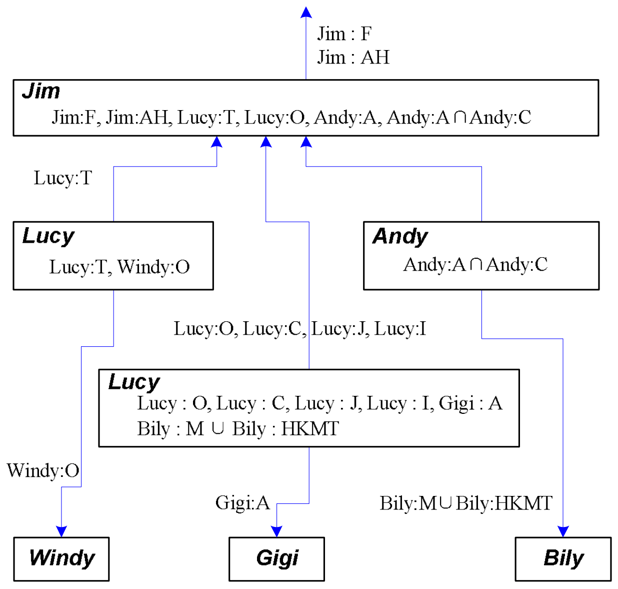

Jim : F
Jim : AH
Jim
Jim:F, Jim:AH, Lucy:T, Lucy:O, Andy:A, Andy:A∩Andy:C
Lucy:T
Lucy
Lucy:T, Windy:O
Andy
Andy:A∩Andy:C
Lucy:O, Lucy:C, Lucy:J, Lucy:I
Lucy
Lucy : O, Lucy : C, Lucy : J, Lucy : I, Gigi : A
Bily : M ∪ Bily : HKMT
Windy:O
Gigi:A
Bily:M∪Bily:HKMT
Windy
Gigi
Bily

Figure 3. *Reasoning in the first level*

**Step 2:** Node Lucy receives the message $Lucy : T$ Jim sending and processes as following: reason $Lucy : T$ in local, and get the result $\{Lucy : T, Windy : O\}$, in which $Lucy : T$ is the share variable between Lucy and Jim, $Windy : O$ is the share variable between Lucy and Windy. Because Jim shares $Lucy : T$ with Lucy, Lucy should send $Lucy : T$ back to Jim, and send Windy a reason request message to reason $Windy : O$.

**Step 3:** Node Lucy receives message sent by Jim to reason $Lucy : O$. Lucy should reason in local and get the result $\{Lucy : O, Lucy : C, Lucy : J, Lucy : I, Gigi : A, Bily : M \cup Bily : HKMT\}$, in which $Gigi : A$ is the share variable between Lucy and Gigi, $Bily : M \cup Bily : HKMT$ is share variable between Lucy and Bily. Lucy sends reason message to Gigi to reason $Gigi : A$, meanwhile asks Bily to reason $Bily : M \cup Bily : HKMT$.

**Step 4:** Node Andy receives message sent by Jim to reason $Andy : A \cap Andy : C$, Andy should reason in local and get the result: $\{Andy : A \cap Andy : C\}$, because Jim shares $Andy : A \cap Andy : C$ only with Andy, Jim is the request message's sender, Andy should return $Andy : A \cap Andy : C$ back to Jim, and send a message to Jim that the reasoning of $Andy : A \cap Andy : C$ is finished.

**Step 5:** Node Windy receives the message sent by Lucy to reason $Windy : O$. Windy should reason in local and get the result $\{Windy : O\}$. $Windy : O$ is the share variable between Windy and Lucy, however, because Lucy is the only neighbor of Windy sharing $Windy : O$ , Windy return back to Lucy that the reason on $Windy : O$ is finished.

**Step 6:** Node Gigi receives message from Lucy about reason on $Gigi : A$, Gigi should reason in local and the result is $\{Gigi : A\}$. Just like Step 5, Gigi returns back to Lucy that reason on $Gigi : A$ is finished.

**Step 7:** Node Bily receives message from Lucy about reasoning on $Bily : M \cup Bily : HKMT$, similarly, Bily returns back to Lucy that reason on $Bily : M \cup Bily : HKMT$ is finished.

**Step 8:** Node Lucy receives the result sent by Windy and stores $Windy : O$ in the result set, labels the reason on $Windy : O$ is finished. The two local reason result $\{Lucy : T, Windy : O\}$ are all finished, therefore Lucy sends message to Jim that the reason on $Lucy : T$ is finished. Similarly, Lucy should deal with the message sent by Bily and Gigi. As shown in Figure 4.

**Step 9:** Finally, user can get reason result: $\{Jim : F, Lucy : T, Lycu : O, Lucy : C, Lucy : I, Lucy : J, Windy : O, Gigi : A, Andy : A \cap Andy : C, Bily : M \cup Bily : HKMT\}$, as shown in Figure 5.

## V. DL-DECA Reasoning Algorithm

Adjiman provides the first consequence searching algorithm in a P2P setting: DeCA [6] [7]. Based on DeCA, a P2P reasoning algorithm suitable for DL called DL-DeCA is proposed by the means of modifying algorithm DeCA according to the definitions and characteristics of DL.

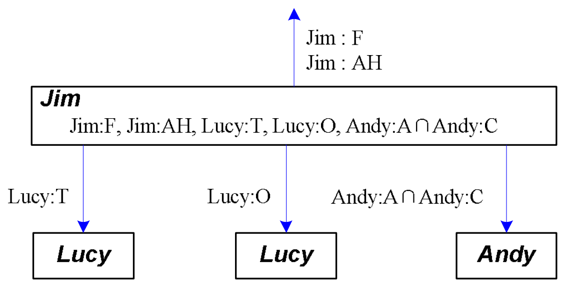


Figure 4. *Reasoning in the second level*

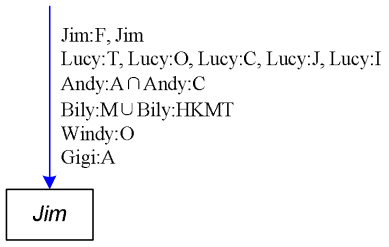


Figure 5. *Reasoning result*

### *A. Algorithm Description*

**Definition 8.** ***Reason(C).*** *As for a legal DL concept $C$, node $p$ executes the result of $Resolvent(C)$ is the $Tbox$ of node $p$, satisfying $D \subseteq C$ or $D \equiv C$.*

**Definition 9.** ***Neighbor(C).*** *As for a legal DL concept $C$, the result of node $p$ executing $Neighbor(C)$ is the all node set sharing with $C$.*

**Definition 10.** ***FindLastItem(queryID,C).*** *Based on data structure $RedvRecord$, find concept $C$ is the reasoning result of $queryID$'s item.*

**Definition 11.** ***FindLastPeer(queryID,C).*** *Based on data structure $RedvRecord$, find which node sends message to local node to reason on concept $C$.*

**Definition 12.** ***ForthMessage.*** *ForthMessage is defined as $\{Sender, Receiver, forth, queryID, lastQuery, C\}$ , which means Sender requests Receiver to reason on $queryID$, C is the local $lastQuery$ reason result processed by Sender.*

**Definition 13.** ***BackMessage.*** *BackMessage is defined as $\{Sender, Receiver, forth, queryID, lastQuery, C, D\}$, which means Receiver has sent message to Sender to reason on $queryID$, and the reason result D Sender sending back to Receiver.*

**Definition 14.** ***FinalMessage.*** *FinalMessage is defined as $\{Sender, Receiver, final, queryID, lastQuery, C\}$, which means Receiver notify Sender, the reason on $queryID$ is finished.*

Algorithm 1 shows how to process $ForthMessage$.

```
Algorithm 1 : DL-DeCA processing ForthMessage
1  if $C \in \text{RecvRecord}(queryID)$
2      send $m(Self, Sender, final, queryID, lastQ, C)$
3  else if exist $C' \in \text{RecvRecord}(queryID)$ having $ComplementOf(C, C')$
4      send $m(Self, Sender, final, queryID, lastQ, C)$
5  else
6      $\text{RecvRecord} \leftarrow \text{RecvRecord} \cup \{(queryID, Sender, lastQ, C)\}$
7      $\text{Local}(queryID, C) \leftarrow Reason(C) \cup \{C\}$
8      $\text{FurtherFlag} \leftarrow false$
9      foreach $D \in \text{Local}(queryID, C)$
10         send $m(Self, Sender, back, queryID, C, D)$
11         if $Neighbor(D) = \varnothing$
12             $FINAL(queryID, D) \leftarrow true$
13         else if $D = C$ and $Neighbor(D) \setminus \{Sender\} = \varnothing$
14             $FINAL(queryID, D) \leftarrow true$
15         else
16             $FINAL(queryID, D) \leftarrow false$
17             $\text{FurtherFlag} \leftarrow true$
18             foreach $p \in Neighbor(D)$
19                 if $not(D = C$ and $p = Sender)$
20                     send $m(Self, p, forth, queryID, C, D)$
21                     $\text{SendRecord} \leftarrow \text{SendRecord} \cup \{(queryID, D, p)\}$
22     if $\text{FurtherFlag} = false$
23         send $m(Self, Sender, final, queryID, lastQ, C)$
```

Firstly local node should check whether this node has reasoned the same request or its supplement (with the same $queryID$). If so, this node send back $FinalMessage$ to finish this reason branch (line 1-4), otherwise continue this process (line 5-23). During processing reason, record reason request (line 6), reason in local (line 7), and label further reason as false (line 8). For each local reason result $D$, send back a $BackMessage$ to request (line 10). If $D$ is not the share variable, then the reason of $D$ is labeled finished (line 11-12). If $D$ is reason request item, and only shared with request, the reason of $D$ is labeled finish (line 13-14). Otherwise, the reason of $D$ is labeled as false (line 16), further reason is labeled as true (line 17). For each neighbor $p$ which sharing $D$, request $p$ further reasons on $D$ (line 20) and record reason result (line 21). When all reason result is finished, if the further reason label is false, then send $FinalMessage$ and finish this request (line 22-23).

```
Algorithm 2 : DL-DeCA processing BackMessage
1  $CONS(queryID, C) \leftarrow CONS(queryID, C) \cup D$
```

It is simple to process $BackMessage$ (shown in algorithm 2), further reason result from neighbor need to be stored as reason result.

Algorithm 3 shows how to process $FinalMessage$. Firstly local node should label the further reason of request $D$ sent by $Sender$ is finished (line 1), then query last reason item $C$ $lastQuery$ and request local node to reason $C$ on node $p$ (line 2-3). ($lastQuery$ is the local reason result $C$

on $p$, $D$ is the local node last result $lastQuery$ on local). If the further reason of all neighbors sharing with $D$ has been finished, then label the reason of request $D$ is finished, meanwhile return the reason result of $D$ back to $p$ one by one. If $p$ is local node, notify interface to display result (4-10). If all of the further reason $lastQuery$ results of local node have been finished, then send $p$ Final message, i.e. reason $lastQuery$ on local node is finished. If $p$ is local node, notify interface to display result (11-15).

```
Algorithm 3 : DL-DeCA processing FinalMessage
1  $PROCESS(queryID, Sender, D) \leftarrow true$
2  $C \leftarrow FindLastQuery(queryID, lastQ)$
3  $p \leftarrow FindLastPeer(queryID, lastQ)$
4  if  foreach  $p \in SendRecord(queryID, D), PROCESS(queryID, p, D) = true$
5      $FINAL(queryID, D) \leftarrow true$
6      foreach  $D' \in CONS(queryID, D)$
7          if  $p = Self$
8              $ShowResult(D')$
9          else
10             send  $m(Self, p, back, queryID, lastQ, D')$
11 if  foreach  $E \in LOCAL(lastQ), FINAL(queryID, E) = true$
12     if  $p = Self$
13         $ShowComplete(lastQ)$
14     else
15         send  $m(Self, p, final, queryID, C, lastQ)$
```

```
Algorithm 4 : DL-DeCA Reasoning in Local
1 ShallowRusult ← GetSubclasses(C) ∪ GetEquivalentClasses(C)
2 RESULT ShallowRusult
3 foreach D ∈ ShallowRusult
4     RESULT ← RESULT ∪ Reason(D)
5 return RESULT
```

Algorithm 4 shows how to reason in local node. Firstly we get the initial result of all direct sub-concept and equal concept, then add to result set. Subsequently, local reason for each concept is called recursively, reason results are collected and stored as result set.

### B. Algorithm analysis

Clearly, Algorithm 4 is related to $subsumption\ reasoning$ of DL. Because DL is decidable, furthermore algorithm 4, 5, and 6 are not related to reasoning process, it is clear the DL-DeCA is decidable for only one node. In order to prove DL-DeCA is decidable for more than one node, it is necessary to prove the reasoning length is limited.

Firstly, reasoning tree should be introduced. Reasoning tree is a multi-branch tree, whose node is peer with allowing duplicating and arc is labeled as $ForthMessage$ sent from sender to receiver. The request user submitting is regarded as the root node of reasoning tree. $ForthMessage$ is sent from root node. when node $P_i$ sends a $ForthMessage$ to $P_j$, concept $C$ should be asked to reason, meanwhile node $P_j$ and an arc linking $P_i$ and $P_j$ is added into this reason tree. One triple $< P_i, P_j, C >$ corresponds to an arc of the reason tree. Therefore, the reasoning process is the process of generating reasoning tree. Example described in section IV is shown in Figure 6.

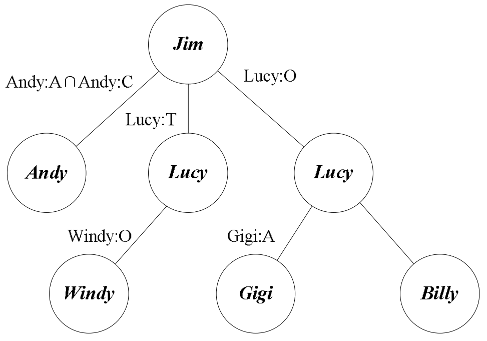


Figure 6. *Reasoning tree of the example shown in section IV*

Because the number of node in DL-P2PRS is limited, the number of triple $< P_i, P_j, C >$ is limited. Because there is no loop reason, i.e. there do not exit duplicated arcs, it is clear that the number of arcs in reasoning tree is limited. Clearly, the length of reasoning trees is limited. By the means of algorithm 4, when a peer receive a $ForthMessage$, it will reason in local, if so, this reasoning branch can be finished. It is easy to conclude that reasoning process can be finished in limited step.

## VI. DL-P2PRS IMPLEMENT

The P2P reasoning system reasoning process is triggered by message-driven process, with which transmission can be concluded with delivering messages.

### A. Reasoning Message

Reasoning message is used to pass reasoning request between nodes, return result and notify the reason is finished. (shown in Table II).

**Table Ⅱ.** *Reasoning message description*

| Type | Message Content | Description |
|---|---|---|
| ForthMesg | sender, receiver, forth, queryID, lastQuery, query | Sender requests receiver to reason on query. |
| BackMesg | sender, receiver, back, queryID, query, cons | Sender returns the reasoning result cons of query to receiver. |
| FinalMesg | sender, receiver, final, queryID, lastQuery, query | Sender notices receiver that sender has finished the task of reasonin. |
| IndividualsRequestMesg | sender, receiver,individualsRequest, concept | Sender requests receiver return the instance of concept to receiver. |
| IndividualsReplyMesg | sender, receiver, ndividualsReply, concept, Set<individual> | Sender returns the instance of concept in knowledge to receiver. |

### B. Control Message

Control message consists of three parts, network topology maintenance, concept consistence maintenance, and mapping maintenance, which can maintain online node list and its mapping. (Shown in Table III)

**Table Ⅲ.** *Network topology maintenance message*

| Type | Message content | Description |
|---|---|---|
| JoinMesg | sender, online | Notice other node sender to join network by UDP message. |
| JoinReplyMesg | sender, receiver, joinReply | Sender replies receiver that sender is online. |
| KeepAliveMesg | sender, receiver, keepAlie | Sender requests receiver to return back aliveReplyMesg, meaning that receiver can work. |
| AliveReplyMesg | sender, receiver, aliveReply | Sender replies KeepAliveMesg to receiver，meaning that sender can work. |
| QuitMesg | sender, receiver, offline | Sender notifies receiver that sender will leave out of network. |

When mapping relationship between nodes is built, the initial node should store mapping information. Part variable of other nodes is stored and maintained in local knowledge base. Concept consistence maintenance is used to maintain the ontology of local node, such as concept renaming, concept deleting, etc. (Shown in Table IV)

**Table Ⅳ.** *Concept consistance maintenance message*

| Type | Message content | Description |
|---|---|---|
| ConceptRenameMesg | sender, receiver, rename, oldName, newName | Sender notifies receiver that the oldName is renamed as newName. |
| ConceptDeleteMesg | sender, receiver, rename, concept | sender notifies receiver that the concept has been deleted and receiver should do either. |

When one node prepares to build mapping relationship with another node, it should ask another node to send back its ontology concept information. On the other hand, if one node wants to delete mapping relationship, it should notify another to delete some share variables between them either. (Shown in Table V)

**Table Ⅴ.** *Mapping maintenance message*

| Type | Message content | Description |
|---|---|---|
| OntologyRequestMesg | sender, receiver, askForOntology | Sender requests Receiver to return back ontology and other information. |
| OntologyReplyMesg | sender, receiver, ontology | Sender send ontology and other information to Receiver |
| MappingRequestMesg | sender, receiver, mappingRequest, mapping | Sender requests receiver to build mapping in each other. |
| MappingReplyMesg | sender, receiver, mappingReply, mapping, reply | Sender and receiver replies the mapping request：reply=0 reject；reply=1 accept. |
| MappingDeleteMesg | sender, receiver, deleteMapping, mapping | Sender notifies receiver to delete mapping. |

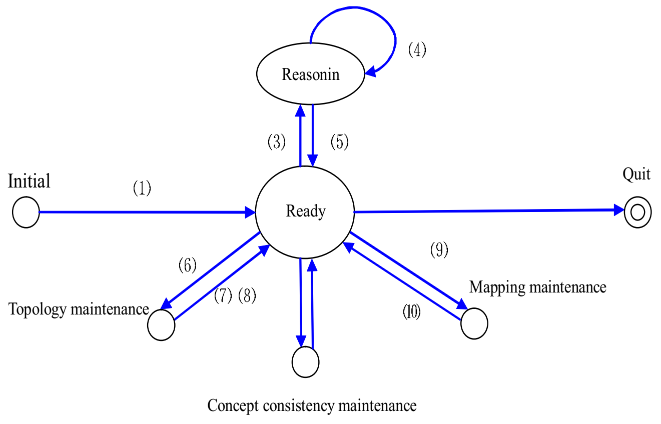
## C. System State Transition Diagram

Figure 7 shows message driven state transition diagram in DL-P2PRS when one node is added or quit.

## D. System Implement

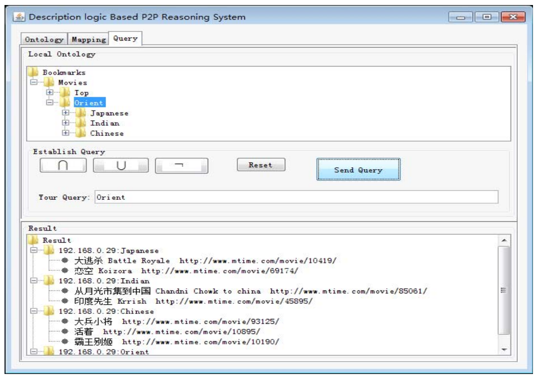
We developed DL-P2PRS with Java, which implements all of the functions described above. In order to simplifying, the collected network resource comprises two attribute: resource name and URL. Knowledge structure is organized as multi-level contents, with the functions of adding, modifying, deleting, etc. (Shown in Figure 8)

Figure 9 shows that if node Jim receives a query on Favorite, describing in section IV, expected return result is $\{Jim : Favorite, Jim : ActionHumorous, Lucy : Top,$ $Lucy : Orient, Lucy : Chinese, Lucy : Japanese,$ $Lucy : Indian, Andy : A \cap Andy : C\}$.



(1) send JoinMesg.
(2) send QuitMesg.
(3) receive ForthMesg.
(4) send ForthMesg, receive BackMesg, receive FinalMesg, send BackMesg.
(5) send FinalMesg.
(6) receive JoinMesg. receive KeepAliveMesg. System call. receive QuitMesg.
(7) send JoinReplyMesg. send AliveReplyMesg. send KeepAliveMesg. finish.
(8) receive ConceptRenameMesg, receive ConceptDeleteMesg.
(9) receive MappingRequestMesg. receive MappingDeleteMesg.
(10) send MappingReplyMesg. finish MappingDeleteMesg.

Figure 7. *DL-P2PRS message-driven state transition diagram*



Your Query: Orient

```
Result
Result
192.168.0.29:Japanese
    大逃杀 Battle Royale  http://www.mtime.com/movie/10419/
    恋空 Koizora  http://www.mtime.com/movie/69174/
192.168.0.29:Indian
    从月光市集到中国 Chandni Chowk to china  http://www.mtime.com/movie/85061/
    印度先生 Krrish  http://www.mtime.com/movie/45895/
192.168.0.29:Chinese
    大兵小将  http://www.mtime.com/movie/93125/
    活着  http://www.mtime.com/movie/10895/
    霸王别姬  http://www.mtime.com/movie/10190/
192.168.0.29:Orient
```

Figure 8. *Ontology maintenance interface*

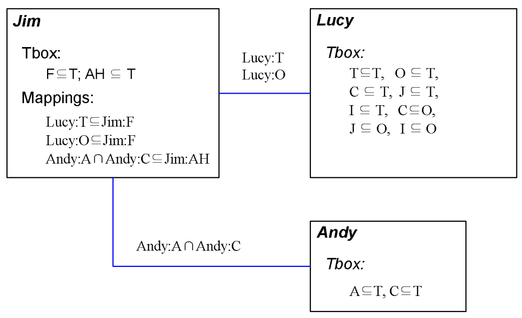
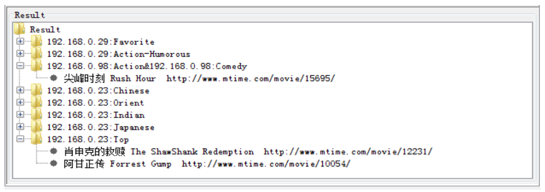


Figure 9. *DL-P2PRS example: composed of node Jim, Lucy, and Andy*



```
Result
Result
  192.168.0.29:Favorite
  192.168.0.29:Action-Humorous
  192.168.0.98:Action&192.168.0.98:Comedy
    ● 尖峰时刻 Rush Hour  http://www.mtime.com/movie/15695/
  192.168.0.23:Chinese
  192.168.0.23:Orient
  192.168.0.23:Indian
  192.168.0.23:Japanese
  192.168.0.23:Top
    ● 肖申克的救赎 The ShawShank Redemption  http://www.mtime.com/movie/12231/
    ● 阿甘正传 Forrest Gump  http://www.mtime.com/movie/10054/
```

Figure 10. *DL-P2PRS example: reasoning result of node Jim*

Test the example in our system, node Jim's IP is 192.168.0.29, node Lucy's is 192.168.0.23, and node Andy's IP is 192.168.0.98. Reasoning result shown in Figure 10 demonstrates that our system can work correctly and effectively.

## VII. CONCLUSION

This paper proposed a P2P reasoning algorithm suitable for description logic DL-DeCA. Based on DL-DeCA, the P2P reasoning system DL-P2PRS is implemented, by reasoning message and control message design, as well as system maintenance mechanisms. Experiment results show that DL-P2PRS can work correctly and effectively.

There are several issues to be addressed: maintenance of shared-variable model should be preserved consistency of models between neighbor peers, if reasoning with conflict knowledge, consequence found should become meaningless. We will do some research on conflict detection algorithm which can effectively detect tree-like conflict or cycle-with conflict.

## ACKNOWLEDGMENT

This work is Supported by the National Natural Science Foundation of China under Grant Nos.60573160,60873162; the Natural Science Foundation of Guangdong Province under Serial No.10151027501000028; the Research Foundation of Science and Technology Plan Project in Guangzhou City under Grant No.2010Z1-D00041; the Fundamental Research Funds for the Central Universities under Grant No.10000-3165002.

Many thanks to our colleagues and the same thanks to the anonymous reviewers for their helpful suggestions that improved the quality of this paper.

## REFERENCES

[1] Jon Curtis, Gavin Matthews, and David Baxter. On the effective use of cyc in a question answering system. In *Proceedings of the Joint Conference on Artificial Intelligence (IJCAI) Workshop on Knowledge and Reasoning for Answering Questions*, 61–70, 2005.

[2] Richard Fikes and Adam Farquhar. Large-scale repositories of highly expressive reusable knowledge. *IEEE Intelligent Systems*, 14:1-16, 1999.

[3] Eyal Amir and Sheila McIlraith. Partition-based logical reasoning. In *Proceedings of the 11th International Conference on Principles of Knowledge Representation and Reasoning (KR 2000)*, 389–400, 2000.

[4] Eyal Amir and Sheila McIlraith. Partition-based logical reasoning for first-order and propositional theories. *Artificial Intelligence*, 162:49–88, February 2005.

[5] Francois Goasdou and Marie-Christine Rousset. Querying distributed data through distributed ontologies: a simple but scalable approach. *IEEE Intelligent Systems*, 18(5):60–65, 2003.

[6] P. Adjiman, P. Chatalic, F. Goasdou, M. C. Rousset, and L. Simon. Distributed reasoning in a peer-to-peer setting. In *Proceedings of The 16th European Conference on Artificial Intelligence (ECAI 2004)*, 945–946, 2004.

[7] P. Adjiman, P. Chatalic, F. Goasdou, M. C. Rousset, and L. Simon. Scalability study of peer-to-peer consequence finding. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI 2005)*, 351–356, 2005.

[8] Salem Benferhat, Didier Dubois, and Henri Prade. Some syntactic approaches to the handling of inconsistent knowledge bases: A comparative study part 1: The flat case. *Studia Logica*, 58(1):17–45, 1997.

[9] Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki. Consistent query answers in inconsistent databases. In *Proceedings of the Eighteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS 1999)*, 68–79, 1999.

[10] Leopoldo Bertossi. Consistent query answering in databases. *SIGMOD Rec.*, 35:68–76, June 2006.

[11] Leopoldo Bertossi and Loreto Bravo. Query answering in peer-to-peer data exchange systems. In *Proceedings of the 9th International Conference on Extending Database Technology (EDBT) Workshop on Peer-to-Peer Computing and Databases*, 476–485, 2004.

[12] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Inconsistency tolerance in P2P data integration: an epistemic logic approach. *Information Systems*, 33(4-5):360–384, 2008.

[13] Leopoldo Bertossi and Jan Chomicki. Query answering in inconsistent databases. *Logics for Emerging Applications of Databases*, 1–49, 2003.

[14] Ph. Chatalic, G. H. Nguyen, and M. Ch. Rousset. Reasoning with inconsistencies in propositional peer-to-peer inference systems. In *Proceedings of 17th European Conference on Artificial Intelligence (ECAI 2006)*, 352–356, 2006.

[15] Arnold Binas and Sheila A. McIlraith. Peer-to-peer query answering with inconsistent knowledge. In *Proceedings of the 11th International Conference on Principles of Knowledge Representation and Reasoning (KR 2008)*, 329–339, 2008.

[16] Daniele Nardi and Ronald J. Brachman. An introduction to description logics. In *The Description Logic Handbook. Theory, Implementation and Applications*, 1–40. 2003.

[17] Ian Horrocks. The fact system. In *Proceedings of the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX98)*, 307–312, 1998.

[18] Goutam Kumar Saha. Web ontology language (owl) and semantic web. *Ubiquity*, 2007:1–1, 9 2007.