Design and implementation of P2P reasoning system based on description logic

Hai Wan*, Yang Yu and Jian-tian Zheng

Software School,
Sun Yat-Sen University,
Guangzhou 510275, China
Email: wanhai@mail.sysu.edu.cn
Email: yuy@mail.sysu.edu.cn
Email: zsu_tim@163.com
*Corresponding author

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 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.

Reference to this paper should be made as follows: Wan, H., Yu, Y. and Zheng, J. (2013) 'Design and implementation of P2P reasoning system based on description logic', *Int. J. Ad Hoc and Ubiquitous Computing*, Vol. 13, No. 2, pp.140–154.

Biographical notes: Hai Wan has done his PhD degree in Computer Software and Theory from Sun Yat-Sen University, Guangzhou, PRC, in 2006. He is presently working as a Senior Lecturer in the School of Software, Sun Yat-Sen University, Guangzhou, PRC. He is a CCF member, an ACM member, and an IEEE member. His research interest includes knowledge representation and reasoning, intelligence diagnosis, P2P reasoning, and code automata debugging and repair.

Yang Yu is now a full Professor of School of Software, Sun Yat-sen University, Guangzhou, PRC. He is a CCF senior member and an ACM member. His research interests include workflow, service computing, social computing and software engineering. He received his PhD degree in Computer Software and Theory from Sun Yat-sen University, China, in 2007. Before he devoted to academic research and teaching in SYSU 10 years ago, he had 11 years of experience in software industry. He is a reviewer of IEEE Transactions on Automation Science and Engineering and Journal of Supercomputing.

Jian-tian Zheng received his BTech, MTech degrees in School of Software, Sun Yat-Sen University, Guangzhou, PRC in 2008, 2010, respectively. He is presently a Software Engineer in China Guangdong Nuclear Power Group.

1 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 (Yuan et al., 2011; Zhang et al., 2011). 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.

Many description logic (DL) models can be considered as decidable fragments of first order logic and provide a logical formalism and reasoning tools for ontology and semantic web. Therefore this paper concentrates on P2P reasoning technology based on 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 Description logic P2P reasoning system, called DL-P2PRS, 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 organised as follows. In the next section, we discuss related works, including P2P technologies, 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, control message, and system maintenance mechanisms of DL-P2PRS are given in section 6.

2 Related works

In P2P reasoning system, each peer node maintains a knowledge base. When asked for query, peer should reason on local knowledge base and propagate through mappings relationship between neighbour peers in entire P2P network. Some P2P reasoning systems present this feature, such as Cycorp's Cyc (Curtis et al., 2005) and HPKB (Fikes and Farquhar, 1999).

Because logical reasoning in P2P reasoning system is processed in collaboration with each peer 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 (Amir and McIlraith, 2000, 2005). Based on (Amir and McIlraith, 2000, 2005), Goasdoue's approach is characterised by a simple class-based language for defining peer schemas as hierarchies of atomic classes (Goasdou and Rousset, 2003). Adjiman provides the first consequence finding algorithm in a peer-to-peer setting, named as DeCA, which can compute sequences gradually from the solicited peer to more and more distant peers (Adjiman et al., 2004, 2005).

The second problem in P2P reasoning is how to eliminate conflict and get reasonable conclusion between peer nodes. Benferhat's algorithms can combine answers from different peers so as to compute implicates involving target variables (Benferhat et al., 1997). 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 (Arenas et al., 1999). 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 (Bertossi and Bravo, 2004; Bertossi, 2006). Calvanese studies query answering in new nonmonotonic logic, which can establish its decidability and computational complexity (Calvanese et al., 2008). Bertossi describes different approaches to the issue of computing consistent query answers (Bertossi and Chomicki, 2003). Adjiman's 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 (Adjiman et al., 2004; Adjiman et al., 2005). Chatalic exhibits a distributed algorithm detecting inconsistencies in a fully decentralised propositional P2P inference systems (Chatalic et al., 2006). Binas provides a formal characterisation of a prioritised 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 (Binas and McIlraith, 2008).

3 Description logic and OWL reasoning

3.1 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 (Nardi and Brachman, 2003). 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). 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. Figure 1 shows typical DL grammar and semantics.

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 optimisation techniques, more advanced current DL systems have been developed. FaCT (Horrocks, 1998) is the most significant example.

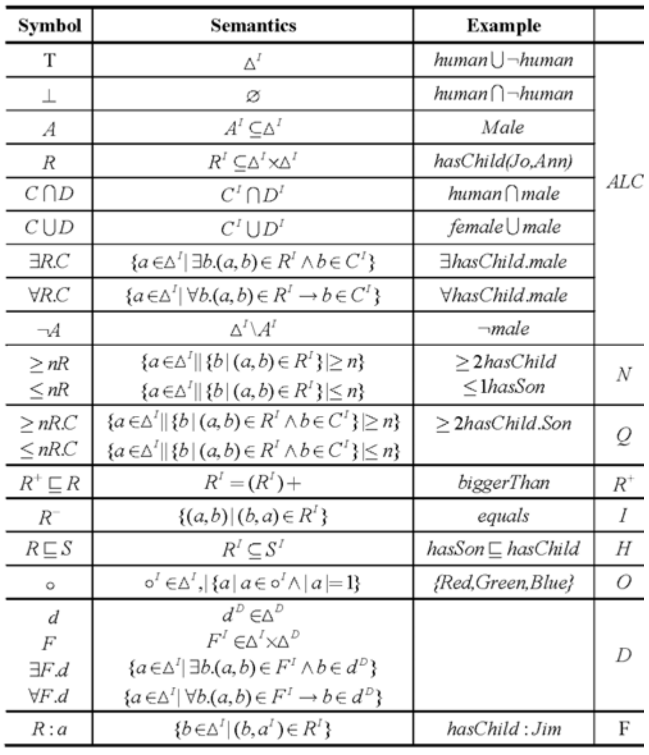


Figure 1 DL grammar and semantics

Symbol	Semantics	Example	
T	Δ^I	$human \sqcup \neg human$	*ALC*
$\bot$	$\varnothing$	$human \sqcap \neg human$	
A	$A^I \subseteq \Delta^I$	*Male*	
R	$R^I \subseteq \Delta^I \times \Delta^I$	*hasChild(Jo,Ann)*	
$C \sqcap D$	$C^I \cap D^I$	$human \sqcap male$	
$C \sqcup D$	$C^I \cup D^I$	$female \sqcup male$	
$\exists R.C$	$\{a \in \Delta^I \mid \exists b.(a,b) \in R^I \wedge b \in C^I\}$	$\exists hasChild.male$	
$\forall R.C$	$\{a \in \Delta^I \mid \forall b.(a,b) \in R^I \rightarrow b \in C^I\}$	$\forall hasChild.male$	
$\neg A$	$\Delta^I \setminus A^I$	$\neg male$	
$\geq nR$ $\leq nR$	$\{a \in \Delta^I \mid \lvert\{b \mid (a,b) \in R^I\}\rvert \geq n\}$ $\{a \in \Delta^I \mid \lvert\{b \mid (a,b) \in R^I\}\rvert \leq n\}$	$\geq 2 hasChild$ $\leq 1 hasSon$	*N*
$\geq nR.C$ $\leq nR.C$	$\{a \in \Delta^I \mid \lvert\{b \mid (a,b) \in R^I \wedge b \in C^I\}\rvert \geq n\}$ $\{a \in \Delta^I \mid \lvert\{b \mid (a,b) \in R^I \wedge b \in C^I\}\rvert \leq n\}$	$\geq 2 hasChild.Son$	*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 F $\exists F.d$ $\forall F.d$	$d^D \in \Delta^D$ $F^I \in \Delta^I \times \Delta^D$ $\{a \in \Delta^I \mid \exists b.(a,b) \in F^I \wedge b \in d^D\}$ $\{a \in \Delta^I \mid \forall b.(a,b) \in F^I \rightarrow b \in d^D\}$		*D*
$R : a$	$\{b \in \Delta^I \mid (b,a^I) \in R^I\}$	$hasChild : Jim$	F

3.2 *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 recognised as a member of the class. In addition, the OWL reasoning service provides a way to perform automatic classification (Saha, 2007).

4 DL-P2PRS framework

For the purpose of reasoning in P2P network, DL-based P2P Reasoning System based on DL, called as DL-P2PRS, is presented in this section. DL-P2PRS consists of peer nodes which maintain local DL-based knowledge base and communicate with each other by the means of mapping relationship. When one peer node is asked for query, it will reason in local knowledge base and propagate query through mappings relationship between neighbour peers in entire P2P network, finally return reasoning result back to the initial node. Prior to describing how to use DL-P2PRS, some definitions are given as follows.

4.1 *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 relationship 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 system.

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 $mapping_{i,j}$.

After constructing DL-P2PRS system, the semantics of which can be described by four aspects in the following definition 5.

Definition 5 DL-P2PRS semantics.

1 The interpretation of a DL-P2PRS $P = \bigcup_{i=1\cdots n} P_i$ is the interpretation set of all nodes, i.e. $I = \bigcup_{i=1\cdots n} I_i$;

2 The schema definition of P is $Schema(p) = \bigcup_{i=1\cdots n} TBox_i \cup ABox_i \cup Map_i$;

3 If an interpretation of P satisfies $Schema(p)$, regarded I as the model of P;

4 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 each model I of P, there is $D^I \subseteq C^I$ or $D^I = C^I$, then regards D as implication concept of C. i.e. if concept D is a implication concept of C, then there exits a transition link in P, in which the number of D and C is more than or equal to zero, any $E \sqsubseteq F$ (or $E \equiv F$) in transition link 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). Reasoning problem in Γ can be defined as: given a legal *DL* concept C, find all implication concept D as well as instance set of D in Γ.

4.2 DL-P2PRS framework description

Figure 2 shows DL-P2PRS framework using MVC design principle, with which interface, business logic, and data storage can be designed hierarchically. DL-P2PRS is comprised of the following modules:

1 Interface module: including knowledge base maintenance interface, mapping maintenance interface, and reasoning query interface. Knowledge base maintenance interface queries local knowledge base, add, modify, and delete classes or instances. Mapping maintenance interface displays current mapping, build mapping and delete mapping. Reasoning query interface provides user the query interface and return querying result.

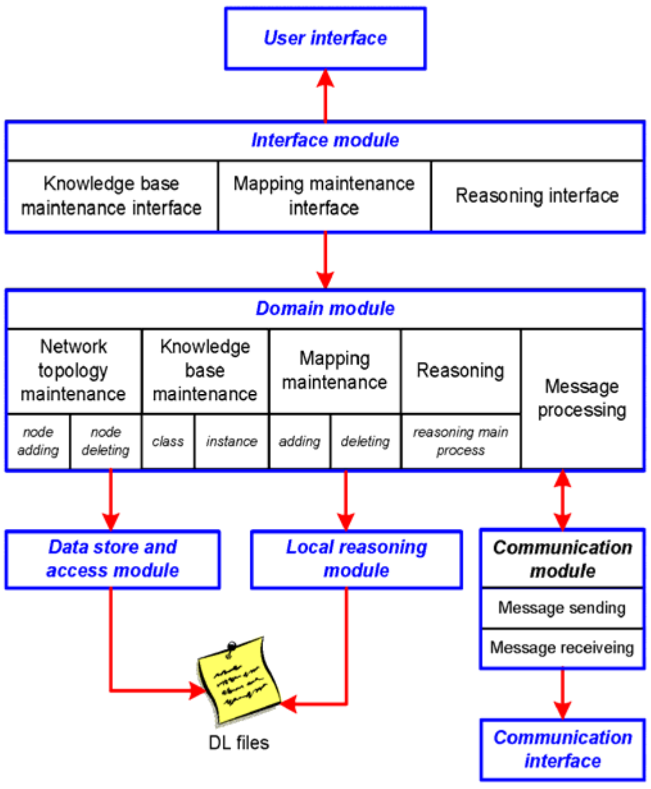
2 Domain module: is composed of business related interfaces and classes, which is the core module of the DL-P2PRS, responsible for maintaining network topology, knowledge base, mapping relationship, distributed reasoning, and processing various types of messages.

3 Local reasoning module: providing local reasoning in local knowledge base.

4 Communication module: defining system communication protocol interface, responsible for sending and receiving messages.

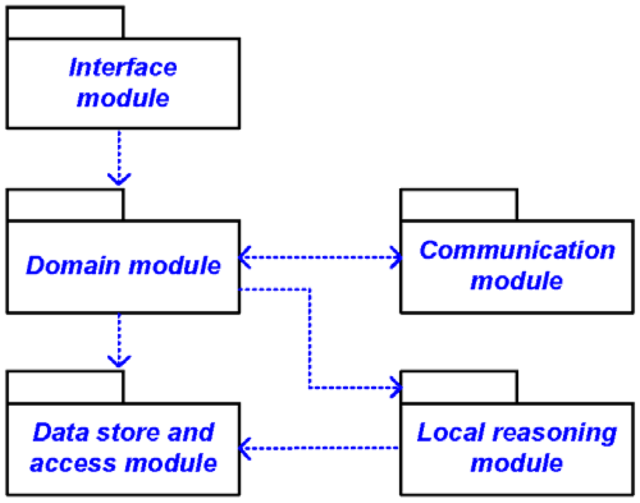
5 Data store and access module: defining data storage interface, type and class, providing the OWL data files creating, accessing, adding, modifying, and deleting functions.

Figure 2 DL-P2PRS framework (see online version for colours)



The dependency relationship between various modules is shown in Figure 3.

Figure 3 DL-P2PRS module dependency (see online version for colours)



4.3 Typical example

Following the above definitions, a typical example with 6 peer nodes is shown how to use DL-P2PRS. This example is composed of 6 peer nodes, including Jim, Lucy, Andy, Bily, Gigi, and Windy, all of which are movie lovers, collecting movie information and saving URL in each peer node (shown in Figure 4):

1 Jim collects and organises movie information as two categories:

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

2 Lucy focuses on high level movies and likes the Oriental culture. She organises as follows, in which $C \sqsubseteq O$, $J \sqsubseteq O$, $I \sqsubseteq O$.

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

3 Andy organises his movies according to the following 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 organises 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 the Oriental she defines, so she claims $Lucy:O \equiv Gigi:A$.

We can show how to reason in this DL-P2PRS. Assuming user sends a query request in node Jim, shown in Figure 5. DL-P2PRS can reason as the following steps:

Step 1: 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 the share variables between Jim and Lucy, $Andy:A \cap Andy:C$ is the 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 reasoning request message to node Andy, asking Andy to reason $Andy:A \cap Andy:C$.

Figure 4 DL-P2PRS: an example with 6 peer nodes (see online version for colours)

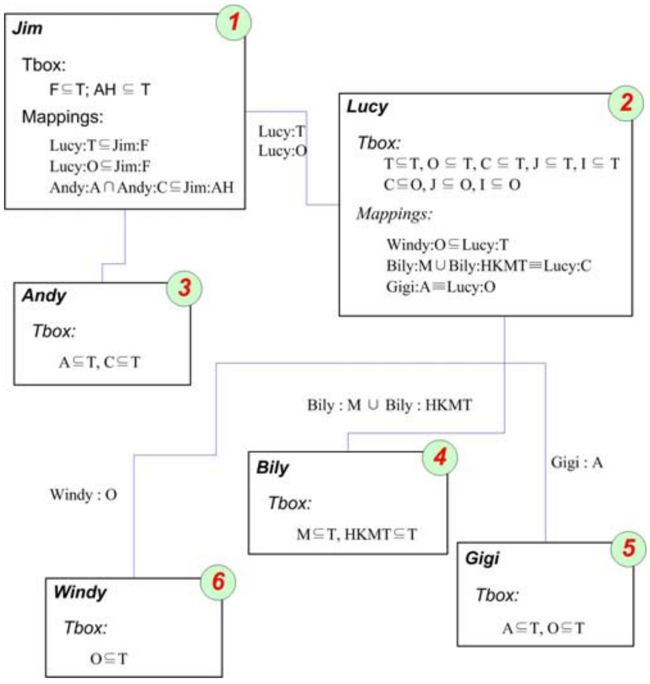


Figure 5 User sends a reasoning request in node Jim (see online version for colours)

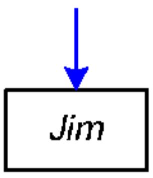
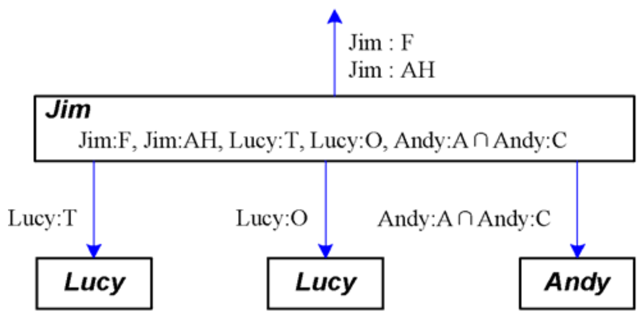


Figure 6 shows reasoning in the first level, including step 1.

Figure 6 Reasoning in the first level (see online version for colours)



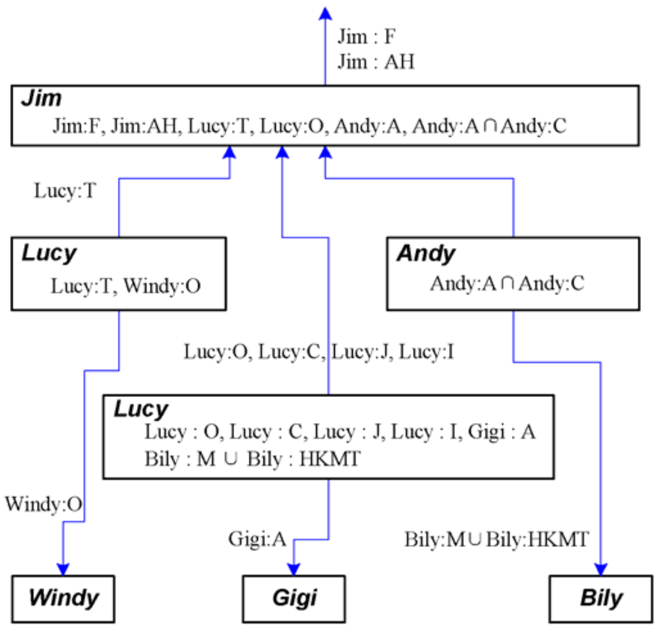
Step 2: Node Lucy receives the message $Lucy:T$ sent by Jim and processes as follows: reason $Lucy:T$ in local, and get the result $\{Lucy:T, Windy:O\}$, in which $Lucy:T$ is the share variable between Lucy and Jim, $Windy:O$ is the share variable between Lucy and Windy. Because Jim shares $Lucy:T$ with 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 the 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.

Figure 7 describes reasoning in the second level, including step 2–4.

Figure 7 Reasoning in the second level (see online version for colours)


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

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 neighbour of Windy sharing *Windy* : *O*, Windy return back to Lucy that the reason on *Windy* : *O* is finished.

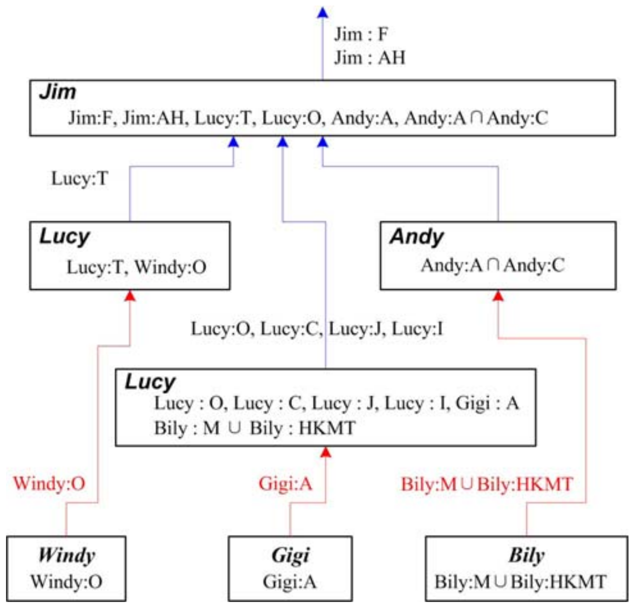
Step 6: Node Gigi receives message from Lucy about reason on $Gigi:A$, Gigi should reason in local and the result is $\{Gigi:A\}$. Just like Step 5, Gigi returns back to Lucy that reason on $Gigi:A$ is finished.

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 reasoning 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.

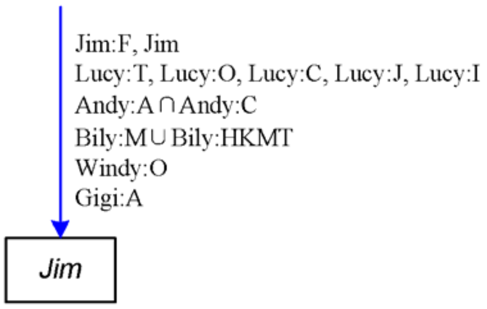
Figure 8 describes reasoning in the third level, including step 5–8.

Figure 8 Reasoning in the third level (see online version for colours)



Step 9: Finally, the user can get reasoning 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 9.

Figure 9 Reasoning result (see online version for colours)



5 DL-DECA reasoning algorithm

Adjiman provides the first consequence searching algorithm in a P2P setting: DeCA (Adjiman et al., 2004; Adjiman et al., 2005). 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.

5.1 Algorithm description

In order to depict the DL-DeCA reasoning algorithm in detail, some definitions of processes and related messages should be given firstly.

Definition 8 Reason(C).

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

Definition 9 Neighbour(C).

As for a legal DL concept *C*, the results of peer node *p* executing *Neighbour*(*C*) is the all node set sharing with *C*.

Definition 10 FindLastItem(queryID,C).

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

Definition 11 FindLastPeer(queryID,C).

Based on data structure *RedvRecord*, *FindLastPeer* (*queryID*,*C*) means 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* reasoning 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 reasoning result *D Sender* sending back to *Receiver*.

Definition 14 FinalMessage.

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

When one reasoning with DeCA, the more depths it reasons, the more levels it processes, and the length of reasoning record becomes longer. Theoretically, as for a P2P reasoning system with *n* nodes, the length of reasoning record should be *n*. It is clear that the message should increase network burdens, especially for large-scale network.

Compared with DeCA (Adjiman et al., 2004; Adjiman et al., 2005), considering the characteristics of DL, DL-DeCA is improved by adding some data structure in each peer nodes as shown in the above definitions, which has the following advantages:

1 Because the concept name in each node is unique, it can avoid generating conflict result.
2 Reasoning results can be stored in each peer node. When receiving a reasoning request, each node should check and reason in local firstly, which helps avoiding loop reasoning.
3 By adding request reasoning information in *Back* and *Final* message, it can help finding the information in the last node when returning *Back* message.

DL-DeCA is composed of 4 algorithms, including processing *ForthMessage*, processing *BackMessage*, processing *FinalMessage*, and reasoning in local algorithm.

Algorithm 1 shows how to process *ForthMessage*. 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 reasoning result *D*, send back a *BackMessage* to request (line 10). If *D* is not the share variable, then the reason of *D* is labelled finished (line 11-12). If *D* is reason request item, and only shared with request, the reason of *D* is labelled finish (line 13-14). Otherwise, the reason of *D* is labelled as false (line 16), further reason is labelled as true (line 17). For each neighbour *p* which sharing *D*, request *p* further reasons on *D* (line 20) and record reasoning result (line 21). When all reasoning result is finished, if the further reason label is false, then send *FinalMessage* and finish this request (line 22-23).

It is simple to process *BackMessage* (shown in algorithm 2), further reasoning result from neighbour should be stored as reasoning 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 reasoning result *C* on *p*, *D* is the local node reasoning result *lastQuery* on local).

If the further reason of all neighbours sharing with *D* has been finished, then label the reason request *D* is finished, meanwhile return the reasoning 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 1: DL-DeCA processing *ForthMessage*

```
1  if  C ∈ RecvRecord(queryID)
2      send  m(Self, Sender, final, queryID, lastQ, C)
3  else if  exist  C' ∈ RecvRecord(queryID)  having  ComplementOf(C, C')
4      send  m(Self, Sender, final, queryID, lastQ, C)
5  else
6      RecvRecord ← RecvRecord ∪ {(queryID, Sender, lastQ, C)}
7      Local(queryID, C) ← Reason(C) ∪ {C}
8      FurtherFlag ← false
9      foreach  D ∈ Local(queryID, C)
10         send  m(Self, Sender, back, queryID, C, D)
11         if  Neighbor(D) = ∅
12             FINAL(queryID, D) ← true
13         else if  D = C  and  Neighbor(D) \ {Sender} = ∅
14             FINAL(queryID, D) ← true
15         else
16             FINAL(queryID, D) ← false
17             FurtherFlag ← true
18             foreach  p ∈ Neighbor(D)
19                 if  not(D = C  and  p = Sender)
20                     send  m(Self, p, forth, queryID, C, D)
21                     SendRecord ← SendRecord ∪ {(queryID, D, p)}
22     if  FurtherFlag = false
23         send  m(Self, Sender, final, queryID, lastQ, C)
```

Algorithm 2: DL-DeCA processing *BackMessage*

```
CONS(queryID;C) ← CONS(queryID;C) ∪ D
```

Algorithm 3: DL-DeCA processing FinalMessage

```
1  PROCESS(queryID, Sender, D) ← true
2  C ← FindLastQuery(queryID, lastQ)
3  p ← FindLastPeer(queryID, lastQ)
4  if  foreach  p ∈ SendRecord(queryID, D), PROCESS(queryID, p, D) = true
5      FINAL(queryID, D) ← true
6      foreach  D' ∈ 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 ∈ 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 shows how to reason in local node. Firstly we get the initial result of all direct sub-concept and equal concept and add to result set. Subsequently, local reason for each concept is called recursively, reasoning results are collected and stored as result set.

Algorithm 4 DL-DeCA reasoning in local

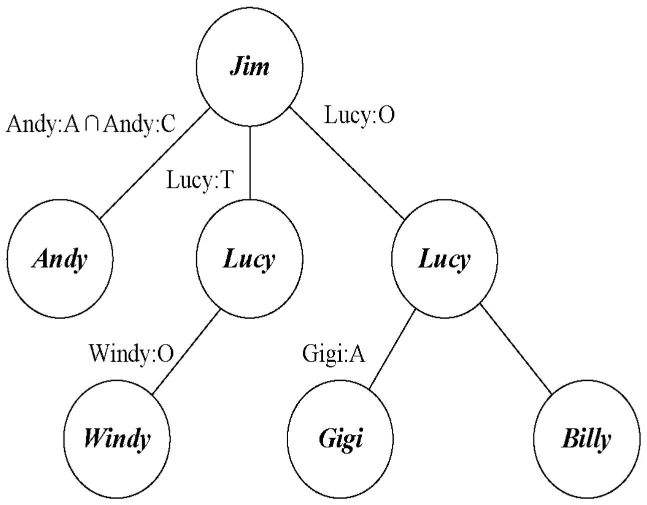
```
1 ShallowRusult ← GetSubclasses(C) ∪ GetEquivalentClasses(C)
2 RESULT ShallowRusult
3 foreach D ∈ ShallowRusult
4     RESULT ← RESULT ∪ Reason(D)
5 return RESULT
```

5.2 Algorithm decidable analysis

Clearly, Algorithm 4 is related to the *subsumption reasoning* of DL. Because DL is decidable, furthermore algorithm 1, 2, and 3 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 duplicated and arc is labelled 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 10.

Figure 10 Reasoning tree of the example in section IV



Because the number of nodes in DL-P2PRS is limited, the number of triples $<P_i, P_j, C>$ is limited. Because there is no loop reason, i.e. there do not exist 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 receives 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.

5.3 Algorithm soundness and completeness

By the definition of *implication concept* (definition 6), the soundness and completeness of algorithm DL-DeCA can be given in theorem 1 and 2 respectively.

Theorem 1: *(Soundness) Given any model I of a DL-P2PRS and a query concept C, the result D given by DL-DeCA (algorithm 1-4) satisfies $D^I \sqsubseteq C^I$ or $D^I = C^I$.*

Proof: The soundness proof can be transferred as: given any model *I* of a DL-P2PRS and a query concept *C*, there exists relationship transition link $D \vee \ldots \vee C$, where $\vee$ means $\sqsubseteq$ or $\equiv$, and the number of concepts in the link more than or equal to one.

The proof can be given by mathematical induction.

1 if the node p_1 is the first peer node of the reasoning path, the return result is $LOCAL(C, p_1) = Reason(C, p_1) \cup \{C\}$. From line 7 in algorithm 1 and algorithm 4, it is clear that for each $D \in LOCAL(C, p_1)$ there exists $D \vee \ldots \vee C$.

2 if the node p_k is the *k* peer node in the reasoning path, assuming the return result from p_k satisfies $D \vee \ldots \vee C$.

3 if the node p_{k+1} is the $k+1$ peer node in the reasoning path, there exist the last two node p_k and p_{k+1}. Assume the correlation mapping between p_k and p_{k+1} is $C_{k+1} \vee C_k$, where C_{k+1} is the concept defined by p_{k+1}, and C_k is the reasoning result satisfying $C_k \vee \ldots \vee C$.

If $C_{k+1} \vee C_k$ is built by p_k, C_{k+1} is the share variable. Because $C_{k+1} \vee C_k$ is stored in the knowledge base p_k. It is clear that there exists $C_{k+1} \vee \ldots \vee C$ by $C_k \vee \ldots \vee C$ and $C_{k+1} \vee C_k$. Because p_k send *Forth* massage $m(p_k, p_{k+1}, forth, id, lastQ, C_{k+1})$, p_{k+1} cannot get new reasoning result when the reasoning process come into end by the line 1 and 3 in algorithm 1. By algorithm 4, as for each $C' \in LOCAL(C_{k+1}, p_{k+1})$, there exists $C' \vee \ldots \vee C_{k+1}$, while $C_{k+1} \vee \ldots \vee C$, then $C' \vee \ldots \vee C$ is satisfied.

If $C_{k+1} \vee C_k$ is built by p_{k+1}, C_{k+1} is the share variable. Because p_k send *Forth* massage $m(p_k, p_{k+1}, forth, id, lastQ, C_{k+1})$ to p_{k+1}. It is clear p_{k+1} can not get new reasoning result when the reasoning process come into end by the line 1 and 3 in algorithm 1. By algorithm 4, as for each $C' \in LOCAL(C_k, p_{k+1})$, there exists $C' \vee \ldots \vee C_k$, while $C_k \vee \ldots \vee C$, then $C' \vee \ldots \vee C$ is satisfied.

It can be concluded that each reasoning result C' computed by the $k+1$ node p_{k+1} in reasoning path satisfies $C' \vee \ldots \vee C$.

Therefore, the soundness of algorithm DL-DeCA can be proved by the above three steps.

Theorem 2: *(Completeness) Given any model I of a DL-P2PRS and a query concept C, if a concept D satisfies $D^I \sqsubseteq C^I$ or $D^I = C^I$, then the result D can be given by DL-DeCA (Algorithm 1-4).*

Proof: The proof can be given by mathematical induction that if the knowledge base does not change when reasoning, as for a *Forth* message $m(User, p, forth, id, \phi, C)$, algorithm DL-DeCA can compute concept *D* satisfying $D \vee \ldots \vee C$.

1 if the number of concepts in transition link is one, i.e. there does not exist sub-concept in concept *C* when querying, and the satisfying concept set is {*C*}. When node *p* receives message $m(User, p, forth, id, \phi, C)$, it should be processed by algorithm 4. It is clear that the reasoning result return to *User* is satisfied.

2 if the number of concepts in transition link is $k\ (k > 1)$, assume that algorithm DL-DeCA can compute correct result, i.e. as for transition link $C_{k-1} \vee C_{k-2} \vee \ldots \vee C_1 \vee C$, algorithm DL-DeCA can return $C \vee C_1 \vee \ldots \vee C_{k-2} \vee C_{k-1}$ to *User* correctly.

3 if the number of concepts in transition link is $k+1 (k > 1)$, the transition link is $C_k \vee C_{k-1} \vee C_{k-2} \vee \ldots \vee C_1 \vee C$.

If C_k and C_{k-1} are the concepts defined in the same peer node, assuming this node is p_x and $C_{k-1} \in Reason(C_i, p_x)$, it is clear that $C_k \in Reason(C_i, p_x)$ by algorithm 4. By line 10 in algorithm 3, as for each result in $LOCAL(C_i, p_x)$, p_x will send back a *Back* message. Because C_{k-1} can return back to *User* correctly, C_k can return back too.

If C_k and C_{k-1} are the concepts defined in different peer node, assuming C_{k-1} is defined in node p_x and C_k is defined in node p_y respectively, there exists mapping $C_k \vee C_{k-1}$ between p_x and p_y. If $C_k \vee C_{k-1}$ is built by p_x, then it should be stored in the knowledge base of p_x. Similarly, C_k can return back to *User* correctly. If $C_k \vee C_{k-1}$ is built by p_y, C_{k-1} is the share variable between p_x and p_y. By line 18-21 in algorithm 3, p_x should send *Forth* message $m(p_x, p_y, forth, id, C_i, C_{k-1})$ to p_y and ask p_y to reason in further. Because $C_k \vee C_{k-1}$ is stored in p_y, by algorithm 4, there exists $C_k \in Reason(C_{k-1}, p_y)$ and $C_k \in LOCAL(C_{k-1}, p_y)$. By line 7 in algorithm 3, it is clear that $C_{k-1} \in LOCAL(C_{k-1}, p_y)$. By line 9-10 in algorithm 3, when p_y processes $m(p_x, p_y, forth, id, C_i, C_{k-1})$, it should send *Back* message to p_x which packs C_{k-1} and C_k together. Because C_{k-1} can return to *User* correctly, C_k can return to *User* correctly too.

It can be concluded that algorithm DL-DeCA can return to *User* $\{C, C_1, \ldots, C_{k-2}, C_{k-1}, C_k\}$ correctly when the number of concepts in transition link is $k+1$.

Therefore, the completeness of algorithm DL-DeCA can be proved by the above three steps.

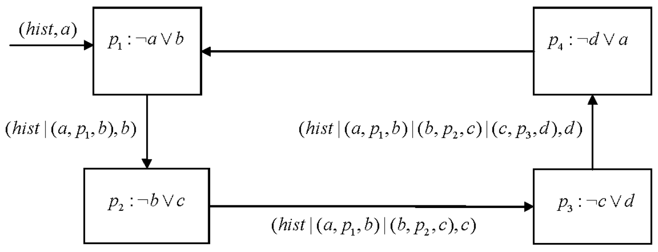
5.4 Compared with DeCA and complexity analysis

Algorithm DeCA (Adjiman et al., 2004, 2005) is based on message delivering and processing, which is composed of three types of message: *ForthMessage* $(Sender, Receiver, forth, hist, literal)$, *BackMessage* $(Sender, Receiver, back, hist, r)$, and *FinalMessage* $(Sender, Receiver, final, hist, true)$, in which *hist* represents the reasoning results of literal q in node p is clause c.

The reasoning results *hist* stored in message can avoid conflict results, prevent from falling into loop reasoning, and help returning message when querying last node. Figure 11 shows 4 peer nodes in P2P system. When node p_1 receives *Forth* message from node p_4 which asks for reasoning on literal a. With the help of *hist*, node p_1 can find that literal a has been reasoned. Hence node p_1 will send *Final* message to node p_4 to terminate reasoning, which can avoid falling into loop reasoning.

However, it is clear that the more depth reasoning, the more length *hist* stored in message, which will increase the burden on the network. Supposing a P2P reasoning system with n peer nodes, the space complexity of *hist* is $O(n)$. Compared with DeCA, by adding some data structure in each peer node and changing the message data structure shown in Section 5.1, DL-DeCA can reduce the *hist* space complexity from $O(n)$ to $O(1)$.

Figure 11 Reasoning in 4 peer nodes with DeCA


$(hist, a)$
$p_1 : \neg a \vee b$
$p_4 : \neg d \vee a$
$(hist \mid (a, p_1, b), b)$
$(hist \mid (a, p_1, b) \mid (b, p_2, c) \mid (c, p_3, d), d)$
$p_2 : \neg b \vee c$
$p_3 : \neg c \vee d$
$(hist \mid (a, p_1, b) \mid (b, p_2, c), c)$

6 DL-P2PRS implementation

The network topology, mapping, concept consistence, and reasoning service, etc. of DL-P2PRS are built upon message-driven sending and processing, the types of which consist of reasoning message and control message.

6.1 Reasoning message

Reasoning message is used to pass reasoning request between peer nodes, return result and notify the initial node that the reasoning is finished, with which one reasoning task can be achieved collaboratively in DL-P2PRS (shown in Table 1).

Table 1 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 reasoning.
Individuals RequestMesg	sender, receiver, individualsRequest, concept	Sender requests receiver return the instance of concept to receiver.
Individuals ReplyMesg	sender, receiver, individualsReply, concept, Set<individual>	Sender returns the instance of concept in knowledge to receiver.

6.2 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.

As far as network topology maintenance is concerned, when other nodes are added or quit network, it should modify the online node list and mapping. When one node is added, other node will be notified and reply an online message. When one node quit, it should notify other to delete the mapping relationship (shown in Table 2).

Table 2 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, keepAlive	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.

If the mapping relationship between nodes has been built, the initial node should store mapping information. Partial variable of other nodes is stored and maintained in local knowledge base. Concept consistence maintenance is used to maintain the ontology of local node, including concept renaming, concept deleting, etc. (Table 3 describes the message in detail).

Table 3 Concept consistency maintenance message

Type	*Message content*	*Description*
Concept RenameMesg	sender, receiver, rename, oldName, newName	Sender notifies receiver that the oldName is renamed as newName.
Concept DeleteMesg	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 other 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 4).

Table 4 Mapping maintenance message

Type	*Message content*	*Description*
OntologyReq uestMesg	sender, receiver, askForOntology	Sender requests Receiver to return back ontology and other information.
OntologyRep lyMesg	sender, receiver, ontology	Sender send ontology and other information to Receiver
MappingReq uestMesg	sender, receiver, mappingRequest, mapping	Sender requests receiver to build mapping in each other.
MappingRep lyMesg	sender, receiver, mappingReply, mapping, reply	Sender and receiver reply the mapping request: reply=0 reject; reply=1 accept.
MappingDel eteMesg	sender, receiver, deleteMapping, mapping	Sender notifies receiver to delete mapping.

6.3 System state transition diagram

Figure 12 shows state transition diagram which is driven by messages in DL-P2PRS when one node is added or quit. The detailed transition processes are described as follows:

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*.

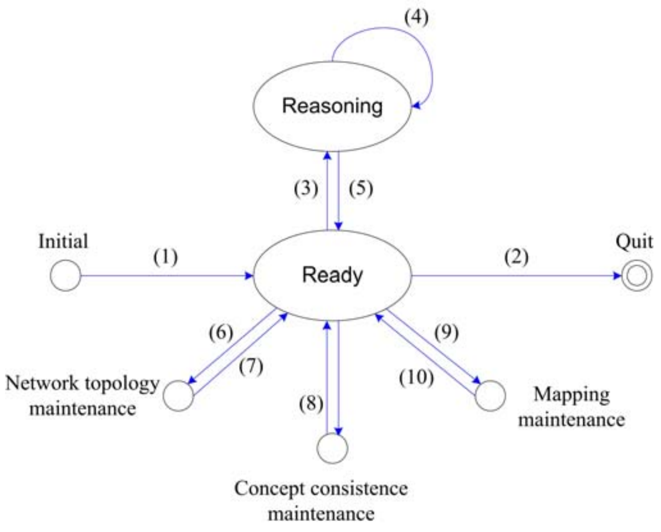
7 send *JoinReplyMesg*. send *AliveReplyMesg*. send *KeepAliveMesg*. finish.

8 receive *ConceptRenameMesg*, receive *ConceptDeleteMesg*.

9 receive *MappingRequestMesg*. receive *MappingDeleteMesg*.

10 send *MappingReplyMesg*. finish *MappingDeleteMesg*.

Figure 12 DL-P2PRS message-driven state transition diagram (see online version for colours)



6.4 System application

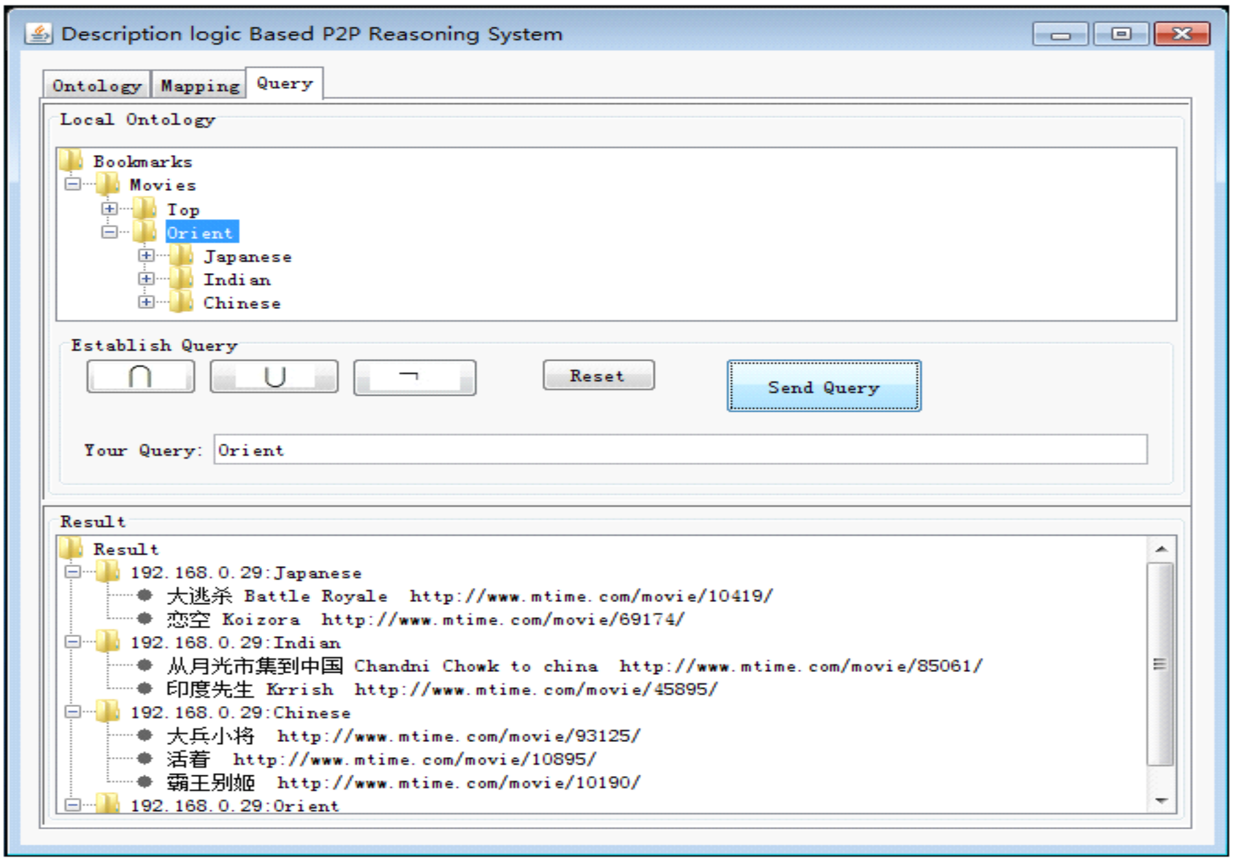
We developed DL-P2PRS with Java, which implements all of the functions described above. In order to simplifying, the collected network resource comprises two attributes: resource name and URL. Knowledge structure is organised as multi-level contents, with the functions of ontology adding, modifying, deleting, etc. (shown in Figure 13).

Mapping maintenance interface (Figure 14) shows how to add or delete local mapping relationship when other peer nodes communicate firstly.

Reasoning interface shows how to construct reasoning items and present reasoning result (shown in Figure 15).

Figure 16 shows that if node Jim receives a query on Favourite, describing in section 4, expected return result is $\{Jim : Favorite, Jim : ActionHumorous, Lucy : Top, Lucy : Orient, Lucy : Chinese, Lucy : Japanese, Lucy : Indian, Andy : A \cap Andy : C\}$.

Figure 13 Ontology maintenance interface (see online version for colours)

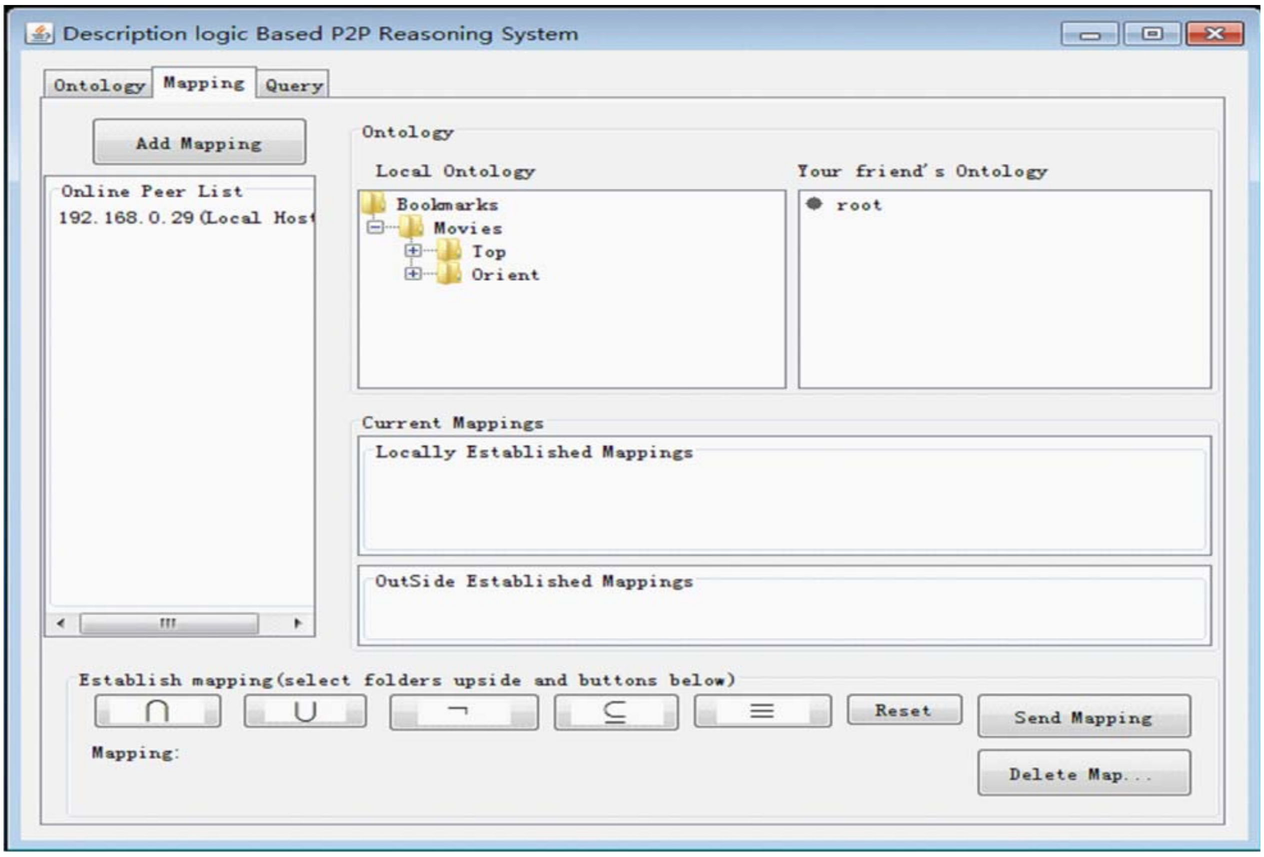


```
Bookmarks
  Movies
    Top
    Orient
      Japanese
      Indian
      Chinese
```

Your Query: Orient

```
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 14 Mapping maintenance interface (see online version for colours)

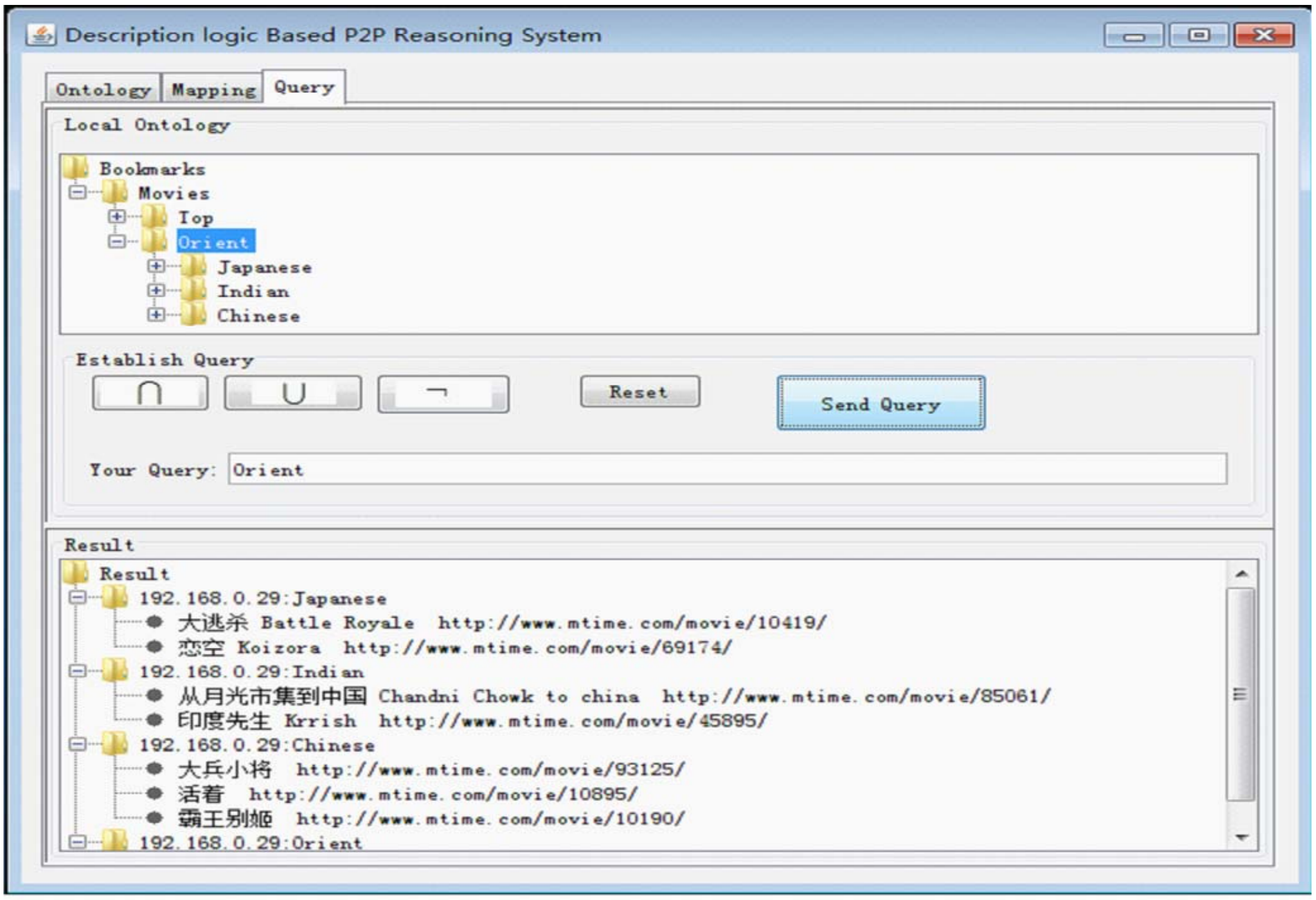

Description logic Based P2P Reasoning System
Ontology
Mapping
Query
Add Mapping
Online Peer List
Ontology
Local Ontology
Your friend's Ontology
Current Mappings
Locally Established Mappings
OutSide Established Mappings
Establish mapping(select folders upside and buttons below)
∩
∪
¬
⊆
≡
Reset
Send Mapping
Delete Map...
Mapping:

192.168.0.29 (Local Host

```
Bookmarks
  Movies
    Top
    Orient
```

● root

Figure 15 Reasoning interface (see online version for colours)



```
Bookmarks
  Movies
    Top
    Orient
      Japanese
      Indian
      Chinese
```

Your Query: Orient

```
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 16 DL-P2PRS example: composed of node Jim, Lucy, and Andy (see online version for colours)

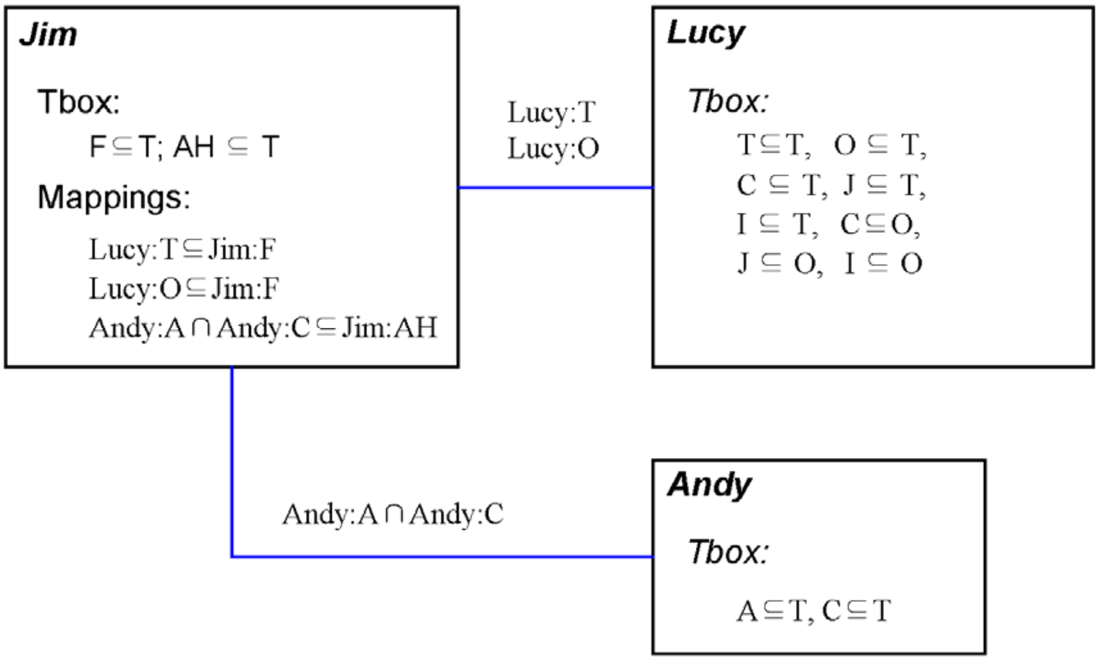
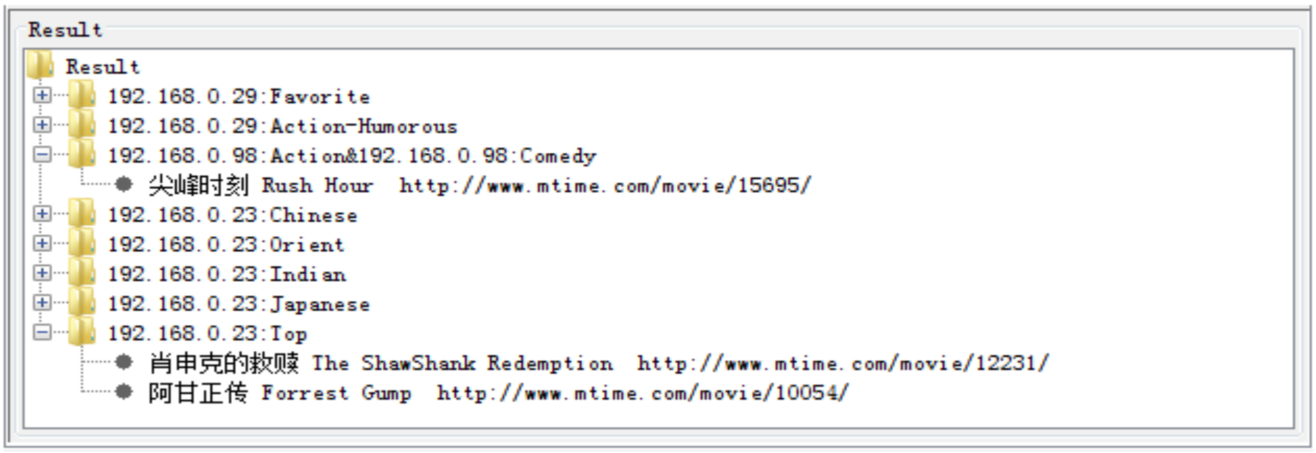


Figure 17 DL-P2PRS example: reasoning result of node Jim (see online version for colours)



```
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/
```

Test the example in our system - DL-P2PRS, 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 17 demonstrates that our system can work correctly and effectively.

7 Conclusion

This paper has given an account of and the reasons for the use of reasoning technique in P2P network. Because many description logic models are decidable fragments of first order logic and of particular importance in providing a logical formalism and reasoning tools for ontology and semantic web. This paper investigated and proposed a P2P reasoning algorithm: DL-DeCA suitable for description logic. 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 neighbour 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.

Acknowledgements

This research is supported by Doctoral Program Foundation of Institutions of Higher Education of China under Grant 20110171120041; as well as 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

Adjiman, P., Chatalic, P., Goasdoue, F., Rousset, M., Simon, L. (2004) 'Distributed reasoning in a peer-to-peer setting', *Proc. of the 16th European Conference on Artificial Intelligence (ECAI 2004)*, Aug.23–27, Valencia, Spain, pp.945–946.

Adjiman, P., Chatalic, P., Goasdoue, F., Rousset, M., Simon, L. (2005) 'Scalability study of peer-to-peer consequence finding', *Proc. of the 19th International Joint Conference on Artificial Intelligence (IJCAI 2005)*, 30 July–5 August, Edinburgh, Scotland, UK, pp.351–356.

Amir, E. and McIlraith, S. (2000) 'Partition-based logical reasoning', *Proc. of the 11th International Conference on Principles of Knowledge Representation and Reasoning (KR 2000)*, Apr.12-15, Breckenridge, Colorado, USA. pp.389–400.

Amir, E. and McIlraith, S. (2005) 'Partition-based logical reasoning for first-order and propositional theories', *Artificial Intelligence*, Vol. 162, No. 2, pp.49–88.

Arenas, M. and Bertossi, L. and Chomicki, J. (1999) 'Consistent query answers in inconsistent databases', *Proc. of the Eighteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS 1999)*, May 31–Jun.2, Philadelphia, Pennsylvaniapp, USA. pp.68–79.

Benferhat, S., Dubois, D. and Prade, H. (1997) 'Some syntactic approaches to the handling of inconsistent knowledge bases: A comparative study part 1 - The flat case', *Studia Logica*, Vol. 58, No. 1, pp.17–45.

Bertossi, L.E. and Chomicki, J. (2003) 'Query answering in inconsistent databases', *Logics for Emerging Applications of Databases*, pp.1–49.

Bertossi, L.E. and Bravo, L. (2004) 'Query answering in peer-to-peer data exchange systems', *Proc. of the 9th International Conference on Extending Database Technology (EDBT) Workshop on Peer-to-Peer Computing and Databases*, Mar.14–18, Heraklion, Crete, Greece, pp.476–485.

Bertossi, L.E. (2006) 'Consistent query answering in databases', *SIGMOD Record*, Vol. 35, No. 1, pp.68–76.

Binas, A. and McIlraith, S.A. (2008) 'Peer-to-peer query answering with inconsistent knowledge', *Proc. of the 11th International Conference on Principles of Knowledge Representation and Reasoning (KR 2008)*, Sep.16–19, Sydney, Australia, pp.329–339.

Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., and Rosati, R. (2008) 'Inconsistency tolerance P2P data integration: an epistemic logic approach', *Information Systems*, Vol. 33, No. 4-5, pp.360–384.

Chatalic, Ph., Nguyen, G. H., and Rousset, M. Ch. (2006) 'Reasoning with inconsistencies in propositional peer-to-peer inference systems', *Proc. of 17th European Conference on Artificial Intelligence (ECAI 2006)*, Aug.29–Sep.1, Riva del Garda, Italy, pp.352–356.

Curtis, C., Matthews, G., and Baxter, D. (2003) 'On the effective use of cyc in a question answering system', *Proc. of the Joint Conference on Artificial Intelligence (IJCAI) Workshop on Knowledge and Reasoning for Answering Questions (KRAQ'05–IJCAI 2005 Workshop)*, Jul. 30, Edinburgh, Scotland, UK. pp.61–70.

Fikes, R. and Farquhar, A. (1999) 'Large-scale repositories of highly expressive reusable knowledge', *IEEE Intelligent Systems*, Vol. 14, No. 2, pp.1-16.

Goasdou, F. and Rousset, M.C. (2003) 'Querying distributed data through distributed ontologies: a simple but scalable approach', *IEEE Intelligent Systems*, Vol. 18, No. 5, pp.60–65.

Horrocks, I. (1998) 'The fact system', *Proc. of the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX'98)*, May 5-8, Oisterwijk, The Netherlands, pp.307–312.

Nardi, D. and Brachman, R.J. (2003) 'An introduction to description logics', *The Description Logic Handbook. Theory, Implementation and Applications*, pp.1–40.

Saha, G.K. (2007) 'Web ontology language (OWL) and semantic web', *Ubiquity*, Vol. 8, Issue 35, pp.1–24.

Yuan, D., Yang, Y., Liu, X. and Chen, J. (2011) 'On demand minimum cost benchmarking for intermediate datasets storage in scientific cloud workflow systems', *Journal of Parallel and Distributed Computing*, Vol. 71, pp.316–332.

Zhang, G., Yang, Y. and Chen, J. (2011) 'A historical probability based noise generation strategy for privacy protection in cloud computing', *Journal of Computer and System Sciences*, Vol. 78, No. 5, pp.1374–1381.