Systemics and Informatics World Network
Volume 13 ▪ December 2012 ▪ pp. 53-71
sai: siwn.0013.2012.065

ISSN 2044-7272 (print)
ISSN 2051-5634 (online)

Semantic Enhanced Argumentation Based Group Decision Making for Complex Problems

Haibo Jia

Chapter 5 A Prototype System Implementation

Abstract: In this chapter, a prototype system using agent based technology to support group argumentation based decision making is implemented. In this system, the group tasks are collaboratively undertaken by different types of agent (facilitator agent, expert agent, knowledge agent and decision making agent). The group utterances are proposed by the expert agent, semantically annotated and analysed by the knowledge agent and finally consumed by decision making agent. At the same time, the facilitator agent controls the whole process and collects the relevant data from other agents; the expert agent dynamically analyse the performance of each expert to update the expert's profile. In order to automatically discover the application agents and their services, the agent management mechanisms are discussed; the behaviour of different agents and the communication protocols in different stages between agents are implemented; a novel semantic annotation approach and algorithm are proposed and some potential semantic services scenarios are also presented. Finally an automatic decision making approach based on semantic enriched argumentation information is developed.

Keyword: agent based prototype system implementation, semantic support argumentation and decision making.

5.1 Overview

In Chapter 4, the agent based architecture for decision-making oriented group argumentation has been presented. In order to achieve the group task by the cooperation of different agents, the application agents and system agents have been clearly identified complying with the FIPA specification. The interactions between the different agents for different scenario are modelled.

In this chapter, a prototype system using agent based technology to support group argumentation based decision making is implemented. In this system, the group tasks are collaboratively undertaken by different types of agent (facilitator agent, expert agent, knowledge agent and decision making agent). The group utterances are proposed by the expert agent, semantically annotated and analysed by the knowledge agent and finally consumed by decision making agent. At the same time, the facilitator agent controls the whole process and collects the relevant data from other agents; the expert agent dynamically analyse the performance of each expert to update the expert's profile. In order to automatically discover the application agents and their services, the agent management mechanisms are discussed; the behaviour of different agents and the communication protocols in different stages between agents are implemented; a novel semantic annotation approach and algorithm are proposed and some potential semantic services scenarios are also presented. Finally an automatic decision making approach based on semantic enriched argumentation information is implemented.

5.2 Agent Based Group Communication

5.2.1 Agent Management Platform

5.2.1.1 The Overview of Agent Management Platform

In our prototype system, the facilitator agent initiates a group discussion by assigning a task to the group members. The expert agent representing the group member can join the system at the any time during the group discussion as the discussion client start up. Once a new expert agent joins in, the facilitator agent needs to be informed and sends the assigned task and current discussion utterances to the new comer. One expert agent also needs to be aware the presence of all other expert agents so that it can broadcast its utterance to all the group members. In addition, the knowledge agent and decision making agent as the service agent should be discovered by other agents to request services from them. In order to fulfil the above requirement, the agent management platform should maintain the lifecycle of the agents, provide the mechanisms to describe the agent and the service provided by agent, and also should have two different functional mechanisms for the agents to be aware the presence of other agents: agent discovery and agent subscription.

By agent discovery, the agent can discover other agent by the service description; however by agent subscription, an agent can perceive the agent-born/agent-dead event of its subscribed agent from the agent platform. The mechanism of the agent discovery can assure the new joined agent have the capability to know the existing surrounding environment, but cannot assure that update of the surrounding environment is detected simultaneously by one agent. In this case, the mechanism of agent subscription could complement this gap. The combination of these two mechanisms can guarantee one agent could know all the surrounding agents with some specified services at any time. So it is crucial for agent management platform to implement these two mechanisms.

In FIPA's Agent Management specification, normative frameworks are proposed within which FIPA agents exist and operate. It establishes the logical reference model for the creation, registration, location, communication, migration and retirement of agents. (Belifemine et al, 2006) Referring to this model, the schema of agent management can be illustrated as Figure 5.1.

Agent Platform (AP): It provides the physical platform in which agents are deployed. It consists of the machines, operating systems, FIPA agent management components (DF, AMS), the functional agents and any additional support software.

Agent: An agent is a computational process that inhabits an AP and offers some computational services that can be published as a service description, which is often used to discover one agent by others. In our system, the agents are classified as facilitator agent, expert agent, knowledge agent and decision making agent.

Directory Facilitator: (DF) The DF provides yellow pages services to other agents. Every agent that wishes to publish its services to other agents should find an appropriate DF and request registration with its agent description. In addition, an agent may request a search command to a DF to discover other agent with matched criteria.

Agent Management System (AMS): The AMS is responsible for managing the operation of an Agent platform, such as the creation and deletion of agents. Each agent must register with an AMS in order to obtain agent identification (AID). The AMS maintains AID as a directory of all agents within the AP. The life of an agent within an AP terminates with its deregistration from the AMS. In our system, functional agents can subscribe other agents' registration/deregistration event to know their state. Particularly it is useful for a facilitator agent and expert agent to monitor the presence/leaving of other expert agents so that they can more efficiently send the utterance message.

5.2.1.2 JADE Based Implementation For Agent Management Platform

JADE is a java implemented software platform fully compliant with FIPA specification. It provides simple and friendly API to implement software agent abstraction. The main container of JADE includes two special agents which respectively are DF agent and AMS agent. Both of them are in

accordance with the FIPA Agent Management model shown in Figure 5.1. Among them DF agent provides the yellow page service allowing agents to publish descriptions of one or more services in order that other agents can easily discover and exploit them. Unlike DF agent, AMS agent provides white pages service as specified by FIPA to support a subscription mechanism by means of which interested agents can be notified about platform events such as new agent creation, old agent terminations and so on. Besides that, the AMS agent is the only agent able to perform platform management operation such as creating and killing agent etc.

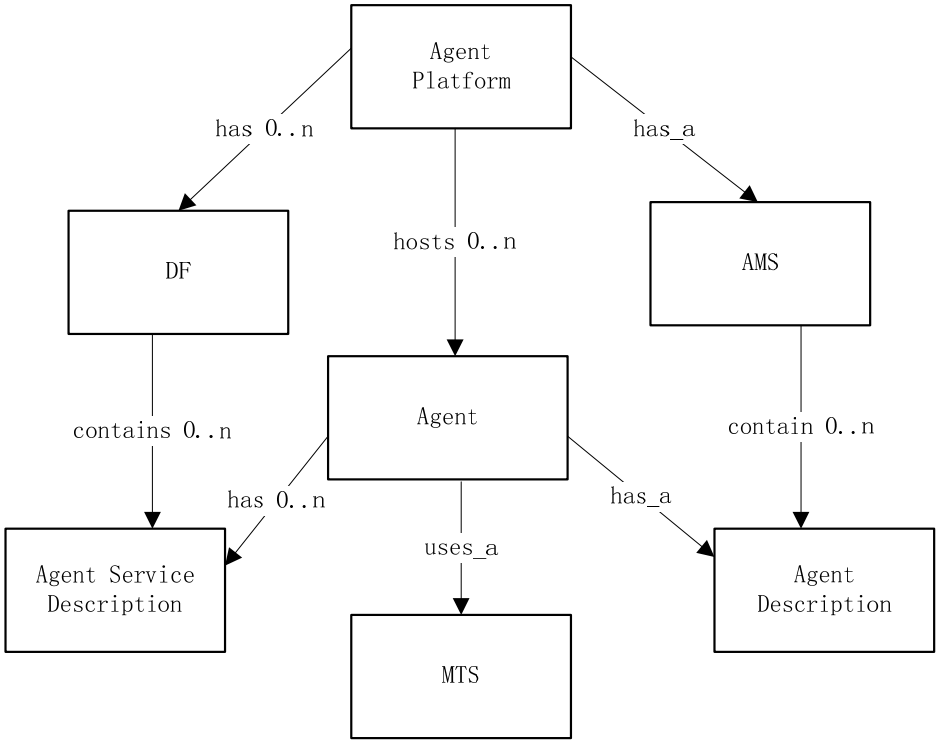

Figure 5.1 Agent management schema

(1) Agent Publishing and Discovery

In order to simplify the interaction between DF agent and other agents, JADE provides the *jade.domain.DFService* class with which it is possible to publish and search for services through a variety of method calls. An agent must provide its own Identification (AID) and a list of provided services for publishing its services. Each published service description must include the service name, the service type and a collection of service-specific properties. The *DFAgentDescrption*, *ServiceDescription* classes represent these abstractions. A java code snippet in Figure 5.2 shows the process of the facilitator agent publishing itself.

An agent wishing to discover other agents for services must provide the DF with a template description. The result of the search is a list of agent descriptions that match the provided template. In Figure 5.3, a java code snippet shows how one expert agent searches the published facilitator agent from the DF agent. Each agent constructs an acquaintance table and maintains discovered agents in this table.

(2) Agent Subscription and Notification

In Jade, all platform events are described by the concepts of *JADE-Introspection* ontology which is

```
DFAgentDescription dfd =new DFAgentDescription();
dfd.setName(getAID());
ServiceDescription sd=new ServiceDescription();
sd.setType("Facilitator");
sd.setName(getLocalName()+"-Facilitator");
dfd.addServices(sd);
DFService.register(this,dfd);
```

Figure 5.2 Publish an agent code snippet

```
ArrayList <AID> fagent=new Arraylist();
 DFAgentDescription template=new DFAgentDescription();
  ServiceDescription sd = new ServiceDescription();
  sd.setType("Facilitator");
  template.addServices(sd);
  DFAgentDescription[] result=
  DFService.search(PAgent.this, template);
  for (int i=0; i<result.length; ++i)
    {
     fagent.add(result[i].getName());
    }
```

Figure 5.3 Search published agent code snippet

```
HashSet pagents= new HashSet(); // acquaintance table
AMSSubscriber myAMSSubscriber = new AMSSubscriber(){
   protected void installHandlers(Map handlers) {
EventHandler creationsHandler = new EventHandler()
   {
            public void handle(Event ev) {
            BornAgent ba=(BornAgent) ev; // return BornAgent event
         AID aid=ba.getAgent();
        if (ba.getClassName().equals("my.research.PAgent"))
        {
          pagents.add(aid);
        }
      }  };
handlers.put(IntrospectionVocabulary.BORNAGENT,creationsHandler);
  }  };
addBehaviour(myAMSSubscriber);
```

Figure 5.4 Subscribe platform event code snippet

implemented by the *IntrospectionOntology* class. These concepts have corresponding event class implementation including *BornAgent* class, *DeadAgent* class, *RemovedContainer* class etc. JADE platform dispatch all the platform events to the AMS. In order to dynamically perceive the other agents' status in the platform, one agent needs to subscribe to the AMS in order to receive notifications about platform events. Similar to DF agent, JADE implements AMS as a special agent and also provides *jade.domain.introspection.AMSSubscriber* class to model FIPA-Subscribe protocol. By which, the agent can interact with AMS to subscribe its interested platform event. The snippet

code within facilitator agent shown in Figure 5.4 exemplifies how the facilitator agent subscribes the born-agent event so that it can be notified when a new expert agent joins in the platform.

5.2.2 Agent Communication and Behaviour

In the multi-agent system, agents communicate with each other by sending messages. Speech acts designate all intentional actions carried out in the course of communication. Based on it, FIPA define a list of communicative acts such as Call for Proposal, Subscribe, Propose, Request etc. In our system, at the different stage of group argumentation most agents only take a single action when they receive the message. For example, in the stage of registration once an expert agent receives the task and the current utterance list it will start to construct its utterance tree; in the stage of group discussion once an expert agent receives the message it will update its utterance tree and profile; in the stage of criteria generation once an expert agent receives the message it will send back weighted criteria and its credibility profile; in the stage of decision making, once decision making agent receives the message it will start to compute the score of solutions and update the state of utterance. For simplicity, in our system we only use 'inform' as communicative act and record the name of stage in the communication message so that in the different stage the agent can automatically select the proper behaviour. A code snippet in Figure 5.5 exemplifies how the expert agent behaves to response others' request in the different stage. However in the stage of group discussion the knowledge agent not only can provide annotation service but also can provide other semantic match services to query related utterances or argumentation pairs. For this case, we build a message content analysis function within the knowledge agent. For example, if the knowledge agent receives a non- annotated utterance it will enter the annotation process, otherwise it will go for semantic query process.

```
CyclicBehaviour  behaviour=new CyclicBehaviour() {
private  MessageTemplate  mt=
MessageTemplate.MatchPerformative(ACLMessage.inform);
 public void action() {
   ACLMessage msgRx=receive(mt);
   if (! msgRx ) block();
   else {
       if ( msgRx.getConversationId().equals("register")){
           myGui.contruct_tree1(msgRx.getContent());}
       else if (msgRx.getConversationId().equals("discussion")){
           //other participant send a utteraqnce to me
            Utterance ut=(Utterance) msgRx.getContentObject();
           myGui.updateModel (ut);
               myGui.updateTree(ut);
          //the coming utterance argue my utterance, update my profile
          if ((ut.getPrecedent().getCreator()).equals(getAID().getLocalName()))
               updateProfile(ut);
            }
     else if (msgRx.getConversationId(). equals("voting")) {
          HashSet<Creteria> creterias=(HashSet) msgRx.getContentObject();
           myGui.updatevoting(creterias); }
        }
    }}
addBehaviour (behaviour);
```

Figure 5.5 Expert agent's behaviour code snippet

The JADE platform provides a mechanism to allow an agent to use the java object as the message content and translate java object to a proper content language such as FIPA-SL (semantic language) and vice versa. It could simplify the process of development. The message content objects used in our system are shown in Figure 5.6. In each interaction, the agent will select one of these objects as communication content. For example, when the expert agent request utterance annotation, it will send non-annotated utterance to knowledge agent and knowledge agent will return an annotated utterance afterwards. In the stage of criteria generation, the expert agent will send the weighted criteria object and credibility profile object to the facilitator agent. In the stage of decision making, the decision making agent will update the score and state of all utterances and send a list of utterance objects to the facilitator agent.

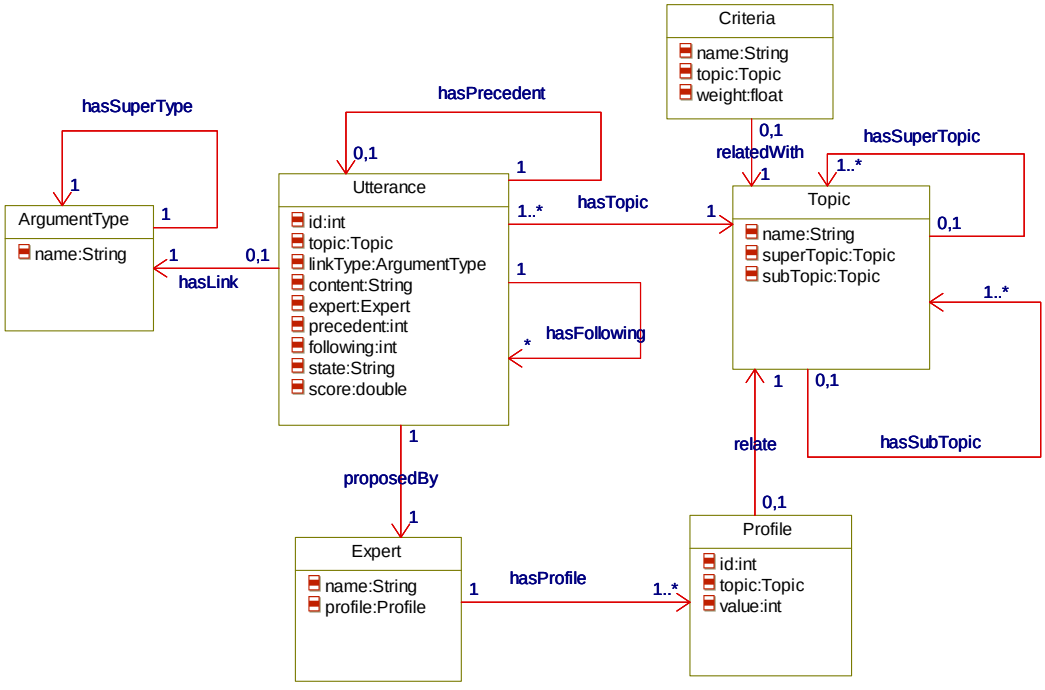

Figure 5.6 Communication message content object diagram

5.3 Semantic Support Argumentation system

5.3.1 Argumentative Information Representation

In Chapter 4, a group argumentation based decision making data model (Refer to Figure 4.2) and argumentation ontology (Refer to Section 4.3.2) are proposed. During the group argumentation, the agents populate the ontology using their own utterance to represent argumentative information. So the ontological category of utterance, the semantic link between utterances and the topic of the utterance must be explicitly identified. Among them, the expert agent can derive semantic link and utterance category from the user interface; and the topic of the utterance needs to be derived by the knowledge agent from external ontology resource. The semantic of topic are attached to the utterance by the semantic annotation, the process of which called as semantic annotation will be discussed in the following section (Section 5.3.2.1).

In our prototype system, the Jena semantic web framework[1] has been adopted as the backbone to represent and manipulate the argumentative information inside the agents. Jena provides a programmatic environment for RDF, RDFS and OWL, SPARQL. The agent employs Jena API to construct the semantic model based on the predefined argumentation ontology and populate the instance into the model when it receives the utterance message. Agent also can back up the semantic model into the persistent repository such as file or database, which could be reused by other discussion or decision making session.

(1) Create ontology model. Ontology is a formal specification of concepts and relation between concepts. In the prototype system, ontology model consists of class, property, the relations among classes and restriction. Jena framework provides API to construct ontology model (Figure 5.7), which can be used as data schema of argumentation utterance. This model will be populated by the agent using real argumentation information during the group argumentation.

```
OntModel create_model (String namespace)
{
OntModel m =
ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM_RULE_INF, null );
m.setNsPrefix("", namespace);
// construct property
OntProperty p0=m.createObjectProperty(namespace+"#semantic_link");
OntProperty p1=m.createObjectProperty(namespace+" #raise_issue");
……..
//construct class
OntClass c1=m.createClass(namespace+"#Task");
OntClass c4=m.createClass(namespace+"#Argumentation");
OntClass c5=m.createClass(namespace+"#Practical_Argumentation");
OntClass c6=m.createClass(namespace+"#Epistmic_Argumentation");
OntClass c6=m.createClass(namespace+"#Topic”);
…..
// construct relationship
p1.setSuperProperty(p0);
c5.setSuperClass(c4);
c6.setSuperClass(c4);
……..
return m;
}
```

Figure 5.7 Construct argumentation ontology model code snippet

(2) Populate ontology by instance. In order to semantically represent group argumentation information, each utterance instance needs to be instantiated by the form of argumentation ontology. Here we design an ontology population method using Jena API to convert the utterance java object to an instance of a semantic model class (Figure 5.8). Once an agent receives or publishes an utterance, it will be converted to the instance of a semantic model and triplet statement which can be used to populate the ontology.

(3) Query an instance. Based on the semantic model and the populated instances, the expert agent is equipped with a functionality to semantically query the utterance instance collection. It is possible for the expert agent answer its user complex questions like “what other utterances talk about a similar topic?” “What issues about a particular aspect (topic)

[1] http://jena.sourceforge.net/

```
void populate_model (Utterance utt, String namespace, OntModel m)
  {
     ArrayList<Statement> st1=new ArrayList<Statement>();
    // retrieve property from utterance java object
    String id=utt.getId();
    Category cat=utt.getCategory();
    int precedentid=utt.getPrecedentid ();
    ArgumentType link=utt.getLink();
    Topic tp=utt.getTopic();
     ……..
   // construct instance
    OntClass oc= m.getResource(namespace+cat.name).getClass();
    OntResource os=m.createIndividual(namespace+id,oc);
    OntResource  precedent=m.getIndividual(namespace+precedentid);
    ObjectProperty op3=m.getObjectProperty(namespace+link.name);
 ……..
   // construct triplet statement
   st1.add(m.createStatement(os, m.getObjectProperty(namespace+"#has_id"), id));
   st1.add(m.createStatement(os, op3, precedent));
   st1.add(m.createStatement(os, m.getObjectProperty(namespace+"#has_topic"), tp));
   …….
    m.add(st1);
  }
```

Figure 5.8 Populate argumentation ontology model code snippet

```
 ArrayList queryIssue(OntModel m, Topic topic1)
{
  String querystring="select  ?z "
                      +"where {?x :has_topic ?y."
                      +"?x :has_content ?z."
                      +"?x rdf:type " + "argumentation:Issue"
                      +"?y rdf:type " +"topic:"+topic1.name+ "}";
  QueryExecution qexec = QueryExecutionFactory.create(querystring, m);
  ResultSet results=null;
  ArrayList al=new ArrayList(); // a collection for holding issue list
  results = qexec.execSelect() ;
  for ( ; results.hasNext() ; )
          {
              QuerySolution soln = results.nextSolution() ;
               Literal r=soln.getLiteral("z");
             al.add(r.getLexicalForm());
                   }
        return al; }
```

Figure 5.9 Semantic query code snippet

are raised?" "What solution utterances address the issue about a particular topic?" or "From which aspects (topics) argumentation utterances challenge/support the solution?" In addition, the expert agent can derive its user's interest by analysing his utterance and proactively remind its user to pay attention to some incoming utterances from others such

as the utterance solving his raised issue, the utterance being contra-argument against his argumentation etc. These kinds of service can facilitate users to better focus on particular aspects of the problem based on their interest or expertise during the group argumentation. In the prototype implementation, we use Jena framework API and SPARQL engine to construct query around the semantic argumentation model. In Figure 5.9, a code snippet demonstrates how to query all the issue utterances about a particular topic.

(4) Backup the model. The semantic argumentation model can be backed up to the persistent repository by the form of OWL – a standard ontology language. It can be imported into any OWL compliant editors such as Protégé and visualized by these tools. And also, the argumentation repository can be queried by the agent in other topic related group argumentation to reuse previous decision making rationale in the similar context. In Figure 5.10, a code snippet shows how to backup a semantic model in to the file system.

```
void backup(OntModel m)
{
  FileOutputStream   is=new FileOutputStream(new File("c:\\test\\Congestion.owl"));
    RDFWriter writer=m.getWriter();
    writer.setProperty("showXmlDeclaration", true);
    writer.write(m, is, null);
 }
```

Figure 5.10 Backup semantic model code snippet

5.3.2 Argumentation User Interface

Argumentation support system should have an interface for the individual users to structure their own utterances and visualize others' utterance in the group. Based on the requirement of the argumentative semantic model described above, the category of utterance, semantic link between utterances and the topic of the utterance needs to be explicitly identified.

The Argumentation user interface of the prototype system is illustrated in Figure 5.11. The discussion tree is visualized in the left panel of the window; the root of tree represents the decision task and the nodes of tree represent different type of utterance. The user can expand or collapse the node to navigate the information. The right panel is claim making area, via which the user can make a claim targeting a selected utterance. The utterance the user makes will be passed to expert agent, who is responsible to populate the utterance to the semantic model and broadcast it to the other expert agent; and also the utterance received by the expert agent will be used to update the discussion tree. Due to the underlying semantic model, the expert agent can derive the utterances related with the users' interest and answer the users' question by querying previous argumentation repository.

5.3.3 Semantic Support

5.3.3.1 Ontology Based Semantic Annotation

Semantic annotation is an approach to annotate the entities using the formal concepts with well-defined meaning and relationship. The entities could be anything including the document, piece of information, physical stuff or even abstract concept. Comparing with the entity in the original state, the semantic annotated entity can be better for classification, retrieval and even some intelligent consumption. In our research, the entities to be annotated are the utterances generated in the group argumentation. In the context of group argumentation based decision making, the experts' utterances reflect the different aspects of decision task and their rationale of problem solving. The semantic annotation could facilitate agents better to interpret and analyse those utterances so that providing more intelligent service for the decision making.

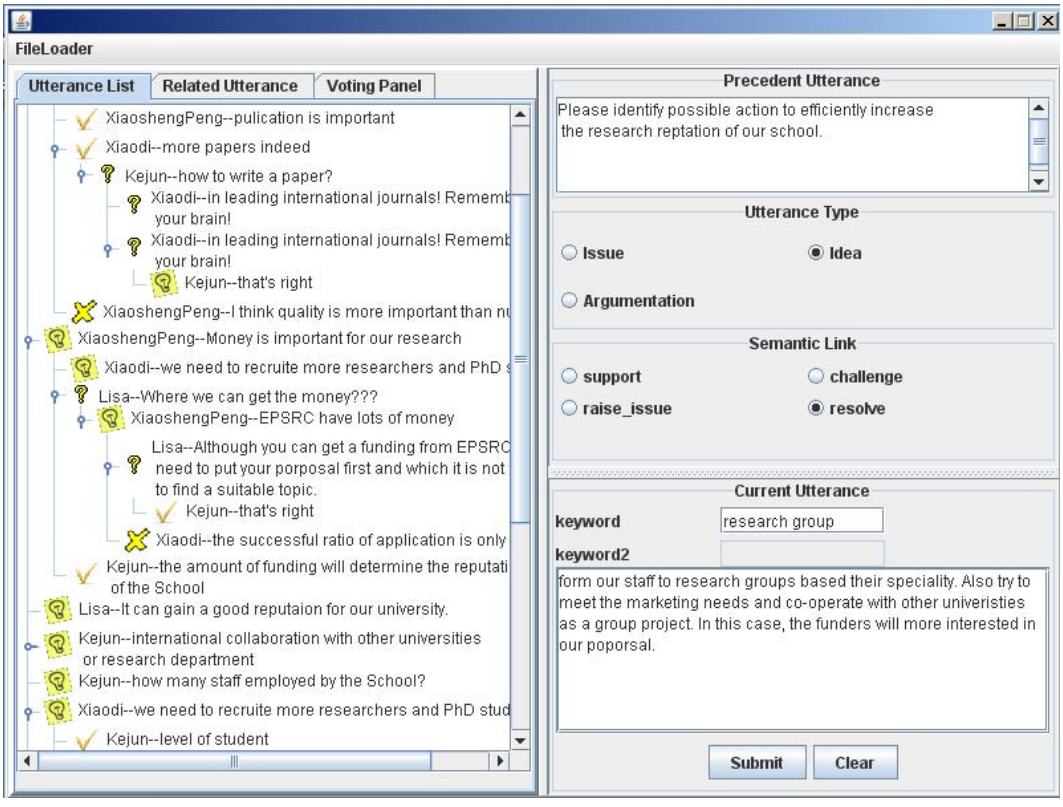

Figure 5.11 Argumentation user interface

In the annotation practice, an external well represented knowledge base, such as Wordnet, DBpedia or domain specific ontology, is often used as the formal concept bases. (Yang et al, 2010; Samwald et al, 2009; Niaraki et al) In essence, the process of annotation is to find an appropriate concept terms in the knowledge base to represent the addressed information. And also the semantic relationship between related concepts can be discovered.

In the group argumentation, due to divergent background and expertise of the expert group the decision task could be discussed from different aspects based on the experts' interest. So it is necessary to opt for cross-domain ontologies to be annotation knowledge base. DBpedia is one of the largest multi-domain ontologies that currently exist in the world, which is built by a community effort to extract structured information from Wikipedia[2] and to make this information open available on the Web. Compared with other ontologies, which usually only cover specific domains, are created by relatively small groups of knowledge engineers, and are very cost intensive to keep up-to-date as domains change, the DBpedia has a number of advantages as follows:

(i) It covers many domains and contains lots of instances;
(ii) It represents real community agreement;
(iii) It (automatically) evolves as Wikipedia changes.

Besides that, the dataset of the DBpedia is encoded as OWL format and a public available SPARQL endpoint can be online accessed. So it is quite straightforward to integrate our framework with DBpedia dataset and easy for the agent to query the dataset of DBpedia and annotate its utterance. Based on the above characteristic of DBpedia, we opt for it as the ontology knowledge base for the semantic annotation purpose.

[2] www.wikipedia.org

5.3.3.2 The Implementation of Semantic Annotation Using DBpedia

The DBpedia knowledge base currently describes more than 3.4 million things, most of which are classified in a consistent ontology. The things in the DBpedia correspond to the articles in Wikipedia. The abstract, title and category of the article are structurally represented as the property of the things. Based on the different categories like persons, places, music albums etc., there are more semantic properties related with article being represented, such as birthdate, birthplace, placeof, and producer. In our application, the purpose of the semantic annotation is to simply identify the formally defined categories of the individual utterances and potential subsumption relationship among these categories. So in our case, the semantic properties of the concept are not considered. The ontology snippet of the DBpedia knowledge base in our research can be shown as Figure 5.12. The resource in the ontology represents the concept within the DBpedia, which has abstract and label property with a simple literal data type. Each concept has subject defined as the name of the category shown in Wikipedia, which respectively links to a semantic broader and narrower category by the property 'broader' or 'broader of'. In addition, the property 'is redirect of' can link a concept to a list of concepts with similar semantics. In Figure 5.13, a dataset example regarding the concept 'Vehicle' in the DBpedia is shown in the triplets. The DBpedia dataset will be used as the knowledge resource in our framework to annotate the new proposed utterance during the group argumentation.

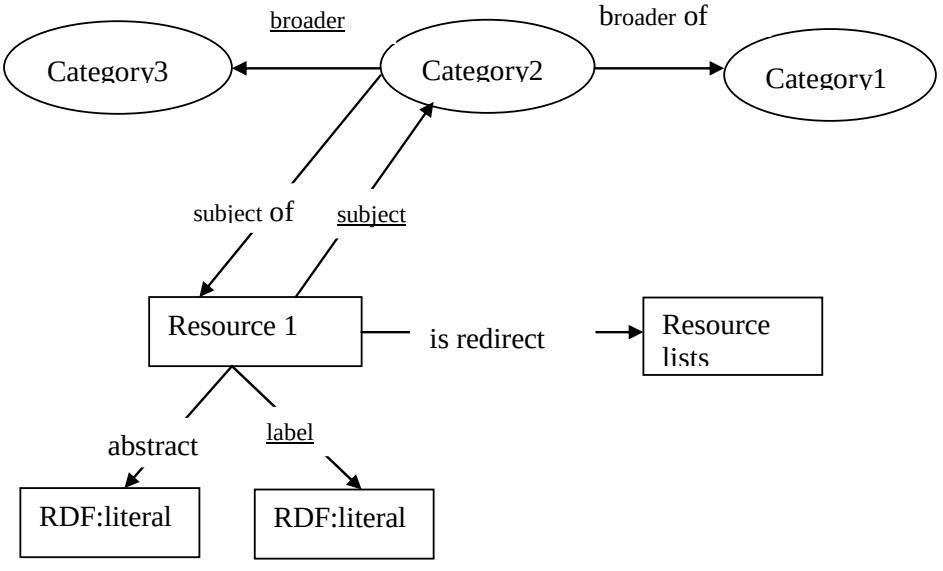

Figure 5.12 The ontology snippet of DBpedia

Based on the above ontology structure, the process of semantic annotation to the utterance can be described as the following steps:

Step 1. Get the concept by matching the keyword of the utterance with the label of the resource in the DBpedia;
Step 2. If the matched concept is not found, go to Step 6. Otherwise, get a category list by searching the value of 'subject' property of this concept in the DBpedia;
Step 3. Select the best matched category from the category list manually, and insert the returning category to the argumentation ontology as the topic of this utterance;
Step 4. Get super-category and sub-category of the identified category by searching its 'broader' property and 'broader of' property in the DBpedia;

Step 5. The results from Step 4 are used to search the topic list in the current argumentation ontology. If find matched topic, then construct the 'subclass' or 'subclassof' relation between the found topic with new inserted topic in Step 3, Go to Step 7.

Step 6. Search all the topics in the argumentation ontology with the keyword of the utterance. If found, use the found topic as the topic of the utterance; if not found, construct a new topic by the name of the keyword in the argumentation ontology and annotate the utterance with this new topic;

Step 7. Process terminates.

```
dbpedia:Vehicle    rdfs:label    "Vehicle"
dbpedia:Vehicle  dbpedia-owl:abstract  "A vehicle is a mechanical means of ……"
dbpedia:Vehicle    is dbpprop:redirect of    dbpedia:Automotives
dbpedia:Vehicle    is dbpprop:redirect of    dbpedia:Transportation_device
dbpedia:Vehicle    is dbpprop:redirect of    dbpedia:Road_vehicle
dbpedia:Vehicle    is dbpprop:redirect of    dbpedia:Land_vehicle
…….
dbpedia:Vehicle    skos:subject   category:Vehicles
dbpedia:Vehicle    skos:subject   category:Transportation
…….
category:Vehicles   skos:broader   category:Machines
category:Vehicles   skos:broader   category:Trasnportation
category:Vehicles   skos:broader   category:Transportation_systems
…….
category:Vehicles   skos:broader of    category:Human-powered_vehicles
category:Vehicles   skos:broader of    category:Tracked_vehicles
category:Vehicles   skos:broader of    category:Vehicle_design
category:Vehicles   skos:broader of    category:Vehicle_rental
……..
category:Transportation   skos:broader   category:Travel
category:Transportation   skos:broader   category:Tourism
category:Transportation   skos:broader   category:Industries
…….
category:Transportation   skos:broader of   category:Transport_associations
category:Transportation   skos:broader of   category:Transport_authorities
category:Transportation   skos:broader of   category:Shipping
category:Transportation   skos:broader of   category:Commuting
```

Figure 5.13 A dataset example of DBpedia

The user interface of DBpedia based semantic annotation can be illustrated using a simple example as Figure 5.14. In which, "advertising" as user input keyword is used to search topical definition in the DBpedia. As result, 4 semantic related topical terms are listed in a data panel which can be selected by user. In this example, "promotion and marketing communication" is opted to be the topical term to replace the original keyword. This term also will be used to define a class in the domain ontology, the relation and sub- or super- class around this term in DBpedia is also utilized to enrich current ontology. So by the semantic annotation, the user defined terms are conceptualized by a formal defined topical term in DBpedia. Which will facilitate system to categorize utterance, evaluate utterance and analyse utterance to provide semantic level decision support.

The process of semantic annotation of the utterance in the system can be illustrated as Figure 5.14. All the queries used to process the DBpedia ontology and the argumentation ontology is

performed using the SPARQL. The DBpedia provide a public SPARQL endpoint [3] over the DBpedia data set, which can be online accessed by our application system. To process argumentation ontology, the system uses SPARQL API provided by the Jena 2.3 toolset[4]. Using the sample SPARQL query statements illustrated in Figure 5.15, the related topic and relation between the topics in DBpedia can be exploited to annotate the utterance.

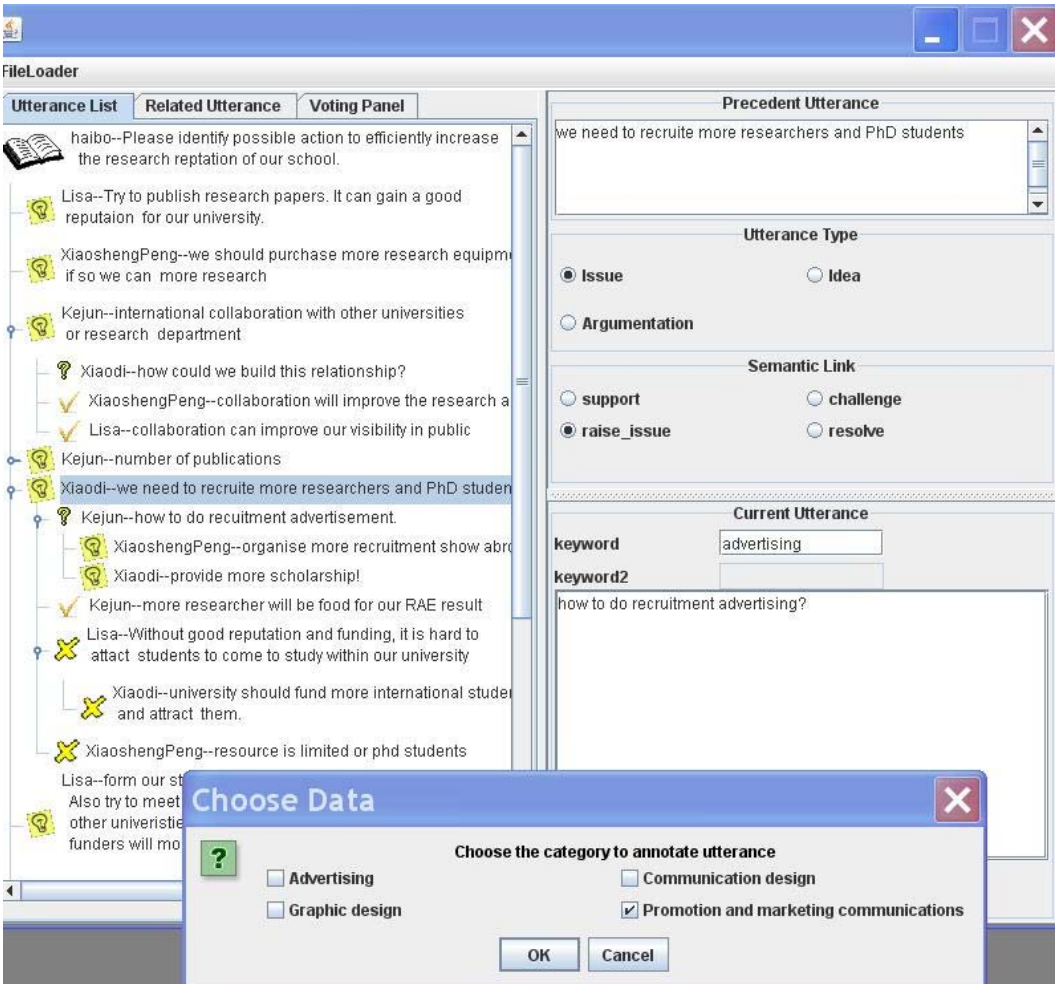

Figure 5.14 Dbpedia based semantic annotation example

5.3.3.3 Potential Semantic Service Scenario

In our research, argumentation ontology formulates the concept of argumentative elements and the relations between them and semantic annotation provides formal and enriched category information to the content of argumentation. In (Passant et al, 2009), similar approach has been proposed to align scientific discourse representation and social semantics using two web ontology namely SIOC (Semantically-Interlinked Online Communities) and SWAN (Semantic Web Application in Neuromedicine). Authors claim that this alignment enables the discourse structure and content relationships to be much more accessible to computation so that information can be navigated,

[3] http://DBpedia.org/sparql
[4] http://jena.sourceforge.net

compared, understood in a more semantic context. Differently, decision making support is one of the major goals in our research. So, some decision making related concepts or properties are added in the argumentation ontology such as the proposer's expertise, the evaluation criteria, the state of utterance and the score of the solution. In addition because our proposed group decision making framework doesn't depend on any particular domain, our argumentation structure is more general and simpler than discourse structure defined in SWAN. Although the implemented prototype system is not implemented in the context of web, it could be easily extended to the web context for a large group or community level decision making support. The potential semantic services based on underlying semantic representation of information can be summarized as the follows:

(i) Discovering topic related utterances (content level) or structure similar discourses (discussion level) from current discussion;
(ii) Matching the topic of the utterance with the expertise of expert to find suitable experts to carry on the argumentation.
(iii) Enabling the agent to carry out automatic decision making based on the semantic of the type of discourse and the link between them.
(iv) Publishing highly structured group argumentation information for share and reuse in the similar decision task. This kind of reuse is not restricted in the information level but in the intensive knowledge level such as rationale reuse level.
(v) Visualizing decision making result from different point of view (positive, negative or uncertain).

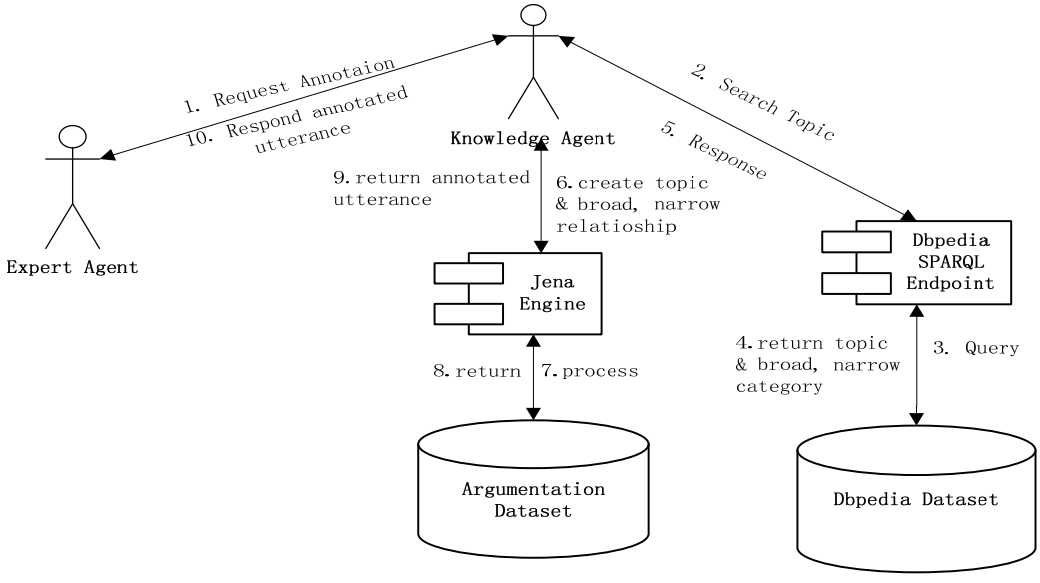

Figure 5.15 Illustration of the utterance semantic annotation

5.4 Decision Making System

The general purpose of decision making is to order all alternatives in order to get the optimization. Argumentation based decision making offers a possible way to automatically evaluate proposed solutions in the group discussion by analysing and comparing the related cons/pros arguments. In addition, the semantic model of argumentative information can better present decisions with explanations, further decomposition and more considerations. So the decision space can be visualized from different aspects.

In Chapter 3, an argumentation based decision function model has been proposed. In this section, the implementation of decision function in the prototype system will be discussed in detail.

5.4.1 The Generation of Evaluation Criteria

In the designed decision function, decision is made as ordering the solution utterances via evaluating the strength of practical argument. The criteria modelled as preference, which reflect the importance of the aspect related with the argument toward achieving task, need to be identified during the decision making.

In the prototype system, each practical argument has been annotated by a formal concept term which implicitly represents the different topic or aspect of the decision task, so that a list of these concept terms could be the possible criteria candidates. In the context of group decision making, the preference of these criteria should be the aggregation of the group member's preference. So during the criteria generation stage, the facilitator agent needs to retrieve all the possible criteria candidates from global utterance collection and inform the group members to select and vote these criteria based on their preference. And the expert agent will receive these criteria lists and present them to the users. All the criteria with voting score from the expert agent can be returned to the facilitator agent who will aggregate them to generate the evaluation criteria. The snippet code illustrated as Figure 5.16 demos how the facilitator agent derives the criteria list from the semantic argumentation model populated during the group discussion. The criteria list is sent to expert agent and presented to the

Search all the possible categories in the DBpedia to which the utterance belongs
SELECT ?o where
{?r rdfs:label keyword@en . ?r skos:subject ?o}
union
{?r rdfs:label keyword@en . ?r dbpedia2:disambiguates ?d .
?d skos:subject ?o}

Search the broader category in the DBpedia
SELECT ?broad_category where
{ category1 skos:broader ?broad_category}

Search the narrower category in the DBpedia
SELECT ?narrow_cateory where
{ category1 skos:broader of ?narrow_category}

Search topic list in the utterance ontology
SELECT ?topics where
{?utterance has_Topic topic1. topic1 rdf:type ?topics}

Construct new topic and annotate the utterance with the topic
CONSTRUCT {topic1 rdf:type category1 }
CONSTRUCT {?u hasTopic topic1}
WHERE {?u hasKeyword keyword1}

Construct subclass relation between new insert topic and existing topic (narrow)
CONSTRUCT {?x rdfs:subClassOf ?y }
WHERE { { narrow_category rdf:type ?x }
Union {new_topic rdf:type ?y} }

Construct subclassof relation between new insert topic and existing topic (broad)
CONSTRUCT {?y rdfs:subClassOf ?x }
WHERE { { broad_category rdf:type ?x }
Union {new_topic rdf:type ?y} }

Figure 5.16 SPARQL queries formulated for semantic annotation

user in the user interface shown as Figure 5.17. So the group members can weigh the criteria according to the importance with scale 1 to 5 and send back to the facilitator agent to generate the aggregated evaluation criteria for decision making. This approach mainly focus on analysing related argument utterances to derive the criteria rather than predefining criteria beforehand, which is normally employed in the conventional decision making. Comparably, our approach is more suitable for the complex problem, the situation of which is not well known before group wide discussion.

```
ArrayList queryTopic(OntModel m)
{
  String querystring="select  ?y "
                      +"where {?x :has_topic ?y. ?x rdf:type ?z.
                     +FILTER (?z = :Practical Argument) }";
 QueryExecution qexec = QueryExecutionFactory.create(querystring, m);
  ResultSet results=null;
  HashSet al=new HashSet();
  results = qexec.execSelect() ;
   for ( ; results.hasNext() ; )
     {
        QuerySolution soln = results.nextSolution() ;
          Literal r=soln.getLiteral("y");
           al.add(r.getLexicalForm());
      }
       return al;
   }
```

Figure 5.17 Derive criteria list by Facilitator agent code snippet

5.4.2 The Update of Social Parameter

As described in Chapter 3, the expert agent represents expert to propose argumentation utterance and records the expert's social parameters in the group wide. The social parameter of the agent mainly represents the interaction relationship among the different agents. In our system, we model the agent's social parameter as his credibility. During the group argumentation, the credibility of the expert is dynamically updated based on the argumentative response from other expert agent and it implicitly reflects the expertise of the expert. Which means more credible expert's opinion is more possible to be accepted within the group. Due to the multiple perspectives existing in a decision task, in our system we model the expert's credibility by the agent's profile from different domains of interest. In the implementation of the system, the profile is designed as a HashMap with a list of name-value pairs. The name represents the domain name and the values means the credibility value in this domain. The algorithm of dynamically updating credibility within the agent can be illustrated as Figure 5.18.

5.4.3 Argument Based Reasoning for Global Decision

The process of group argumentation produces a discussion tree, which is composed by the utterances, semantic link and evaluation criteria with the aggregated preference. During this process, the experts' domain credibility can be dynamically updated and derived. Based on the decision model discussed in Chapter 3, all the utterances defined as the solution in the argumentation ontology can be evaluated by their following arguments. The argument itself also can be evaluated by other argument or issue related with it. In a generic argumentation system, different proof standards are used for argumentation evaluation (called burden of the proof in the AI and Law). (Karacapilidis et al, 1996)

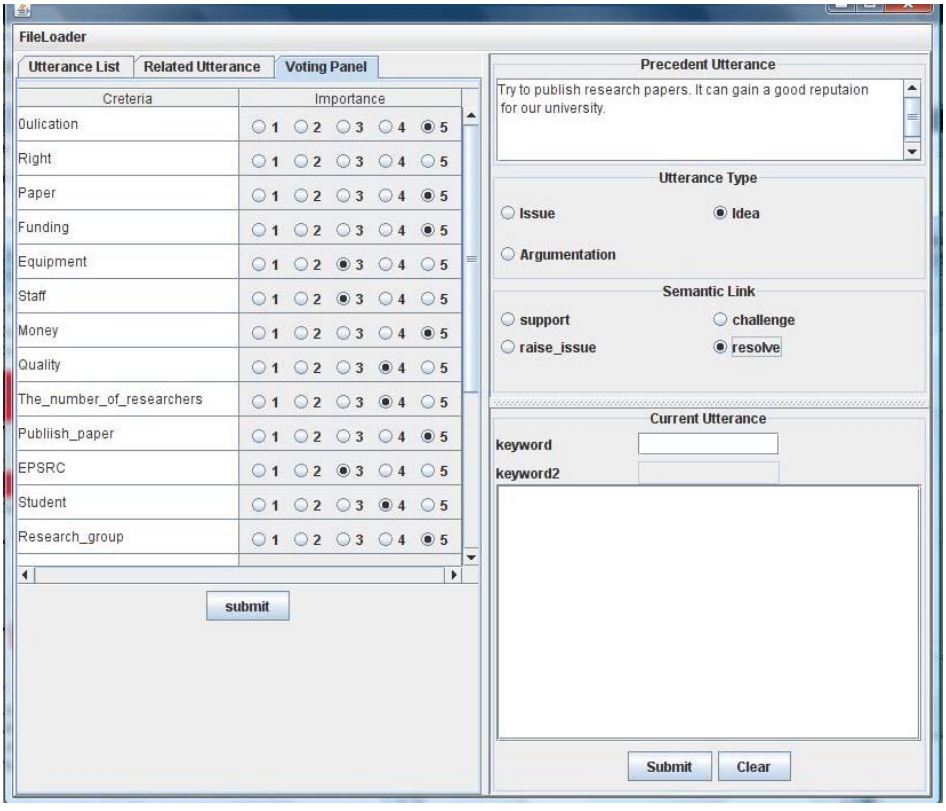

Figure 5.18 Present criteria list by Participant agent interface

(i) Scintilla of Evidence (SoE): a position is active if at least one active position is supporting it.

$$active(s_i) \Leftarrow \exists s_j \ni (active(s_j) \wedge support(s_j, s_i))$$

(ii) Beyond Reasonable Doubt (BRD): a position is active if there are not any active positions that are against it.

$$active(s_i) \Leftarrow \nexists s_j \ni (active(s_j) \wedge challenge(p_j, p_i))$$

(iii) Preponderance of Evidence (PoE): a position is active when the active positions that support it outweigh those that speak against it.

$$active(s_i) \Leftarrow score(s_i) \geq 0$$

$$score(s_i) = \sum_{active(arg_j) \wedge support(arg_j, s_i)} strength(arg_j) - \sum_{active(arg_k) \wedge challenge(arg_k, s_i)} strength(arg_k)$$

The above proof standard not only can be used for solution evaluation but also is appropriate for argumentation evaluation. The aim of our implementation is to better aggregate the group's expertise, so the PoE is adopted in the prototype system. However, SoE and BRD have their own usage in different contexts. The evaluation approach for strength of argumentation has been discussed in the decision function section of Chapter 3. Evaluation procedure in the decision making can be regarded as the process of updating the state and score of utterance. Similar to conventional decision making, the outcome of the evaluation is to assign a score to each solution utterance to reflect its order in all alternatives. The evaluation algorithm for an utterance is implemented in a function setState(Utterance utt), which is depicted in Figure 5.19. The evaluation for a whole discussion tree will start from the root – a task utterance and the evaluation for an utterance will be up to the

evaluation result of its following related argumentations. So the evaluation algorithm for an utterance will recursively call evaluation function of its following utterances, as described in Figure 5.19, until it reaches the leaf of the discussion tree.

```
void updateProfile(Utterance ut){
    Topic topic=ut.getPrecedent().getTopic();
     if (topic!=null)
      {
        Integer value=(Integer) profile.get(topic);
        if (ut.getAttribute().equalsIgnoreCase("challenge"))
        {
            value=value-2; //decrease the crediability
          if (value<0) value=0;
          profile.put(topic, value);
        }
      if(ut.getAttribute().equalsIgnoreCase("support"))
        {
         value=value+2; //increase the crediability
         profile.put(topic,value);         } }
   }
```

Figure 5.19 Update expert's credibility code snippet

From this decision making approach based on analysing and processing the discussion tree, the different aspects of top level solutions can be questioned and sub-solution (detail in some aspects) is allowed to be presented to answer these questions. So the solution can be decomposed into the sub-solution in the different levels. Due to all the utterances are enriched with semantic information and linked by other utterances, it is possible to visualize the solution space from user defined aspects such as displaying the solution with unsolved/solved issue under it, displaying all the solutions and related accepted positive/negative reasons, or displaying solution with its sub-solution in the detail aspects etc. This characteristic of our approach is distinctive compared with the conventional decision making.

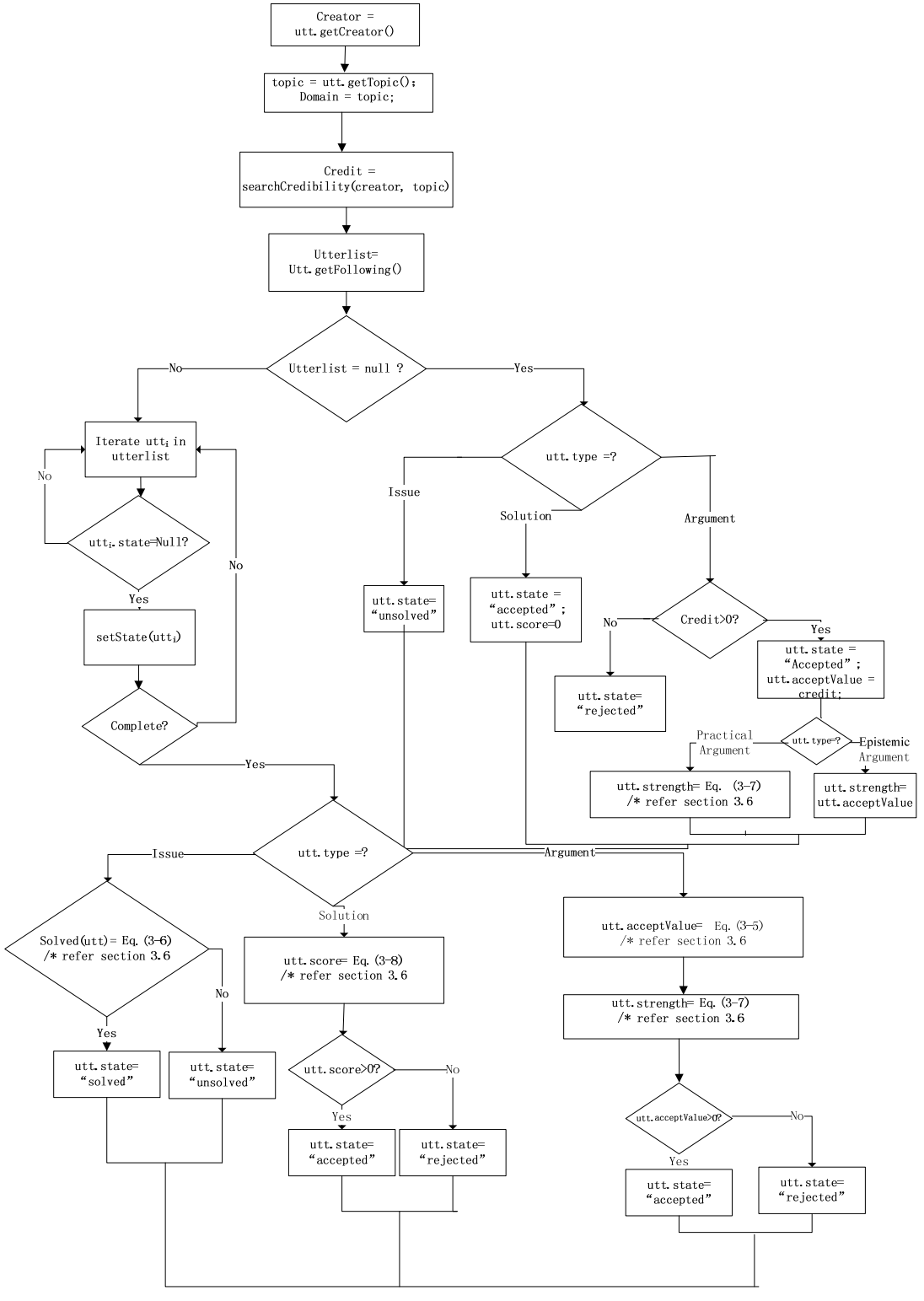

Figure 5.20 Argumentation based Decision Making algorithm