Auton Agent Multi-Agent Syst (2007) 14:187–206
DOI 10.1007/s10458-006-0013-z

Modeling conversation policies using permissions and obligations

Lalana Kagal · Tim Finin

Published online: 23 August 2006

Abstract Both conversation specifications and policies are required to facilitate effective agent communication. Specifications provide the order in which speech acts can occur in a meaningful conversation, whereas policies restrict the specifications that can be used in a certain conversation based on the sender, receiver, messages exchanged thus far, content, and other context. We propose that positive/negative permissions and obligations be used to model conversation specifications and policies. We also propose the use of ontologies to categorize speech acts such that high level policies can be defined without going into specifics of the speech acts. This approach is independent of the syntax and semantics of the communication language and can be used for different agent communication languages. Our policy based framework can help in agent communication in three ways: (i) to filter inappropriate messages, (ii) to help an agent to decide which speech act to use next, and (iii) to prevent an agent from sending inappropriate messages. Our work differs from most existing research on communication policies because it is not tightly coupled to any domain information such as the mental states of agents or specific communicative acts. Contributions of this work include: (i) an extensible framework that is applicable to varied domain knowledge and different agent communication languages, and (ii) the declarative representation of conversation specifications and policies in terms of permitted and obligated speech acts.

Keywords Agent communication · Conversation policy · Conversation specifications · Permissions · Obligations · Speech act ontology · Ontology languages

L. Kagal (✉)
Massachusetts Institute of Technology, Cambridge, MA 02139, USA
e-mail: lkagal@csail.mit.edu

T. Finin
University of Maryland, Baltimore County, Baltimore, MD 21250, USA
e-mail: finin@cs.umbc.edu

1 Introduction

Multi-agent systems require that agents interact and collaborate to satisfy their goals. Agent communication plays a very important part in these systems. A *conversation* can be defined as a sequence of communicative acts exchanged between interacting agents towards satisfying a particular goal [10, 12, 23]. Most rich agent communication languages have protocol specifications that require meaningful conversations to adhere to various structural constraints. For example, if one agent sends a *PROPOSE* message to another, meaningful responses are restricted to *ACCEPT-PROPOSAL* or *REJECT-PROPOSAL* messages. However, these conversation specifications or interaction protocols solely define the order in which communicative acts can be performed and do not take into consideration the content of the message, the attributes of the sender or the recipient or any other context. Similar to Phillips and Link [23], we propose that along with conversation specifications, agents should use policies that define constraints over different aspects of the conversation in order to provide more flexible control over agent communication. This also allows the communication modules of agents to be less dependent on the communication protocols permitting the modification of conversation specifications and policies without requiring the modules to be changed.

We differentiate between conversation specifications that define the order of communicative/speech acts and policies that affect how conversation specifications are used and how conversations are carried out. *Conversation specifications*, or interaction protocols as they are known within Foundation for Intelligent Physical Agents (FIPA) [9], define the order in which communicative acts can occur within a conversation. For example, on receiving a REQUEST communicative act, an agent can reply with REFUSE or AGREE [9]. On the other hand, we define *conversation policies* as restrictions on the conversation based on the content of the communicative act, the attributes of the sender and recipient including their beliefs, desires and intentions and other context like the current team they belong to, the time of day, and their location. For example, a conversation policy would *oblige* an agent to provide an evasive answer to a QUERY about a political issue in an office setting but *permit* it to provide a more truthful answer in a social setting. However, we also consider other policies such as privacy, work, and social that may establish additional restrictions and limitations on the communicative capabilities of the agent. Consider an agent that has a privacy policy prohibiting it from disclosing the social security number (SSN) of the user. Though the conversation specification provides the set of communicative acts the agent can use to reply to a QUERY, its privacy policy prohibits it from responding to any query that asks about the user's SSN.

In this paper, we describe our preliminary work in modeling conversation specifications and policies as positive/negative permissions and obligations. We also discuss how ontologies can help in developing high level policies that are independent of the specific agent communication language being used. We describe mechanisms for resolving conflicts between specifications and policies that enable an agent to decide what communicative act to perform next within a conversation. Our work differs from most previous research in communication policies in that it is not tied to a specific model of agent mental states, a specific agent communication languages, nor to a specific set of communicative acts [8, 12, 27, 28]. Rather, we provide an extensible framework that can be used to develop conversation protocols and policies over different kinds of domain-specific knowledge and different agent communication languages.

As an example, we describe the issue with the Medicare prescription drug bill in the United States [2] in terms of agent communication. According to the CNN article, Rick Foster, chief actuary for the Centers for Medicare and Medicaid Services, stated that he was asked not to answer questions from Democratic members of the United States Congress regarding the cost of the bill before a series of key votes summer of 2003. We describe how this would have worked within a multi-agent system driven by our conversation specifications and policies. Agents, including Foster, have a conversation specification that states that in response to a QUERY, the agent is *permitted* to use either AGREE followed by an INFORM/FAILURE or REFUSE or ignore the message. Their work policy states that all state employees are *obliged* to answer queries from members of Congress. However, agency chief Thomas Scully, has enforced a temporary policy of the highest priority on Foster stating that he is *obliged* not to answer queries from congressional Democrats regarding the estimated cost of the Medicare prescription drug bill until the end of summer. A sanction is associated with the failure to fulfill this obligation that states that Foster could lose his job.

Whenever Foster receives a message, he reasons over his conversation specifications and policies to figure out how he should respond. When he receives a QUERY from a congressional Democrat asking about the estimated cost of the bill he knows from the conversation specifications that the correct response is AGREE or REFUSE. As his work policy *obliges* him to answer all queries from congressional members, under normal circumstances Foster would agree. However, as Scully's temporary policy overrides the work policy and because of the associated sanction, Foster follows Scully's policy and REFUSEs the query. Scully's policy could also include rules *obliging* Foster to send an evasive reply to the congressional Democrats instead of refusing to answer.

2 Framework

A communicative or speech act is defined in terms of the set of actions that are implied when an agent makes an utterance. Generally, there are three actions that can be identified: (i) locution, which is the action of uttering the speech act, (ii) illocution, which deals with conveying the intentions of the sender, and (iii) perlocution, which are actions that occur due to the illocution. Conversation specifications define the sequence in which communicative acts can be performed in order for agents to have a meaningful dialog. We model them as a set of permissions and obligations on speech acts based on the communicative acts exchanged thus far. We believe that conversation specifications should be very simple and only provide a list of possible speech acts that can be performed at a given time. This list of possible acts is then restricted by the policies acting on the agent. However, within our framework it is also possible to develop complex specifications that resemble policies.

Though our policy language [13, 14] is defined in Web Ontology Language (OWL) [31], an ontology language used to describe vocabularies on the web, for conciseness and ease of explanation, we use expressions in predicate logic to describe speech acts, positive and negative deontic objects, and policies. Variables begin with upper case alphabets, instances and property names begin with lower case alphabets.

- A *communicative or speech act* is performed by an agent to achieve a certain intention. A speech act is usually assumed to have two main components; the performative and the proposition.
  We describe a communicative act as a tuple

  ```
  performative(Sender, Receiver, Proposition)
  ```

  For example, a QUERY-REF speech act sent from agent X to agent Y asking agent Y what he believes the values of the included proposition to be

  ```
  query-ref(agent X, agent Y, estimatedCostOfBill(Cost))
  ```

- Domain actions are actions that an agent can perform and are described by the following tuple

  ```
  action(Actor, Target, PreCondition, Effect)
  ```

  The *printAPage* domain action can be described as

  ```
  printAPage(X, hpLaitPrinter,
      (numPages(hpLaitPrinter, N), N>0),
      (numPages(hpLaitPrinter, N-1)))
  ```

- Deontic concepts of permissions, prohibitions (negative permissions), obligations and dispensations (waiver from an obligation) are used to describe the behavior of the agent. These concepts are represented by the following tuple

  ```
  deontic(Actor, Action, [Constraint],
  [StartingConstraint], [EndingConstraint], [ObligedTo],
  [Sanction])
  ```

  Consider the permission of an agent to perform an AGREE speech act to any agent regarding the estimated cost of the Medicare prescription bill. This is considered a policy as it includes domain knowledge of the proposition used to model the cost of the bill.

  ```
  permission(X, agree(X, Y, estimatedCostOfBill(Cost)))
  ```

  Permissions and prohibitions are used to describe positive/negative authorizations whereas obligations and dispensations describe positive/negative responsibilities. All these objects could be represented in terms of a single concept, either permission or obligation, but we use different terms for simplicity.

  Associated with each deontic object is either a *constraint* field, which defines the conditions under which the deontic object is applicable, or *startingConstraint* and *endingConstraint* that together define the window within which the deontic object is applicable. These constraints could also include conditions on time providing time validity to the deontic object. Obligations and dispensations have an additional field, *obligedTo*, which describes whom the agent is obliged. Another property called *sanctions* is associated with both obligations and prohibitions and is used to describe the penalties imposed on the agent if it fails to fulfill the obligation or violates the prohibition. Consider a policy of a graduate assistant that obliges him to turn in a weekly status report to his advisor or risk missing a pay check.

```
obligation(X, inform(X, Y, weeklyStatus(X, W, Status)),
  (advisor(Y, X), endOfWeek(W)), Y, missPayCheck (X, W))
```

A *permission* allows an agent to perform the associated action as long as the constraint is true or while the startingConstraint is true and until the endingConstraint is true.[1] A *prohibition* prevents an agent from performing the associated action as long as the constraint is true or while the startingConstraint is true and until the endingConstraint is true. An agent must perform an *obligation* before the endingConstraint is true and while the startingConstraint is true. An agent is no longer obliged to fulfill an obligation if there is an associated *dispensation* freeing the agent from the obligation.

In order to impose a *sanction* it is important to reason about violations. An agent is in violation of a prohibition if it performs the prohibited action while the *constraint* on the prohibition is true or within the window defined by the *starting Constraint* and *endingConstraint*. Similarly an agent is in violation of its obligation if it fails to complete it before the *endingConstraint* is true. In order to simplify reasoning over violations we make the following assumptions: (i) the constraints are conditions that are true at some point of time, (ii) the *endingConstraint* will be true after the *startingConstraint* is true, and (iii) obligations do not exist after the *endingConstraint* is true [6].

- Boolean combinations of actions including *AND* (logical conjunction), *OR* (logical disjunction), and *XOR* (exclusive disjunction or exclusive OR) are possible within deontic concepts.

  An agent is permitted to perform both an AGREE speech act as well as a REFUSE speech act to any agent regarding the estimated cost of the Medicare prescription bill.

```
permission(X, AND(agree(X,Y, estimatedCostOfBill(Cost)),
              refuse(X, Y, estimatedCostOfBill(Cost)) ))
```

- In our framework conflicts can occur between permissions and prohibitions, obligations and prohibitions, and obligations and dispensations. In order to resolve conflicts, our framework includes meta-policies that are used to correctly interpret policies. There are two kinds of meta-policies namely setting the modality precedence (negative over positive or vice versa) or stating the priority between rules within a policy or between policies [22].

  In a multi-policy environment, it is possible to state that one policy overrides another. For example, it is possible to say that in case of conflict the CS department policy always overrides the Lait lab policy. As another example, consider the CS department policy. Students are prohibited from using the faculty printer but research assistants are permitted to. There is a potential conflict if a student is a research assistant and needs to use the faculty printer. This can be solved by setting the priority between the rules and stating that the permission overrides the prohibition.

```
rule1 : prohibition(X, print(X, facultyPrinter),
        student(X), _)
```

[1] Though startingConstraint and endingConstraint can be combined into a single constraint, we have separated them for pragmatic reasons.

```
rule2 : permission(X, print(X, facultyPrinter),
        researchAssistant(X))
        overrides(rule2, rule1)
```

Our approach is based on statements that one policy has priority over or dominates another and within a policy that one rule dominates another. The assertions produce a *priority graph* over policies and, for each policy, a priority graph for policy rules. In such a scheme we have to worry about two issues: the presence of cycles in the priority graph and of having only a partial ordering when a total ordering is desired. The first problem is a serious one can be avoided by analyzing the priority graphs to detect cycles and reporting any found as "bugs" to be corrected by the policy authors. Whether the second issue is a problem or not is a design issue. We propose to induce a possible total ordering, if necessary, in the priority graph for policies and for rules within a policy.

If instead of priorities, modality precedence is used, then when a conflict occurs the rule with the preferred modality overrides the other. For example, if positive modality is preferred then in case of conflict, permissions and obligations will override prohibitions and obligations will override dispensations. The conflict in the CS department policy in the earlier example can also be resolved if positive modality is given precedence.

```
precedence(positive-modality)
```

- Our framework allows the default behavior of a policy to be set such that all actions being controlled by the policy are permitted if not explicitly prohibited by the policy, prohibited if not explicitly permitted by the policy, or require explicit permissions and prohibitions.

```
default-behavior(implicitpermexplicitproh)
```

- We also use some additional expressions to describe the sequence of message that have been exchanged so far in an actual dialog.
  - received(M): states that a message, M, was received.
  - rec-notrespondedto(M): states that a message, M, was received but no response has been sent as yet.
  - rec-respondedto(M): states that a message, M, was received and a response has been sent.
  - sent(M): states that a message, M, was sent.
  - incomplete-sent(M): states that a message, M, was sent, but another message(s) is required to complete the conversation, e. g., if an agent sends an AGREE as a response to a QUERY-REF, the agent is required to follow up with either an INFORM or a FAILURE.

3 Conversation specifications

Using the semantics of the deontic objects and domain actions and the syntax of speech acts, we can model conversation specifications in agent communication languages such as Knowledge Query and Manipulation Language (KQML) [8, 18] or FIPA [9] as a set of permissions and obligations on the sender or the receiver depending on the performatives used thus far in the conversation.

As examples, we describe one possible interpretation of the QUERY-REF specification and proposed interaction in FIPA.

3.1 QUERY–REF Specification

- Speech acts used: QUERY-REF, REFUSE, AGREE, FAILURE, INFORM
- Sequence of messages: An agent sends a QUERY-REF message to another agent. The latter can reply either with a REFUSE or an AGREE stating its intent to either provide an answer or refuse to answer. Once an agent has sent an AGREE, it is obliged to send an INFORM providing the information required.

  – Every agent has the permission to perform a QUERY-REF performative

  ```
  permission(X, query-ref(X, Y, Proposition))
  ```

  In the above expression, the constraint field is left empty to specify that there are no constraints on the performing of a QUERY-REF performative.
  – On receiving a QUERY-REF, the recipient has the permission to REFUSE the query or AGREE to provide the answer if the query has not already been answered.

  ```
  permission(Y, XOR(agree(Y, X, Proposition),
                    refuse(Y, X, Proposition))))
  permission(Y, refuse(Y, X, Proposition),
      rec-notrespondedto(query-ref(X,Y, Proposition)))]]
  permission(Y, agree(Y, X, Proposition),
      rec-notrespondedto(query-ref(X,Y, Proposition)))
  ```

  Though the recipient has both permissions to start with, as soon as one permission is used the other one becomes invalid because of the status of the rec-notrespondedto predicate. The constraint ensures that the agent has received a QUERY-REF speech act but has not sent a response to it as yet. This allows the agent to either send a REFUSE or AGREE in reply to a QUERY-REF but not both.
  – Once an agent has accepted a QUERY-REF, it is obliged to answer to it either with a FAILURE or with an INFORM and the agent is obligated to the recipient of the agree message.

  ```
  obligation(Y, XOR(failure(Y, X, Proposition),
                    inform(Y, X, Proposition) ),
           incomplete-sent(agree(Y, X, Proposition) ) ).
  ```

  Unlike the earlier permissions, the agent has only one obligation—the obligation to either send a failure or an inform.

3.2 Propose Interaction Protocol

- Speech acts used: PROPOSE, REJECT-PROPOSAL, ACCEPT-PROPOSAL
- Sequence of messages: An agent sends a PROPOSAL message to another agent. The recipient can either use the REJECT-PROPOSAL or the ACCEPT-PROPOSAL.

– Every agent has the permission to perform a PROPOSAL performative

```
permission(X, proposal(X, Y, Proposition))
```

– On receiving a PROPOSAL, the recipient has the permission to either reject the proposal or accept it.

```
permission(Y, X or (accept-proposal(Y, X, Proposition),
      reject-proposal(Y, X, Proposition)))
permission(Y, accept-proposal(Y, X, Proposition),
      rec-notrespondedto(proposal(X,Y, Proposition)))
permission(Y, reject-proposal(Y, X, Proposition),
      rec-notrespondedto(proposal(X,Y, Proposition)))
```

4 Policies

Policies such as conversation, social, and privacy add restrictions on the performatives that can be used, the content of the speech the receiver, time of the message, and other attributes of a dialog based on current attributes of the sender, receiver, content and all other context of the conversation. Policies can be defined at two levels; one that is independent of the syntax and semantics of the communication language and the second that is tightly integrated with them. In the latter case, the policies use the semantics of the performative and define constraints on how performatives can be used and under what conditions. Though this may be true in the case of conversation policies, we generally assume that policies such as privacy, and social norms define restrictions at the higher level of abstraction that regulate the general behavior of the agent. Whenever these policies deal with information flow between agents, they need to be translated into lower level policies using the semantics of the communication language. For example, an agent's privacy policy might state that the SSN must not be disclosed. This is irrespective of the agent communication language being used or the specific performative. If FIPA is being used, the privacy policy could be translated in our framework as 'The agent is prohibited from sending an INFORM communicative act to any agent when the content involves the SSN of the agent'. However, if KQML is the language being used for communication, the semantics specify that TELL is the only assertive performative that causes the agent to reveal its belief about a proposition. In this case, the policy could translate to 'The agent is prohibited from sending a TELL communicative act to any agent when the content involves the SSN of the agent'. Similarly, a social policy can specify that an agent should not be rude. However, what it means to be rude and how it translates into speech acts and their content depends on the application domain. The agent would have to ensure that the effect of any speech act does not violate this social policy.

Following from the first example dealing with the Medicare bill, it is evident that there could be several policies acting on an agent. This could lead to *conflicts* between policies. Foster's conversation specifications gave him the permission to reply to requests, however, the agency head prohibited him from replying to queries about the estimated cost of the Medicare bill. In Foster's case, Scully's policy would be enforced if it was of higher priority than Foster's other policies.

5 Speech act ontology

As a first step towards automatic translation of high level policies to speech act specific policies, we suggest that a general ontology be defined for different kinds of the speech acts. An ontology is a representation of the knowledge domain—in this case of the speech acts, their properties, and their relationships. Having an ontological description of speech acts allows policies to be defined in terms of the high level ontological concepts and properties and not in terms of the actual speech acts. This helps in further decoupling policies from the agent communication language and allows for greater interoperability. Figure 1 illustrates a simple and partial ontology for speech acts.

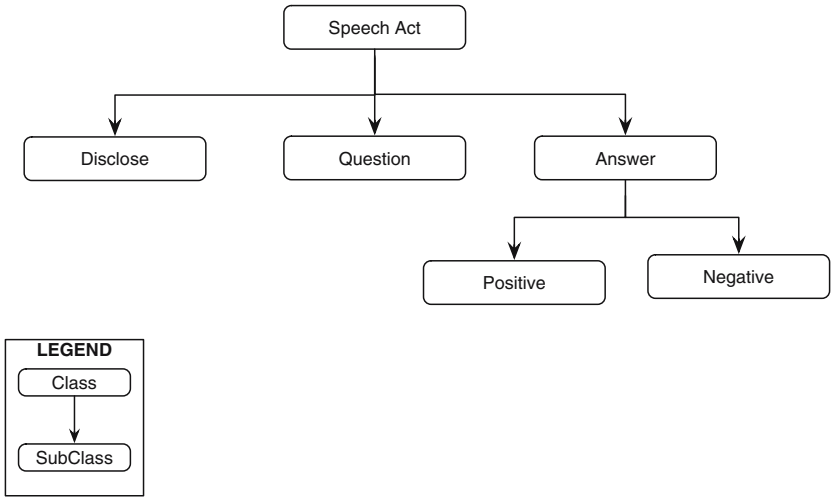

Fig. 1 A simple and partial ontology of generalized speech acts provides a framework to accommodate numerous specific agent communication languages

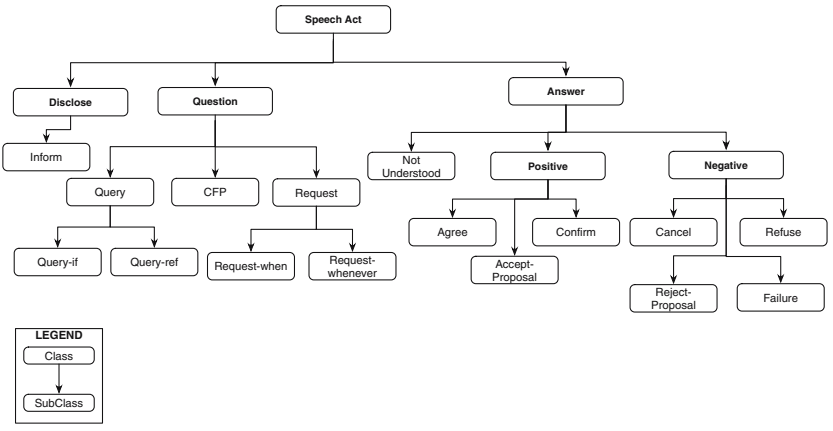

Fig. 2 Our simple "upper model" ontology for speech acts can be extended to include the major communicative acts specified by the FIPA ACL

The simple speech act ontology act can be extended to express speech acts in different agent communication languages. Figure 2 shows how the speech act ontology can be extended to describe FIPA speech acts.

High level policies can be defined in terms of categories or classes of speech acts instead of individual speech acts. These policies will automatically apply to all speech acts that belong to the specified categories. This kind of policy representation and reasoning is supported by our OWL-based policy language [13].

As example, consider a previously described privacy policy that states that the SSN number should not be disclosed. If the use of *disclose* speech acts is prohibited, the agent is not able to let any one know what the SSN number is. On the other hand, *answer* speech acts allow the agent to respond positively or negatively to questions about the SSN numbers. These responses could lead to someone figuring out what the SSN number is by asking enough questions. So this privacy policy can be defined by adding a prohibition to all *disclose* and *answer* type of speech acts in the ontology.

```
prohibition(X, disclose(X, Y, ssn(X)), _, _)
prohibition(X, answer(X, Y, ssn(X)), _, _)
```

The above policy works for both FIPA (INFORM is a disclose type of speech act) and KQML (TELL is a disclose type of speech act) agent communication languages.

As another example, consider a permission described earlier. An agent is permitted to perform an AGREE speech act to any agent regarding the estimated cost of the Medicare prescription bill. This can be restated in terms of the ontology as an agent being permitted to use a *positive answer* type of speech act when asked by any agent about the estimated cost of the Medicare prescription bill.

```
permission(X, positive-answer(X, Y, estimatedCostOfBill
                                              (Cost)) )
```

This policy will apply to any agent communication language that can be described in the above speech act ontology and not just FIPA.

6 Example

We now discuss the Medicare bill example in terms of our approach. We assume that the agent communication language used is FIPA and both Foster and Scully share the same conversation specifications. These specifications include the QUERY-REF specification described in Sect. 3. We also assume that the FIPA speech act ontology is being used to define policies.

- Foster has a work conversation policy that specifies that he should respond to all queries from members of Congress. This is defined as two obligations: (i) an obligation to reply positively to all query type of speech acts from members of Congress, and (ii) and an obligation to disclose required information if they have answered positively to a request.

```
ConvPolicy :
obligation(X, positive-answer(X, Y, Proposition),
  (rec-notrespondedto(query(Y, X, Proposition)),
  congressionalMember(Y)),
  X, _)
obligation(X, disclose(X, Y, Proposition),
  (sent-incomplete(positive-answer (X, Y, Proposition)),
  congressionalMember(Y)),
  X, _)
```

- Scully decides that Foster should not answer any queries from congressional Democrats that ask about the estimated cost of the Medicare prescription bill. This is a high level policy and could be translated based on the syntax and semantics the FIPA ACL in two ways:

  1. As an obligation to provide a negative answer to all queries about the estimated cost of the bill from congressional Democrats.

```
TempPolicy:
obligation(X, negative-answer(X, Y, estimated CostOfBill(Cost)),
  (rec-notrespondedto(query(Y, X, estimated CostOfBill(Cost))),
  congressionalDemocrat(Y)),
 scully, loseJob(foster))
```

  2. As a prohibition from disclosing information about the estimated cost of the bill to any congressional Democrat.

```
TempPolicy:
prohibition(X, disclose(X, Y, estimated CostOfBill(Cost)),
       (congressionalDemocrat(Y)), _)
```

  We discuss the rest of the example using both choices.
- Scully gives his policy higher priority than the existing conversation policy.

```
overrides(TempPolicy, ConvPolicy)
```

- At some point of time, a congressional Democrat, Walter, sends a query to Foster asking about the estimated cost of Medicare bill.

```
query-ref(walter, foster, estimatedCostOfBill(Cost))
```

- On receiving this speech act, Foster looks up the conversation specifications for QUERY-REF, and finds that he can respond either with an AGREE or REFUSE.
- Foster checks his work conversation policy, which states that he is obliged to answer all query kind of speech acts (both QUERY-IF and QUERY-REF speech acts) from congressional Democrats with a positive answer (AGREE) and then send the required information.
- Foster then reasons over Scully's policy that is of higher priority than his work conversation policy. If Scully's policy states that Foster is obliged to answer negatively to all queries from congressional Democrats about the estimated cost of the

bill, then Scully is forced to send a REFUSE to Walter as by violating the policy he could lose his job.

```
refuse(foster, walter, estimatedCostOfBill(Cost))
```

On the other hand, if Scully's policy states that Foster is not permitted to disclose any information to congressional Democrats about the estimated cost of the bill, then Foster can act in two ways and still comply with Scully's policy. He can either send a REFUSE to Walter or he can send an AGREE to Walter but then not follow up with an INFORM. The second option is a common social phenomenon where people commit to performing certain tasks, which they would prefer to turn down, just because they do not want to say no. However, they do not actually fulfill their commitments and either make an excuse when asked or hope the requester has forgotten. Using this option allows Foster to maintain good relations with Walter and still conform to Scully's policy. However, he violates his work policy which states that after sending a positive answer he is obliged to send the required information.

```
agree(foster, walter, estimatedCostOfBill(Cost))

Foster does not send:
inform(foster, walter, estimatedCostOfBill(Cost))
```

7 Specification language

Though we use an abstract syntax to describe the examples in this paper for ease of explanation and for conciseness, our framework has actually been tested in Rei [13, 14], a policy language described in OWL [31]. OWL is a vocabulary extension of RDF (Resource Description Framework) [30], which is a language for asserting facts about resources and their properties. OWL can be used to describe ontologies as well as relationships between ontologies. Rei extends OWL with variables in order to support rule-based policies. Rei policies are defined over Rei ontologies in OWL and domain knowledge express in RDF. This allows them to take advantage of the growing use of RDF and OWL as an interlingua in which to publish ontologies and data on the web. The policies themselves are expressed in OWL that allows the policies to be mapped into any of several reasoning engines that can then be used to reason over the policies and the ontologies and data they reference.

Figure 3 shows part of the Rei ontology.

Our policy language is modeled on deontic concepts of permissions, prohibitions, obligations and dispensations [14, 15]. We believe that most policies can be expressed as what an entity can/cannot do and what it should/should not do in terms of actions, services, and conversations, making our language capable of describing a large variety of policies ranging from security policies to conversation and behavior policies. The policy language has some domain independent ontologies but will also require specific domain ontologies. The former includes concepts for permissions, obligations, actions, speech acts, etc. The latter is a set of ontologies, used by the entities in the system, which defines domain classes and properties associated with the classes.

As an example, consider a policy that allows all members of congress to perform printing actions on a certain resource. The Rei namespaces include 'entity', 'action',

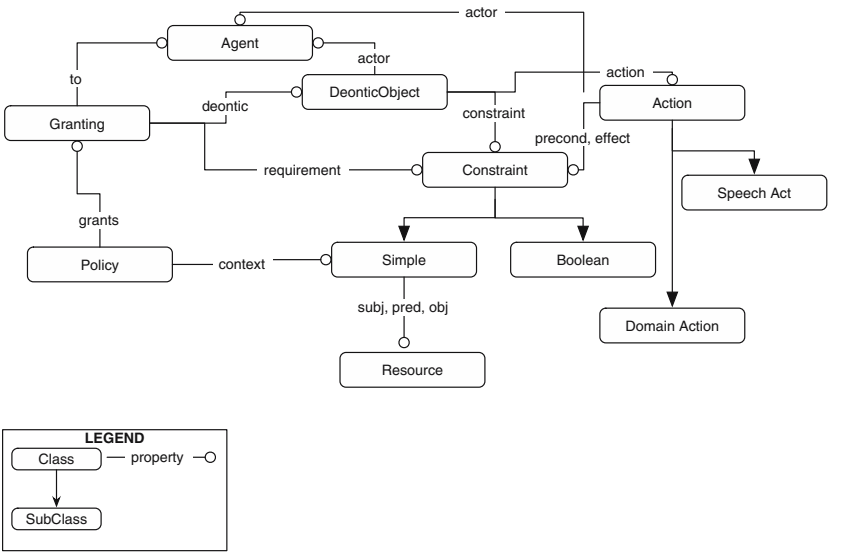

Fig. 3 Rei's ontology defines concepts and relations that can be used to express policies over actions

'constraint', 'metapolicy', and 'deontic'. The members of congress is defined in the 'gov' namespace whereas the printing action ontology is in the 'abc' namespace.

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<!DOCTYPE rdf:RDF [
        <!ENTITY constraint 'http://rei.umbc.edu/
         ReiConstraint.owl#'>
        <!ENTITY deontic 'http://rei.umbc.edu/
         ReiDeontic.owl#'>
        <!ENTITY metapolicy 'http://rei.umbc.edu/
         ReiMetaPolicy.owl#'>
        <!ENTITY entity 'http://rei.umbc.edu/
         ReiEntity.owl#'>
        <!ENTITY action 'http://rei.umbc.edu/
         ReiAction.owl#'>
        <!ENTITY gov 'http://example.org/gov.rdf#'>
        <!ENTITY abc 'http://example.org/abc.owl#'>
]>
<entity:Variable rdf:ID="Actor"/>
<entity:Variable rdf:ID="Action"/>
```

```
<constraint:SimpleConstraint rdf:ID="IsMemberOfCongress"
     constraint:subject="#Actor"
     constraint:predicate="&rdf;type"
     constraint:object="&gov; CongressMember"/>

<constraint:SimpleConstraint rdf:ID="IsPrintingAction"
     constraint:subject="#Action"
     constraint:predicate="&rdf;type"
     constraint:object="&abc; Printing"/>
<constraint:SimpleConstraint rdf:ID="OnHPPrinter"
     constraint:subject="#Action"
     constraint:predicate=" &action;target"
     constraint:object=" &abc; AHPPrinter"/>

<constraint:And rdf:ID="ActionConstraint">
     <constraint:first rdf:resource=" # IsPrintingAction"/>
     <constraint:second rdf:resource=" # OnHPPrinter"/>
</constraint:And>

<constraint:And rdf:ID="MainConstraint">
<constraint:first rdf:resource="#IsMemberOfCongress"/>
<constraint:second rdf:resource="#ActionConstraint"/>
</constraint:And>

<deontic:Permission rdf:ID="Perm_MemberCongPrinting">
<deontic:actor rdf:resource="#Actor"/>
<deontic:action rdf:resource="#Action"/>
<deontic:constraint rdf:resource="MainConstraint"/>
</deontic:Permission>
```

The policy language includes two constructs for specifying meta-policies that are invoked to resolve conflicts; setting the modality precedence (negative over positive or vice versa) or stating the priority between policies [19, 20]. When modality precedences are used to resolve a conflict, the rule of the preferred modality overrides the other. As an example of using priority consider a meta policy that states that in case of conflict the Federal policy always overrides the State policy.

```
<metapolicy:PolicyPriority rdf:ID="PriorityFederalState">
     <metapolicy:policyOfGreaterPriority
                              rdf:resource="&gov; Federal"/>
     <metapolicy:policyOfLesserPriority
                              rdf:resource="&gov; MAState"/>
</metapolicy:PolicyPriority>
```

Another important aspect of the framework is that it models speech acts such as delegation, revocation, request and cancel that can be used to dynamically modify existing policies [13, 14]. Delegations and revocations cause the permissions of agents to be modified, whereas requests and cancels affect the obligations. A delegation speech act, if valid, causes a permission to be created. A revocation speech act nullifies an existing permission (whether policy based or delegation based) of an agent. An agent can request another agent for a permission or to perform an action on its behalf. The former if accepted causes a delegation and the latter leads to an obligation. An agent can also cancel any previously made request causing a dispensation.

Rei also provides two forms of analysis : use-cases (also known as test-case analysis) and what-if analysis (also known as regression testing) [13, 14]. The policy engine includes analysis tools in the form of a Java interface that can be executed by policy engineers to check the consistency and validity of the policies and ontologies. In use-case management, the policy maker can specify a set of use cases that are verified against a set of policies. If the policies and ontologies are consistent, the use case will be true. If the use case is false, the policy maker is aware that there is an error in specifications of the policies. On the other hand, what-if analysis is used to help the policy maker decide what changes to make to the policy or ontology. For example, if Rule 1 no longer applies, will Walter still have the permission to print to the HPPrinter is modeled as follows:

```
<analysis:WhatIfIRemoveRule rdf:ID="RemovingRule1">
    <analysis:policy rdf:resource="&govpolicy; PrintPolicy"/>
    <analysis:granting rdf:resource="&govpolicy; Rule1"/>
</analysis:WhatIfIRemoveRule>

<analysis:PermissionUseCase rdf:ID="CanWalterPrint">
    <analysis:actor rdf:resource="&gov; Walter"/>
    <analysis:action rdf:resource="&abc; Printing"/>
    <analysis:target rdf:resource="&abc; HPPrinter"/>
</analysis:PermissionUseCase>
```

8 Policy reasoning

Rei has a reasoning engine in Flora [32], an extension of XSB[2] based on F-logic [16]. The engine is built over F-OWL [34], a reasoner for OWL and RDF, enabling it to understand and reason over policies and specifications defined in both OWL and RDFs. Though the framework includes boolean combinations of actions within deontic concepts, the policy engine does not currently support this. The engine reasons over policies, meta policies, history of speech acts, and domain information to answer the following types of questions :

- What are the current permissions of X?
  The engine looks for all those permissions whose actor property unifies with X and whose constraints are satisfied. If there is a conflicting prohibition or revocation, the engine uses the meta policies to decide whether the permission overrides

[2] XSB: http://xsb.sourceforge.net/

the prohibition/revocation or vice versa. If the latter case is true, the permission is not valid. If the permission is valid, the policy engine checks the preconditions associated with the action over which the permission is specified. The permission is returned only if the precondition is satisfied.

The engine also looks for valid delegations from any agent to X. The delegation is valid if the delegatee has the permission to make the delegation or has been delegated the permission to make the delegation. The entire delegation chain is checked by policy engine. At every level, the engine also checks that there is no conflicting prohibition or revocation.

- What are the current obligations of Y?

  The engine locates all obligations whose actor property unifies with Y and whose startingConstraint is satisfied but whose endingConstraint is false. The engine ensures that there is no conflicting dispensation.
- Does X have the permission to perform action A or speech act S?

  This is similar to the first case, but in this case, the policy engine also checks the action property of the permission and verifies that it unifies with A or S.
- Does X have any permissions on a resource R?

  This is similar to the first case, but the policy engine also tries to unify the target property of the action associated with the permission with R.
- If policy P no longer applies, does agent X still retain the permission to use the QUERY speech act for Proposition P?

  This is part of the policy analysis provided by Rei. The policy engine deletes P but stores it in a temporary list. It then tries to verify that X has the permission to use QUERY speech act over P. It returns the answer and then restores P.

We envision that the reasoning engine will be used together with domain knowledge such as the mental state of the agents, the history of the speech acts performed and other context by either a planning component or a workflow component to enable enforcement of policies over agent communication.

Using the policy engine, our earlier example of the Medicare bill would be inferred by Foster as

1. Received QUERY-REF from congressional Democrat enquiring about the cost of the Medicare bill
2. What are my current obligations? I am obliged to AGREE to answer by my conversation work policy. I am also obliged to REFUSE the query that deal with the cost of the bill by Scully's policy.
3. As the meta policy states that Scully's policy has the highest priority, I execute it first.
4. So, I REFUSE the query.
5. However, do I have the permission to REFUSE a query? Yes, from the conversation specifications.

9 Related work

Cohen and Levesque model the cognitive state of agents and base allowable speech acts on the cognitive states of collaborating agents [3]. In his earlier work, Singh provides semantics for speech acts in terms of beliefs and intentions of the agents

[25, 26] and later work [29, 33] has focused on constraints imposed by the underlying commitments that agents acquire. Fornara and Colombetti [11] describe an approach based on the notion of social commitment. Labrou and Finin also describe the semantics of KQML based on the beliefs and desires of agents [8, 18]. These models are very tightly coupled to the mental states of agents and the semantics of the language that makes it difficult to extend them to work in different environments and with different agent communication languages. Pitt and Mamdani [24] have advocated an approach in which the ACL semantics itself is defined by protocols which, in their model, can involve domain actions and constraints. Cost et al. [4] develop a model using colored petri nets that can take into account various contextual properties and attributes. Greaves et al. define conversation policies as restrictions on how the agent communication language is used [12]. Though the last approach is similar to ours, we believe that conversation policies should be at a higher level of abstraction and should not involve specifics of the communication language. We also propose that all policies related to communication be translated into permissions and obligations that the agent has on speech acts, which are supported by the communication language being used.

Dignum formally describes agent communication in terms of the effects of this communication [5, 7]. He defines four speech acts for requesting, committing, asserting, and declaring and describes their effects in terms of added permissions and obligations, and changes in beliefs and intentions of agents. Dignum uses these speech acts and their effects to model communication protocols. This approach, though interesting, is based on the beliefs and intentions of agents and has a different focus from ours. It focusses on how commitments and permissions caused by communication allow dynamic protocols to occur based on certain situations and goals of agents. On the other hand, our main goal is to develop a domain and communication language independent framework that can be used to represent pre-defined protocols using permissions and obligations and that uses policies to restrict how these protocols are used.

Kollingbaum and Norman discuss how normative agents estimate the effect of adopting a new norm [17]. The current beliefs, norms and the selected plan are taken into consideration while estimating the level of consistency that will be brought about by the adopted norm. This work approaches the adoption of norms (or what we call policies) under the assumption that the agent can decide whether or not to accept a norm. Though this is advisable for contracting agents, we believe that certain policies are enforced by the environment and must be accepted by the agent irrespective of whether they cause conflicts or inconsistencies in the agent's current state. Also, Kollingbaum's approach does not try to resolve conflicts, it only categorizes the type of conflict in terms of consistency and uses this information to decide whether or not to accept a new norm.

Broersen et al. use agent types to resolve conflicts between beliefs, obligations, intentions and desires [1]. The agent types are determined by their characteristics namely social (obligations overrule desires), selfish (desires overrule obligations), realistic (beliefs overrule everything else) and simple-minded (intentions overrule obligations and desires). In our framework, conflicts basically occur between permissions and prohibitions, obligations and prohibitions, and obligations and dispensations. In order to resolve conflicts, our framework includes meta-policies namely setting the modality precedence (negative over positive or vice versa) or stating the priority between rules within a policy or between policies. Broersen et al. approach conflict resolution from the agent's point of view whereas we try to resolve conflicts in policies within the environment and not within agents themselves. We believe that Broersen's

approach or something similar could be used by agents after conflict resolution is provided by our framework as the enforced policies may conflict with the agent's internal beliefs, desires, intentions, and policies.

10 Summary

In this paper, we do not try to define the semantics of a particular agent communication language or a set of performatives but provide a flexible model that can be used to describe conversation specifications and policies over different agent communication languages such as KQML and FIPA using different domain-specific information. The framework allows specifications to be described as a sequence of permitted and obligated speech acts. Policies are described at a high level of abstraction and are translated into positive/negative permissions and obligations over speech acts using the semantics of the agent communication language. A speech act ontology is used to aid in this translation. The permissions and obligations in a policy establish restrictions over attributes of the sender, receiver, content, and other context of the conversation such as time, and location. As part of our future work, we are looking into automating the translation process from high level policies to performative specific permissions and obligations.

Though we described all our examples in an abstract syntax, our actual specification language is in OWL. We have developed a reasoning engine for our language that reasons over domain knowledge, speech act semantics, protocols, policies, and meta policies to answer questions about the permissions and obligations of an agent with respect to the actions and speech acts it can/should perform. We envision that this reasoning engine will be coupled with the planning/workflow component of an agent to provide policy enforcement over agent communication. We are also interested in integrating into our framework work on commitments like that by Mallya et al. [21], which involves reasoning over the status of obligations of agents.

References

1. Broersen, J., Dastani, M., Hulstijn, J., Huang, Z., & L. van der Torre. (2001). The BOID architecture conflicts between beliefs, obligations, intentions and desire. In *fifth international conference on autonomous agents* (pp. 9–16).
2. Cable News Network (CNN) (2004). Probe under way on medicare cost. http://www.cnn.com/2004/ALLPOLITICS/ 03/17/medicare.investigation/
3. Cohen, P. R., Levesque, H. J. (1990). Intention is choice with commitment. In *Artificial Intelligence, 42*, 213–261.
4. Cost, R. S., Chen, Y., Finin, T., Labrou, Y., & Peng, Y. (2000). Using colored petri nets for conversation modeling. In F. Dignum & M. Greaves (Eds.), Issues in *agent communication (pp. 178–192)* Heidelberg, Germany: Springer-Verlag.
5. Dignum, F. (1997). Social interactions of autonomous agents: Private and global views on communication. In *ModelAge Workshop* (pp. 103–122).
6. Dignum, F., Broersen, J., Dignum, V., & Meyer J.-J. (2004). Meeting the deadline: Why, when, and how. In *Third conference on formal aspects of agent-based systems (FAABS III) (April 26–27) Greenbelt, Maryland, USA.*
7. Dignum, F., & Weigand, H. (1995). Communication and deontic logic. In R. Wieringa & R. Feenstra, (Eds.), *Information systems, correctness and reusability, pp. (242–260). World Scientific. Singapore.*

8. Finin, T., Fritzson, R., McKay, D., & McEntire, R. (1994). A semantics approach for kqml – a general purpose communication language for software agents. In *third international conference on information and knowledge management (CIKM'94)*.
9. FIPA. Foundation for intelligent physical agents specifications. http://www.fipa.org.
10. Flores, R. A., & Kremer, R. C. (2002). A model for flexible composition of conversations: How a simple conversation got so complicated. In M. P. Huget, F. Dignum & J. L. Koning (Eds.), *Third workshop on agent communication languages and conversation policies, First international joint conference on autonomous agents and multiagent systems (AAMAS 2002)*. Bologna, Italy, July 15–19. 2002.
11. Fornara, N., & Colombetti, M. (2003) Defining interaction protocols using a commitment-based agent communication language. In *Second international joint conference on Autonomous agents and multiagent systems* (pp. 520–527). Melbourne, Australia. ACM Press.
12. Greaves, M. Holmback, H., & Bradshaw, J. (1999). What is a conversation policy? In F. Dignum & M. Greaves (Eds.), *Workshop on specifying and implementing conversation policies, Autonomous Agents (AA 1999)* (pp. 118–131). Heidelberg, Germany: Springer-Verlag.
13. Kagal, L. (2004). A policy-based approach to governing autonomous behavior in distributed environments. PhD Dissertation, September.
14. Kagal, L., Finin, T., & Joshi, A. (2003). A policy based approach to security for the semantic web. In *second international semantic web conference (ISWC2003)*. September 2003.
15. Kagal, L., Finin, T., & Joshi, A. (2003). A policy language for pervasive systems. In *IEEE international workshop on policies for distributed systems and networks*, Lake, Italy, June 2003.
16. Kifer, M., Lausen, G., & Wu, J. (1995). Logical foundations of object-oriented and frame-based languages. *Journal of ACM, 42*(4), 741–843.
17. Kollingbaum, M. J., & Norman, T. J. (2003). Norm consistency in practical reasoning agents. In *Proceedings of international workshop on programming multiagent systems*.
18. Labrou, Y., & Finin, T. (1994). A semantics approach for kqml – a general purpose communication language for software agents. In *Third international conference on information and knowledge management (CIKM'94)*, November 1994.
19. Lupu, E. C., & Sloman, M. (1996). Towards a role based framework for distributed systems management. *Journal of Networks and Systemss Management*. Plenum Press.
20. Lupu, E. C., & Sloman, M. (1999). Conflicts in policy-based distributed systems management. *IEEE transactions on software engineering*, *25*(6), 852–869.
21. Mallya, A. U., Yolum, P., & Singh, M. P. (2003). Resolving commitments among autonomous agents. In *Proceedings of International workshop on agent communication languages and conversation policies (ACL)*.
22. Moffett, J., & Sloman, M. (1993). Policy conflict analysis in distributed systems management. *Journal of Organizational Computing, 4*(1), 1–22.
23. Phillips, L. R., & Link, H. E. (2000). The role of conversation policy in carrying out agent conversations. In F. Dignum & M. Greaves, (Eds.), *Issues in agent communication*, volume 1916 of *Lecture Notes in Computer Science*. Springer.
24. Pitt, J., & Mamdani, A. (1999). A protocol-based semantics for an agent communication language. In *Proceedings of the international joint conf. on artificial intelligence IJCAI* (pp. 486–491).
25. Singh, M.P. (1991). Towards a formal theory of communication for multiagent systems. In *Proceedings of International Joint Conference of Artificial Intelligence* (*IJCAI'91*), Sydney, Australia, August.
26. Singh, M.P. (1993). A semantics for speech acts. *Annals of Mathematics and Artificial Intelligence, 8*, 47–71.
27. Smith, I. A., & Cohen, P. R. (1996). Toward a semantics for an agent communication language based on speech acts. In H. Shrobe & T. Senator (Eds.), *Proceedings of the thirteenth national conference on artificial intelligence*, (vol. 2, p. 24–31). Menlo Park, California: AAAI Press.
28. Smith, I. A., Cohen, P. R., Bradshaw, J. M., Greaves, M., & Holmback, H. (1998). Designing conversation policies using joint intention theory. In *Proceedings of third international conference on multi-agent systems (ICMAS98)*.
29. Venkatraman, M., & Singh, M. P. (1999). Verifying compliance with commitment protocols: Enabling open web-based multiagent systems. *Autonomous Agents and Multi-Agent Systems, 2*(3), 217–236.
30. W3C. Resource Description Framework. Resource Description Framework (RDF) Model and Syntax Specification, http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/, 1999.
31. W3C. Owl web ontology language. http://www.w3.org/2001/sw/WebOnt/, 2004.

32. Yang, G., & Kifer, M. (2000). FLORA: Implementing an efficient DOOD system using a tabling logic engine. In *Proceedings of Computational Logic — CL-2000*, number 1861 in Lecture Notes in Artificial Intelligence (pp. 1078–1093). Springer-verlag.
33. Yolum, P., & Singh, M. (2004). Reasoning about commitments in the event calculus: An approach for specifying and executing protocols. *Annals of Mathematics and AI, 42, (1–3).*
34. Zou, Y., Finin, T., & Chen H. (2004). *F-OWL: An inference engine for the semantic web*, vol. 3228 of *Lecture Notes in Computer Science*. Springer-verlag, November 2004. In Proceedings of the *third international workshop (FAABS)*, April 16–18, 2004, Greenbelt, MD, USA.