Information Sharing among Autonomous Agents in Referral Networks*

Yathiraj B. Udupi and Munindar P. Singh

Department of Computer Science
North Carolina State University
Raleigh, NC 27695-8206, USA
{ybudupi,singh}@ncsu.edu

Abstract. Referral networks are a kind of P2P system consisting of autonomous agents who seek and provide services, or refer other service providers. Key applications include service discovery and selection, and knowledge sharing. An agent seeking a service contacts other agents to discover suitable service providers. An agent who is contacted may autonomously ignore the request or respond by providing the desired service or giving a referral. This use of referrals is inspired by human interactions, where referrals are a key basis for judging the trustworthiness of a given service. The use of referrals differentiates such networks from traditional P2P information sharing systems, which are based on request flooding. Not only does the use of referrals enable an agent to control how its request is processed, it also provides an architectural basis for four kinds of interaction policies. *InterPol* is a language and framework supporting such policies.
InterPol provides an ability to specify requests with hard and soft constraints as well as a vocabulary of application-independent terms based on interaction concepts. Using these, InterPol enables agents to reveal private information and accept others' information based on subtle relationships. In this manner, InterPol goes beyond traditional referral and other P2P systems in supporting practical applications. InterPol has been implemented using a Datalog-based policy engine for each agent. It has been applied on scenarios from a (multinational) health care project. The contribution of this paper is in a general referrals-based architecture for information sharing among autonomous agents, which is shown to effectively capture a variety of privacy and trust requirements of autonomous users.

1 Introduction

In an open distributed system, (discovering and) selecting among service providers is a key challenge. Traditional peer-to-peer systems such as Gnutella and Kazaa focus on file sharing among peers. In traditional P2P systems, a peer begins a search by sending a request for a file to some of its peers, who either provide the requested file or, if a count is not exceeded, forward the request to other peers. When a peer provides the requested file, the file is propagated back to the request initiator. Traditional P2P systems have certain drawbacks. First, their free flooding mechanism can cause a large number of message transmissions and be inefficient in their use of bandwidth. Second, and more

* We thank National Science Foundation (grant ITR-0081742) for their partial support.

importantly, from the perspective of this paper, traditional approaches complicate trust and privacy management. A request that is forwarded by a peer Y on behalf of a peer X has the effect of being executed by the receiving peer Z as if the request originated with Y. In other words, Z may respond or not because the request came from Y, whereas any information Z provides would be viewed by X.

Referral systems are a less well-known but powerful kind of P2P system [1, 2]. Briefly, referral systems are multiagent systems whose member agents follow a (generally, but not necessarily) cooperative protocol by issuing referrals to one another, thus sharing their knowledge about service providers and enabling improved service selection. An agent seeking a service requests a set of *neighbors* (who can be thought as its favorite peers) for services. The requested agents autonomously decide on providing the service, a referral, or neither. The request initiator can autonomously decide whether to follow any of the referrals received. Traditional referral networks are difficult to engineer since they lack a declarative characterization of how the agents interact.

This paper describes *InterPol*, an implemented framework and a specification language for interaction policies in multiagent service networks. Policies capture requirements perspicuously and are used in many practical settings, such as for business or security. InterPol enables each agent to set its policies unilaterally. InterPol supports easy administration based on a flexible and yet practical approach for agents to decide with whom to interact and how. It provides an application-independent vocabulary geared toward interaction policies in service networks. InterPol's novel features include capturing social primitives to capture relationships among agents; an ability to model trust among agents; an ability to specify requests via hard and soft constraints; and, support for privacy-preserving information sharing among agents.

Our work is motivated by the needs of emerging P2P information systems. An important and natural class of such systems arise in health care information management. Our examples are inspired by those studied in the EU project Artemis [3], which is developing an approach to enable the sharing of health care information across organizational and sometimes national boundaries.

Health care is a natural fit for P2P service networks, especially one supporting rich interaction policies. For example, a patient may have as neighbors his primary care physician and his close friends, and would contact them to request services or referrals. A physician would have knowledge of the credentials of several specialists and would refer his patients to them. Social relationships apply naturally here. A patient would stop seeing a physician with whom his interactions were not effective. And he would form additional relationships based on his evolving needs. For example, someone who ends up with clogged arteries is likely to begin seeing a cardiologist on a regular basis.

Privacy is an important concern in health care and policies are natural for privacy management. For example, a specialist's policy might reveal the specialist's observations only to the patient's primary care physician or to another specialist.

Consider a scenario when a person from North Carolina falls sick on her visit to California. To find a good physician, she contacts her primary care physician back home, who returns a referral to a friend in California. As the patient is not aware of the quality of this newly referred physician, she would apply her requesting policies and verify that this physician has board certification from the ABMS, e.g., by checking on a suitable

web-site. The selected physician now requires the patient's medical records, for which the patient's primary care physician's answering policies kick in. InterPol was evaluated on the above kinds of scenarios. Agents request each other for names of physicians meeting various criteria. Here, an answer typically involves names of physicians, sometimes with additional information about them. And, a referral typically is to an agent who might be able to provide the names of some physicians meeting the specified criteria.

Contributions. To develop a policy-based approach for interactions requires that we construct a suitable conceptual model in which we can express the desired interactions. In essence, the conceptual model should support social knowledge cleanly separated from domain knowledge. This paper addresses this challenge, developing a conceptual model and vocabulary geared toward policy-driven multiagent systems, and implementing it using a logic programming engine.

Organization. Section 2 introduces the basic functioning of InterPol: its policies and representations of messages. Section 3 shows the application of policies and tacks and illustrates important scenarios considering trust, privacy, utility of interactions, and social relationships among agents. Section 4 offers a study of related work with a comparative evaluation of the present approach. Section 5 concludes with a discussion of contributions and future work.

2 InterPol Framework

The InterPol architecture consists of *agents*, representing *principals* who remain behind the scenes. The agents are heterogeneous and differ in their policies and needs. For simplicity, we assume they share a communication language.

2.1 Agent Interactions

As explained above, traditional P2P systems employ a request flooding mechanism where a request initiated from a peer is forwarded until the requested file is found. In practical settings of such networks, flooding is limited by specifying either a maximum depth of request path or a time-to-live (TTL) for each request. Consequently, not every request may result in a hit, either because of the non-availability of the requested resource, or because of the early death of the request. The originator must decide these limits ahead of time, which is nontrivial. If it decides to search a little deeper, it would have to repeat the search already completed by the network.

InterPol employs a multiagent referral architecture wherein agent interactions are based on the following mechanism. An agent seeking a service requests some agents from among its *neighbors*. A requested agent may ignore a request, perform the specified service, or give referrals to other agents. An *answer* is a response based on performing the requested service; a *referral* is a response consisting of names of other agents (or *referands*) who might provide the requested service.

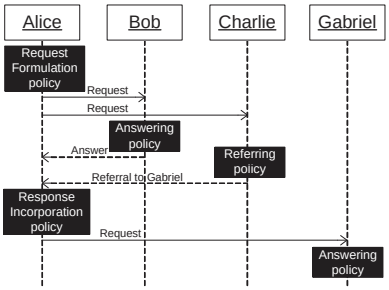

Fig. 1. Example referrals scenario

Figure 1 shows a simple scenario (ignore policies for now), where Alice queries Bob and Charlie for a service. Bob returns an answer, while Charlie refers Gabriel. Alice then queries Gabriel. This contrasts with request flooding in Gnutella, by making the querying agent directly responsible for how the computation proceeds.

InterPol goes beyond traditional referral approaches by providing a sophisticated means for specifying interaction policies among the participants. The following examples give a flavor of the kinds of policies that might be constructed. A user may specify that his personal information can be shared only with a physician P who has credentials from a local hospital to which the user has revealed personal information and if P is given a referral by the user's current primary care physician. A user may select a surgeon for an outpatient procedure based on referrals from friends as well as board certification in the specialty of interest. A user may not want to reveal any private information to any one but his friend. InterPol supports the following kinds of policies. It is important how inserting these policies leads to a much richer treatment of interactions than in traditional P2P systems.

InterPol supports four kinds of policies namely *request formulation (RF)*, *response incorporation (RI)*, *answering* and *referring* policies. An agent applies its *request formulation* policies to decide on what to request and whom to ask. An agent applies its *response incorporation* policies to evaluate the responses and decide on further action. An agent, when requested, applies its *answering* and *referring* policies to decide whether and how to provide an answer or a referral.

Figure 1 illustrates these policies. Alice applies its request formulation policy to decide on requesting Bob and Charlie. Bob checks with its answering policy before returning an answer. Charlie, not being able to answer, applies its referring policy and returns a referral to Gabriel. Alice now applies its response incorporation policy and accepts Bob's answer and Charlie's referral and forwards the request to Gabriel.

2.2 Enactment

We have implemented *InterPol* to demonstrate the effect that the above approach has on modeling and reasoning about the interactions among agents in a service network. Each agent is implemented around a reasoner (built using the tuProlog interpreter [4]) that handles policies and tacks. Each agent has a knowledge base (KB): storing domain knowledge related to the agent's domains of interest and expertise, social knowledge about neighbors, agent models, and social relationships, and privacy related knowledge. There is a policy base for the policies introduced earlier. Our agents follow the architecture typical in referral systems, e.g., [2]. The algorithms for requesting and responding are described below.

Algorithm 1: Ask-Request

1: **for** Each neighbor to ask based on RF policies **do**
2: Send request including a predicate and any constraints
3: **if** (response.type == referral) **then**
4: Send request to referred agents based on RI policies
5: **end if**
6: **end for**
7: **for** Each response that is an answer **do**
8: Evaluate and incorporate the answer based on RI policies
9: Update models of responding agents
10: **end for**

Algorithm 2: Respond-Request

1: **if** Answering policies allow **then**
2: Solve for the request predicate with its arguments
3: **return** answers after marking up the requested tacks
4: **end if**
5: **if** Neighbors match and referring policies allow **then**
6: **return** referrals
7: **end if**

Requests. Algorithm 1 implements the Ask-Request() method. An agent who is looking for a service finds the neighbors selected based on the RF policies. For each such neighbor selected according to the RF policy, a request for the service is created and it may include any constraints (hard or soft "tacks"). This request is sent to all the matching neighbors in step 2 and an answer is awaited. The response received can be a referral or an answer. RI policies evaluate the response received. If the received response is a referral and if the RI policies are satisfied, the query is forwarded to the referred agents, again using Ask-Request(); otherwise, answers are evaluated and incorporated in step 8. Finally, in step 9, the agent models of the responding agents are updated with an improved rating in the case of a good answer or a good referral, and with a decreased rating for a bad answer or a bad referral. This step is the essence of how referral systems evolve.

Responses. Algorithm 2 implements the method Respond-Request(), which is invoked when an agent receives a request. If the requested agent is willing to answer, the InterPol reasoner solves for the request predicates with its arguments in step 2. Valid answers generated by the reasoner are returned after marking up if they satisfy the requested tacks (if any) in step 3. In step 5 if the referring policies of the agent allow, it responds with referrals having its matching neighbors as referands in step 6.

2.3 Conceptual Model and Representation

InterPol incorporates a conceptual model for specifying the facts and policies of agents. Figure 2 illustrates a part of this conceptual model. The key concepts are explained below.

Facts and Policies. In InterPol an agent's knowledge base comprises sets of facts and rules. The knowledge base (KB) is dynamic: facts and rules may be continually added or retracted. InterPol uses *Constraint Datalog* [5] to express policies and facts. Policies are logic rules. Facts are special cases of rules whose right-hand sides are empty. A fact forms the head of a rule, and a set of facts appear in the body of a rule. Facts include

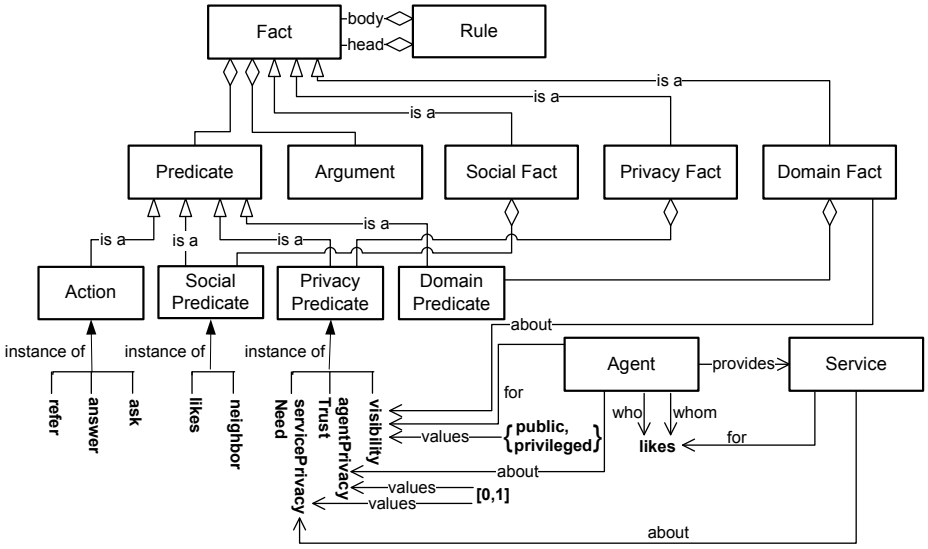

Fig. 2. Part of the conceptual model of the vocabulary

domain facts, social facts, and privacy facts. Variable names begin with an uppercase letter and constant names with a lowercase letter. A fact comprises a predicate and a set of arguments. Predicates include domain, social, privacy, and action predicates. Figure 2 shows example predicates in bold. The arguments of the facts may be constants or simple variables. A nested domain fact can appear as an argument in the case of the *visibility* predicate (illustrated in Listing 1.1). For example, Listing 1.1 shows facts and policies in Alice's KB. These indicate that: Dave is a physician specializing in cardiology, Alice likes Charlie for the *findPhysician* service, and a fact (illustrating the use of a nested predicate) that the first fact is public. Alice's referring policy allows her to refer any agent Y for a service P if she likes that agent.

Listing 1.1. Facts and policies in Alice's KB (part 1)

```
/* facts */
physician(dave, cardio).
likes(alice, charlie, findPhysician).
visibility(alice, physician(dave, cardio), public).
/* policies */
refer(alice, Y, P) :- likes(alice, Y, P).
```

Requests: Queries and Tacks. Let's first consider a simple form of a request, which consists of a *query rule* whose head is the predicate *ask* applied to some variables. The variables free in the head are used along with other variables in the body of the rule. $\mathit{ask}(X_i, \ldots) : -P_1(X_j, \ldots, l_i, \ldots), \ldots$ is a generic query, where the P_k are predicates, the X_i are variables, and the l_i are constants. Listing 1.2 shows a simple request consisting of a query.

Listing 1.2. Alice's simple request

```
[ask(X) :- physician(X, cardio), medicalSchool(X, duke), certifiedBy(X, abms),
           experience(X, Y), Y > 10]
```

To improve the effectiveness and efficiency of interactions, InterPol supports requests that consist of a query rule and a list of *tacks*. Each tack is a conjunction of one or more clauses. A tack having a predicate T_k with variables X_i, and so on is represented as $T_1(X_l, X_m, \ldots)$. In other words, a tack is syntactically like the body of a query rule. However, whereas a query body expresses a hard constraint, a tack expresses a preference of the requester. In simple terms, a request containing a tack can be interpreted as two requests: one consisting of the query rule alone, and another consisting of the query rule augmented with the tack "tacked on" to the body of the rule. When an agent responding to a request is able to accommodate a specified tack, it facilitates the requester pruning the search space and reducing the communication overhead.

Listing 1.3 shows a request sent out by Alice for a physician specializing in cardiology. She has preferences expressed in tacks such as about the physician's medical school, ABMS certification, and experience.

Listing 1.3. Alice's request with tacks

```
[ ask(X) :- physician(X, cardio), {medicalSchool(X, duke), certifiedBy(X, abms),
             experience(X, Y), Y > 10 } ]
```

Responses: Answers and Referrals. A response returned by an agent is either an answer or a referral. An *answer* is a set of solutions. For a simple request, each solution is a vector of bindings of the variables in the *ask* of the given query to constants that satisfy the query rule. A *referral* is a set of facts describing the agents referred. These facts are generated by the reasoner to find the matching referrals for the stated request.

Listing 1.4. Answers and referrals

```
/* Bob's answer (response to Alice) */
{ [ask(watson)] }
/* Charlie's referral (response to Alice) */
{ refer(charlie, gabriel, physician) }
```

Listing 1.4 shows Bob's answer (one cardiologist who matches the body of the query rule) and Charlie's referral (a singleton set) in response to Alice's request of Listing 1.2.

For a request with tacks, each solution has two parts: (1) a vector of bindings of the variables in the *ask* of the given query to constants that satisfy the query rule and (2) a list of *remarks* in the same order as the tacks in the given request. Each remark on a variable binding merely states whether the corresponding tack is true (*T*) or not (*F*) for that binding.

Listing 1.5. Answers and referrals

```
/* Bob's answer (response to Alice) */
{ [ask(watson), {T, T, T}],
  [ask(dave), {F, T, F}] }
```

Listing 1.5 shows Bob's answer to Alice's request of Listing 1.3. Bob's answer lists two physicians specializing in cardiology. Watson satisfies all the tacks, whereas Dave satisfies only the tack about certification.

In general, a request that places some clauses in the tacks instead of the query would produce more results, but some of them might be superfluous. A request that placed more clauses in the query rule would produce fewer, but more precise results. However, in some cases, it might produce no results at all.

3 Applying InterPol

InterPol incorporates a conceptual model and predicates for interactions, social relationships, trust evaluations, and privacy and utility management. The following form the major scenarios, which motivate the development of a rich vocabulary for the policy specification language in InterPol.

3.1 Accommodating Privacy

Policy-based approaches are natural for privacy. Traditionally, privacy is treated via access control policies, often based on credentials or roles. A multiagent approach can better model subtle social and organizational relationships among agents, which govern the agents' interactions in any practical setting. These models lead to policies that are more appropriate and acceptable. And, multiagent approaches provide an architecture where the resolution of the policies is carried out in a cooperative manner, wherein agents can naturally share information that might help others whom they trust.

For example, an agent may not want to reveal his medical records to anyone but his primary care physician. InterPol provides two low-level primitives for handling privacy. First, it allows a fact or a rule in the KB to be marked with its *visibility* (*public* or *privileged*). Second, InterPol supports a notion of privacy measures with respect to services and agents. These concepts enable formulating precise answering policies that restrict revealing private information to certain agents. InterPol models these concepts using the privacy predicates *visibility*, *servicePrivacyNeed*, and *agentPrivacyTrust* (values in the range $[0, 1]$) to specify the visibility and the privacy measures of a service and an agent, respectively. Here a privacy measure of 0 (1) means highly private (public).

To demonstrate elementary privacy, consider a scenario described in Listings 1.6 and 1.7, which are Alice and Bob's initial KBs, respectively.

Knowledge. Here, Alice is a neighbor of Bob and Alice has no neighbors. She has expertise in the domain of medicine, and an answering policy that expects the privacy trust measure of the requesting agent to be higher than that of the service privacy need. Alice's KB has a public fact that Dave is a physician specializing in cardiology. She has a domain policy that means that *physician* names and specialties can be revealed only if they are *public*. Other facts capture the *agentPrivacyTrust* of *Bob* and the *servicePrivacyNeed* of the predicate *physician*.

Listing 1.6. Initial KB of Alice (part 2)

```
answer(alice, X, P) :- agentPrivacyTrust(X, V1), servicePrivacyNeed(P, V2), V1 > V2.
visibility(alice, physician(dave, cardio), public).
physician(X, Field) :- visibility(alice, physician(X, Field), public).
agentPrivacyTrust(bob, 0.75).
servicePrivacyNeed(physician, 0.5).
```

Listing 1.7. Initial KB of Bob

```
neighbor(bob, alice).
query(bob, X, P) :- neighbor(bob, X).
hasDirectExperience(bob, X, P) :- likes(bob, X, P).
refer(bob, X, P) :- hasDirectExperience(bob, X, P).
```

Bob has a request formulation policy under which he can request any neighbor. Bob's referring policy requires him to have *direct experience* with a prospective referand. Bob's policy defines *hasDirectExperience* based on *likes*.

Interactions. Bob is looking for a physician specializing in cardiology and hence generates a query with body *physician*(*X, cardio*). He applies his request formulation policy by solving for *query*(*bob, Y, physician*). Alice qualifies for this policy, being a neighbor. Thus Bob sends the request to Alice. Now Alice's answering policy is satisfied and she returns the answer *physician*(*dave, cardio*) to Bob. Upon receiving the answer, Bob asserts the fact *likes*(*bob, alice, physician*) to indicate that Alice gave a good answer.

3.2 Strategies for Requests

InterPol provides tacks as a facility for expressing soft preferences. How tacks are constructed can have consequences on the efficiency of service selection and on the privacy of the agents involved.

Privacy preservation. An agent's requests can potentially reveal too much information, e.g., about the agent's true needs. A public request modifies a true, private request so as to hide some of the private information. To formulate privacy preserving queries, an agent must infer public requests from its private needs. There are two main ways of accomplishing this. In *generalization*, a weaker request is revealed. In Listing 1.8, a private request specifies a physician for skin allergy. However, the agent's request instead specifies a physician who treats any allergy.

Listing 1.8. Using the generalization approach

```
/* private need */
physician(X, skinAllergy).
/* public request */
physician(X, allergy).
```

In the *association* approach, a request that is a sibling of the actual (private) need is used. In Listing 1.9, the agent requests a dermatologist, based on the association between skin allergy and dermatology.

Listing 1.9. Using the association approach

```
/* private need */
physician(X, skinAllergy).
/* public request */
physician(X, dermatology).
```

Iterative exploration. For reasons of privacy, an agent may generate not one but a series of requests. For simplicity, let's consider that only tacks are varied across such requests. Successive requests may make the tacks weaker (less constraining) or stronger (more constraining). We can think of the tacks as forming a hierarchy, where lower tacks are stronger than upper tacks.

Listing 1.10. Example tack hierarchy

```
experience(X, Y), Y > 10
      ⇓
certifiedBy(X, abms)
      ⇓
medicalSchool(X, duke).
```

Listing 1.10 shows three tacks in order for a query predicate *physician*(X, *cardio*). The top tack allows a physician with at least 10 years of experience and is the weakest. The middle tack requires a certification by ABMS, whereas the bottom tack requires the physician to be from Duke. In the *bottom up* strategy, if a specified tack yields no valid answers, the agent weakens the tack in a subsequent request. This increases potential space of answers. In the *top down* strategy, the agent begins at the top and refines its tack until an acceptable answer is found.

Conflict management. Tacks can conflict. To accommodate handling conflicts between tacks, InterPol supports assigning priorities to them. For example, consider a scenario where a request for a physician is composed of two conflicting tacks, specifying that the physician should be from Harvard and Duke respectively. The tack with the higher priority is preferred.

3.3 Trust and Social Relationships

We model *trust* in relational terms: a *trustor* trusts a *trustee* with respect to a particular service. For example, we may trust a cardiologist for all heart-related problems but not for other ailments. Because of different bodies of evidence or different evaluations of the same evidence, two trustors can have different assessments of trust for a particular trustee. *Social trust* is based on the relationships among the agents and is well suited for P2P information systems.

InterPol supports social relationships such as *neighborhood*, *competition*, *collaboration*, *friendship*, *enmity*, and *service dependency*. These relationships lead to succinct policies that govern agent interactions well. For reasons of brevity, they are not presented here. Instead we describe an example of a generic means to evaluate relationships, which provides the heart of evidence-based reasoning. Social network analysis models trust in the presence of social relationships based on evaluating the participants' experiences [6]. The knowledge of these relationships at various strength levels can feature in an agent's policies to evaluate trust among agents. InterPol captures the strength I (values in the interval $[0, 1]$) of a relationship R via a measure *rStrength*(R, I).

4 Related Work

Policies are widely used for access control and trust management in distributed systems. InterPol differs from traditional policy approaches, because it focuses on a multiagent service network, and provides a set of primitives that are designed for expressing natural policies in it. These policies can be thought of as supporting subtle kinds of access control where each agent determines how much of its domain or social knowledge to share, when, and with whom.

Reputation-based access control. Reputation-based trust mechanisms are becoming common for the management of decentralized peer-to-peer networks because of the threat of malicious peers. Xiong and Liu propose an adaptive trust model using community-based reputations to predict the trustworthiness of peers in P2P e-commerce communities [7]. Boella et al. discuss authorization and permission in policies for virtual communities consisting of resource consumers and providers, and authorities [8].

Each community includes an authority, which keeps track of membership and fine-grained access control policies.

A common feature of current reputation and access control systems is that they employ centralized mechanisms to store reputation values or to provide fine-grained access control policies. By contrast, InterPol is decentralized and thus maximizes the agents' autonomy. Further, its use of policies simplifies the management of P2P systems by placing control in the hands of the individual peers.

Policy languages. Of the several policy specification languages, two are particularly important. *Rei* is a policy language implemented in Prolog for pervasive environments [9]. *PeerTrust* has an expressive policy and trust negotiation language based on first order Horn rules which form the basis for logic programs [10]. PeerTrust establishes trust using a dynamic exchange of certificates. Rei does not model the privacy preserving policies like in InterPol and PeerTrust. Like in PeerTrust, trust between entities in InterPol is built over time, but unlike the dynamic exchange of certificates in PeerTrust, trust in InterPol depends on the quality of the answers or referrals provided by the entities, and the trust models generated by the policy framework.

Role-based trust management. Role-based trust management languages emphasize the properties of roles such as their hierarchy. They specify role delegation, and support credential chain discovery and trust negotiation. Like InterPol, RT [11] and Cassandra [12] are based on Datalog with constraints. InterPol models deeper social relationships and considerations of privacy. Via tacks and policies, InterPol supports a more flexible kind of trust negotiation.

Privacy preserving systems. Several trust negotiation systems have introduced mechanisms to safeguard the privacy of the entities and their policies involved in a negotiation by using privacy preserving policies. PeerTrust [10] uses a protection scheme that uses named policies, so that policies can have their own policies. InterPol can support named policies, because it can support nested policies. Also, InterPol supports sophisticated privacy preserving mechanisms by supporting policies that use agent relationships to evaluate agent privacy levels.

5 Conclusion

Referral systems provide an alternative approach to realizing service networks than traditional P2P systems. They place control of the computation in the hands of the requesting agent (even as it relies upon cooperation from others), because it is involved in all interactions. Thus it can better control the information it reveals to other or the information it receives and incorporates from others.

The referrals approach supports four types of policies to be formulated for each agent. As a result, a far richer variety of interactions are supported than in traditional P2P systems. This richer variety of interactions is essential for the engineering and management of practical P2P information systems.

InterPol shows how its algorithms can be realized over a conventional Prolog engine. It provides a rich vocabulary to enable to proper expression of policies, and supports various heuristics by which agents can interact with each other. Future work will

consider enhancing the algorithms for evaluating policies to support better exchange of information among the agents to perform cooperative search. A referral system evolves as agents unilaterally can change their neighbor sets so that their "better" peers become their neighbors. Interesting properties emerge and are related to how individual agents act [2, 13]. It would be interesting to study such properties in the context of the policies discussed above.

References

1. Bonnell, R., Huhns, M., Stephens, L., Mukhopadhyay, U.: MINDS: Multiple intelligent node document servers. In: Proceedings of the 1st IEEE International Conference on Office Automation. (1984) 125–136
2. Singh, M.P., Yu, B., Venkatraman, M.: Community-based service location. Communications of the ACM **44**(4) (April 2001) 49–54
3. Dogac, A., Laleci, G., Kirbas, S., Kabak, Y., Sinir, S., Yildiz, A.: Deploying semantically enriched web services in the healthcare domain. Information Systems Journal (Elsevier Science) (2005)
4. Denti, E., Omicini, A., Ricci, A.: tuProlog: A light-weight Prolog for Internet applications and infrastructures. In: Proceedings of 3rd International Symposium on Practical Aspects of Declarative Languages. Volume 1990 of LNCS., Springer-Verlag (January 2001) 184–198
5. Li, N., Mitchell, J.C.: Datalog with constraints: A foundation for trust management languages. In: Proceedings of 5th International Symposium on Practical Aspects of Declarative Languages. (January 2003)
6. Sabater, J., Sierra, C.: Reputation and social network analysis in multi-agent systems. In: Proceedings of 1st International Joint Conference on Autonomous Agents and Multiagent Systems. (2002) 475–482
7. Xiong, L., Liu, L.: A reputation-based trust model for peer-to-peer ecommerce communities. In: Proceedings of IEEE Conference on E-Commerce (CEC). (June 2003)
8. Boella, G., van der Torre, L.: Permission and authorization in policies for virtual communities of agents. In: Proceedings of Third International Workshop on Agents and Peer-to-Peer Computing (AP2PC). (2004)
9. Kagal, L., Finin, T., Joshi, A.: A policy language for a pervasive computing environment. In: Proceedings of 4th International IEEE Workshop on Policies for Distributed Systems and Networks (POLICY). (June 2003) 63–74
10. Nejdl, W., Olmedilla, D., Winslett, M.: PeerTrust: Automated trust negotiation for peers on the semantic web. In: VLDB Workshop on Secure Data Management (SDM). Volume 3178 of LNCS., Springer-Verlag (August 2004) 118–132
11. Li, N., Mitchell, J.C.: RT: A role-based trust-management framework. In: Proceedings of 3rd DARPA Information Survivability Conference and Exposition (DISCEX), Washington (April 2003)
12. Becker, M.Y., Sewell, P.: Cassandra: Distributed access control policies with tunable expressiveness. In: Proceedings of 5th International IEEE Workshop on Policies for Distributed Systems and Networks (POLICY). (June 2004)
13. Yolum, P., Singh, M.P.: Engineering self-organizing referral networks for trustworthy service selection. IEEE Transactions on System, Man, and Cybernetics, Part A **35**(3) (May 2005) 396–407