Distributed Policy Resolution Through Negotiation in Ubiquitous Computing Environments

Venkatraman Ramakrishna
Computer Science Department
University of California, Los Angeles
Los Angeles, California
vrama@cs.ucla.edu

Peter Reiher
Computer Science Department
University of California, Los Angeles
Los Angeles, California
reiher@cs.ucla.edu

Leonard Kleinrock
Computer Science Department
University of California, Los Angeles
Los Angeles, California
lk@cs.ucla.edu

***Abstract*—Ensuring spontaneous ad hoc interoperation in decentralized ubiquitous computing environments is challenging, because of heterogeneous resources and divergent policies. Centralized cross-domain service access agreements can be made with a priori knowledge of the interacting entities' policies, but privacy concerns make this approach impractical. Environments should not be too rigid nor too open in their interactions, and should support varying contexts and scenarios. We describe the modeling, design, and implementation of a general purpose negotiation protocol for cross-domain service access agreements between entities that do not share trust agreements or application level protocols. This protocol resolves the constraints and needs of the participants, described in the form of declarative logical policies, in a fully distributed manner, avoiding the need for a third party. We describe how we tested the system and show how negotiation performance was evaluated against an optimal case computed by a centralized oracle.**

Keywords-interoperation; middleware; negotiation; policy

I. INTRODUCTION AND MOTIVATION

Spontaneous interoperation is one of the key unmet requirements of ubiquitous computing [17]. More thought is required to handle security and privacy, and to balance security with automation. Future environments are likely to consist of autonomous domains with defined security and administrative boundaries, accessing services across such boundaries to the extent that their security policies permit. A global centralized interoperation solution is neither scalable nor enforceable.

Though *de facto* standards such as TCP/IP and 802.11-based MAC protocols enable mobile devices to connect and obtain services for their users, true ubiquitous interoperation requires intelligent mechanisms in the application layer, such as the ability to communicate information about objects, pose queries, request action performance, offer services, and access external resources. These mechanisms should be highly automated, limiting or eliminating manual intervention. In today's systems, automation is possible if all the interaction constraints are known beforehand. Yet inventing a new mechanism for each possible combination of parameters that define a scenario is infeasible. The huge variety of interaction contexts, the large number of interacting partners of highly varying trustworthiness, and the heterogeneity of resources and services offered, would result in a combinatorial explosion of separate protocols for each scenario. Another roadblock is that the security and resource usage policies of interacting domains differ, and any service access agreement must be compliant with both sets of policies. Existing solutions for cross-domain interoperation in the face of divergent policies are insufficient. Ensuring proper static configuration for seamless service access in all possible contexts is impractical, but existing solutions for dynamic interactions are limiting. Some allow free service discovery and access, making them vulnerable to abuse; others have inflexible security models, preventing interactions that should be feasible. A more flexible solution would balance the needs of service access and security.

Most interactions follow a client-server model, but also involve give-and-take by both parties, blurring the service producer/consumer distinction. Consider a scenario where interoperation fails without prior configuration. An ACM-conducted conference is being held in a room with a wireless network, a display device, a projector and a printer. To access conference room services, an attendee would have to manually configure his device. Many attendees have ACM credentials which should ideally permit automated configuration. All that is needed is an effective procedure for the network to ask for proof of satisfactory accreditation, verify it and grant access; the user's device is then ready to provide private information in return for needed services after ensuring that the conference room can be trusted. The network could impose constraints on the requesting device, such as an audio-silent request during the conference, and grant service access only upon compliance. These tasks can be achieved through a step-by-step process of trading information and making agreement decisions.

Our goal is to help domains reach policy-compliant service access agreements. A centralized solution (using a third party) could achieve this task if it had complete knowledge of the domains' goals, state, and policy constraints. But this solution has a serious disadvantage: the participants must surrender their privacy to the third party by exposing their policies, opening themselves to potential abuse. Worse, one of the participants may take over the role of the third party, forcing the other participant to reveal its private information, thereby gaining an advantage. A privacy-preserving procedure would be better.

This paper describes a simple automated negotiation protocol that generates dynamic cross-domain service access agreements compliant with the negotiators' policies. We show how the protocol is generic and applicable to a wide range of ubicomp scenarios, being based on illocutionary speech acts

978-1-4244-3304-9/09/$25.00 ©2009 IEEE

[22]. The minimum domain-dependent variable necessary for application of our protocol is domain policy, describing state, invariants, and constraints, written in a declarative logical language. This protocol avoids the scalability issues of having a large number of scenario-dependent protocols. We demonstrate that the application of this protocol results in flexible agreements in a typical scenario. Much of the paper is devoted to the analysis of the protocol as a distributed, privacy-preserving policy resolution procedure. We show that this protocol generates one among a set of best results (except under certain conditions), and is equivalent to a centralized policy resolver. Though the protocol is sub-optimal, we show that it performs sufficiently well to be feasible in real-world scenarios. In most scenarios, the privacy and distributed nature of the solution outweigh the potential performance loss.

II. Negotiation Protocol and Panoply

Our model of ubiquitous computing consists of autonomous domains that offer services, have goals requiring access to other domains and their services, and have policies expressing their constraints. Each domain's policies are described in a declarative language built on logical semantics, with Prolog syntax [20]. This policy model is expressive, non-application-specific, and captures interdependencies between policies [21]. A logic-based policy language also lends itself to theoretical analysis of mechanisms built on top of it (Section IV). Some example statements in this language (with associated semantic meaning) are given in Table I. Each domain possesses a collection of facts and rules in a policy database. Individual statements could be related to each other through shared predicates. The database offers a consistent (no contradictory statements) view through a Prolog-based query API.

TABLE I. Policy Language Examples

1) *fileType('song.mp3',audio).* **['song.mp3' is an audio file]**
2) *certificate('UCLA').possess(john,'UCLA').* **['UCLA' is a certificate, and is possessed by 'john']**
3) *access(S,V) :- candidate(S), teamMember(S), voucher(location,V).* **[entity S can be granted access to voucher V if S is a 'candidate' and a team member, and if V is a 'location' voucher]**

Formally, each domain has a triple $<S, P, G>$, where S is its set of possessed services and resources, G is its set of goals (services it needs from external sources) and P is the set of policy rules that describe its current state, its possessions, invariants, and security and resource usage policies. Interaction between two domains $D_1<S_1, P_1, G_1>$ and $D_2<S_2, P_2, G_2>$ involves a negotiation resulting in D_1 getting access to a set of services $Q_1 \subseteq G_1 \cap S_2$ and D_2 getting access to a set of services $Q_2 \subseteq G_2 \cap S_1$. The remainder of this section describes the negotiation mechanism and algorithms, its implementation in a ubiquitous policy manager and a demonstrative application.

A. Negotiation Protocol

We designed a negotiation protocol that is generic in application, allowing the transaction of information, objects, commands and obligations. Such negotiation is a policy-guided operation where the parties satisfy each other's service access goals to the extent permissible within policy constraints.

TABLE II. Supported Speech Acts in the Negotiation Protocol

Speech Acts	Directive	Commissive	Assertive	Declarative
Message Type	REQUEST	OFFER	POLICY	TERMINATE

- Requests
  - Action *<Do A>*
  - Action *<Allow me to do A>*
  - Possession *<Give me P>*
  - State *<Let me change to state S>*
  - Question *<Tell me …>*
- Policies
  - Obligation *<Promise to abide by condition O>*
- Offers
  - Agreement *<Yes, I agree to do what you ask>*
  - Refusal *<No, I will not do what you ask>*
  - Rejection *<I cannot accept your offer>*
  - Answer *<Here is what you asked/inquired about>*

To make the protocol generic, its units (negotiation messages) and vocabulary are based on *illocutionary speech acts* [22]; a speech act expresses intent to perform an action. A negotiation is an illocutionary conversation, or a sequence of speech acts that begins with goals and end with declarations. Individual negotiation messages could be REQUEST, POLICY, or OFFER messages. Table II lists the speech acts that can be expressed through negotiation messages in our protocol. This fairly small set can express transactions in a large range of ubiquitous interoperation scenarios. In practice, requests in our framework are made for: {*possessions*, *actions*, *state changes*, *data/information queries*}. Almost all scenarios we envisioned consisted of goals that could be modeled as combinations of these request types. For example, a REQUEST for access to a printer could be framed logically as *'possess(Printer_ID)'*, and a command to close a network port as *'action(closePort, Port#)'*. OFFERs, which are replies to REQUESTs, are framed as logical *affirmatives* and *negatives*, with substantiating proofs (e.g., a printer access certificate) appended to affirmative messages. An example of a POLICY obligation is a prohibition of sound emission during certain hours. Combined with logical reasoning mechanisms, the negotiation protocol as a sequence of such speech acts provides a powerful mechanism for service access across domains.

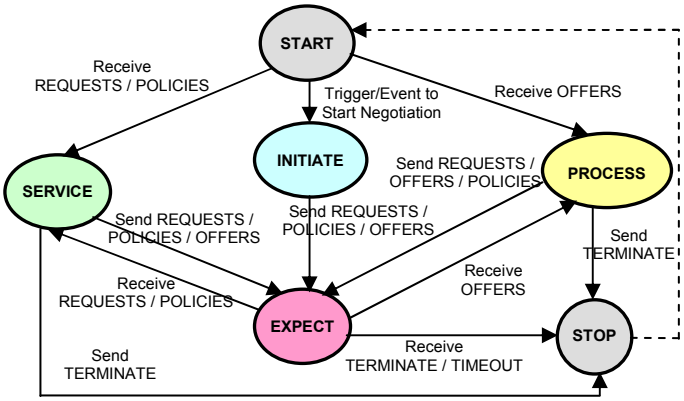

Figure 1. Negotiation Protocol State Machine (High-Level)

Our protocol state machine exchanges REQUEST (includes POLICY) and OFFER messages (Figure 1). Negotiation assumes that policies of a domain are private and not known to others. Without this assumption, our protocol could not be

applied to scenarios with stringent privacy concerns. Also, though one domain initiates negotiation by posing requests, the remainder of the protocol is peer-to-peer and bi-directional, as requests and offers may be sent by either side (Figure 1). We also assume that negotiators are aware of each other's presence and that they can establish a secure communication channel.

B. Protocol Engineering and Semantics

In a negotiation, entities express queries and intents to each other, the contents of each message depending on the prior message received and the current context. The protocol starts with one entity deriving requests from its goals and sending these requests to the other. In response, the other may send definitive offers (indicating acceptance or refusal) or counter-requests and policies. The protocol is driven by logical policy; i.e., if a received request R does not violate local policy, a negotiator (say N_1) immediately sends an affirmative offer. On the other hand, non-compliance of a request with local policy does not result in an outright refusal or rejection. All attempts are made by N_1, now in the *service* state (see Figure 1), to infer unsatisfied constraints that can be sent as counter-requests (see Section C). These counter-requests, if satisfied, would result in R being compliant with N_1's local policy. Many alternative counter-requests may be possible. If a counter-request is refused, an alternative counter-request is sent. An alternative may have less utility than the one attempted earlier (Section III.C.3). Attempts at different alternatives introduce flexibility in negotiation, but the protocol state machine by itself is agnostic of the relative importance of requests (as these may vary with scenario). Any concessions that N_1 is willing to make must be encoded in the form of logical policy statements, from which feasible counter-requests and alternatives that might lead to agreement are determined through a logical constraint-extraction algorithm (described in Section C).

Each negotiator maintains two lists, one of directives (requests and policies) posed to the other, and the second containing directives received from the other. Sending counter-requests grows the lists. A received request can be 'satisfied' by sending an affirmative or a negative offer (both of which are definitive, indicating whether the posed request can be obeyed in current circumstances), either of which results in removing the corresponding request from the posed requests list at the other end. If an acceptance OFFER is received, the policy database is suitably updated, reflecting changed state. Actions resulting from a negative OFFER could be: (i) trying an alternative counter-request, (ii) rollback (popping of requests from the posed request list), or (iii) a declaration of failure (termination). The protocol terminates when both request lists at both ends are empty.

C. Policy-Guided Reasoning Mechanisms

We describe the counter-request generation algorithm through an example. Let N_1 negotiate for access to a printer possessed by N_2. N_2 receives N_1's request in the form {*possess(F), printer(F)*} and converts it to {*possess(F), printer(F),* ***access(S,F)***} to check whether N_1 should be allowed access. Facts and rules relevant to N_2's counter-request generation are listed in Table III.

The policies governing access are selected; in this case, rule R1 is selected and parsed. This rule says that any domain may have access to a printer if it reveals its location, closes port 25 for security reasons, and possesses a valid voucher from a trusted domain (like *'ACM'* or *'UCLA'*). Conjuncts in the body of each such policy statement are evaluated; unsatisfiable predicates represent objects or actions that can be requested. {*location(S,L), closedPort(S,25), possess(S,V)*} are added to a queue, with extra support predicates that add semantic meaning to the request predicate (*voucher(V,M)* is a support predicate for *possess(S,V)* because it shares variable *V*). If an unsatisfied predicate in the body is not part of the shared global vocabulary (e.g.,*closedPort(S,25)*), the algorithm recursively examines all policies having that predicate as the head until *leaves* (or facts) are reached. Here, recursion results in the examination of rule R2, which returns the request predicate *action(closePort, 25)*. Finally, the following set of counter-requests is generated:

- **Set 1**: {*location(S,L); action(S,order,closePort,25); possess(S,V), voucher(V,'ACM')*}
- **Set 2**: {*location(S,L); action(S,order,closePort,25); possess(S,V), voucher(V,'UCLA')*}

TABLE III. RELEVANT PORTION OF N_2'S POLICY DATABASE

trustedDomain('ACM'), trustedDomain('UCLA'). (Facts)
access(S,F) :- *printer(F),* ***location(S,L), closedPort(S,25), possess(S,V),*** *voucher(V,M), trustedDomain(M).* (R1)
closedPort(S,P) *:- pred$_1$, pred$_2$, pred$_3$,* ***action(S, order, closePort, P).*** (R2)

This procedure generates multiple alternative sets of counter-requests, one of which is selected and returned in a reply message. The remaining sets are saved and tried only if the original set of counter-requests cannot be satisfied by the opposite party. If no counter-requests exist for a received request, an alternative offer generation procedure finds the closest matching satisfiable predicates to the one requested. Our protocol analysis (Section IV) and measurements (Section V) only consider the counter-request generation procedure. In Section III, we will see that our negotiation protocol, which runs this logical counter-request generation procedure, effectively performs distributed policy resolution. It could therefore be visualized in the form of a tree (see Figure 2). The root of the tree consists of the initial goals that negotiators begin with, and every request node has children in the form of offers or counter-requests. In some respects, the tree is structurally similar to a two-player game tree.

D. Negotiation Protocol State Maintenance

During a negotiation, acceptance and refusal of requests (OFFER messages) cause state changes, which are recorded in the local policy database of each negotiator. The decisions made in the remaining negotiation steps use the updated policy databases. Every negotiation step is determined on the basis of the immediate state of the policy database, which consists of a set of statements that are true in a first-order logical sense. Changes are made through logical assertions and retractions of statements. For example, if a request for access to a printer is granted, both the sender and recipient will record a logical predicate in their respective databases indicating that the recipient is in possession of a printer access certificate (or a voucher). When a new request is received, it is evaluated against the updated database. Thus each negotiator

'remembers' what has occurred thus far, or equivalently, what portion of the tree has been examined up to that instant.

E. System Design and Applications

The utility of our negotiation system is best demonstrated within a larger framework for managing policies within domains. Our implementation used the Panoply middleware [10][11] as a platform, with our notion of domains mapped to the *spheres of influence* concept. Negotiation is used in Panoply to mediate interactions between spheres (groups of devices characterized by location or social affiliations) and to enable context-sensitive access control through content filtering [20] [21]. A detailed description of Panoply or the policy manager design is beyond the scope of this paper.

A number of scenarios were designed and implemented, whereby two Panoply spheres could negotiate to reach agreements [21]. Table IV illustrates one example here: a ubiquitous conference room (C) and an attendee's device (D) negotiating to reach an agreement (see Section I). We omit mention of the policy rules due to space constraints.

TABLE IV. CONFERENCE ROOM NEGOTIATION INSTANCE

REQUEST (C→D)
<membership; printer access; display access>
→ REQUEST (D→C)
<Show me ACM accreditation (for membership and printer access); Show me ACM Official accreditation (for display access); Agree to turn off sound between 1700 and 1800 hours>
→ OFFER (C→D)
<No ACM accreditation; No ACM Official accreditation; Can I instead turn off sound between 0 to 1200 hours?>
→ OFFER<REJECT> (D→C)
<No, I will not accept your alternative sound turn-off offer>
→ OFFER (C→D)
<Can I instead turn off sound between 1720 and 2400 hours?>
[D accepts this offer]
→ REQUEST (D→C)
<Show me UCLA (ACM-affiliated institution) accreditation>
→ REQUEST (C→D)
<Show me valid NSF accreditation>
→ OFFER (D→C)
<NSF credential>
→ OFFER (C→D)
<UCLA credential>
→ OFFER (D→C)
<Membership granted; Printer access granted; Display access denied>
→ TERMINATE (C→D)

III. NEGOTIATION MODELING AND PROTOCOL PROPERTIES

The purpose of interaction between two domains is the satisfaction (partial or full) of their goals within the bounds of their collective policy constraints. This process requires the resolution of one domain's goals and its' policies against the other's goals and policies. We do not refer to the standard AI resolution procedure here. In ubiquitous interoperation, policy resolution is the process of determining whether, and to what extent, goals are satisfiable; this is done by evaluating logical consistency of the policy and goal sets of interacting domains.

A. Centralized vs. Distributed Policy Resolution

An oracle with complete information about the requirements and policies of the two negotiators can infer a set of feasible solutions. Such an oracle knows both $D_1<S_1, P_1, G_1>$ and $D_2<S_2, P_2, G_2>$. It can then combine the policy sets P_1 and P_2; i.e., it merges the compatible policies and disregards the contradictory policies for querying purposes. Then it formats the given goals (sets G_1 and G_2) into suitable logical predicates and runs a query through the combined database. Backward-chaining in Prolog is the basis of this query (and of our negotiation algorithms); the policy database is examined and all possible solutions are found. This procedure is known to be logically correct and exhaustive.

Negotiation generates similar results, but in a different manner. Since there is no oracle, both negotiators have partial knowledge, which is not good enough to generate a non-trivial agreement. Mapping from the oracle to the negotiating parties, we can imagine the combined database to be distributed among the two parties. Whereas the oracle could make recursive queries (as part of the search through the centralized database, a negotiator will find itself in situations where it cannot get full information from its database. These are the points where it infers unsatisfied constraints through the counter-request procedure, formats them and sends them to the other end; in effect, this is a remote query. The protocol is therefore identical to a policy resolution tree (see Figure 2), the difference being that nodes in the tree are distributed among the negotiators.

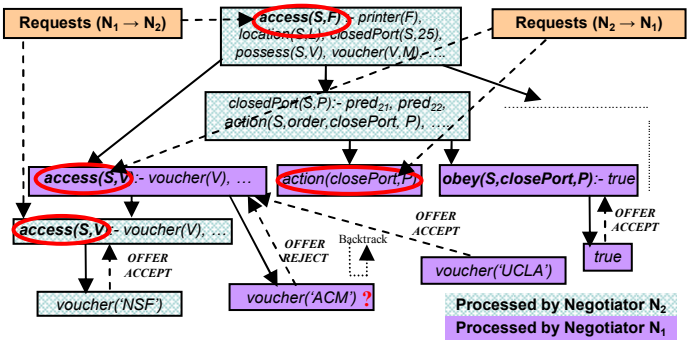

Figure 2. Negotiation Protocol Viewed as a Distributed Policy Resolution

Intermediate steps in a negotiation may result in failure. That is why alternative sets of counter-requests are generated and tried. In a policy resolution tree (see Figure 2), this is equivalent to backtracking. Structurally, this is an AND-OR tree. If negotiator N_1 sends a set of counter-requests in response to received request R, then the conjunction (AND) of the counter-requests represents the satisfaction of R. But if multiple alternative counter-requests are attempted, the satisfaction of R is represented by the disjunction (OR) of the conjunction of requests in each alternative set.

B. Properties of the Negotiation Protocol

To analyze the negotiation protocol, we make certain reasonable assumptions about policy statements and databases.

I) There are no cycles in the policy database; i.e., predicates are not self-referential either directly or through transitive links.

II) The policy database is of finite length; i.e., the number of statements (facts and rules/clauses) is bounded.

III) Each policy rule is of finite length; i.e., the number of predicates in the body of a clause is bounded.

IV) A policy rule is examined for the purpose of generating counter-requests at most once in a negotiation.

Properties derived from these assumptions and the algorithms used by the negotiation protocol are listed below. We use the fact that a reply MUST be sent whenever a non-TERMINATE message is received, requests MUST be popped out when an OFFER is sent and received and that a TERMINATE message is sent whenever the posed and received request lists get empty at both ends. Due to lack of space, we do not provide proofs of these assertions; detailed proofs can be obtained from [21].

- *Termination: the counter-request generation algorithm terminates, and both request lists become empty after a finite number of steps. Consequently, the negotiation protocol terminates within a finite number of steps*
- *The negotiation protocol is deadlock-free.*
- *The negotiation protocol is livelock-free.* This assertion holds only if property IV is enforced. Checking for duplicate policy rules is expensive in practice, so we omit that in our performance measurements. We anticipate that livelocks will occur rarely in practice.

C. Analysis of the Negotiation Protocol

These properties listed in Section B enable us to prove formal logical properties about the protocol, and to determine how *good* it is compared to the ideal. We evaluate the negotiation protocol based on metrics defined below.

1) Qualitative Metrics

- *Correctness*: A negotiation protocol is correct if the result, which is a mapping from the goal set to the level of satisfaction (either modal: *true*/*false*, or an alternative of lower utility compared to the original goal), is an improper subset of the oracular result, and is also consistent with the policies of the negotiators.
- *Completeness*: A negotiation protocol is complete if it always generates a result that is qualitatively comparable or identical to the oracular result.
- *Optimality*: A negotiation protocol is optimal if it always generates a result that is identical or comparable to the oracular result in the minimum number of steps (using the least number of messages).

By definition, a *complete* protocol will always be *correct* and an *optimal* negotiation protocol will always be *complete*. These metrics are related to each other through the inequality ***'Correctness < Completeness < Optimality'***. (The relation $A < B$ indicates that metric A is less stringent than metric B; i.e., a protocol that satisfies B must satisfy A).

2) Protocol Effectiveness and Exceptions

A negotiation result is consistent with, and does not violate, the collective policies of the negotiators. It is also guaranteed to terminate. Therefore, it is trivially correct. But the procedure, by inferring alternative sets of counter requests, also exhaustively examines the search space. If a solution exists, it will be eventually found. Multiple invalid alternatives may be examined before a satisfactory one is obtained, but if such an alternative exists, it is guaranteed to be examined and a correct result generated. By virtue of exhaustively examining the search tree, a negotiation will generate one among a set of *best* solutions, and the one encountered first is returned. Therefore, negotiation in such cases is both correct and complete. But it is not guaranteed to be theoretically optimal. Decentralized policy resolution, or negotiation, is a best-effort procedure, and the number of steps depends on the alternative selection heuristic. The best we can do is measure the statistical efficiency of the protocol, and we show the performance results in Section V.

Exception: Our above analysis considered policies only from a logical perspective. In practice, invocation of helper functions result in side effects (when requests are processed.) These could involve operating system calls and transfer of objects, which are non-logical. The ordering of alternatives and their selection impacts the final result in these cases, preventing the protocol from being correct in the non-trivial case (and by implication, complete). For example, an operation involving disk space allocation could result in both alternatives C_1 and C_2 failing when C_1 is tried before C_2, whereas if C_2 had been attempted first, it would have succeeded. Even here, if such non-logical actions are revocable, our protocol could keep track of such intermediate operations, and revoke them upon failure, thereby maintaining logical consistency and completeness. Unfortunately, not all possible actions are revocable, and our protocol would be correct though incomplete in those cases. In our performance analysis in the following section, we restrict our analysis to scenarios that do not involve non-logical operations, thereby ensuring the completeness of the protocol.

3) Other Protocol Aspects and Extensions

Heuristics and Strategies: Whereas many negotiation protocols are modeled on games or utilitarian economic transactions, ours is primarily a logical policy resolution scheme. Yet it leaves ample room for strategizing and incorporating utility-based heuristics. We have mentioned that our request types are drawn from the set {*possessions*, *actions*, *state requests*, *queries*}. Based on the degree of irrevocability and the risk involved in granting requests, we may conclude that query requests are safer to grant than action requests, which are safer to grant than possession requests (this is true in many, though not all, scenarios). Available alternatives could be ordered based on perceived risk, which is inversely related to utility. Utility could also guide negotiation strategy (the decision function involved in selecting the next step). In scenarios having real-time constraints, utility would vary inversely with the expected time to finish; multiple alternatives may be sent in a bunch to enable a quicker termination (though at a higher privacy cost). Other systems, including trust negotiation [24], have dealt with such issues. Discussing utility-based heuristics is beyond the scope of this paper, and we would just like to impress upon the reader that our protocol lends itself easily to strategizing.

Security: Though confidentiality is easy to achieve by ensuring that negotiation takes place over secure TLS sockets, avoiding denial-of-service (DoS) attacks is hard. As neither negotiator has prior knowledge of the other's policies, one party could execute a DoS attack on the other simply by sending an endless sequence of requests. Though such an attack would result in a waste of the victim's resources, it would hardly cause

catastrophic failures that a ping flood (for example) would cause, as messages are exchanged synchronously. In fact, all protocols of this nature are susceptible to such an attack. Yet, the design of our framework allows for mitigating solutions. The negotiation heuristic (logically separate from the protocol state machine) could be changed so that the utility of a goal/request decreases with the number of steps taken thus far, thereby steering the negotiation to a timely (albeit failed) end.

Multi-Party Collective Negotiation: Could our protocol be used as a basis for negotiation among multiple domains, the *k*th domain represented by $D_k<S_k, P_k, G_k>$? It could, though real scenarios would involve more variables than our model incorporates. A simple form of negotiation would proceed in lock-step, one negotiator allowed to send messages to some of the others in every round. Though offers are unicast to the original requesters, counter-requests may be communicated to multiple negotiators. Multiple offers could be received for the same request; one will be selected based on some criterion that our bilateral negotiation model does not incorporate. Designing such a protocol would be challenging, as would be ensuring its correctness and completeness. Our counter-request generation procedure would also have to be adapted to compute unsatisfied constraints for more than one recipient.

Examination of heuristics, avoidance of DoS attacks, and enabling multi-party negotiations are not the goals of this paper, though we do provide a sound basis for the development of these aspects. A primary aim was to show that our protocol could be feasibly used to resolve policies in a distributed manner; we will show this through our results in Section IV.

IV. Performance: Centralized vs. Distributed Policy Resolution

As the negotiation protocol is agnostic of the nature of the policy rules and the goals, the nature of the agreement is not a meaningful basis for measuring its success. Negotiation finds one among a number of equivalent goal assignments; i.e., if an oracle can find a way to satisfy a set of goals, a complete negotiation protocol must do so as well. A different but equally "good" agreement is acceptable within this definition. For example, in our ubiquitous conference room, any one among a set of printers may be offered, since the guest device simply requested a printing service. To a user, a color printer may be qualitatively superior to a black-and-white printer. But the negotiation protocol cannot distinguish between them unless the user explicitly expresses a preference in his goals. Therefore, the number of steps is the yardstick by which we measure negotiation efficiency. For every scenario, a theoretically optimal negotiation exists whereby negotiators can reach one among a number of qualitatively equivalent agreements in the least number of steps. The number of steps in an actual negotiation depends on the heuristic used to select an alternative counter-request at any step. In an optimal negotiation, the first alternative selected always succeeds, whereas multiple "false leads" are followed in real negotiations, resulting in increase in the number of steps.

A. Oracle: Centralized Policy Resolution

We designed and implemented an oracle that gains full knowledge of negotiators' policies by merging the two policy databases. Facts and rules in both databases are rewritten in terms of the negotiators' identities and asserted in a new database. E.g., if N_1 possesses resource R, this fact being asserted in its database, the new database contains the assertion "N_1 possesses resource R". Facts and rules now reflect globally consistent knowledge from the point of view of the oracle.

The oracle takes a goal as input and outputs the full policy derivation tree, the total processing time, and the minimum number of steps within which a negotiation could terminate. This minimum number of steps is linear in the depth of the tree. In an optimal tree, every node evaluation succeeds, every downward arrow maps to a REQUEST message, and every upward arrow maps to an OFFER message, and all requests and offers in a given level are bunched into a single message.

B. Performance Metrics

We compared the performance of the negotiation protocol against the oracle by three metrics: ***number of steps***, **s*ize (number of nodes) of the policy resolution tree***, and ***processing time***. A fourth metric (***fraction of the tree explored***) compared the number of alternatives examined with the total number generated. This metric indicates the extent of tree exploration, and the alternative selection heuristic efficacy.

C. Generation of Test Cases

We generated a large number of random test cases for measurement of these metrics. Each case consists of a pair of policy databases and an initial goal. Each database consists of a set of facts and rules. First we generated the databases and then obtained candidate goals by examining the heads of clauses.

One way of characterizing a database is by observing its size, which is the *total number of facts and rules*. Another characteristic is the total size of all policy rules. But these parameters, though relevant, may have little or no correlation with the actual performance of policy resolution (either centralized or decentralized) as reflected by our metrics. This is because not all facts and rules are relevant to a particular negotiation goal. Indeed, it is hard to characterize a policy database (or pair of databases given as inputs) quantitatively in an exact manner. Therefore we used the next best option, or the expected size of a derivation tree (the number of tree nodes) that results when a goal is resolved against the collective policies of the participants. We implicitly assume uniform processing time per node; as we do not use helper functions in these tests, this assumption is valid. To generate non-trivial database pairs, we set bounds on the maximum size of derived trees, which can be controlled using the following parameters.

- *Maximum Branching Factor (b_{max})*: This sets a bound on the number of immediate descendants (children of a node) in any tree generated from the database pair.
- *Maximum Depth (d_{max})*: This sets a bound on the distance from the root node to a leaf node in any tree generated from the database pair.

A database can only be characterized using bounds (or averages). Exact *b* and *d* values are characteristic only of a tree and not of a database. In a test case, a random request will result in the generation of a proof tree with branching factor at most b_{max} and depth at most d_{max}, with many requests resulting

in trees with lower breadth and depth. Each case was generated by specifying as parameters b_{max}, d_{max}, and the initial number of rules. The final database may contain more statements. Predicates and constants were drawn from our policy language ontology (describing possession, action, membership, information, objects, etc.). First, facts describing the state of each negotiator were generated. Rules were generated next, first with one predicate in the body, and then augmented in a loop to the extent that the branching factor and depth parameters permitted. The full procedure is described in [21].

D. Measurement Inputs

A test input consisted of the IDs of the negotiators, two policy databases, and an initial goal posed by one negotiator to another. These inputs were provided to both the oracle and the Panoply negotiation framework; *least cardinality of a counter-request set* was used as the alternative selection heuristic. We chose to have exactly one initial goal/request for simplicity, as our metrics can be measured using single-request negotiations. Though some multiple request negotiations will shed some light on the solution quality, the vast majority of negotiations will only tell us how processing overhead increases with more requests. Also the time taken to run our tests was quite large for single request negotiations, and the running time would have been prohibitively high for multiple request negotiations.

E. Performance Results

The experiments were conducted on an Intel P4 (2.53 GHz, 512MB RAM) desktop. We were interested in the performance of the negotiators and their processing times, which are independent of network latency; therefore we ran both negotiating spheres on the same computer, which yielded quicker results. Since each negotiator process waits for a response while the other is running, and these processing times don't overlap, we were able to record accurate readings. Test cases were generated for these parameter values: (**$1 \leq b_{max} \leq 10$, $1 \leq d_{max} \leq 20$,** and **initial number of rules = 28**). For each <b_{max}, d_{max}> tuple, **40** test cases (database pairs) were generated. Therefore, a total of 200*40 = **8000** database pairs were generated. The total number of policy statements in all these databases was equal to **988263**, or an average of **~62** statements per database. A variable number of initial requests were generated for each test case. A total of **194953** scenarios (<*database-pair*, *initial-request*>) were generated, or an average of ~**24** requests per database pair. To make sense of the results, we aggregated them based on certain parameters and ran statistical measurements, which are detailed below. All confidence intervals in the graphs are 99%. Every negotiation consists of an odd number of steps, since every request is matched by a corresponding offer, and a single termination message is sent by one negotiator at the end. Also, since the test cases were generated randomly, a large number of the negotiations resulted in failure.

1) Length of Optimal Negotiation (l_{min})

Failed negotiations are indicated in the graphs in this sub-section by l_{min} = -1. The confidence intervals for higher values of l_{min} are large because our test case generator produced few test cases at those values. As expected, the number of nodes (see Figure 3) seems to vary roughly exponentially with the number of optimal steps (a measure of tree depth). The tree generated by the oracle does not contain any nodes that lie along failed paths, and therefore contains many fewer nodes than the tree generated by the negotiation protocol. Figure 4 indicates actual processing times, and gives a better idea of the comparative performance. These times also vary exponentially, though negotiation does not perform much worse than the oracle, even for higher values of l_{min}. But the time taken per negotiation step is comparable to the corresponding time taken per unit step by an oracle (see Figure 5); at higher values of l_{min}, the negotiation time per step is actually lower, an encouraging result. This may be because an oracle has to examine a large number of tree paths that eventually fail, even for short negotiations; whereas the negotiation protocol runs standard logical procedures and manages lists, resulting in a proportional increase in processing time with additional steps. Therefore, developing better heuristics to decrease the number of negotiation steps could also end up improving system processing time significantly.

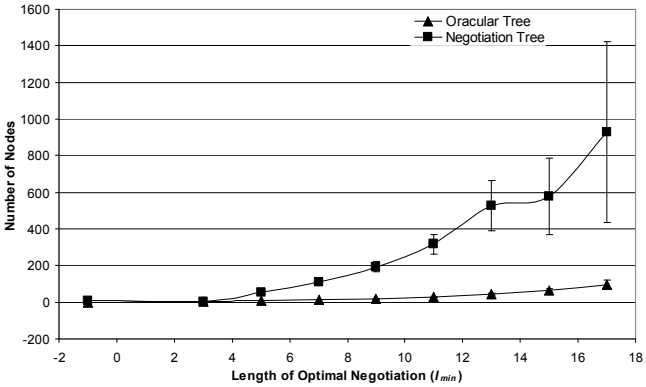

Figure 3. l_{min}: Number of Nodes in the Policy Resolution Tree

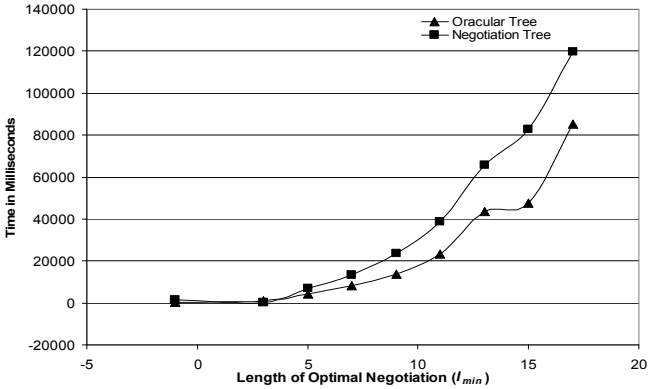

Figure 4. l_{min}: Total Processing Time for Policy Resolution

The most interesting result is the almost perfectly linear variation of the number of steps taken by an actual negotiation with l_{min} (see Figure 6). A linear regression on the mean curve yields an R^2 value of 0.99, and a slope of 2.11. The special case of l_{min} = -1 is also interesting. More than half of our negotiation scenarios (~55%) resulted in failure, so we obtained a very large number of data points. Still, both mean and median were very close to 3 (actually 3.74), which is the shortest negotiation we could hope for even if the original goal remains unsatisfied.

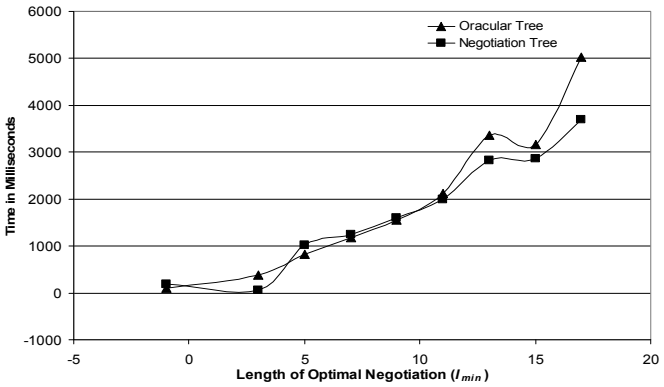

Figure 5. l_{min}: Average Processing Time for Policy Resolution per Negotiation Step

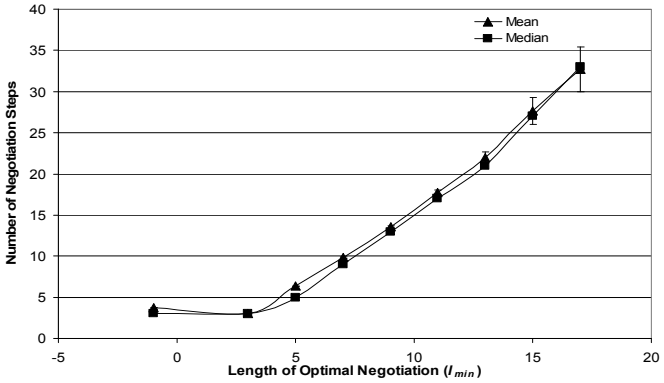

Figure 6. Comparison of Actual and Optimal Negotiation Steps

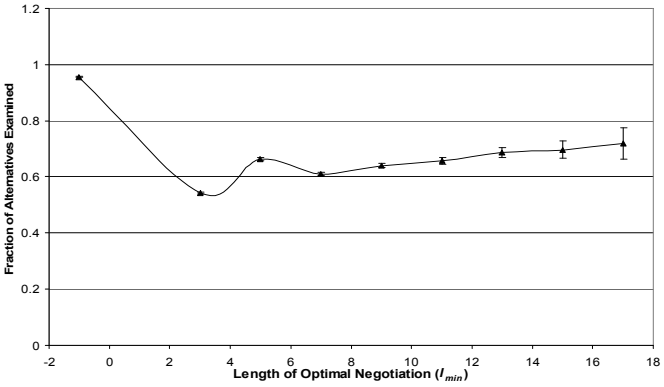

Figure 7. l_{min}: Average Fraction of Alternatives Examined

From Figure 7, we can see that the fraction of alternatives examined (indicating how much of the complete tree was explored) increases roughly linearly with increase in l_{min}. Fewer alternatives exist at lower values of l_{min}, and the correct alternative apparently gets examined earlier, leading to a smaller fraction value. Almost all fraction values are equal to 1 for failed negotiations, since the negotiation protocol engine attempts an exhaustive search. It is not exactly equal to -1 because alternatives at some intermediate steps may succeed; these successes are not sufficient for the overall negotiation (for the original goal) to succeed though.

2) Database Branching Factor Bound (b_{max})

Processing times are significantly affected by increase in the branching factor bound, and we can observe a sharp increase for $b_{max} > 6$ (see Figure 8); our conjecture is that this increase is polynomial rather than exponential, as tree sizes are polynomial in terms of their branching factors. But realistic scenarios ($b_{max} \leq 6$) have significantly lower processing times on average, for both oracular and distributed policy resolution.

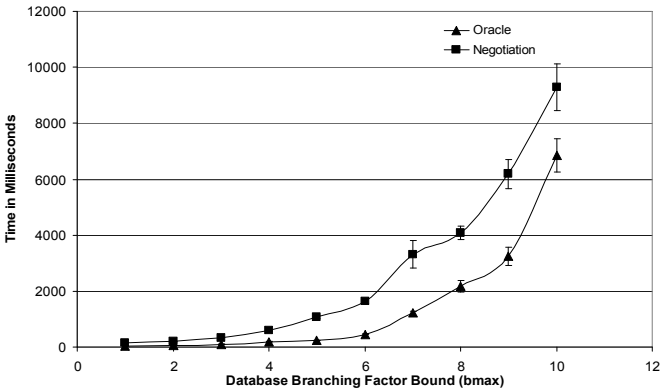

Figure 8. b_{max}: Processing Time

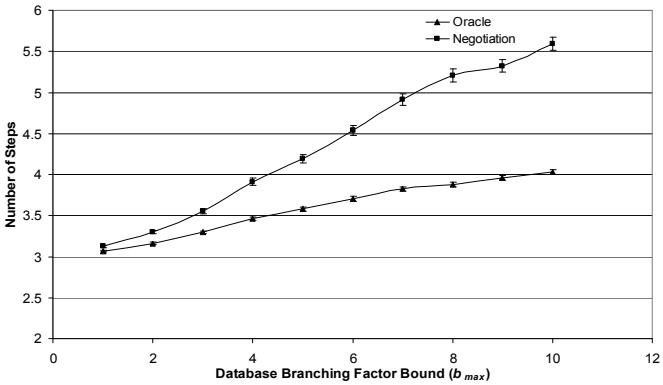

Figure 9. b_{max}: Average Number of Policy Resolution Steps

On the other hand, the number of steps in both kinds of policy resolution increases roughly linearly with b_{max} (Figure 9): ($R^2 = 0.98$, slope = 0.11) for optimal negotiations, and ($R^2 = 0.99$, slope = 0.3), for actual negotiations. Still, the actual number of negotiation steps does dominate the optimal number of steps and the curves diverge at higher values of b_{max}.

3) Database Depth Bound (d_{max})

The processing time for a test scenario increases roughly linearly for lower depth bound values (see Figure 10) and remains roughly constant for higher depth bound values (though with significant noise). This is probably because the parameter is an upper bound rather than a tight characteristic of the database pairs. As we increase d_{max}, a relatively low fraction of really deep trees are generated, resulting in the averages being almost constant. The depth parameter is therefore less meaningful than l_{min}, with which the processing time has an exponential relation. The numbers of steps vary in similar ways to processing times (compare Figures 11 and 10), being more pronounced at lower values of d_{max} (less than 10)

and flattening out at higher values. In the worst case, the number of steps taken for a negotiation appears to be 1.3 times the optimal (least) number of steps possible on average. Since the test cases at higher values of d_{max} probably involve a large number of shorter negotiations, only the parts of the curves for d_{max} <= 10 should be considered meaningful.

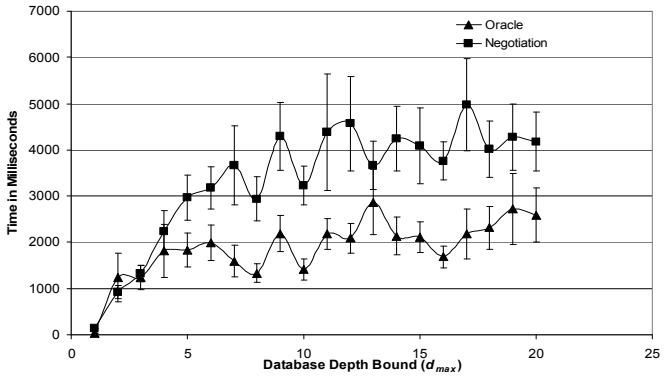

Figure 10. d_{max}: Processing Time

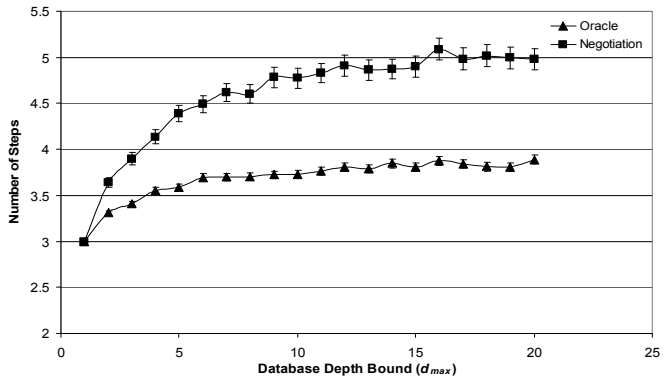

Figure 11. d_{max}: Average Number of Policy Resolution Steps

F. Conclusions

As we have seen, the length (number of steps) of an actual negotiation increases linearly with the length of an optimal negotiation for the same test scenario on average (implies scalability); failed negotiations are remarkably efficient, taking less than 4 steps on average. And though total processing times for negotiations dominate total oracular processing times, the picture is reversed when we consider the average time per step. Scenarios with realistic branching factor bound (b_{max}) values performed within the tolerable range, but increase in b_{max} significantly affects processing times. Based on our results, the branching factor bound appears to be a more significant performance indicator and bottleneck than the depth bound.

We also gain some insights into the way real ubicomp environments behave, and how they ought to be administered. It is feasible for small, self-governing domains to interoperate through negotiations. There is no vital need to delegate the responsibility of facilitating interactions to third parties, with the loss of privacy that centralized solutions entail. The organic growth of decentralized domains is not a threat to the ubiquitous computing vision that must be countered by standardization at all levels. In fact, ad hoc negotiation for external services is quite fast when the domains are small and possess relatively few resources and policies that are not very complex. Users' mobile device collections, coffee shops, homes and offices, which comprise a large majority of domains, will benefit from technology like our protocol.

V. Related Work

Automated trust negotiation [24], through which entities can establish trust on the web, is closely related to our research. It is a flexible way of doing access control, where entities can control what private information is released at fine granularities, though it is only a special case of the general kind of negotiation that we achieve. Protune (Provisional Trust Negotiation) [4], which builds on older trust negotiation frameworks like PeerTrust [13], is the closest working system to our own. Both the language, based on declarative logic program-based policy rules, and the protocol, consisting of requests and counter-requests, are similar to ours in many ways. But our negotiation protocol supports a wider range of information and action requests. It also handles multiple goals, supports alternatives, and determines compromises through alternate offer generation. Protune does not provide these features yet, though it could conceivably be extended to support these functions. We also have extensive practical test results comparing centralized and distributed policy resolution. Protune has not been modeled as a distributed policy resolution nor have its designers conducted such tests.

Negotiation and service-level agreement protocols have been developed for a number of areas, though they fall short of our goals in many respects. The WS-Agreement [1] standard enables universal negotiation for web services, and has the advantage of being based on the widely adopted SOAP and XML standards. On the other hand, it ignores the actual negotiation mechanism and protocol engineering. Dang and Huhns have designed a protocol for concurrent negotiations among multiple service providers and consumers [9]. Their protocol is based on utility functions that must be reconciled, whereas our framework is based on logical security and resource usage policies. A negotiation protocol proposed by Andreoli and Castellani views negotiation as a distributed proof tree [2], but uses a centralized coordinator. Interest-based negotiation [19] facilitates collaboration to achieve goals, and is application-independent, but does not consider policies to be private. Neither is the procedure modeled as a distributed policy resolution, nor have its theoretical properties been studied as ours has. Other speech acts-based negotiation protocols have been developed [5], but they do not leverage negotiators' policies, nor do they consider security and privacy.

Using policies as a flexible way of controlling systems and managing security is not new [23]. Languages like Rei [16] and ASL [15] have adopted a logic-based approach, thereby being applicable to distributed policy resolution. Our language was inspired by Rei, which is targeted towards ubicomp and the Semantic Web. It is based on logical domain-independent semantics, and supports specification of actions, speech acts, and modality. Other significant general purpose policy languages are Ponder [8] and PSPL [3], which have restrictive semantics. IBM's TPL [14] and WS-Policy [6] (a widely accepted standard), are based on XML, a non-logical language.

Our negotiation framework can also be viewed as a process of finding feasible ways of providing access to services. Advanced role-based access control systems, such as GRBAC [7] and dRBAC [12] provide more expressive and flexible policy-based access control than traditional ACLs and capabilities, but do not consider disagreements and private policies, where negotiation would be required. Minami's and Kotz's secure context sensitive authorization, [18] is much closer to our distributed policy resolution model. Access control in their framework results in a distributed proof tree spanning multiple entities possessing logically-framed policies. On the other hand, their system provides no scope for negotiation, and they assume that policies are public, which facilitates selection of suitable hosts for building the distributed proof tree. Yet we have gained valuable conceptual insights from these models, and obtained pointers for protocol performance evaluation from Minami's and Kotz's research.

VI. Conclusion and Future work

Planned interoperation does not scale, and one cannot anticipate every possible eventuality and put in place a suitable, efficient mechanism to deal with it. We have demonstrated how spontaneous decentralized interactions can be enabled through a generic policy-guided negotiation protocol running in the application layer. Negotiation is a form of distributed policy resolution when the policies are specified using first-order logical semantics. Our protocol terminates as long as the policy databases are bounded and cycle-free. It is also provably correct and complete as long as non-logical external functions do not irrevocable modify database state. Though sub-optimal, negotiation times fall within the range of what users will tolerate in a mobile or ubiquitous computing scenario. The average negotiation length varies linearly with the optimal negotiation length, and failed negotiations are short, which indicates that our protocol is scalable. Database branching factor bounds affect performance much more than depth bounds, indicating that scenarios where individual policy rules are not very complex could feasibly use our protocol.

There is ample scope to extend (and measure) the protocol with different heuristics incorporating semantic information associated with the request predicates. Using set cardinality as the selection heuristic works fairly well on average simply because the probability of satisfaction of a set of k requests decreases with increase in k. Instead, alternatives could be ordered using utility functions that take the benefit and risk/cost associated with requests into account. Cost metrics could be determined using measures of privacy loss, trust level, and even the expected time to termination. We could experiment with game-theoretic strategies, and examine ways of thwarting possible denial of service attacks on the protocol. We could also try to model multi-party negotiation as a distributed policy resolution, where each party has goals that could be satisfied by a combination of other parties. Devising a theoretically complete protocol that also handles failure and synchronization issues would be a significant challenge.

Acknowledgment

This research would not have been possible without Dr. Kevin Eustice's research, which yielded the *spheres of influence* concept and the Panoply middleware [11].

References

[1] A. Andrieux, et al., "Web Services Agreement Specification (WS-Agreement)," http://www.ogf.org/documents/GFD.107.pdf, May 2007.

[2] J. M. Andreoli and S. Castellani, "Towards a flexible middleware negotiation facility for distributed components," DEXA Workshop on E-Negotiation, Munich, Germany, 2001.

[3] P. Bonatti and P. Samarati, "Regulating service access and information release on the web," CCS 2000, Athens, November 2000.

[4] P. A. Bonatti and D. Olmedilla, "Driving and monitoring provisional trust negotiation with metapolicies," 6th IEEE POLICY Workshop (POLICY 2005), pp. 14—23, Stockholm, Sweden, June 2005.

[5] M. K. Chang and C. C. Woo, "A speech-act-based negotiation protocol: design, implementation, and test use," ACM Transactions on Information Systems (TOIS), vol. 12, Issue 4, pp. 360—382, Oct. 1994.

[6] S. Bajaj et. al., "Web Services Policy 1.2 – Framework (WS-Policy)," W3C Member Submission, http://www.w3.org/Submission/WS-Policy/, 25th April 2006.

[7] M. J. Covington, M. J. Moyer, and M. Ahamad, "Generalized role-based access control for securing future applications," 23rd National Information Systems Security Conference, Baltimore, MD, Oct. 2000.

[8] N. Damianou, N. Dulay, E. Lupu, and M. Sloman, "The Ponder policy specification language," POLICY 2001, Bristol, U.K., January 2001.

[9] J. Dang and M. N. Huhns, "Concurrent multiple-issue negotiation for internet-based services," IEEE Internet Computing, vol. 10, no. 6, pp. 42—49, November/December 2006.

[10] K. Eustice et al., "Enabling secure ubiquitous interactions," 1st Intl. Workshop on Middleware for Pervasive and Ad-Hoc Computing (at Middleware 2003), Rio de Janeiro, Brazil, 17 June 2003.

[11] K. F. Eustice, "Panoply: active middleware for managing ubiquitous computing interactions," PhD Thesis, Computer Science Department, UCLA, April 2008.

[12] E. Freudenthal, T. Pesin, L. Port, E. Keenan, and V. Karamcheti, "dRBAC: distributed role-based access control for dynamic coalition environments," ICDCS 2002, IEEE Computer Society, July 2002.

[13] R. Gavriloaie, W. Nejdl, D. Olmedilla, K. Seamons, and M. Winslett, "No registration needed: how to use declarative policies and negotiation to access sensitive resources on the Semantic Web," 1st First European Semantic Web Symposium, Heraklion, Greece, May 2004.

[14] A. Herzberg, Y. Mass, L. Mihaeli, D. Naor, and Y. Ravid, "Access control meets Public Key Infrastructure, or: assigning roles to strangers," IEEE Symposium on Security and Privacy, pp. 2—14, 2000.

[15] S. Jajodia, P. Samarati, and V. S. Subrahmanian, "A logical language for expressing authorizations," IEEE Symposium on Security and Privacy, May 4-7, 1997.

[16] L. Kagal, T. Finin and A. Joshi, "A policy language for a pervasive computing environment," 4th IEEE POLICY Workshop (POLICY 2003).

[17] T. Kindberg and A. Fox, "System software for ubiquitous computing," IEEE Pervasive Computing, vol. 1, no. 1, pp. 70—81, Jan.-Mar. 2002.

[18] K. Minami and D. Kotz, "Scalability in a secure distributed proof system," 4th Intl. Conference on Pervasive Computing, May, 2006.

[19] P. Pasquier, L. Sonenberg, I. Rahwan, F. Dignum, and R. Hollands, "An empirical study of interest-based negotiation," 9th Intl. Conf. on Electronic Commerce (ICEC), Minn., MN, pp. 339—348, Aug. 2007.

[20] V. Ramakrishna, K. Eustice, and P. Reiher, "Negotiating agreements using policies in ubiquitous computing scenarios," IEEE SOCA 2007, Newport Beach, California, June 19-20, 2007.

[21] V. Ramakrishna, "Policy Management and Interoperation Through Negotiation in Ubiquitous Computing," PhD Thesis, Computer Science Department, UCLA, September 2008.

[22] J. R. Searle and D. Vanderveken, "Foundations of Illocutionary Logic," Cambridge University Press, Cambridge, UK, 1984.

[23] M. Sloman and E. Lupu, "Security and management policy specification," IEEE Network, Special Issue on Policy-Based Networking, (invited) 16(2), Mar. 2002.

[24] M. Winslett, "An introduction to trust negotiation," 1st International Conference on Trust Management, Crete, Greece, May 2003.