2007 IEEE/WIC/ACM International Conference on Intelligent Agent Technology

A Flexible Policy-Driven Trust Negotiation Model

Juri L. De Coi, Daniel Olmedilla
L3S Research Center & Hannover University
Hannover, Germany
{decoi, olmedilla}@L3S.de

Abstract

*Policy-driven negotiations are gaining interest among the research community. A large number of policy languages with different expressiveness have been developed in order to suit different scenarios. This paper summarizes the general requirements a negotiation framework must cover and presents a flexible negotiation model that addresses all these requirements and subsumes existing models to date. An instantiation of this model and an architecture with reusable components that integrates two existing trust negotiation languages (*PEERTRUST *and* PROTUNE*) are provided.*

1 Introduction

During the last decade, the amount of users with Internet access has dramatically grown all over the world. While previously the set of potential users accessing a system was mainly restricted to those already known, now any two strangers should be able to communicate and perform transactions with such systems. Traditional authorization mechanisms relies on the fact that client identities can be mapped to a set of permissions. This authorization process is not applicable anymore since clients may not be known in advance.

A new authorization scheme called Trust Negotiation [7] has emerged and is gaining interest among the research community. It allows two strangers to bilaterally establish trust in an incremental process. This process is driven by statements that specify what is released and under which conditions. These statements are generally called policies and many languages have been developed to date [3, 2, 1, 5] in order to represent them. These languages differ on semantics and expressiveness since they were developed in order to suit different scenarios.

In this paper, we describe the requirements a general negotiation model must address and present and describe in detail a flexible negotiation model which addresses those requirements and subsumes existing models to date.

This paper is organized as follows: §2 highlights the main concepts that should be held in mind when designing a trust negotiation framework. A review of how these concepts are addressed by current state of the art is provided in §3. §4 describes our negotiation model whereas §5 describes the negotiation algorithm which instantiates the model presented as well as the system architecture in which it is implemented. Finally in §6 we conclude and outline some future work.

2 Negotiation Requirements

In this section we outline the main requirements a negotiation framework should take into account: some of them were already described in previous literature [2, 5, 3, 4], others are clearly stated here for the first time.

Negotiation The ground requirement a client-server transaction should fulfill is obviously the possibility of having not just one-shot interactions between actors but bilateral negotiations.

Actors Each negotiation implies two actors (e.g., Alice and some on-line bookshop).

External actions During a negotiation each actor may ask the other one for carrying out some actions (e.g., delivering a book or registering at a web site).

Notifications Each actor needs to be notified about whether the other actor performed the actions it was asked for. A notification may either need to be proved, so that the other actor can verify it (e.g., through a signed statement from the bank stating that a money transfer has been made), or not (e.g., provision of a delivery address).

Local actions During a negotiation each actor may need to carry out some actions which are not explicitly requested by the other actor. For example, it may be needed to access legacy systems (e.g., log a message into a file or querying a database of clients).

Action Selection Function In order to satisfy the negotiation's overall goal, an actor may be requested for following (at least) one out of n paths (e.g., in order to finalize a purchase, an on-line bookshop may request Alice either to subscribe a new account or to provide a credit card

0-7695-3027-3/07 $25.00 © 2007 IEEE
DOI 10.1109/IAT.2007.21

number). It is typically the case that the actor does not want to perform all actions, but only a subset of them. In order to make this selection automatically, an Action Selection Function is required.

Policy In general, an external action (e.g., the disclosure of a credential) is performed only under certain circumstances. Therefore a means is required to specify under which conditions an action can be executed (e.g., "credit card number can be provided only to on-line shops belonging to the Better Business Bureau"). Typically such a means is a policy language.

Policy filtering Actors need to tell each other the conditions under which a requested action can be performed. However typically a whole policy does not need to be released, since some information contained in it may be irrelevant for the other actor (e.g., local actions to be performed) or sensitive (e.g., giving away that the bookshop delegates a decision to a third company may be private).

Termination Algorithm A Termination Algorithm can be seen as a means to ensure that a negotiation eventually terminates (and guarantee that it does not get looped).

Explanation An actor should be able to ask and get information about the ongoing negotiation (e.g. how to buy a book at the bookshop) or finished ones (e.g., why the negotiation failed).

3 Related work

Concepts like *negotiation*, *policy* and *actor* are grassroots ones in the field of Trust Negotiation, and hence are mentioned in most of the literature about this topic (cf. [2, 5, 3, 4]). In all these approaches credential exchange is the only addressed external action, hence a need for *notifications* does not arise (credential reception is a particular notification on its own). Moreover local actions, when foreseen, usually limit to checking whether a credential is still valid. Similarly a distinction between *Action Selection Function* and *Termination Algorithm* is usually not made, being both concepts grouped under the common label of *Negotiation Strategy* [8]. All papers on trust negotiation assume that policies may be sensible (otherwise negotiations would not be needed) but they differ in the protection mechanisms they offer. [2, 5, 3, 4] suggest to set policies to protect policies as a whole, but fine-grained filtering of a policy, allowing to select just some parts of it to be disclosed to the other actor, is not foreseen.

4 Negotiation model

This section describes a general negotiation model which addresses all the requirements presented in §2.

Let A_1 and A_2 be the two actors involved in the negotiation (see figure 1). In the following we will assume that A_1 is the initial requester (e.g., Alice) whereas A_2 is the provider (e.g., the bookshop), i.e. A_1 is assumed to send a request to A_2 thus starting the negotiation.

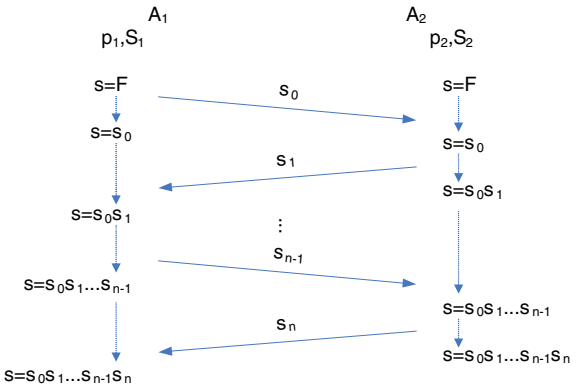

Figure 1. Sequence Diagram of a Negotiation

As shown above, a means (a language) is required to specify under which conditions the request of the other peer can be fulfilled. Let our policy language be based on normal logic program rules of the form $A \leftarrow L_1, \ldots L_n$. where A is a standard logical atom (called the *head* of the rule) and $L_1, \ldots L_n$ (the *body* of the rule) are literals, i.e. L_i equals either B_i or $\neg B_i$, for some logical atom B_i.

Definition 1 (Policy) *A Policy is a set of rules, such that negation is not applied to any predicate occurring in a rule's head nor to atoms representing the execution of external actions.*

This restriction ensures that policies are *monotonic* in the sense of [6], i.e. as more external actions are executed, the set of permissions does not decrease.

Definition 2 (Negotiation Message) *A Negotiation Message is an ordered pair (p, N) where p is a policy and N a set of notifications. We will denote with M the set of all possible Negotiation Messages.*

Definition 3 (Negotiation History) *Let A_1 and A_2 be the actors involved in a negotiation. Let A_1 be the initial requester, i.e. the sender of the first message in the negotiation. A Negotiation History σ for the actor A_j ($j = 1, 2$) is a list of Negotiation Messages $\sigma_1, \ldots \sigma_n \mid \sigma_i \in M$. We will denote with σ_i the i-th element of σ. Moreover let*

- $M_{snt}(\sigma) = \{\sigma_i \mid i = 2k - (j \ mod 2), 1 \leq k \leq \lfloor n/2 \rfloor\}$
- $M_{rcv}(\sigma) = \{\sigma_i \mid i = 2k - 1 + (j \ mod 2), 1 \leq k \leq \lfloor n/2 \rfloor\}$

denote the sequence of messages sent and received respectively. A Negotiation History may also be referred to as Negotiation State.

Intuitively, a message sent by an actor provokes the same message to be received by the other actor. In addition, messages between actors are sent alternately, i.e. a message sent by one actor is followed by another message sent by the other actor. Notice that according to this definition, the Negotiation History σ is shared by the two actors A_1 and A_2, but the sets $M_{snt}(\sigma)$ and $M_{rcv}(\sigma)$ are swapped between them. Therefore it holds that $M_{snt}^{A_1}(\sigma) = M_{rcv}^{A_2}(\sigma)$ and $M_{rcv}^{A_1}(\sigma) = M_{snt}^{A_2}(\sigma)$

Definition 4 (Negotiation State Machine) *A Negotiation State Machine is a tuple (S, s_0, Σ, t) such that*

- $S \equiv$ *a set of Negotiation States*
- $s_0 \equiv$ *the empty list (initial state)*
- $\Sigma \equiv$ *a set of Negotiation Messages.*
- $t \equiv$ *a function $S \times \Sigma \rightarrow S$ such that given $s = (\sigma_1, \ldots \sigma_n)$ then $t(s, \sigma) = (\sigma_1, \ldots \sigma_n, \sigma)$ (transition function)*

Intuitively a Negotiation State Machine models how an actor evolves during the negotiation by exchanging messages. Σ contains both sent and received Negotiation Messages and can therefore be partitioned into two subsets Σ_{snd} and Σ_{rcv}.

Definition 5 (Negotiation Model) *A Negotiation Model is a tuple (A, P, p_0, NSM, ff, ns) where*

- $A \equiv$ *the set of possible external actions*
- $P \equiv$ *the set of possible Filtered Policies*
- $p_0 \equiv$ *the actor's local Policy*
- $NSM \equiv$ *a Negotiation State Machine (Σ, S, s_0, t)*
  - s_0 *represents the initial state, i.e. the state in which the actor is at the beginning of the negotiation*
- $ff \equiv$ *a function $S \rightarrow P$ (Filtering Function)*
- $ns \equiv$ *an ordered pair (asf, ta), a.k.a. Negotiation Strategy, where*
  - $asf \equiv$ *an action selection function $S \rightarrow \mathcal{P}(A)$*
  - $ta \equiv$ *a function $S \rightarrow \{true, false\}$ (termination algorithm)*

Each occurrence of S is supposed to refer to the same set of Negotiation States.

Definition 6 (Bilateral Negotiation) *Let NM_1 and NM_2 be the negotiation models of A_1 and A_2 respectively. The two models represent a valid bilateral negotiation if*

- *for each message sent by A_i, an identical message (i.e. with the same parameters) is received by $A_{i \bmod 2+1}$.*
- *it is allowed that a message repeats information which has previously been disclosed. However, a message containing no new information is considered empty.*

- $rfp \equiv$ Received filtered policy
- $s \equiv$ Negotiation state
- $rn \equiv$ Received notifications
- $lp \equiv$ Local policy
- $g \equiv$ Overall goal
- $oa \equiv$ Other actor
- $ta \equiv$ Termination Algorithm
- $asf \equiv$ Action Selection Function

```
add(rfp, s)
add(rn, s)
Action[] la = extractLocalActions(g, lp, s)
while(la.length != 0)
    Notification[] ln = perform(la)
    add(ln, s)
    la = extractLocalActions(g, lp, s)
if(isUnlocked(g, lp, s))
    send(SUCCESS, oa)
    return
if(terminate(s, ta))
    send(FAILURE, oa)
    return
Action[] ea = extractExternalActions(g, lp, s)
Action[] ua
for each action in ea
    if(isUnlocked(action, lp, s)) add(action, ua)
Action[] aa = selectActions(asf, ua, s)
Notification[] sn = perform(aa)
FilteredPolicy sfp = filter(g, lp, s)
add(sfp, s)
add(sn, s)
send(sfp, oa)
send(sn, oa)
```

Figure 2. Negotiation algorithm pseudocode

- *a negotiation model (its termination algorithm) must not allow never-ending exchange of empty messages.*
- *the information provided in the filtered policy sent by an actor in subsequent messages must be monotonic, that is, let fp_{i+2} be the result of the filtering process at step $i+2$*

$$fp_{i+2} = ff(s_{i+2})$$

and fp_i the filtered policy sent at step i, then it must hold that $\mathcal{H}(fp_i) \subseteq \mathcal{H}(fp_{i+2})$

5 Implementation

In this section we describe a negotiation algorithm which instantiates the model presented in §4 as well as the architecture we implemented in order to support the integration of different trust negotiation languages (PROTUNE and PEERTRUST have already been integrated)[1].

Negotiation Algorithm A negotiation algorithm that instantiates the model presented in previous sections is described in pseudocode form in Figure 2.

At each negotiation step an actor (let say A_1) sends the other one (A_2) a (potentially empty) filtered policy rfp and

[1] A more detailed description can be found in the longer version of the paper: `http://www.l3s.de/~olmedilla/pub/2006/2006_L3S_TNModel.pdf`

a (potentially empty) set of notifications rn, stating the conditions to be fulfilled by A_2 as well as notifying the execution by A_1 of some actions it was asked for. As soon as A_2 receives them, it adds them to the negotiation state.

The local policy is then inspected in order to identify the local actions that can be executed taking into account the new information provided in the received notifications. Those local actions are performed and as a consequence other local actions may become ready for execution, for this reason local action selection and execution are performed in a loop, until no more actions are ready to be executed.

After having executed all possible local actions the policy is evaluated in order to see whether the overall goal of the negotiation is fulfilled. If this is the case, a message is sent to A_1 telling that the negotiation can be successfully terminated. Otherwise the Termination Algorithm is consulted in order to decide whether the negotiation should continue or be terminated. According to the answer, either the negotiation goes on or a message is sent to A_1 telling that the negotiation was unsuccessfully terminated.

If the negotiation is not terminated yet, then two processes have to be performed

- A_2 filters its local policy and thereby generates the new version of the filtered policy to be disclosed to A_1
- A_2 has to decide which actions requested by A_1 it will perform. Therefore, it inspects its own policy and the filtered policy received from A_1 in order to retrieve the actions requested. Since an action can be executed only if the policy protecting it is fulfilled, a check needs to be performed for each retrieved action. Those whose policies are fulfilled (*unlocked actions*) are collected and the other ones discarded.

  Unlocked actions represent potential candidates to the execution, that is, those actions which may be performed according to A_2's policy and the current negotiation state. However, just a subset of them will be actually performed, namely the one selected by the Action Selection Function

Finally, the filtered policy and the notifications of the performed external actions are added to the state and sent to A_1.

Policy Framework Architecture The negotiation model presented in this paper allows to support different policy languages. We have implemented an architecture which conforms to our negotiation model and allows not only for coexistence of different policy engines but also for reuse of components among them. Figure 3 depicts the high level architecture of our policy Agent in which we already integrated two different policy engines: a PROTUNE engine and a PEERTRUST engine.

6 Conclusions and future work

Many languages with different expressiveness have been developed to date in order to provide systems with trust negotiation capabilities. However, they typically assume different features depending on the scenarios being targeted. This paper reviewed existing approaches and frameworks for trust negotiation and summarized the main features which must be covered by a general negotiation framework. In addition, a flexible negotiation model is presented which addresses such features and an instantiation of this model is presented.

The negotiation model and the implemented algorithm presented in this paper allow for arbitrary negotiation strategies to be plugged into it. We plan to explore the use of different termination algorithms and different action selection functions in order to (semi-)automatically perform negotiations.

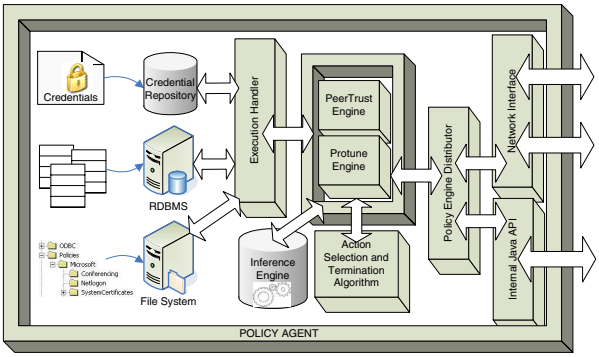

Figure 3. Policy Framework Architecture

References

[1] P. A. Bonatti and D. Olmedilla. Driving and monitoring provisional trust negotiation with metapolicies. In *IEEE POLICY*, pages 14–23, Stockholm, Sweden, June 2005.

[2] P. A. Bonatti and P. Samarati. Regulating service access and information release on the web. In *ACM CCS*, 2000.

[3] R. Gavriloaie, W. Nejdl, D. Olmedilla, K. E. Seamons, and M. Winslett. No registration needed: How to use declarative policies and negotiation to access sensitive resources on the semantic web. In *ESWS*, Heraklion, Greece, May 2004.

[4] A. J. Lee, M. Winslett, J. Basney, and V. Welch. Traust: a trust negotiation-based authorization service for open systems. In *ACM SACMAT*, Lake Tahoe, California, USA, June 2006.

[5] N. Li, J. C. Mitchell, and W. H. Winsborough. Design of a role-based trust-management framework. In *IEEE Symposium on Security and Privacy*, pages 114–130, 2002.

[6] K. E. Seamons, M. Winslett, T. Yu, B. Smith, E. Child, J. Jacobson, H. Mills, and L. Yu. Requirements for policy languages for trust negotiation. In *IEEE POLICY*, pages 68–79, Monterey, CA, USA, June 2002. IEEE Computer Society.

[7] W. H. Winsborough, K. E. Seamons, and V. E. Jones. Automated trust negotiation. DARPA Information Survivability Conference and Exposition, IEEE Press, Jan 2000.

[8] T. Yu, M. Winslett, and K. E. Seamons. Interoperable strategies in automated trust negotiation. In *ACM CCS*, 2001.