# Advanced Policy Explanations on the Web[1]

**P. A. Bonatti** and **D. Olmedilla** and **J. Peer**[2]

**Abstract.** The frameworks for protecting security and privacy can be effective only if common users—with no training in computer science or logic—increase their awareness and control over the policy applied by the systems they interact with. Towards this end, we introduce a mechanism for answering *why, why-not, how-to*, and *what-if* queries on rule-based policies for trust negotiation. Our framework is *lightweight* and *scalable* but it fulfills the main goals of modern explanation facilities. We adopt a novel *tabled explanation structure*, that simultaneously shows local and global (intra-proof and inter-proof) information, thereby facilitating navigation. Answers are focussed by removing irrelevant parts with suitable heuristics.

## 1 Introduction

The area of trust management—and in particular trust negotiation (TN)—is intersecting semantic web issues. In recent approaches [2, 5], software agents communicate their security and privacy requirements by exchanging policies formulated as rules, that is, simple ontologies. In this case, semantic descriptions concern the semantics of access control and information disclosure, which constitutes part of service and user agent semantics.

There is increasing awareness that advanced security and privacy techniques cannot be effective unless users are able to understand and possibly personalize the policy enforced by the systems they interact with (cf. [3] and the CUPS project on `http://cups.cs.cmu.edu/`). In order to enhance user awareness and control on policies, researchers are advocating a form of *cooperative policy enforcement* where policy decisions can be inspected by non-specialized users, and negative responses are enriched with suggestions and explanations.

In this paper, we describe an *advanced explanation mechanism* designed to help users understand what rule-based policies prescribe and control. Our first contribution consists in a requirements analysis for explanations in the context of trust negotiation. Moreover, we define explanation mechanisms for *why, why-not, how-to*, and *what-if* queries. There are several novel aspects in our approach:

- We adopt a *tabled explanation structure* as opposed to more traditional approaches based on single proof trees. The tabled approach makes it possible to describe infinite failures (which is essential for *why not* queries).
- Our explanations show the outcome of different possible proof attempts and let users see both local and global proof details at the same time. Such combination of intra-proof and inter-proof information is expected to facilitate navigation across the explanation structures.
- We introduce suitable heuristics for focussing explanations by removing irrelevant parts of the proof attempts. Anyway, we provide a second level of explanations where all the missing details can be recovered, if desired.
- Our heuristics are *generic*, i.e. domain independent. This means that they require no manual configuration.
- The combination of tabling techniques and heuristics yields a completely novel method for explaining failure. In the past, the problem has been ignored or formulated differently (e.g., by suggesting how to complete proofs by introducing new facts [4]).

Moreover, we aim at a *lightweight* and *scalable* explanation mechanism, that fits the requirements of web applications.

This paper is structured as follows. First, in Section 2, we recall the latest developments on explanations for expert systems. Then we outline in Section 3 the requirements for explanations in the context of trust negotiation, and briefly compare the two frameworks. Section 4 briefly introduces trust negotiation in our context and anticipates the functionalities of the explanation modules by means of a reference scenario. Then the explanation mechanisms are formally defined in two steps: First, the internal structures are defined (Section 5), then we describe how to render the explanation in natural language (Section 6). Section 7 concludes the paper with a final discussion of the results.

## 2 State of the Art

Integrated Explanation Facilities (IEFs) have been a goal for the development of intelligent systems early on. During the last fifteen years, a variety of design approaches for explanation facilities have been proposed, often classified as *second generation* frameworks by the literature [14, 6]. Examples of second generation IEFs include EES, the Explainable Expert System [11], the Mission Planning Assistant (MPA) [13, 12], the Reconstructive Explainer Rex [14]. These frameworks manage to decouple reasoning and explanation, whith the purpose of creating high quality explanations, at the cost of producing and synchronizing two versions of knowledge, one for reasoning and one for explaining. This is one of the main reasons that forced us to depart from these approaches (see next section).

From the Semantic Web perspective, the most comprehensive work on this new frontier is Inference Web (IW) [9, 10], a toolkit that aims at providing generic explanation tools for (Semantic) Web based systems. We could not use IW's facilities for our purposes because there is no support for explaining infinitely failed derivations. To navigate why-not queries we found it useful to explore multiple proof attempts simultaneously, while IW APIs are designed to manipulate single proofs. Moreover, IW's approach at removing the parts of the proof irrelevant to explanations (based on derived inference rules)

[1] Partially supported by REWERSE, IST-2004-506779.

[2] Universities of Naples, Hannover, and St. Gallen, respectively. D. Olmedilla is also a member of L3S. Contact: `bonatti@na.infn.it`

is not suitable for the kind of ellipsis and focussing we need, that we had to achieve by introducing a notion of *cluster* (see Section 5) and by exploiting predicate dependency graphs.

Finally, in the context of security, the KNOW system [7] focusses on the provision of feedback after a request is denied. However, KNOW does not return an explanation but a set of changes to the policy that would make it fulfilled. Similarly, the WhyNot system [4] suggests which sets of facts might be added to the system to make a given goal succeed, based on some heuristic weights calculated on proofs.

## 3 Requirements

Since TN frameworks can be applied in many application domains, rule-based policies almost always refer to domain specific concepts and application specific information sources. So a TN framework has to be suitably instantiated for each application by defining the set of application specific predicates, and interfacing them with legacy software and data. *Almost no further effort should be added to the framework instantiation phase.*

A second requirement is related to scalability: *explanations should not increase significantly the computational load of the servers* which is already increased by rule manipulation.

The first requirement is incompatible with the methodology of [1], that prescribes an actor and five distinct phases entirely devoted to the development of an independent explanation module equipped with an ad-hoc domain ontology and special rules for creating explanation-related data structures.

The second requirement is incompatible with the methodology of [1], too. The special rules for creating explanation-related data structures are meant to be executed during the reasoning process and may potentially affect performance.

A third requirement, instead, is shared with second generation explanation systems: explanations should be closer to the users' problem solving strategies than the system's automated reasoning strategy. The reason is that policy explanation systems should be understood by any user.

According to modern explanation approaches, the support for explanation presentation is characterized by the following features [9]:

- **Methods for asking for explanations.** We have identified the following kinds of queries: *why/why not, how to, what if*, that may be asked before, during, and after a negotiation to understand which pieces of information are actually used (some information may be unnecessarily released), what remains to be done. The same queries can be used to inspect and monitor policies.
- **Methods for breaking up proofs into manageable pieces.** The local view (the rules that directly apply to a given goal), should be enriched with global information such as the different answer substitutions of each subgoal, to help users in deciding which proof and which proof branches should be visited next.
- **Methods for pruning proofs and explanations to highlight relevant information.** The negotiation protocol determines what is relevant: Peers fulfil conditions by submitting credentials and other information, so the pieces of information that have been submitted and those that have not determine the focus of attention. State predicates constitute another kind of crucial information in TN, because their semantics is often *blurred* [2], i.e. it is not communicated to the client (either for privacy reasons or for efficiency). This raises the need for special explanations.
- **Methods and user interfaces for proof and explanation navigation:** We see a proof as a (potentially cyclic) hypertext, whose links help the user in exploring single as well as alternative proofs and proof attempts.
- **Different presentation formats.** Natural language is an appealing, user-friendly format. It can be complemented by graphical representations.
- **Methods for obtaining justifications for conflicting answers.** Today the languages involved in TN are not expressive enough to derive contradictions, so this aspect is currently marginal but this may change in the future.

Finally, the entities referred by the policies should be denoted in a user-friendly way. Their internal encoding (XML, object handles, etc.) is generally not suitable because it is meaningless to most users.

## 4 TN and Explanations

We apply our techniques to PROTUNE [2], one of the most recent trust negotiation frameworks. In summary, each party makes decisions on access control and information disclosure according to a set of rules that entail decision atoms such as `allow`$(X)$. The rule language extends Datalog with syntactic sugar. Policy rules are extended with a time-dependent set of facts, including currently available credentials and declarations (sent by the other party), other negotiation-related information, user profiles, etc. The argument of `allow`$(X)$ may refer to a service or it may denote a credential release, declaration release or the execution of some specified action. To explain its disclosure requirements to the other peers, each peer sends out the *rules* themselves, as a compact representation of all the possible ways of fulfilling the request. First, however, the rules are suitably filtered to protect the sensitive parts of the policy (the policy itself may be confidential). Explanations are built from filtered policies (so they can be built on the clients). The following scenario illustrates some of the explanations that our method produces from the filtered rules.

A digital library protects its resources according to the policy partially illustrated in Figure 1. John Smith tries to download a paper with file name "paper01234.pdf" and authenticates himself by providing an id credential. He receives the answer "permission denied" from the library service. To understand why, he sends a *why-not* query to the service. His personal assistant gathers the policies provided for why-not explanations and filters them highlighting the parts most relevant to the user, i.e. those requirements the user did not fulfill, and hiding some other aspects of the policy (e.g., those that do not depend on the user). The first output is:

**I can't prove that** it is allowed to download paper01234.pdf **because**:
Rule $[r_3]$ is not applicable:
**there is no** User **such that**
User is authenticated [details]
**and**
rule $[r_4]$ is not applicable:
**there is no** User **such that**
User is authenticated [details]

```
    ...
[r2] : allow(download(Resource)) ←
         public(Resource) .
[r3] : allow(download(Resource)) ←
         authenticated(User),
         has_subscription(User, Subscription),
         available_for(Resource, Subscription) .
[r4] : allow(download(Resource)) ←
         authenticated(User),
         paid(User, Resource) .
    ...
[r6] : authenticated(User) ←
         id(Credential),
         Credential.name : User,
         Credential.public_key : K,
         challenge(K).
[r7] : authenticated(User) ←
         declaration([username = User, password = P]),
         passwd(User, P).
    ...
```

**Figure 1.** Digital Library Policy

**moreover**
**there is no** User **such that**
User has paid for paper01234.pdf [details]

Concise explanations (like the one presented above) do not show all the details and focus primarily on those conditions that depend on the user, giving him the opportunity to fulfill the conditions quickly. For example rule $\mathtt{r_3}$ talks about subscriptions, but these details are omitted in the explanation above because if John is not authenticated it makes no sense to inspect his subscription. However, John could still request the full explanation by clicking on the [details] link.

John did disclose his id credential and wonders why the authentication failed. By clicking on that condition he eventually finds why (the definition of `id/1` is not shown in Figure 1):

**I can't find any** Cred **such that** Cred is an id **because**:
c012 is a credential with
type student-id and issuer Open University [details]
student-id is an id-type [details]
**but**
**it is not the case that**
Open University is trusted for id [details]

This explains why John's request has not been accepted: the certification authority (CA) 'Open University' on his credential is not among the trusted CAs for id credentials at the library service. Note that the above explanations anticipate the results of each subgoal (e.g. a failure, or the answer substitution fixing the credential's identifier, type and issuer). In case of multiple answer substitutions, the user can inspect the list of all answers of a subgoals and apply some, to focus on a subset of all proof attempts (see *refinement links* in the following sections).

John might ask for a complete explanation of the possible ways of obtaining the paper by issuing a *how-to* query. The result would be:

**to make sure that** it is allowed to download Resource
**nothing needs to be done if**
Resource is public [details]
**alternatively**
**please make sure that for some** User
User is authenticated [details]
**where, for some** Subscription,
User has subscription Subscription
**and**
Resource is available for Subscription [details]
**alternatively**
**please make sure that for some** User
User is authenticated [details]
**and**
User has paid for Resource [details]

Again, John can get a specific how-to explanation for each of the above conditions by clicking on it. The "make sure" part contains predicates that depend on user actions (including `credential`); they are identified via a standard *predicate dependency graph*. Due to space constraints, *what-if* queries, *why* queries, and "technical" (level 2) explanations that provide full proof details are not discussed in this paper.

## 5 Explanation Structures

To meet page limitations, we present here slightly simplified explanation structures that do not support blurred predicates (which involve notions similar to abductive explanations and verbalizations such as: "it might be the case that...").

We assume the reader to be familiar with the basics of logic programming, including the notions of *most general unifier* (mgu) and *(computed) answer substitution*. The reader is referred to [8] for these matters. There are some delicate technical aspects in handling substitutions, related to standardization apart. So we assume a function $\mathtt{ans}^E_P(G)$ that for all programs $P$, all goals $G$, and all expressions $E$ returns a complete (up to variable renaming) set of answer substitutions whose range does not contain any variable occurring in $P$, $E$, $G$.

Before formalizing explanations we tackle the problem of referring to structured objects, such as credentials, that are typically denoted by means of internal identifiers (*handles*) that have no meaning to the common user. We noted that the attribute atoms *obj.attr:val* in a rule body are typically the relevant, characterizing attributes of the complex object whose handle is *obj*—see for instance the why-not explanation referring to credential c012 in Section 4. In a given negotiation, these attributes almost always identify a unique object when their values are fixed. So our idea is using these attributes as a key to identify the object. For this purpose, we look for subsets of the body called *clusters* that correspond to concepts with attributes (a sort of terminological expression) and treat them as a description of the complex object.

**Definition 1 (Clusters)** *Let $L = p(t)$ where $p$ is a unary predicate and $t$ is a term. If $L \in \mathtt{body}(r)$, then a* cluster *of $L$ (in $r$) is a set containing $L$ and some attribute subgoals $(u.a : v) \in \mathtt{body}(r)$ with $u = t$. A cluster is* complete *if it contains all such subgoals. If $t$ is a variable, then it is called the* main variable *of the cluster; $L$ and $p$ are called the* main literal *and the* main predicate *of the cluster, respectively.*

**Example 1** The policy rule instance $\mathtt{r_6}$ in Figure 1 contains one complete cluster (the first three literals in the body) whose main literal is `id(Credential)`. The cluster denotes "the id with name User and public key K". □

**Remark 1** Clusters are more flexible than key attributes because they can be applied also to those classes of objects that

have no key attributes. Moreover, clusters reduce the framework instantiation effort, because they need no manual intervention (while key attributes must be specified by knowledge engineers). Clusters are very useful in *why not* queries, where incomplete clusters allow to explain situations such as: "*there is an id with name J. Smith, but the public key is not k*012".

While a single atom like $\texttt{credential}(X)$ may have multiple solutions, often its cluster has just one answer (in this sense the attributes of $X$ *characterize* $X$). By applying this substitution, we specify the credential we are talking about. The process of exhaustively applying substitutions when they are the unique answer of a cluster or a subgoal is formalized as a binary (rewrite) relation $\longrightarrow_U$ over annotated rules $(r, \theta)$. The precise definition and the heuristics for the cases where different rewrite sequences yield different results[3] are in the full report. The framework can be adapted to different heuristics simply by changing the definition of $\longrightarrow_U$.

We are ready to formalize explanations. They are graphs (abstracting a hypertext) where each node illustrates: (i) the local context for a goal, that is, the rules whose head unifies with that goal; (ii) a global view of all the proof attempts, consisting in the answers of each rule body and subgoal thereof. Such answers provide a sort of *lookahead* on proof outcomes that may help the user in navigating the proof. We proceed by defining the explanation graph, starting with its nodes and entry points.

**Definition 2** *An* explanation node *for a program $P$ is a finite set of pairs $(r, \theta)$ where $r \in P$ and $\theta$ is a substitution.*

*The* explanation entry point *for an atom $A$ w.r.t. $P$, denoted by $\mathsf{entry}_P(A)$, is the (unique) explanation node $X$ for $P$ such that*

$$X = \{(r, \theta) \mid r \in P,\ A \text{ is unifiable with } \mathsf{head}(r), \text{ and } (r, \mathsf{mgu}(A, \mathsf{head}(r))) \longrightarrow_U (r, \theta)\ \}.$$

There are two kinds of navigation links: *detail links* and *refinement links*. Informally, the former expand the proof details by showing the rules that can be used to prove a subgoal; the latter apply answer substitutions locally to the rule to see the effects on the other subgoals and focus on a subset of all the possible proofs.

Now we can proceed with the formalization of navigation links. They lead directly to nodes where unique answers have been propagated, and consider only maximally general answer substitutions to reduce redundancy. So we need a function $\mathsf{mg}(S)$ that for all sets of substitutions $S$ returns the maximally general elements of $S$. In the following we extend $\longrightarrow_U$ to explanation nodes as follows:

$$X_1 \longrightarrow_U X_2 \text{ iff } X_2 = \{(r, \theta') \mid \text{for some } (r, \theta) \in X_1, (r, \theta) \longrightarrow_U (r, \theta')\}.$$

**Definition 3 (Level 1 navigation link)** *For all explanation nodes $X_1$ and $X_2$ for $P$ we define:*

**detail links:** $X_1 \overset{L}{\rightsquigarrow}_D X_2$ *iff for some $(r, \theta) \in X_1$ and some $L \in \mathsf{body}(r)$, $\mathsf{entry}_P(L\theta) \longrightarrow_U X_2$;*

**refinement links:** $X_1 \overset{\sigma}{\rightsquigarrow}_R X_2$ *iff for some $(r, \theta) \in X_1$ and some $L \in \mathsf{body}(r)$, $\sigma \in \mathsf{mg}(\mathsf{ans}^r_P(L\theta))$ and $\{(r, \theta\sigma)\} \longrightarrow_U X_2$*

[3] This may happen if $\mathsf{body}(r)$ fails.

**Example 2** Let $P$ be the program illustrated in Figure 1 extended with time-dependent facts, and $A = \texttt{allow}(\texttt{'paper0123.pdf'})$. If the body literals of $\texttt{r}_2$, $\texttt{r}_3$, and $\texttt{r}_4$ have no solutions, then $\mathsf{entry}_P(A) = \{(\texttt{r}_2, \theta), (\texttt{r}_3, \theta), (\texttt{r}_4, \theta)\}$ where $\theta = [\mathit{Resource} = \texttt{'paper0123.pdf'}]$. If only $\texttt{authenticated}(\mathit{User})$ has a unique solution, say, $[\mathit{User} = \texttt{'John'}]$, then $\mathsf{entry}_P(A) = \{(\texttt{r}_2, \theta), (\texttt{r}_3, \theta'), (\texttt{r}_4, \theta')\}$ where $\theta' = [\mathit{Resource} = \texttt{'paper0123.pdf'}, \mathit{User} = \texttt{'John'}]$. Now suppose that the subgoal $\texttt{has\_subscription}(\texttt{John}, \mathit{Subscription})$ of $\texttt{r}_3\theta'$ has two answer substitutions $\sigma$, $\sigma'$. The action of selecting and applying $\sigma$ to $\texttt{r}_3$ is formalized by crossing the refinement link $\mathsf{entry}_P(A) \overset{\sigma}{\rightsquigarrow}_R \{(\texttt{r}_2, \theta), (\texttt{r}_3, \theta'\sigma), (\texttt{r}_4, \theta')\}$. Otherwise, the action of expanding the details of the subgoal $L = \texttt{authenticated}(\mathit{User})$ of $\texttt{r}_3\theta'$ is formalized by the detail link $\mathsf{entry}_P(A) \overset{\sigma}{\rightsquigarrow}_R \{(\texttt{r}_6, \gamma), (\texttt{r}_7, \gamma')\}$, where $\gamma$, $\gamma'$ result from applying the unique answers of the subgoals of $\texttt{r}_6$ and $\texttt{r}_7$. □

**Definition 4** *A* level 1 explanation graph *for an atom $A$ w.r.t. a partially specified program $P$ is a minimal structure $(V, E^D, E^R)$ closed under the following properties:*

1. *$V$ contains $\mathsf{entry}_P(A)$;*
2. *if for some $X_1 \in V$, $X_1 \overset{L}{\rightsquigarrow}_D X_2$ then $V$ contains exactly one variant $\tilde{X}_2$ of $X_2$, and $E^D$ contains an edge $(X_1, L, \tilde{X}_2)$;*
3. *if for some $X_1 \in V$, $X_1 \overset{\sigma}{\rightsquigarrow}_R X_2$ then $V$ contains exactly one variant $\tilde{X}_2$ of $X_2$, and $E^R$ contains an edge $(X_1, \sigma, \tilde{X}_2)$.*

Since our programs are basically Datalog programs, it can be shown that the explanation graph is always finite. It is cyclic in the presence of infinitely failed proofs, and the browser can highlight loops by marking the nodes that have already been visited. In the full paper, *explanation structures* are defined as explanation graphs labelled with suitably filtered answer substitutions (the *global*, possibly *inter-proof* information).

## 6 Verbalization

Explanation structures are translated into natural language sentences by instantiating text patterns. The verbalization patterns for application dependent literals must be supplied during the framework instantiation phase. They are typically formulated with simple PROTUNE metafacts like:

$$\texttt{is\_authenticated(X).explanation} : [\texttt{X}, \texttt{is}, \texttt{authenticated}]$$

If necessary, proper rules (with nonempty body) can be used and it is also possible to define text patterns for negative literals. By default, variable names are taken from the original rule—which is the first element of the current node $(r, \theta)$.

A cluster $C = \{L, t.a_1 : v_1, \ldots, t.a_n : v_n\}$ of $(r, \theta)$ with main literal $L$ is verbalized as a single condition:

$$\mathsf{verb}(L_1\theta) \textbf{ with } a_1\ v_1\textbf{, } a_2\ v_2\textbf{, ..., and } a_n\ v_n.$$

The rest of this section is implicitly formulated w.r.t. the following context:

- a node $X = \{(r_1, \theta_1), \ldots, (r_z, \theta_z)\}$ of $XG$ (the current node);
- an atom $A$ (the one being currently explained), more general than $\mathsf{head}(r_i\theta_i)$ for $1 \le i \le z$.

For all clusters or literals $Y$ and for all substitutions $\theta$, the verbalization of $Y\theta$ will be denoted by $\mathsf{verb}(Y, \theta)$. For the sake of simplicity, the navigation links will not be shown.

We show the verbalization details for *why-not* queries (the details for the other queries are in the full paper). They start with:

**I can't prove that** $\mathsf{verb}(A, \varepsilon)$ **because:** (if $A$ is ground)

**I can't find any <*var list*> such that** $\mathsf{verb}(A, \varepsilon)$ **because:** (if $A$ is not ground).

The above *incipit* is followed by the verbalization of all $(r_i, \theta_i) \in X$. Each pair $(r_i, \theta_i)$ is formatted according to the following pattern:

`Rule` <*rule identifier*> `is not applicable:`
<*Success Slot*> **but** <*Failure Slot*>

where **but** is omitted if any of the two slots is missing. The Success Slot in its most complete version is verbalized as:

<*quantification*>
$\mathsf{verb}(Y_1, \theta_i)$ **and ...and** $\mathsf{verb}(Y_m, \theta_i)$ **and** $\mathsf{verb}(L_1, \theta_i)$ **and ...and** $\mathsf{verb}(L_n, \theta_i)$

where $Y_1 \dots Y_m$ are all the $\subseteq$-maximal clusters in $\mathsf{body}(r_i\theta_i)$ with a single answer (i.e. $|\mathsf{ans}_P(Y_i)| = 1$), and $L_1 \dots L_n$ are all the literals with a single answer (i.e. $|\mathsf{ans}_P(L_j)| = 1$) that do not occur in $Y_1 \dots Y_m$.

If $\mathsf{body}(r_i\theta_i)$ contains some failed clusters or literals (with zero answers), then the Failure Slot is verbalized as[4]

$\mathsf{verb}(\mathtt{not}\ Z_1, \theta_i)$ **moreover...moreover** $\mathsf{verb}(\mathtt{not}\ Z_j, \theta_i)$ **moreover** $\mathsf{verb}(\mathtt{not}\ L'_1, \theta_i)$ **moreover...moreover** $\mathsf{verb}(\mathtt{not}\ L'_k, \theta_i)$

where $\{Z_1 \dots Z_j\}$, denoted by minfailed, consists of the $\subseteq$-minimal failed clusters in $\mathsf{body}(r_i\theta_i)$, and $L'_1 \dots L'_k$ are the failed literals in $\mathsf{body}(r_i\theta_i)$ that neither occur in minfailed nor belong to the set of attribute atoms $t.a{:}v$ such that $t$ is the main variable of a singleton cluster in minfailed. Wit the latter condition, the attributes of non-existing objects are ignored.

**Example 3** If no id has been provided, then (with this condition) the *why not* verbalization of $\mathtt{r_6}$ is simply: "**there is no** `Credential` **such that** `Credential is an id`" as expected. A naive approach would have produced something like "**there is no** `Credential` **such that** `Credential is an id` **with** `name User and public_key K`", which is misleading. □

If no clusters or literals fail, the Failure Slot is verbalized as

**each of the following facts has some solution**
<*quantification*> $\mathsf{verb}(L''_1)$ ... <*quantific.*> $\mathsf{verb}(L''_k)$
**but there is no common solution**

where $L''_1 \dots L''_k$ are the literals with two or more solutions in $\mathsf{body}(r_i\theta_i)$. Via refinement links, the user may apply a chosen answer substitution and explore why it eventually fails (i.e. which other subgoals become failed after applying that substitution).

## 7 Conclusions

In TN it is possible to construct good explanations in response to *why, why not, how to*, and *what if* queries, using simple generic explanation strategies based on the intended meaning of a few core predicates with a special role in negotiations. The only extra workload needed during the framework instantiation phase to support explanations consists in writing literal verbalization patterns. Moreover, the extra computational burden on the server can be limited to adding a few more rules to the filtered policies (the literal verbalization rules) because the explanations can be independently produced on the clients. Despite its simplicity, our explanation mechanism supports most of the advanced features of second generation explanation systems.

Another contribution of our work is a novel explanation structure combining local and global information about single and multiple proof attempts. The new explanation structures are analogous to the tables of tabled logic programming engines, so our work might be applied to execution tracing in tabled Prolog implementations like XSB. Our method provides an effective way of explaining infinitely failed proofs. Global proof information and heuristics help in focussing the explanation strictly on relevant details. The resulting explanations have no counterpart in previous literature.

[4] In the following expression we identify $\mathtt{not\,not}\ A$ and $A$.

## REFERENCES


[1] R. Barzilay, D. McCullough, O. Rambow, J. DeChristofaro, T. Korelsky, and B. Lavoie. A new approach to expert system explanations. In *9thInternational Workshop on Natural Language Generation*, pages 78–87. 1998.

[2] P. A. Bonatti and D. Olmedilla. Driving and monitoring provisional trust negotiation with metapolicies. In *6th IEEE International Workshop on Policies for Distributed Systems and Networks*, pages 14–23, jun 2005.

[3] Bonatti et al. The rewerse view on policies. In *Proc. of the ISWC Semantic Web Policy Workshop (SWPW)*, `http://ebiquity.umbc.edu/get/a/publication/215.pdf`, 2005.

[4] H. Chalupsky and T. A. Russ. Whynot: debugging failed queries in large knowledge bases. In *14th national conference on Artificial intelligence*, pages 870–877, 2002.

[5] 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 *1st European Semantic Web Symposium*, volume 3053, pages 342–356, may 2004.

[6] S. R. Haynes. *Explanation in Information Systems: A Design Rationale Approach*. PhD thesis, London School of Economics and Political Science, Dept. of Information Systems and Dept. of Social Psychology, 2001.

[7] A. Kapadia, G. Sampemane, and R. H. Campbell. Know why your access was denied: regulating feedback for usable security. In *11th ACM conference on Computer and communications security*, pages 52–61, 2004.

[8] J. Lloyd. *Foundations of logic programming*. Springer-Verlag, 1984.

[9] D. L. McGuinness and P. P. da Silva. Explaining answers from the semantic web: The inference web approach. *Journal of Web Semantics*, 1(4):397–413, 2004.

[10] D. L. McGuinness and P. P. da Silva. Trusting answers from web applications. In *New Directions in Question Answering*, pages 275–286, 2004.

[11] W. Swartout, C. Paris, and J. Moore. Explanations in knowledge systems: Design for explainable expert systems. *IEEE Expert: Intelligent Systems and Their Applications*, 6(3):58–64, 1991.

[12] M. C. Tanner, A. Keunecke, and B. Chandrasekaran. Explanation using task structure and domain functional models. In *Second Generation Expert Systems*, pages 586–613. 1993.

[13] M. C. Tanner and A. M. Keuneke. Explanations in knowledge systems: The roles of the task structure and domain functional models. *IEEE Expert: Intelligent Systems and Their Applications*, 6(3):50–57, 1991.

[14] M. R. Wick. Second generation expert system explanation. In *Second Generation Expert Systems*, pages 614–640. 1993.