I2-D4

Advanced Policy Queries

Project number:	IST-2004-506779
Project title:	Reasoning on the Web with Rules and Semantics
Project acronym:	REWERSE
Document type:	D (deliverable)
Nature of document	R (report)
Dissemination level:	CO (confidential, only for REWERSE partners)
Document number:	IST506779/Naples/I2-D4/D/CO/a0.0
Responsible editor(s):	P. A. Bonatti
Contributing participants:	Hannover, Naples, St-Gallen
Contributing workpackages:	I2
Contractual date of delivery:	31 August 2005
Actual date of delivery:	31 August 2005

Abstract
We argue that policy-aware systems can be effective only if: (i) common users—with no training in computer science or logic—become aware of the policy applied by their system; (ii) common users can personalize those policies; (iii) secure systems guide the user in getting the required permissions (*cooperative enforcement)*. Towards this end, we introduce a mechanism for answering *why, why-not, how-to*, and *what-if* queries. Our framework is *lightweight* and *scalable* because it does not require any major effort when the general framework is instantiated in a specific application domain, and most of the computational effort can be delegated to the clients. Some novel aspects in our approach: First, we adopt a *tabled explanation structure*, that simultaneously shows local and global (intra-proof and inter-proof) information, thereby facilitating navigation. Second, we introduce generic heuristics for removing irrelevant parts of the derivations. Third, our heuristics do not require the complex machinery needed by second-generation explanation systems, but have a comparable quality.

Keyword List
Explanations for rule-based policies and trust negotiation, Cooperative enforcement, Tabled explanations, Verbalization

Advanced Policy Queries

P. A. Bonatti[1], **D. Olmedilla**[2], **and J. Peer**[3]

[1] Università di Napoli Federico II
Email: `bonatti@na.infn.it`

[2] L3S Research Center and Hanover University
Email: `olmedilla@l3s.de`

[3] St. Gallen University
Email: `joachim.peer@unisg.ch`

26 September 2005

Abstract
We argue that policy-aware systems can be effective only if: (i) common users—with no training in computer science or logic—become aware of the policy applied by their system; (ii) common users can personalize those policies; (iii) secure systems guide the user in getting the required permissions (*cooperative enforcement).* Towards this end, we introduce a mechanism for answering *why, why-not, how-to*, and *what-if* queries. Our framework is *lightweight* and *scalable* because it does not require any major effort when the general framework is instantiated in a specific application domain, and most of the computational effort can be delegated to the clients. Some novel aspects in our approach: First, we adopt a *tabled explanation structure*, that simultaneously shows local and global (intra-proof and inter-proof) information, thereby facilitating navigation. Second, we introduce generic heuristics for removing irrelevant parts of the derivations. Third, our heuristics do not require the complex machinery needed by second-generation explanation systems, but have a comparable quality.

Keyword List
Explanations for rule-based policies and trust negotiation, Cooperative enforcement, Tabled explanations, Verbalization

Contents

1 Introduction

One of the causes of the enormous number of computer security violations on the Internet is the users' lack of technical expertise. In particular, users are typically not aware of the security policies applied by their system, not to speak about how those policies can be changed and how they might be improved by tailoring them to specific needs. As a consequence, most users ignore their computer's vulnerabilities and the corresponding countermeasures, so the system's protection facilities cannot be effectively exploited.

For example, it is well known that the default, generic policies that come with system installations—biased toward functionality rather than protection—are significantly less secure than a policy specialized to a specific context, but very few users know how to tune or replace the default policy. Moreover, users frequently do not understand what the policy is really checking up on, and hence they are unaware of the risks involved in many common operations.

Similar problems affect privacy protection. In trust negotiation, credential release policies are meant to achieve a satisfactory tradeoff between privacy and functionality (many interesting services cannot be obtained without releasing some information about the user). However, one cannot expect such techniques to be effective unless users are able to understand and possibly personalize the privacy policy enforced by their system.

Additionally, a better understanding of a web service's policy makes it easier for a first-time user to interact with the service. If a denied access results simply in a "*no*", then the user has no clue on how he or she can possibly acquire the permission to get the desired service (e.g., by completing a registration procedure, by supplying more credentials, by filling in some form, etc.) We are advocating a form of *cooperative policy enforcement*, where negative responses are enriched with suggestions and other explanations whenever such information does not violate confidentiality (sometimes, part of the policy itself is sensitive).

For these reasons, the working group on policies of the Network of Excellence REWERSE selected as one of its main objectives *greater user awareness and control on policies* (including security and privacy policies as well as more general notions of policy discussed elsewhere, such as business rules and quality of service). REWERSE is mainly concerned with the technical side of the problem, which is complex enough to require an articulated approach. In particular we are making policies easier to understand and formulate to the common user in the following ways:

- We adopt a *rule-based policy specification language*, because such languages are very flexible and at the same time structurally similar to the natural way in which policies are expressed by nontechnical users.

- We are making the policy specification language more friendly by developing a *controlled natural language* front-end to translate natural language text into executable rules.

- We are developing *advanced explanation mechanisms* to help the user understand what policies prescribe and control.

This deliverable is focussed precisely on the explanation mechanisms. Our first contribution consists in a requirements analysis for explanations in the context of *automated trust negotiation* (ATN). 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 derivations or proof trees. The tabled approach makes it possible to describe infinite failures (which is essential for *why not* queries).
- Our explanations show simultaneously different possible proof attempts and allow users to see both local and global proof details at the same time. Such combination of local and global (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 [Chalupsky and Russ, 2002]).
- We study the interactions between policy filtering and explanations, and propose a refined policy filtering process to enhance the quality of explanations.

We aim at a *lightweight* and *scalable* explanation mechanism that does not require any major effort when the general framework is instantiated in a specific application domain, and where most of the computational effort can be delegated to the clients.

The deliverable 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 automated trust negotiation, and briefly compare the two frameworks. Section 4 anticipates the functionalities of the explanation modules by means of a reference scenario and some use cases. Then the explanation mechanisms are formally defined in two steps: First, the internal structures are defined (Section 6), then we describe how to render the explanation in natural language (Section 7). Section 8 concludes the paper with a final discussion of the results.

The policy specification language adopted here is PROTUNE, the core policy language of REWERSE. The reader is referred to [Bonatti and Olmedilla, 2005b, Bonatti and Olmedilla, 2005a] for the language's definition and examples.

2 State of the Art

The task of answering queries of users who inquire why the system made a certain decision or who want to know what is needed to reach a given goal, is called *explanation*. Explanation is one of the most common methods used by humans to support decisions [Schank, 1986].

The ability to provide explanations for the results of reasoning tasks has been a goal for the development of intelligent system early on. One early expert system with an integrated explanation facility (IEF) is MYCIN [Shortliffe, 1976], a system designed to support physicians in diagnosing bacterial infections. MYCIN uses backward chaining, i.e. it selects a hypothesis that may explain the symptoms entered by the user, and then attempts to confirm the hypothesis by going through the rules that constitute the respective illness while checking the consistency of the rules with the facts provided by the user. In MYCIN, a user can ask a “why” query, which causes MYCIN to display the rules that led it from the hypothesis to the basic medical facts established in the case.

This idea of accessing proof trees or rule traces is appealing because it is usually not difficult to implement and does not imply any additional effort for the domain engineers. The concrete implementation of the approach depends on the underlying expert system infrastructure. For instance, in Prolog the explanation facility can be realized using a simple Prolog meta interpreter, as described in [Schnupp, 1989].

Critical reviews of this approach and its usability led to the definition of more ambitious requirements for explanations. Notably [Hasling et al., 1984] demand from IEFs (i) that explanations should not presuppose a particular user population, i.e. it should be possible to adapt them for the various user groups that work with the system, (ii) that explanations should be informative and readable for the user, and (iii) that the level of detail of the explanations should suit the respective situation, i.e. the explanation should allow both fine grained and coarse grained levels of detail.

Along the same lines [Clancey, 1983] notes that much of MYCIN’s reasoning and explanation behavior is implicitly hidden in the rule base, e.g. in the ordering of rules. In particular, he distinguishes three categories of system knowledge that could be explicitly represented in intelligent applications: *strategic* knowledge, which represents the high-level problem solving approaches behind the inference rules in the system, *support* knowledge, which contains methods to justify the applicability of a given rule (e.g. by identifying facts that trigger the rule), and *structural* knowledge, which refers to way the various rules are classified, ordered or chained together.

Some of these ideas have been implemented in the NEOMYCIN project [Clancey and Letsinger, 1984]; in particular, the problem solving strategy was represented explicitly in the application and meta-rules have been used for explanation planning.

MYCIN and its derivatives are frequently viewed as the *first generation* of expert systems with explanation facilities. During the past 15 years, a variety of different approaches for the design of explanation facilities have been proposed, often classified as *second generation* frameworks by the literature [Wick, 1993, Haynes, 2001].

One such project is EES, the Explainable Expert System [Swartout et al., 1991], where the knowledge representation is enriched (i) by explicit strategic knowledge (i.e. an explicit representation of the reasoning methods to be applied by the system) and (ii) by structured domain knowledge (i.e. definitions of terms).

A similar approach was taken for the Mission Planning Assistant (MPA) [Tanner and Keuneke, 1991, Tanner et al., 1993], a DARPA sponsered project: The various tasks in MPA’s application domain make use of several different reasoning strategies (e.g. hypothesis generation, deduction), hence MPA requires a more flexible explanation facility than systems where only a single reasoning method is used. In particular, explaining a solution of the MPA requires (i) showing how the logical requirements of the given tasks were satisfied by the solution and (ii) showing how the reasoning strategy adopted achieved the task.

Another second generation approach is Rex, the Reconstructive Explainer [Wick, 1993]. In Rex, after a reasoning task is finished, the expert system creates a causal chain for explanations in a separate process.

While these frameworks manage to decouple reasoning and explanation, which is to some extent necessary for the creation of high quality explanations, some of the more sophisticated aspects of those frameworks may become limitations, if applied in application contexts that do not really require them: For instace, EES and MPA require knowledge engineers to manage separate knowledge bases, and to explicitly deal with strategic knowledge. However, if the selection of strategic knowledge plays no role in the application domain, then the effort cannot be justified, because it provides no real benefits. Similarly, the problem of the Rex approach lies in the separation of the reasoning and explanation process, which can be seen as a redundancy and might be avoided using a simpler approach where the explanation structure is created ad-hoc during the reasoning process [Barzilay et al., 1998].

While we aim at achieving high quality explanations in our framework – characterized by different levels of granularity, support for different user groups, interactivity and verbalization – we also desire to keep the explanation system as lightweight and usable as possible. In fact, we shall argue in the next section that the conceptual model of Protune does not call for a full-fledged second generation framework as sketched above.

On the other hand, since the application context of Protune is the area of distributed information systems, in the form of the envisioned "Semantic Web", it is imperative to review the existing work on explanations in such systems. The most comprehensive work on this new frontier is Inference Web (IW) [McGuinness and da Silva, 2004a, McGuinness and da Silva, 2004b], a toolkit that aims at providing useful explanations for the behavior of (Semantic-) Web based systems. In particular, [McGuinness and da Silva, 2004a] propose support for knowledge provenance information using meta-data (e.g. Dublin Core information) about the distributed information systems involved in a particular reasoning task. Further, [McGuinness and da Silva, 2004a] deal with the issue of representing heterogeneous reasoning approaches, domain description languages and proof representations; the latter issue is addressed by using PML, the OWL-based Proof Markup Language [da Silva et al., 2004].

In the context of security, the KNOW system [Kapadia et al., 2004] focus on the provision of feedback to users after a request is denied. However, they do not provide an explanation but a set of changes on policy conditions that must hold in order to have a policy satisfied. Furthermore, they have a separate system for the generation of feedback which builds ordered binary decision diagrams (OBDDs) from policies. These OBDDs and shortest-path algorithms based on cost functions are used to find the relevant set and return as feedback only the k-least costly subset. In this document we provide an approach for policy explanation in query answering (not just feedback) integrated within the policy engine, reducing the complexity of its generation. Still, filtering based on cost functions could be integrated based on the cost metapolicy defined in our language.

Similarly, the WhyNot system [Chalupsky and Russ, 2002] 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. The similarity between the name of this system and the name of our *why not* queries is misleading, because the latter do not necessarily suggest any theory extensions (which sometimes make little sense when a user is inspecting the answer of a server) and do not give different preferences to proof attempts based on their size or the number of failed conditions, because this kind of heuristics is meaningless in the ATN context.

3 Policy Explanations: Analysis and Requirements

In this section we examine all the design principles for explanation systems adopted by second generation approaches and evaluate them with respect to the specific needs of ATN frameworks.

Sometimes we add new requirements arising in this area. As we discuss design principles and solutions, we compare our approach with related work.

3.1 Instantiation effort

As a general requirement, the PROTUNE ATN framework should be easy to instantiate in any given application domain. At the very least, as part of the instantiation process, one needs to provide the set of application-specific state predicates, and interface them with legacy software and data (if the policy needs to refer to such things). Application specific predicates can be identified by parsing the natural language policy, so the process can be partially assisted with automated tools. *We are aiming at adding almost no further effort to the framework instantiation phase.*

This objective is incompatible with the methodology of [Barzilay et al., 1998], that prescribes an actor and five distinct phases entirely devoted to the development of an explanation module which is completely independent from the expert system, being equipped with an ad-hoc domain ontology and special rules for creating explanation-related data structures. The five steps would cause an undesired overload during the instantiation phase.

In our framework, the instantiation phase should include only a simple extra step consisting in creating so-called *literal verbalization rules*, that visualize logical application-dependent literals in natural language. We are planning to (partially) automate this activity by means of the natural language front-end. We are *not* planning to have any steps as complex as designing a domain ontology or arbitrary sets of rules.

To achieve a satisfactory explanation mechanism under these constraints we are relying on the fact that PROTUNE lies somewhere in between an abstract (non instantiated) reconstructive explanation framework and a specific (instantiated) explanation system. The set of predicates is partly application specific, but there is a core set or pre-defined predicates (such as `credential`, `declaration`, and `do`) whose special role in negotiations can be exploited to craft clever explanation strategies. In other words, part of the activities prescribed by [Barzilay et al., 1998] are anticipated to the design of PROTUNE's general framework.

3.2 Performance

A second general requirement related to scalability is that *explanations should not increase significantly the computational load of the servers.*

This objective is incompatible with the methodology of [Barzilay et al., 1998], too. The special rules for creating explanation-related data structures are meant to be executed during the reasoning process, so their execution may potentially affect performance.

In PROTUNE servers have to compute *filtered policies* for ATN to take place. The only extra effort for explanation should consist in adding the literal verbalization rules and some facts (state predicate information) to the filtered policy. Explanation-specific structures are meant to be built on the clients starting from this basic set of rules and facts. In this way, most of the computational burden related to explanations is moved to the clients.

3.3 Explanation method

Today, it is commonly accepted that the proofs built by reasoning engines (and more generally their reasoning strategy) are not suited to building satisfactory explanations. The user is typically forced to explore the search space like the engine does, going through a number of uninteresting details.

Like second-generation explanation systems, PROTUNE creates ad-hoc data structures built expressly for explanations (see Section 6). One of the distinctive features of our structures is that they give simultanously a local view of how to prove a fact—based on the rules that immediately apply to the fact—and a global view, comprising the final outcome of alternative proof attempts for the rule body. Such global information guides the user in navigating the set of proof attempts, focussing on the parts of the search space that do not meet the user's expectations.

This approach relies on the nice computational properties of our policy rules, that are basically notational variants of Datalog rules. Since the consequences of Datalog programs are decidable and can be computed efficiently (e.g. by means of tabling, like in XSB Prolog), we can afford to compute the global information associated to explanation structures. A novel, *tabled* form of explanation structures solves the problem of presenting infinitely failed proofs.

The effectiveness of explanation structures is enhanced through suitable presentation heuristics that produce concise explanations, typically focused on the details relevant to the query (see sections 3.5 and 6.3). If the heuristics delete too much information, then the user may ask for a more detailed and more technical explanation (cf. Section 6.2).

3.4 Query purpose and explanation-oriented metadata

While the users of a question-answering or a decision-support system are interested in how the system reaches its conclusions to evaluate their reliability, the users of a secure system can only accept the system's access control decisions.

Other differences between ATN and expert systems are specific to *why not* queries. The common reasons for query failures are [Chalupsky and Russ, 2002]:

Missing knowledge

Limitations such as incomplete reasoners, resource limitations, timeouts, etc.

Misconceptions of the user such as using terms that are not defined in the knowledge base, queries that do not match the intended question, or inherently contradictory questions.

Bugs in the knowledge base and/or in the inference engine.

In ATN, engines are complete (for reasons explained below) and the only possible limitation may arise from timeouts in distributed proofs, when third parties are involved. Moreover, misconception are not an issue, because (i) in the typical context in which ATN systems are embedded, permission requests are automatically crafted by the web server, not by the user, and (ii) in PROTUNE, subsequent queries are guided by the explanation navigation system.

In ATN, users expect explanations to illustrate aspects such as which of the submitted credentials were actually needed to get a resource, or why a given combination of credentials was not sufficient to obtain the desired service—hence the importance of *why not* queries. Moreover, users and security administrators may be interested in understanding why their own (access control or credential release) policy does not make the expected decisions.

In both cases, the query has essentially a *diagnostic purpose*. As a consequence, several explanation-oriented metadata adopted in previous works with the purpose of measuring inference reliability and ranking alternative conclusions are not needed in our explanation framework.

An example of such marginal metadata is *knowledge provenance* [McGuinness and da Silva, 2004a], that includes information such as source names, date, author, authoritativeness of the source, degree of belief (that the author has in some piece of information), and degree of completeness (of the source w.r.t. a particular scope).

Interestingly, these pieces of information are explicitly handled by some policies, that in making their decisions may take into account who issued a certain credential (source name, author, authoritativeness, see also the reference scenario in Section 4), numerical reputation models [Bonatti et al., 2005] (degree of belief), credential search strategies (that may consider the degrees of completeness of a server before keeping on searching other hosts). In other words, information provenance is not explanation-oriented metadata but rather policy domain knowledge. Similarly, term inter-relationships such as ontological relations (subclass, superclass, part-of), if present in PROTUNE applications, are domain knowledge, not metadata, and can be used also during object-level reasoning.

Another example of marginal metadata is the *information about the reasoner* [McGuinness and da Silva, 2004a], comprising the reasoner used, its reasoning method (e.g. tableaux, resolution, etc.), supported inference rules, soundness and completeness properties, nonmonotonic assumptions (closed world, unique names assumption), reasoner author, version, etc.

In ATN, semantic aspects such as nonmonotonic assumptions are fixed (all the current approaches are based on standard logic programming semantics because of its constructive nature and its simplicity), and the engines are always assumed to be sound and complete, to ensure that no unintended permission is granted (soundness) and no legal access is denied (completeness). Of course, it is possible to assume soundness and completeness because policies belong to an efficiently decidable fragment of logic, as we already pointed out.

Some explanation-oriented metadata are important also in ATN. In PROTUNE we have *term meaning* expressed in natural language, encoded in the form of literal verbalization rules (see Section 3.1).

3.5 Presentation strategies

The support for explanation presentation is characterized by the following features [McGuinness and da Silva, 2004a]:

Methods for asking for explanations. We have identified the following kinds of queries: *why/why not, how to, what if.* Such questions may be asked before, during, and after a negotiation to understand which pieces of information are actually used (sometimes information is unnecessarily released in negotiations), what still needs to be done to obtain the desired service, as well as to inspect and monitor access control and credential release policies. The answers to such queries may determine the next negotiation steps.

Methods for breaking up proofs into manageable pieces. We give a local view (the rules that directly apply to a given goal), enriched with global information, as explained above. Moreover, we group together related literals (such as all the constraints on a single credential's attributes) and treat them as a unique condition called *cluster*.

Methods for pruning proofs and explanations to highlight relevant information. The negotiation protocol determines what is relevant. Since clients have to fulfil conditions by submitting credentials and other information to servers, the pieces of information that have been submitted and those that have not determine the focus of attention in explaining success and failure. State predicates constitute another kind of crucial information in ATN, because their semantics is often *blurred* [Bonatti and Olmedilla, 2005b], i.e., it is not communicated to the client (either for privacy reasons or for efficiency). Therefore, it is important to focus the user attention on the success and failure of blurred predicates. All the information that seems not helpful in illustrating these aspects (according to some heuristics described in the following) is not included in the explanations. We provide a more detailed level of explanations for the cases where the complete details of the proof attempts are needed.

Methods for proof and explanation navigation,

Presentation solutions compatible with web browsers, and

Methods for obtaining alternative justifications for answers: 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. The explanations we support are written in natural language. We are also planning to support a graphical representation of single proof trees as well as the graph of technical explanations.

Methods for obtaining justifications for conflicting answers. This feature is related to the phenomenon of inconsistencies in single and multiple data sources. Since the logic programming languages used in ATN are not expressive enough to derive inconsistent facts, this aspect is marginal in the ATN framework.

The heuristics for pruning proofs and focussing explanations may exploit the following metaproperties [Barzilay et al., 1998]:

Importance level of relations and attributes (which explanations should focus on);

Key attributes that identify individuals and may be used to denote them (internal identifiers are surely not user friendly);

Mutual dependencies among relations and attributes, stating that a given symbol should be presented only if another given symbol is included in the explanation;

Order in which the attributes of a concept should be presented, and

Second-order relations over the attributes of a concept.

In ATN the importance level is determined by the negotiation protocol, as we have already explained. We have developed a solution which is invariant across all application domains, therefore the framework specialization phase is not affected. For the same reason we do not use any domain-specific key attribute information to identify credentials and other structured objects; in Protune complex objects are characterized through the constraints they have to satisfy, as specified in rule bodies. Similarly, we adopt a uniform strategy for selecting the "auxiliary" predicates that must be included in explanations because some other predicate with a high importance level must be included, too.

We are currently finding second order relations over attributes unnecessary for our purposes.

4 Use Cases and Reference Scenario

In the following, the terms "user" and "server" may refer to any actor and any peer, respectively, because all peers may act as servers in some steps of the negotiation and as clients in other steps. So for example, a server's administrator may use the user version of a query for getting explanations on the counter-requests of the client.

4.1 Query types and their contexts

Use Case:	**1 - Why/Why-not queries for users**
Primary Actor:	User
Secondary Actor:	Personal Assistant (PA)
Preconditions:	A negotiation has taken place

Step	User	Personal Assistant (PA)
1	The user asks why a literal is successful/failed by clicking on either the explanation button activated after negotiation (first explanation request), or a link associated to a condition (during explanation browsing)	
2		The PA returns an explanation built using the following information • the final filtered policy • credentials and declarations released by the user during negotiation • a portion of the server's state • literal verbalization rules

Variation : *Some of the items in step 2 are not available*

2a		The PA asks the server for explanation related information, then proceeds as in step 2

Use Case:	**2 - How-to queries for users**
Primary Actor:	User
Secondary Actor:	Personal Assistant
Preconditions:	–

Step	User	Personal Assistant
1	The user asks how to satisfy a literal by selecting either the how-to explanation option associated to a service request (first explanation request), or the link associated to a condition (during explanation browsing)	
2		The PA returns an explanation built using the following information • a filtered policy • literal verbalization rules • expected outcome of state predicates • the user's portfolio

Variation : *First explanation request* (cf. step 1)

2a		The PA first sends a how-to request to the server, that returns the necessary information (first three items in step 2); then the PA proceeds as in step 2

Use Case:	**3 - What-if queries for users**
Primary Actor:	User
Secondary Actor:	Personal Assistant
Preconditions:	An explanation is being browsed

Step	User	Personal Assistant
1	The user selects the what-if option associated to a condition (during explanation browsing) and specifies an instance of that condition by choosing a value for each variable	
2		The PA adds the new fact to the filtered policy and recalculates the current explanation. If the user wishes to see the effects of the new assumption on the main goal, he or she may select the link pointing back to the root of the explanation hypertext

Administrator's use cases

Administrators may want to monitor and debug the server's access control policy. For this purpose it seems necessary to log the reasoning carried out by the server. Since it may be unfeasible to record all the transactions, the administrator should be able to specify which service requests have to be monitored (let's call it a "spy" facility, like in Prolog traces). Other cases may be supplied by clients in case of problems (as a consequence, the administrator may set a spy point on the request involved).

Use Case:	**4 - Why/Why-not queries for administrators**
Primary Actor:	Administrator
Secondary Actors:	Personal Assistant (PA), Server
Preconditions:	A negotiation has taken place and has been logged; The administrator is authenticated to the server

Step	Administrator	Personal Assistant (PA)	Server
1	The administrator selects a logged negotiation (first explanation request) or the link associated to a condition (during explanation browsing)		The server may have to let the administrator browse the log file
2		The PA returns an explanation built using the logged information: • final filtered policy • credentials and declarations released during negotiation • a portion of the server's state • literal verbalization rules	

Variation : *First explanation request:* (cf. step 1)

2a		The PA asks the server for explanation related information. Note: Verbalization rules are persistent and may be cached locally.	
3a			The server returns it
4a		The PA buids the explanation and displays it to the administrator	

Use Case:	**5 - How-to queries for administrators**
Primary Actor:	Administrator
Secondary Actors:	Personal Assistant (PA), Server
Preconditions:	–

Step	Administrator	Personal Assistant (PA)	Server
1	The administrator selects either the how-to explanation option associated to a service request (first explanation request), or the how-to link associated to a condition (explanation browsing)		
2		The PA returns an explanation built using: • all relevant policy rules • literal verbalization rules • expected outcome of state predicates	

Variation : *Explanation-related information not available*

2a		The PA first sends the how-to request and the administrator's credentials to the server	
3a			The server returns the necessary information (the items in step 2)
4a		PA proceeds as in step 2. Note that in this case the explanation-related information is persistent and typically small enough to be cached on the administrator's client.	

Use Case:	**6 - What-if queries for administrators**
Primary Actor:	Administrator
Secondary Actor:	Personal Assistant (PA)
Preconditions:	An explanation for administrators is being browsed

Step	Administrator	Personal Assistant (PA)
1	The administrator selects the what-if option associated to a condition (during explanation browsing) and specifies an instance of that condition by choosing a value for each variable	
2		The PA adds the new fact to the filtered policy and recalculates the current explanation. If the administrator wishes to see the effects of the new assumption on the main goal, he or she may select the link pointing back to the root of the explanation hypertext

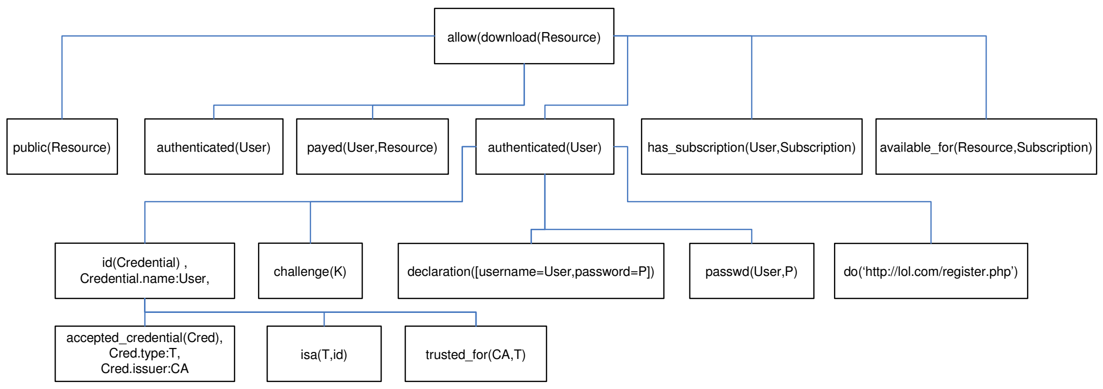
allow(download(Resource))
public(Resource)
authenticated(User)
payed(User,Resource)
authenticated(User)
has_subscription(User,Subscription)
available_for(Resource,Subscription)
id(Credential) , Credential.name:User,
challenge(K)
declaration([username=User,password=P])
passwd(User,P)
do('http://lol.com/register.php')
accepted_credential(Cred), Cred.type:T, Cred.issuer:CA
isa(T,id)
trusted_for(CA,T)

Figure 1: Overview of the digital library policy example from Appendix A

4.2 Running example

John Smith is a researcher at Open University and is currently working in a hot area. He is searching the internet for related work and finds a nice paper he would like to read. This paper is hosted at the website of a digital library under the file name "paper01234.pdf". However, not all the content of this digital library is accessible for free. In fact, the library service utilizes the policy specified in Appendix A (see figure 1 for an overview) to protect its resources. Since John Smith has never used the library service before, he does not know how to retrieve documents from there.

4.2.1 Finding out about a party's policy: How-to queries

While John Smith knows that probably not all of the papers are freely accessible, he does not yet know *how to* access the paper he is interested in. However, he has the possibility of asking the digital library about the policy applied to the action of downloading a paper and therefore, using his personal assistant agent, he sends the query "how-to : allow(download(Resource))" to the library.

The digital library receives the request, generates a *how-to explanation* and sends it back to John Smith's personal assistant. This explanation does not use any information about the requester (in fact the library still does not know anything about John or his personal assistant) but provides general information about the policy governing the requested service. The personal assistant is, among other tasks, in charge of the presentation of such explanations in a user-friendly and human-oriented way. As the library's policy might be too complex to be visualized at once, the personal assistant breaks the explanation down to manageable pieces to be displayed to John. Additionally, it provides hyper-links that allow John to navigate through those parts of the explanation he is most interested in.

For instance, the personal assistant may display the following information in response to John's how-to query:

```
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]
```

This explanation states that John can be granted access to a resource if:

- The resource is classified as public by the digital library. In such a case, John does not have to do anything and he is automatically authorized.
- John authenticates at the library services and if he has a subscription that covers the desired resource.
- John manages to authenticate and to pay for the desired resource.

Note that the structure of the explanation is given by a set of policy rules provided by the digital library (based on the policies from Appendix A) like:

$$
\begin{array}{l}
\texttt{allow}(\texttt{download}(\mathit{Resource})) \leftarrow \\
\quad \texttt{public}(\mathit{Resource})\,. \\
\texttt{allow}(\texttt{download}(\mathit{Resource})) \leftarrow \\
\quad \texttt{authenticated}(\mathit{User}), \\
\quad \texttt{has\_subscription}(\mathit{User}, \mathit{Subscription}), \\
\quad \texttt{available\_for}(\mathit{Resource}, \mathit{Subscription})\,. \\
\texttt{allow}(\texttt{download}(\mathit{Resource})) \leftarrow \\
\quad \texttt{authenticated}(\mathit{User}), \\
\quad \texttt{paid}(\mathit{User}, \mathit{Resource})\,.
\end{array}
$$

The personal assistant is in charge of generating the explanation and its concrete layout, reducing the burden on the library service.

The previous explanation provides a high level overview of the digital library's policy. However, John does not yet know exactly what he requires in order to "authenticate" and therefore he clicks on the "[details]" link accompanying that condition. The personal assistant then displays the requested part of the explanation:

```
TO MAKE SURE THAT User is authenticated

   PLEASE MAKE SURE THAT FOR SOME Credential and K
   Credential is an id with name User and public_key K   [details]
   AND
   respond to the challenge procedure for K              [info]

 ALTERNATIVELY

   PLEASE DECLARE THAT username=User and password=P      [info]
   WHERE
   P is the correct password of User

 ALTERNATIVELY

   PLEASE EXECUTE the registration procedure on
   http://lol.com/registration.php                       [info]
```

According to this explanation, there are three possibilities for John to authenticate at the library service: either (i) by providing an id credential (and by demonstrating the ownership of the id), (ii) by providing a username/password combination or (iii) by performing an online registration procedure.

One part of the digital policy, defining the authentication based on a username/password combination, is worth some additional elaboration: As shown below, the responsible rule says that after the username/password declaration is received, the security engine needs to check the provided information against the user database of the library service.

$$\begin{aligned}&\texttt{authenticated}(User) \leftarrow\\ &\quad \texttt{declaration}([username = User, password = P]),\\ &\quad \texttt{passwd}(User, P).\end{aligned}$$

However, the predicate `passwd` is defined as `private` by the meta policy, as shown below:

```
passwd(_,_)   .type          : state_predicate.
passwd(_,_)   .sensitivity   : private.
```

From the meta policy follows that the literal "`passwd(_,_)`" is blurred and not given to the personal assistant. However, explanations about the blurred conditions could still be generated if the *intended meaning* of the predicate itself is not private[1]. This means that we require a slight modification of the blurring mechanism presented in [Bonatti and Olmedilla, 2005b, Bonatti and Olmedilla, 2005a]. To generate such explanations, we require to relate some blurred predicates with the appropriate explanation and therefore blurred conditions should include this binding.

Continuing our example, John finally just wants to know a bit more about the id credentials he is expected to provide; clicking on the "[details]" link provides him with the following explanation:

[1]Note that the idea behind blurring is either to hide private information or to point out in a machine understandable way that there exist additional conditions that need to be checked by the server. Although the answer substitutions of such literals may be secret (and therefore not disclosable), the names and meaning of the predicates themselves may be communicated.

```
TO MAKE SURE THAT Credential is an id

   PLEASE MAKE SURE THAT
   you have submitted Credential with type T and issuer CA      [info]
   WHERE
   T is an id                                                   [details]
   AND
   CA is trusted for T                                          [details]
```

Thus he knows that the credential must be an id and its certification authority (CA) must be trusted for such type of credentials.

Remark 1 *It is not* necessary *to issue any how-to query before contacting a server for the first time. The filtered policy can be used directly by the PA to carry out an automated negotiation with the server. The filtered policy is in fact nothing but a machine-understandable how-to explanation.*

4.2.2 Investigating why a request failed: Why-not queries

John is now informed about the main requirements he must satisfy for accessing the paper he needs for his research. He decides to request access to the paper (via his personal assistant), providing an id credential. However, he receives the answer "permission denied" from the library service. He does not understand why, as he provided together with the request his id credential which is signed by the "Open University" CA. Fortunately, he knows that he can demand some extra information about his failed request. Therefore, he sends a *why-not* query to the service, using his personal assistant, which may either use a cached version of a why-not explanation or may send a fresh request (Appendix B.1 shows the state used at the digital library). In any case, the 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 in order to satisfy the policy, and hiding some other aspects of the policy (e.g., those requirements that the user did fulfill or those that do not depend on the user). To satisfy the individual information needs of its user, the personal assistant allows John to gather more detailed information by expanding the different parts of the explanation.

At first, the personal assistant may display the following why-not explanation:

```
I CAN'T PROVE THAT it is allowed to download paper01234.pdf BECAUSE:

   Rule [r3] is not applicable:
   THERE IS NO User SUCH THAT User is authenticated                   [details]

 AND

   Rule [r4] is not applicable:
   THERE IS NO User SUCH THAT User is authenticated                   [details]
   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 `r3` talks about subscriptions, but these details are omitted in the explanation above because if the user is not authenticated it makes no sense to inspect her subscription. However, John can also request the full explanation; in that case he would receive

```
TO PROVE THAT it is allowed to download paper01234.pdf ONE CAN TRY:

   Rule [r2]:
     Everybody can download public objects            [see internal format]
   the rule is not applicable [details]

   Rule [r3]: (no summary available)
     it is allowed to download paper01234.pdf IF
        THERE EXIST User AND Subscription SUCH THAT
        User is authenticated
        AND
        User has subscription Subscription
        AND
        paper01234.pdf is available for Subscription
   the rule is not applicable [details]

   Rule [r4]:
     Users can download any object if they pay for it [see internal format]
   the rule is not applicable [details]
```

John did disclose his id credential and wonders why the authentication did still fail. Hence he clicks on the "[details]" link of rule r3 in order to find out why the rule did not succeed. The personal assistant expands that explanation with

```
I CAN'T FIND ANY User SUCH THAT User is authenticated BECAUSE:

   Rule [r6] is not applicable:
   THERE IS NO Credential SUCH THAT Credential is an id               [details]

 AND

   Rule [r7] is not applicable:
   THERE ARE NO User AND P SUCH THAT it has been declared that
      username=User and password=P

 AND

   Rule [r8] is not applicable:
   the registration procedure on http://lol.com/registration.php
      has not been (successfully) completed
```

(again, the explanations for `r6` and `r7` omit part of the rule body to improve explanations).

It is true that John did not provide a username/password combination, as he does not possess one, neither did he register at the URL provided. However, he did disclose his id credential and wonders why his request was not accepted. Therefore, John expands the relevant part of the explanation, receiving

```
I CAN'T FIND ANY Cred SUCH THAT Cred is an id BECAUSE:

   c012 is a credential with type id and issuer Open University   [details]
   id is an id_type                                               [details]
   BUT
   IT IS NOT THE CASE THAT Open University is trusted for id      [details]
```

Now John finally knows why his request has not been accepted. The reason is that the certificate authority (CA) 'Open University' of his id credential is not among the trusted CAs for id credentials at the library service.

Note that the credential is not denoted only by means of its internal identifier `c012` (which is not particularly meaningful to the user); the attributes of the credential and their properties explain which credential is being referred to, which may be helpful when multiple credentials have already been released.

4.2.3 Exploring hypothetical possibilities: what-if queries

In previous steps John tried to access a service but he was informed, by means of the explanation facility, about why his request was denied. The reason was that his credential was not accepted because "Open University" is not a trusted CA at the digital library. Fortunately, John has the possibility to use an id from the "UK Government" but before he gathers it (and discloses it), he would like to know whether this credential would be accepted[2]. Therefore, he submits a what-if query together with a *mockup credential* from "UK Government". In this case, the evaluation of the query at the digital library receives a hypothetical credential and produces a hypothetical answer. That means that even if the answer returns that access would be granted, it is still not 100% guaranteed to succeed in a real negotiation, because the simulation is based on several assumptions (e.g., the digital library assumes that the challenge for the credential will succeed during a real negotiation).

Assume that the state contains the fact `has_subscription('J.Smith',computer(full))`. Then the personal assistant displays the following explanation to John:

```
IT MIGHT BE TRUE THAT it is allowed to download paper01234.pdf BECAUSE

    Rule [r3] might be applicable:
    BY ASSUMPTION
      J. Smith is authenticated                           [details]
    AND THERE MAY EXIST Subscription SUCH THAT
        J. Smith has subscription Subscription            [details]
    AND
        paper01234.pdf is available for Subscription      [details]
```

[2]Note that in the previous why-not query John might continue expanding rules in order to find which are the trusted CA for the digital library. This is possible due to the fact that the predicate "trusted_for" is public. Otherwise, this information would not be available in a why-not query.

In this case, the state predicates `has_subscription` and `available_for` cannot be better explained or even checked for a common solution because the extension of `has_subscription` is private.

This *concise what-if* explanation focuses, as in why/why-not explanations, on successful conditions (if the final answer is positive) or on unsuccessful conditions (if negative). In our case, the explanation shows that John would probably be granted access if he provided such a credential. The alternative possibility involving payment is not shown because John did not provide a real or hypothetical (i.e. mockup) credential of his credit card and therefore it is not assumed that it exists.

If John wants to check the requirements for authentication in presence of hypothetical credentials, he can expand that information into

```
IT WOULD PROBABLY HOLD THAT J. Smith is authenticated BECAUSE

    Rule [r6] would probably be applicable:
    BY ASSUMPTION
      c012 is an id WITH name J. Smith and public_key a7s687dfghg
    AND IT WOULD PROBABLY HOLD THAT
      the challenge procedure for a7s687dfghg is successfully completed
```

which uses the hypothetical id credential provided and assumes, based on the meta policy rule

```
challenge(_)  .expected_outcome : success.
```

that the ownership challenge will be successfully completed. The explanation does not include the other two possibilities of authentication due to the fact that no declaration of username and password has been provided and that the expected outcome of a “do” operation is “failure”.

4.2.4 Checking why a request succeeded: why queries

In the previous step, John learned that an id from “UK Government” would be accepted by the digital library. Therefore, he includes his real credential in a new request that is sent to the library. In this case, he is granted access to the paper and is able to download it. Still he might be interested in receiving information about the process that led to the desired result. This is particularly useful if different paths are followed in parallel during policy evaluation (e.g., if a user tries to reach authentication based on a credential and based on a declaration at the same time, in order to speed up the negotiation in case one of the options fails), in order to get a proof of the request or just for information purposes (e.g., to double-check that the process was executed as expected or to gather more insight on the policy in case no how-to, why-not or what-if queries have been sent previously).

Therefore, John’s personal assistant provides the following “why”-information:

```
it is allowed to download paper01234.pdf BECAUSE:

   Rule [r3] can be applied:
   RELEASED INFORMATION GUARANTEES THAT
     J. Smith is authenticated                      [details]
   FURTHERMORE
     J. Smith has subscription computer(full)       [details]
```

```
AND
  paper01234.pdf is available for computer(full)    [details]
```

whereby he may expand the explanation to gather more focused or more advanced information.

5 Explanations and filtering

The policy filtering process carried out by Protune [Bonatti and Olmedilla, 2005b, Bonatti and Olmedilla, 2005a] changes the structure of the policy so much that it would be difficult to formulate satisfactory explanations given a filtered policy as an input. In particular:

- partial evaluation removes from the rule body all the state conditions that can be immediately evaluated;
- blurring hides all the deferred state conditions in such a way that the blurred condition cannot possibly be matched to any policy literal.

Therefore, it is practically impossible to explain why a credential or a declaration meets or violates a state condition.

Fortunately, there is space for improvement. It turns out that in most cases literals are blurred because their extension is too large, or because it is confidential; however the user may be safely told what the blurred condition consists of. This does not mean that we can avoid blurring; we need to mark the condition as "blurred" to let the personal assistant understand that some condition without a machine-understandable semantics will be checked by the other peer (this kind of information may be useful during the credential selection phase).

Furthermore, it is frequently possible to evaluate a specific ground instance of a sensitive literal; only the set of answers in its *entirety* should be kept confidential. As an example, consider the login-passord check: authentication systems say whether a specific pair of strings is correct, but they would never disclose the password file. This means that in some cases, the filtered policy may safely include some specific facts about protected literals. A similar example is given by `has_subscription` in our policy. It is legal to check whether a specific authenticated user has a given subscription, but privacy reasons forbid to release the extension of this predicate.

Therefore, a refined filtering process should be adopted, with the following distinctive features:

- immediately evaluable state literals are not partially evaluated; their solutions are included in the policy as sets of facts;[3]
- a blurred literal L is normally replaced with `blurred`(L) to make the original literal available to the explanation mechanism; if L is not to be explained (for confidentiality reasons), then L is replaced with `blurred`(n) where n is a random number.
- the final response may include some facts with the shape `blurred`(L) and `blurred`(n) to tell the client which blurred conditions were successful.

[3]A minor technicality: a negative literal `not` A may have to be treated like a new atom `not_`A to be asserted like a fact or be made to fail in case there are some solutions that should not be included in the policy.

One word is needed on including facts like `blurred`(n) in the filtered policy. In this way, the client does not know which condition is tested, but it can automatically detect which proof branches succeed. This makes it possible to identify interesting information, such as the minimal sets of credentials that were really needed to make the negotiation succeed. However, the metapolicy may forbid the disclosure of such facts, if necessary.

The refined filtering process has a few additional advantages. First, the number of rules is not increased as a consequence of the partial evaluation of state literals. In some cases, the filtered policy may be more compact with the new method. Second, the computational load on the server may be reduced because less inferences are needed for partial evaluation.

We expect service users to be interested in explanations about the server's policy, while service administrators are not likely to ask for explanations about the counter-requests of the clients (with possible exceptions during an initial debugging phase). This means that clients might still adopt the original filtering process for their counter-request, to reduce the number of inference steps on the server.

6 Explanation Structures

We assume the reader to be familiar with the basics of logic programming, including the notions of *most general unifier* (mgu), *free variable*, and *(computed) answer substitution.* The reader is referred to [Lloyd, 1984, Apt, 1990] for these matters. As usual, we denote by $\mathsf{var}(E)$ the variables occurring in an expression E (be it a term, a literal, a goal or a rule). By $\mathsf{outvar}(\sigma)$, where σ is a substitution, we denote the set of variables that occur in the range of σ—also called *output variables* in the following. Finally, recall that if two expressions or sets thereof are identical up to variable renaming, then they are said to be *variants.* We say that two substitutions are *variants* if they are more general than each other (that is, they are identical up to output variable renaming).

There are some delicate technical aspects in handling substitutions. Each of them is a variant of infinitely many other substitutions. Moreover, during deduction, unification should always produce fresh variables to avoid overconstraining (unnecessary identification of individuals). So we assume that there exists a partial function $\mathsf{mgu}^E(E_1, E_2)$ that for all expressions E, E_1, and E_2 returns one mgu σ of E_1 and E_2, such that

$$\mathsf{outvar}(\sigma) \cap (\mathsf{var}(E) \cup \mathsf{var}(E_1) \cup \mathsf{var}(E_2)) = \emptyset\,.$$

Similarly, we assume a function $\mathsf{ans}_P^E(G)$ that for all programs P, all goals G, and all expressions E returns a set containing exactly one variant σ for each answer substitution of G from P, such that

$$\mathsf{outvar}(\sigma) \cap (\mathsf{var}(E) \cup \mathsf{var}(P) \cup \mathsf{var}(G)) = \emptyset\,.$$

6.1 Partially specified programs and quasi-answers

Explanations must be built using an incomplete view of state predicates, that are not fully defined due to confidentiality and efficiency constraints (see the discussion of *blurring* in [Bonatti and Olmedilla, 2005b]). We extend inference to handle conclusions that *might* be drawn, depending on the truth value of the unspecified predicates.

Definition 1 *A* partially specified program *is a pair $\Pi = (P, U)$ where P is a logic program and U is a set of so-called* unspecified literals*.*

In practice, U would be the set of blurred literals. Next we have to characterize the solutions that might be derived if unspecified literals were available.

Definition 2 *A* quasi-answer *of a goal G w.r.t. a partially specified program $\Pi = (P, U)$ is a substitution σ such that there exists an SLD-derivation $G, G_1, \ldots, G_n$ from P with mgu's $\theta_1, \ldots, \theta_n$, where all the atoms occurring in G_n belong to U and such that $\sigma = \theta_1 \cdots \theta_n$.*

By analogy with answer substitutions, we assume a function $\mathsf{qans}^E_\Pi(G)$ that returns a set containing one variant σ for each quasi-answer of A w.r.t. Π, such that

$$\mathsf{outvar}(\sigma) \cap (\mathsf{var}(E) \cup \mathsf{var}(\Pi) \cup \mathsf{var}(G)) = \emptyset\,.$$

In the following we shall sometimes abuse notation and write ans^E_Π instead of ans^E_P.

Note that the set of quasi answers includes the set of answers, because G_n may be the empty goal. Accordingly, we assume w.l.o.g. that $\mathsf{qans}^E_\Pi(G) \supseteq \mathsf{ans}^E_\Pi(G)$. We say a quasi-answer is *proper* if G_n is nonempty.

6.2 Level 2 (technical) explanations

Level 2 explanations are meant to provide the full details of proofs and proof attempts. They can be seen as graphs (abstracting a hypertext) where each node illustrates:

- the local context for a goal, that is, the rules whose head unifies with that goal;
- a global view of all the proof attempts, consisting in the answers and quasi-answers of each rule body and subgoal thereof.

Such answers and quasi-answers provide a sort of *lookahead* that may help the user in navigating the proof. We proceed by defining the explanation graph.

Definition 3 *An* explanation node *for a partially specified program $\Pi = (P, U)$ is a finite set of pairs (r, θ) where $r \in P$ and θ is a substitution.*

Some of these explanation nodes constitute the starting point for explaining how a given atom A succeeds or fails. Basically, such *entry points* collect all the applicable rules:

Definition 4 *The* explanation entry point *for A w.r.t. Π, denoted by $\mathsf{entry}_\Pi(A)$, is the (unique) explanation node X for Π such that*

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

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 some of the possible proofs.

Definition 5 (Level 2 navigation links) *For all explanation nodes X_1 and X_2 for Π define:*

detail links: *$X_1 \xrightarrow{L}_D X_2$ iff for some $(r, \theta) \in X_1$ and some $L \in \mathsf{body}(r)$, $X_2 = \mathsf{entry}_\Pi(L\theta)$;*

refinement links: *$X_1 \xrightarrow{\sigma}_R X_2$ iff for some $(r, \theta) \in X_1$ and some $L \in \mathsf{body}(r)$,*

$$\sigma \in \mathsf{ans}^r_\Pi(L\theta) \cup \mathsf{qans}^r_\Pi(L\theta) \text{ and } X_2 = \{(r, \theta\sigma)\}.$$

Example 1 In the following we revisit our example presented in Sect. 4.2. Consider John's initial query "how-to : allow(download(Resource))". Given the policy used by the online library service (cf. Appendix A), the explanation entry point for the requested literal is:

$$\mathsf{entry}_\Pi(\mathit{download}(\mathit{Resource})) = \{(r_2, \varepsilon), (r_3, \varepsilon), (r_4, \varepsilon)\}$$

whereby ε denotes the empty substitution.

The content of this explanation entry point is presented to the user, who may choose to navigate through the explanation structure to gather more information about the policy. In particular, the user may follow a *detail link* to find out the possible ways of satisfying the literal $L_{auth} = \mathit{authenticated}(User)$ required by r_3 and r_4:

$$\mathsf{entry}_\Pi(\mathit{download}(\mathit{Resource})) \xrightarrow{L_{auth}}_D \{(r_6, \varepsilon), (r_7, \varepsilon), (r_8, \varepsilon)\}$$

Next, John may inspect the consequences of applying substitution

$$\sigma_U = \{User/\texttt{'J. Smith'}\}$$

which binds the variable $User$ occurring in the rules r_3 to a particular value, in order to traverse the *refinement link*

$$\mathsf{entry}_\Pi(\mathit{download}(\mathit{Resource})) \xrightarrow{\sigma_U}_R \{(r_3, \sigma_U)\}$$

In this way, John has reached the rule:

```
allow( download(Resource) ) :-
        authenticated('J. Smith'),
        has_subscription('J. Smith',Subscription),
        available_for(Resource,Subscription).
```

□

The navigation structure is modeled by the following graph:

Definition 6 *A* level 2 explanation graph *for an atom A w.r.t. a partially specified program* Π *is a minimal structure (V, E^D, E^R) closed under the following properties:*

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

Note that explanation graphs must be unique up to variable renaming. Moreover, if the program is Datalog, then the explanation graph is finite. In practice, the explanation graph needs not necessarily be computed in advance; it may be generated on demand, as the user navigates it.

Now the complete explanation structure is obtained by adding the labels to the graph:

Definition 7 *A* level 2 explanation structure *for A w.r.t. Π is a tuple (XG, s, qs) where:*

- *XG is a level 2 explanation graph for A w.r.t. Π; let V be the set of vertices of XG;*
- *(success label) $s = \mathsf{ans}_{\Pi}^{V}$;*
- *(quasi-success label) $qs = \mathsf{qans}_{\Pi}^{V}$.*

Now given a node element (r, θ), the answers and quasi-answers labelling the rule are given by $s(\mathsf{body}(r))$ and $qs(\mathsf{body}(r))$, and the labels of each subgoal $L \in \mathsf{body}(r)$ are given by $s(L)$ and $qs(L)$.

Remark 2 In the presence of infinite failures, the data structure is cyclic but still finite (because the policy is a Datalog program). To help the user in identifying those cycles, the hypertext rendering of the graph may label the backward links that close a cycle in a particular way (*deja vu* links).

Remark 3 Note the similarity between the explanation graph and the tables of Prolog engines like XSB. In both cases, subgoal calls become entries in a potentially cyclic data structure. In both cases, there is a unique entry for all goal variants so that infinite computations are finitely represented by graph cycles. In both cases the entries are labelled with answer substitutions. A similar approach, where the set U of unspecified literals is empty and the function $qs()$ can be dropped, may be interesting for tracing tabled programs. It would be interesting to study an efficient implementation based on XSB's primitives for handling tables.

6.3 Level 1 (concise) explanations

Well-crafted explanations focus on the details that are relevant to the user. Since ATN is mostly about asking for credentials, declarations, and actions, a concise explanation should be centered around these predicates and the abbreviations that depend on these predicates, in order to highlight which requests have been fulfilled by the client and which have not.

For this purpose we adopt a standard *(predicate) dependency graph*, which is a graph $G = (V, E)$ where V (the set of vertices) is the set of predicates of the given program P, and E the set of edges (p, q) such that for some rule $r \in P$, p occurs in $\mathsf{head}(r)$ and q occurs in $\mathsf{body}(r)$. As usual, we say that a node p *depends* on another node q if there is a directed path from p to q in the dependency graph.

Definition 8 *A predicate is* user-dependent *if it depends on some of* `credential`, `declaration`, *or* `do`. *If this is the case then we call the predicate a* ud-predicate. *If a literal L contains an occurrence of a ud-predicate, then we say that L is a user-dependent literal (ud-literal, for short).*

One problem to be solved is: how can explanations refer to structured objects, such as credentials? Typically these objects are denoted by means of internal identifiers (sometimes called *handles*) that have no meaning to the common user. However, we noted that the attribute atoms $t.a{:}v$ in a rule body are typically the relevant, characterizing attributes of the complex object whose handle is t. These attributes almost always identify a unique object, when their values are fixed. So our idea is using these attributes as a key to specify which object we are referring to. For this purpose, we look for subsets of the body called *clusters* that correpond to concepts with attributes (a sort of terminological expression) and treat them as a description of the complex object.

Definition 9 (Clusters) *Let $L = p(t)$ where p is a unary predicate and t is a term. If $L \in \mathsf{body}(r)$, then a* cluster *of L (in r) is a set containing L and some attribute subgoals $(u.a : v) \in \mathsf{body}(r)$ 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. A* user-dependent *cluster (ud-cluster, for short), is a cluster whose main predicate is user-dependent.*

Example 2 The policy rule instance

```
authenticated('J. Smith') :-
    id(Credential),
    Credential.name:'J. Smith',
    Credential.public_key:K,
    challenge(K).
```

contains one complete cluster (the first three literals in the body) whose main literal is `id(Credential)`. The cluster denotes "the id with name 'J. Smith' and public key K". $\square$

Remark 4 Clusters are more flexible than key attributes and reduce the framework instantiation effort, because they constitute a generic technique that needs no preliminary manual intervention (on the contrary, key attributes must be specified by the knowledge engineer). Clusters are more flexible because they can be applied also to those classes of objects that have no key attributes. Even if several objects match the attribute values, still these objects are at least partially characterized, while keys work in an all-or-nothing fashion. Partial characterization is 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 $k012$*".

While a single atom like $\mathtt{credential}(X)$ 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 values of the cluster variables, and hence the credential we are talking about. Similarly, any literal with a single answer substitution may be exploited to fix the values of variables and determine more precisely the context in which the conditions are evaluated. The process of applying unique answers is formalized as follows.

Definition 10 *A* unique-answer propagation *of a node element (r, θ) for a given Π is a pair (r, θ') such that, for some $n \leq |\mathsf{body}(r)|$, the following conditions hold: for each $i = 1, \ldots, n$, there exist a substitution θ_i, a literal or a cluster Y_i, and a substitution σ_i such that:*

1. *$\theta = \theta_1$ and $\theta' = \theta_n$;*
2. *$\mathsf{ans}_{\Pi}^{r\theta_i}(Y_i) = \{\sigma_i\}$ $(i < n)$;*
3. *$\theta_{i+1} = \theta_i \sigma_i$ $(i < n)$;*
4. *for all clusters and literals Z of $\mathsf{body}(r)$ not occurring in $Y_1, \ldots, Y_n$, $|\mathsf{ans}_{\Pi}^{r\theta_n}(Z\theta_n)| \neq 1$.*

We call $Y_1, \ldots, Y_n$ the rewrite sequence *of (r, θ').*

In other words, condition 2 ensures that only unique answers are applied during the rewriting, and condition 4 ensures that all unique answers are applied, i.e., the rewrite process is exhaustive. As a consequence, it is not hard to see that if (r, θ') is a unique-answer propagation, then for all clusters and literals Z such that $|s_n(Z)| = 1$, the unique element of $s_n(Z)$ must be equivalent to the empty substitution ε.

Note that if some instance of $\mathsf{body}(r\theta)$ is a consequence of P, then unique-answer propagations must be unique up to variable renaming. On the contrary, if $\mathsf{body}(r\theta)$ fails, then an annotated rule may have multiple unique-answer propagations incompatible with each other, obtained from the answers of different maximal derivable subsets of the body.

In this case, we choose each step of the rewrite sequence according to the following priority order among the (not yet processed) literals and clusters with a single answer substitution (from the highest priority to the least):

- First we choose complete ud-clusters. This strategy tends to describe in detail the structured objects released by the user, such as credentials and declarations.
- Next we choose incomplete ud-clusters. This strategy tends to describe the structured objects released by the user that "almost satisfy" policy requirements.
- Then we choose the other ud-literals. Again, we are giving precedence to the facts that are under the responsibility of the user.
- Next we choose the other clusters, in order to describe the structured objects contained in the server's state.
- Finally we choose the remaining (non-ud) literals.

Moreover, if two clusters or literals have the same priority, we are currently adopting a greedy left-to-right strategy to make a choice. We write

$$(r, \theta) \longrightarrow_U (r, \theta')$$

when (r, θ') is the unique-answer propagation of (r, θ) according to the adopted deterministic choice. The left-hand side of $\longrightarrow_U$ uniquely determines the right-hand side, up to variable renaming.

Remark 5 At the abstract level we are adopting in this section, any deterministic choice of the rewrite sequences can be adopted as the semantics of $\longrightarrow_U$. In other words, the framework can be adapted to heuristics different from the above one just by changing the specification of $\longrightarrow_U$.

Now we proceed with the definition of the navigation structure, by analogy with level 2 explanations.

Concise explanations are aimed at avoiding irrelevant and redundant information. As a consequence we are filtering the set of answer substitutions to extract only those that are maximally general. Assume a function $\mathsf{mg}(S)$ that for all sets of substitutions S returns the maximally general elements of S. Moreover, level 1 links lead directly to nodes where unique answers have been propagated. 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 11 (Level 1 navigation link) *For all explanation nodes X_1 and X_2 for Π define:*

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

refinement links: $X_1 \overset{\sigma}{\leadsto}_R X_2$ *iff for some* $(r, \theta) \in X_1$ *and some* $L \in \mathsf{body}(r)$,

$$\sigma \in \mathsf{mg}(\mathsf{ans}^r_\Pi(L\theta)) \cup \mathsf{mg}(\mathsf{qans}^r_\Pi(L\theta)) \text{ and } \{(r, \theta\sigma)\} \longrightarrow_U X_2.$$

Example 3 In the following we continue our running example using the state illustrated in Appendix B.2, that makes `authenticated('J. Smith')` true. Suppose that from the entry point for `allow(download(paper01234.pdf))` John selects the solution σ of `authenticated(User)` in rule r_3 that binds `User` to `'J. Smith'`. Next, John asks for the details of the proof of $L = \mathtt{authenticated('J.\ Smith')}$.

Roughly speaking, John has just followed the path:

$$\begin{array}{lll} \mathsf{entry}_\Pi(\mathtt{allow(download(paper01234.pdf))}) & \overset{\sigma}{\leadsto}_R & \{(r_3, \theta\sigma)\} \\ & \longrightarrow & \{(r_6, \{\mathtt{User/'J.\ Smith'}\})\} \end{array}$$

where $\theta = \{\mathtt{Resource/paper01234.pdf}\}$. The last node corresponds to the following rule instance:

```
authenticated('J. Smith') :-
      id(Credential),
      Credential.name:'J. Smith',
      Credential.public_key:K,
      challenge(K).
```

Now, `id(Credential)` has a unique solution $\sigma' = \{\mathtt{Credential/c012}\}$. After applying it to the above rule, we can find other literals with a unique solution (all the remaining literals). So the node $\{(r_6, \{\mathtt{User/'J.\ Smith'}\})\}$ is not the final target of the detail link coming from $\{(r_3, \theta\sigma)\}$; first we have to apply the above unique susbstitutions, and then we obtain the actual formalization of John's navigation:

$$\begin{array}{lll} \mathsf{entry}_\Pi(\mathtt{allow(download(paper01234.pdf))}) & \overset{\sigma}{\leadsto}_R & \{(r_3, \theta\sigma)\} \\ & \overset{L}{\leadsto}_D & \{(r_6, \sigma'')\}\} \end{array}$$

where σ'' = { `User/'J.\ Smith', Credential/c012, K/'HJK789?%^&'` }. The last node corresponds to the rule:

```
authenticated('J. Smith') :-
        id(c012),
        c012.name:'J. Smith',
        c012.public_key:'HJK789?%^&',
        challenge('HJK789?%^&').
```

If the state contained more than one valid credential, then `id(Credential)` might have different solutions, e.g., credentials with different names. Still, the (nonmaximal) cluster $\{\mathtt{id(Credential)}, \mathtt{Credential.name:'\ J.Smith'}\}$ might have a unique solution and lead to the same result. □

Definition 12 *A* level 1 explanation graph *for an atom A w.r.t. a partially specified program* Π *is a minimal structure* (V, E^D, E^R) *closed under the following properties:*

1. *V contains* $\mathsf{entry}_\Pi(A)$*;*

2. *if for some $X_1 \in V$, $X_1 \overset{L}{\leadsto}_D X_2$ 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 $X_1 \in V$, $X_1 \overset{\sigma}{\leadsto}_R X_2$ then V contains exactly one variant $\tilde{X}_2$ of X_2, and E^R contains an edge $(X_1, \sigma, \tilde{X}_2)$.*

The complete explanation structure is obtained by adding concise labels to the graph:

Definition 13 *A* level 1 explanation structure *for A w.r.t. Π is a tuple (XG, s, qs) where:*

- *XG is a level 1 explanation graph for A w.r.t. Π; let V be the set of vertices of XG;*
- *(success label) $s = \mathsf{mg} \circ \mathsf{ans}_\Pi^V$,*
- *(quasi-success label) $qs = \mathsf{mg} \circ \mathsf{qans}_\Pi^V$,*

(where $\circ$ denotes function composition.)

Recall that verbalization patterns have *slots* that are filled in with subgoals according to the number of their answers and quasi-answers. The appropriate selection conditions for each of these slots are formally defined as follows.

Definition 14 (Slots) *Let $\mathcal{X} = (XG, s, qs)$ be a level 1 explanation structure. For all pairs $\rho = (r, \theta)$ occurring in a node of XG let:*

- $\mathsf{single}_\mathcal{X}(\rho) = \{L \mid L \in \mathsf{body}(r)$ *and* $|s(L\theta)| = 1\}$*;*
- $\mathsf{maxsingle}_\mathcal{X}(\rho) = \max_\subseteq\{C \mid C \subseteq \mathsf{body}(r)$ *is a cluster and* $|s(C\theta)| = 1\}$*;*
- $\mathsf{multi}_\mathcal{X}(\rho) = \{L \mid L \in \mathsf{body}(r)$ *and* $|s(L\theta)| > 1\}$*;*
- $\mathsf{failed}_\mathcal{X}(\rho) = \{L \mid L \in \mathsf{body}(r)$ *and* $|s(L\theta)| = 0\}$*;*
- $\mathsf{minfailed}_\mathcal{X}(\rho) = \min_\subseteq\{C \mid C \subseteq \mathsf{body}(r)$ *is a cluster and* $|s(C\theta)| = 0\}$*;*
- $\mathsf{quasi}_\mathcal{X}(\rho) = \{L \mid L \in \mathsf{body}(r),\ |s(L\theta)| = 0,$ *and* $|qs(L\theta)| > 0\}$*;*
- $\mathsf{strongly\_failed}_\mathcal{X}(\rho) = \{L \mid L \in \mathsf{body}(r)$ *and* $|s(L\theta)| = |qs(L\theta)| = 0\}$.

Moreover, if S is a set of literals or clusters, we denote by $\mathsf{ud}(S)$ *the set of all user-dependent elements of S.*

The subscript $\mathcal{X}$ will be omitted when it is clear from the context. Note that level 1 nodes are closed under unique-answer propagation; consequently, the unique answer of the literals in $\mathsf{single}(\rho)$ is always (a variant of) the empty substitution ε.

7 Verbalization

Explanation structures are translated into natural language sentences by instantiating predefined text patterns.

Some of these patterns are application independent, and cast into the explanation construction algorithm as shown in the rest of this section. They are based on the meaning of logical connectives and on the intended semantics of basic predicates such as `credential`, `declaration`, and `do`. These patterns are associated statically to the "slots" of explanation structures, and to the aforementioned special predicates.

The verbalization patterns for application dependent literals must be supplied during the framework instantiation phase. They are typically formulated with simple metafacts like:

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

$$\texttt{has\_subscription(X,Y).explanation} : [\texttt{X}, \texttt{has}, \texttt{subscription}, \texttt{Y}]\,.$$

If necessary, proper rules (with nonempty body) can be used.

It is also possible to define text patterns for negative literals, for example:

$$(\texttt{not is\_authenticated(X)}).\texttt{explanation} : [\texttt{X}, \texttt{is}, \texttt{not}, \texttt{authenticated}]$$

In the absence of such patterns, negative literals are verbalized by prefixing the pattern for the atom with `IT IS NOT THE CASE THAT`... (for ground literals) or `THERE IS/ARE NO` <*variable list*> `SUCH THAT`... (for nonground literals). By default, variable names are taken from the original rule—which is the first element of the current node (r, θ).

Remark 6 This kind of metafacts can be automatically or semi-automatically generated from the parse trees generated by the natural language front-end. In most cases, verb phrases immediately generate the value of the explanation attribute. The corresponding predicate names can be obtained by concatenating the same words. Using grammar rules, the dual, negative version of the explanation attribute can be obtained automatically, too. The knowledge engineer can override the automated proposal of the system when required.

7.1 Application independent verbalization

Application independent patterns share a common structure:

- an *incipit* summarizing what is being proved or disproved;
- a sequence of *rule verbalizations* that depend on the query modality (why, why not, etc.);
- a *separator* between each pair of contiguous rule verbalizations; the separator depends on the query modality.

A uniform treatment of quantification is adopted. Let the *local variables* of a rule r be the variables that occur only in the body of r. Local variables are existentially quantified. At the beginning of each slot verbalization, the local variables that appear for the first time in that slot are quantified with expressions such as `THERE EXIST` ... and `FOR SOME` ... depending on the context. Variable names are copied from the original rule as explained above.

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

$\mathsf{verb}(L_1\theta)$ `WITH` a_1 v_1`,` a_2 v_2`, ...,` `AND` a_n v_n.

Note that different clusters may share the same main term. Therefore, to avoid redundancy, the attribute atoms that have already been verbalized in a previous cluster are omitted, unless the cluster being verbalized belongs to the Failure Slot of *why not* queries.

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

- an arbitrary but fixed explanation structure $\mathcal{X} = (XG, s, qs)$ of the appropriate level;
- a node $X = \{\rho_1, \ldots, \rho_z\}$ of XG (the current node), where $\rho_i = (r_i, \theta_i)$ $(1 \leq i \leq z)$;
- an atom A (the one being currently explained), which is more general of all $\mathsf{head}(r_i\theta_i)$ $(1 \leq i \leq z)$.

For all clusters or literals Y and for all substitutions θ, the verbalization of $Y\theta$ will be denoted by $\mathsf{verb}(Y, \theta)$. For the sake of simplicity, the navigation links of level 1 queries will not be shown.

Detailed (level 2) queries

incipit	`TO PROVE THAT` $\mathsf{verb}(A, \varepsilon)$ `ONE CAN TRY`
verbalized rules	all $(r_i, \theta_i) \in X$
separator	<*blank line*>

Each verbalized rule (r_i, θ_i) is formatted according to the following pattern:

```
Rule <rule identifier>:
      <rule summary>
      verb(head(r_i), θ_i) IF
            <Body Slot>
<rule status>
```

where rule summaries can be either provided manually during the instantiation phase, or filled in with the natural language source of the rule.

The Body Slot is verbalized as follows:

```
<quantification>
verb(Y_1, θ_i) <Y_1 status>
AND
           ⋮
AND verb(Y_m, θ_i) <Y_m status>
AND verb(L_1, θ_i) <L_1 status>
           ⋮
AND verb(L_n, θ_i) <L_n status>
```

where $Y_1 \ldots Y_m$ are all the complete clusters in $\mathsf{body}(r_i)$, and $L_1 \ldots L_n$ are all the literals in $\mathsf{body}(r_i)$ that do not occur in $Y_1 \ldots Y_m$.

The rule status verbalization is

- `the rule is applicable` if $s(\mathsf{body}(r_i) \neq \emptyset)$;

- `the rule is not applicable` if $qs(\mathsf{body}(r_i) = \emptyset)$;
- `the rule might be applicable` if $s(\mathsf{body}(r_i) = \emptyset)$ and $qs(\mathsf{body}(r_i) \neq \emptyset)$

The status of a cluster or literal Z is visualized as

- `(it has [solutions] [details])` if $s(\mathsf{body}(r_i) \neq \emptyset)$;
- `(it fails [details])` if $qs(\mathsf{body}(r_i) = \emptyset)$;
- `(it might [succeed] [details])` if $s(\mathsf{body}(r_i) = \emptyset)$ and $qs(\mathsf{body}(r_i) \neq \emptyset)$

where the square brackets identify hypertext links.

"Why" queries (level 1)

incipit	$\mathsf{verb}(A, \varepsilon)$ `BECAUSE:`
verbalized rules	all the $(r_i, \theta_i) \in X$ such that $s(\mathsf{body}(r_i\theta_i)) \neq \emptyset$ (applicable rules)
separator	`AND, ALTERNATIVELY,`

Each verbalized rule (r_i, θ_i) is formatted according to the following pattern:

```
Rule <rule identifier> can be applied:
    <UD Slot>
    FURTHERMORE
    <UI Slot>
```

where `FURTHERMORE` is omitted if any of the two slots is missing. The UD (User Dependent) Slot in its most complete version is verbalized as:

```
RELEASED INFORMATION GUARANTEES THAT <quantification>
verb(Y_1,θ_i) AND ...AND verb(Y_m,θ_i) AND verb(L_1,θ_i) AND ...AND verb(L_n,θ_i)
AND FOR SOME <var list> verb(L'_1,θ_i) AND ...AND verb(L'_l,θ_i)
```

where $Y_1 \dots Y_m$ are the ud-clusters in $\mathsf{maxsingle}(\rho_i)$, $L_1 \dots L_n$ are the ud-literals in $\mathsf{single}(\rho_i)$ that do not occur in $Y_1 \dots Y_m$, and $L''_1 \dots L''_h$ are the ud-literals in $\mathsf{multi}(\rho_i)$ that do not occur in $Y_1 \dots Y_m$. Clearly, if any of these classes of clusters and literals is empty, then the corresponding pattern is removed (the same holds in the rest of the section, for all queries).

Similarly, the UI (User Independent) Slot is verbalized as

```
verb(Ŷ_1,θ_i) AND ...AND verb(Ŷ_j,θ_i) AND verb(L̂_1,θ_i) AND ...AND verb(L̂_k,θ_i)
AND FOR SOME <var list> verb(L̂'_1,θ_i) AND ...AND verb(L̂'_h,θ_i)
```

where $\hat{Y}_1 \dots \hat{Y}_j$ are the clusters in $\mathsf{maxsingle}(\rho_i) \setminus \{Y_1, \dots, Y_m\}$, $\hat{L}_1 \dots \hat{L}_k$ are the literals in $\mathsf{single}(\rho_i) \setminus \{L_1 \dots L_n\}$ that do not occur in $Y_1 \dots Y_j$, and $\hat{L}'_1 \dots \hat{L}'_h$ are the literals in $\mathsf{multi}(\rho_i) \setminus \{L'_1, \dots, L'_l\}$ that do not occur in $Y_1 \dots Y_j$.

"Why not" queries (level 1)

incipit	`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)
verbalized rules	all $(r_i,\theta_i) \in X$
separator	`AND`

Each verbalized rule (r_i,θ_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 $Y_1 \ldots Y_m$ are all the clusters in $\mathsf{maxsingle}(\rho_i)$, and $L_1 \ldots L_n$ are all the literals in $\mathsf{single}(\rho_i)$ that do not occur in $Y_1 \ldots Y_m$.

If $\mathsf{minfailed}(\rho_i) \cup \mathsf{failed}(\rho_i) \neq \emptyset$, 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 $Z_1 \ldots Z_j$ are the members of $\mathsf{minfailed}(\rho_i)$, and $L'_1 \ldots L'_k$ are the members of $\mathsf{failed}(\rho_i)$ that neither occur in $\mathsf{minfailed}(\rho_i)$ nor belong to the set of attribute atoms $t.a : v$ such that t is the main variable of a singleton cluster in $\mathsf{minfailed}(\rho_i)$.

Remark 7 The latter condition makes it possible to ignore the attributes of objects that do not exist.

Example 4 If no id has been provided, then (thanks to this special treatment of attributes) the *why not* verbalization of the rule

```
authenticated('J. Smith') :-
    id(Credential),
    Credential.name:'J. Smith',
    Credential.public_key:K,
    challenge(K).
```

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 J. Smith and public_key K`", which may be misleading. Alternatively, it might have added redundant information such as "`THERE IS NO Credential SUCH THAT the name of Credential is J. Smith`". □

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

If $\mathsf{minfailed}(\rho_i) = \mathsf{failed}(\rho_i) = \emptyset$ then the Failure Slot is verbalized as

```
EACH OF THE FOLLOWING FACTS HAS SOME SOLUTION
      <quantification> verb(L''_1)
                  ⋮
      <quantification> verb(L''_k)
BUT THERE IS NO COMMON SOLUTION
```

where $L''_1 \dots L''_k$ are the literals in $\mathsf{multi}(\rho_i)$.

Remark 8 *In such cases, there seems to be no concise and meaningful way of giving a more detailed account of failure. Or solution consists in including a link to the detailed explanation of the rule, where the individual answer substitutions of each literal can be tried out.*

"How to" queries (level 1)

incipit	`TO MAKE SURE THAT` $\mathsf{verb}(A, \varepsilon)$
verbalized rules	all $(r_i, \theta_i) \in X$ such that $qs(\mathsf{body}(r_i)) \neq \emptyset$ (possibly applicable)
separator	`ALTERNATIVELY`

Each verbalized rule (r_i, θ_i) is formatted according to the following pattern.

<*UD Slot*>
<*UI Slot*>

If $\mathsf{body}(r_i)$ contains no ud-literals, then the UD Slot is verbalized simply as

```
NOTHING NEEDS TO BE DONE IF <quantification>
```

(if the UI Slot is empty then `IF` clause is omitted). Otherwise, the UD Slot in its most complete version is verbalized as:

```
PLEASE MAKE SURE THAT <quantification>
verb(Y_1, θ_i) AND ...AND verb(Y_m, θ_i) AND verb(L_1, θ_i) AND ...AND verb(L_n, θ_i)
WHERE <quantification>
```

where $Y_1 \dots Y_m$ are the maximal ud-clusters of $\mathsf{body}(r_i)$, and $L_1 \dots L_n$ are the ud-literals in $\mathsf{body}(r_i)$. (If the UI Slot is empty then the `WHERE` line is omitted).

The UI (User Independent) Slot is verbalized simply as

$\mathsf{verb}(\hat{Y}_1, \theta_i)$ `AND ...AND` $\mathsf{verb}(\hat{Y}_j, \theta_i)$ `AND` $\mathsf{verb}(\hat{L}_1, \theta_i)$ `AND ...AND` $\mathsf{verb}(\hat{L}_k, \theta_i)$

where $\hat{Y}_1 \dots \hat{Y}_j$ are the maximal non-ud-clusters of $\mathsf{body}(r_i)$, and $\hat{L}_1 \dots \hat{L}_k$ are the non-ud-literals in $\mathsf{body}(r_i)$.

“What if” queries (level 1)

What if queries are evaluated against an extended policy, including the user’s assumptions. The assumptions are treated as true facts (not as unspecified literals), therefore the assumptions may create new query answers. Here we are assuming that $\mathcal{X}$ has been constructed using the extended program, so that s and qs return also the answers and the quasi-answers of the assumptions, respectively. Still we assume an auxiliary label s' that returns the answers that critically depend on the assumptions (i.e., those answers would not be derivable without the assumptions).

The verbalization of *what if* queries depends on the labels of A:

- if $s(A) \neq \emptyset$ then the explanation is like a *why* explanation with a different incipit:

  `IT WOULD BE TRUE THAT` $\mathsf{verb}(A, \varepsilon)$ `BECAUSE`

- if $qs(A) = \emptyset$ then the assumptions would definitely not suffice to prove A, so the explanation is like a *why not* explanation with a different incipit:

  `IT WOULD NOT FOLLOW THAT` $\mathsf{verb}(A, \varepsilon)$ `BECAUSE`

- finally, if $s(A) = \emptyset$ and $qs(A) \neq \emptyset$ (i.e., A’s derivability depends on some unspecified predicate), then the hypothetical reasoning is verbalized as explained below.

incipit	`IT MIGHT BE TRUE THAT` $\mathsf{verb}(A, \varepsilon)$ `BECAUSE`
verbalized rules	all $(r_i, \theta_i) \in X$ such that $qs(\mathsf{body}(r_i)) \neq \emptyset$ (possibly applicable)
separator	`AND, ALTERNATIVELY,`

Each verbalized rule (r_i, θ_i) is formatted according to the following pattern:

```
Rule <rule id> might be applicable:
<UD Slot>
AND
<UI Slot>
AND <Unspecified Slot>
```

The UD Slot is similar to the corresponding slot for *why* queries, with the only difference that the literals that critically depend on the assumptions (as specified by the auxiliary label s') are listed in a separate sub-slot prefixed by

```
BY ASSUMPTION <quantification>
```

Analogously, the UI Slot is similar to the corresponding slot for *why* queries, with the only difference that the literals that critically depend on the assumptions (as specified by the auxiliary label s') are listed in a separate sub-slot prefixed by `BY ASSUMPTION`.

Finally, the Unspecified Slot collects all the unspecified literals $L \in \mathsf{quasi}(\rho_i)$, and verbalizes them after the prefix:

```
IT MAY HOLD THAT / THERE MAY EXIST <quantification>
```

(depending on the number of variables in the unspecified literals).

If the expected outcome information is available, then verbalization can be refined as follows:

- introduce a new label qs'' that returns the quasi-answers that depend only on unspecified literals with expected outcome *success*;
- if for some verbalized rule (r_i, θ_i), $qs''(\mathsf{body}(r_i)) \neq \emptyset$, then use as an incipit the phrase:

  `IT WOULD` ***`PROBABLY`*** `HOLD THAT` $\mathsf{verb}(A, \varepsilon)$ `BECAUSE`

- for such rules, the first line of the verbalization is

  `Rule <`*rule id*`> would` ***`probably`*** `be applicable`

- the verbalization of the unspecified literals with expected outcome *success* is prefixed with

  `IT WOULD PROBABLY HOLD THAT`

7.2 Optional information and What-for queries

Application-dependent explanation information can be enriched—if so desired—with further text patterns that describe the *purpose* of a particular condition, that is, the reason why the condition is checked by the policy. For example, a user might be interested in knowing why a credit card is not enough and an id is needed, too. Moreover, the user might want to know why the credit card is requested: Is it for paying, for a reimbursement, or just for guarantee?

For example, if r_i is the name of the rule defining `credit_card_payment` (see the appendix), then the purpose of the credential requested through the first literal in the body might be further specified as follows:

$r_i[1]$`.purpose : 'This credit card is charged to pay for the resource'.`

($r_i[1]$ denotes the first literal in the body of r_i).

Currently *what-for queries* are not full-fledged queries, they simply show such predefined purpose information. It can be displayed by selecting the corresponding link associated to literals in the context of other queries. The machine-understandable version of what-for queries (useful for improving credential release policies) may exploit suitable standard ontologies such as P3P.

Clearly, the creation of purpose information has a cost during the framework's instantiation.

8 Conclusions and Future Work

In ATN 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. The only extra workload needed during the framework instantiation phase to support explanations consists in writing literal verbalization patterns, and we expect this phase to be assisted by the NLP front-end.

Moreover, the extra computational burden on the server can be limited to adding a few more rules to the filtered policies. When negotiations have to be logged (e.g. during the system's validation phase), then servers should also log the time-dependent portion of the negotiation state used to make the final decision. This logging feature, however, can be either deactivated or selectively activated only on some critical service requests.

Despite its simplicity, our explanation mechanism supports most of the advanced features of second generation explanation systems. We have argued that unsupported features are irrelevant to the ATN framework with one exception: the ad-hoc knowledge base modeling domain knowledge in an explanation-oriented way. We have deliberately decided not to support it to limit the cost of the instantiation phase and the interactions between end users and knowledge engineers. The policies we have experimented with so far have not raised the need for such a knowledge base.

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 table entries 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 failed proofs (including infinite failures). Global proof information and heuristics help in focussing the explanation strictly on the relevant details. The resulting explanations have no counterpart in previous literature.

In future work we are planning to extend the explanation mechanism to the credential selection process based on user preferences, and to the numerical reputation models integrated in PROTUNE [Bonatti and Olmedilla, 2005b, Bonatti et al., 2005]. The latter goal is particularly challenging because to the best of our knowledge there is currently no approach at analyzing the motivations and the provenance of such reputation measures. On the contrary, rule-based reputation estimates can be handled uniformly with the same methods introduced in this paper.

Moreover, it may be possible to enhance *why* and *why not* queries to give some information about quasi-answers (i.e., conditions that *might* be true).

The quality of natural language generation may be improved by replacing fixed text patterns with more sophisticated language generation facilities based on ACE (the controlled natural language front-end of REWERSE).

A The LOL Policy

Main policy:

```
[r1]  allow( browse(index) ) :- true.

[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).

[r5]  allow( comment(X) ) :-
              logged( comment(X) ).
```

Abbreviations:

```
[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).
[r8]  authenticated(User) :-
              do('http://lol.com/register.php').

[r9]  id(Cred) :-
              valid_credential(Cred),
              Cred.type:T,
              Cred.issuer:CA,
              isa(T,id),
              trusted_for(CA,T).

[r10] isa(T,T).
[r11] isa(ssn,id).
[r12] isa(passport,id).
[r13] isa(driving_license,id).

[r14] paid(Usr,Resource) :-
              price(Resource,Price),
              fastpay(Resource,Price).
[r15] paid(Usr,Resource) :-
```

```
            price(Resource,Price),
            credit_card_payment(Resource,Price).

[r16] fastpay(Resource,Price) :-
            valid_credential(FPC),
            FPC.type:fastpay,
            declaration([ pin=XXXX ]),
            charged(Resource,Price,FPC,XXXX).

[r17] credit_card_payment(Resource,Price) :-
            valid_credential(CC),
            CC.type:credit_card,
            CC.owner:Usr,
            authenticated(Usr),
            charged(Resource,Price,CC,na).

[r18] valid_credential(C) :-
            credential(C),
            C.issuer:CA,
            public_key(CA,K),
            verify_signature(C,K).
```

Metapolicy:

```
public(_) .type : state_predicate.
public(X) .evaluation : immediate :- ground(X).

has_subscription(_,_) .type    : state_predicate.
has_subscription(X,_) .evaluation : immediate :- ground(X).

available_for(_,_) .type       : state_predicate.
available_for(X,_) .evaluation : immediate :- ground(X).

price(_,_) .type       : state_predicate.
price(X,_) .evaluation : immediate :- ground(X).

passwd(_,_) .type        : state_predicate.
passwd(_,_) .sensitivity : private.

logged(_) . type : provisional.
logged(_) . evaluation : immediate.
logged(_) . actor : self.
logged(M) . action : (write(M,logfile); make_true( logged(M) )).

trusted_for(_,_) . type       : state_predicate.
trusted_for(_,_) . evaluation : immediate.

known(_,_) . type             : state_predicate.
known(_,_) . evaluation       : immediate.

public_key(_,_) . type        : state_predicate.
public_key(_,_) . evaluation  : deferred.

charged(_,_,_,_) . type : provisional.
charged(_,_,_,_) . evaluation : deferred.
charged(_,_,_,_) . actor : self.
charged(X,Y,Z,T) . action : ( ground((X,Y,Z,T)) ->
               <connect to CC server and make payment>
               ;
               warning('charged(...) nonground - cannot satisfy it')
               ).
```

B Local states used for the examples

B.1 Case 1

```
has_subscription('J. Smith', law(basic)).
has_subscription('J. Smith', computer(basic)).
has_subscription('other user', computer(basic)).
/* etc */

available_for('paper01234.pdf',computer(full)).
available_for('paper01234.pdf',gold).
/* etc */

price('paper01234.pdf',20).
/* etc */

passwd(jsmith,htimsj).
/* etc */

known('UK Government','3%$E%R^%^').
known('EU CA','NKhj34?*&H').
/* etc */

trusted_for('UK Government',id).
trusted_for('EU CA',id).
trusted_for('UK Government',ssn).
/* etc */

credential(c012[type:id, issuer:'Open University', name:'J. Smith']).
verify_signature(c012,'Ghj37tdh*').
known('Open University','Ghj37tdh*').
```

B.2 Case 2

```
has_subscription('J. Smith', law(basic)).
has_subscription('J. Smith', computer(basic)).
has_subscription('other user', computer(basic)).
/* etc */

available_for('paper01234.pdf',computer(full)).
available_for('paper01234.pdf',gold).
/* etc */

price('paper01234.pdf',20).
/* etc */

passwd(jsmith,htimsj).
/* etc */
```

```
known('UK Government','3%$E%R^%^').
known('EU CA','NKhj34?*&H').
/* etc */

trusted_for('UK Government',id).
trusted_for('EU CA',id).
trusted_for('UK Government',ssn).
/* etc */

credential(c012[type:id, issuer:'UK Government', name:'J. Smith',
                             public_key:'HJK789?%^&']).
verify_signature(c012,'3%$E%R^%^').
challenge('HJK789?%^&').
```

References

[Apt, 1990] Apt, K. R. (1990). Logic programming. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, pages 493–574.

[Barzilay et al., 1998] Barzilay, R., McCullough, D., Rambow, O., DeChristofaro, J., Korelsky, T., and Lavoie, B. (1998). A new approach to expert system explanations. In Hovy, E., editor, *Proceedings of the Ninth International Workshop on Natural Language Generation*, pages 78–87. Association for Computational Linguistics, New Brunswick, New Jersey.

[Bonatti et al., 2005] Bonatti, P., Duma, C., Nejdl, W., Olmedilla, D., and Shahmehri, N. (2005). An integration of reputation-based and policy-based trust management. In *Proceedings of the Semantic Web Policy Workshop (SWPW)*. In conjunction with ISWC 2005.

[Bonatti and Olmedilla, 2005a] Bonatti, P. A. and Olmedilla, D. (2005a). Driving and monitoring provisional trust negotiation with metapolicies. In *6th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY 2005), 6-8 June 2005, Stockholm, Sweden*, pages 14–23. IEEE Computer Society.

[Bonatti and Olmedilla, 2005b] Bonatti, P. A. and Olmedilla, D. (2005b). Policy language specification. Technical report, NoE REWERSE. `http://rewerse.net/deliverables/m12/i1-d2.pdf`.

[Chalupsky and Russ, 2002] Chalupsky, H. and Russ, T. A. (2002). Whynot: debugging failed queries in large knowledge bases. In *Eighteenth national conference on Artificial intelligence*, pages 870–877, Menlo Park, CA, USA. American Association for Artificial Intelligence.

[Clancey and Letsinger, 1984] Clancey, W. and Letsinger, R. (1984). Neomycin: Reconfiguring a rule-based expert system for application to teaching. In Clancey, W. and Shortliffe, E., editors, *Readings in Medical Artificial Intelligence: The First Decade*. Addison-Wesley.

[Clancey, 1983] Clancey, W. J. (1983). The epistemology of a rule-based expert system - a framework for explanation. *Artif. Intell.*, 20(3):215–251.

[da Silva et al., 2004] da Silva, P. P., McGuinness, D. L., and Fikes, R. (2004). A proof markup language for semantic web services. Technical Report KSL Tech Report KSL-04-01, January.

[Hasling et al., 1984] Hasling, D. W., Clancey, W. J., and Rennels, G. (1984). Strategic explanations for a diagnostic consultation system. *International Journal of Man-Machine Studies*, 20(3-19).

[Haynes, 2001] Haynes, S. R. (2001). *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.

[Kapadia et al., 2004] Kapadia, A., Sampemane, G., and Campbell, R. H. (2004). Know why your access was denied: regulating feedback for usable security. In *CCS '04: Proceedings of the 11th ACM conference on Computer and communications security*, pages 52–61, New York, NY, USA. ACM Press.

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

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

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

[Schank, 1986] Schank, R. C. (1986). Explanation: A first pass. In Kolodner, J. and Riesbeck, C., editors, *Experience, Memory and Reasoning*, pages 139–165.

[Schnupp, 1989] Schnupp, P. (1989). Building expert systems in prolog.

[Shortliffe, 1976] Shortliffe, E. H. (1976). *Computer-based Medical Consultations: MYCIN*. Elsevier, New York, USA.

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

[Tanner et al., 1993] Tanner, M. C., Keunecke, A., and Chandrasekaran, B. (1993). Explanation using task structure and domain functional models. In David, J.-M., Krivine, J.-P., and Simmons, R., editors, *Second Generation Expert Systems*, pages 586–613. Springer Verlag.

[Tanner and Keuneke, 1991] Tanner, M. C. and Keuneke, A. M. (1991). 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.

[Wick, 1993] Wick, M. R. (1993). Second generation expert system explanation. In David, J.-M., Krivine, J.-P., and Simmons, R., editors, *Second Generation Expert Systems*, pages 614–640. Springer Verlag.