Negotiating Exchanges of Private Information for Web Service Eligibility

Keping Jia[1,2] and Bruce Spencer[1,2]

[1]National Research Council, 46 Dineen Dr., Fredericton, NB, Canada E3B 9W4
[2] Computer Science, University of New Brunswick, Fredericton, NB, Canada E3B 5A3
`Keping.Jia@nrc.gc.ca; Bruce.Spencer@nrc.gc.ca`

Abstract. Private information about individuals that engage in e-commerce business transactions is of economic value to businesses for market analysis and for identifying possible future partners. For various reasons, maintaining the privacy of that information is important to these individuals, including avoiding unwelcome communication, *spam*, from those businesses or their associates. In this paper we advocate a negotiation strategy to be used by an individual deciding whether or not to divulge information to a specific electronic business for a specific purpose, such as achieving preferential status with a service provider or a discounted price from a vendor. The strategy makes use of explanation techniques for expert systems that answer "how", "what if" and "why not" questions. We assume that the business practices of the provider or vendor are available as explicit business rules, including the eligibility criteria for preferential status and price discounts. Our prototype allows the user to obtain a proof that the information to be given is both necessary and sufficient for achieving the eligibility / discount – answering "how" eligibility is established. The communication protocol with the prototype also includes "what if" dialogues allowing a user to assess the difficulty and benefits of achieving eligibility, and "why not" dialogues for identifying missing eligibility criteria. The prototype is built upon the emerging standard Web Services architecture. Thus the prototype allows a business to expose its business practices, educating its customers, so it can provide the most appropriate service for a given individual. The prototype engages the customer to assess the benefit of exposing some private information to the business. Through the "what if" interface, the customer can be aware of the complete set of information that will be necessary to achieve the desired eligibility before any private information is actually transmitted. We offer an example where a user is negotiating a car price discount.

1 Introduction

In the conventional marketplace we exchange goods and services for money; in the electronic marketplace, where buyers and sellers are often unknown to each other, vendors are often willing to provide goods and services in exchange for a user's private information. This information is of value to a business for profiling the demographics of their clientele and compiling lists of potential future customers, both for itself and for its associated businesses. However, preserving the privacy of that information is often a priority for these users, to avoid receiving unwanted communication from these vendors or their associates. Thus one currency of e-commerce is private information.

Currently and more so in the future, users are asked to divulge increasingly specialized information in exchange for higher levels of services. More specialized information in the hands of the service provider leads to better, customized services, but it makes impositions on a user’s privacy. The user may be willing to divulge a private fact if they are informed exactly what it buys them – what new, better service is guaranteed to be provided based on communicating that specific fact.

In the setting of this paper, electronic goods and services are provided within the Web Service architecture. This architecture exposes computational capabilities to consumers across the Internet, and comprises three main facilities: a language for describing such capabilities, a broker for matching an expressed need to a service description and a transport protocol for delivering both the consumer’s data to the service provider and the computed results back to the consumer. One set of standards, endorsed by OASIS and by W3C, uses WSDL (Web Services Description Language)[6], UDDI (Universal Delivery, Discovery and Integration)[1] and SOAP (Simple Object Access Protocol)[11], respectively for these three facilities.

Built on top of this Web Service infrastructure, we are beginning to see proposals for more complex access control, based on various rule systems: Common Rules[8], DAML-S [9], N3[2], RuleML[3] and P3P-APPEL[5]. The accessibility is controlled by policies and regulations that are represented as rules. Far more complex and subtle controls can be achieved under this mechanism. In addition, rule technology’s ability to explain greatly enhances its usability as a mechanism for the access control. These systems promise more flexible and scalable mechanisms which are needed to work together with traditional ones to satisfy today’s usability demands.

We envision that the user is willing to entrust a computerized agent with some of its private information and with the responsibility to communicate that information under conditions specified by the user. This is important for diverting some of the negotiation away from the user’s direct attention. There are several proposals for these languages, including RuleML and P3P APPEL. More private information would remain under direct control of the user.

As the access control becomes more complex, the reasons that a service is denied become more varied. It could be that the service requestor is not an eligible user or the service requestor fails to provide enough or correct information to pass the eligibility check. Denials are often caused by the lack of the knowledge on the user’s side as to what prerequisites are needed for a service.

Our explanation-capable web service works in conjunction with some given web service where eligibility is governed by rules. Suppose a user requests that his trusted agent establish eligibility with a desired web service but the agent and the web service cannot complete this request. Then the explanation-capable web service interacts directly with the user. At this point the user starts to negotiate an exchange of private information for eligibility for specialized, higher quality web services.

In our example, the user is buying a car and trying to negotiate a price discount. Eligibility for the discount is determined by a set of rules maintained by the vendor and stored in the vendor's computer, and is accessible via a web service that we have developed as part of this work. This web service not only determines the user's discount; it also offers explanations. By interacting with the web service, the user can determine why a specific level of service is offered, and why the user is not eligible for a different level of service. Respectively these are answers to "how" and "why not" questions: "How was that derived?" and "Why was this not derived?"

Returning to the privacy question, a user typically wants assurance that by divulging certain private information as part of a dialog with the system, the expected benefit will actually be achieved. Otherwise it is possible that the user's investment, that of sharing information he would rather have kept private, would satisfy only one of the preconditions, while other conditions are left unmet, and the hoped-for qualification is then not granted. The user's investment would then be spent with no return. Instead our prototype allows the user to answer requests for private information with the response "what if" the information were provided. At the end of the series of questions the user can then decide if the return is worth the investment.

The paper proceeds as follows: Section 2 presents some background on expert systems and web services technologies. Section 3 explains how access control, expressed as rules, is useful for generating specific explanations. It contains a fully worked example and a system overview. Section 4 gives the design of the full system and conclusions are offered in Section 5.

2 Background

Expert Systems (or Knowledge-based systems) are computer programs that are concerned with the concepts and methods of symbolic inference, or reasoning, by a computer, and how the knowledge used to make those inferences will be represented inside the machine. The fields of business rules and expert systems do overlap. Because "[r]ules have been used extensively as a way to represent knowledge" [7]. The technology underlying expert systems is widely used to automate business rules.

The knowledge base and the reasoning engine are the two most important constituent parts of an expert system. The knowledge base is a storage of declarative representation of the expertise or knowledge about an application domain. The reasoning engine is the implementation of logical inference mechanisms. It manipulates the symbolic information and knowledge in the knowledge base and applies them to the deduction rules so that conclusions can be reached. The prototype system of this paper uses j-DREW[15] as the rule engine.

It is often claimed that an important aspect of expert systems is their ability to explain themselves[10]. This means the user can ask the system for justification of conclusions or questions at any point in a consultation with an expert system. On the other hand, by looking at explanations, knowledge engineers can see how the system is behaving, and how the rules and data are interacting. They serve as the "logical traces for knowledge bases just like program tracing for conventional programs"[13]. Given that the system knows which rules were used during the inference process, it is possible for the system to provide those rules to the user as a means for explaining the results[10]. In fact, most of the existing expert systems follow this way in the implementation of explanation and debugging systems.

Web Services technology is a newly emerging paradigm for building Web-accessible services across Internet. “On the surface, a Web Service is simply an application that exposes a Web-accessible API. That means you can invoke this application programmatically over the Web.”[14]

Web Services technology mainly aims at the high-level architectures and protocols of the decentralized system over global network. It is designed to work harmoniously with other existing distributed computing technologies like J2EE, DCOM etc. In addition, Web Service protocols do not restrict the implementation techniques for any individual web service.

The Web Services architecture is a message based, service-oriented architecture that is based on the notion that everything is a service. Two important components constitute the main infrastructure of web services: provider and broker. Together with the service requestor, these three distinct actors compose the lifecycle of a web service.

- Service provider: In one aspect, service provider is the implementer of the web service. As a technical term, service provider also denotes web service itself or the hosting environment that the web service is running on.
- Service broker: Service broker is itself a service provider. Service broker usually has a logically centralized directory of services (UDDI registry) and provides relevant services like intelligent search and business classification or taxonomy.
- Service requester: Service requester is the consumer of the web services. Service requester could be a client side program or another web service.

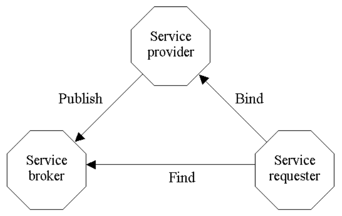


Fig. 1. Web Service Components

The life cycle of particular web service starts from the service provider. There are several ways that a service provider can establish a web service. Service providers can build their own web services by first developing the core functionality of the service, then extracting the interface that the service provider wants to expose to the outside world and last wrapping the interface so that it is SOAP accessible. Next the service provider needs to build an XML based web service interface description (WSDL) that includes all the necessary information for invoking a service: the signature of the service, which communication protocol is used, where to locate the web service etc. In addition to deploying the service on a machine that can be accessed through the Internet, the service provider still need to publish the service interface description to a UDDI directory or broker with necessary taxonomy information so that the web service would be easily found by the service requester. Also, a service provider has the freedom of providing his own implementation of a published service interface or wrapping an existing software application or program into a web service.

A service requester needs to find the required web services and invoke them. The UDDI registry provides flexible mechanisms for service discovery. For example, a service requester can find a particular web service interface through one or more taxonomies individually or the combination of all of them. If a requester knows the name of the service or the company that provides this service, he can directly use this information to get the service. Furthermore, if a service requester gets the interface of a web service, he can easily get all the web service implementations for this interface. The interface description is all that a user needs in order to develop client side programs that can integrate the web service functionalities.

Web service broker or UDDI registry acts as an intermedium between web service providers and web service requesters. To carry out this role, a service broker must be publicly known to both service providers and requesters. For web service providers, it provides rich standard taxonomies and flexible mechanisms so that service providers can easily publish their services in a most discoverable way or establish their own information hierarchies. For web service requesters, it provides diverse searching services so that the data stored in the UDDI registry can be accessed easily and in an organizable way. In fact, a service broker is itself a service provider.

The goal of web services is to achieve high web based interoperability and integretability among software applications regardless of their implementing languages and running environments.

3 Rule-Based Access Control And Explanations

Under the web service architecture, we propose a paradigm of accessibility control of level of service that is governed by policies in the form of rules. Under this paradigm, the system makes decisions based on the result of applying static policies (rules) to the user's information under the current context or environment. Questions could be initiated by the system for the information that is relevant for the decision-making yet not volunteered by the user and his agent.

We assume that questions may be asked by the user on aspects of the web services that are governed by rules, such as questions about eligibility, level of service, membership in reward programs, etc. The explanation service can answer three types of questions: “how”, “why not” and “what if”.

The “how” question is asked by the user to see the proof of some conclusion the system has reached. “How” questions can be asked repeatedly until asked about a fact in the knowledge base. For example, assume that a customer was offered a 5% discount toward the purchase of a Honda car. By asking “how?”, he will get the answer that he got 5% discount because (1) he is a premium customer and (2) Honda is categorized as regular car. By continuing asking “how?” on (1), he will get answer that he is a premium customer because he spent more than $5000 at this car dealership last year. Asking “how?” on this answer could result in a list of purchases made by this user, showing the total. No further “how” answer could be offered by the system because this is basic information (facts).

In order to show the user how a goal is achieved, the system will generate a proof tree [12] with the derived goal as the root. Each internal node of the proof tree is itself a goal with a sub-proof tree rooted at it. Each node and its siblings, together with their parent node, will compose an instance of the rule that is used in the proof procedure.

The “why not” question may be asked when the system fails to derive a goal. Repeadedly asking the “why not” question will lead the user through the rules to find out what causes the goal to fail. The customer of the previous example could ask “why did I not get a 7.5% discount on the Honda?”. The answer will be that in order to get 7.5% discount, (1) he must be a premium customer and (2) Honda must be categorized as luxury goods. If the client decides to trace this rule, he will be told by the system that (1) is satisfied but (2) failed because “Honda is a luxurious car” is not a fact in database and no other rules can lead to this conclusion. In this case, the user may be asked to consider buying an Acura.

A “why not” question needs a slightly different approach because there is no proof tree when a goal fails. However, the proof tree idea is still useful because a “why not” question is concerned with “why a proof tree cannot be built”. By keeping track of the whole proof procedure and marking down all the failure points that prevent a complete proof tree from being built, we will have gathered enough relevant information to construct partially completed proof tree with gaps.

A “what if” question may be asked together with the “why not” question. “What if” questions give the user a chance to know the consequence of assuming that a condition is true. For example, a client may find out through “why not” questions that he did not get the discount because his purchase value is less than $100. Then he could use a “what if” question to check if this is the only condition that prevents him from getting the discount. If the discount is granted after asking “what if the purchase is more than $100”, the client then has the choice to purchase more than $100 to get the discount. But if after asking that “what if” question, the discount is still not granted because other conditions still need to be satisfied (for example, the user should also be a golden card holder), then the client could again ask “why not” questions to find out other conditions to satisfy.

“What if” questions may be asked only on unsatisfied nodes. The response is an explanation tree in which the node is assumed to be satisfied. An explanation tree is a tree-like structure containing all the updated information the user received so far by asking above three questions. A “how” explanation tree is the same as a proof tree and a “why not” explanation tree is a combination of “why not” and “how” trees and “what if” (assumed) leaf nodes. Assuming to satisfy a goal or undoing such assumptions will make temporary changes to the rule base. In certain situations, these changes will affect the other branches of the explanation tree. So, propagating this effect all over the explanation tree is needed to keep the tree consistent with the rule base.

3.1 Car Purchase Example

The following is an example of a business policy controlling the level of service for car dealerships. In this example, the level of discount for buying a car is governed by a set of rules that makes decisions based on the type of car, payment method, user category and insurance information, etc. In the example, a customer Peter negotiates with the system for a 5% discount on his purchase of Volvo-S60. The rules maintained by the service level control system that are directly related to the example are shown in Fig.2.

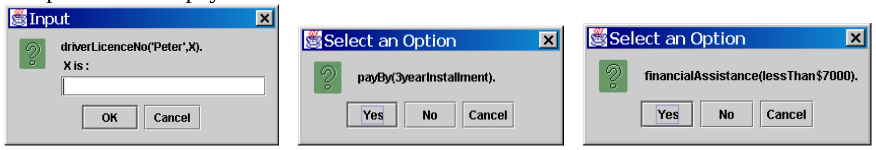
Policy	Rule
An elite customer can get 5% discount on decent or better cars if his payment type is “silver”.	discount(V0,V1,'5%percent')? eliteCustomer(V0), decentCarOrAbove(V1), paymentType(silver).
“Silver” payment type is pay by 2 year installments with financial assistance of less than $10000.	paymentType(silver)? payBy(2yearInstallment), financialAssistance(lessThan$10000).
“Silver” payment type is pay by 3 year installments with financial assistance of less than $7000.	paymentType(silver)? payBy(3yearInstallment), financialAssistance(lessThan$7000).
“Silver” payment type is pay by 5 year installments without financial assistance.	paymentType(silver)? payBy(5yearInstallment), financialAssistance($0).
The senior preferred customer who is insurance affiliated automatically becomes an elite customer.	eliteCustomer(V0)? preferedCustomer(V0), senior(V0), insuranceAffiliator(V0).
A customer is a preferred customer if he bought a regular car from this car dealership in the past five years.	preferedCustomer(V0)? purchased(V0, V1, V2), regularCar(V1), withtinLast5Years(V2).
The customer who is older than 60 is the senior customer.	senior(V0)? moreThan60YearsOld(V0).
The customer who buys car insurance in First Rate Co. is insurance affiliated.	insuranceAffiliator(V0)? driverLicenceNo(V0, X), insuredAt(‘First Rate Co.’, X).

Fig.2. Example Pricing Policy and Corresponding Rule Base

The access control system needs to resort to three information sources for the process:

1. Insurance_IS: An information source provided by First Rate Co. in form of web service. The access control system resorts to this information service to get information about whether a people of a particular driver license number is insured in this company.
2. User’s Trusted Agent: A personal information service that Peter has registered as an information source. The access control system resorts to this service for the user related information like the driver license number and age.
3. Interactive_agent: A temporary web service that exposes exactly the same interface as Insurance_IS and User’s Trusted Agent but processes differently. It simply redirects the query to the user through a GUI and sends the user’s response back to the invoker.

As will be described in 3.2, the control system will first resort to the User’s Trusted Agent for the user related information and then query the Interactive_agent for any questions that the User’s Trusted Agent fails to answer. In the example, the User’s Trusted Agent has Peter’s age, but does not know Peter’s driver licence number and his preference of payment method.

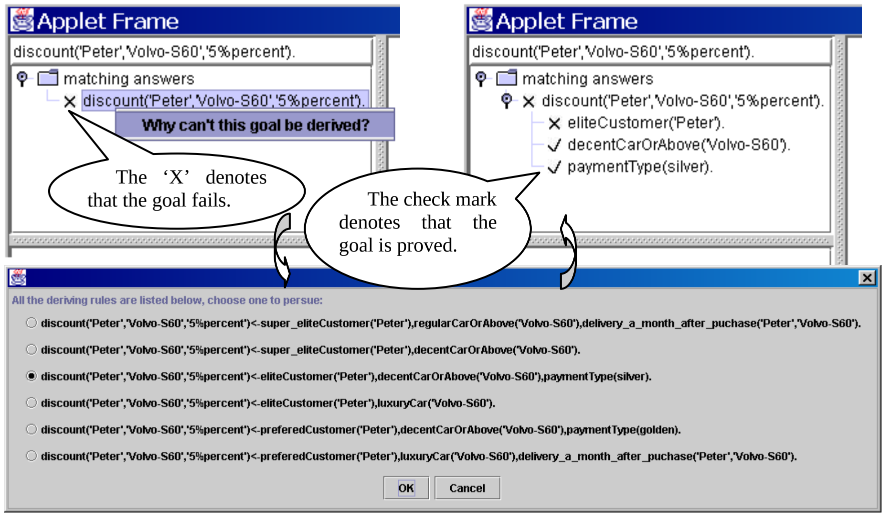


driverLicenceNo('Peter',X).
X is :

payBy(3yearInstallment).

financialAssistance(lessThan$7000).

Fig. 3. Questions redirected by Interactive_agent


Applet Frame
Why can't this goal be derived?
The 'X' denotes that the goal fails.
The check mark denotes that the goal is proved.
OK
Cancel

```
discount('Peter','Volvo-S60','5%percent').
matching answers
  X discount('Peter','Volvo-S60','5%percent').
```

```
discount('Peter','Volvo-S60','5%percent').
matching answers
  X discount('Peter','Volvo-S60','5%percent').
    X eliteCustomer('Peter').
    ✓ decentCarOrAbove('Volvo-S60').
    ✓ paymentType(silver).
```

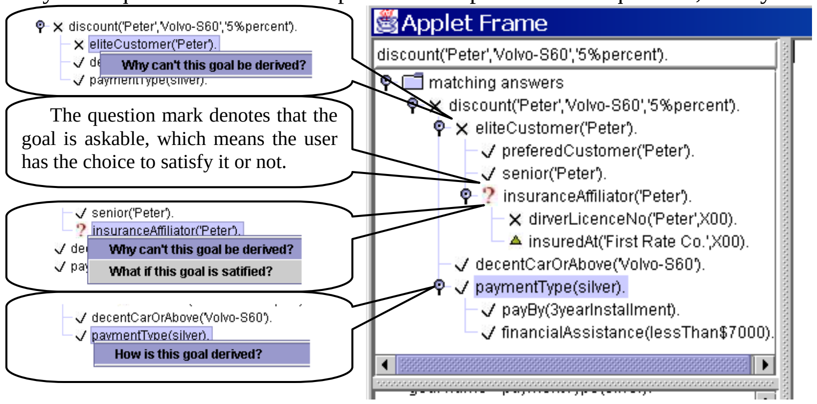
All the deriving rules are listed below, choose one to persue:

- ○ discount('Peter','Volvo-S60','5%percent')<-super_eliteCustomer('Peter'),regularCarOrAbove('Volvo-S60'),delivery_a_month_after_puchase('Peter','Volvo-S60').
- ○ discount('Peter','Volvo-S60','5%percent')<-super_eliteCustomer('Peter'),decentCarOrAbove('Volvo-S60').
- ● discount('Peter','Volvo-S60','5%percent')<-eliteCustomer('Peter'),decentCarOrAbove('Volvo-S60'),paymentType(silver).
- ○ discount('Peter','Volvo-S60','5%percent')<-eliteCustomer('Peter'),luxuryCar('Volvo-S60').
- ○ discount('Peter','Volvo-S60','5%percent')<-preferedCustomer('Peter'),decentCarOrAbove('Volvo-S60'),paymentType(golden).
- ○ discount('Peter','Volvo-S60','5%percent')<-preferedCustomer('Peter'),luxuryCar('Volvo-S60'),delivery_a_month_after_puchase('Peter','Volvo-S60').

Fig. 4. Interaction on "why not" question

Fig.3 shows some possible questions that may be redirected by the Interactive_agent to the user. We can see that the question itself does not give much hint as to how the answer will help the user towards or hinder the user from obtaining the intended discount level. In this case, the explanation system provides a way for the user to know how the decision is made and also to give user a chance to change his choice.

Fig. 4. shows the interaction model of the "why not" question under the assumption that Peter does not give out his driver license number and answers yes to the payment type questions. This will lead to the denial of his intended discount. By pressing the right mouse button on the item that Peter wants to ask questions about, a menu will pop up showing all the eligible questions for this item, which is only the "why not" question in this example. As a response to this question, the system will

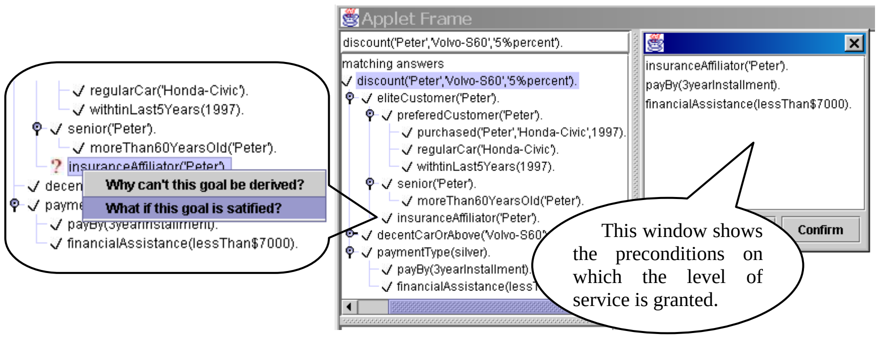

Applet Frame
Why can't this goal be derived?
The question mark denotes that the goal is askable, which means the user has the choice to satisfy it or not.
Why can't this goal be derived?
What if this goal is satified?
How is this goal derived?

```
discount('Peter','Volvo-S60','5%percent').
matching answers
  X discount('Peter','Volvo-S60','5%percent').
    X eliteCustomer('Peter').
      ✓ preferedCustomer('Peter').
      ✓ senior('Peter').
      ? insuranceAffiliator('Peter').
        X dirverLicenceNo('Peter',X00).
        ▲ insuredAt('First Rate Co.',X00).
    ✓ decentCarOrAbove('Volvo-S60').
    ✓ paymentType(silver).
      ✓ payBy(3yearInstallment).
      ✓ financialAssistance(lessThan$7000).
```

Fig. 5. Interaction on "How" question

display a popup window showing all the possible ways that Peter can get the 5% discount. It is up to Peter to choose a way to continue the process. After selection, the popup window disappears and the explanation window will show the user what preconditions he must satisfy in order to get the intended level of service, what preconditions he already satisfies and what he does not.

The user can continue the process by asking "how" questions on satisfied preconditions and "why not" questions on unsatisfied preconditions. As shown in Fig.5, the user first asks a "why not" question on "eliteCustomer('Peter')", then another "why not" question on "InsuranceAffiliator('Peter')" and then a "how" question on "paymentType(Silver).".


Applet Frame
Why can't this goal be derived?
What if this goal is satisfied?
Confirm
This window shows the preconditions on which the level of service is granted.

```
✓ regularCar('Honda-Civic').
✓ withtinLast5Years(1997).
✓ senior('Peter').
  ✓ moreThan60YearsOld('Peter').
? insuranceAffiliator('Peter').
✓ decen
✓ payme
  ✓ payBy(3yearInstallment).
  ✓ financialAssistance(lessThan$7000).
```

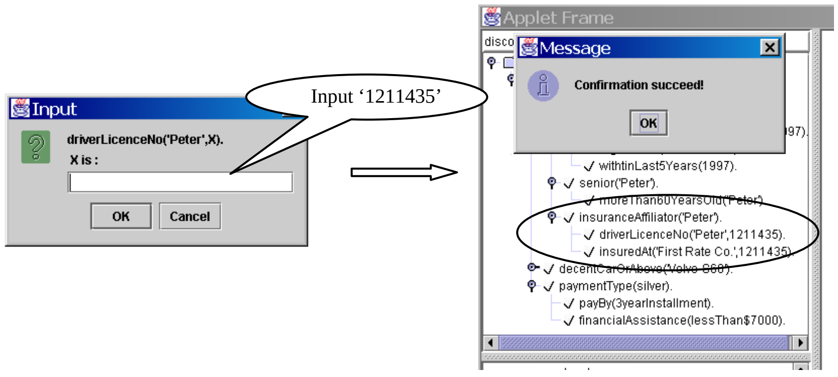
```
discount('Peter','Volvo-S60','5%percent').
matching answers
✓ discount('Peter','Volvo-S60','5%percent').
  ✓ eliteCustomer('Peter').
    ✓ preferedCustomer('Peter').
      ✓ purchased('Peter','Honda-Civic',1997).
      ✓ regularCar('Honda-Civic').
      ✓ withtinLast5Years(1997).
    ✓ senior('Peter').
      ✓ moreThan60YearsOld('Peter').
    ✓ insuranceAffiliator('Peter').
  ✓ decentCarOrAbove('Volvo-S60'
  ✓ paymentType(silver).
    ✓ payBy(3yearInstallment).
    ✓ financialAssistance(less
```

```
insuranceAffiliator('Peter').
payBy(3yearInstallment).
financialAssistance(lessThan$7000).
```

Fig. 6. Interaction on "why" and "why not" question

Fig. 6. shows how the system responses when the user asks "what if" question on the askable goal on Fig. 5. The system grants Peter 5% discount on the Volvo-S60 under the assumption that Peter is insurance affiliated. Some assumed facts need to be confirmed in order to take effect.

Fig.7 shows a demo confirmation interaction. When Peter clicks the "Confirm" button in the Fig.6, a dialog box will pop up, asking Peter to input his driver license number. After Peter inputs his driver license number, the system will use this information to confirm Peter's "affiliated" status on Insurance_IS. We should notice that not all the assumed facts could be confirmed online. For those assumed facts that



```
driverLicenceNo('Peter',X).
X is :
```

```
Confirmation succeed!
```

```
disco
...97).
✓ withtinLast5Years(1997).
✓ senior('Peter').
  ✓ moreThan60YearsOld('Peter').
✓ insuranceAffiliator('Peter').
  ✓ driverLicenceNo('Peter',1211435).
  ✓ insuredAt('First Rate Co.',1211435).
✓ decentCarOrAbove('Volvo-S60').
✓ paymentType(silver).
  ✓ payBy(3yearInstallment).
  ✓ financialAssistance(lessThan$7000).
```

Fig. 7. Confirmation on assumed facts cannot be confirmed online, different application areas have different solutions. For example, the system could simply show an information box to inform the user how to confirm the assumed conditions. When the confirmed information becomes available, Peter can ask for the discount again and it will be granted.

3.2 Architecture Overview

Usually, the information stored in the rule base belongs to one of the two parts: rules or facts. Each rule states that a set of conditions, expressed as atomic predicates, that give rise to a single conclusion, also an atomic predicate. Each fact makes a declaration that a predicate is true. Thus we are using only definite clauses. For the rule-based access control mechanism to work in a distributed environment, we also need to divide the facts into two parts: static facts and case specific facts. Static facts are usually service related facts that apply to all the service requestors for this service and do not change from one case to another. Case specific facts are user related and vary among users.

Since the service designer or service owner maintains most of the rules and static facts, they are easier to configure and deploy. Even though in some cases that information from sources outside the system is needed, the relations between them tend to be stable. We could treat the need for the outsource of this kind as a special "knowledge" in the rule base and, therefore, no extra mechanism is needed for it.

Case specific facts are the most unpredictable factors in this system. The user usually does not know what a service wants from him. Also, the system will not know what information it wants from the beginning because, in many cases, the actual demand for information at a particular point depends on the previous information the user has given. Thus this part of information needs to be requested and given incrementally. This could be done by the access control system interacting with the user for the information or with the user's trusted agent, or both.

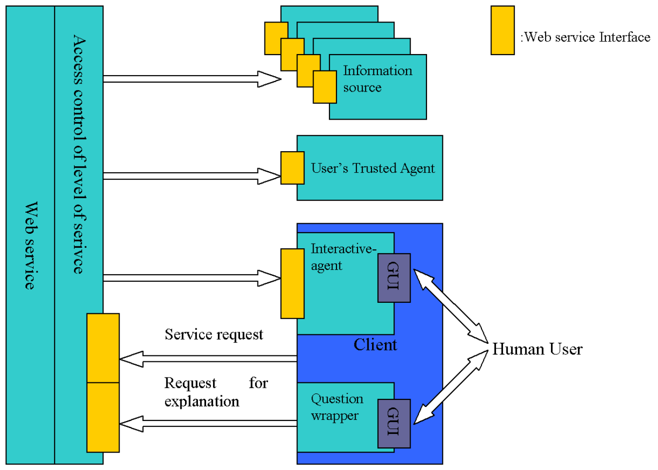


Fig. 8. The interactions between the web services and the information agents.

Fig. 8. illustrates how the client, access control and all related information services are coordinated to work together. Under this architecture, when a user applies for a service, he must provide the access point of his trusted software agent or the user himself (represented by an interactive agent) or both. The user's trusted software agent is a web service that has preinstalled user knowledge. The interactive agent is a temporary web service that simply provide a GUI for interaction, it is temporary because it need not register itself to UDDI and the access point information is in form of IP:Port rather than bindingKey[1]. The access control will use this information to interact with the agent or the user for the credential check.

In practice, the storage of the user's information tends to be distributed. This is because:

1. Many government or private departments have stored much accurate and detailed information about the user (no mater whether it is a human being or a business entity).
2. Some information about a user is more trustworthy if it is given by a third party. e.g. financial information by bank, health information by hospital etc.
3. A third party is usually more financially sound to build its own credentials than individuals.

In order to handle the complexity of the real world, the access point of the User's Trusted Agent could point to the main entrance of a publicly accessible repository instead of a real software agent. This main entrance stores all the possible information services that may provide the information about the user.

We can see that we need a taxonomy, or hierarchical repository with categories for this to work. A hierarchical repository enables the classification of information in a hierarchical way. Categories provide the classification scheme so that the information requester can find out the information in the same way as the information is put into the repository.

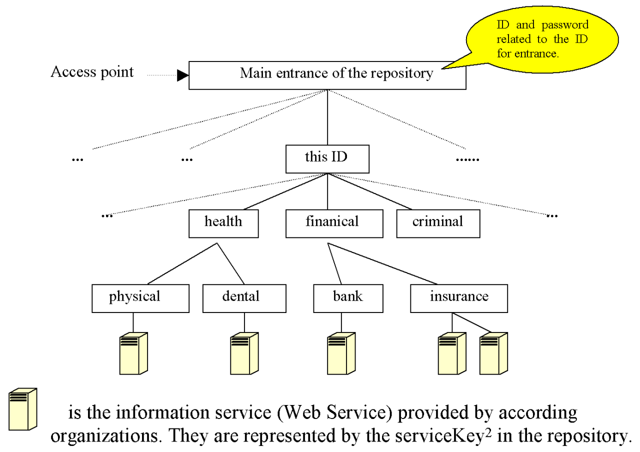

ID and password related to the ID for entrance.
Access point
Main entrance of the repository
...
...
this ID
......
...
health
finanical
criminal
...
physical
dental
bank
insurance
is the information service (Web Service) provided by according organizations. They are represented by the serviceKey2 in the repository.

Fig. 9. User information repository

[1] BindingKey is an auto-assigned UUID in UDDI that uniquely represents a binding of a service.

[2] ServiceKey is an auto-assigned UUID in UDDI that uniquely represents a service.

Fig.9 illustrates the structure of such a repository. Not all the items under the category must have an information provider. Many of them may be empty (null). In this case, the information requestor could resort to the user for the remaining information or it could fail directly with error messages returned to the user (e.g. the information must to be provided by a service from an authorized organization).

4 System Design

Fig.10 shows the XML data structure used by the client side program for the purpose of informing the control system of the user information source.

```
Access point
Syntax:
<access_point  generic = “1.0”  xmlns = “urn:uddi-org:api”>
        <User’s trusted agent>
                    <bindingKey>……</bindingKey>
       </User’s trusted agent>
        <interactive_agent>
                    <IPAddress>……</IPAddress>
                    <port>……</port>
                    <tModelKey 3>……</tModelKey>
        </interactive_agent>
<access_point>
```

Fig. 10. Syntax of Access point

The system uses the predefined predicate $askable to denote if a predicate belongs to the user’s domain of knowledge. So $askable could also be used to indicate if the system need to go to the access point provided by the user for the proof. In this case, the eligibility checking system acts as a user to the User’s Trusted Agent for the proof of the goal. If the User’s Trusted Agent cannot provide a satisfactory answer, the system can simply accept it as an unsatisfiable goal or resort to the user (Interactive_agent) for the final answer.

We can generalize the problem of this category into outsourcing the proof of one or more subgoals to external systems. In the prototype system, we introduce a predefined predicate $outsource(query, bindingInfo) to denote where to find service to process the target query. Here, the “query” takes the form of atomic sentence of first order logic and the “bindingInfo” is a XML string that is similar to the accessing point syntax.

We can connect rule-based web services by assigning $outsource facts or rules in the rule base. For static outsource, we could add the fact

1. $outsource (primeRate(X), ‘<***access_point***> <***Info_source***> <***bindingKey***> uuid:51890f8b-eac5-45fe-8aaa-59ca745f0fc3</***bindingKey***> </***Info_source*** > **</*access_point*>**’)

to denote that primate rate information checking will be carried out by a web service with bindingKey of “uuid:51890f8b-eac5-45fe-8aaa-59ca745f0fc3” which is assumed to be a service provided by a national bank. We can also use rules for the control of

[3] TModelKey is an auto-assigned UUID. Here it is used to represent a service interface.

the conditional outsourcings. For example, the following two rules realize that food product storage queries go to one web service while furniture storage queries go to another one.

2. $outsource(inStore(X, Amount), '<***access_point***> < ***Info_source*** > <***bindingKey*** > …… </ ***bindingKey***> ***</Info_source*** > </***access_point***>') ⇐food(X).
3. $outsource(inStore(X, Amount), '<***access_point***> < ***Info_source*** > <***bindingKey***> …… ***</bindingKey***> ***</Info_source*** > </***access_point***>')⇐furniture(X).

The user-related outsourcing is also done with rules, shown here:

4. $outsource(gender(Person), X) ⇐ $userAccessPoint(X).
5. $userAccessPoint('<***access_point***> < ***User's trusted agent*** > <***bindingKey***> ….. </***bindingKey***> </***User's trusted agent*** > </***access_point***>'>
6. $userAccessPoint('<***access_point***> <***interactive_agent***> <***IPAddress***> …… </***IPAddress***><**port**>……</***port***><***tModelKey***>……</***tModelKey***></***interactive_ agent***> </***access_point***>').

Rule 4 is a static rule that is pre-stored in the rule base. Rule 5 and 6 are dynamically added facts when the user provides access point information as a parameter when applying for a service.

The $outsource predicate acts as a bridge that connects the inference procedures to the universal discovery mechanism of the web services--UDDI. From the endpoint information, the client side program or web service knows where the web service is deployed. In the prototype system, the client side program or web service will use the ***get_bindingDetail*** function for the purpose of searching a web service programmatically at run time. This function will return bindingTemplate information that includes binding port information of the target service. The recommended approach [4] is to cache the bindingTemplate information locally and use the cached information for the repeated calls to the same web service. In the case of the web service invocation failure, the get_bindingDetail function needs to be called again to refresh the binding information. By using the "bindingkey" instead of binding information itself, the system obtains the capacity of tracking web services that might relocate over time.

```
get_bindingDetail
Syntax:
<get_bindingDetail  generic = "1.0"  xmlns = "urn:uddi-org:api">
        <bindingKey> …</bindingKey>
</get_bindingDetail>
```

Fig. 11. Syntax of get_bindingDetail

The operational relationship among inference engine, UDDI and web service is shown in Fig. 12. Based on it, all the participating web services are organized together in a hierarchical way according to their positions in the whole search tree. Each web service is involved in the deduction procedure for the proof of a subgoal and exits when the proof of the subgoal is finished. Also, each web service is autonomous itself and takes the full responsibility to work alone or to involve other services into the local proof procedure. In addition, when the user provides access point of both User's Trusted Agent and Interactive_agent, we may have more than one candidate service for one query. Relating more than one service to a goal is also very useful in many other scenarios. For this kind of query, we need to relate more than one web services to a subgoal and invoke them one at a time during the proof. This invocation mode can be achieved by adding multiple $outsource facts for a query each of which maps to a candidate web service. Of course, this solution needs the collaboration of the inference engine so that all the matching instances are returned instead of just the first one.

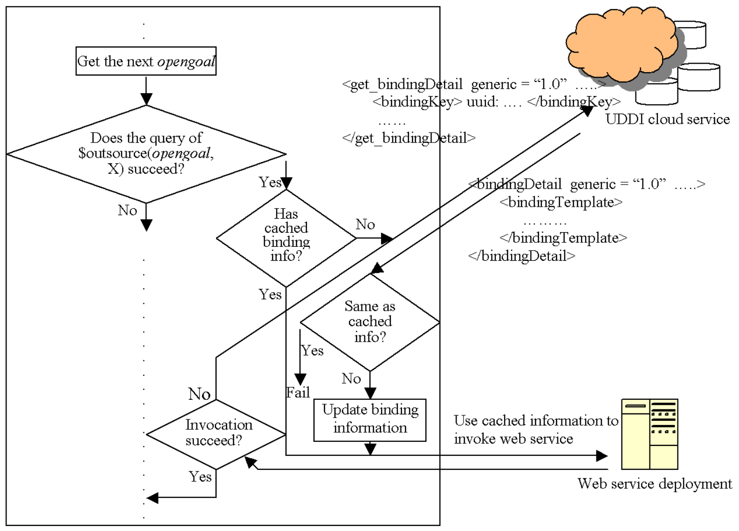

Get the next *opengoal*
Does the query of $outsource(*opengoal*, X) succeed?
Yes
No
Has cached binding info?
No
Yes
Same as cached info?
Yes
No
Fail
Update binding information
No
Invocation succeed?
Yes
<get_bindingDetail generic = "1.0" …..>
<bindingKey> uuid: …. </bindingKey>
……
</get_bindingDetail>
UDDI cloud service
<bindingDetail generic = "1.0" …..>
<bindingTemplate>
………
</bindingTemplate>
</bindingDetail>
Use cached information to invoke web service
Web service deployment

Fig. 12. Operational relationship among inference engine, UDDI and web service

5 Conclusion

This paper suggests that privacy of information is a currency, since it is of value to a user to keep it private and of value to a business to access it. It is now common to exchange private information for digital products and services. We perceive that a user will need to choose between privacy and establishing eligibility for a desirable level of service, and that the negotiation will not be entirely straightforward. A business's rules for eligibility commonly have several conditions that a user needs to meet. The user should not divulge any private information for meeting one of these conditions until it is clear that he/she can meet all of the conditions; thus a simple question and answer protocol is not sufficient. In this paper we apply ideas from previous work for generating explanations for expert systems, where the interaction includes "what if", "why not" and "how" questions. We have chosen to deploy our prototype system[4] using the current web services architecture, composed of UDDI, WSDL and SOAP. In response to an initial request from the user, the web service

[4] The prototype system is a 125k package written in JAVA, which has been deployed and tested on the IBM WebSphere Application Server.

attempts to access some private information by interacting either with the user or with an information agent acting on behalf of the user and entrusted with private information. When the web service asks for more private information to which the user has attached a high value, the interaction is elevated to using the more sophisticated protocols. Through "what if" questions the user may construct an exhaustive list of the valued private information required. The user can also review what information has been accessed via "how" questions, and finally the user can diagnose why the web service did not offer an expected level of service via "why not" questions.

References

1. Bellwood,T. et al: UDDI Version 3.0. At http://uddi.org/pubs/uddi-v3.00-published-20020719.htm
2. Berners-Lee, T.: Ideas about Web Architecture - yet another notation At http://www.w3.org/DesignIssues/Notation3.
3. Boley, H., Tabet, S.: The Rule Markup Initiative. At http://www.dfki.uni-kl.de/ruleml/.
4. Cerami, E.: *Web Services Essentials,* O'Reilly & Associates, Inc., Sebastopol, CA, 2002.
5. Cranor, L., Langheinrich, M., Marchiori, M.: A P3P Preference Exchange Language 1.0 (APPEL1.0), W3C Working Draft 15 April 2002, At http://www.w3.org/TR/P3P-preferences/
6. Christensen, E., Curbera., F., Meredith, G., Weerawarana, S.: Web Services Description Language (WSDL) 1.1, W3C Note 15 March 2001, At http://www.w3.org/TR/2001/NOTE-wsdl-20010315
7. Gottesdiener, E.: Business Rules Show Power and Promise. *Application Programming Trends*, vol. 4, n. 3, March, 1997.
8 Grosof, B.: IBM releases CommonRules 1.0: Business Rules for the Web, At http://www.research.ibm.com/rules/commonrules-overview.html.
9. Martin, D.: DAML-S: Semantic Markup for Web Services. At http://www.daml.org/services/daml-s/0.7/daml-s.html
10. Merritt, D.: Building Expert Systems in Prolog. Springer-Verlag, New York, USA, 1989.
11. Mitra, N.: SOAP Version 1.2 Part 0: Primer, W3C Working Draft, June, 2002, At http://www.w3.org/TR/soap12-part0/.
12. Poole, D., Mackworth, A., Goebel, R.: *Computational Intelligence – A Logical Approach,* Oxford University Press, New York, 1998.
13. Russell, S. J., Norvig, P.: *Artificial Intelligence – A Modern Approach*, Prentice Hall, New Jersey, USA, 1995.
14. Shohoud, Y.: Real World XML Web Services, At http://www.learnxmlws.com/book.
15. Spencer, B.: The Design of j-DREW: A Deductive Reasoning Engine for the Web, In *Proceedings of the First CologNet Workshop on Component-based Software Develoment and Implementation Technology for Computational Logic Systems*, Madrid, Spain, Universidad Politécnica de Madrid, Facultad de Informática TR Number CLIP 4/02.0, pages 155-166. Sept 18-19, 2002.