Trust Management Tools for Internet Applications

Tyrone Grandison and Morris Sloman

Department of Computing, Imperial College, University of London,
Huxley Building, 180 Queen's Gate, London SW7 2RH, UK
{tgrand, m.sloman}@doc.ic.ac.uk

Abstract. Trust management has received a lot of attention recently as it is an important component of decision making for electronic commerce, Internet interactions and electronic contract negotiation. However, appropriate tools are needed to effectively specify and manage trust relationships. They should facilitate the analysis of trust specification for conflicts and should enable information on risk and experience information to be used to help in decision-making. High-level trust specifications may also be refined to lower-level implementation policies about access control, authentication and encryption. In this paper, we present the SULTAN trust management toolkit for the specification, analysis and monitoring of trust specifications. This paper will present the following components of the toolkit: the Specification Editor, the Analysis Tool, the Risk Service and the Monitoring Service.

Keywords

Trust Management Tools, Trust Specification, Trust Analysis, Risk Service, Trust Monitoring Service.

1 Introduction

Trust management is becoming an important topic as indicated by the substantial number of research projects that have been initiated in the last few years and the efforts by leading software and hardware vendors to incorporate trust management into their products. The impetus behind the interest in this field is the fact that trust is crucial to Internet business transactions, but these do not permit personal, face-to-face interactions, so methods have to be devised to engender trust without these important human elements. The perception of customers that vendors' products are often insecure and not trustworthy has also fuelled research into trust management. The need for software agents to traverse smoothly through a web of interconnected networks in a trustworthy manner and the benefits from separating trust management code from application code are all factors contributing to the present popularity of trust management research.

Current trust management projects, e.g. KeyNote [1-3], REFEREE [4, 5], SD3 [6], etc., have been readily accepted by mainstream corporate entities. The vast majority

of these projects are primarily concerned with public key authorization, authentication and access control but do not consider experience, risk and trust analysis. The reason for this overemphasis on a small part of the trust management problem lies in the state of affairs in the security world in the mid-90s. At the time, cryptography was used as a solution to the Internet's security shortcomings. However, the problem with cryptography was that it required a complex key management infrastructure. The solution to this problem was to create public key infrastructures (PKIs). The trust in the certificate authority and the keys in a PKI led to another problem. This is what came to be seen as the trust management problem.

According to Blaze et. al. [7], trust management is "a unified approach to specifying and interpreting security policies, credentials, relationships which allow direct authorisation of security-critical actions". This has been the dominant view of trust management since the mid-90s. However, from the definition and subsequent implementations, we can see that this focuses on the problem of managing public key authorisation, but ignores other aspects, namely: 1) the analysis of trust relationships, 2) the use of auxiliary factors, such as risk and experience, in trust decision making, and 3) the fact that trust is a dynamic concept that evolves with time. All these solutions assume that public key infrastructures (PKIs) will be the basis of future security solution, but this may not be a prudent assumption, due to the problems of large-scale deployment of PKIs for the Internet.

In this paper, the view taken of trust management is that it is "the activity of collecting, codifying, analysing and presenting evidence relating to competence, honesty, security or dependability with the purpose of making assessments and decisions regarding trust relationships for Internet applications" [8, 9]. Evidence could include credential such as certificates for proof of identity or qualifications, risk assessments, usage experience or recommendations. Analysis includes identification of possible conflicts with other trust requirements. Thus, trust management is concerned with collecting the information required to make a trust relationship decision, evaluating the criteria related to the trust relationship as well as monitoring and re-evaluating existing trust relationships.

In this paper, we present the SULTAN Toolkit, a set of tools for specifying trust relationships, analyzing them, evaluating risk and monitoring the conditions of trust relationships. In section 2, we present the basic concepts of trust and highlight the SULTAN trust model and abstract architecture. Section 3 introduces a scenario that will be used in our discussion of the tools. Section 4 provides an overview of the SULTAN Specification Editor, while section 5 provides a discussion on the Analysis Tool. In section 6, we present a discourse on the Risk Service and highlight the Monitoring Service in section 7. Section 8 contains information on related work and we look at conclusions and future directions in section 9.

2 Overview of the SULTAN Trust Management Model

In this section, we will present the basic notions concerning trust. We will also give a brief introduction to the SULTAN Trust Management Model.

Trust is normally specified in terms of a relationship between a *trustor*, the subject that trusts a target entity, which is known as the *trustee* i.e. the entity that is trusted (Figure 1). Each trust relationship must be defined with respect to a particular scenario or context, can be viewed as a mathematically-defined binary relation, must have an associated trust level, can be stated as adhering to some property and has auxiliary factors that influence it (see [9] for more details).

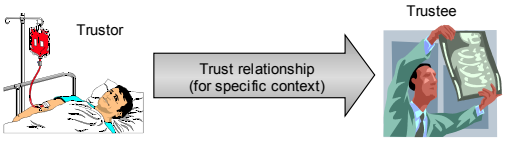

Fig. 1. A Typical Trust Relationship

Our view of trust that is that it is "the quantified belief by a trustor with respect to the competence, honesty, security and dependability of a trustee within a specified context". Quantification reflects that a trustor can have various degrees of trust (distrust), which could be expressed as a numerical range or as a simple classification such as low, medium or high. Competence implies that an entity is capable of performing the functions expected of it or the services it is meant to provide correctly and within reasonable timescales. An honest entity is truthful and does not deceive or commit fraud. Truthfulness refers to the state where one consistently utters what one believes to be true. In this context, to deceive means to mislead or to cause to err (whether accidentally or not), while fraud refers to criminal deception done with intent to gain an advantage. A secure entity ensures the confidentiality of its valuable assets and prevents unauthorised access to them. Dependability is the measure in which reliance can justifiably be placed on the service delivered by a system [10]. Thus by definition, we see that a dependable entity is also a reliable one. Timeliness is an implicit component of dependability, particularly with respect to real-time systems. What about the concept of distrust?

Distrust is essential for revoking previously agreed trust for environments or when entities are trusted by default. We define distrust as "the quantified belief by a trustor that a trustee is incompetent, dishonest, not secure or not dependable within a specified context". The view of trust (and distrust) taken by some researchers has led to the terms trust, access control, authorisation and authorisation being confused and used interchangeably.

The fact that Tony trusts Darren to perform network security testing does not necessarily imply that Tony will allow Darren access to the network. The principle is simple: trust does not necessarily imply access control rights and vice versa. Authorisation is the outcome of the refinement of a (more abstract) trust relationship. For example, if I develop a trust relationship with a particular student, I may authorise him to install software on my computer and hence set up the necessary access control rights to permit access. Authentication is the verification of an identity of an entity, which may be performed by means of a password, a trusted authentication service or using certificates. There is then an issue of the degree of trust in the entity, which issued the certificate. Note that authorisation may not be necessarily specified in terms of an identity. Anonymous authorisation may be implemented using

capabilities or certificates. This leads us to the SULTAN Trust Management Model (Figure 2).

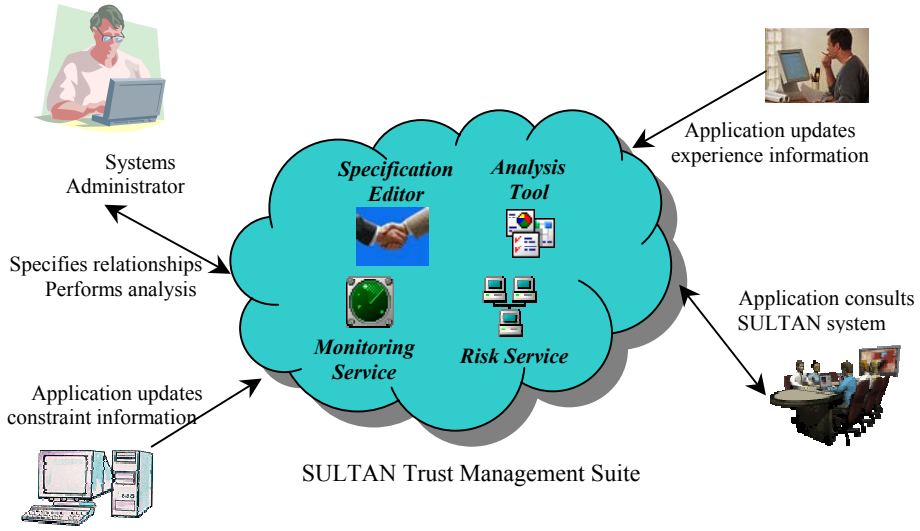

Fig. 2. The SULTAN Trust Management Model

The SULTAN Trust Management Suite consists of four primary components: 1) The Specification Editor – which integrates an editor for trust specifications, a compiler for the SULTAN specifications, and auxiliary tools for storage, retrieval and translation of the specifications, 2) The Analysis Tool, which integrates a Query Command Builder, basic editor and front-end to Sicstus Prolog, 3) The Risk Service, which allows for the calculation of risk and the retrieval of risk information, and 4) The Monitoring Service, which updates information relating to experience or reputation of the entities involved in the trust relationships.

The Specification Editor is used by the system administrator to encode the initial set of trust requirements. He may also perform analyses on the specifications he has entered. This information is then stored in the system. Over time, applications present SULTAN with new information regarding the state of the system, experiences or even risk information. Whenever a decision is to be made, the SULTAN system may be consulted to provide information that will enable one to make a better decision. We will not be discussing SULTAN trust consultation in this paper. The architecture that supports this model is shown in Figure 3.

Constraint, risk and experience information are stored in a state information server, and specifications are stored in a Specification Server. The Analysis Tool is a front-end for the Analysis Engine, which uses the state information, specifications and risk information to determine the existence of conflicts. The Risk Service is allowed to only retrieve information from the State Information Server, and the Monitoring Service can only add or change state information. For the prototype the servers are implemented as whole structures. Fragmentation and distribution issues would serve to complicate our main goal. However, these issues will be addressed in later releases.

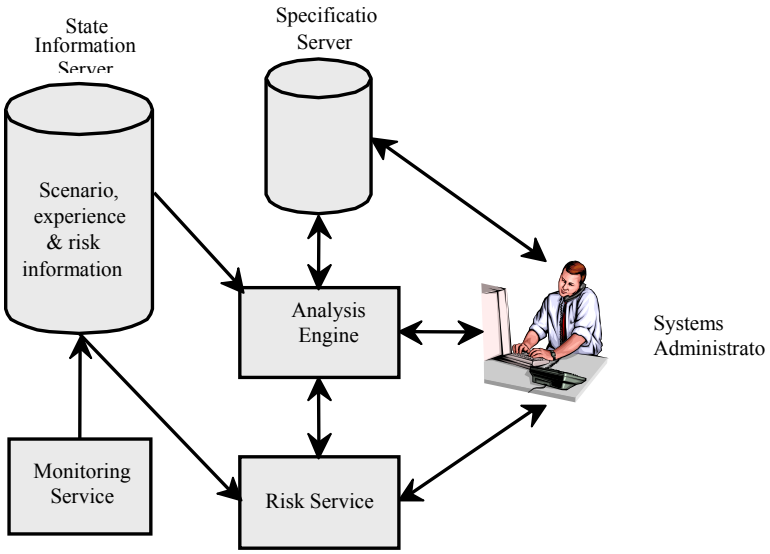

Fig. 3. Abstract Architecture for the SULTAN Trust Management System

In the following section we use a simple scenario to elaborate on the concepts of the SULTAN system.

3 Scenario

This scenario is based on an online multimedia station. We will consider the management of the trust relationships relating to employee data access and client service provision. We will refer to our online station as JamRock. JamRock provides streaming audio and video to clients with either broadband or dialup connections. Only registered users have access to JamRock's stock of forty thousand titles.

To register, a prospective client must provide a valid name, a valid email address add credit card information. IP address information for a user's last login is automatically taken. Credit card information is taken only to ensure that minors do not have direct access to the adult content section of JamRock's library. Once the name, email and credit card information are provided, a unique, randomly-created password is sent to the client.

JamRock is a small business run by Tom Hackett. Tom employs a system administrator, Calvin, a programmer, Steve and two marketing people to help in the day-to-day running of JamRock. For legal reasons, Tom must maintain records about his employees, for tax purposes, etc. Thus, there are databases required for JamRock's employee information, its client base and its content. All three databases are considered separate entities. In Figure 4, we see the organizational structure of JamRock, which we will be referring in sections to follow.

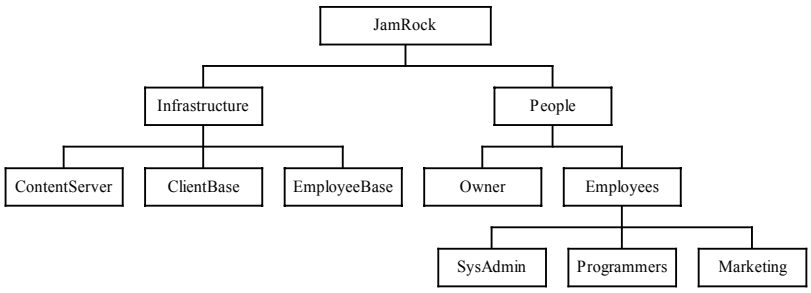

Fig. 4. Organizational Structure of JamRock

Calvin has the enviable task of ensuring that the client's trust is not betrayed, that the employees trust in the integrity of their boss is protected and that the owner's trust in his ability to ensure that only valid clients use the system is never compromised. Clients trust that their personal data will not be misused. Employees trust that the owner will use their data for legitimate purposes only. The content base should be used by legitimate clients only.

We will now discuss the SULTAN specification framework to see how Calvin can explicitly specify his trust requirements in the SULTAN specification notation.

4 Specification Editor

The Specification Editor is a composite toolkit for creating, storing, retrieving, editing and translating SULTAN Specifications. In this section, we will look at the language used by the SULTAN Editor, the compiler framework and a few other features of the Editor.

The Specification Language

In the specification language, one can specify trust, distrust, negative recommendation and positive recommendation statements. Entity names are assumed to be abstract. Thus, relationships can be defined between IP addresses, domains, public keys, software agents or even router ids. Concrete names are assigned when trust specifications are refined into a lower-level framework for purposes of security and or privacy enforcement.

A trust/distrust statement has the following form:

PolicyName : **trust** (Tr, Te, As, L) ← Cs;

Tr trusts/distrusts Te to perform As at trust/distrust level L if constraint(s) Cs is true.

PolicyName is the unique name for the assertion. Tr is the trustor, i.e. the entity that is trusting. Te is the trustee, i.e. the entity is to be trusted. As is the action set, i.e. a colon-delimited list of actions (function names) or action prohibitions. The first parameter in an action name specifies the entity the action is performed on (whether on the trustor, or the trustee, or some other entity that is a component of either). L – is

the level of trust/distrust. L can be an integer or a label. Labels are converted to integers for analysis and management. For integer values of L, $-100 \leq L < 0$ represents distrust assertions and $0 < L \leq 100$ represents trust assertions. Cs is the constraint set, i.e. a set of delimited constraints that must be satisfied for the trust relationship to be established. The delimiters are the logical and (&) and logical or (|). Cs must evaluate to true or false.

An example of a SULTAN trust statement is:

CustomerVer: trust(Supplier, Customers, view_pages(Supplier), _X)
← _X > 0 & GoodCredit(Customers) &
risk(Supplier, Customers, _) <= 2;

(This example shows the use of two features of the notation: variables and auxiliary functions. A variable in SULTAN is a series of characters with the underscore prefixed. In the above example, _X is a named variable that is used to refer to the trust level. The anonymous variable is represented by the underscore and it signifies a value that we have no interest in. There is an anonymous variable in the risk auxiliary function used in the constraints section)

Interpretation: Supplier trusts Customers to perform view_pages(Supplier) at trust level _X if _X is greater than 0, if GoodCredit(Customers) is true and if the risk that the Supplier will undertake in interacting with Customers is less than or equal to 2. (Please note that the view_pages function is performed on Supplier by Customers)

Realtor: trust (Jenny, Realtor, send_deals(Realtor, Jenny), HighTrust)
← trust (Jenny, Tom, ProvideInfo(Jenny), MediumTrust) |
trust (Tom, Realtor, send_deals(Realtor, Tom), MediumTrust);

Interpretation: Jenny trusts Realtor to perform send_deals(Realtor, Jenny) at trust level HighTrust if Jenny trusts Tom to perform ProvideInfo(Jenny) at trust level MediumTrust or if Tom trusts Realtor to perform send_deals(Realtor, Tom) at trust level MediumTrust.

PDA: trust (Morris, Symantec, do_definition_update(Morris, Computer), HighTrust)
←DefinitionState(Symantec) = "old";

Interpretation: Morris trusts Symantec to perform do_definition_update(Morris) at trust level HighTrust if DefinitionState(Symantec) = "old".

A negative/positive recommendation statement has the following format:

PolicyName : **recommend** (Rr, Re, As, L) ← Cs;

Rr recommends/does not recommend Re at recommendation level L to perform As if constraint(s) Cs is true.

PolicyName is the unique name of the rule being defined. Rr is the recommendor, i.e. the name of the entity making the recommendation. Re is the recommendee, i.e. the name of the entity that the recommendation is about. L is the recommendation level, i.e. the level of confidence in the recommendation being issued by Rr. L can either be a label or an integer. All labels are translated to integers for analysis and management. L is ≥ -100 and < 0 for negative recommendations and L is > 0 and ≤ 100 for positive recommendations. As is the recommended action set, i.e. a colon delimited set of actions or action prohibitions that Rr recommends Re be trusted/distrusted to perform. Each action name stipulates the entity on which the action is performed. Cs is the constraint set, i.e. a delimited set of constraints that must be satisfied for the recommendation to be valid. Delimiters include the logical and (&) and logical or (|). One should be aware that: 1) The recommendation level and trust level are assumed to be independent of each other, unless otherwise specified, and 2) A recommendation may result in a trust specification (i.e. a

recommendation may be the basis for one's trust specification). However, the trust level need not correspond to the recommendation level.

An example of a recommendation is:

Credit: recommend (NatWest, Clients, GetCreditCard(NatWest), HIGH)
← Balance(Clients) > 10000;

Interpretation: NatWest recommends Clients at recommendation level HIGH to perform GetCreditCard(NatWest) if Balance(Clients) is greater than 10000.

AttPol: recommend (UCL, OpenU, do_research(OpenU), -10)
← Researchquality(OpenU) <= 3 ;

Interpretation: UCL does not recommend OpenU at recommendation level –10 to perform do_research(OpenU) if Researchquality(OpenU) is less than or equal to 3.

For details about the intricacies of the components of trust and recommend statements, please see [9].

Now that we know how to specify statements in SULTAN, look us codify Calvin's trust requirements for JamRock. We will use the abstract names from the leaves in the tree in Figure 3. His first task is to ensure that the client's trust is not betrayed. This means the client believes that the data collected by JamRock is not being used by any unauthorized individuals, for unscrupulous pursuits. It is JamRock's policy that only the owner is trusted to view their data (nothing else). Let us codify this as:

ClientPrivacy1: trust(JamRock, Owner, view_client(ClientBase, _AnyEntity) , 50);

At times, users may forget their login information or may have queries about their data. At this point Calvin may need to provide this information to them provided they have been sufficiently authenticated. One could represent this as:

ClientPrivacy2: trust(JamRock, SysAdmin, provide_info(ClientBase, _Client, _Inf), 50)
← auth_tokens_provided(_Client) & is_registered(_Client) &
is_Information(_Client, _Inf);

Calvin's requirement concerning the employees' trust in the integrity of the boss relating to their personal records may be expressed in SULTAN as:

EmpTrust1: trust(Employees, Owner, _X, 50)
← defined_on_for(_X, EmployeeBase, _EmployeeY) &
not_compromised(_EmployeeY);

The above statement says that the Employees trust Owner to perform an action _X (at trust level 50), if this _X is an action defined on _EmployeeBase and concerns EmployeeY and _EmployeeY is not compromised by this action being performed.

Calvin final trust requirement is to ensure that the Owners' trust in his ability to ensure that only valid clients use the system. This requirement must be stipulated in two statements. The first stating that the Owner trusts the SysAdmin and the second stating that only valid clients can access the content database. These can be specified in SULTAN as:

ClientAccess1: trust(Owner, SysAdmin, _A, 50)
← defined_on_for(_A,_ContentServer, _Y) & not_abuses(_A, JamRock);
ClientAccess2: trust(JamRock, _AnEntity, get_stream(ContentServer, _ConnType), 50)
← is_registered_client(ClientBase, _AnEntity) &
has_Connection_Type(ClientBase, _AnEntity, _ConnType);

After typing these specification into the editor, Calvin can then compile them, save them and translate them to Prolog to start performing analysis on them. Note that the trust levels in the examples for JamRock are arbitrarily set to 50. Let's take a quick look at the compiler.

The SULTAN Compiler Framework

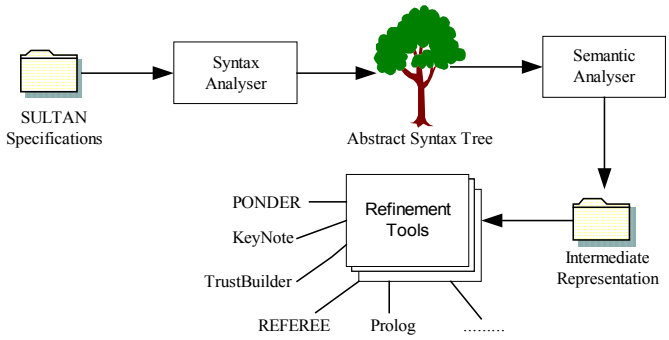

Fig. 5. SULTAN Compiler Framework

Figure 5 shows the SULTAN Compiler Framework. The main modules of the compiler are based on a LALR(1) parser, which was generated by SableCC [11]. SableCC is an object-oriented Java parser generator. The syntactic and semantic phases produce a intermediate representation of the code entered. This intermediate representation is essentially the abstract syntax tree with an added label. To refine from the SULTAN specification language to a lower-level notation/framework, one needs only to provide a Java file that traverses the intermediate representation and generates the necessary. As a proof of concept and a requirement for direct use in analysis, a SULTAN to Prolog translator is included in the Specification Editor.

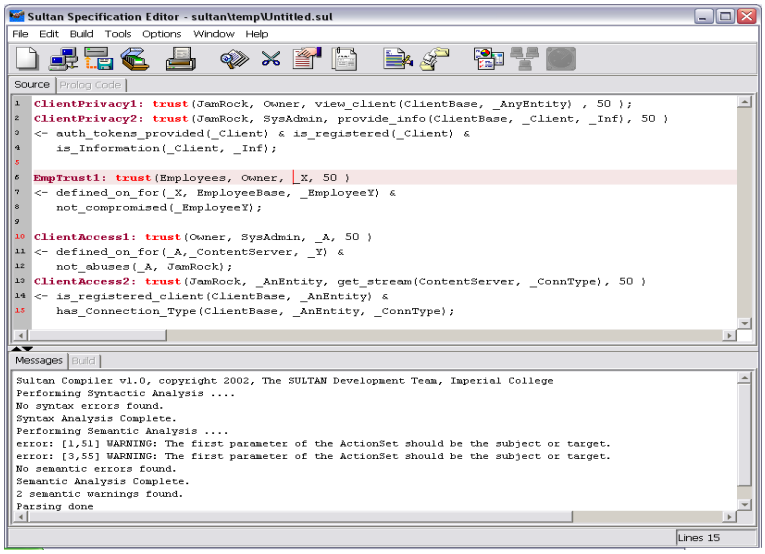

Fig. 6. Compiled version of Specifications for JamRock

Figure 6 shows the results of using the SULTAN Compiler on Calvin's specifications. Let us now look at the other features of the Editor.

Other Features

The Editor also includes an Abstract Syntax Tree Viewer (Figure 7). This is included as an aid for translator developers.

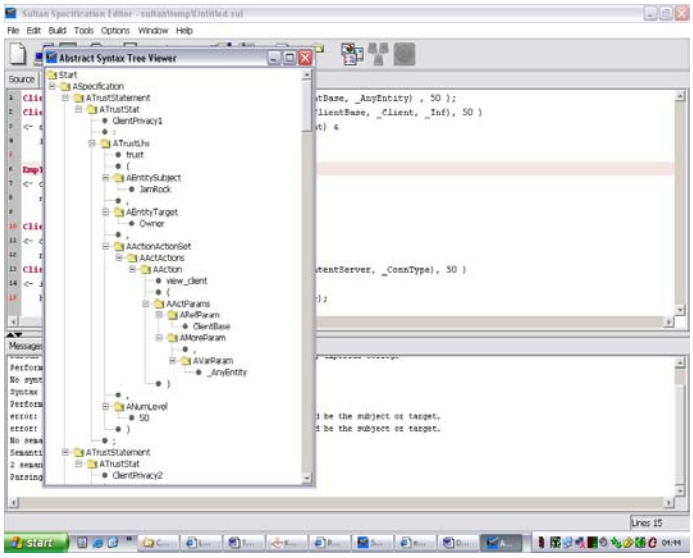

Fig. 7. Abstract Syntax Viewer for specs for JamRock

Once translators have been created then the Editor provides a facility where one can specify the file and its location and have it integrated as a part of the Editor, until one chooses to remove it. Figure 8 shows the dialog that one uses to do this integration.

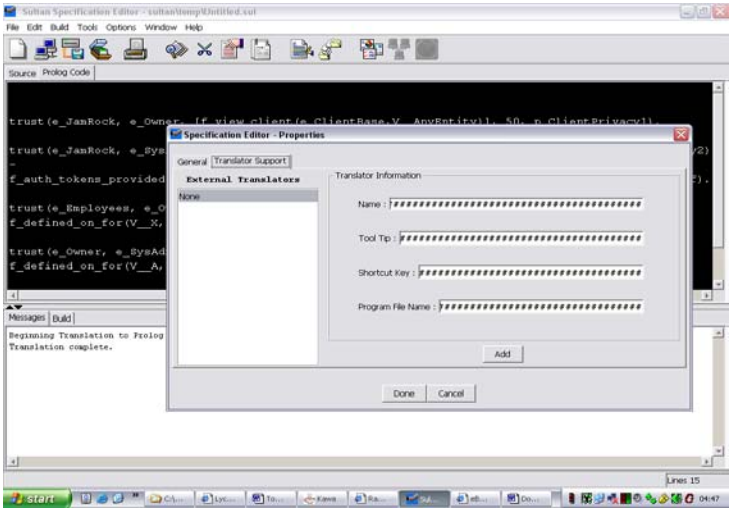

Fig. 8. How to integrate translators into the Editor

Let us turn our attention to the Analysis Tool.

5 Analysis Tool

The Analysis Tool is a front-end for both the Analysis Engine and Sicstus Prolog. The Analysis Tool encapsulates the Analysis Engine, which is the bridge between the Sicstus Prolog System, the Specification Server, the State Information Server and the Risk Service. The Analysis Tool allows the administrator to produce simulation analysis and property analysis.

Simulation analysis is performed by using the Analysis Tool to directly add information to the Prolog System (with an option to store them to the State Information Server) and then using one's own knowledge of Prolog or the built-in SULTAN Analysis Model to ask questions.

Property analysis involves checking whether specified properties hold on trust and recommendation rules. The properties can be with respect to the specification source, which is essentially program reasoning. Source analysis ignores the constraints, i.e. assume they are true. The properties can also be with respect to examining trust relationships to identify scenarios of interest. Scenario analysis involves reasoning about the state of the system, and the current state of constraints. For example, in recommendation Credit, the system must have a value for Balance(Clients) in order to evaluate the constraint. The monitoring system would be responsible for updating this information.

The prototype for the analysis model is implemented in Prolog because this allows arbitrary application specific analysis to be performed to meet the requirements of a particular organization. To perform an analysis query, we use the rules in the analysis model. A query is a predicate which defines the property that the administrator is interested in, and is used to retrieve a set of results. The format of this construct is:

query(Vars, Conds, ResultSet).

This finds all Vars that meet condition(s) Conds and stores the result in ResultSet, where Vars are the variables to be collected that must be used in the Conds section; Conds are the conditions to be satisfied and ResultSet is the set of Vars that meet Conds. Note that Conds can be any application-specific predicate defined by the administrator relating to the source rules, scenario data or a system integrity check.

Examples:

query([T,D], (**p_pos_trust**(T), **p_neg_trust**(D), **p_trustor**(Tr,T),
p_trustor(Tr, D), **p_trustee**(Te,T), **p_trustee**(Te, D), **p_actionset**(ACT, T),
p_actionset(ACT, D)), Result).

The above query asks the question "Is there a trust rule and a distrust rule (in my specification source) concerning the same trustor, trustee and actionset?". (This is an example of source code analysis).

query([T, NR], (**pos_trust**(T), **neg_rec**(NR), **subject**(Rr,T),
subject(Rr, NR), **target**(Re,T), **target**(Re, NR), **actionset**(ACT, T),
actionset(ACT, NR)), Result).

Is there a scenario in which there is a trust relationship and a recommendation that concern the same subject, target and actionset? (This is an actual scenario analysis)

It is important to highlight that the Analysis Tool starts with the SULTAN Analysis Model, the state information, the domain's specifications and a template of common conflicts and redundancies that may be of interest loaded into its current memory.

Let's suppose that Calvin wishes to find out if there are any potential conflicts of interest in his specifications. This can be specified in the SULATAN analysis notation as:

query([X, Y], (**p_pos_trust**(X), **p_pos_trust**(Y), **p_target**(Te,X), **p_target**(Te, Y), **actionset**(A, X), **actionset**(A, Y)), Result).

The above asks if there are two policies that have the same trustee and actionset. Figure 9 shows such a query. From the Tool, one sees that the specification does contain such potential conflicts.

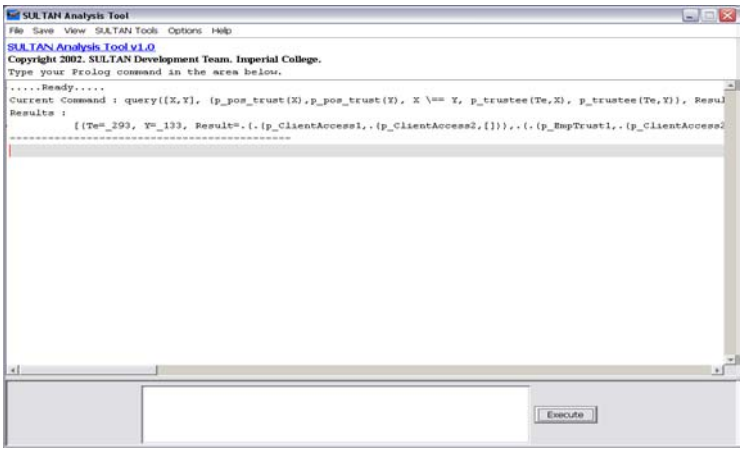

Fig. 9. A Sample Query

The results in Figure 9 show that trust policies ClientAccess1 and ClientAccess2 (amongst others) may pose a conflict of interest (this is based on analysis of the source code only). More complex queries can be built using the Query Statement Builder (shown in Figure 10).

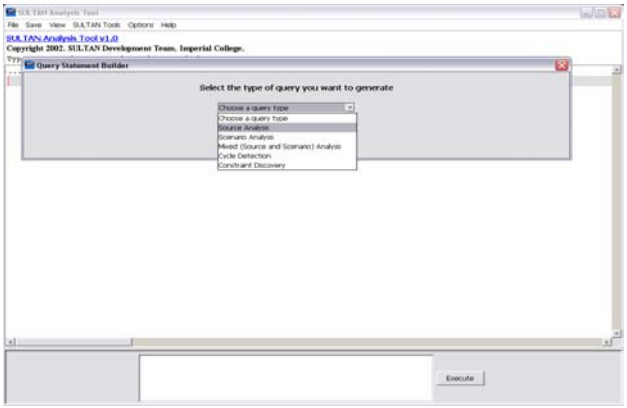

Fig. 10. SULTAN Query Statement Builder

The value of the Analysis Tool increases over time, as it acquires new information and is able to provide the administrator with less obvious results. Note that our

discussion on the Analysis Tool excluded details on cycle detection, resolution and on the use of abduction to enable queries concerning the constraints yet to be satisfied. These topics are covered in [9]. We now turn our attention to the Risk Service and its operation.

6 Risk Service

Risk is a measure of the probability of a transaction failing. In general, if there is a high risk associated with a service provider or business partner, then the lower the trust in the entity. However risk is also related to value of a transaction. A client may trust a supplier who is considered a high risk when the services supplied only cost $10 but may not trust a medium risk supplier when purchasing goods for $10,000.

The Risk Service has two primary functions. The first is to retrieve any risk information stored in the State information Server. The second is to perform a risk calculation based on the information provided to it. When asked to retrieve risk information the Service is passed information on about the subject, target and action set in question. The following algorithm is used in this task:

Search the State Information Server for the information.
If the risk information is present for the subject, target and action(s) then simply return the information found.
If there is risk information present for the target and the actionset, then calculate the cumulative risk for the target-action pair information and provide this as a guide.
If there is no risk information present, then perform a risk calculation and return the result as a guide.

The Risk Calculation Model uses a list of common risks (e.g. receipt of malicious code, refusal to produce goods, service failure, theft of information, fraud, etc.) and their probability of occurring (likelihood), a list of action dependencies, a list of trustors and their maximum allowable losses and risk thresholds and a list of the system resources and their values to help in doing its job. The model also incorporates the use of Josang's Opinion and the Capital Asset pricing Model.

When risk is to be calculated, the service is given subject, target, action(s) and risk id. A risk value is returned after the following steps have been performed:

Use the target action information to find the Maximum Allowable Loss (MAL) and Risk Threshold (RT) for the target.

Uses the risk id to search for the probability of the risk occurring (p).

If the action(s) are dependent on others, then the risk values for all dependent subject-target-action triples are converted to Opinions and a consensus Opinion is created. The new value of p is the belief value produced is used as the new probability.The potential loss of the transaction (L) is either the cost of the transaction or (if a resource) calculated using modified CAPM equation. For the prototype, it is assumed that the potential loss is always with respect to a resource.The Expected Loss (EL) is the product of L and p.

If EL < MAL, then the risk value is ((EL/MAL) * 100) else it is 100.

A design decision was made in situations where dependent actions were identified. As evident from the algorithm, in such a case, a new probability value is calculated using Opinions and not Bayes' Rules. This is done because Bayes' Rules often require information on other dependencies, which the system may be ignorant of.

The architecture of the risk service is based on client-server technology. Risk Service clients run as daemons on user machines and connect to a Risk Service Server when the Service is called upon to perform a task. Any application that can open a socket, read from it and write to it will be able to access the Service (once properly authenticated).

Ignoring the details of authentication, applications can send the following requests to the Risk Service:

- ✓ (**get**, subject, target, actionset) – this retrieves risk information on the specified subject, target and actionset.
- ✓ (**calcRV**, subject, target, actionset, riskID) – this calculates the risk value
- ✓ (**calcEL**, subject, target, actionset, riskID) – this calculates the expected loss

Out of curiosity, Jane, one of the marketing personnel at JamRock wants to know an estimation of the risk involved when the owner views her data. Her machine sends the request (**calcRV**, Jane, Owner, view_emp(ClientBase, Jane), 3). Risk id 3 represents "Theft of Information". Following the algorithm used by the Risk Service Server. The MAL and RT are retrieved for Jane regarding the view_emp action are £3,000 and 0.3 respectively (These figures are gathered by the Monitoring Service and or by the system administrator). The probability, *p*, of risk id 3 occurring is 0.3 (the risks and they probability of occurrence are garnered from E-Commerce studies. Though they are initially set by the system, they can be modified by the administrator). Since this action is not dependent on others, there is no need for the re-calculation of *p*. *L*, the loss, is determined by looking at the relative weighting of the associated asset, in this case information, to the other assets in JamRock. Total Assets in JamRock are worth £60,000 and information contributes 0.10 of this figure (this information is stored in the asset repository). Thus, the loss may be, at worse, £6,000. Thus the expected loss is £1,800 (6,000 * 0.3). Since the EL is lower than the MAL, the risk value would be 60. So, it is fairly risk for Jane.

Let us turn our attention to the Monitoring Service

7 Monitoring Service

The monitoring service provides the State Information Server with up-to-date risk, experience and system state information. The architecture of the Monitoring Service is similar to that of the Risk Service. However, communication is uni-directional. No response is returned to the user application. However, upon adding state information, the Monitoring Service Server performs an analysis of the information in the State Information Server and the Specification Server, against the template of conflicts and ambiguities supplied with the SULTAN System. If a potential conflict is detected then the system sets a flag to alert the administrator the next time the Specification Editor or Analysis Tool is used by him.

The Monitoring Service client has two interfaces: A GUI-based one (Figure 10), which is accessible only to the system administrator, and the socket connection.

Information sent to the Monitoring Service client may have the following format:

(risk, subject, target, actionset, riskvalue) – provides risk information

(experience, subject, target, actionset, expvalue) – provides experience information

(attribute, value) – provides state information

Fig. 11. GUI-based interface for Monitoring Service Client

It is the monitoring service that enables the system to learn and adapt. Suppose, Tom is a new employee at JamRock and he is skeptical about the owner. Initially, the default setting for a skeptical new employee would be not_compromised(Tom) is false. This would imply that the owner is not trusted to manipulate Tom's employee's files (however, this does not stop the owner from doing so anyway). Over time, Tom's confidence in the owner's integrity may increase and at a certain point, the not_compromised(Tom) attribute may be set to true. Note that trust and access control are not concerned the same entities. Trust is only viewed in terms of access control when one is using trust policies for refinement to access control policies.

Let us focus on related work in the field.

8 Related Work

SULTAN trust management is a culmination of work from the fields on logic-based formalisms of trust and on trust management.

All the logic-based frameworks that have attempted to deal with the issue of trust (namely, Jøsang's subjective logic [12-15], Jones and Firozabadi's model [16], Rangan's model [17], etc) suffer from one or more of the three basic problems: 1) their underlying assumptions make them non-applicable to the distributed framework, 2) there is no associated tool support, or 3) they address a subsection of the trust management problem. The SULTAN Trust Management Framework deals with a large subset of the trust problem, makes no assumptions that may render it unusable in reality and comes with software support.

The trust management solutions developed in the last century (namely, AT&T's PolicyMaker and KeyNote, REFEREE, IBM Role-based Access Control Model [18], Poblano [19], etc.) have the following problems: 1) the responsibility of checking the conditions of the establishment of a trust relationship is entirely the application developer's concern, 2) the relationships are assumed to be monotonic 3) these solutions do not learn from the information available to them and 4) the emphasis is on access control decisions rather than general analysis of trust, whereas SULTAN (with Ponder) can cater for both.

9 Conclusions and Future Work

The main thrust of SULTAN is not to offer a concrete, exhaustive, logic theoretic framework for trust. The primary purpose is to identify and analyse the effects of changing specifications on a business and to utilize these specifications to augment the security of Internet commerce. Modeling the dynamic nature of trust relationships, requires the inclusion of risk and experience information. The use of mnemonic names for trustees and trustors, ensure that identity of entities is a non-issue. Unknown transaction partners can be referred to by some arbitrary symbolic representation. SULTAN allows for the separation of trust management code from application code to support scalable Internet based applications.

The future direction of the SULTAN project involves moving the system from a proof of concept to a production line system. This involves organizing the bases by namespaces to allow context-specific facts to be stored, viewed, analysed and manipulated together. The SULTAN system will also be modified to facilitate reasoning under uncertainty. A generic trust establishment framework that not only allows SULTAN trust rules to be used for establishing relationships, but also allow the use of other trust establishment protocol systems (e.g. Trustbuilder [20]).

Acknowledgments

We must acknowledge the help of our colleagues in the Policy Group at Imperial College who provided comments on this work, namely, Naranker Dulay, Emil Lupu, Alessandra Russo and Marek Sergot.

References

1. Blaze, M., J. Feigenbaum, and A.D. Keromytis. *KeyNote: Trust Management for Public-Key Infrastructures*. in *Security Protocols International Workshop*. 1998. Cambridge, England. http://www.cis.upenn.edu/~angelos/Papers/keynote-position.ps.gz.
2. Blaze, M., *Using the KeyNote Trust Management System*, . 1999, AT&T Research Labs. http://www.crypto.com/trustmgt/kn.html.
3. Blaze, M., *et al.*, *RFC2704 - The KeyNote Trust Management System (version 2)*. 1999. http://www.crypto/papers/rfc2704.txt.
4. Chu, Y.-H., *et al.*, *REFEREE: Trust Management for Web Applications*. 1997, AT&T Research Labs. http://www.farcaster.com/papers/www6-referee/.
5. Chu, Y.-H., *Trust Management for the World Wide Web*. 1997. Massachusetts Institute of Technology. http://www.w3.org/1997/YanghuaChu/.
6. Jim, T. *SD3: a trust management system with certified evaluation*. in *IEEE Symposium on Security and Privacy*. 2001. Oakland, California, USA: IEEE

Computer Society. http://www.research.att.com/~trevor/papers/JimOakland2001.pdf.
7. Blaze, M., J. Feigenbaum, and J. Lacy. *Decentralized Trust Management*. in *IEEE Conference on Security and Privacy*. 1996. Oakland, California, USA: IEEE. http://www.crypto.com/papers/policymaker.pdf.
8. Grandison, T. and M. Sloman. *Specifying and Analysing Trust for Internet Applications*. in *2nd IFIP Conference on e-Commerce, e-Business, e-Government (I3E2002)*. 2002. Lisbon, Portugal. http://www.doc.ic.ac.uk/~mss/Papers/I3e2002.pdf.
9. Grandison, T., *Trust Specification and Analysis for Internet Applications*. MPhil/PhD Transfer Report. 2001, Imperial College of Science, Technology and Medicine: London. http://www.doc.ic.ac.uk/~tgrand.
10. Verissimo, P. and L. Rodrigues, *Distributed Systems for System Architects*. 2001: Kluwer Academic Publishers.
11. Gagnon, E., *SableCC: An Object-Oriented Compiler Framework*, MSc Thesis. 1998, McGill University: Montreal.
12. Jøsang, A. *Artificial Reasoning with Subjective Logic*. in *2nd Australian Workshop on Commonsense Reasoning*. 1997. http://www.idt.ntnu.no/~ajos/papers.html.
13. Jøsang, A. *Prospectives for Modelling Trust in Information Security*. in *Australasian Conference on Information Security and Privacy*. 1997: Springer. http://www.idt.ntnu.no/~ajos/papers.html.
14. Jøsang, A. *A subjective metric of authentication*. in *5th European Symposium on Research in Computer Security (ESORICS'98)*. 1998: Springer-Verlag. http://www.idt.ntnu.no/~ajos/papers.html.
15. Jøsang, A. *The right type of trust for distributed systems*. in *ACM New Security Paradigms Workshop*. 1996. http://www.idt.ntnu.no/~ajos/papers.html.
16. Jones, A., J. I. and B.S. Firozabadi. *On the characterisation of a Trusting agent - Aspects of a Formal Approach*. in *Workshop on Deception, Trust and Fraud in Agent Societies*. 2000.
17. Rangan, P.V. *An Axiomatic Basis of Trust in Distributed Systems*. in *Symposium on Security and Privacy*. 1988. Washington, DC: IEEE Computer Society Press.
18. IBM, *IBM Trust Establishment Policy Language*. http://www.haifa.il.ibm.com/projects/software/e-Business/TrustManager/PolicyLanguage.html.
19. Chen, R. and W. Yeager, *Poblano: A Distributed Trust Model for Peer-to-Peer Networks*. 2000. Sun Microsystems. http://www.ovmj.org/GNUnet/papers/jxtatrust.pdf.
20. Winsborough, W., K. Seamons, and V. Jones, *Automated Trust Negotiation: Managing Disclosure of Sensitive Credentials*. 1999, Transarc Report.