

Imperial College of Science, Technology and Medicine

University of London

Department of Computing

Trust Management for Internet Applications

Tyrone W. A. Grandison

A thesis submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy in the Faculty of Engineering of the University of London, and for the Diploma of the Imperial College of Science, Technology and Medicine

London, July 2003

(To my parents, Lloyd and Pearline)

Abstract

The Internet is now being used for commercial, social and educational interactions, which previously relied on direct face-to-face contact to establish trust relationships. Consequently, there is a need to establish and evaluate trust relationships relying only on electronic interactions over the Internet. For example, trust plays an important role in all e-commerce interactions. Customers must trust that sellers will provide the services they advertise, and will not disclose private customer information. Trust in the supplier's competence and honesty will influence the customer's decision as to which supplier to choose. Sellers must trust that the buyer is able to pay for goods or services, is authorised to make purchases on behalf of an organisation or is not underage for accessing services or purchasing certain goods. Thus, trust management has to be an intrinsic part of e-commerce for it to achieve the same acceptance levels as traditional commerce. However, business transactions span multiple organisations, possibly in different countries and not all of these administrative domains may be trusted to the same degree. A domain may need to support a range of different trust relationships and hence be capable of supporting different types of security management policy. Applications will need to be able to navigate through these possibly inconsistent trust relationships. There is a need for a general-purpose trust management system that supports specification and reasoning about trust and its relationship to risk and experience for Internet applications. Trust management involves the specification of trust requirements, the analysis of these requirements to ascertain possible conflicts and the use of risk and experience information to aid in the on-line monitoring of these trust relationships.

This thesis presents the SULTAN Trust Management framework, which consists of a simple notation for specifying trust and recommendation concepts and a set of tools for specifying, analysing and monitoring trust specifications. The SULTAN notation is simple compared to other trust notations and caters for the key concepts relating to a trust relationship between a *trustor*, the subject that trusts a target entity, called the *trustee*; the specific context with associated level of trust, and a set of constraints which must be true for the trust relationship to hold. Recommendations have similar components and the notation also caters for distrust and negative recommendation specifications. The toolkit includes a compiler for translating the Sultan notation into Prolog, and a Prolog-based analysis query builder, as well as pre-defined, typical queries relating to detecting conflicts or inconsistencies in the trust specification. In addition to queries about the source specifications, the system caters for queries that are related to the actual trust scenarios for which state information is held in the system. A monitoring service updates the experience information on which trust is based. The toolkit also includes a

risk service for evaluation of simple risk queries, which can form constraints within a trust or recommendation.

This thesis also suggests how the SULTAN trust management system can be used to enforce security, to help in access control and generally to make informed trust decisions. Abstract trust specifications can be refined to lower-level security authorisations or obligations that are related to encryption. Authorisation policy can query the trust database for policies dependent on current levels of trust etc.

This thesis presents a novel analysis of the trust literature, giving a useful classification scheme for trust contexts and the properties of trust while highlighting the relationship between trust, risk and experience.

Acknowledgements

I must first thank my supervisor Professor Morris Sloman. This thesis would not have been possible without him. His guidance, understanding, criticism and support were inspiring and motivational. I wholeheartedly thank Morris for organising the financial support required for me to pursue my PhD studies. I am also grateful to all the other members of the Distributed Software Engineering Group at Imperial College, in particular Dr. Emil Lupu, Dr. Alessandro Russo and Dr. Naranker Dulay. Their input has been instrumental in this project.

I would also like to thank my colleagues at Imperial College for their friendship and lively discussions, namely: Dr. Nicodemus Damianou, Dr. Ioannis Georgiadis, Leonidas Lymberopoulos and Siv Sivzattian. To all my other friends in the Department of Computing whose names I have forgotten to mention, thank you.

Special thanks to my friends and family who have been a beacon of strength. To my best friend, Prudence Kahawa. Thank you for understanding when I needed a break and when I needed solitude. To my parents, Lloyd and Pearline, to whom this thesis is dedicated. Thanks for providing me with the inspiration to never give up.

Table of Contents

ABSTRACT	III
ACKNOWLEDGEMENTS	V
TABLE OF CONTENTS	VI
LIST OF FIGURES	XI
LIST OF TABLES	XIV
LIST OF ABBREVIATIONS	XV
CHAPTER 1 INTRODUCTION	17
1.1 MOTIVATION.....	20
1.2 REQUIREMENTS FOR TRUST MANAGEMENT.....	21
1.3 OBJECTIVES	22
1.4 CONTRIBUTION	23
1.5 THESIS STRUCTURE	24
CHAPTER 2 BACKGROUND AND RELATED WORK	26
2.1 TRUST DEFINITIONS.....	26
2.2 TRUST PROPERTIES	30
2.2.1 <i>Context of Trust Relationship</i>	30
2.2.2 <i>Arity of a Trust Relationship</i>	30
2.2.3 <i>Measurability of a Trust Relationship</i>	30
2.2.4 <i>Applying Mathematical Properties To Trust Relationships</i>	31
2.2.5 <i>Relationship Indicators</i>	33
2.3 TRUST CLASSIFICATION.....	34
2.3.1 <i>Access to a Trustor's Resources</i>	35
2.3.2 <i>Provision of Service by the Trustee</i>	36
2.3.3 <i>Certification of Trustees</i>	37
2.3.4 <i>Delegation</i>	38
2.3.5 <i>Infrastructure Trust</i>	38
2.3.6 <i>Dominant Attributes for Trust Contexts</i>	39
2.4 TRUST FORMALISMS	41
2.4.1 <i>Logic-based Formalisms</i>	42
2.4.2 <i>Computational Models</i>	45
2.4.3 <i>Human Computer Interaction (HCI) Based Models</i>	46
2.5 VIEWS OF TRUST MANAGEMENT.....	48
2.6 CONTEMPORARY TRUST MANAGEMENT SOLUTIONS.....	50
2.6.1 <i>Public Key Certificates</i>	50
2.6.2 <i>PICS</i>	51

2.6.3 <i>PolicyMaker and KeyNote</i>	55
2.6.4 <i>REFEREE</i>	58
2.6.5 <i>SD3</i>	60
2.6.6 <i>Fidelis</i>	61
2.6.7 <i>IBM Trust Establishment Framework</i>	62
2.6.8 <i>Trustbuilder Framework</i>	64
2.6.9 <i>TCPA</i>	66
2.6.10 <i>Poblano Distributed Trust Model</i>	68
2.6.11 <i>Emerging Trust Management Solutions</i>	68
2.7 SUMMARY.....	70
CHAPTER 3 SPECIFYING TRUST	72
3.1 REQUIREMENTS FOR A TRUST NOTATION	72
3.2 THE SULTAN SPECIFICATION NOTATION	73
3.2.1 <i>The trust construct</i>	73
3.2.2 <i>The recommend construct</i>	74
3.2.3 <i>Specifying Policy Names</i>	76
3.2.4 <i>Specifying Entity Names</i>	76
3.2.5 <i>Specifying Levels</i>	76
3.2.6 <i>Specifying Action Sets</i>	77
3.2.7 <i>Specifying Constraints</i>	78
3.2.8 <i>The Trust-Recommendation Interaction</i>	82
3.3 MODELLING OTHER NOTATIONS	83
3.3.1 <i>Public Key Certificates</i>	83
3.3.2 <i>PICS</i>	84
3.3.3 <i>PolicyMaker</i>	85
3.3.4 <i>KeyNote</i>	86
3.3.5 <i>REFEREE</i>	87
3.4 THE SPECIFICATION PROCESS.....	88
3.4.1 <i>Organizational Diagram Construction</i>	89
3.4.2 <i>SULTAN Rule Specification</i>	91
3.5 SUMMARY.....	92
CHAPTER 4 ANALYSING TRUST	93
4.1 REQUIREMENTS FOR ANALYSIS	93
4.2 HOW TO ANALYSE IN THE SULTAN TMF	94
4.2.1 <i>Analysis on the specification source</i>	95
4.2.2 <i>Analysis about a scenario</i>	97
4.2.3 <i>Detecting cycles</i>	99
4.2.4 <i>Identifying constraints to be satisfied</i>	100
4.3 GENERIC ANALYSIS QUERIES	101
4.3.1 <i>Trust-Recommend Conflict</i>	101

4.3.2 <i>Recommend Redundancy</i>	102
4.4 SUMMARY.....	103
CHAPTER 5 RISK IN TRUST MANAGEMENT	105
5.1 RISK MODELS	106
5.1.1 <i>Quantitative Model</i>	106
5.1.2 <i>Qualitative Model</i>	107
5.1.3 <i>Software Development Risk Model</i>	108
5.2 THE PROBLEMS WITH THE RISK MODELS	110
5.3 SULTAN RISK MODEL	111
5.3.1 <i>Determining Risks and their Probability of Occurrences</i>	112
5.3.2 <i>Determining Potential Losses</i>	113
5.3.3 <i>Handling Dependencies</i>	114
5.3.4 <i>Determining Risk Profiles</i>	115
5.3.5 <i>Calculating Risk</i>	116
5.3.6 <i>Retrieving Risk Information</i>	117
5.4 SUMMARY.....	118
CHAPTER 6 EXPERIENCE, MONITORING AND RE-EVALUATION.....	120
6.1 EXPERIENCE.....	121
6.1.1 <i>Experience Representation</i>	121
6.1.2 <i>Usage Strategies</i>	121
6.2 MONITORING	123
6.2.1 <i>Active Design Architecture</i>	123
6.2.2 <i>Passive Design Architecture</i>	124
6.2.3 <i>The SULTAN Monitor Architecture</i>	124
6.2.4 <i>Updating entity connections</i>	126
6.2.5 <i>Updating risk, experience and state information</i>	126
6.2.6 <i>Updating action dependency information</i>	127
6.2.7 <i>Updating risk profiles</i>	128
6.3 RE-EVALUATION	128
6.4 SUMMARY.....	129
CHAPTER 7 SULTAN TRUST MANAGEMENT	131
7.1 TRUST MANAGEMENT LIFE CYCLE	132
7.2 BASIC DATA STRUCTURES.....	133
7.2.1 <i>Specification Server</i>	134
7.2.2 <i>Entity-Connections Server</i>	134
7.2.3 <i>State Information Server</i>	135
7.2.4 <i>Risk Likelihood Server</i>	136
7.2.5 <i>The Other Risk Calculation Oriented Structures</i>	137
7.2.6 <i>The Other Analysis Oriented Structures</i>	137
7.2.7 <i>Basic Data Structure Overview</i>	137

7.3 SPECIFICATION EDITOR	139
7.3.1 <i>Standard Editor</i>	139
7.3.2 <i>Compiler</i>	141
7.3.3 <i>AST Walker</i>	142
7.3.4 <i>SULTAN to Prolog Translator</i>	143
7.3.5 <i>External Translators</i>	144
7.3.6 <i>Software Hooks</i>	145
7.4 ANALYSIS TOOL	145
7.4.1 <i>The Console</i>	147
7.4.2 <i>The Loader</i>	148
7.4.3 <i>The Viewer</i>	148
7.4.4 <i>Query Statement Builder</i>	150
7.4.5 <i>State Manager</i>	150
7.5 TRUST MONITOR	151
7.6 RISK SERVICE	153
7.7 TRUST CONSULTANT	155
7.8 SUMMARY	157
CHAPTER 8 USES OF THE SULTAN TMF	158
8.1 SIMULATION ANALYSIS	158
8.2 USING SULTAN WITH PONDER	159
8.2.1 <i>Using SULTAN in Ponder policies</i>	159
8.2.2 <i>Refinement of SULTAN rules to Ponder policies</i>	161
8.3 NEGOTIATION	164
8.4 CONTRACT EVALUATION	164
8.5 RECOMMENDATION FORMATION	164
8.6 INFRASTRUCTURAL SECURITY	165
8.7 ACCESS CONTROL DECISIONS	165
8.8 RESOURCE ALLOCATION	166
8.9 SUMMARY	166
CHAPTER 9 CASE STUDY	167
9.1 OVERVIEW	167
9.2 THE PLAYERS	167
9.3 PROCESSES AND INFRASTRUCTURE	168
9.4 INITIALISATION TASKS	168
9.4.1 <i>Organizational Chart Diagram</i>	169
9.4.2 <i>Asset Repository Construction</i>	170
9.4.3 <i>Risk Profile Data</i>	170
9.5 SPECIFICATION	172
9.6 ANALYSIS	175
9.7 INFORMATION COLLECTION	179

9.8 APPLICATION USE.....	181
9.9 SUMMARY.....	183
CHAPTER 10 CRITICAL EVALUATION	184
10.1 RELATIONSHIP TO RELATED WORK	184
10.2 EVALUATION OF THE FRAMEWORK.....	187
<i>10.2.1 Specification Language Design.....</i>	<i>187</i>
<i>10.2.2 Analysis Model Design.....</i>	<i>189</i>
<i>10.2.3 Architecture Design.....</i>	<i>189</i>
10.3 EVALUATION OF THE IMPLEMENTATION	191
10.4 SUMMARY	193
CHAPTER 11 CONCLUSIONS.....	194
11.1 REVIEW AND DISCUSSION.....	194
11.2 FUTURE WORK.....	198
<i>11.2.1 Specification Language.....</i>	<i>198</i>
<i>11.2.2 Analysis Model.....</i>	<i>198</i>
<i>11.2.3 Architecture.....</i>	<i>199</i>
<i>11.2.4 Implementation.....</i>	<i>199</i>
<i>11.2.5 General.....</i>	<i>199</i>
11.3 CLOSING REMARKS	200
BIBLIOGRAPHY	201
APPENDIX A SYNTAX SPECIFICATION.....	212
APPENDIX B MISSING DEFINITIONS.....	217
APPENDIX C REFINING SULTAN TO TRUST RULES.....	219
APPENDIX D SULTAN ANALYSIS MODEL.....	226
APPENDIX E TEMPLATE OF CONFLICTS AND AMBIGUITIES	244
APPENDIX F SULTAN SPECIFICATIONS MODELLING CONTEXTS	248

List of Figures

Figure 1.1: Trust Relationship	17
Figure 2.1: Service Provision Trust with Competence	39
Figure 2.2: Service Provision Trust with Honesty Factor Dominant.....	40
Figure 2.3: Access to Trustor Resources Trust with Competence Factor Dominant.....	40
Figure 2.4: Access to Trustor Resources Trust with Dependability and Timeliness Factors Dominant.....	40
Figure 2.5: Trustbuilder Negotiation Process	65
Figure 3.1: Trust-based Recommendation Scenario	83
Figure 3.2: Bob’s Music Warehouse (BMW).....	89
Figure 3.3: Organization Diagram for BMW.....	89
Figure 4.1: The link between Specification and Analysis in the SAM.....	94
Figure 4.2: Analysis Types	95
Figure 5.1: A typical risky transaction.....	107
Figure 5.2: Interaction of the Components of a Quantitative Risk Model.....	108
Figure 5.3: The ‘Risk Assessment in Trust Management’ Issues.....	112
Figure 5.4: Stereotypical Example of Resource Value Calculation.....	113
Figure 5.5: Risk Calculation in the SRS	117
Figure 5.6: SRS RISK Information Retrieval	118
Figure 6.1: Experience, monitoring and re-evaluation.....	120
Figure 6.2: A Generalised Trust Monitor.....	123
Figure 6.3: Overview of SULTAN Monitoring.....	125
Figure 6.4: Computer Id Generation	125
Figure 6.5: Monitoring and Analysis	128
Figure 7.1: SULTAN Tools and their interactions to the external system.....	132
Figure 7.2: SULTAN Trust Management Life Cycle	133
Figure 7.3: Specification Server.....	134
Figure 7.4: Entity-Connections Server.....	135
Figure 7.5: State Information Server.....	136
Figure 7.6: Risk-Likelihood Server	136
Figure 7.7: Data Structures, Tools and their connections	138
Figure 7.8: Components of the Specification Editor.....	139
Figure 7.9: Snapshot of the Specification Editor	140
Figure 7.10: Mini-Editor used to update Entity-Connections Database	140

Figure 7.11: SULTAN Compiler Processes.....	141
Figure 7.12: Compiled Specifications for BMW	142
Figure 7.13: AST Walker run on BMW specifications.....	143
Figure 7.14: Using the SULTAN to Prolog translator on the BMW specifications	144
Figure 7.15: Adding an External Translator	145
Figure 7.16: Software Hook to the Risk-Profile Mini-Editor	146
Figure 7.17: Analysis Tool – Analysis Engine Connection	147
Figure 7.18: Snapshot of the Analysis Tool.....	148
Figure 7.19: Loading a file.....	149
Figure 7.20: Using the Viewer on the Template	149
Figure 7.21: SULTAN Query Statement Builder (First Level)	150
Figure 7.22: Source Analysis using the SULTAN Query Statement Builder.....	151
Figure 7.23: More detailed SULTAN Monitor Architecture	152
Figure 7.24: SM Client Socket Server Interactions	152
Figure 7.25: Architecture of the Risk Service.....	154
Figure 7.26: Snapshot of Admin’s interface to Risk Service.....	154
Figure 7.27: Architecture of the Sultan Trust Consultant (STC).....	155
Figure 8.1: Source Trust Conflict for BMW	159
Figure 8.2: Deletion then Source Trust Conflict for BMW	159
Figure 9.1: Organizational Chart for ResWorld.....	169
Figure 9.2: Compilation Results for ResWorld	175
Figure 9.3: Analysis Result for Source Trust Conflict Query.....	176
Figure 9.4: Analysis Result for Source Conflict of Interest Query.....	177
Figure 9.5: Analysis Result for Source Separation of Duties Query	178
Figure 9.6: Re-evaluation Results.....	180
Figure 9.7: Translated Trust Consultation Query	181
Figure 9.8: Translated Experience Consultation Query	182
Figure 10.1: General Conceptual Structure of Contemporary Trust Management Solutions ...	186
Figure 10.2: Basic Architecture of the SULTAN TMF	190
Figure F1: Access to Trustor Resources Trust.....	248
Figure F2: Provision of Service by the Trustee Trust.....	249
Figure F3: Another example of Provision of Service by the Trustee Trust	249
Figure F4: Certification Trust (Phase One).....	250
Figure F5: Certification Trust (Phase Two).....	250

Figure F6: Delegation Trust.....	251
Figure F7: Infrastructure Trust.....	252

List of Tables

Table 3.1: Key for BMW Organization Chart	90
Table 3.2: Action Abstractions for BMW	91
Table 5.1: Risk Metrics for Jane & Mike.....	107
Table 7.1: Abbreviations for Data Structure Chart.....	138
Table 9.1: Key for ResWorld's Organization Chart	169
Table 9.2: Initial set of actions for ResWorld.....	171
Table 9.3: Additional actions for ResWorld	172

List of Abbreviations

ALE	Annual Loss Expectancy
ASL	Authorization Specification Language
ASP	Application Service Provider
AST	Abstract Syntax Tree
BAN	Burrows, Abadi and Needham
BIOS	Basic Input Output System
BMA	British Medical Association
BMW	Bob's Music Warehouse
codat	Code and Data
CMC	Computer Mediated Communication
EAC	Estimated Annual Cost
ECTR	Electronic Commerce Trust Relationship
EG	Expected Gain
EL	Expected Loss
GO	Guard Object
HCI	Human Computer Interaction
IT	Interface Thread
LALR	Look Ahead Left-to-right parse, Rightmost-derivation
MAL	Maximum Allowable Loss
MAS	Multi Agent Systems
OS	Operating System
PAL	Property-based Authentication Language
PD	Prisoner's Dilemma
pdf	Probability Distribution Function
PGP	Pretty Good Privacy

PICS	Platform for Internet Content Selection
PKI	Public Key Infrastructure
RAID	Redundant Arrays of Independent Disks
RAL	Role-based Authorization Language
REFEREE	Rule-controlled Environment For Evaluation of Rules and Everything Else
RMI	Remote Method Invocation
RT	Risk Threshold
SAM	SULTAN Analysis Model
SD3	Secure Dynamically Distributed Datalog
SM	SULTAN Monitor
SRS	SULTAN Risk Service
STC	SULTAN Trust Consultant
SULTAN	Simple Uniform Logic-oriented Trust Analysis Notation
TCB	Trusted Computing Base
TCPA	Trusted Computing Platform Alliance
TEF	Trust Establishment Framework
TMF	Trust Management Framework
TPM	Trusted Platform Module
TPL	Trust Policy Language
UCI	Unique Computer Id
URL	Uniform Resource Locator
XML	Extensible Markup Language

Chapter 1 Introduction

“Without trust we cannot stand.”
– Confucius [1]

The concept of trust has been widely studied in many other fields, namely: psychology, sociology, business, political science, law and economics. There is a wealth of information on trust as it pertains to the human experience. Trust permeates every activity that is performed and is a key facilitator for current commercial transactions. Encapsulating trust in Internet applications is also a key enabler for Internet Commerce. A trust relationship is usually specified between a *trustor*, the subject that trusts a target entity, which is known as the *trustee* i.e. the entity that is trusted (see Figure 1.1). In this thesis, and for the context of Internet applications, trust is defined as:

“the quantified belief by a trustor with respect to the competence, honesty, security and dependability of a trustee within a specified context.” [2-5]

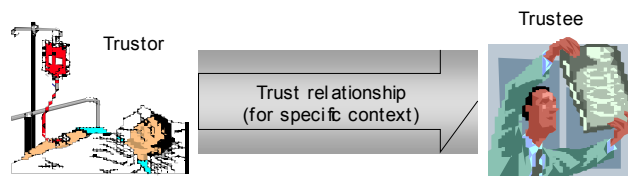


Figure 1.1: Trust Relationship

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. A competent 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 setting, 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 [6]. 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. Figure 1.1 illustrates a typical real-world trust scenario. A

patient trusts a doctor to perform a specific task, say read and interpret X-ray results. This example not only embodies the definition presented, but also illustrates a few hidden points, such as the dominant role played by an attribute or set of attributes in a trust relationship. For example, a particular relationship may have an emphasis on honesty and security, while another may have an emphasis on dependability. In Figure 1.1, the dominant attribute in the trust relationship is the doctor's competence. This issue of the dominance of trust attributes will be further dealt with in Chapter 2. This completes the introduction to the notion of trust. However, the related concept of distrust must also be addressed.

Distrust diminishes the spectrum of possible present and future interactions, while trust does the opposite. Distrust is a useful concept to specify as a means of revoking previously agreed trust or for environments when entities are trusted, by default, and it is necessary to identify some entities which are not trusted. In this context, distrust is defined as:

“the quantified belief by a trustor that a trustee is incompetent, dishonest, not secure or not dependable within a specified context.” [2-5]

The major problem that arises from integrating trust/distrust into the computing world is how to transform a complex social concept into an easy-to-use technical product that embodies the basic principles of trust/distrust.

Though trust modelling and interaction in a computer system is a complicated issue, extra complexity is introduced in the context of the Internet. For a transaction to occur over the Internet, trust must exist between the consumer and: 1) the technology 2) the transaction process, and 3) the producer, proxies, third parties and intermediate software agents. This scenario not only requires a technical solution, but it also requires regulatory and legislative protections on the transactions themselves. This is a plausible requirement, since it is the standard for off-line commercial transactions. Traditional commerce uses legal (dis-) incentives as one of the tools to discourage trust being betrayed. In off-line scenarios, trust can be generated by legislation and regulation, company policy, personal experiences and or pre-existing relationships. Building trust into a system requires the inclusion of many factors (e.g. legislature, insurance, service level agreements, trusting attitudes/propensities, etc.).

Risk and experience are two factors that contribute significantly to the trust decision. Risk refers to the probability of failure of a transaction with respect to a specific context. Experience refers to the cumulative view of the interactions (or rather the outcomes of interactions) with respect to a party within a context. Both concepts are subjective and can be used in the

determination of a trust decision. A very risky venture has a higher chance of being designated a decision not to trust, while a not so risky transaction will normally lead to a positive trust decision. Entities with a positive experience stand a higher chance of being trusted for future interactions, while entities that you have a negative experience with will normally lead to a distrust decision. There is the additional concept of reputation, which is sometimes confusingly used in place of experience and/or anonymous recommendation. Reputation refers to recommendations based on a third parties' perspective that the subject is willing to accept and use. These recommendations may be: 1) directed or undirected, and 2) anonymous or signed. The dominant view taken by current recommendation systems, such as Ebay's and Amazon's, is that a recommendation is by default undirected and anonymous. This view is normally translated to be the dominant view of reputation. However, reputation is a superset of this perspective. Thus, there is a difference between reputation, anonymous recommendation and experience. Having presented the concepts of trust and distrust and highlighted the fact that risk and experience are important contributors to the trust/distrust phenomena, the concept of trust management is now defined.

The trust management problem is defined in the following context. There is a large domain of heterogeneous systems, each with (possibly) different trust requirements and mechanisms. These systems need to interact securely and seamlessly with each other. They should be able to interoperate irrespective of prior knowledge of another system and should interact intelligently with other systems. Simply put, trust management is the management of trust relationships. Formally, this thesis purports that trust management is:

“the activity of collecting, encoding, 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.” [2, 4, 5].

Evidence may include credentials such as certificates for proof of identity or qualifications, risk assessments, usage. Thus, trust management involves the acts of specifying trust relationships, analysing them to uncover new (and/or unwanted) relationships or side-effects and presenting evidence that can be used to make trust decisions. This evidence should be collected from the source of the interactions and should be used to allow the subject to adapt his trust requirements based on this (new information). Thus, trust management also involves the monitoring and (re)-evaluation of the subject's trust information. This perspective is the view taken by the Trust Management Framework (TMF) that will be described in this thesis.

1.1 Motivation

Trust is a vital component in every business transaction. Customers must trust that sellers will provide the services they advertise, and will not disclose private customer information (name, address, credit card details, purchases etc.). Trust in the supplier's competence and honesty will influence the customer's decision as to which supplier to choose. Sellers must trust that the buyer is able to pay for goods or services, is authorised to make purchases on behalf of an organisation or is not underage for accessing services or purchasing certain goods. Thus, for Internet commerce to achieve the same levels of acceptance as traditional commerce, trust has to be an integral element of Internet applications. Internet services are increasingly being used in daily life for electronic commerce, web-based access to information and inter-personal interactions via electronic mail rather than voice or face-to-face, but there is still major concern about the trustworthiness of these services. There is a need for a high-level means of specifying and managing trust, which can be easily integrated into applications and used on any platform. Typical applications requiring a formal trust specification include content selection for web documents [7], medical systems [8], telecommuting [9], mobile code and mobile computing [10-12], as well as electronic commerce [13-20]. The main motivation in studying trust management is to help the consumer to make more informed decisions.

The migration from centralised information systems to Internet-based applications will mean that transactions have to span a range of domains and organisations [21], not all of which may be trusted to the same degree. There is also the added complication that a domain may need to support a range of different trust relationships and hence be capable of supporting different types of security policy [22]. For example, IBM Research Labs may support two different trust systems, say A and B. A uses a declarative programming language to verify that the public keys of target entities can be trusted and B uses a logic-based language that produces a proof or disproof of the target's trustworthiness for the specified context. A and B employ two very different trust architectures or topologies. Trust relationships specified in the language used by A may have problems (or may lead to problems) when used by B's trust topology. Suppose target X, which has its trust information encoded in a functional trust framework MX6, wishes to interact with IBM Research Labs. Which trust architecture should or can be used? Is there a hierarchy of trust with respect to A and B from X's perspective? Inconsistencies such as these, both within a domain and across domains, highlight the need for a flexible, general-purpose trust management system that can navigate these (possibly) complex trust domains.

Currently, trust decisions are hard-coded into an application. This adds to the complexity of the application, increases its inability to adapt to changes in trust and augments its inflexibility when trying to set up new relationships. A separation of the application's purpose and its Trust Management Framework (TMF) will offer a more scalable and flexible solution for the distributed environment. This separation leaves the application to focus on its core functionality. This scenario should lead to a smaller application code base and fewer application bugs. Thus, this thesis posits that the management of trust at an abstract level, distinct from the application implementation, will lead to less buggy application source.

Trust forms the basis for allowing a trustee to use or manipulate resources owned by a trustor or may influence a trustor's decision to use a service provided by a trustee. Thus, trust can form an important factor in decision-making [23-25]. Trust can be used by the consumer to help in making decisions or it can be used to automate the decision-making relating to system resources, for example, trust information may be used to determine the amount of CPU time given to different entities, who are trusted at different levels to perform activities of varying importance to the trustor. Thus, a useful side-effect of trust management is that it can be used as the starting point for subsequent refinement into security policies related to authorisation and management of security [26].

1.2 Requirements for Trust Management

When developing a Trust Management Framework, there are a number of issues that must be addressed. A TMF should possess the following facilities:

- A clear and semantically well-defined means of specifying trust relationships, i.e. a trust specification language. This language should also be expressive enough to allow the specification of relationships that require a diverse combination of trust conditions. These conditions help to determine if the relationships should be established or not.
- A platform that facilitates the examination of the trust relationships to discover (and remove) unwanted assumptions.
- A design that integrates the notion of trust non-monotonicity, i.e. the addition of new trust relationships means that the current set of relationships must be updated. Formally, the Encyclopedia of Cognitive Science [27] states that:

“a logic is called non-monotonic if, given a theory in the logic, adding new information to the theory may cause one to retract some conclusions which were previously made.” [27]

- An architecture that is scalable and that facilitates trust decision-making with respect to Internet applications.

1.3 Objectives

The Trust Management Framework that will be described in this thesis is called the SULTAN TMF. This TMF is to be used by a system administrator with a global view of the system resources and needs. The system allows the administrator to encode the trust requirements for her domain in a trust notation and then to analyse her requirements to ascertain the presence of latent (and possibly harmful) relationships and to uncover unnecessary (and or unwanted) assumptions that she has made in her specification. In order to achieve these objectives, the specification notation should be sufficiently high-level to specify relationships between computer hardware, public keys, user proxies, etc. The notation should also be flexible enough to handle positive and negative trust relationships and the specification of the fact that the trustee may be trusted not to perform some actions. The analysis of trust relationships requires a framework that is expressive enough to allow the administrator to ask both general queries and queries that are particular to her organization. The analysis framework should have facilities for assisting in the query creation process and for handling common problems relating to established assertions such as cycle detection. Thus, the primary objectives of the SULTAN TMF is to provide the administrator with a useful specification and analysis tool.

Although specifically a tool for the administrator, the SULTAN TMF should be useable by a domain user who wishes to make a more informed decision relating to engaging in a transaction. In this scenario, the system helps the user by answering a question posed by the user. Generally, the answer will provide the domain user with information, based on the experience of all the domain users, the trust information entered by the administrator and data gathered by the SULTAN TMF, that may help him in making a decision. Thus, an added objective of the SULTAN TMF is to provide the domain user with a decision making tool. In order for such decision-making facilities to be provided, the TMF should contain facilities for the collection of experience information. This information will also help in determining if trust relationships should be established, terminated or re-established. In order to model the dynamic

nature of a trust relationship (i.e. it changes over time), information applicable to trust relationships need to be continuously re-evaluated to ascertain if relationships are still valid. It becomes obvious that in order for a TMF to function as an effective decision-making tool, it should contain a facility for monitoring and evaluating information pertinent to the trust relationships. Due to the fact that risk plays a significant role in trust decision-making, risk management has to be an important part of a TMF. The level of trust could be inversely related to the level of risk, ie. high risk, low trust. In order to include facilities for addressing risk, the TMF must be able to store and retrieve risk information and be able to calculate risk. Thus, a further requirement of a TMF that is used as a decision-making tool is that it should include a risk management component.

The SULTAN TMF is neither a complete mapping of the social concept of trust nor a computational model of trust for the computing environment. The SULTAN Framework addresses the issue of specifying trust relationships, analysing those relationships and using all the information related to these relationships to enable decision making.

1.4 Contribution

The SULTAN Trust Management Framework is a framework that is designed to facilitate the management of trust relationships. It is a collection of specification, analysis and management tools. The primary specification tool is the Specification Editor, which integrates a specification notation, compiler for the notation, an example translator from SULTAN to Prolog and standard storage and retrieval facilities. The example translator illustrates that the SULTAN specification notation can be easily translated to a diversity of languages. Chapter 3 demonstrates that the SULTAN notation can be mapped into current trust policy languages, which will be discussed further in Chapter 2. This implies that the SULTAN specification notation can be used as a cross-domain specification platform. For analysis, the key tool is called the Analysis Tool, which incorporates an analysis notation, a query statement builder that makes it easier to formulate queries and a template of queries common to most situations, e.g. conflict of interest, separation of duties, implicit (and possibly dangerous) relationships, etc. The TMF includes a risk service, a trust monitor and a trust consultant. The risk service encapsulates the TMF's risk management strategy. The risk service collects risk information and performs risk calculations. The trust monitor keeps the information in the SULTAN TMF up to date, by gathering information on the outcome of transactions that involve domain users.

The trust consultant is the component that the domain user interacts with. The user may ask questions that may help in the determination of a trust decision.

The novel aspects of the SULTAN TMF are that 1) the specification notation is high-level and refinable to many lower level languages 2) the analysis notation is highly expressive, i.e. a wide range of queries may be constructed 3) the TMF includes a component that handles risk 4) information is (re-)evaluated to ascertain if relationships should remain unchanged, i.e. trust relationships are monitored in order to re-evaluate trust relationships 5) experience (and other information) is used to enable better decision making.

1.5 Thesis Structure

In Chapter 2, background information on trust and trust management is presented. Current definitions of trust and their implications are discussed. A review of the properties of trust relationships will be given and a trust classification scheme introduced. A survey of the contemporary approaches to trust management, recent trust management solutions and their problems will be provided.

Chapter 3 presents the SULTAN specification notation. It starts by providing a definition and a discussion of trust as it applies to Internet applications. Then the requirements for E-Commerce are highlighted and a detailed overview of the SULTAN Trust Model and Specification language are given.

In Chapter 4, the issue of how to perform analysis using the SULTAN TMF is presented. A discussion on how to specify analysis queries is provided and then a template of generic queries is presented and examples are given to illustrate the use of the analysis notation.

Chapter 5 highlights how the SULTAN TMF incorporates the concept of risk. Particular emphasis is placed on the SULTAN risk service: its functions and basic operation.

Chapter 6 describes the use of experience, monitoring and (re-)evaluation in the TMF. A brief discussion on experience and its use is provided. The basics of the monitoring system are also provided and the topic of the (re-)evaluation of trust specifications is presented.

Chapter 7 is devoted to the SULTAN toolkit. The basic data structures, and the architecture of the prototype developed to illustrate the ideas presented in this thesis. The components of the tools used in this prototype will also be discussed.

Chapter 8 presents a description of the ways in which the framework can be used on the Internet. A general discussion on the possible uses is presented, with examples for added clarity.

Chapter 9 provides a case study, which incorporates the SULTAN TMF. A critical evaluation of the framework is done in Chapter 10 and then we conclude and suggest directions for future work in Chapter 11.

Chapter 2 Background and Related Work

“A complete absence of trust would prevent even getting up in the morning.”
– Niklas Luhman [28]

The theoretical underpinnings of a Trust Management Framework should be explicitly stated in order to make the issues that it is addressing unequivocally clear. The work on the SULTAN TMF incorporates and builds on work done in the myriad of research areas. In this chapter, the basic concepts will be discussed. The various perspectives of trust are highlighted and their point of convergence identified. The attributes of a typical trust relationship are explicitly stated. A taxonomy of trust use, which was identified by a literature survey [3], is presented. Various approaches to solving the trust problem are described. Contemporary viewpoints of trust management are discussed and current trust management solutions are presented.

2.1 Trust Definitions

Currently, there is no consensus on the meaning of trust in the field of trust management. Due to the fact that trust is an integral part of human nature, it is normally treated as an intuitive and universally understood concept. However, intuition is determined by people’s experiences. Thus, their view of trust will be different based on their experiences. Many researchers have proposed definitions of trust because they realize that it is unwise to assume an intuitive, universal and well-understood definition of trust. However, definitions vary depending on the researcher’s background, outlook on life and the application domain of the problem being solved.

The Webster dictionary states that trust is:

“An assumed reliance on some person or thing. A confident dependence on the character, ability, strength or truth of someone or something.”, or “A charge or duty imposed in faith or confidence or as a condition of a relationship.” or “To place confidence (in an entity).”

These definitions demonstrate the common interpretations of social context. The Oxford Dictionary says that trust is:

“the firm belief in the reliability or truth or strength of an entity.”

This is a more general view of trust, but it would be difficult to interpret strength generally for computer science.

Kini and Choobineh [29] in their considerations on the theoretical framework of trust, use the Webster dictionary's definition of trust and examine trust from the perspectives of personality theorists, sociologists, economists and social psychologists. They highlight the implications of these definitions and combine their results with the social psychological perspective of trust to create their definition of trust in a system –

“a belief that is influenced by the individual's opinion about certain critical system features.”

This definition only expresses system trust, and excludes dimensions, such as trust in a transaction process and trust in the trustee's competence, honesty, dependability, etc., which are critical for Internet applications. The European Commission Joint Research Centre defines trust as:

“the property of a business relationship, such that reliance can be placed on the business partners and the business transactions developed with them.” [30]

This view of trust is from a business management perspective and neglects to highlight the reliance that will need to be placed in the hardware and software infrastructure required to enable Internet transactions.

According to Lewis and Weigert [31], trust is expressed as:

“observations that indicate that members of a system act according to and are secure in the expected futures constituted by the presence of each other for their symbolic representations.”

Their view entails the notion of system members having expectations. However, the definition seems to assume that expectations are positive. This is not always the case in Internet Commerce. Mayer and Davis [32] introduce an extra dimension in their definition. They view trust as:

“the willingness of a party to be vulnerable to the actions of another party based on the expectation that the other will perform a particular action important to the trustor, irrespective of the ability to monitor or control that other.”

This definition implies an element of being able to monitor trust and re-evaluate a decision. Zand [33] define trust as:

“the willingness to be vulnerable based on positive expectations about the actions of others.”

However, this is not always the case in real commerce. I may be willing to be vulnerable when there is a low value product or service.

In [34], Cural and Judge define trust as:

“an individual’s reliance on another party under conditions of dependence and risk.”

Dependence implies a strong degree of reliance by all parties involved. This implies that there is a mutual relationship between the trustor and trustee. This is not a suitable assumption to make for Internet applications. Mui et al. [35] define trust as

“a subjective expectation an agent has about another’s future behaviour based on the history of their encounters.”

This illustrates the subjective nature of trust. It also demonstrates the need to learn from past experiences.

All the above definitions can be easily justified. They highlight the fact that trust is multi-faceted. However, there is a central theme that can be found in these definitions. This theme is that trust is a measurable, subjective belief about some action (or set of actions), that this belief expresses an expectation [36] and that this belief has implications on the features, properties and or attributes of a system. However, the above definitions do not take into account the particular needs of a networked environment. Each definition is specific to its area of application, namely: business, logic, philosophy, social behaviour, etc. This limits their general use and applicability. This thesis purports that the definition of trust given in Chapter 1 merges the important aspects of the definitions discussed above and also highlights the principles that are most appropriate for Internet application. Not only is trust defined as:

“the quantified belief by a trustor with respect to the competence, honesty, security and dependability of a trustee within a specified context”,

but distrust is also defined as:

“the quantified belief by a trustor that a trustee is incompetent, dishonest, not secure or not dependable within a specified context.”

The definitions make no assumptions about the nature of the entities involved (whether they are humans, software, hardware, etc.), emphasize the facts that trust is subjective and a belief, highlight that trust is context-specific, is related to a property (or group of properties) and illustrate that trust can be either positive or negative.

Current trust management literature [7, 8, 11, 37-59] uses the terms trust, access control, authorization and authentication interchangeably. This is a mistake. The four terms are distinct. To demonstrate the difference between trust and access control, a stereotypical example can be used. Tony trusts Darren to perform network security testing. This does not necessarily imply that Tony will allow Darren access to the network to perform the tests. The principle is simple: trust does not necessarily imply access control rights and vice versa. For some situations, it may be true that trust may lead to access rights being granted, but being trusted does not automatically mean that access will be given. Authorisation can be viewed as a policy decision assigning access control rights for a subject to perform specific actions on a specific target with defined constraints. Thus, 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. Again, a trust relationship may lead to a positive authorisation decision, but it is not a guarantee. 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. Obviously, trust and authentication should not be used interchangeably.

The discussion of definitions of trust here is an abridged one. However, it is sufficient for us to present a representative sample of the definitions in current literature. McKnight et al. [60], Lamsal [61], Gerck [62] and Corritore et al. [63] provide a more detailed discourse on trust definitions in philosophy, sociology, psychology, business management, marketing, ergonomics and human-computer interaction (HCI).

2.2 Trust Properties

Every trust relationship has basic properties to which it adheres. In this section, the characteristics that were identified and subsequently included in the construction of the SULTAN TMF are presented.

2.2.1 Context of Trust Relationship

In general, a trust relationship is not absolute – A will never trust B to do any possible action it may choose. A trustor trusts a trustee with respect to its ability to perform a specific action or provide a specific service within a *context*. For example, a person is only trusted to deal with financial transactions less than \$2000 in value.

A trust relationship is always defined with respect to a scenario, situation or context [64]. Even trust in oneself is not usually absolute and there is a need to protect resources you own from mistakes or accidents you may cause. Examples include protecting files from accidental deletion or mechanisms to prevent a person driving a car when under the influence of alcohol.

2.2.2 Arity of a Trust Relationship

A trust relationship can be one-to-one between two entities. It may be a one-to-many relationship in that it can apply to a group of entities such as the set of students in a particular year. It can also be many-to-many such as the mutual trust between members of a group or a committee, or many-to-one such as several departments trusting a corporate head branch. In general, the entities involved in a trust relationship will be distributed and may have no direct knowledge of each other so there is a need for mechanisms to support the establishment of trust relationships between distributed entities. If trustees and trustors are considered to be taken from the same set (the set of all entities in a domain), then the trust relation can be thought of as mathematically-defined binary relation. This is a sensible assumption given that the trust relation is defined on sets (which are collection of one or more entities).

2.2.3 Measurability of a Trust Relationship

There is often a *level* of trust associated with a relationship [65]. The trust level is a measure of belief in another entity and thus by our definition, it is a measure of *belief* in the honesty,

competence, security and dependability of this entity (not a measure of the *actual* competence, honesty, security or dependability of a trustee).

Some entities may be trusted more than others with respect to performing an action. It is not clear whether this level should be discrete or continuous. If discrete values are used, then a qualitative label such as high, medium or low may be sufficient. Some systems support arithmetic operations on recommendations so numeric quantification is more appropriate. It is also possible to provide a mapping from qualitative to numeric labels. However, there is still a problem relating to representation of ignorance (or the unknown) with respect to trust.

2.2.4 Applying Mathematical Properties To Trust Relationships

As a trust relationship is a mathematically-defined binary relation, it is a useful exercise to determine if the trust relation satisfies any of the common properties of binary relations, which are reflexivity, irreflexivity, nonreflexivity, symmetry, nonsymmetry, anti-symmetry, asymmetry, transitivity, nontransitivity and intransitivity. This exercise helps to determine which properties can be used in the analysis of trust relationships. Note that for this discussion, it will be assumed that the trust relation is defined for a particular context and with a particular trust level attached. In the rest of this section, the trust relation is referred to as T and xTy represents the fact that x trusts y for a particular context and with a particular trust level. The set of all entities is defined as W . Analysis of the properties merges reasoning about the explicit semantics of their definition and reasoning about their applicability to real-world scenarios.

Reflexivity

T is defined as reflexive if $\forall a : W \bullet aTa$. This means that T is reflexive if for all entities in the domain of interest, every entity trusts itself. An entity may not always trust itself, as it may not have the necessary competence to perform an action.

Irreflexivity

T is said to be irreflexive if $\neg\exists a : W \bullet aTa$. This states that there is no entity in our domain such that the entity trusts itself. PriceWaterHorse may trust itself to carry out in-house software production. This counterexample shows that trust is, in general, not an irreflexive relationship. Thus, though irreflexivity may be true for an overly paranoid domain (i.e. a domain that distrusts everything by default), it cannot be claimed to be true for all domains.

Symmetry

T is defined as symmetric if $\forall a, b : W \bullet aTb \Rightarrow bTa$. This states that for all entities in W , if the trustor trusts the trustee, then the trustee will also trust the trustor. If T were symmetric, then this would imply that trust relationships are always mutual. Jean's trust in the Norton Anti-virus Suite to provide her with a virus list update service does not (and should not) automatically imply that Norton trusts Jean to provide a virus list update service. Thus, trust relationships are not symmetric; however there may be scenarios where symmetric (mutual) relationships need to be specified.

Asymmetry

T is said to be asymmetric if $\forall a, b : W \bullet aTb \Rightarrow \neg bTa$. This assertion states that the trust relation is asymmetric if a trusts b then b does not trust a . The truth of this statement is dependent on organizational-specific factors. Angie and Brenda may both trust each other to test the implementation for a collaborative project. In this case, Angie's trust in Brenda does not automatically imply that Brenda does not trust Angie. In fact, the converse is true for the above example. Thus, T cannot be assumed to be asymmetric.

Anti-symmetry

T is defined as anti-symmetric if $\forall a, b : W \bullet aTb \wedge bTa \Rightarrow a = b$. This states that if a trusts b and b trusts a , then a and b are the same entity. Using the example stated in the explanation of T not being asymmetric, it is clear that the fact that Angie and Brenda have a mutual trust relationship does not imply that Angie and Brenda are the same person. This property is not well suited to real-life trust relationships. Therefore, T is not anti-symmetric.

Transitivity

T is defined as transitive if $\forall a, b, c : W \bullet aTb \wedge bTc \Rightarrow aTc$. This means that T is transitive if for all entities a , b and c , if a trusts b and b trusts c then a trusts c . For an entire system, this is not normally true. Thus, T is not transitive. However, it may be true that $\exists a, b, c : W \bullet aTb \wedge bTc \wedge aTc$ (or even $\exists a, b, c : W \bullet aTb \wedge bTc \Rightarrow aTc$). This property may be useful to computer scientists, whether it be as a form of explanation or as a means to model an interesting concept.

Christianson and Harbison [66] argue that the concept of transitivity should be avoided at all costs. They state that a situation may arise where entity b simply adds trust assertions and in so

doing creates assertions for entity a , without a 's explicit consent. Such a scenario highlights the problem of *unintentional transitivity*. This thesis agrees that transitivity can be dangerous and may lead to a scenario where attackers simply state that they trust external entities and in so doing create assertions for a firm. This has potentially disastrous security ramifications for a firm. However, the concept of transitivity can be useful in modelling notions such as delegation.

Intransitivity

T is said to be intransitive if $\forall a, b, c : W \bullet aTb \wedge bTc \Rightarrow \neg aTc$. This states that given any three entities a , b and c , if a trusts b and b trusts c then a does not trust c . Again this is not true for all trust relationships as contradictions can be found, however, there may be a case where $\exists a, b, c : W \bullet aTb \wedge bTc \wedge \neg aTc$ or $\exists a, b, c : W \bullet aTb \wedge bTc \Rightarrow \neg aTc$.

Summary of Results

The trust relation cannot be said to possess any of the common mathematical properties that binary relations exhibit. This is supported by Gerck's investigation into real-world models of trust [62]. It should also be noted that because this relation is not reflexive, symmetric or transitive, this does not mean that it is nonreflexive, nonsymmetric or nontransitive. By reviewing the definitions of these properties and changing the world of interest, it can be shown that T is also not nonreflexive, not nonsymmetric and not nontransitive. Thus, none of the properties can be applied to trust and thus, none of them can be used as axioms for the trust relationship. This is due to the nature of the definition of these mathematical properties, the nature and diversity of distributed systems and or the nature of the trust relation. There are often cases where the conditions of a particular property may apply to a particular network. This highlights the need for a TMF to be flexible enough to allow properties to be randomly included and or excluded.

2.2.5 Relationship Indicators

Trusting behaviour is determined based on a myriad of indicators. Consumers may be willing to (repeatedly) trust a business, despite the fact that their goals and the goals of the producers are diametrically different. For example, I know that DKNY is focused on profit maximization and I know that my personal philosophy is to buy the cheapest goods. However, I still decide to trust DKNY, repeatedly, for extraneous factors, such as their excellent return policy or their AMEX-funded reward scheme. Thus, a trust relationship is influenced by market forces, social

and interaction cues, legislative mechanisms, assurance systems, insurance and the lack of choice or effective alternatives. Sometimes my trust propensity may dictate that I stick to companies, products and services that I know about, irrespective of all the other factors. Some of these social devices are hard (and possibly unrealistic) to represent in a technological solution to the trust management problem. However, the factors that can be easily incorporated are experience and risk.

Experience with an entity determines the level of trust/distrust that will exist in a relationship. The better the experience that you have had with a business, the more likely you are to trust them. This implies that 1) experience influences our trust decision 2) there is a direct relationship between experience and trust. In traditional commerce, the more experience that can be gained, over a longer period of time, will be a dominant factor in determining trusting behaviour. Suppose that I have been a customer of PhonyMicroSof for ten years and have had a majority of bad experiences with them (failures to deliver goods, defective products, etc.). When faced with an option to trust them or trust another more reliable (or even new) company, experience would lead to not doing business with PhonyMicroSof. However, there may be exceptions to this rule. For example, if there is a monopoly on a product or service that is routinely used, then the product may normally be accepted, in spite of experience information.

Another factor that influences a trust decision is the risk involved in the venture. Risk is an integral part of our everyday lives. A risk is taken when you cross the street, when you buy a product from a catalogue or when you give your credit card to a waiter. Risk is normally expressed in terms of a monetary loss. For transactions, the risk is dependent on a number of factors, such as the cost of the transaction and the capital loss or gain that could be incurred. A trust relationship is normally established between trustor and trustee when both can ascertain that the risk involved in performing the transaction is low. Thus, it becomes apparent that the basic relationship (or rule) between trust and risk is that the lower the risk, the more trust one is willing to place. However, there are aberrations from this rule. Danraj may trust a high-risk venture of low value, but not a medium risk transaction of high value. Gamblers may trust highly valued, highly risky transactions.

2.3 Trust Classification

Different forms of trust have been identified in the literature relating to whether access is being provided to the trustor's resources, the trustee is providing a service, trust concerns

authentication or it is being delegated. This is not meant to be an exhaustive taxonomy, but merely a useful way of classifying the literature relating to trust in Internet services.

2.3.1 Access to a Trustor's Resources

A trustor trusts a trustee to use resources that he owns or controls, which could be a software execution environment or an application service [67-69]. Abrams and Joyce [67] highlight the fact that resource access trust has been the focus for security specialists for many decades, although the emphasis has mostly been on mechanisms supporting access control.

There is an obvious distinction between trusting an entity to read or write a file on your server and trusting an entity to execute code within your workstation. Simple file access requires that the trustee will follow the correct protocol, will not divulge information read, and will write only correct data etc. Allowing an entity to execute code on your workstation implies a much higher initial level of trust. The code is expected not to damage the trustor's resources, to terminate within a reasonable finite time and not to exceed some defined resource limits with respect to memory, processor time, local file space etc.

In [67, 68], the authors implicitly map trust decisions to access control decisions. Generally, resource access trust can form the basis for specifying authorisation policy, which then is implemented using operating system or database access control mechanisms, firewall rules etc. The trust relationship can be refined into authorisation policies that specify actions the trustee can perform on the trustor's resources and constraints that apply, such as time periods for when the access is permitted.

Examples of Resource Access Trust:

- Fred is trusted to do Linux installations and Joe is trusted to do NT installations on our section workstations.
- Third year and above students are trusted to use the parallel processing service.
- I trust XY Cleaners to send someone to clean my house even when I am not there.
- I distrust AB Garage so I will not take my car to be repaired there.

These rather abstract specifications of trust and distrust would need to be refined into specific authorisations policies that define permitted operations to specific resources.

2.3.2 Provision of Service by the Trustee

The trustor trusts the trustee to provide a service that does not involve access to the trustor's resources. Note this may not be true of many services such as web services that download applets and cookies, and so do require access to resources owned by the trustor.

Service bureaux and application service providers (ASPs) [70-72] are prime examples of entities that would require service provision trust to be established. Currently, in these domains, trust is often an unstated implication of establishing a relationship, which is difficult to enforce or monitor. Mobile code and mobile agent based applications obviously must trust the execution environment provided by the remote system (provision of service trust) but the execution environment should not be damaged by the mobile code (access to resources trust).

Examples of Service Provision Trust :

- I trust a film recommendation service to only recommend films that are not pornographic.
- I trust website xyz to provide information that is non-offensive.
- I distrust sexy-Susan website.

The above examples are a form of *confidence trust* in that the trustor has confidence in (or specifically distrusts) the standard of service provided by the service provider. This type of trust maps into a form of access control, which is subject-based, in that the subject is only permitted to access trusted services. This type of access control can be implemented by some web browser as a means of screening sites visited by children [7, 48, 49, 55, 59, 73].

Some forms of service trust relate to *competence* of the trustee:

- I only trust fourth year students who have an aggregate A grade to do this project.
- I will only purchase PCs from Company ABC.

A trustor's trust in the competence of the trustee's ability to provide a service differs from confidence trust in that, confidence applies to entities the trustor will use and competence applies to entities that perform some action on behalf of the trustor.

Another form of service trust relates to *reliability* or *integrity* of the trustee. In E-Commerce and E-Banking, the customer trusts the vendor or bank to support mechanisms that will ensure that passwords are not divulged and to prevent transactions from being monitored. The vendor or bank is also trusted to maintain the *privacy* of any information such as name, address and

credit card details, which it holds about the customer. There have been some high-profile incidents in the UK recently where this trust has been broken. Examples of this form of service trust are:

- I will store these critical files on Groucho (as it has a RAID file system and it is archived every 2 hours). Note that in this case the trustee does have access to the trustor's resources.
- I trust the newsagent to email me an electronic newspaper every morning before 8am.
- I trust my Internet bank not to divulge my name and address to companies for electronic marketing.

2.3.3 Certification of Trustees

This type of trust is based on certification of the trustworthiness of the trustee by a third party, so trust would be based on a criteria relating to the set of certificates presented by the trustee to the trustor. Certificates are commonly used to authenticate identity or membership of a group in Internet applications [38, 74-79]. This may imply competence if the identity is a well-known organisation. However, professional certification is a common technique used to indicate competence in the medical world, commerce and engineering so could be applied to Internet services [80].

Trustee Certification Examples:

- I trust Dr. Tom's medical advice site as he is registered with the BMA.
- I will only use downloaded software updates, which have Microsoft certificates.
- I trust only VeriSign to certify programs that can run on my machine.
- I trust anyone with a PGP certificates signed by two people I trust (each must have an average level of trust in my view).
- I trust the identity of anyone authenticated by the Kerberos server in my domain.

Note that the certification authority is in fact providing a trust certification *service* so this is a special form of service provision trust but involves a third party in establishing the trust. There are many papers discussing this specific form of trust service, which is the reason a separate classification is included.

2.3.4 Delegation

A trustor trusts a trustee to make decisions on its behalf, with respect to a resource or service that the trustor owns or controls [81]. This is also a special form of service provision – a trust decision-making service.

Ding and Peterson [81] illustrate a novel way of implementing delegation, with hierarchical delegation tokens. Their work relies heavily on cryptography. They propose a classification of delegation schemes, with appropriate protocols, which they analyse, based on efficiency, and compare with related work. The ideas they express represent lower-level mappings from the concept of delegation purported in this thesis, in that they concentrate on access control.

Delegation Trust Examples:

- I trust my database manager to decide who has access to my database.
- I delegate all decisions concerning my investments to my financial advisor.
- I accept anonymous authorisation certificates for access to my resources issued by the WXYZ authorisation service.

2.3.5 Infrastructure Trust

This refers to the base infrastructure that the trustor must trust [67-69]. He must trust himself (implicit trust). He should be able to trust his workstation, local network and local servers, which may implement security or other services in order to protect his infrastructure.

It was recognised in early computing that in order to incorporate security (actually resource access trust) into applications, there was a need not only to implicitly trust the reference monitor, but also the administrative procedures that kept the monitor working. The culmination of this work was the U.S. Department of Defense specification for a set of resources, known as the Trusted Computing Base (TCB) [82] that had to be trusted by all applications executing on a machine to support the required security policy. The TCB can be viewed as the set of hardware, firmware and software elements, which are used to implement the reference validation mechanism i.e. the “*validation of each reference to data or programs by any user (or program) against a list of authorized types of reference for that user*” [82]. The TCB was seen as the primary component of a trusted computer containing all of the system elements supporting the isolation of objects (code and data) on which the protection is based. It was aimed more at

centralised systems implementing information labelling and preventing information flow to unauthorised users, rather than commercial or networked systems.

Over the years, the TCB has increased in terms of number of components, and therefore size, leading to a higher probability of it being compromised. To make the PC platform more trustworthy, an initiative was launched to develop and formalize a trusted PC framework [51]. This initiative is being led by the Trusted Computing Platform Alliance, an amalgamation of several leading technology companies and research centres.

Infrastructure Trust Examples:

- I trust hardware that has been certified by the Trusted PC Computer Base Certification Board.
- The PC's application software trusts the operating system.

2.3.6 Dominant Attributes for Trust Contexts

The importance of competence, honesty, security and dependability depends on the context in which they are used. However, it is impossible to state that there is a particular combination of attributes that uniquely define a particular context. This is due to the fact that the example scenarios for each of these contexts are so varied that a different importance hierarchy may be found for examples within the same context. In Figure 2.1 it can be seen that the competence of the Doctor is the most important concern for the patient

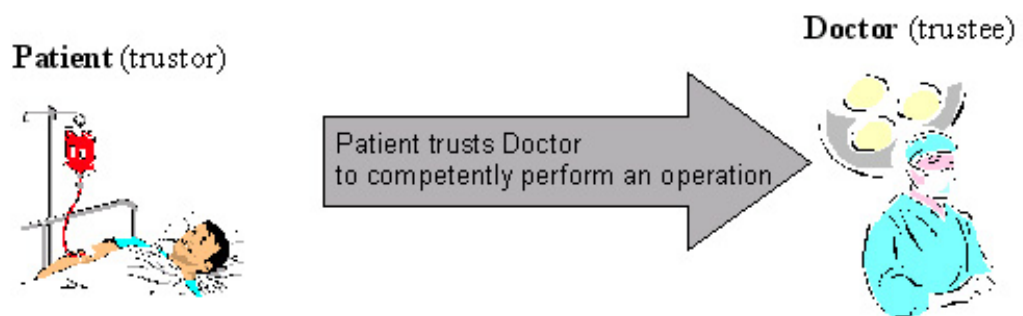


Figure 2.1: Service Provision Trust with Competence

This is an example of service provision trust: Doctor is providing a service to Patient. Figure 2.2 is also an example of service provision trust, but here the primary concern of Chris is with the honesty of CarSalesman.

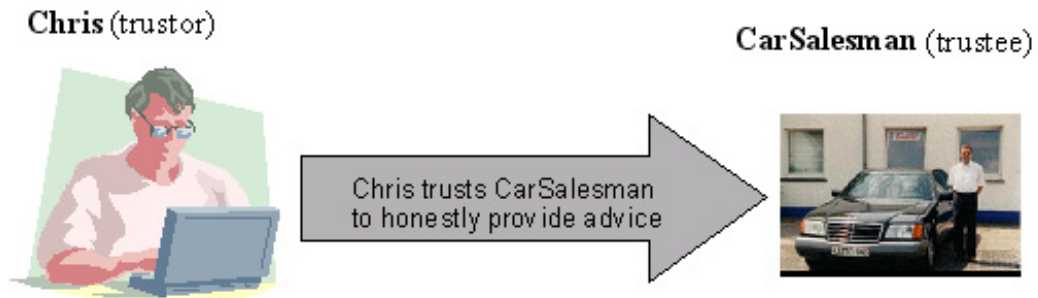


Figure 2.2: Service Provision Trust with Honesty Factor Dominant

These two examples highlight the point that it is impossible to find a particular mix of attributes that can be used to characterize a particular context. To further emphasize this fact, let's look at two examples of Access to Trustor Resources trust.

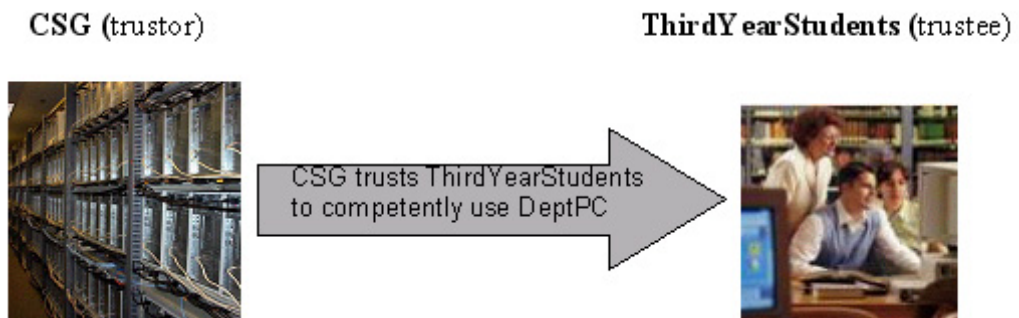


Figure 2.3: Access to Trustor Resources Trust with Competence Factor Dominant

Figure 2.3 shows that CSG trusts ThirdYearStudents to use DeptPC. From CSG's perspective, the important factor is that ThirdYearStudents can use DeptPC competently.

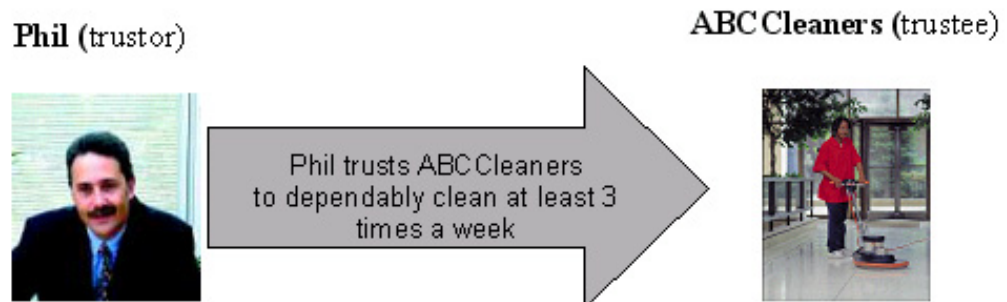


Figure 2.4: Access to Trustor Resources Trust with Dependability and Timeliness Factors Dominant

In Figure 2.4, Phil's primary concern is in the dependability and timeliness of ABCCleaners. Figures 2.3 and 2.4 illustrate further that it is impossible to find a set of attributes that characterize each trust context. All the examples may be viewed as applicable to Internet applications by realizing that the trustees in each example can easily be software agents acting on behalf of real individuals.

Figures 2.1 to 2.4 illustrate that it is not very practical to use trust attributes to characterize trust relationships. From the above discussion, it can be seen that there is very little correlation between trust attributes and specific types of trust relationships or contexts. Thus, we came to the conclusion that explicitly modelling the attributes of competence, honesty, security and dependability in the trust relation and trying to assign values for each attribute would add considerable complexity with little additional benefit. The trust level represents a belief measure for these combined attributes, and implicitly gives most weight to those that are most relevant to the explicitly stated context.

The complexity of estimating levels for individual attributes such as dependability has been discussed in the literature. Helvik [83] discusses the attributes of dependability (namely: reliability, availability and safety) and highlights the fact that developing and evaluating dependability models (even simple ones) is a complex and time-consuming process. Galin [84], Kan [85] and Littewood and Strigini [86] state that the process of applying software quality metrics (which include dependability models) to realistic situations is a difficult task that still requires considerable further research. Thus, formulating, developing and applying a model for individual measures of honesty, competence, security and dependability would be very difficult to do for real-time scenarios involving Internet applications.

In this work, we assume that a human administrator would assign an initial value to a trust level, which combines trust attributes relevant to the context of the trust relationship. Although the framework allows for a range of integer values, we anticipate the administrator to only make use of a few discrete values, representing, for example, low medium and high levels of trust. However, strategies to automatically update the trust levels based on experience can be included in the framework.

2.4 Trust Formalisms

In this section, the attempts by computer scientists to create formal models of trust are discussed. These formalisms are based on formal logic representations of the social process of trust, on computational models that try to convey the core of a trust relationship and on the application of human cues and economic models in engendering trust in software.

2.4.1 Logic-based Formalisms

Trust involves specifying and reasoning about beliefs. Forms of first order predicate logic [46, 64, 87, 88] or (modified) modal logic [89-92] have been used to represent trust and its associated concepts. A logic used to formalise trust must represent actions and interactions to cater for distributed agents [93, 94].

Simple relational formalisms are used to model trust with statements of the form $T_a b$, which means ‘ a trusts b ’ [46, 64, 87, 88]. Each formalism extends this primitive construct with features such as temporal constraints and predicate arguments. Given these primitives, traditional conjunction, implication, negation and disjunction operators, these logical frameworks express trust rules (such as trust is not transitive) in their language and reason about these properties. These simple formalisms are not capable of modelling the trust relationships found in the Internet. This is discussed in more detailed below.

Burrows, Abadi and Needham [46] propose a language to specify the steps followed in the authentication process between two entities (resource access protocol analysis). The language is founded on cryptographic reasoning with logical operators defined to deal with notions of shared keys, public keys, encrypted statements, secrets, nonce freshness and statement jurisdiction (for authentication servers and certificate authorities). It should be possible to answer the following questions about a protocol specified in this language: What does this protocol achieve? Does this protocol need more assumptions than another one? Does this protocol do anything unnecessary that could be left out without weakening it? Does this protocol encrypt something that could be sent in clear without weakening it? The language can be used to specify protocol assumptions and interactions and appears to be suitable for modelling trust establishment protocols. In designing the language many simplifying assumptions were made. As stated in [46]:

“Since we operate at an abstract level, we do not consider errors introduced by concrete implementations of a protocol, such as deadlocks, or even inappropriate use of cryptosystems. Furthermore, while we allow for the possibility of hostile intruders, there is no attempt to deal with the authentication of an untrustworthy principal, nor detect weaknesses of encryption schemes or unauthorised release of secrets. Rather, our study concentrates on the beliefs of trustworthy parties involved in the protocols and on the evolution of these beliefs as a consequence of communication.”

Also, analysis using this language requires a language expert to read through the logical statements, look at the implications and reason about them.

The Authorization Specification Language (ASL) by Jajodia, Samarati and Subrahmanian [88] is used to specify authorization rules and makes explicit the need for the separation of policies and mechanisms. ASL supports the specification of the closed policy model (all allowable accesses must be specified) and the open policy model (all denied accesses must be explicitly specified) using a common architectural framework. It also supports role-based access control. The separation of the concepts of policy and mechanism allows the specification and implementation of more flexible systems, as the access control model need not be hard coded into the system. This language is an excellent tool for the specification and analysis of resource access trust.

Modal logics can be used to express possibility, necessity, belief, knowledge, temporal progression and other modalities [89-92]. It is an extension of traditional logics (propositional and predicate). The \Box (necessity) operator and the \Diamond (possibility) operator are added to the traditional syntax. The notion of possible worlds (or multiple worlds) is fundamental to the interpretation of modal logics and simply states that a statement will have a different meaning depending on the world it is in. Kripke structures are used to represent possible worlds, where a Kripke structure consists of the set of all possible worlds and an accessibility relation (which may be referred to as a possibility relation depending on the modal logic). The accessibility relation states the conditions for which an agent can access a world.

In [91], Jones and Firozabadi address the issue of the reliability of an agent's transmission. They use a modal logic of action developed by Kanger, Porn and Lindahl to model agent's actions. For example, $E_i p$ means 'agent i brings it about that p '. They use a variant of a normal modal logic of type KD45 as the foundation for their belief system. For example, $B_i A$ means 'agent i believes that A '. The topic of institutional power is incorporated through the use of a *counts as* operator. Institutional power refers to the fact that a person performing an act in a particular institution will lead to the formation of an institutional fact. In a different institution, this fact cannot be established. For example, a minister performing a marriage ceremony at a church leads to the fact that two people are married; yet in a different church this fact will not exist. They adopt the relevant axiomatic schemas into their formalism and use their composite language to model various trust scenarios. For example, b 's belief that a sees to it that m is

expressed as: $B_b E_a m$. They also use their language to model the concepts of deception and an entity's trust in another entity. In their own words:

“We do not investigate the formal representations of procedures by means of which trust-relations between agents can be established. Assuming the existence of a trust-relation, we try to make explicit the reasoning patterns characteristic of a trusting agent.”

This formalism can be easily modified to express and reason about establishing trust, within any particular context.

Rangan [92] views a distributed system as a collection of agents communicating with each other through message passing in which the state of an agent is its message history (all sent and received messages). The state of the system is the state of all the agents in the system. He then devises conditions that function as his accessibility relation (in this context, possibility relation is a more accurate term). His model consists of simple trust statements (for example $B_i p$, which means ‘agent i believes proposition p ’) and properties such as transitivity, Euclidean property, etc. are defined. These constructs are then used to specify systems and analyse them with respect to the property (or properties) of interest. Rangan's model more fluently follows the traditional lines of modal logics of beliefs than does Jones and Firozabadi's, but the model is simpler in the sense of the non-treatment of actions and their effects.

Subjective Logic [19, 23, 95-100] is a logic that operates on subjective beliefs about well-defined, uncertain propositions. The basic component of this logic is an opinion, which is the 4-tuple consisting of b – belief in the proposition, d – the disbelief in the proposition, u – the uncertainty in the proposition and a – the relative atomicity of the proposition. The four elements are related by the facts that $b + d + u = 1$ and the probability expectation value is $b + a * u$. The relative atomicity expresses the relative weighting of the proposition with respect to the other propositions defined. The probability expectation value represents the value obtained when an opinion is translated to a standard probability value. The logic builds on both a Shaferian belief model and probability calculus. In order to reason about uncertainty, six operators are defined: conjunction, disjunction, negation, consensus, discounting and conditional inference. In situations characterized by high uncertainty and quick response times (‘hard real time systems’), subjective logic produces better results than the classical Dempster-Shafer model [100, 101]. This model's strength lies in the ability to reason about the opinions on a mathematically sound basis by using its operators, which are based on probability density

functions and standard logic. However, its major weakness is that it cannot be guaranteed that users will accurately assign values appropriately. This problem is experienced by most formalisms, even the computational ones, which will now be presented.

2.4.2 Computational Models

These formalisms take a natural science approach to the trust problem, i.e. a phenomenon can always be examined by various methods. Computational models provide a mathematical framework with which trust can be examined. There are two categories of these models: numerical models and fuzzy logic based models. Both types function in the same manner. Both model a typical scenario with a typical context and try to explain the trust each agent should have in each other. Thus, they try to evaluate whether trust should be established and, if it should be, then what is the appropriate initial trust value.

Numerical models view the trust value as an arbitrary real number that can be used in further computation, while fuzzy logic models view the trust value as a linguistic label that represents a (range of) value(s). Fuzzy logic models also define a mathematical framework that allows the manipulation of these labels. Let's look at two numerical models and a fuzzy-logic based models.

Marsh [64] introduced his computational model of trust to the distributed artificial intelligence community in 1994. Trust is represented in his model by a subjective real number between -1 and +1. He assumes that trust is separated into three aspects: basic, general and situational trust. He devises a notation for basic and general trust and outlines a formula for determining situational trust taking into consideration agent dispositions, cooperation, risk, competence, memory and reciprocation. The problems with his model are:

- the model exhibits anomalous behaviour when the trust value is -1, 1 and 0,
- the representations of agent dispositions etc are simplified versions of reality,
- the model has problems dealing with negative trust and its propagation, and
- the operators for manipulating trust values are limited.

In 'Computational Model of Trust and Reputation for E-businesses', Mui et al. [35] describe their numerical model of trust. Their model is based on the concepts of trust, reputation and reciprocity. Reputation (c.f. experience) is defined as the perception that an agent creates through past actions about its intentions and norms. Reciprocity is defined as the mutual exchange of deeds (such as revenge or favour). Their model is based on a cyclic, re-enforcing

relationship between the three concepts. A notation is defined that allows for the simple actions of cooperate or defect and the assignment of real values for reciprocity and reputation. A cooperate action signifies that both parties will engage in a relationship, while a defect action signifies otherwise. A mathematical model for determining trust given simple representations for an encounter and history and for propagating reputation information is formulated. The problems with this model are that:

- it is hard to accurately initialise values for reciprocity and reputation,
- it assumes that trust is inferred only from reputation, and
- its reputation propagation mechanism only works for simple networks.

In [25], Manchala proposes a model for the measurement of trust variables and the fuzzy verification of E-Commerce transactions. He highlights the fact that trust can be determined by evaluating the factors that influence it, namely risk. He defines cost of transaction, transaction history, customer loyalty, indemnity and spending patterns as trust variables. Each variable is measured using semantic labels. His notation is focused on defining when two trust variables are related by an Electronic Commerce Trust Relationship (ECTR). Using this ECTR, a trust matrix is constructed between the two variables and a Trust Zone is established. He also describes a method for trust propagation and the construction of a single trust matrix between vendor and customer that governs the transaction. The problem with Manchala's model is that 1) it is unclear which variables should be used by default for the best results, 2) it is unclear if it is actually possible for a computer to automatically establish that two variables are related by an ECTR. In his definition, he mentions a semantic relationship between the variables, but neglects to mention how this fact will be specified to the computer so that evaluation can be automated and 3) it is unclear if ECTR merging will scale in the face of large trust matrices. These concerns are all related to the viability of implementing his model.

Let us now discuss models that focus on the HCI community's efforts to model trust.

2.4.3 Human Computer Interaction (HCI) Based Models

People trust other individuals by using a set of social clues, such as a gaze, facial expression, and subtle details of their conversational style. The absence of similar cues online led the HCI community to investigate how such trust could be engendered using technology. HCI-based models [102-106] are concerned with how trust is developed between two interacting individuals using computer-mediated communication (CMC) technology (chat, audio, email,

video). These formalisms seek to answer the questions: To what extent do users evaluate and establish trust with each other via CMC technology? What are the conditions necessary to establish trust for a collaborative project using CMC technology? When trust is established, what is the initial level of trust? Which CMC technology is best for producing trust levels comparable to levels produced by standard face-to-face interactions? And how can the trustworthiness of CMC technology be increased? There are two dominant types of HCI-based models of trust, namely: social models or economics-based models.

The social models [107-110] incorporate the use of CMC technology and human interaction cues to either assess the trustworthiness of entities [107, 108, 110] or to help in the design of trustworthy interfaces [109]. These models take into account the notions of the user's psychology (propensity to trust, trust in IT and the Internet, utility, usefulness, risk), reputation information (reputation of the industry, of the company and information from trusted third parties), interface properties (branding, usability), information quality and pre-transaction and post-transaction relationship management [111]. The major problem with these models is that they are essentially design methodologies with little or no tool support and with too many complex social notions. For interacting Internet applications, it would be hard to incorporate these types of models.

Economics-based models assume that individuals make trust choices based on rationally derived costs and benefits [112]. They are based on game theory and firmly rooted in experiments and surveys to identify the action that should be taken by individuals embarking upon an interaction. Most economics-based models [29, 113, 114] are founded on the Prisoner's Dilemma (PD) game [115, 116]. The game is defined as follows: There are two men, A and B who are charged with a joint law violation, being held separately by the police and given three options. The first option is that someone confesses, say A, and B does not, then A will be rewarded and B fined. The second option is that if they both confess, each will be fined. The final option is that if neither confesses, both will go clear. So, should each man cooperate (keep quiet) or defect (confess)? Ideally, if all the parties are rational, then the best independent choice to make is defection. This may not lead to the optimal utility for each player, but a player will not find himself at a relative disadvantage. Researchers of economic-based HCI trust models use PD games to design experiments to evaluate the actions of participants engaging in cooperative task using various CMC technologies. The trust level is determined by the rate of cooperation, which is measured by the collective payoff. The problems with these models are that PD games

are 1) synchronous and thus cannot be applied to all trust scenarios and 2) too simple to model an E-Commerce application that includes many other factors such as experience, etc.

There are models that combine both the social and economics perspectives [117, 118]. These models tend to be based on more rigid game theory, which allows for better modelling in a wider range of situations. They also include specific social concepts that they consider pertinent to the trust problem and thus avoid the complexity of trying to model complex social ideas.

HCI-based models address the problem of trying to model the social trust building process and apply it to the user's trust in an interface, which is a representation of another person or firm. This approach bridges the gap between human and computer and offers an important starting point for developing trustworthy system. However, it does not discuss the issues involved when multiple software agents, acting on behalf of real persons, must interact in a computer network environment. HCI-based models assume that the use of human-like interfaces (anthropomorphism) will allow interfaces to be viewed as more trustworthy. However, anthropomorphism can diminish trust when interfaces do not meet expectations [119]. HCI-based models address part of the trust problem. They tackle the issue of how anthropomorphism can be used to engender trust in technology. Stated more generally, HCI trust models try to show how to engender trust in technology by using technology.

2.5 Views of Trust Management

The paper by Blaze et al. [39] was one of the first to introduce the term *trust management*, although prior security solutions for networked applications had an implicit notion of trust management based on PGP [75] or X.509 public-key certificates [38, 74]. Blaze et al. defined trust management as:

“a unified approach to specifying and interpreting security policies, credentials, relationships which allow direct authorisation of security-critical actions.” [39]

Although, this was the dominant view of trust management until the turn of the century, it is flawed. The reason for this flaw is embedded in the dilemma faced by the security community at the time. The dilemma is defined in the following context. The Internet was not designed with security in mind. Cryptography was used as a solution to the network's security problems, but 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 a trust problem, which is the trust management problem that Blaze refers to. Thus, [39] does not focus on the problem of managing trust, but on the problem of managing public key authorisation. Blaze's problem can be succinctly stated as that of allowing public keys to be authorised and linked directly to their allowable actions.

In [120], Josang and Tran define trust management as:

“the activity of collecting, codifying, analysing and presenting security relevant evidence with the purpose of making assessments and decisions regarding E-Commerce transactions.”

This definition offers a broader, more abstract and more intuitive interpretation of the trust management problem than presented in [39]. Josang and Tran stress the need for trust assessment to be based on evidence that can be practically collected and then used for decision-making. This emphasis is made because systematic and reliable ways of obtaining evidence in the e-commerce environment are non-existent. However, [120] neglects to expound on the issues surrounding the encoding, analysis and presentation of this evidence.

In this thesis, trust management is defined as:

“the activity of collecting, encoding, 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.” [4, 5]

Evidence could include credentials 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. A more comprehensive discussion on trust management, as it pertains to this thesis, is given in Chapter 7, but now let's discuss some of the current trust management solutions.

2.6 Contemporary Trust Management Solutions

Most trust management systems focus on protocols for establishing trust in a particular context. Some make use of a trust policy language to allow the trustor to specify the criteria for a trustee to be considered trustworthy. In this section, a few of these solutions, namely: Public Key Certificates, PICS, PolicyMaker, KeyNote, REFEREE, SD3, Fidelis, IBM TEF, TrustBuilder, TCPA, Poblano and a few other emerging trust management solutions, will be discussed.

2.6.1 Public Key Certificates

A digital certificate is issued by a certification authority and verifies that a public key is owned by a particular entity [75]. The certification authority does not vouch for the trustworthiness of the key owner, but simply authenticates the owner's identity. This is necessary to establish a resource access or service provision trust relationship and may implicitly reduce the trustor's risk in dealing with the trustee [80]. However, the policy governing what resources or services the trustee is permitted to access is not handled by the certificate infrastructure, but is left up to the application. Two of the main certificate systems dealing with authentication, PGP and X.509, are described below.

The PGP trust model [75] is used for authentication relating to electronic mail type of applications between human users. It supports a *Web Of Trust* model in that there is no centralised or hierarchical relationship between certification authorities as with X.509. The underlying assumptions of the model are that a trustor may trust other entities, may validate certificates from other entities or may trust third parties to validate certificates. An *introducer* is an entity that signs someone else's public key (and thus vouches for a name-public key binding). A *meta-introducer* can sign keys as well as specify who is a (trusted) introducer. Thus, any entity can function as a certification authority. Every key that a user trusts or signs has to have a degree of trust associated with it, namely: unknown, untrusted, marginally trusted or completely trusted. It is also assumed that a user has an implicit trust (the highest form of trust in this model) in her own key. It is possible to use these labels to specify complex criteria about the trustworthiness of keys. For example, a user can specify that she only completely trusts a key if it is marginally trusted by a meta-introducer and completely trusted by a (trusted) introducer. Once keys are registered (along with their degree of trust) with the PGP system, then it computes a validity score (this measures how sure we are that this key belongs to this

person). It is now the responsibility of each entity to query the system and to acquire keys as they are needed.

The X.509 trust model [38, 74] is a strictly hierarchical trust model for authentication. Each entity must have a certificate that is signed by the central certification authority or another authority, which has been directly or indirectly certified by it. This model assumes that certification authorities are organised into a universal ‘certification authority tree’ and that all certificates within a local community will be signed by a certification authority that can be linked into this tree [39].

It is important to note that neither of these models can be used to model trust in all domains. Due to PGP’s lack of official mechanisms for the creation, acquisition and distribution of certificates it is considered unreliable for E-Commerce, but appropriate for personal communication. X.509’s rigid hierarchical structure may lead to unnatural business alliances between competing companies that violate the natural order of trust. Some applications, such as the reference information distribution systems need certificates to have a lifespan longer than is currently allowed by either scheme. Additionally, current PKI implementations contain no systematic and reliable method of obtaining evidence about entities involved in an Internet transaction [120].

2.6.2 PICS

PICS (Platform for Internet Content Selection) was developed by the World Wide Web Consortium (W3C) as a solution to the problem of protecting children from pornography on the Internet. The basic idea behind PICS is that there needs to be a filter between the potential viewer and web documents. It is a result of the ‘Censorship of the Internet’ debate that took place in the US legislature. PICS defines standards for the format and distribution of labels, which are meta-documents describing web documents. It does not specify a vocabulary that a label must use, nor does it state which labels are important for a particular circumstance. It is similar to stating “*where on a package a label should appear, and in what font it should be printed, without specifying what it should say*” [7] or what part of the label is important. A PICS-compliant application should be able to read PICS labels and use the user-defined filtering rules to decide whether to accept or reject the document. PICS makes no assumptions about the number of labels that can be attached to a document. In concept, a document may have several labels that may be issued by different organisations. A user has the right to choose any PICS filtering software and any label source (this entity is called a rating service).

A rating system defines the label attributes and their corresponding range of values used by the rating service. The following example is adapted from the W3C Recommendation on rating services and rating systems [59].

```
( (PICS-version1.1)
  (rating-system "http://www.doc.worldwide.com/ratings/")
  (rating-service "http://www.doc.worldwide.com/descrip.html")
  (icon "icons/good.gif")
  (name "The Computing Department Rating System")
  (description "All about the rating of the pages offered by computing departments all over the world")
  ( category
    (transmit-as tc)
    (name "Teaching Material Content")
    (min 0.0)
    (max 5.0)
  )
  ( category
    (transmit-as rc)
    (name "Research Content")
    (label (name "very little") (value 0) (icon "icons/little.gif") )
    (label (name "a lot") (value 1) (icon "icons/lots.gif") )
  )
  ( category
    (transmit-as subject)
    (name "Document Subject")
    (multivalue true)
    (unordered true)
    (label (name "SE") (value 0) )
    (label (name "AI") (value 1) )
    (label (name "PC") (value 2) )
    (label-only)
  )
  ( category
    (transmit-as ref)
    (name "Number of references to other computing sites")
    (integer)
  )
  ( category
    (transmit-as importance)
    (min 0)
    (max 100)
  )
)
```

The first section identifies the version of PICS being used, the rating system and the rating service. The URL in the rating-system clause specifies the location of the document that has the human-readable description of the rating system. The URL in the rating-service clause identifies the document with the human-readable description of the rating service. This URL will also be included in all labels created by this service. The icon, name and description clauses are self-explanatory. The rest of the example specifies five label attributes; each identified by the keyword *category*. Each category has a transmission name and may be followed by clauses that define the attribute's allowable values. The BNF for the syntax of a

rating system can be found in [59] and a sample PICS label that uses the above rating system is shown below.

```
( (PICS-version1.1)
  "http://www.doc.worldwide.com/descrip.html"
  labels on "1998.11.05T08:15-0500"
  until "1999.09.32T23:34-0000"
  for "http://www-dse.doc.ic.ac.uk/~per/index.html"
  by "Joe Green"
  ratings (tc 1.0 rc "a lot" subject "SE" ref 19 importance 90)
)
```

The above example identifies the rating service that created the label, sets the lifespan of the label, identifies the page being labelled, the person labelling the page and the actual values of the label's attributes. More complex labels can be constructed using the PICS label syntax described in [59]. There are three ways that labels can be distributed, namely: they can be embedded in web documents (through the use of a META tag); they can be requested by a user (the HTTP GET is used to request both the label and the web document) and they may be requested separately from label bureaux.

The W3C has also published the PICSRules recommendation, which describes a rule-based, filtering policy language. Some policies expressed in PICSRules (version 1.1) are now given, adapted from [55].

```
(PicsRule-1.1
(
  Policy (RejectByURL ( "http://*@www.doc.ic.ac.uk*/*"
                       "http://*@www.yahoo.com*/*" )
  )
  Policy (AcceptIf "otherwise")
)
)
```

The above example states that access to any Yahoo web page or any site at the department of computing at Imperial College is forbidden, but access to any other page is permitted. It does not use PICS labels.

```
(PicsRule-1.1
(
  ServiceInfo (
    name "http://www.raters.org/ratings/v1.html"
    shortname "serv"
    bureauURL "http://labelbureau.raters.org/Ratings"
  )
  Policy (RejectUnless "(serv.pics)")
  Policy (AcceptIf "( (serv.pics > 3) and (serv.nudity = 0) )" )
  Policy (RejectIf "otherwise")
)
)
```

The above example sets the rating service in the ServiceInfo and uses the labels from the service to select pages. It states that all pages with labels must have the *pics* attribute in order for them to be viewed. Additionally, the pages must have a *pics* attribute with value of three or more and also the *nudity* attribute should be equal to zero in order to be allowed. All other pages will be rejected.

```
(PicsRule-1.1
  (
    name (
      rulename "More Complex"
      description "Highlight more features of PICSRules"
    )
    source (
      sourceURL
      "http://www.complex.com/complex.html" )
    ServiceInfo (
      name "http://www.doc.ic.ac.uk/ratings/v1.html"
      shortname "DOC"
    )
    ServiceInfo (
      name "http://www.raters.org/ratings/v1.html"
      shortname "serv"
      bureauURL "http://labelbureau.raters.org/Ratings"
    )
    Policy (RejectByURL ("http://*@www.badnews.com:*/*"
      "http:// *@www.baddernews.com:*/**")
    )
    Policy (AcceptByURL "http://*good-entertainment.org/plays*")
    Policy (AcceptIf "(DOC.educational = 1)"
      Explanation "Always allow educational content"
    )
    Policy (RejectIf "(DOC.violence >= 4)"
      Explanation "This is too scary"
    )
    Policy (RejectUnless "(serv.graphics < 4)")
    Policy (AcceptIf "otherwise")
  )
)
```

In the above example, the name clause defines a human readable name for the rule and a description. The source clause specifies where the rule came from. The source URL may contain a document that has information about the rule. The first ServiceInfo clause specifies a rating service that the user wishes to use, giving it an alias. The absence of the bureauURL tag means that only embedded labels will be used. The other clauses reject pages from two sites, accept good plays, allow educational documents, reject documents with too much violence (unless they are educational), block any page with too many graphics (with the exceptional of educational documents) and allow all other pages. The BNF of the PICSRules syntax is outlined in [55]. PICSRules is a powerful tag-based language that allows resource-access trust

on the Internet. The effectiveness of the PICS framework lies in the expressiveness of the filtering languages and the quality of the rating services.

2.6.3 PolicyMaker and KeyNote

PolicyMaker [40, 42, 43, 56] is a trust management application, developed at AT&T Research Laboratories that specifies what a public key is authorised to do [39]. Traditional certificate frameworks such as PGP and X.509 do not bind access rights to the owner of the public key within the certificate framework. Schemes such as these require a two-step process: a) the binding of a public key to its owner, which occurs within the certificate framework, and b) the binding of access rights to the identified key owner, which occurs outside the certificate framework. In PolicyMaker, both occur in a single step that binds access rights to a public key. The PolicyMaker system is essentially a query engine which can either be built into applications (through a linked library) or run as a ‘daemon’ service. It evaluates whether a proposed action is consistent with local policy [7]. The inputs to the PolicyMaker interpreter are the local policy, the received credentials and an action string (which specifies the actions that the public key wants to perform). The interpreter’s response to the application can either be yes or no or a list of restrictions that would make the action acceptable. A policy is a trust assertion that is made by the local system and is unconditionally trusted by the system. A credential is a signed trust assertion made by other entities and the signatures must be verified before the credentials can be used. Policies and credentials are written in an assertion language. The syntax of an assertion is:

Source ASSERTS AuthorityStruct WHERE Filter

Source represents the source of the assertion, *AuthorityStruct* represents the public key(s) to whom the assertion is applicable and *Filter* is the predicate that action strings must satisfy for the assertion to hold. Filters are interpreted programs that can accept or reject action strings. Note that Policymaker does not stipulate that a particular filter language (or assertion language) has to be used. Any safe interpreted language can be used to implement either of these languages. Filter programs take as input, the current action string and the environment, which contains information about the current context (e.g. date, time, application name, etc.). This environment can be used by the filter to enforce contextual constraints such as expiration times. A filter also has access to information about the rest of the chain in which it is being evaluated, which makes it possible to design certificates that limit the degree to which their authority can be deferred. Although the filter language interpreter is external to PolicyMaker, the name of the

language is given in assertions and must be known by anyone who needs to use the assertion. Any unknown or unsupported filter languages are ignored by PolicyMaker.

The prototype for PolicyMaker had three associated assertion languages: AWKWARD (a safe version of AWK), Java and Safe-TCL. It was hoped that leaving the assertion language an open issue would mean flexibility and greater programmability for PolicyMaker. However, it was realised that the choice of an assertion language can affect the decision processing in PolicyMaker. For a local policy, the source is always *policy*. The following policy specifies that any doctor who is not a plastic surgeon should be trusted to give a check-up.

```
policy
  ASSERTS doctor_key
  WHERE filter that allows check-up if the field is not plastic surgery
```

For Policymaker to make a decision there must be at least one policy in the input supplied to it from the trust database. The following credential states that the BMA asserts that the person with key “0x12345abcd” is not a plastic surgeon.

```
BMA_key
  ASSERTS “0x12345abcd”
  WHERE filter that returns “not a plastic surgeon”, if the field is not plastic surgery
```

An assertion (whether policy or credential) states that the source trusts the public keys in the authority structure to be associated with action strings that satisfy the filter. It is important to note that assertions can modify the action strings that they accept, through the use of *Annotations*. Annotations are essentially a mechanism for communication between assertions (inter-assertion communication), as well as communication between the application and the credentials. This allows PolicyMaker to append conditions to the action strings, if necessary. A query to the PolicyMaker interpreter has the following format:

key₁, key₂, key₃, REQUESTS ActionString

To check if “0x12345abcd” is allowed to give me a check-up, the interpreter is asked:

“0x12345abcd” REQUESTS “do check-up”

The semantics of the action string is not known to PolicyMaker. The processing of the action strings, as well as signature verification, is left entirely up to the calling application. Action strings are generated and interpreted by the calling applications. The filters however should have knowledge of the action strings. The fact that signature verification is done by the calling application means that any signature scheme can be used, once the application provides the appropriate programs to perform the verification. This allows Policymaker to exploit existing signature schemes. PolicyMaker uses the credentials given to it to prove that the requested

action complies with the policy (this process is referred to as compliance checking [40]). In summary, an application gives the PolicyMaker engine, a (set of) requested action(s), a set of credentials and a policy and the engine tries to prove that the credentials contain a proof that the requested action(s) complies with the policy.

KeyNote [41, 44, 45], the successor to PolicyMaker, was developed to improve on the weaknesses of PolicyMaker by AT&T Research Laboratories. It has the same design principles of assertions and queries [42, 44, 45] but includes two additional design goals, namely: standardisation and ease of integration [45]. In KeyNote, more is done in the trust management engine, rather than in the calling application (as was the case in PolicyMaker). Signature verification is done in the KeyNote engine and a specific assertion language is used. PolicyMaker allowed any choice of assertion language, which made its compliance checker difficult to integrate with applications. KeyNote's predefined assertion language allows simpler integration with its compliance checker. The KeyNote engine is passed a list of credentials, policies, the public keys of the requester and an 'Action Environment' (which is essentially a list of attribute-value pairs) by the calling application. The action environment is generated by the application and contains all the information relevant to the request, and so accurately reflects the application's security requirements. Identifying the attributes of this environment is the essential task in integrating KeyNote into any application. The result of the KeyNote evaluation process is an application-defined string, the simplest response being 'authorized'. The KeyNote assertion format is similar to email headers and is outlined in [45]. An example of a KeyNote assertion, taken from [42], is:

```
KeyNote-Version: 1
Authorizer: rsa-pkcs-hex:"1023abcd"
Licensees: dsa-hex "986512a1" || rsa-pkcs1-hex:"19abcd02"
Comment: Authorizer delegates read access to either of the Licensees
Conditions: ($file == "etc/passwd" && $access == "read") -> {return "ok2"}
Signature: rsa-md5-pkcs1-hex:"f00f5673"
```

As in PolicyMaker, assertions can either be policies or credentials. POLICY in the Authorizer field identifies policies which are locally trusted and so do not need a signature. The Licensees field specifies the principal(s) to which authority is given. A simple, lightweight assertion language with no loops or recursion is used in order to enforce resource usage restrictions, to allow the assertions to be easily understood by humans and easily refined from high-level languages, etc. [42]. Compliance checking in PolicyMaker required repeated evaluation of assertions, along with an arbitrated 'blackboard' for storage of intermediate results and communication between assertions, while compliance checking in KeyNote involves a depth-

first search that tries (using recursion) to satisfy at least one policy. KeyNote has no inter-assertion communication mechanisms. Satisfying an assertion entails satisfying both the Conditions and Licensees fields. The current implementation of the KeyNote Toolkit is written in C. Neither system addresses the problem of how to discover that credentials are missing, and neither system supports negative assertions. The authors claimed that both these systems are a more general solution to the trust management problem than public-key certificates. However, they address only authorisation based on public keys, which still does not comprehensively cover the entire trust management problem. They focus on establishing resource access trust, and possibly service provision trust.

2.6.4 REFEREE

REFEREE [48, 49] (Rule-controlled Environment For Evaluation of Rules and Everything Else) is a trust management system for making access decisions relating to Web documents developed by Yang-Hua Chu based on PolicyMaker. It considers a PICS label as the stereotypical web credential and uses the same theoretical framework as PolicyMaker to interpret trust policies and administer trust protocols, which are represented as software modules. Like PolicyMaker and KeyNote, REFEREE is a recommendation-based, query engine so it needs to be integrated into a host application. It evaluates requests and returns a tri-value and a statement-list, which is the justification for the answer. A tri-value is either true, false or unknown. *True* means ‘yes, the action may be taken because sufficient credentials exist for the action to be approved’, *false* means ‘no, the action must not be taken because sufficient credentials exist to deny the action’ and *unknown* means ‘the trust management system was unable to find sufficient credentials either to approve or deny the requested action’. Boolean operators were modified to allow reasoning about tri-values and special operators were added to create a complete logical framework for tri-values. For example, true-if-unknown and false-if-unknown operators were defined to simulate negation of the *unknown* value. An ordered statement-list specifies information acquired during the execution of modules. They are the means by which inter-module communication takes place. All statements are ‘two element *s*-expressions’, similar to attribute value pairs. The first item specifies the context of the statement and the second stating the statement’s content. For example, the statement that John is untrustworthy in a certification REFEREE module would be:

```
( ( "certification module" ) ( "John" (untrustworthy yes) ) )
```

The REFEREE system is essentially a collection of modules as basic building blocks, each dealing with a particular policy decision. A module can delegate subtasks to other modules and make decisions based on the returned assertions. All modules have the same interface as REFEREE – they accept inputs and return a tri-value and a statement-list. The inputs are an action name and other arguments that provide information about the action and form the module’s trust database. For example, a content selection module may have either a URL or the keys of the raters that make assertions about the URL as its input and its output is a tri-value with a statement-list. At the implementation level, a module consists of a policy and zero or more interpreters. A policy is a code segment written in a trust policy language and the interpreters are programs for interpreting the policy or other interpreters. The set of interpreters in a module is hierarchical; the module policy is interpreted by the highest-level interpreter; which in turn is interpreted by a lower-level interpreter and so on.

REFEREE goes through two phases in its lifetime. In the bootstrap phase, the host application gives it the unconditionally trusted assertions and a module database. A module database is a repository of action names, similar to a DNS server in that it allows a module to be referred to by an action name. In the query phase, the host application provides the action and other arguments such as credentials, which are passed onto the appropriate module from the module database. REFEREE runs the module’s interpreter with the policy and list of arguments, which may result in other modules being invoked, then returns an answer to the host application.

Profiles-0.92 is a rule-based trust policy language, designed to work with REFEREE, in which each rule is an s-expression with an operator as the first element followed by operands. Rules are evaluated top down and the returned value of the last rule is the policy’s returned value. Rules return tri-values and statement-lists. The BNF for the syntax can be found in [49]. The following policies highlight some features of this language.

```
(
  threshold-and
  2
  (not (url-match URL ("http://www.cam.ac.uk" "http://www.bath.ac.uk")))
  (url-match URL ("http://www.ic.ac.uk"))
  unknown
)
```

The above policy states that all material from Cambridge University and the University of Bath will be blocked, and only material from Imperial College will be automatically downloaded. The user will be prompted about all other material.

```
(
  invoke "load-label" STATEMENT-LIST URL
  "http://web.mit.edu/ratings/CodeSafety.html"
```

```

        ("http://bureau.mit.edu" "http://bureau.cmu.edu")
    )
    (
        match
            (("load-label")
            (((version "PICS-1.1") *
            (service "http://web.mit.edu/ratings/CodeSafety.html") *
            (ratings (RESTRICT > virus 8))
            )))
        STATEMENT-LIST
    )

```

This policy states that labels from the MIT and CMU bureaus should be used and only pages with labels that state that the document has been thoroughly checked for viruses can be downloaded. For this example, the invoke clause runs the load label module, which loads the labels from the bureaus. The match clause searches all the labels for the pattern described.

2.6.5 SD3

SD3 (Secure Dynamically Distributed Datalog) [58] is a trust management system that uses logic to represent security policies. It consists of a high-level policy language, a local policy evaluator and a certificate retrieval system. The policy language is an extension of datalog, which is a database programming language. SD3 is an extension of work done by Gunter and Jim on QCM [121-123], which is a system for automatic certificate management. Like PolicyMaker, KeyNote and REFEREE, SD3 is a recommendation-based query system. Unlike these other systems, SD3 uses a logic-based language as its default notation.

SD3 defines a scenario using a set of rules of the form:

$$\begin{aligned}
 T(x,y) &:- K\$E(x,y) && \dots\dots\dots(1) \\
 T(x,y) &:- (K@A)\$E(x,y) && \dots\dots\dots(2)
 \end{aligned}$$

Rule 1 states that $T(x,y)$ holds provided that $K\$E(x,y)$ holds, where $K\$E(x,y)$ is the name for relations $E(x,y)$ under the control of the keyholder of public key K . Rule 2 stipulates that $T(x,y)$ holds if $E(x,y)$, which is defined under the control of the keyholder of public key K at IP address A , holds.

After the scenario is coded into a program, evaluation is started by passing this program, a query and the input certificates to the SD3 optimiser. An optimised set of rules is then passed to the SD3 core evaluator. The evaluator produces an answer to the query and a proof that the answer follows from the security policy (specified in the program). Before the answer is returned, the proof is checked and incorrect proofs are reported as errors. Thus, SD3 is conceptually very similar to KeyNote. The difference between the two systems lies in the

nature of their specification languages and their verification mechanism. In both cases, SD3 tends to be more explicitly logic-based. SD3 suffers from the same problems as KeyNote does. However, though the treatment of negative credentials is not mentioned in current literature on SD3, it may be theoretically able to handle some negative credentials due to its use of datalog.

2.6.6 Fidelis

Fidelis [124] is a policy-driven trust management framework which originates from the work on OASIS (Open Architecture for Secure, Internetworking Services) [125-127], which is a role-based architecture for distributed authorization management. In [124], Yao states that the Fidelis policy framework is an abstract, conceptual foundation, which is used as a starting reference point for the implementation of a policy language.

Conceptually, the Fidelis framework assumes the same view of trust management as PolicyMaker-based systems. However, Fidelis defines policies and credentials differently from PolicyMaker-based systems and includes the notions of a trust network and trust conveyance. Credentials are assertions that have no processing semantics and policies interpret credentials with locally-defined semantics. Policies are specified independently and separately from the principal which specifies and issues credentials. Principals represent uniquely identifiable individuals or processes. Fidelis, which is public-key oriented, facilitates three types of principals: simple, group and threshold principals. A simple principal is a single public key, while group and threshold principals (i.e. composite principals) are one or more simple principals grouped into a logical unit.

The basic construct of the Fidelis framework is the trust statement, which is “*a template of an assertion, which may be instantiated to create trust instances*” [124], where a trust instance represents “*specific trust that a principal has with respect to another principal*” [124]. A trust instance has the form:

$$A.t(v_1, v_2, \dots, v_n) : B \rightarrow C$$

$A.t(v_1, v_2, \dots, v_n)$ is a concrete assertion, A is a principal, t is a trust statement identifier defined by A , v_i is a value of the type of the i -th attribute of $A.t$, B (the trustor) is the principal who issues the trust instance and C (the subject) is the principal with whom the trust instance is related. The trustor and subject cannot be threshold principals and every trust instance has an associated validity condition.

A trust network is a collection of trust instances. Trust conveyance refers to the act of passing a trust instance from one principal to another. Fidelis also models actions as “*operations that are subject to trust computation*” [124]. It is assumed that actions represent application behaviour and that their representation allows interfacing with the trust computation.

In Fidelis, a policy is a set of rules. Trust policies determine the issuance of trust instances and action policies determine the invocation of actions. There is no standard for policy specification in the framework. Policies are local and expressed in the language or mechanism used by a principal. The Fidelis policy evaluation model checks that a policy, P , is true for a trust network T .

$$T \vdash P$$

The Fidelis Policy Language (FPL) is the concrete instantiation of ideas in the Fidelis framework. The FPL, which is described in [124], is a language that allows the representation of principals, trust statements, actions, trust policies, action policies and validity conditions. The syntax and semantics of FPL has a strong association with first-order predicate logic. The primary issues with Fidelis are that: 1) there seems to be no differentiation between the various degrees of trust that may be possible, 2) though a policy is checked against the trust network, there seems to be no means of arbitrarily analysing the policy with respect to the trust network, 3) it is not clear how the Fidelis system would handle policy specifications from principals using different local languages in order to perform policy evaluation, and 3) the assumption of the abstract nature of the framework is weakened by the key-centric perspective taken in defining its basic elements.

2.6.7 IBM Trust Establishment Framework

IBM views trust establishment as the enabling component of E-Commerce [57, 128]. They state that the underlying trust implications involved in an e-business transaction can be solved using certificates. Certificates can be issued by various bodies, vouching for an entity in a particular role, for example vouching for someone’s status as a buyer or seller or both. IBM has developed a role-based access control model that uses certificates, a Java-based Trust Establishment module and a Trust Policy Language (TPL). Their system is similar to PolicyMaker, but permits negative rules preventing access. The default certificate scheme used is X.509 v3, though other certificate formats are supported. The Trust Establishment module validates the client’s certificate and then maps the certificate owner to a role. The certificate need not bind to a user’s identity, but could just state that the user is an employee of Company

XYZ or a public key can be used to map onto an anonymous role. Local policy specified in their TPL defines what a role is permitted to do. The syntax for TPL is written in XML and is described in [57, 128]. The primitive structure in TPL is a group. For each group, there are rules governing group membership. These rules essentially specify which certificates to check. The following example is taken from [57].

```
<POLICY>
  <GROUP NAME="self">
  </GROUP>
  <GROUP NAME="partners">
    <RULE>
      <INCLUSION ID="partner" TYPE="partner" FROM "self">
      </INCLUSION>
    </RULE>
  </GROUP>
  <GROUP NAME="departments">
    <RULE>
      <INCLUSION ID="partner" TYPE="partner" FROM="partners">
      </INCLUSION>
    </RULE>
  </GROUP>
  <GROUP NAME="customers">
    <RULE>
      <INCLUSION ID="customer" TYPE="employee" FROM="departments">
      </INCLUSION>
      <FUNCTION>
        <GT>
          <FIELD ID="customer" NAME="rank"></FIELD>
          <CONST>3</CONST>
        </GT>
      </FUNCTION>
    </RULE>
  </GROUP>
</POLICY>
```

The first group defined is the originating *retailer*. Then, it is stated that entities that have *partner* certificates, signed by the original retailer, are placed in the group *partners*. The group *department* is defined as any user having a *partner* certificate signed by the *partners* group. Finally, the *customer* group consists of anyone that has an *employee* certificate signed by a member of the *departments* group who has a rank greater than 3. In summary, the policy states that a customer is an employee of a department of a partner company. After the Trust Establishment module has determined that an entity can be assigned to a particular role, it then sends this information to another module, which stipulates the access rights that are bound to the particular role. The main problem with this framework is that it assumes that all the information in the user's certificates is accessible. This has severe privacy implications.

2.6.8 Trustbuilder Framework

Trustbuilder [129-134] is a trust negotiation framework that extends the work on IBM Trust Establishment Framework. Trustbuilder seeks to address the problem of establishing trust between strangers through credential exchange. In this context, a credential is defined as a digitally signed assertion by a credential issuer about a credential owner. Assertions are essentially a set of attribute-value pairs. A sensitive credential is one that contains private information. The Trustbuilder negotiation model works under the assumption that a negotiation is a sequence of credential disclosures, which alternates between a client and a server. Both client and server own a set of credentials and each has a policy governing access to its credentials and services. Each participant in the negotiation is represented by a security agent (SA), which manages the disclosure of local credentials, enforces service-governing credentials and credential access policies, stores and processes accumulated credentials and incoming and outgoing requests.

Figure 2.5 shows the protocol used in a TrustBuilder negotiation and thus highlights the role of the security agent. Local Site can represent either a client or server. When it is a client, processes **21** and **102** are not manifested. When it represents a server, process **26** is not manifested. When it represents a stateless server, processes **26**, **105** and **106** are not manifested. The arrows in Figure 2.5 show the direction of the interaction. A typical negotiation would proceed as follows. Initially, the client makes a service request, through its SA, to the server (**21**). Upon receipt of the request, the SA for the server makes an authorization decision based on the service-governing policy (**102**). If the service cannot be authorized immediately, then the client and server engage in a negotiation strategy to determine if authorization is possible. When the client SA has attached the appropriate credentials to the service request so that the service will be authorized, the server SA passes the request to the server and the negotiation is successful. The exchange/disclosure of credentials is governed by a negotiation strategy, which will be presented later. Not all negotiations are successful. The information required by each party may be set at a value that prohibits a successful negotiation.

The negotiation strategy employed may either be an eager strategy, a parsimonious strategy or a hybrid of both strategies. In the eager strategy, each entity takes turn sending each other every unlocked credential that they possess. The negotiation is successful when each party has received enough credentials to be confident in engaging in the transaction. The process is unsuccessful when the client receives a set of credentials that it has already received and or does

not add any value to his current knowledge base. In the parsimonious strategy, the parties start exchanging credential requests (not actual credentials) and try to find a possible sequence of credential disclosures that can lead to a successful negotiation. The parsimonious strategy allows limited, controlled release of credentials.

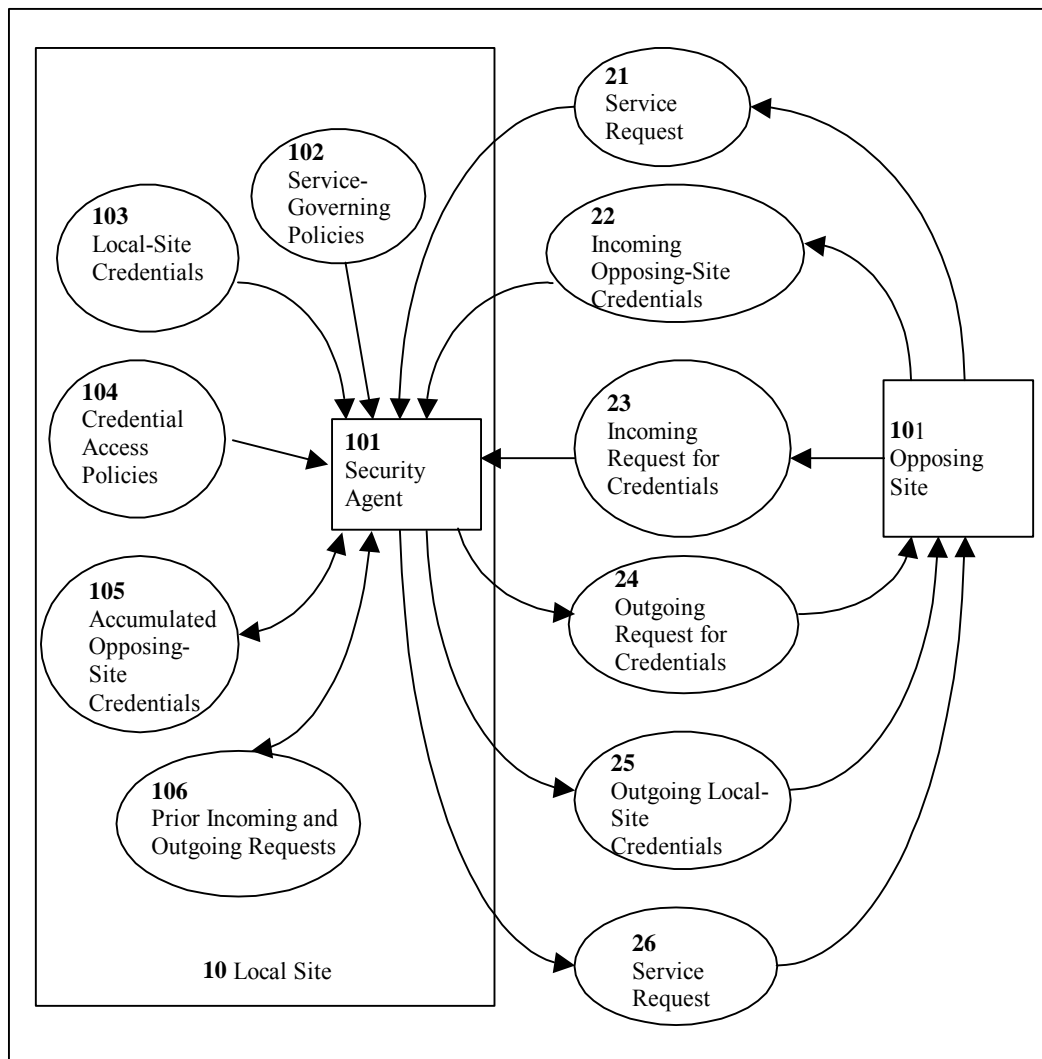


Figure 2.5: Trustbuilder Negotiation Process

Credential Access Policy (called credential expressions) is expressed using a credential expression language that is separated into two parts, a Property-based Authentication Language (PAL) and a Role-based Authorization Language (RAL). Conceptually, this distinction mirrors the work on IBM TEF. However, in IBM TEF the Trust Establishment Service that would handle the mapping from roles to access rights was viewed as a black box. A PAL policy defines one or more roles, where a role is a property of subjects defined in terms if the

credentials they possess and the attribute values of those credentials. A PAL policy has the following form:

$$\begin{aligned} & \text{role}_1(\text{Attribute}_1, \dots, \text{Attribute}_n) \\ & \leftarrow \text{CredentialVariable} : \text{CredentialType}, \dots, \\ & \quad \text{role}_m(\text{CredentialVariable.issuer}, \text{Attribut}_1, \dots, \text{Attribut}_n), \dots, \\ & \quad \text{credentialConstraint}_1, \dots, \text{credentialConstraint}_n . \end{aligned}$$

The above states that any entity that provides a credential of type `CredentialType` that satisfies `rolem` and that satisfies the credential constraints should be assigned to `role1` with the specified attributes. Credential variables and role attributes must start with Capital letters. Roles, credential types and credential attributes must start with lowercase letters. A RAL authorization policy consists of a role-constraint expression, together with a PAL policy defining each role used in the expression. A role-constraint expression is a formula consisting of role applications and attribute constraints combined by logical AND and logical OR. An example of a RAL policy that would govern a request for a service that schedules shipping is (adapted from [131]):

$$\begin{aligned} & \text{HasCredit}(\text{Client}, \text{Amount}) \text{ AND} \\ & \text{Amount} > \text{Tons} \times \text{costPerTon}(\text{PickupLocation}, \text{Destination}) \end{aligned}$$

The TrustBuilder framework is focused on trust establishment, but does not include facilities for handling credentials regarding negative assertions and assumes nothing about the consistency of the credential expressions.

2.6.9 TCPA

TCPA [52-54] (Trusted Computing Platform Alliance) is a consortium of companies to make the computer platform trustworthy. It was formed by Compaq, HP, IBM, Intel and Microsoft. The motivation behind the decision to create an alliance was the realization by these companies that customers viewed their products as not worthy of their trust. In their own words:

“... came to an important conclusion: the level, or amount, of trust they were able to deliver to their customers, and upon which a great deal of the information revolution depended, needed to be increased and security solutions for PC's needed to be easy to deploy, use and manage.”

The five founding companies invited a group of hardware, software, communications and technology vendors to help in the definition and implementation of a hardware and operating system-based platform that would implement trust into client, server, networking and communication systems.

In the TCPA main specification document [53] and the TCPA PC specific implementation document [52], a complex web of data structures and processes is defined. From the document, it becomes apparent that TCPA is heavily rooted in using cryptography to verify each and every component being used on a system. From boot-up, encryption keys and certificates in the hardware are validated and integrity checks are performed to validate operating system elements as well as any software that may be run on the system. The specification documents propose a X.509-like infrastructure, with the TCPA Subsystem as its root. The TCPA Subsystem consists of two basic building blocks, namely: a Trusted Platform Module (TPM), which is a hardware instantiation of the TCPA specification, and software to perform integrity metrics, in conjunction with the TPM. A TCPA-compliant computer would behave in the following manner:

- The PC is turned on.
- The TCPA-compliant ‘BIOS Boot Block’ and TPM have a conversation, which attests to the fact that the BIOS can be trusted.
- The BIOS queries to ensure that the user is authorised to use the platform.
- The BIOS has a conversation with the operating system (OS) loader and the TPM, which attests to the fact that the OS loader can be trusted.
- The OS loader has a conversation with the OS kernel. When the OS kernel loads, it knows what software has had access to the system ahead of it. This should establish that whatever happens within the system from that point is 100 percent controlled by the OS kernel.

The above steps are taken from [51]. When an application wishes to be executed on a TCPA-compliant PC, it must first be deemed trustworthy by:

- providing the PC with integrity metrics that verify it as trustworthy,
- having integrity metrics provided on its behalf by a trusted third party, or
- having metrics about its trustworthiness stored on the PC itself.

From the information available in the public domain, it seems that there is very little conceptual difference between the TCPA framework and a PKI. The TCPA’s notion and use of integrity metrics seem to be equivalent to that of public keys. The TCPA Framework will not only suffer from the same problems faced by PKIs, but may lead to more dangerous scenarios. The framework may inadvertently infringe on end user privacy and give consortium members an unfair business advantage (the ability to effectively shut out any rival software by labelling them from an untrustworthy source). It is not clear from the literature whether the actions of trusted applications are constrained or even if proof of trustworthiness is done with respect to a

particular (set of) task(s) that the application wants to perform. Some other problems with the TCPA framework are outlined in [37]. A detailed description of the TCPA initiative can be found in their Design Philosophies and Concepts paper [50].

2.6.10 Poblano Distributed Trust Model

Poblano [47] is an attempt by Chen and Yeager of Sun Microsystems to build a decentralized trust model based on the JXTA platform. The JXTA project is concerned with designing and implementing free software that would enable the easy creation, use and maintenance of peer-to-peer networks. Poblano is based on the assumption that each individual has his or her opinion on the trustworthiness of another. It is believed that these opinions can be collected, exchanged and evaluated. The initial application areas for this distributed trust model were 1) a reputation guided searching system, and 2) a recommendation system for security purposes. The broader objective of this project is to allow this model to be adapted to work in any scenario where distributed trust relationships are used. The model describes trust relationships between peers and also between peers and codat (code and data), a protocol that allows the dissemination of trust and algorithms for the updating of trust. Each peer has information available to it on either the group's or its confidence in the codat and peer and about the risk involved. Using this information, the model describes the formulas and mechanisms used to calculate trust values and propagate trust information. The first problem with the Poblano trust model is that the equations for calculating/updating trust are simple and arbitrary. Thus, the accuracy and general applicability of this formula is questionable. The second problem with the model is that it seems specifically designed for solving the problem of searching distributed networks and the validation of the search results and sources. Finally, the Poblano framework appears to have no facilities for the specification of constraints. This severely limits its usability. Thus, Poblano may not be easily adapted to work in applications domains that are dissimilar to this one. The third problem is that it may suffer from the same key problems faced by public key certificates.

2.6.11 Emerging Trust Management Solutions

In this section, emerging trust management systems will be presented. These systems are still in development.

RT [135, 136], which stands for Role-based Trust Management Framework is a successor to TrustBuilder. It also builds upon work done by Ninghui Li on Trust Management and

Delegation Logic, while he was at New York University. RT introduces the notion of attribute acknowledgement policies, which protects information about the possession of credentials. It also adds notions of a trust target graph protocol and the distributed storage of credentials.

In [137], an Active Trust Management System for Autonomous Adaptive Survivable Systems is proposed. Survivable systems tend to assume absolute trust. This assumption is not feasible given the non-existence of an impenetrable and incorruptible TCB. Shrobe et al. propose the development of a survivable system that could function in an imperfect environment. The system should constantly collect and analyse security-related data from a broad variety of sources, including the application systems, intrusion detection systems, system logs, network traffic analysers, etc. The nature and scope of the collected data are determined by the user of this system and his particular needs. The result of these analyses forms their trust model, which is a probabilistic representation of the trustworthiness of each computational resource in the system. The applications use this trust model to help decide which resources should be used to perform each major computational step by maximizing the ratio of expected benefit to risk.

Kagal et al. [138] describe a trust management framework that uses a system of rights and delegations, as well as digital certificates, to facilitate trust management. Their architecture assumes that each group of agents is protected by security agents, which are responsible for authorizing access to services/resources within the group. The idea is that a client can request access to a resource or service by providing its identity information along with any delegations it may have to the security agent for the domain. The security agent uses its policies to verify the identity and delegations of the client, granting access only if everything is valid. This system uses a specification language that seems to be derived from work done by Jajodia et al. [88] and combines it with work done by Blaze et al. [39].

A Hybrid Trust Management Model For MAS (Multi Agent Systems) Based Trading Society is proposed in [139]. The theoretical underpinnings are based on [3, 4], while the trust management model is based on Witkowski et al.'s work [140] and Abdul-Rahman et al.'s work [87]. This work is in its infancy. It proposed that its model will combine the notions of reputation, subjective trust and objective trust and will focus on managing trust-based trading relationships.

2.7 Summary

In this chapter, the various views of trust were discussed and their common traits identified. For Internet applications, trust is defined as:

“the quantified belief by a trustor with respect to the competence, honesty, security and dependability of a trustee within a specified context.”

A definition of distrust was also given, because it is useful in some scenarios.

It was stated that a trust relationship always has an associated context, always occurs between two entities (or sets of entities), has an associated measure/level of trust associated to it and that it is a mathematically-defined binary relation that adheres to none of the standard properties of binary relations. The factors that influence trust (risk, experience, trust propensity, market forces, etc.) were briefly discussed.

A classification scheme of the various contexts in which trust is used was presented. The fact that a definite combination of attributes cannot be assigned to a particular context, because different circumstances require different attributes to be dominant, was highlighted.

Attempts to create logic-based, computational and HCI-based models of trust were reviewed. The contemporary view of trust management was discussed and trust management for Internet applications defined as:

“the activity of collecting, encoding, 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.”

Finally, a few current trust management solutions were presented, namely: public key certificates, PICS, PolicyMaker, KeyNote, REFEREE, SD3, Fidelis, IBM TEF, Trustbuilder, TCPA, Poblano and a few others. A common flaw with all these solutions is that they are used to identify a static form of trust, usually at the discretion of the application coder (that is, the programmer inserts code to evaluate trust, often at the start of a session). However, trust can change over time. Typically, a customer uses an unknown service provider with some trepidation but if the service provided is high quality over a period of time, the customer's trust in the service provider increases. In order to handle this dynamic property of trust, solutions should design a framework that assumes non-monotonicity. They should be able to adapt to the

changing conditions of the environment in which the trust decision was made. Systems should be able to incorporate their own experiences (or that of others) in their decision-making process. Systems change and evolve so there is a need to monitor trust relationships to determine whether the criteria on which they are based still apply. This could also involve the process of keeping track of the activities of the trustee and of determining the necessary action needed when the trustee violates the trustor's trust.

Chapter 3 Specifying Trust

*“The notion of having to specify trust relationships – central as it is to the analysis of protocols – is an unfamiliar one.”
– Simmons [141]*

This chapter introduces the SULTAN specification notation and the tasks involved in specifying trust relationships. Examples will be used to demonstrate use of the constructs of the notation. The following convention will be used in presenting language syntax in this and the remaining chapters. Reserved words will be in bold. Definitions will use * to represent zero or more repetitions, ? to represent an optional element (present or not) and | to represent a choice between components. The complete syntax of the language, written in SableCC, is presented in Appendix A.

3.1 Requirements for a Trust Notation

A trust notation that is to be used by Internet applications should have facilities for the following:

- The specification of trust and distrust statements
- The specification of positive and negative recommendations
- The specification of conditions on trust and recommendation relationships
- Access to risk and experience facilities
- Easy inclusion in an analysis framework

Additionally, a trust notation should satisfy the following:

- Be expressive
- Possess a clear, unambiguous and well-defined semantics

Expressiveness refers not only to the notation’s ability to represent a range of trust relationships, but also refers to its ability to allow the encoding of validity constraints, credentials, credential chains, arbitrary constraint and credential combinations and system or organisation-specific properties with a trust specification. As Internet applications are expected to cross network boundaries, it is crucial that the meaning of a trust relationship is clear. Specifications written in such a semantically well-defined, clear and unambiguous notation will be harder to misinterpret (and thus misuse) on foreign networks.

3.2 The SULTAN Specification Notation

The SULTAN specification notation allows for the definition of trust requirements and recommendations. There are two primitive constructs: the trust construct and the recommendation construct. Trust and distrust statements are specified using the trust construct, while positive and negative recommendations are specified using the recommend construct.

3.2.1 The trust construct

The trust construct has the following syntactic form:

$$\text{PolicyName} : \mathbf{trust} (Tr, Te, As, L) \leftarrow Cs;$$

The semantic interpretation of a statement in the form above is that: *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*, the trustor, is the entity that is trusting. *Te*, the trustee, is the entity to be trusted. *As*, the action set, is a colon-delimited list of actions (function names which effectively specify the context) or action prohibitions (discussed further in section 3.2.6). 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 the trustor or trustee). *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*, the constraint set, is 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. Each of the elements of this construct is discussed later in this chapter. Now, a few examples of SULTAN trust statements are given.

```
CustomerVer: trust(Supplier, Customers, view_pages(Supplier), 100 )
← GoodCredit(Customers) &
  risk(Supplier, Customers, _) <= 2;
```

Supplier trusts Customers to perform view_pages(Supplier) at trust level 100 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. It is important to note view_pages(Supplier) represents the function defined on the Supplier entity. This example illustrates the use of two features of the notation: the anonymous variable and auxiliary functions. A variable in the notation is a series

of characters with the underscore prefixed. `_Var` is an example of a named SULTAN variable. The anonymous variable is represented by the underscore and it signifies a value that is of no interest, but which has to be included due to the definition of a function. There is an anonymous variable in the risk auxiliary function used in the constraints section. The risk auxiliary function will be explained in more detailed later in this chapter.

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

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";
```

Morris trusts Symantec to perform `do_definition_update(Morris, Computer)` at trust level `HighTrust` if `DefinitionState(Symantec) = "old"`.

```
Store: trust ( Naranker , TicketMachine, SupplyTicket(TicketMachine, Amount, Destination), 50 )
← Amount <= 5;
```

Naranker trusts TicketMachine to perform `SupplyTicket(TicketMachine, Amount, Destination)` at trust level 50 if Amount is less than or equal to 5.

```
WebUserCheck: trust(WebServer, _User, access_se(WebServer):view_pages(WebServer), 10 )
← RealEstatePassport(_User);
```

WebServer trusts any entity, `_User`, to perform `access_se(WebServer)` and `view_pages(WebServer)` at trust level 10 if `RealEstatePassport(_User)` is true.

3.2.2 The recommend construct

A recommendation has the following form:

$$\text{PolicyName} : \mathbf{recommend} (Rr, Re, As, L) \leftarrow Cs;$$

Semantically, the above statement means that *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*, the recommender, is the name of the entity making the recommendation. *Re*, the recommendee, is the name of the entity that the

recommendation is about. L , the recommendation level, is 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. It is important to point out that the recommendation level and trust level are assumed to be independent of each other, unless otherwise specified. As , the recommended action set, is 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 , the constraint set, is a delimited set of constraints that must be satisfied for the recommendation to be valid. Delimiters include the logical and (&) and logical or (|).

A recommendation may result in a trust specification (i.e. a recommendation may be the basis for a trust specification) and vice versa. However, the trust level need not correspond to the recommendation level. The interaction between trust constructs and recommend constructs will be discussed in more detail in sections to follow. The following example illustrates a typical recommend statement.

```
TomTCPA: recommend (TomTPM, _App, provideUpdate(_App, _TomMachine), 70)
← isOn(TomTPM, _TomMachine) & verified(TomTPM, _App);
```

TomTPM recommends any entity, $_App$, at recommendation level 70 to perform $provideUpdate(_App, _TomMachine)$ if $isOn(TomTPM, _TomMachine)$ is true and $verified(TomTPM, _App)$ is true.

```
ABM: recommend ( NatWest, _Client, getCredit(_Client, _SwitchCard), 100)
← isClient(NatWest, _Client) & isValidCard(Natwest, _SwitchCard) ;
```

NatWest recommends any entity, $_Client$, at recommendation level 100 to perform $getCredit(_Client, _SwitchCard)$ if $isClient(NatWest, _Client)$ and $isValidCard(Natwest, _SwitchCard)$ are both true.

```
Veri: recommend ( Verisign, _KeyHolder, loadScript(_X), -50)
← isCustomer(VeriSign, _KeyHolder) & isUsedBy(Verisign, _X) &
  outStandingBalance(VeriSign, _KeyHolder) > 40;
```

VeriSign does not recommend, any entity, $_KeyHolder$, at recommendation level -50 to perform $loadScript(_X)$ if $isCustomer(VeriSign, _KeyHolder)$ is true, $isUsedBy(Verisign, _X)$ is true and $outStandingBalance(VeriSign, _KeyHolder)$ is greater than 40.

The examples above give an overview of the primary constructs of the SULTAN specification notation. However, the details involved in specifying the elementary components of a construct

need to be addressed. There will be a few basic definitions that will be assumed. They are covered in Appendix B.

3.2.3 Specifying Policy Names

Policy names are identification tags for SULTAN specification statements. They uniquely identify policies and provide the administrator a means of grouping related statements. This is a precursor to the task of grouping the specifications into namespaces. Policy names are used when performing analysis. They are normally the answers to SULTAN analysis questions.

In defining the syntax for a policy name, an abstract type called reference that will be used in the definitions of other elements must be introduced. A reference is a sequence of characters, starting with a letter with the remaining characters being either an underscore, letter or digit. A policy name is simply defined as a reference.

```
reference      = letter (underscore | letter | digit)*;
policyname    = reference;
```

3.2.4 Specifying Entity Names

An entity name is a symbolic name for an object that will be used in a specification. Entity names can represent domains (group of entities) or individual entities. The reserved word *everyone* can be used as an entity name for a trustee to refer to all the entities in the user's domain space. Note that *everyone* cannot be used as a trustor name. Specifying that everyone trusts would not make sense. The reserved word *foreign* may be used as a trustee name to refer to external entities that are not under the control of the administrator. It helps in the definition of default policies for unknown entities. Syntactically, an entity name is either a reference (as defined above) or a variable, which is an underscore followed by a reference.

```
variable      = underscore reference?;
word          = (reference | variable);
```

3.2.5 Specifying Levels

The level is a measure of either trust (in the case of trust statements) or of confidence in a recommendation (for a recommend statement). It is crucial because both trust requirements and recommendation are concepts that require a degree of quantification. Levels can be expressed either as an integer in the range -100 to 100 (inclusive, 0 excluded) or a label (where a label is defined as a reference) that represents such an integer. It must be explicitly stated that labels can be SULTAN variables or a mnemonic tag.

```

number    =  '-? '1'..'9'+ digit*;
level     =  (number | reference | variable);

```

An anonymous variable can never be used as a trust level. When a named variable is used as a trust level, the named variable must be present in the specification's constraints as a part of a comparison. Recommendation levels may be used as the criteria upon which a decision is made. Recommendation levels also have the same restriction as trust levels, in that a recommendation level cannot be an anonymous variable and when used as a named variable it must be present in the constraints. It should be noted that trust and recommendation levels are not the same and there is no assumed connection between them. If a connection is required, then it can be specified.

3.2.6 Specifying Action Sets

Action sets define the context, in terms of a set of trusted or recommended actions (depending on the type of statement being specified). Action sets are collections of actions and or action restrictions. An action is similar to a function or object method and an action restriction specifies that the trustee is trusted not to perform an action. An action restriction is specified by using an action name as the parameter to the **not** function. The **not** function can be used for either a single action or an action set (without action restrictions). Examples of action restrictions are: **not**(view_certificate(Client)) and **not**(executeScript(MyComp,_ScriptName) : disableNAV(MyComp)). Currently, action restrictions should only be used with positive values for the level component of the specification. This is due to the fact that 'X distrusts Y not to perform actions A' is meaningless.

The action name indicates the entity on which the action is defined (the first parameter of an action name). An action set is a colon-delimited list of actions or action prohibitions. For added expressiveness, an action set may also be a variable. The syntax rules for an action set are:

```

action     =  function ;
actions    =  action (colon action)*;
nactions   =  'not(' actions ')';
actionset  =  variable | (( action | nactions ) (colon (action | nactions) )*) ;

```

Note that the first parameter of an action can never be an anonymous variable. However, it may be a named variable if that variable is used somewhere else in the head or in the constraints of the specification.

3.2.7 Specifying Constraints

Constraints are conditions that must be satisfied before a trust relationship can be established. They can be viewed as pre-requisites to trust establishment, guidelines for evidential acquisition, or necessary assertions for contract establishment. In the specification language, constraints are broadly viewed as conditions necessary for the initial establishment and the re-initiation of a trust relationship.

It is assumed that constraints for a SULTAN specification statement must evaluate to a Boolean value. Constraints are either function calls (which return true or false) or references (which evaluate to true or false) or logical expressions involving variables, functions or references. Logical expressions have a left hand side, an operator and a value, which is either a number or a reference or a string. The syntax rules for a constraint are:

expression	=	(variable function reference) op value;
op	=	'>' '<' '=' '!=' '<=' '>=';
value	=	number reference string;
constraint	=	function reference expression;

It should be mentioned that the = operator has two functions. The first function is an assignment operator. When the left hand side is a variable, the = operator performs an assignment (instantiation). The variable is given the value of the right hand side and the entire expression is assumed true. The second function is comparison. If the left-hand side is not a variable, then an evaluation of both sides take place to see if they are identical.

A constraint set is a collection of constraints delimited by the operators (& and |). The syntax rules for a constraint set are:

log_op	=	'&' ' ';
constraints	=	constraint (log_op constraint)*;

Though, the construction of the entire set of SULTAN constraints has been discussed, there is a need to take a look at constraints from a more abstract perspective. Conceptually, there are two classes of constraints, namely: SULTAN-defined constraints and user-defined constraints. SULTAN-defined constraints can be either trust constraints or recommend constraints or functions from the auxiliary specification library.

Trust Constraints

Trust statements can be based on trust in others. Thus, trust can be used as constraint. For example:

```
AccFin: trust(Accounts, Finance, run_payroll(Accounts), 50)
← trust(CEO, Finance, run_payroll(Accounts), _X) & _X > 0;
```

Accounts' trust in Finance is dependent on CEO's trust in Finance. As shorthand, the functions `trust+` and `trust-` are provided. The `trust+` function represents a trust statement with an assumed positive trust level, while the `trust-` function represents a trust statement with an assumed negative trust level. `AccFin` could be redefined using the `trust+` function in the constraint section. This modification would produce:

```
AccFin2: trust(Accounts, Finance, run_payroll(Accounts), 50)
← trust+(CEO, Finance, run_payroll(Accounts));
```

Note that the `trust+` and `trust-` functions can only be used as constraints.

Recommend Constraints

Security solutions often establish trust relationships based on a recommendation from trusted third parties. For example, CIG may trust anyone that is recommended by Microsoft.

```
Arbitrary: trust(CIG, _X, _A, HighTrust) ← recommend(Microsoft, _X, _A, MediumTrust);
```

For ease of specification, `recommend+` and `recommend-` functions (which can only be used as constraints) are defined. The `recommend+` function is shorthand for a `recommend` statement with an assumed positive recommendation level, while the `recommend-` function represents a `recommend` statement with a negative recommendation level. Both have three parameters, the recommender, the recommendee and the recommended action set. An example of a trust statement that uses the `recommend+` function is:

```
Arbitrary2: trust(Internal, _X, _, HighTrust) ← recommend+(Microsoft, _X, _);
```

`Arbitrary2` is equivalent to the following statement:

```
Arbitrary3: trust(Internal, _X, _, HighTrust) ← recommend(Microsoft, _X, _, _L) & _L > 0;
```

Auxiliary Specification Library Functions

The auxiliary specification library contains useful functions that can be used in SULTAN specifications. Currently, there are two functions in this library: the `risk` function and the `experience` function.

The `risk` function is defined to allow trust relationships to utilize risk information. This is discussed further in Chapter 5. The format of the `risk` function is:

risk(B, C, A)

The semantic interpretation of the above is: the risk entity B undertakes when entity C performs A. The risk value is the probability (expressed as an integer percentage) for the failure of an activity A. The risk value is an integer between 0 and 100 (inclusive), where 0 represents no risk and 100 is the highest risk possible. The SULTAN system contains a facility that performs risk calculation. This will be discussed further in later chapters. As SULTAN constraints must always return a Boolean value, the risk function must be used as a part of a comparison (i.e. SULTAN expression). The next two examples show the risk function in use.

```
Contract: recommend(ICLondon, Sun, viewDoc(ICLondon, _UserID, _DocName), -100 )
← isvalid(ICLondon, _UserID) & isInternalFinancialDoc(ICLondon, _Docname) &
  risk(ICLondon, Sun, viewDoc(ICLondon, _UserID, _DocName)) >= 50;
```

ICLondon does not recommend Sun at recommendation level -100 to perform viewDoc(ICLondon, _UserID, _DocName) if isvalid(ICLondon, _UserID) is true and isInternalFinancialDoc(ICLondon, _DocName) is true and the ICLondon's risk in allowing Sun to perform viewDoc(ICLondon, _UserID, _DocName) is greater than or equal to 50.

```
Amaz: trust(Amazon, _AnyOne, buy_product(Amazon, _ProductID), -50 )
← risk(Amazon, _Anyone, buy_product(Amazon, _ProductID)) > 20;
```

Amazon distrusts any entity, _AnyOne, at trust level -50 to perform buy_product(Amazon, _ProductID) if Amazon's risk in allowing _Anyone to perform buy_product(Amazon, _ProductID) is greater than 20.

The other function in the auxiliary library is the experience function. The experience function allows the definition of constraints based on the experience of entities. The experience function has the following format:

experience(B, C, A)

The above is interpreted as 'B's estimate of the experience it had with C with respect to action set A.' As with the risk function, the experience function must be used in a comparison. The experience value is an integer between -100 and 100 (inclusive, 0 excluded). Negative integers (< 0) represent a negative (or bad) experience and positive integers (> 0) a positive (or good) experience.

```
Info: trust(EGovernment, _AnyCountry, provide_leg_info(_AnyCountry), 35 )
  ← experience(EGovernment, _AnyCountry, verify_leg_info(EGovernment, _AnyCountry))
    >= 10;
```


EGovernment trusts any entity, `_AnyCountry`, to perform `provide_leg_info(_AnyCountry)` at trust level 35, if the EGovernment's estimate of the experience it has had with `_AnyCountry` with respect to `verify_leg_info(EGovernment, _AnyCountry)` is greater than or equal to 10.

Supply: **recommend**(EDistributor, EReseller, market(EReseller, _Products), 100)
 ← **experience**(EDistributor, EReseller, _) > 0;

EDistributor recommends EReseller at recommendation level 100 to perform `market(EReseller, _Products)` if the EDistributor's estimate of the experience it has had with EReseller is greater than zero.

PDtrust: **trust**(NYPDHQ, GSM, provide_info(GSM, NYPDHQ), 100)
 ← **experience**(NYPDHQ, GSM, provide_info(GSM, NYPDHQ)) > 0;

NYPDHQ trusts GSM to perform `provide_info(GSM, NYPDHQ)` at trust level 100, if NYPDHQ's estimate of the experience it has had with GSM with respect to `provide_info(GSM, NYPDHQ)` is positive.

User-Defined Constraints

User-defined constraints are constraints that the user includes to tailor the specification to his/her application domain. The constraints can either be application-specific function calls or comparisons. It is assumed that all these return Boolean values. Let us look at a few examples of statements that make use of user-defined functions.

Law: **trust**(Client, ELawyers, advice(Client), 100)
 ← **Accredited**(ELawyers, USBar);

Client trusts ELawyers to perform `advice(Client)` at trust level 100, if **Accredited**(ELawyers, USBar) is true.

Doc: **recommend**(BMA, EDoctor, sell_drugs_online(EDoctor), 100)
 ← **certified**(EDoctor, BMA);

BMA recommends EDoctor at recommendation level 100 to perform `sell_drugs_online(EDoctor)`, if **certified**(EDoctor, BMA) is true.

Site: **trust**(I, WebSites, load(I), -100)
 ← **SiteSecurityLevel**(WebSites) < 3;

I distrust WebSites to perform `load(I)` at distrust level 100, if **SiteSecurityLevel**(WebSites) is less than 3.

Constraint information, i.e. data about risk, experience and the constraints of the relationships, are stored in the State Information Database. For example, the State Information Database may record the fact that `SitSecurityLevel(_X)` is 5 or that `certified(EDoctor, BMA)` is false or that **experience**(NYPDHQ, GSM, `provide_info(GSM, NYPDHQ)`) is 100. The State Information Database will be presented in more detail in Chapter 7.

3.2.8 The Trust-Recommendation Interaction

As mentioned previously, interactions between trust specifications and recommendations are possible. Shand, Dimmock and Bacon demonstrate in [142] that collaboration in ubiquitous systems may be enabled through the utilization of the trust-recommendation connection. In this section, example scenarios are used to demonstrate these interactions in more detail.

A recommendation-based trust specification

Many decisions to trust unknown (or even known) entities are based on the recommendations of individuals. Security solutions often establish trust relationships based on a recommendation from third parties. Digital certificates, which were created to vouch for the key-name binding of an entity, have been frequently used as trust decision-making tools. The possession of a digital certificate is assumed to mean that the software that contains it is from a trustworthy source. This may often be an erroneous assumption. However, it is a prime example of the influence of a third party's recommendation on trust decisions. For example, a web user trusts a program only if it has been recommended by VeriSign. Specified in the SULTAN language, this is:

```
RecBasedTrust: trust(WebUser, _X, _, HighTrust )  
← recommend(VeriSign, _X, _, GOOD);
```

A trust-based recommendation

It is not common practice to base recommendations on trust specifications, in the computer security world. However, in traditional commercial scenarios a businessman, say Gary, may recommend an informal acquaintance, say Fran, based on his trust in Fran's competence, honesty or dependability. His trust in Fran is often based on his assessment of his experiences with Fran. Just focusing on the higher-level relationship, Gary will recommend Fran based on his trust in her ability. This shows that a trust assertion can conceptually be viewed in the same light as any other constraint in a recommend statement. Figure 3.1 shows an example of a trust-based recommendation.

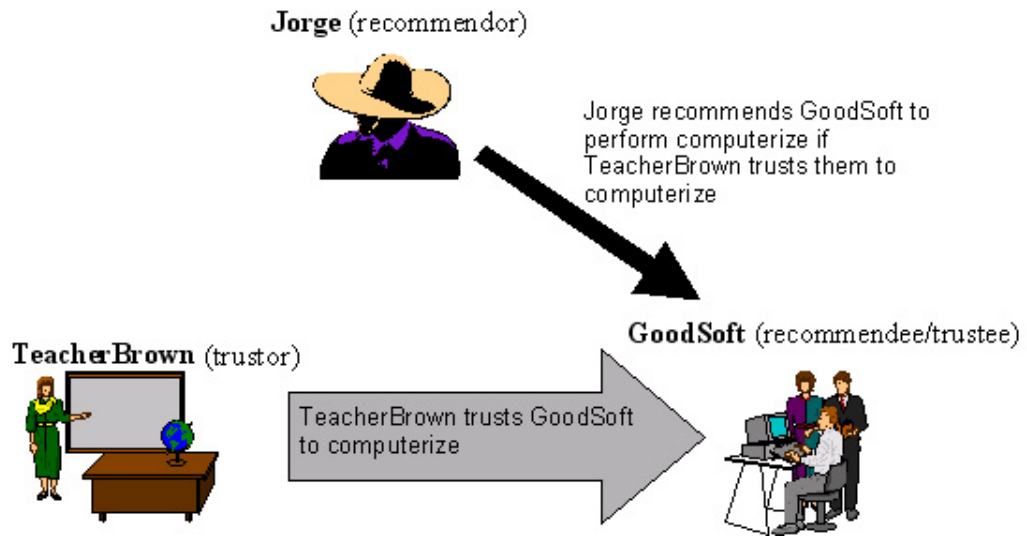


Figure 3.1: Trust-based Recommendation Scenario

In Figure 3.1, Jorge’s recommendation is based on TeacherBrown’s trust specification (probably because Jorge trusts TeacherBrown’s judgement). Written in the SULTAN specification notation, this is:

```
TBR: recommend(Jorge, GoodSoft, computerize(Jorge), CONFIDENT )
← trust(TeacherBrown, GoodSoft, computerize(_X), HighTrust );
```

3.3 Modelling Other Notations

This section presents a brief discussion on the mapping of the SULTAN specification language to a few of the trust management systems presented in Chapter 2. The demonstration of the fact that these systems can be mapped using the SULTAN specification language illustrates that the SULTAN specifications can be refined into each of these systems. In this discussion, examples from Chapter 2 are used.

3.3.1 Public Key Certificates

The public key certificate framework has no known or standard trust policy language. The tools that implement this framework allow the making of decisions relating to which certificates and/or certificate authorities to trust. As mentioned in Chapter 2, the certification authority does not vouch for the trustworthiness of the key owner, but simply authenticates the owner’s identity. This is not normally the perception of the end user. The end user normally views the holder of a digital certificate from a known or trusted source as validation of the security of the holder’s software. As this is the state of affairs, the status quo will be assumed and a few

example situations, which a user may want to state, will be modelled. The examples will be based on the PGP system. Thus, it will be assumed that trust levels are labels, with one of the following values: **Unknown**, **Untrusted**, **MarginallyTrusted** or **CompletelyTrusted**. It will also be assumed that the following functions exist: **Key(Entity)** – returns true if Entity is a key, **MetalIntroducer(Key)** – returns true if Key is a meta introducer and **Introducer(Key)**- returns true if Key is an introducer.

Examples:

Emil marginally trusts a key if it is marginally trusted by a meta-introducer and completely trusted by an introducer.

```
PK1: trust(Emil, _Key, _, MarginallyTrusted) ←
    Key(_Key) &
    trust ( _Meta, _Key, _, MarginallyTrusted) & MetalIntroducer(_Meta) &
    trust ( _Intro, _Key, _, CompletelyTrusted) & Introducer(_Intro) & _Meta != _Intro;
```

Jane completely trusts a key if it is completely trusted by Harry and marginally trusted by two people.

```
PK2: trust(Jane, _Key, _, CompletelyTrusted) ←
    trust( Harry, _Key, _, CompletelyTrusted) &
    trust( _X, _Key, _, MarginallyTrusted) &
    trust( _Y, _Key, _, MarginallyTrusted) & _X != _Y;
```

Marie marginally trusts a key if it is completely trusted by three introducers.

```
PK3: trust(Marie, _Key, _, MarginallyTrusted) ←
    trust( _X, _Key, _, CompletelyTrusted) &
    trust( _Y, _Key, _, CompletelyTrusted) &
    trust( _Z, _Key, _, CompletelyTrusted) &
    _X != _Y & _X != _Z & _Y != _Z;
```

3.3.2 PICS

Due to the purpose and application domain of the PICS solution, the following will be defined:

- **load(X, P)** – an action that signifies the loading of a web page.
- **source(labelBureau)** – represents the label bureau to be used.

The first example to be modelled states that I do not trust any Yahoo web page or any site at the department of computing at Imperial College to access my computer, but any other page is trusted to load.

```
(PicsRule-1.1
(
  Policy (RejectByURL ( "http://*@www.doc.ic.ac.uk*/*"
                      "http://*@www.yahoo.com*/*" )
        )
  Policy (AcceptIf "otherwise")
))
```

To model this example, default trust is assumed (i.e. `trust(_X,_Y, load(_X, _Y),100)`). The example can now be expressed as:

PICS1: `trust(MyComputer, DisallowedPages, load(MyComputer, DisallowedPages), -100).;`

It is assumed that `DisallowedPages` maps to `http://*@www.doc.ic.ac.uk/*` and `http://*www.yahoo.com/*`. The second example to be presented illustrates the use of a label bureau (a source of labels).

```
(PicsRule-1.1
(
  ServiceInfo (name "http://www.raters.org/ratings/v1.html"
              shortname "serv"
              bureauURL "http://labelbureau.raters.org/Ratings"
              )
  Policy (RejectUnless "(serv.pics)")
  Policy (AcceptIf "( (serv.pics > 3) and (serv.nudity = 0) )" )
  Policy (RejectIf "otherwise")
))
```

For this example, default distrust is assumed (i.e. `trust (_X, _Y, load(_X, _Y), -100)`).

PICS2: `trust (MyComputer, WebPages, load(MyComputer, WebPages), 100) ←`
`_R = source(Ratings) & present(WebPages, _R, pics) &`
`pictures(WebPages, _R, _X) & (_X > 3) & nudity(WebPages, _R, 0);`

In policy PICS2, we see that `source(Ratings)` is assigned to a variable `_R`. This allows for the inclusion of several label bureaux, as is possible in the PICSRules language.

3.3.3 PolicyMaker

Before stating the basic assumptions that need to be made for modelling to take place, a brief summary of PolicyMaker is given. The PolicyMaker system is a query engine, which evaluates whether a proposed action is consistent with local policy. The inputs to the PolicyMaker interpreter are the local policy, the received credentials and an action string (which specifies the actions that the public key wants to perform). The interpreter's response to the application can either be yes or no or a list of restrictions that would make the action acceptable. A policy is a trust assertion that is made by the local system and is unconditionally trusted by the system. A credential is a signed trust assertion made by other entities and the signatures must be verified before the credentials can be used. Policies and credentials are written in an assertion language. The syntax of an assertion is:

Source ASSERTS *AuthorityStruct* WHERE *Filter*

Source represents the source of the assertion, *AuthorityStruct* represents the public key(s) to whom the assertion is applicable and *Filter* is the predicate that action strings must satisfy for the

assertion to hold. Given this background information, the process of modelling KeyNote assertions in SULTAN may proceed.

In order to facilitate a natural model of the PolicyMaker environment, a distinction must be made between modelling policies and credentials (as defined in the Policymaker system). A PolicyMaker source that is *policy* will be represented as *PolicyMakerSystem* in this discussion. A VerifySignature(Key) function for credentials is also included. Generally, a PolicyMaker assertion can be modelled as:

```
PolicyName: trust(Source, AuthorityStruct, ArbAction, ArbLevel) ← Filter;
```

ArbAction represents an action that is normally linked to *Filter*. ArbLevel is an arbitrary level and will be lost in the translation to PolicyMaker; as the Policymaker and KeyNote systems have no notion of trust levels. For credentials, the constraint that VerifySignature(AuthorityStruct) is added to the constraint of the policy.

Examples:

PolicyMaker:

```
policy
ASSERTS doctor_key
WHERE check_up() <- field(doctor_key) <> "plastic surgery";
```

SULTAN:

```
F1: trust (PolicyMakerSystem, doctor_key, check_up(PolicyMakerSystem, doctor_key), _Arb)
← field(doctor_key) <> "plastic surgery" & _Arb > 0;
```

PolicyMaker:

```
BMA_key
ASSERTS "0x12345abcd"
WHERE field("0x12345abcd") <> "plastic surgery"
```

SULTAN:

```
BMA: trust (BMA_key, KEY, _, _Arb)
← VerifySignature(KEY) & (field(KEY) != "plastic surgery") & _Arb > 0;
```

```
// it is assumed that KEY is a domain with '0x122345abcd' as a member
```

3.3.4 KeyNote

The KeyNote assertion format has the following basic format:

```
KeyNote-Version: VersionNo
Authorizer: Sources
Licensees: Targets
Comment: Comments
Conditions: conds
Signature: sign
```

This assertion can be modelled as the following SULTAN rule:

```
pol: trust (Sources, Targets, _, _) ←
    VerifySignature(sign) & conds;           //Comments
```

As with PolicyMaker, the distinction between policies and credentials is maintained.

Example:

KeyNote:

```
KeyNote-Version: 1
Authorizer: rsa-pkcs-hex:"1023abcd"
Licensees: dsa-hex "986512a1" || rsa-pkcs1-hex:"19abcd02"
Comment: Authorizer delegates read access to either of the Licensees
Conditions: ($file == "etc/passwd" && $access == "read") -> {return "ok"}
Signature: rsa-md5-pkcs1-hex:"f00f5673"
```

SULTAN:

```
KModel: trust (abcd, KEYS, _, _Arb)
    ← VerifySignature(Sig) & _F = file("etc/passwd") & access(_F, "read") & _Arb > 0;
```

It is assumed that *abcd* represents *rsa-pkcs-hex:"1023abcd"* and that *KEYS* represents *dsa-hex "986512a1"* and *rsa-pkcs1-hex:"19abcd02"*.

3.3.5 REFEREE

Profiles-0.92 is the rule-based trust policy language designed to work with REFEREE. The following policy (specified in profiles-0.92) states that all material from Cambridge University and the University of Bath will be blocked, and only material from Imperial College will be automatically downloaded.

```
( threshold-and 2
  (not (url-match URL ("http://www.cam.ac.uk" "http://www.bath.ac.uk")))
  (url-match URL ("http://www.ic.ac.uk"))
  unknown
)
```

In the SULTAN notation, this may be modelled as:

```
REF1: trust (MyComputer, DisAllowedSites, load(MyComputer, WebPages), -100) ;
REF2: trust (MyComputer, AllowedSites, load(MyComputer, WebPages), 100);
```

It is assumed that *DisAllowedSites* will be mapped to *http://www.cam.ac.uk* and *http://www.bath.ac.uk* and that *AllowedSites* will be mapped to *http://www.ic.ac.uk*. It is also assumed that strong distrust (-100) will be mapped into a negative authorisation implementation policy. The next example states that labels from the MIT and CMU bureaus should be used and only pages with labels that state that the document has been thoroughly checked for viruses can be downloaded.

```
( invoke "load-label" STATEMENT-LIST URL
    "http://web.mit.edu/ratings/CodeSafety.html"
    ("http://bureau.mit.edu" "http://bureau.cmu.edu")
  )
( match
  ("load-label")
  (((version "PICS-1.1") *
    (service "http://web.mit.edu/ratings/CodeSafety.html") *
    (ratings (RESTRICT > virus 8))
  )))
STATEMENT-LIST
)
```

In the SULTAN notation, this is modelled as:

```
REF3: trust (MyComputer, WebPages, load(WebPages), 100)
← _CS = source(CodeSafety) & virus(WebPages, _CS, _X) & _X > 8 ;
```

3.4 The Specification Process

The process of trust specification is the sole responsibility of the systems administrator. This is because he has a view of the global organizational picture. The first task that must be done by the administrator is the construction of an organizational diagram. This diagram is used to help in the specification of trust and recommend rules. In this section, an example will be used to illustrate a typical specification process. The example used is Bob's Music Warehouse (adapted from [143]).

Bob is an innovative entrepreneur, who uses the Internet to sell music. His set-up consists of the following elements: a web browser, a client application, a front-end server, a content database and a credit card server. The web browser and client application are run from the users' computer, while the other components are run and maintained by Bob.

Figure 3.2 illustrates the architecture of Bob's Music Warehouse. A user uses the web browser to access the front-end server and buy music. The browser communicates with the front-end server using cryptography. The client application is used to play the user's purchased titles. The content database contains all the titles that can be bought at Bob's site, and this database can be linked with third-party databases if Bob desires.

To prevent the illegal manufacture of copies of purchased music, the purchased titles remain encrypted on the users' computers and only the client application can decrypt and play the

purchased titles. Also, in order to prevent a user from giving encrypted titles to a friend with a copy of the client application, the purchased titles are cryptographically bound to the user.

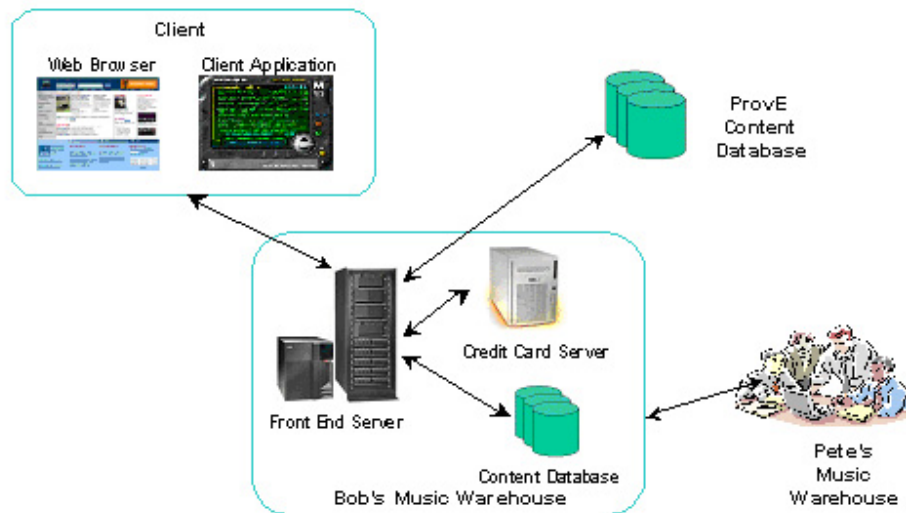


Figure 3.2: Bob's Music Warehouse (BMW)

Bob has established strategic business alliances with Pete's Music Warehouse (PMW) and with ProxE, a provider of music titles. The client applications from BMW and PMW are designed to interact, and Bob uses the content database from ProxE to augment his product base.

3.4.1 Organizational Diagram Construction

From the initial description of BMW, the administrator gets an idea of the entities involved the business and their relationships. This allows him to draw the following organizational diagram for BMW.

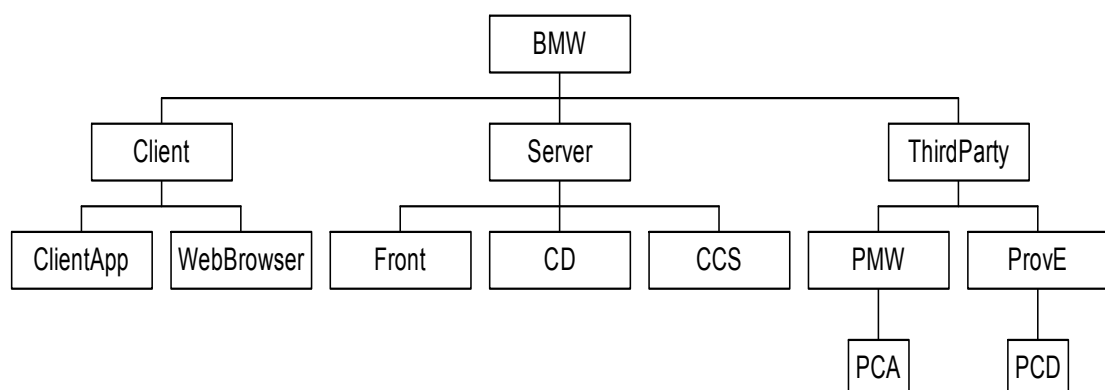


Figure 3.3: Organization Diagram for BMW

The symbols and their representations used in Figure 3.3 are presented in Table 3.1.

Symbol	Meaning
BMW	Bob's Music Warehouse
ClientApp	BMW's client application
Front	BMW's front-end server
WebBrowser	The client's web browser
PCA	PMW's client application
CCS	BMW's Credit Card Server
CD	BMW's content database
ProvE	ProvE
PCD	ProvE's content database

Table 3.1: Key for BMW Organization Chart

The organizational diagram for BMW would be represented by the following statements in the TMF:

isPartOf(ClientApp, Client). isPartOf(WebBroswer, Client). isPartOf(Front, Server).
 isPartOf(CD, Server). isPartOf(CCS, Server). isPartOf(PCA, PMW). isPartOf(PCD, ProvE).
 isPartOf(PMW, ThirdParty). isPartOf(ProvE, ThirdParty). isPartOf(Client, BMW).
 isPartOf(Server, BMW). isPartOf(ThirdParty, BMW).

isPartOf(X,Y) means that X is a part of Y. This composition is strictly done for reasons of functionality and sometimes may not be a realistic view of the scenario. For example, Client is not strictly speaking a part of BMW, but in order to allow for analysis it is useful to make the association. To ensure that entities that are external and not under the control of the administrator are identifiable, the administrator should mark these entities with the foreign tag. Thus, the following facts should also be included:

isPartOf(Client, foreign). isPartOf(ThirdParty, foreign).

Use of this tag may not always be necessary, but the capability to specify default policies for external entities may be useful. All the entity information is stored in the SULTAN Entity-Connections server (explained further in section 7.2.2). Note that it is assumed that names are unique throughout the Entity-Connections Server. As stated above, this server is initially set up by the administrator and thereafter updated by the monitoring system (discussed in Chapter 6). The administrator only needs to update the entity server when re-organization is necessary. The entity information adds to the applicability of the system. Trust statements and recommendations can be specified concerning groups of abstract entities, which themselves may be abstract group representations. Applications using the decision-making facilities of the TMF can be automatically mapped to an abstract group (if so desired). Unknown entities or artefacts can be automatically mapped to default groups. In subsequent chapters, the importance of entity

information is further explained. It should be noted that organizational diagram construction is a one-time task that is performed only when the SULTAN TMF is first being used. Thereafter this task only needs to be performed when re-organization necessitates that it should be re-done.

3.4.2 SULTAN Rule Specification

From the description of BMW given above, the following trust assumptions can be extracted:

- BMW only trusts the client application to decrypt songs.
- The front-end server trusts the client application to play purchased titles.
- The front-end server trusts the web browser to access the music database and to buy music.
- BMW trusts PMW's client application to access its music database.
- The front-end server trusts the credit card server to verify and store credit card information.
- The front-end server trusts the content database to encrypt and to provide titles.
- The front-end server trusts the ProvE content database to provide titles.

For convenience, the following abstractions are made:

Symbol	Meaning
decrypt(Entity, Title, Decrypted)	Decrypts Title to produce Decrypted
encrypt(Entity, Title, Encrypted)	Encrypts Title resulting in Encrypted
play(Entity, Title)	Plays Title
AccessMusic(Entity)	Accesses music database on Entity
BuyMusic(Entity, Title)	Allows Entity to purchase Title
VerifyCreditInfo(Entity, CreditDetails)	Verifies CreditDetails
StoreCreditInfo(Entity, CreditDetails)	Stores CreditDetails in secure form
ProvideMusic(Entity, Titles)	Retrieves music titles.

Table 3.2: Action Abstractions for BMW

Now that the administrator has constructed an organizational chart, has extracted the trust requirements that need to be specified and identified the set of trusted actions, the task before him is to use all this information to define the rules for BMW's domain. These rules may look like the following:

- i1 : **trust**(BMW, ClientApp, decrypt(ClientApp, TitleName, Decrypted), 100);
- i2 : **trust**(BMW, _Y, decrypt(_Y, TitleName, Decrypted), -100);
- ii : **trust**(Front, ClientApp, play(ClientApp, TitleName), 100)
 ← decrypt(ClientApp, TitleName, DecryptedFile);
- iii : **trust**(Front, WebBrowser, AccessMusic(CD): BuyMusic(Front, Title), 100);
- iv : **trust**(BMW, PCA, AccessMusic(CD), 100);
- v : **trust**(Front, CCS, VerifyCreditInfo(CCS, CreditDetails):
 StoreCreditInfo(CCS, CreditDetails), 100);
- vi : **trust**(Front, CD, encrypt(CD, Title, EncryptedFile): ProvideMusic(CD, Title), 100);
- vii : **trust**(Front, ProvE, ProvideMusic(CD, NewTitles), 100);

All specification information is stored in the Specification Server, which will be presented in Chapter 7.

3.5 Summary

This chapter started by introducing the requirements of a trust notation, which are: 1) the ability to specify trust and distrust statements, positive and negative recommendations and conditions on trust and recommendation relationships, 2) access to risk and experience facilities, 3) easy inclusion in an analysis framework, 4) expressiveness, and 5) clear and well-defined semantics. In the discussion of the SULTAN specification notation, it was shown that it fulfills these requirements, with the exception of easy inclusion in an analysis framework. This will be shown in the next chapter. The notation facilitates the encoding of high-level trust requirements. Trust statements, distrust statements, positive recommendations and negative recommendations can all be specified. The SULTAN notation shares a syntactic appearance with logic programming languages, e.g. Prolog. However, no model-theoretic semantics have been defined. Thus, the notation is not claimed to be logic-based, merely logic-oriented. It is essentially a starting point for a range of activities, such as analysis and refinement. After presenting the rules for specifying requirements in the SULTAN notation, a brief discussion on the interactions between trust statements and recommendations was presented. Finally, a small example that highlights the specification process was given.

Chapter 4 Analysing Trust

“Familiar things happen, and mankind does not bother about them. It requires a very unusual mind to undertake the analysis of the obvious.”
- Alfred North Whitehead (1861 - 1947) [144]

Trust specifications contain information that is not always obvious from their specification, which implies that there may be latent implications and associations. It may be possible to have embedded actions being trusted for stakeholders who should not be trusted. Analysis helps to uncover this hidden information and provides people with insight (useful knowledge) that will help them more effectively perform the tasks assigned to them. System administrators, irrespective of their associated activities, normally have a standard set of duties that they must perform. However, these duties must be tailored for the domain they are working in, whether it is banking, manufacturing, retail, law, etc. For the analysis of trust requirements, it is necessary to define the knowledge that would be useful to the administrator. In this chapter, the formulation of analysis questions using the SULTAN Analysis Model (SAM) is discussed. A template of generic analysis queries is also presented.

4.1 Requirements for Analysis

Trust analysis requires the following:

- A notation that allows the construction of analysis questions.
- The ability to specify standard analysis questions, such as an implied dependency, conflict of interest, separation of duties conflict, etc.
- The ability to specify application-specific analysis questions.
- An associated set of specifications, written in a notation that can be transformed into an information database that can be analysed.
- The ability to perform program reasoning on the associated set of specifications.
- The ability to reason about the current state of the relationships in the associated set of specifications.

4.2 How to analyse in the SULTAN TMF

The SULTAN specification notation (discussed in Chapter 3) is the associated specification language. In order to facilitate analysis, the specification notation is translated to Prolog (the translation algorithm is presented in Appendix C). Prolog is chosen because it offers a powerful, logic-based framework, with clear semantics, that can be easily adapted to the field of trust analysis. Thus, the analysis notation is written in and utilizes Prolog.

In general, trust analysis is the process of reasoning about the source and or the state of a set of trust relationships, which are specified in a suitable notation (Figure 4.1).

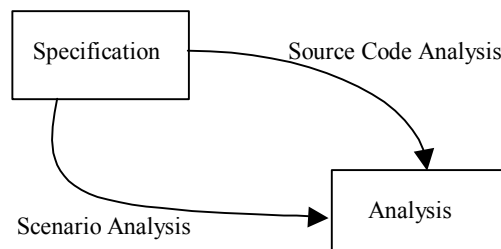


Figure 4.1: The link between Specification and Analysis in the SAM

The SULTAN Analysis Model (SAM) facilitates both simulation and property analysis. In this chapter, the focus will be on how property analysis is performed in the SAM. Chapter 8 will discuss how simulation analysis is performed using the SAM. Property analysis involves checking whether specified properties hold on trust and recommendation rules and is concerned with the discovery of conflicts and redundancies. The properties can be with respect to the specification source, which is essentially program reasoning. Source analysis ignores the constraints, i.e. assumes they are true. The properties can also be with respect to examining trust relationships to identify scenarios of interest, which involves reasoning about the state of the system, and the current state of constraints. When reasoning about scenarios, the issue of detecting cycles and the issue of the constraints that are still to be satisfied for a trust relationship to be valid need to be addressed. All these topics are dealt with in more detail later in this chapter.

A conflict arises as a result of two assertions (trust or recommend) of different polarities (positive and negative), on the same actions and referring to the same subject and target. A redundancy (or ambiguity) is defined as the state where two assertions, of the same type (trust

or recommend), have the same subject, target, actions and levels and where the assertions are of the same polarity, but possess different values.

The model is intentionally general, in order to allow different organizations the diversity and flexibility that they require in defining their analysis requirements. As stated earlier, the implementation of the prototype for the model in Prolog facilitates the formulation of both application-specific queries and general queries.

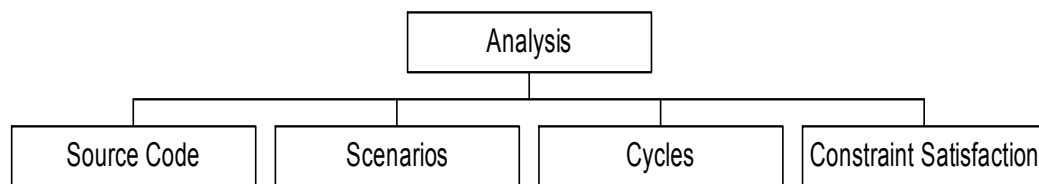


Figure 4.2: Analysis Types

Figure 4.2 illustrates the four components of analysis in the SAM. The fourth facility provided by the SAM is the constraint satisfaction facility. This allows the administrator to identify the constraints that must be true in order for a particular rule to be true. This topic will be presented later in this chapter. The SAM contains a pre-defined set of predicates that allows the types of queries identified in Figure 4.2 to be constructed. The primary predicate used is the **query** predicate, which is polymorphic and each form represents a different type of analysis question. Appendix D contains the complete definition of the SAM.

4.2.1 Analysis on the specification source

Source analysis involves reasoning about a specification (i.e. program reasoning) and ignores the constraints, i.e. assumes they are true. For this reasoning, only the head of the specifications is considered; the constraints are incidental. Essentially this is used to search the trust specification database to determine if specific trust or recommend rules exist (or not) or to see what rules apply to specific entities. Given the following SULTAN rule:

```

TCPA: trust (TPM, _App, provideUpdate(_App, _Machine), 70)
← isOn(TPM, _Machine) & verified(TPM, _App);
  
```

Analysis of the source would not try to determine the values of `isOn(TPM, _Machine)` and `verified(TPM, _App)`. These conditions would automatically be assumed true. To specify a source-based analysis query, a **query** predicate of the following form is used:

query(V, D, R).

The semantic interpretation of the predicate in the form above is: ‘What R (which is a collection of V) satisfies the situation described by D?’ V is the set of variables that should be in the answer to the query, i.e. the variables of interest. D is the description of the situation of interest. R is the result set. The following are the rules for constructing V, D and R:

qvariable	=	UpperCaseLetter (LowerCaseLetter UpperCaseletter)*;
spredicate	=	A SULTAN source predicate
prolog	=	A Prolog statement
V	=	{‘[‘ qvariable (‘,’ qvariable)* ‘]’};
D	=	{‘(‘ (spredicate prolog) (‘,’ ’;’) (spredicate prolog))* ‘)’};
R	=	qvariable;

UpperCaseLetter represents an upper case letter, LowerCaseLetter represents a lower case letter, prolog is a Prolog statement (involving variables introduced in the query) and spredicate is a taken from the following list of predicates used to identify particular components of trust and recommend rules:

- p_policy(P) – P is a SULTAN policy/rule.
- p_rec_pol(P) - P is a recommend rule.
- p_trust_pol(P) - P is a trust rule.
- p_pos_trust(P) - P is a positive trust rule.
- p_neg_trust(P) - P is a negative trust rule.
- p_pos_rec(P) - P is a positive recommend rule.
- p_neg_rec(P) - P is a negative recommend rule.
- p_entity(E, P) - E is the entity in rule P.
- p_subject(E,P) - E is the subject of rule P.
- p_target(E,P) - E is the target of rule P.
- p_trustor(E, P) - E is the trustor of rule P.
- p_trustee(E, P) - E is the trustee of rule P.
- p_recommendor(E, P) - E is the recommendor of rule P.
- p_recommendeed(E, P) - E is the recommendeed of rule P.
- p_level(L, P) - L is the level associated with rule P.
- p_constraints(C, P) - C is the set of constraints associated with rule P.
- p_actionset(A, P) - A is the actionset associated with rule P.
- p_actions(A, P) - A is the actionset associated with rule P.
- p_commonAS(P, Q) – P and Q have a common actionset.
- p_commonAS(P, Q, A) – P and Q have a common actionset A.
- p_trustedby(E, N, L, A, e) - Entity E is trusted by exactly N other entities at level L to perform action(s) A.
- p_trustedby(E, N, L, A, a) - Entity E is trusted by at least N other entities at level L to perform action(s) A.
- p_constraints(X, Y, C) - C is the set of constraints for the rules that relate entities X and Y.

Examples of source analysis queries that can be formulated are:

- query([E], (p_trustee(E, P), p_trustor(microsoft, P)), Result).
What entities, E, satisfy the following: E is the trustee of a policy, which has microsoft as trustor? (i.e. which entities are trusted by microsoft?)

- `query([X, Y], (p_pos_trust(X), p_neg_trust(Y), p_trustor(sun, X), p_trustee(dse, X), p_actions(ACT, X), p_trustor(sun, Y), p_trustee(dse, Y), p_actions(ACT, Y)), Result).`

What rules, X and Y, satisfy the following: X is a positive trust rule between sun (trustor) and dse (trustee) with respect to actionset ACT and Y is a negative trust rule between sun (trustor) and dse (trustee) for that same actionset? (i.e. which conflicting policies represent trust and distrust relationships between sun and dse for the same actionset?)

- `query([PR, D], (p_pos_rec(PR), p_neg_trust(D), p_subject(Rr,PR), p_subject(Rr, D), p_target(Re,PR), p_target(Re, D), p_actionset(ACT, PR), p_actionset(ACT, D)), Result).`

What positive recommendation and a distrust rule, PR and D, satisfy the following: PR and D have the same specified subject, target and actionsets? (This detects conflicts when a recommendation and distrust rule relate to the same subject, target and actionset.)

4.2.2 Analysis about a scenario

Scenario analysis involves reasoning about the state of the system, and the current state of the constraints. This form of analysis requires that both the Specification and State Information Databases be examined to determine the current state of constraints of specified relationships. To perform a scenario analysis, a query predicate with the same form as the one used for source analysis is used.

query(V, D, R).

The difference in the definition of the two queries lies in the construction of D. Syntactically, the only difference is that the source analysis predicates have the letters ‘p_’ as their prefix. Thus, ‘trustee’ is the scenario analysis equivalent of the source analysis predicate ‘p_trustee’. The close relation of the description predicates given here and the ones given in the section on source analysis is intentional; to make it easier for people to learn the predicates and their purposes. Given the following SULTAN recommend rule:

```
Veri: recommend ( Verisign, _KeyHolder, loadScript(_X), -50)
← isCustomer(Verisign, _KeyHolder) & isUsedBy(Verisign, _X) &
  outStandingBalance(Verisign, _KeyHolder) > 40;
```

and the following query:

```
query([R], (neg_rec(R), subject(Verisign, R), actions(loadScript(_),R)), Answer);
```

The Analysis Model will check the State Information Database to determine the current values of `isCustomer(Verisign, _KeyHolder)`, `isUsed(Verisign, _X)` and `outStandingBalance(Verisign, _KeyHolder)` before including rule Veri as an answer. Note that the values are automatically updated by the monitoring system, which will be discussed in Chapter 5. The following are examples of scenario-based analysis:

- query([P, Q], (trustee(E, P), trustee(E, Q), P \neq Q, trustor(sun, P), trustor(sun, Q)), Result).
 What rules, P and Q, satisfy the following: P and Q have the same trustees, their trustor is sun and they have different names?
- query([E], (recommendee(E, P), recommendor(tpm, P), actions(load_script(_, _), Y)), Result).
 What entities, E, satisfy the following: E is recommended by tpm to perform load_script ?
- query([T1, T2], (pos_trust(T1), pos_trust(T2), T1 \neq T2, trustee(Te, T1), trustee(Te, T2), actions(ActionSet, T1), actions(ActionSet, T2)), Result).
 What rules, T1 and T2, satisfy the following: T1 and T2 are different positive trust rules with the same trustee and actionset? (This could be one way of specifying a conflict of interest, where a conflict of interest occurs when one entity is trusted by two (other) competing entities)
- query([T1, T2], (pos_trust(T1), pos_trust(T2), T1 \neq T2, trustee(manager, T1), trustee(manager, T2), actions([sign(_, _)], T1), actions([authorize(_, _)], T2)), Result).
 What rules, T1 and T2, satisfy the following: T1 and T2 are different trust rules with the trustee being manager and T1 containing the sign action in its actionset and T2 containing the authorize action in its actionset? (This expresses a standard separation of duties conflict, where the manager should not be trusted to perform both sign and authorize)
- query([A, C], (constraints(A, B), constraints(B, C), pos_trust(A), pos_trust(B), pos_trust(C)), Result).
 What rules, A and C, satisfy the following: A is the constraint for B, B is the constraint for C and A, B and C are positive trust rules? (This expresses a standard dependency analysis query)

To demonstrate the difference between source and scenario analysis, the specifications from Bob's Music Warehouse (from the Chapter 3) will be used. Their Prolog-equivalent representation is assumed to be:

```
trust(_X, clientapp, [decrypt(clientapp, encrypted, titlename)], 100, d1).
trust(_X, _Y, [decrypt(_Y, encrypted, titlename)], -100, d2).
trust(bmw, clientapp, [play(clientapp, titlename)], 100, p1) :-
    decrypt(clientapp, encrypted, titlename).
trust(provE, front, [encrypt(front, titlename, encrypted),
    send(front, encrypted, provE)], 100, pe1).
trust(clientapp, pca, [send(pca, information, clientapp)], 100, l1).
trust(pca, clientapp, [send(clientapp, information, pca)], 100, l2).
```

The following two analysis queries ask a similar question. However, one is a source query and the other is a scenario query.

```
query([X], (p_pos_trust(X), p_subject(bmw, X), p_target(clientapp, X)), Answer).
query([X], (pos_trust(X), subject(bmw, X), target(clientapp, X)), Answer).
```

The first query (the source-based one) will return the rule *p1* as the answer because it assumes all constraints are true, i.e. it reasons about the possibility or the potential of a rule being true.

The second query would check the State Information Database to determine the value of `decrypt(clientapp, encrypted, titlename)`. If the value is true, then the query will return the rule *p1*, else it will return an empty list.

4.2.3 Detecting cycles

When the truth or falsity of a constraint must be evaluated, the possibility may arise, when executing a query, that the processing mechanism may never return an answer, because of an analysis loop. This occurs when the value of a previously encountered constraint is required to evaluate the current (set of) constraint(s). Thus, there is a circular pattern in the evaluation sequence, i.e. a cycle has been encountered. The code below illustrates a simple case where a specification might contain cycles.

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

JenTom : trust (Jenny, Tom, ProvideInfo(Jenny), MediumTrust )
← recommend (UKRealEstateAssoc, Tom, GiveEstateAdviceProvideInfo(Tom), HighRec );

TomReal : trust (Tom, Realtor, send_deals(Realtor, _X), MediumTrust )
← trust (_X, Realtor, send_deals(Realtor, _X), HighTrust );
```

Using the above specifications, if the question *query([P], (trustor(Jenny, P), trustee(Realtor, P)), Result)* is asked, it would be necessary to evaluate the constraints of rule JenReal, which leads to rule TomReal, which leads back to JenReal. In this case, the query would never be answered (if cycles were not detected and resolved). The SAM ensures that before performing a scenario analysis query, cycles are detected in order to ensure that the query will always return an answer. If a cycle is detected in the course of performing a scenario analysis query, then the administrator is told that a cycle exists and where to look. The method of cycle resolution is left to him. However, a simple cycle resolution strategy is provided in the SAM. This strategy involves renaming the constraint that completes the cycle and setting its value to false. After using the cycle resolution strategy provided, the specification above would become:

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

JenTom : trust (Jenny, Tom, ProvideInfo(Jenny), MediumTrust )
← recommend (UKRealEstateAssoc, Tom, GiveEstateAdviceProvideInfo(Tom), HighRec );

TomReal : trust (Tom, Realtor, send_deals(Realtor, _X), MediumTrust )
← rule_JenReal;
```

This renaming process allows scenario-based analysis to proceed. The constraint is returned to its original name when the analysis tool is closed and at the system administrator's request. The SAM provides the **query**(cycle, R) predicate to allow the administrator to manually detect cycles in the specification. R contains a pair of policy names: the first being the start of the cycle, and the other being the policy that completes the cycle. The resolution strategy employed by the SAM can be employed by using **query**(make_acyclic).

4.2.4 Identifying constraints to be satisfied

When performing scenario analysis, there will be some constraints that evaluate to false, which may occur when either they are absent from the State Information Database or they have explicitly stated false values. In both cases, these constraints are the ones that need to be satisfied for the trust relationship to be valid. In the context of trust establishment, these could be interpreted as the credentials that need to be presented or the tasks that need to be done by the trustee. In the context of proving the trustworthiness of a trustee, these constraints could direct the sequence and focus of credential discovery. Thus, enabling the discovery of these missing or unsatisfied constraints may be valuable in trust negotiation and or in credential discovery. The SAM provides for the formulation of these constraint satisfaction queries using abduction.

Abduction is normally seen as the problem of finding a set of hypotheses (i.e. an explanation or a plan), which when added to a formal specification, allows a goal to be inferred, without causing contradictions. Formally stated, given a specification D and a goal G , abduction attempts to identify a set of assertions, Δ , such that $(D \cup \Delta) \models G$ (i.e. $(D \cup \Delta)$ semantically entails G) and $(D \cup \Delta)$ is consistent. The set Δ consists of only abducible statements, i.e. base assertions. It is important to note that abducible statements are normally domain-specific and are required to be minimal. In the case of constraint satisfaction analysis, D is an arbitrary SULTAN specification for the organization and G is the trust or recommend statement that is being queried. All predicates that are not trust or recommend statements are assumed to be abducible. Thus, Δ is the set of constraints that would make $(D \cup \Delta) \models G$. The predicate that allows this sort of query to be constructed has the following form:

query(A, R).

A can be either a trust statement or a recommend statement, while R is the list of constraints that need to be satisfied for the statement to be true. The following represents a simplified model where a constraint satisfaction query may be used:

```
BankMachine: trust ( natWest, _Client, useNatWestABM(_Client, _SwitchCard), 100)
← isClient(natWest, _Client) & isValidCard(natwest, _SwitchCard) ;
```

Suppose the monitoring system is given the information that `isValidCard(natWest, _SwitchCard)` is true (probably via information gathered from the security application when the Switch Card is passed through the scanner at the ABM's door). If the question **query**(`trust(natwest, X, useNatWestABM(X,Y), 100, _)`, `R`) is asked, then the SULTAN Analysis system would say that `R = [isClient(natWest, X)]`. This may be interpreted as representing that it needs to be determined if `X` is a client of `natWest`. Given a complex web of specifications, the algorithm traverses through the list of constraints to find all the constraints that require information from the monitoring service. A scenario that illustrates the practical use of this type of analysis query is given in Chapter 9.

4.3 Generic Analysis Queries

Although, the analysis model allows the construction of application-specific queries, there are queries that cover all application domains. These are generic conflicts and ambiguities. Appendix E contains the complete template of generic queries that are included as a part of the SAM. In this section, one generic conflict, namely the trust-recommend conflict, and a generic redundancy, namely the recommend redundancy, will be discussed.

4.3.1 Trust-Recommend Conflict

A trust-recommend conflict is, strictly speaking, a conflict between a trust statement and a recommendation. There are two situations that may lead to such a conflict, namely: 1) when there is a positive trust statement and negative recommend statement in conflict, and 2) when there is a distrust statement and a positive recommend statement in conflict. When the conflict concerns a trust statement and negative recommendation, it can be stated as:

```
/* SOURCE */
query( [T, NR], ( p_pos_trust(T), p_neg_rec(NR), p_subject(Rr,T), p_subject(Rr, NR),
                 p_target(Re,T), p_target(Re, NR), p_actionset(ACTT, T),
                 p_actionset(ACTNR, NR), intersect(ACTT, ACTNR, ACTR), not_empty(ACTR) ),
        Result).

/* SCENARIO */
query( [T, NR], ( pos_trust(T), neg_rec(NR), subject(Rr,T), subject(Rr, NR), target(Re,T),
                 target(Re, NR), actionset(ACTT, T), actionset(ACTNR, NR),
                 intersect(ACTT, ACTNR, ACTR), not_empty(ACTR) ),
        Result).
```

The scenario-based query asks ‘What rules, T and NR, satisfy the following : T is a positive trust statement, NR is a negative recommend statement, both T and NR have the same trustor and trustee and both have a common set of actions?’ The predicates `intersect` and `not_empty` are defined in the SAM. They are used to ensure that common actions are identified, in spite of the ordering and grouping of the actions in the actionset of the SULTAN specifications.

A conflict involving a positive recommendation and a distrust statement can be identified by using the following:

```
/* SOURCE */
query( [PR, D], ( p_pos_rec(PR), p_neg_trust(D), p_subject(Rr,PR), p_subject(Rr, D),
                 p_target(Re,PR), p_target(Re, D), p_actionset(ACTPR, PR),
                 p_actionset(ACTD, D), intersect(ACTPR, ACTD, ACTR), not_empty(ACTR) ),
        Result).

/* SCENARIO */
query( [PR, D], ( pos_rec(PR), neg_trust(D), subject(Rr,PR), subject(Rr, D), target(Re,PR),
                 target(Re, D), actionset(ACTPR, PR), actionset(ACTD, D),
                 intersect(ACTPR, ACTD, ACTR), not_empty(ACTR) ),
        Result).
```

The interpretation of these queries is similar to the one given for a trust statement and negative recommendation.

4.3.2 Recommend Redundancy

A recommendation redundancy occurs when two recommend statements exist about the same trustor, trustee and actions and where the levels are of the same polarity but different. Although the inclusion of redundancies may be legitimate and intended, it is important to be cognizant of their presence. Detecting the presence of a recommend redundancy between two positive recommendations can be done using the following:

```
/* SOURCE */
query( [R1,R2], ( p_pos_rec(R1), p_pos_rec(R2), R1 \== R2, p_subject(Rr,R1), p_subject(Rr, R2),
                 p_target(Re,R1), p_target(Re, R2), p_actionset(ACT1, R1),
                 p_actionset(ACT2, R2), intersect(ACT1, ACT2, ACTR), not_empty(ACTR),
                 p_level(L1, R1), p_level(L2, R2), L1 \== L2 ),
        Result).

/* SCENARIO */
query( [R1,R2], ( pos_rec(R1), pos_rec(R2), R1 \== R2, subject(Tr,R1), subject(Tr, R2),
                 target(Te,R1), target(Te, R2), actionset(ACT1, R1), actionset(ACT2, R2),
                 intersect(ACT1, ACT2, ACTR), not_empty(ACTR),
                 level(L1, R1), level(L2, R2), L1 \== L2 ),
        Result).
```

The queries above ask ‘What rules, R1 and R2, satisfy the following: R1 and R2 are different positive recommend rules with the same subject and target, with a common set of actions and with levels with different values?’

Discovering a recommend redundancy between two negative recommendations can be done using:

```

/* SOURCE */
query( [R1,R2], ( p_neg_rec(R1), p_neg_rec(R2), R1 \== R2, p_subject(Rr,R1),
                p_subject(Rr, R2), p_target(Re,R1), p_target(Re, R2), p_actionset(ACT1, R1),
                p_actionset(ACT2, R2), intersect(ACT1, ACT2, ACTR), not_empty(ACTR),
                p_level(L1, R1), p_level(L2, R2), L1 \= L2 ),
        Result).

/* SCENARIO */
query( [R1,R2], ( neg_rec(R1), neg_rec(R2), R1 \== R2, subject(Tr,R1), subject(Tr, R2),
                target(Te,R1), target(Te, R2), actionset(ACT, R1), actionset(ACT, R2),
                level(L1, R1), level(L2, R2), L1 \= L2 ),
        Result).

```

All the generic conflicts and redundancies highlighted in this section (as well as the ones in Appendix E) have been identified based on generalized questions that firms may need answered. However, it is possible to ignore these conflicts if they are deemed irrelevant.

4.4 Summary

This chapter started with a discussion of the requirements of analysis. These requirements are: 1) a notation that allows the construction of analysis questions, 2) the ability to specify standard analysis questions, 3) the ability to specify application-specific analysis questions, 4) an associated set of specifications, written in a notation that can be transformed into an information database that can be analysed, 5) the ability to perform program reasoning on the associated set of specifications, and 6) the ability to reason about the current state of the relationships in the associated set of specifications. The SULTAN specification language is the notation used to represent the trust relationships analysed by the SAM. Though the SULTAN notation cannot be analysed directly, it is easily translated to Prolog to make use of a standard logic based analysis framework. The discussion of the predicates of SULTAN Analysis Model (SAM) illustrate the types of analysis questions that can be asked and also demonstrates that the SAM satisfies the requirements outlined above. The SAM facilitates both simulation and property analysis, but in this chapter we focused only on property analysis. Properties can be with respect to the specification source, which is essentially program reasoning, or with respect to scenarios, which is reasoning about the state of trust relationships. It is also possible to ask questions about the

presence of cycles and about constraints that need to be satisfied. The chapter ended with a presentation of some of the generic analysis queries included in the SAM.

Chapter 5 Risk in Trust Management

“Risk taking is an integral part of progress, and failure a key part of learning.” [145]

For interacting parties in a distributed computing platform, there is always an element of risk involved. Generally, risk is defined as the exposure to uncertainty with a known probability distribution of events. For the context of Internet applications, risk is the probability of a failure with respect to the context of the interaction, e.g. non-payment for service, service failure, etc. The connection between risk and trust is sparsely researched. From general observation, risk and trust are inter-related in the following ways:

- Risk may be used to determine the level of trust.
- Trust may be used to determine the riskiness of a venture.

In both cases, there seems to be an inverse relationship between the two concepts. A transaction that is viewed as being ‘not so risky’ is normally assigned a higher level of trust, while a highly-trusted transaction is considered to be of low risk. The precise nature of the relationship between these concepts (whether linear or exponential or other) is still an open research topic.

For Internet applications, it is possible to use the concepts of trust and risk independently in determining whether or not a transaction should be initiated. For example, a high-risk transaction regarding a low-valued product may still be embarked upon because the transaction may be deemed trustworthy enough for it to be undertaken. This illustrates that scenarios exist where both trust and risk may be used independently, with one concept having more influence on the decision than the other. However, it is normally the case that risk influences trust and vice versa. This is the premise used in this thesis.

There are numerous attempts at risk modelling in computer science. Current research is grounded in decision theory, which is about a decision maker facing several choices and choosing a consequence (or an outcome) based on some strategy (e.g., Maximin, Minimin, Maximax, etc.). Decision theory is subdivided into: 1) decision under certainty, 2) decisions under risk, and 3) decisions under uncertainty. For Internet applications, decisions under risk and decisions under uncertainty are of primary concern. In the context of decisions under risk, each choice will have one of several possible consequences, and the probability of occurrence for each consequence is known. Thus, each alternative is associated with a probability distribution, and a choice among probability distributions. When the probability distributions

are unknown, the context is now decisions under uncertainty. An implicit assumption of decision theory models is that the returns or losses accrue only to the decision maker. For example, if the decision is to carry an umbrella or not, the return (I get wet or not) depends on the state of nature. However, nature is not concerned with the outcome. This assumption may not be appropriate for trust decision making by Internet applications because both parties in a given interaction have a vested interest in the decisions taken by each other with respect to the transaction. This is always the case when the parties wish to engage in future transactions.

The focus of this chapter is the description of the risk model used in the SULTAN TMF. This discussion details the solutions provided to the risk assessment/calculation and risk provision problems. Before presenting the SULTAN risk assessment solution, the current approaches to building risk assessment models is given.

5.1 Risk Models

The standard approach to risk modelling is to adhere to a set of well-defined steps that allow the application of statistical methods to the risk data in order to do some useful task (normally decision-making or analysis). There are two traditional risk model approaches: the quantitative model and the qualitative model. In computing, there is also the de facto standard software development risk model, which is a hybrid of the two traditional models. Each of these models is briefly presented.

5.1.1 Quantitative Model

A quantitative risk model (also called an EL-based model) has two essential elements: the probability of an event occurring, $p(E)$, and the likely loss should the event occur, $L(E)$. These two numbers are used to determine the EL – Expected Loss, which is also called the Annual Loss Expectancy (ALE) or the Estimated Annual Cost (EAC). The formula used is:

$$EL = p(E) * L(E)$$

E is the event of interest and EL represents a quantification of the financial impact of the risk of E. A lower EL implies a lower risk. Thus, given a set of events, their probabilities of occurrence and their expected losses, it is possible to create a hierarchy of events, from most risky to least risky event. Figure 5.1 shows a typical scenario where ELs may be used.

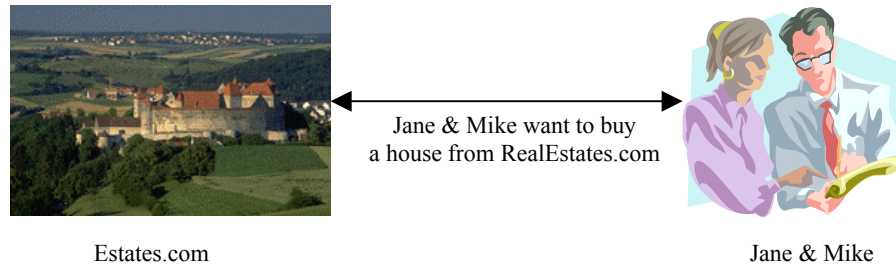


Figure 5.1: A typical risky transaction

Jane & Mike wish to buy a house from Estates.com. For simplicity, it is assumed that there are only two outcomes pertinent to Jane & Mike, namely: the theft of their credit card information and the refusal of the company to honour the transaction. Table 5.1 shows the risk metrics that are defined for the couple.

Event	Loss (in £)	Probability of Event Occurring
Theft of Credit Card Details	100	0.6
Dishonouring Transaction	200	0.1

Table 5.1: Risk Metrics for Jane & Mike

From Table 5.1, the Expected Loss for the transaction will be 80 pounds (the sum of the EL for each event). If this amount is negligible to the couple, then they should initiate the transaction. The couple may place greater importance on the theft of their credit card details, which would imply that they should use the expected loss for that risk as their decision variable. It is important to state that risk may be measured in terms of a gain, rather than a loss. Estimated gain is defined by the following formula:

$$EG = p(E) * G(E)$$

EG is the expected gain and G(E) is the estimated gain (or profit) from event E.

5.1.2 Qualitative Model

In a qualitative risk model, probability is not considered; only estimated potential loss is used. The basic elements of a qualitative model are: threats, vulnerabilities and controls. A threat is something that can go wrong or that can attack the system. A vulnerability makes the system more prone to attack by a threat or makes an attack more likely to be successful and have an impact. A control is a countermeasure for a vulnerability. There are four types of controls:

- *Deterrent controls* reduce the likelihood of a deliberate attack,
- *Preventative controls* protect vulnerabilities and make an attack unsuccessful or reduce its impact,

- *Corrective controls* reduce the impact of an attack, and
- *Detective controls* discover attacks and trigger preventative or corrective measures.

Figure 5.2 (adapted from [146]) shows the interaction between the components of a qualitative risk model.

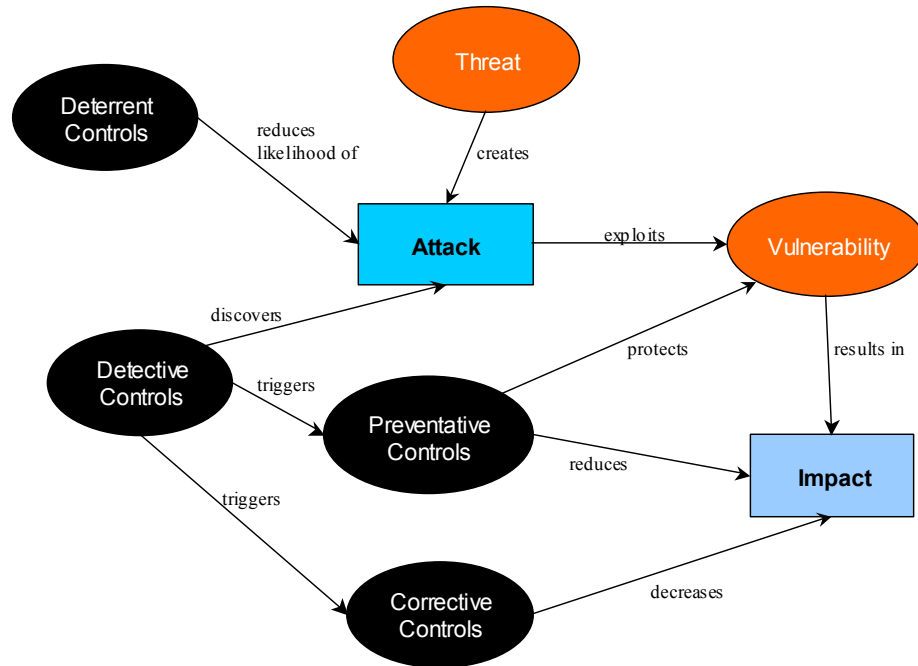


Figure 5.2: Interaction of the Components of a Quantitative Risk Model

5.1.3 Software Development Risk Model

For software risk management, risk is defined as “*exposure to harm or loss*” [147]. On a given project, a software developer’s definition of a risk is refined from the above definition. A developer’s view and identification of a risk is normally a function of the software’s application domain, the development platform and the developer’s experience and knowledge base. Thus, views of ‘what is a risk?’ may vary from developer to developer, and from project to project. The risk management process is normally incorporated into the software development model that is being used, e.g. the waterfall model, the rapid prototyping model, the spiral model, etc. The risk management model that will be presented here is generic and stereotypical of risk management models worldwide [148, 149].

The steps involved in the software risk management model are:

1. *Identify Risk*

All the risks to the software project must be identified. This can be done by using the Taxonomy-Based Risk Identification Questionnaire produced by the SEI (Software Engineering Institute at Carnegie Mellon University) [145]. After identification, the risks are analysed.

2. *Analyse Risk*

In this step, the risks are quantified. This means assigning probabilities to the risks and estimating the impact to the project of these risks occurring. Once this is done, overall risk values are calculated. The above step corresponds to a traditional quantitative risk model. However, due to the imprecise nature of the data, qualitative labels are normally used for both the probabilities and impact estimates. For example, using the scheme outlined in [150], risk probabilities may be: very low, low, medium, high or very high and the impacts of risk may be: negligible, marginal, critical or catastrophic. The calculation of the overall risk value is done by evaluating the impact/probability matrix, which is constructed from the assignments. After analysis, a list of highest risks can be identified. These risks should be the first to be planned for.

3. *Plan for Risk*

Risk planning means formulating methods to address each risk. In planning, the following should be covered: why is the risk important? What is needed to track the risk? Who is responsible for the risk management activity? What resources are needed to perform the activity? A detailed plan of how the risk will be prevented and or corrected needs to be formulated. This includes an action plan – to resolve an immediate risk, and a contingency plan – to monitor the risk and trigger a predetermined response.

4. *Tracking Risks*

Tracking risks ensures that if triggers are activated the entire development team is made aware and plans are put into action. This is useful because past knowledge of risk may improve current and future projects.

5. *Control the Risk Management Process*

A process needs to be in place at the project's start to identify, analyse and track risks. If this process is not adequate, and risks are getting uncontrollable, then it should be re-

formulated. This step is normally considered at the start of a software development process. However, steps 1 to 4 are executed sequentially.

5.2 The Problems with the Risk Models

There are four main problems with quantitative risk models. The first is that data is often unreliable and inaccurate, because it may be extremely hard to consistently assign correct values to $p(E)$ and $L(E)$. The second problem is that incorrect assignment of probabilities in the real world makes them imprecise and their use (or over-use) can promote complacency and a false sense of security. The third problem is that they depend on information that is normally extremely sparse. EL-based models assume that frequency, valuation and efficacy data are always available. Usually, much of this data is largely unavailable. Though there are mechanisms for the valuation of information and the determination of consequences in other research fields, from the survey performed there are no such mechanisms available in the computing field. The fourth problem is that this model does not consider interdependencies between events.

The qualitative model suffers from three similar problems. The first is that the potential loss that can be incurred by a particular attack is often difficult to determine in real world scenarios. Without this, it is impossible to construct a hierarchy of risky attacks. The second is that this model tends to favour a significantly greater detail than is normally efficiently feasible to describe [151]. Take the example of applying a qualitative risk model implementation to Microsoft Windows 95. There are thousands (probably hundreds of thousands) of vulnerabilities. The number of threats (and attacks) will be directly proportional to the number of vulnerabilities. Thus, a proportional number of controls will be needed to counter these attacks. The task of providing the controls for all these attacks is daunting at the least. The third problem is that implementations of these types of risk models do not scale well. This is a direct consequence of the models requiring large volumes of data to be effective. This problem is usually true of EL-based implementations.

It is argued that the fact that current risk models are completely deterministic and variables are assessed as single-point estimates rather than probabilistic ranges of values (probability distribution functions) is a major flaw in the design of these models. This is a reasonable criticism. However, the inclusion of probability distribution functions (pdfs) would compound the scalability issue. In addition, this quest to incorporate uncertainty into risk models requires

that these pdfs be mapped to a single value in order to facilitate ease of use. Though a range of EL values may be useful in certain situations, for applicability a range is normally represented by a single-point estimate (e.g. a mean and or standard deviation).

The software risk management model has the same problems as both the quantitative and qualitative risk models, but to a lower extent. This is because decision under risk theory can be easily adapted for use in software development. The majority of (if not, all) the risks faced by the software developer can be easily identified and prioritised and the assumption that the returns or losses accrue to the decision maker holds for this domain. The key lesson learnt from all the risk models that have been designed and implemented is that every risk model will suffer from the issues mentioned above. The task before the designer of a new risk model is to identify and incorporate techniques that reduce the magnitude and effect of the potential problems, which is often referred to as risk mitigation.

5.3 SULTAN Risk Model

The SULTAN Risk Model is a hybrid risk model, i.e. it incorporates elements of the qualitative and quantitative risk models. The functions of the SULTAN Risk Model are to assess risk for a particular transaction and to retrieve previously stored risk information, which is gathered by the monitoring system and stored in a repository. The component of the SULTAN TMF that encapsulates the Risk Model's functionality is called the SULTAN Risk Service (SRS). The decision to make the Risk Model a hybrid model is based on the fact that a hybrid model could be used to reduce the magnitude of the problems discussed in the previous section. A discussion on how this is done will be presented later. Given the fact that the Risk Model follows from traditional approaches to risk modelling, there are a number of issues that need to be addressed, namely: determining the risks, determining the potential losses, handling dependency and determining the risk profiles. Figure 5.3 presents an overview of the risk assessment in trust management problem and its solution. The root node of Figure 5.3 describes the problem that this chapter is addressing. The root node leads to the four sub-problems that must be solved. Each sub-problem is linked to the solution employed and all four solutions are connected through the SULTAN Risk Model.

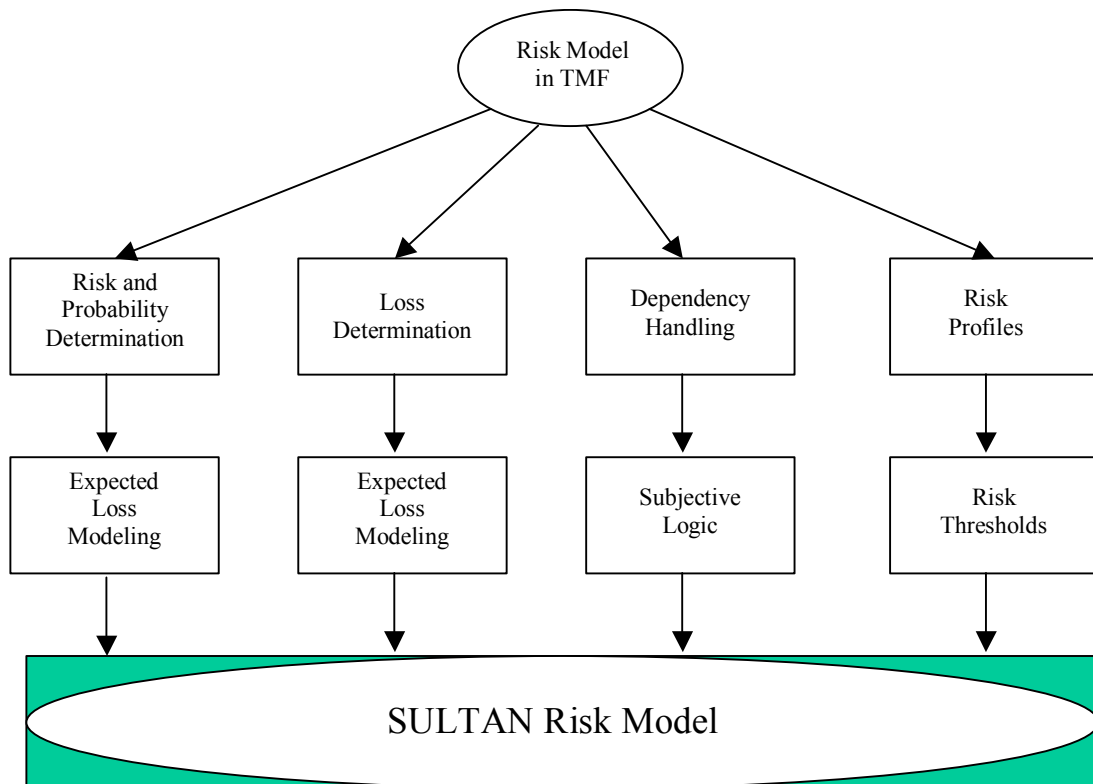


Figure 5.3: The ‘Risk Assessment in Trust Management’ Issues

5.3.1 Determining Risks and their Probability of Occurrences

There are a large number of risks that can be encountered in an Internet transaction. The risks can relate to the transaction protocol, the storage media and mechanism, the software used and standard transaction risks, such as refusal of payment. A system that models all these risks would be prohibitively large and would not scale well. To reduce the information demand, the SULTAN Risk Model does not require an explicit listing of risks, but instead uses a list of ten common categories of risks that may occur. These risk categories are: receipt of malicious code, refusal to produce goods, service failure, theft of information, fraud, transaction error, denial of service, non payment of service, illegal transaction, and security failure. Each risk is given a unique risk id and an initial probability of occurrence, p , and a measure of confidence (i.e. uncertainty) in our estimate of the probability, u , are assigned. Note that the sum of p and u equal to 1. This is consistent with Dempster-Shafer belief theory [152, 153]. All this information is stored in a risk-likelihood repository in the SULTAN TMF. The general categories of risk, their initial probabilities and the uncertainty measures are derived from a collection of articles on E-Commerce risks [154-157]. Note that the category listing of risks may be extended, modified or reduced by the system administrator.

5.3.2 Determining Potential Losses

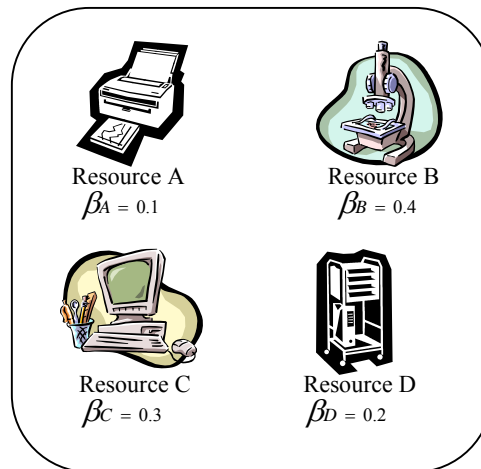
All daily business transactions have a loss value associated with them. This value is normally a function of the cost of the items being purchased and any legal (and or insurance-related) remuneration agreed upon by the stakeholders. For the buyer on the Internet, the transaction loss value is, in most cases, equal to the cost of the item being purchased. Currently, legal and other considerations are not included because these concerns have not been globally addressed in the Internet Commerce framework. Calls to the SRS normally contain the transaction cost as a parameter. However, when an Internet transaction is primarily concerned with the use of the trustor's resources, then the value of the item equals the value of the resource. The value of the resource will not normally be known by the application calling the SRS. Thus, the SRS may be called with a parameter representing the resource in question. The Risk Service calculates its value by querying the trustor's asset repository, which stores the total value of all the resources and the contribution of each resource to the overall portfolio. The formula used in the calculation of the value of the resource is:

$$R_i = \beta_i (R_T)$$

R_i is the value of resource i , β_i represents the contribution of resource i to the overall portfolio of resources and R_T is the total value of all the resources. It should be noted that:

$$\sum_{i=1}^N \beta_i = 1.$$

The formula for the calculation of R_i is derived from the Expected Loss formula discussed earlier in the chapter.



Total Value of Resources = £500

Figure 5.4: Stereotypical Example of Resource Value Calculation

Figure 5.4 shows a stereotypical example domain with 4 resources. From Figure 5.4, the value of resource B is £200 (£500 * 0.4). The initialisation of the asset repository is a one-time task that is performed by the system administrator, with the help of tools from the SULTAN TMF. This process and the asset update service are described in Chapter 7.

5.3.3 Handling Dependencies

In handling dependencies, two issues must be resolved. Firstly, the detection of the dependencies. Secondly, the calculation of the probability of occurrence taking the dependencies into account. The SRS addresses the first issue by encoding the action dependencies in the dependency repository. Note that the dependency repository is checked for cycles before being used. If cycles are discovered then the repository must be modified until it is acyclic. The (initial) population of the repository is done through the monitoring system, which is discussed in Chapter 6. The second issue requires a more involved solution. Since traditional EL-based risk models do not have any facility for handling the re-calculation of the probability of an action that is dependent upon another, a possible solution would be to assume that dependent actions could be treated as conditional events and to apply Bayes' Theorem of conditional probability evaluation:

$$p(A | B) = \frac{p(B | A) * p(A)}{p(B)}$$

A is an action that is dependent on action B. In using Bayes' Theorem, a request for the base actions, $p(A)$ and $p(B)$, and the conditional event $p(B|A)$ is made. In the Internet environment, this data is often unavailable. The problem escalates if action A depends on multiple actions. The number of probabilities required makes the application of Bayes' Theorem infeasible. To handle the action dependency problem, the SULTAN dependency probability algorithm (dpa) is employed. The dpa makes use of Subjective Logic [19, 23, 95-100], an uncertainty trust model that is a generalized model of binary logic and probability calculus. Subjective Logic is used because it is consistent with Shaferian belief theory, but has a consensus operator that provides more realistic answers [101]. The dpa is outlined below:

- 1 Determine the set of dependent actions, S.
- 2 Let n be the cardinality of S.
- 3 Create and initialise a structure, T, of n Subjective Logic opinions.
- 4 Set an action counter variable, x, to 0.
- 5 For each dependent action, d, in the set S, do the following:

- 5.1 Set a probability value variable, v to 0.
- 5.2 If d is dependent, then perform the dpa on the actions that d depends on and store the value to v .
- 5.3 If d is not dependent, then retrieve the probability from the risk-likelihood repository and store it to v .
- 5.4 Set the opinion at position x to a new opinion created from value v .
- 6 Find the consensus of the opinions in T .
- 7 Convert the consensus to its probability expectation value, p .
- 8 Return p .

Subjective Logic opinions are used because an easy and consistent mapping may be performed from the opinion space to the probability space (and vice versa) and the consensus operator offers a novel way to represent (and the calculate) the overall notion of the agreement of many different viewpoints.

5.3.4 Determining Risk Profiles

The subjectivity of loss and risk thresholds is an issue that is not covered by standard risk models, but that must be modelled in the SULTAN system. Depending on the trustee and the trusted action, the trustor may have a different maximum allowable loss and risk threshold. For example, if Microsoft is to be trusted to automatically fetch and install Windows updates on my behalf then the loss I am willing to incur may not be the same if Microsoft is to be trusted to store and manage my personal and financial details. The same is true for the level of risk that will be allowed for each context. People may have different contextual propensities to risk (i.e. they may be risk-averse or risk-loving) and this means that they will have different risk allowances. For this reason, a trustor-risk repository in the SULTAN TMF contains information on the subject, action, maximum allowable loss (MAL) and risk threshold (RT). RT is a value, between 0 and 1, above which an entity deems a transaction too risky to engage in. MAL is the maximum loss that an entity wishes to incur for a particular action(s). Initially, the administrator sets up the risk profiles for the entities identified in the entity-connections database. Updating risk profiles is done by the monitoring system. It should be noted that risk profile creation is a one-time task performed only when the SULTAN TMF is first being used. The update of the risk profile data is discussed in Chapter 6, while the initialisation is presented in Chapter 7.

5.3.5 Calculating Risk

Figure 5.5 shows a generalized risk calculation process. In calculating the risk involved in a transaction, the SRS is provided with the subject, target, action(s), risk id and the transaction cost (or asset name). The SRS searches the trustor-risk repository to retrieve the Maximum Allowable Loss (*MAL*) and Risk Threshold (*RT*) information. If the information is not found, then a NO TARGET-ACTION error is returned to the calling application. Otherwise, the probability of the risk occurring (*p*) is searched for and retrieved from the risk-likelihood repository. If the risk id cannot be found, then an INVALID RISK ID error is returned to the calling application. The action is then checked for dependencies (by checking the dependency repository). If dependencies are present, then the dpa is used to determine the new value of *p*. In the flowchart below (Figure 5.5), it is assumed that only one action is specified in the call from the application.

If there is more than one action, the process of checking for dependencies and finding the new *p* is carried out for each action and then these values are converted to opinions, their consensus found and the result is converted back to a probability. After the checking of the action(s), it is checked whether an asset or transaction cost is specified. If an asset is specified then the asset repository is used to determine the loss *L*, else *L* is set to the transaction cost. The Expected Loss, *EL*, is calculated by multiplying *L* and *p*. If $EL < MAL$, then the risk value is $((EL/MAL) * 100)$ else it is 100.

As stated previously, the issues of massive information demands, large storage requirements and lack of scalability are faced by all risk models. In an effort, to reduce the effect of these issues, general categories have been used in the repositories, which reduces the storage needs and information demands, and an alternative probability calculation mechanism presented for dependent actions, which will again lower the information needs. The issue of scalability will be addressed further in Chapter 7.

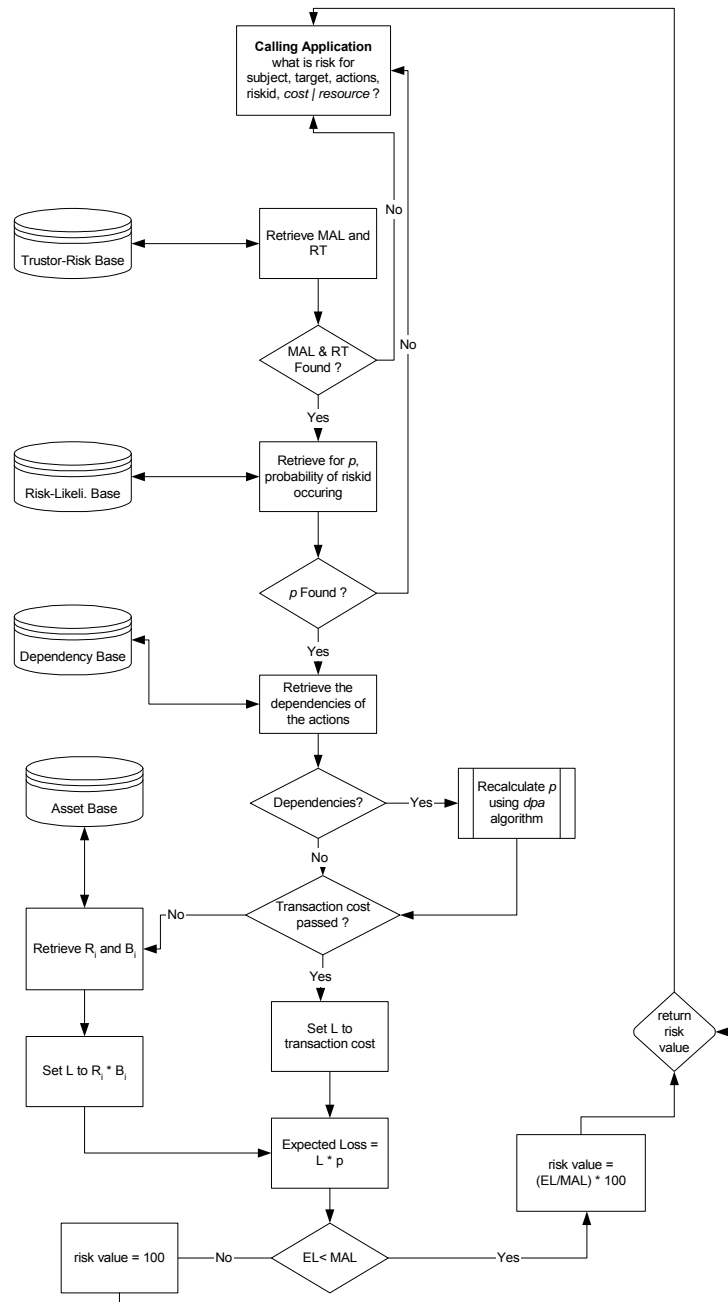


Figure 5.5: Risk Calculation in the SRS

5.3.6 Retrieving Risk Information

The second function of the SRS is to provide a risk information retrieval service (Figure 5.6). As stated in Chapter 3, risk information is gathered by the monitoring system and stored in the State Information Database. The request for risk information is made through the SULTAN Consulting Service, which is described in Chapter 7. On receiving the request, the SRS queries

the State Information Database and always returns an estimate of the risk. This estimate may not always be available from information present in the State Information Database.

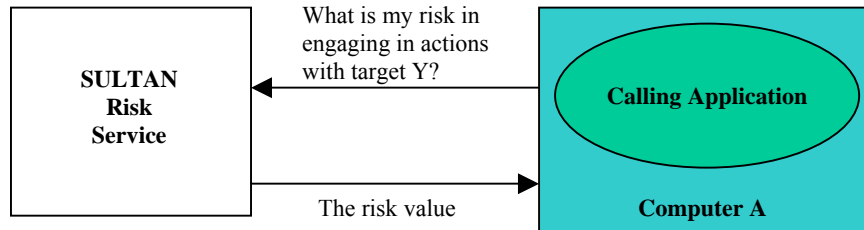


Figure 5.6: SRS RISK Information Retrieval

The following algorithm shows the steps taken by the SRS in retrieving risk information:

1. Search the State Information Server for the information.
2. If the risk information is present for the subject, target and action(s) then simply return the risk value retrieved.
3. If there is risk information present for the target and the actionset, then calculate the weighted average risk value for the target-action pair information present and return this value.
4. If there is no risk information present, then risk calculation is performed.

Steps 3 and 4 return a flag to the calling application that indicates that the information provided is not retrieved from information present in the state database. Risk information is stored using the following format:

(**risk**, subject, target, actionset, riskvalue)

riskvalue is the value of the risk that *subject* undertakes when *target* performs *actionset*, where *riskvalue* is an integer between 0 and 100. The rules for the construction of the elements of the above tuple are:

subject = reference ;
target = reference;

Chapter 3 provides the definition of actionset. Questions posed by user applications are constructed in a similar format. The syntax and operation of the SULTAN Consultant Service calls is provided in Chapter 7.

5.4 Summary

Risk modelling is a difficult task that requires the balancing of large data requirements and applicability. Traditional models utilize very specific, low-level definitions, which reduced their usefulness and limited their commercial applicability. In this chapter, the traditional

approaches to risk model design were reviewed. The problems with these models were highlighted and the fact that every risk model that is designed faces similar problems. The lesson learnt from the problems of traditional risk models is that the potential problems must be managed. The SULTAN Risk Model seeks to reduce the informational demands and storage requirements normally expected from other risk models. The issues of the determination of risks and losses, the treatment of dependencies and the inclusion of risk allowances for a typical domain using the SULTAN TMF were discussed. The primary functions of the SULTAN Risk Model were also stated. These functions: 1) to calculate risk for a transaction, and 2) to retrieve previously stored risk information.

Chapter 6 Experience, Monitoring and Re-evaluation

*“A trusted application is one that can easily render your system insecure.”
- Adapted from the Orange Book [82]*

This chapter is about trust evolution, which involves changing the terms of a relationship based on new evidence. Relationship evolution is necessary because trust is a dynamic concept. A trust relationship does not stay static in the face of new evidence. New data, along with the accumulated older facts, may constitute the experience with respect to the interactions relating to the trust relationship(s). Monitoring is the process of acquiring information about interactions. Thus, monitoring allows experience information to be gathered. To ensure that the relationships actually change as new information is presented, it is necessary for the relationships to be re-evaluated, i.e. the constraints of the relationships must be re-examined.

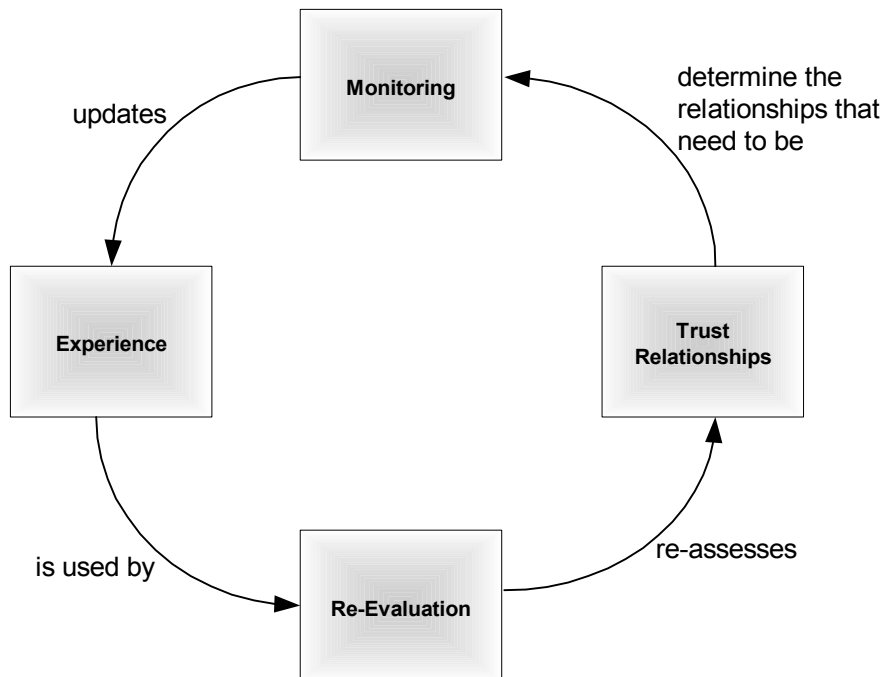


Figure 6.1: Experience, monitoring and re-evaluation

Thus, the concepts of experience, monitoring and re-evaluation are related, and are necessary to model trust relationship evolution. This connection is illustrated in Figure 6.1.

6.1 Experience

In this context, experience is the knowledge acquired as a result of observations of the outcomes of the interactions. A consumer may have an initial level of trust in a company that may change over time as the consumer's experience with the company increases. Thus, experience may help to determine the nature of the trust relationship. As stated in Chapter 2, there seems to be a direct relationship between experience and trust. The better the experience with a firm, the higher the level of trust that may be placed in them. When attempting to model experience in a TMF, the issues that must be resolved are: 1) the representation of the observations of the interaction outcomes, 2) the acquisition of these observations, and 3) the usage strategies for these observations. The collection of experience information is done via the monitoring system, which is discussed later in this chapter.

6.1.1 Experience Representation

To record an experience, the following information is required: the experience observer, the entity that was observed, the observed action(s) and the measure of the experience, i.e. the experience value. An experience record is encoded using the following tuple:

(experience, subject, target, actionset, expvalue)

Subject is the observer, target is the observed entity, actionset is the observed action(s) and expvalue is the experience value, which is an integer between -100 and 100 (0 exclusive). Negative values representing a negative experience and positive values representing positive experiences. The rules for the construction of the elements of an experience tuple are similar to the rules for a risk tuple (given in Chapter 5). Note that all experience information is time-stamped before being stored in the State Information Database.

6.1.2 Usage Strategies

The combined body of experience information relating to a particular subject, target and actionset may increase to become a large (and seemingly contradictory) set of facts. For example, in 1998 Linda had a positive experience with the update service provided by Microsoft. A month later, she gets infected with a virus caused by using Microsoft's update service. These facts may lead to the inclusion of the following facts in the State Information Database:

(**experience**, Linda, Microsoft, update_service(Linda, _X), 50)

(**experience**, Linda, Microsoft, update_service(Linda, _X):run_activeX(Linda,_Y), -100)

When Linda (or any other member of the organisation) wishes to use experience information on Microsoft's update service for a trust decision, the presence of multiple facts means that a usage strategy needs to be employed. A usage strategy is a mechanism that combines a related set of statements into one representative statement. Classically, there are four experience usage strategies: optimistic, pessimistic, cautious and most-recently-used.

Optimistic Strategy

An optimistic perspective assumes that the best experience will always be used. Thus, given a set of experience records, the optimistic strategy returns the maximum experience value. For example, Linda would use experience value 50.

Pessimistic Strategy

A pessimistic strategy assumes the worse case scenario, i.e. the minimum experience value of the set of records is returned. Using the example above, Linda would be given the experience value -100.

Cautious Strategy

A cautious strategy uses a weighted average to get a balanced estimate of all the information present. For the two records presented above, Linda would be given an experience value of -25, ie. $(-100+50)/2$.

Most-Recently-Used Strategy

Using this strategy, Linda would use the experience information relating to her specified context with the most recent time-stamp. In essence, she assumes only short-term memory.

Regular bill payments (by credit card or direct debit) establish a series of experience records accumulated by the bill collector, which may be used to determine if the bill payer may be allowed to access other services provided by the payee. For example, The Royal Bank of Scotland (RBS) may use account activity (i.e. regular deposits and expenses) to determine if a client may be trusted with a RBS credit card. This is an application of the cautious strategy. From the perspective of the client, consistent behaviour delivering products and services generates experience records, which may be used to determine the nature of future interactions with a company. The consumer's strategy is totally dependent on his personal philosophy towards producer evaluation. For example, he may view one bad transaction (in a small set of

high-valued transactions) to be enough information to decide against using that producer again (pessimistic strategy). Another consumer may look at a measure of the entire history with the producer and use this as a basis for a decision (cautious strategy). In the SULTAN TMF, when the experience information is required for analysis or an experience question is asked by a user application, a cautious strategy is employed.

6.2 Monitoring

Trust Monitoring involves the update or addition of information. Figure 6.2 shows the basic outline of a generalised Trust Monitor.

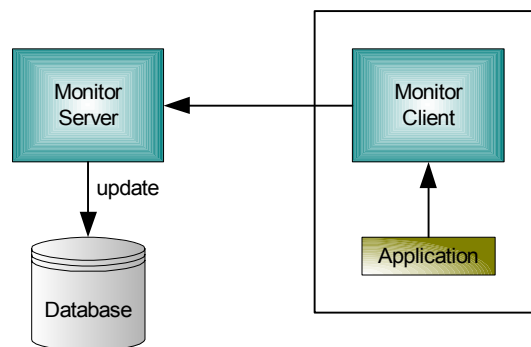


Figure 6.2: A Generalised Trust Monitor

Figure 6.2 illustrates that for trust monitoring there needs to be a monitor client on the machine of each user in the organization, which interfaces with user applications and sends information to the monitor server. When a monitor client is initially installed on a computer, how does the client determine what information is to be monitored? There are two approaches to solving this problem, namely: using active-design architecture or using passive-design architecture.

6.2.1 Active Design Architecture

In active-design architecture, the monitor client actively polls the application for monitoring information. To reduce the overhead involved in polling applications, the system administrator provides a mapping of the constraints to the programs that may generate them. This mapping is included in the monitor client and the client checks constantly if any of these programs are running. If they are, then the application is polled for a fixed time after a socket connection is detected. This is to determine the data to be monitored. The advantage of this approach is that monitoring is more or less automatic. The disadvantage is that at times the data to be monitored may not be reliably and correctly identified. This drawback may be mitigated by the

administrator providing a very detailed and precise mapping of the constraints to the programs that may create them and the circumstances under which they may be created.

6.2.2 Passive Design Architecture

In a passive design architecture, the work of configuring the application to send information to the monitor client is the responsibility of the administrator. The administrator installs the client and uses his knowledge of both the specification and application to configure the application to send the right fact at the right time. The advantage of this approach is that any fact in the database from the monitor will be useful. The disadvantage is that the system administrator is expected to know and understand the programming language and operation of the application. A good rule of thumb that could be used to lessen the job of the system administrator is to use a small, rich set of constraints. This will help in the configuration of the user application(s).

6.2.3 The SULTAN Monitor Architecture

The SULTAN Monitor (SM) utilizes a passive design architecture to gather the information used by the TMF. The duties of the SM are:

- To update the entity connections repository
- To keep state, risk and experience information current
- To update the action dependency database
- To update the risk profiles.

Figure 6.3 shows the basic interaction of the SULTAN Monitor client and server. A user application makes a call to the SM Client, which sends this information on to the SM Server. Note that the SM Client first either generates or looks up the unique id for the application and then sends it with the request to SM Server, which authenticates the source and then performs the required update function on the appropriate repository. The SM Client contains a local list, which contains a unique id for the computer and pairs of application ids and application names. Figure 6.4 illustrates the method used to generate the unique computer id (UCI) for a computer that has a new instance of the SM Client installed.

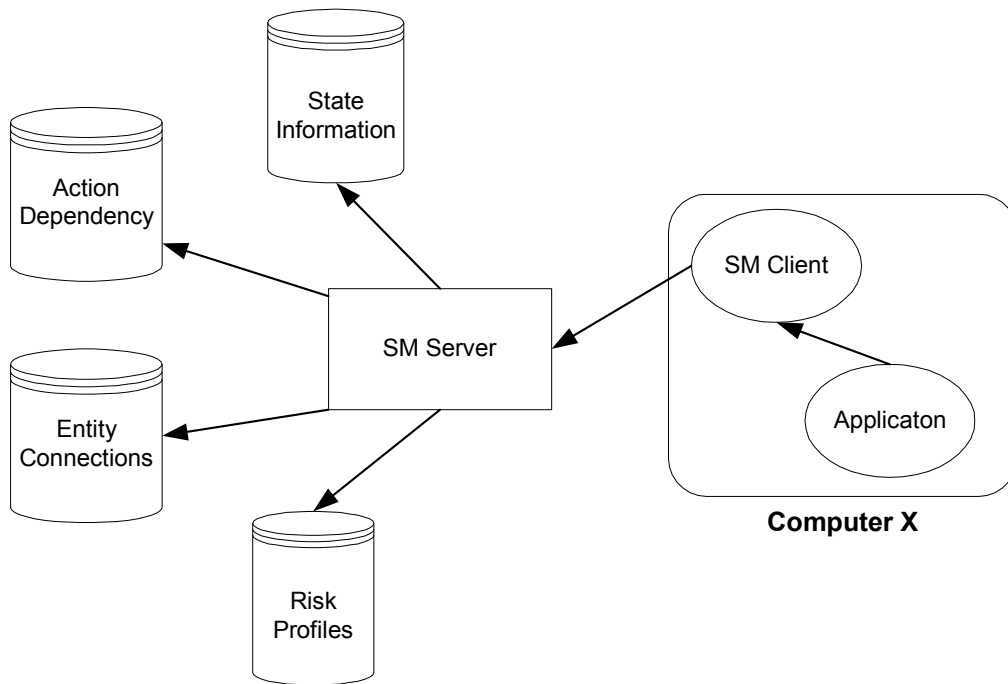


Figure 6.3: Overview of SULTAN Monitoring

The computer characteristic used for the current version of the TMF is the computer's IP address. A hash function, known only to the SM Client and the SM Server, is used to map the characteristic to a UCI, which is sent to the Server, where it is placed in a job queue for the system administrator. The administrator has to associate it to the organizational hierarchy using the `isPartOf` rule.

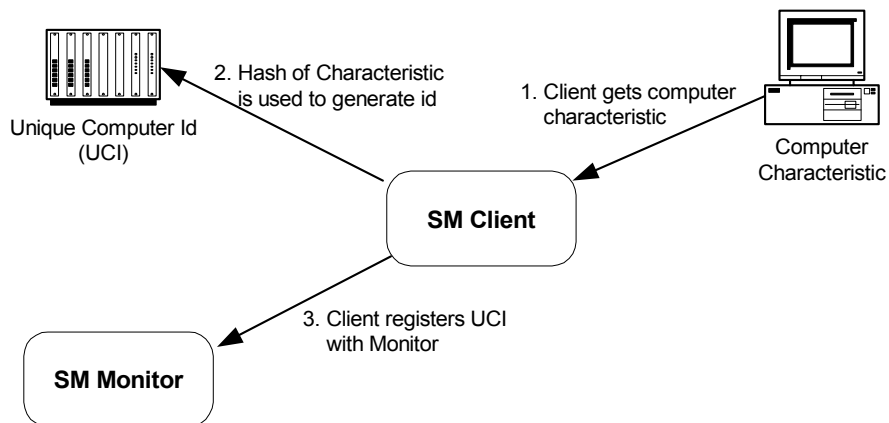


Figure 6.4: Computer Id Generation

When an application uses the SM Client, the following process is followed:

- The application makes a call to the TMF through the SM Client.
- The SM Client checks its list of id-name tuples.
- If the application name is not in the list.

- The SM Client uses the UCI to generate an application id for the application.
- The id-name tuple is stored locally.
- The command `isPartOf(app-id, UCI)` is issued to the SM Server.
- The application id is sent with the request.

For example, given a SM Client with the following local store:

```
Comp1500169
Comp1500169_1 winlogon.exe
Comp1500169_2 snmp.exe
Comp1500169_3 svchost.exe
```

If the application `csrss.exe` wishes to use the monitoring system, then it would be allocated the id `Comp1500169_4` and associated with the computer `Comp1500169`. This association is done to model the distinction between the behaviour of application on computer A from the behaviour of the same application on computer B.

6.2.4 Updating entity connections

Chapter 1 highlighted the fact that the firm's organizational chart has to be initially constructed by the system administrator. This organizational information identifies the basic entity names and is encoded in a series of `isPartOf` facts in the Entity-Connections Database. This Entity-Connections Database may be modified directly only by the administrator. However, as new computers and applications are added to the domain, there is a need to record it. A request of the following format is used:

(isPartOf, app-id, UCI) – app-id is a part of UCI

This is translated to the `isPartOf` facts discussed in Chapter 3.

6.2.5 Updating risk, experience and state information

Risk and experience information are recorded using the following tuples:

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

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

Note that the subject is the application sending the request, which will be identified by an application id. The application sends one of the above tuples to the SM Client, which determines the name of the calling application, and translates the tuple to:

(risk, app-id, target, actionset, riskvalue)

(experience, app-id, target, actionset, expvalue)

It is assumed that target is a foreign entity, which implies that an `isPartOf(target, foreign)` tuple will have to be sent by the SM Client if the target name is not found in the Client's local store. Before being stored in the State Information Database, both risk and experience information are time-stamped. This implies that a usage strategy must be employed when a representative record is necessary for a set of related records. The cautious strategy is used when such a record is required for both risk and experience. The history-based approach to the storage of risk and experience records is used to give a bigger picture of the relationship evolution, i.e. to provide a method of simulating memory in the TMF. This particular approach may lead to demanding storage requirements, but partitioning and distribution may lessen these requirements.

State information is information concerning the constraints used in the rules in the Specification Database, e.g. the current value of a variable. State information is sent to the SM Client via the a record of the following format:

(attribute, value) – provides state information

State information is also time-stamped. However, unlike risk and experience information, state information about a previous stored attribute is not kept in the State Information Database. Currently, it is deleted and the new state record is added. However, old data could be easily archived if it is deemed useable.

6.2.6 Updating action dependency information

Action dependencies are necessary to build a model of the tasks performed by an application. For example, the chat program may make calls to the `test_sound_card` and `start_network_connection` external functions, which implies that the chat program is dependent on both those functions. For the production of risk values, it is necessary to be cognizant of these dependencies. An application sends dependency information using the following:

(**depends**, action1, action2) – for action1 depends on action2

This is converted to a dependency record of the form:

(depends, subject, action1, action2) – for subject, action1 depends on action2

This information is not time-stamped before storage.

6.2.7 Updating risk profiles

A risk profile represents an entity's subjective risk propensity. A risk profile is defined on a 'per-entity' basis and states the entity's limits with respect to targets performing action(s). An application makes a risk profile update by specifying a tuple of the following format:

(**rprofile**, actions, MAL, RT) – MAL and RT are the values for subject and actions

This is converted to tuples of the form:

(**rprofile**, UCI, actions, MAL, RT) – MAL and RT are the values for subject and actions

MAL is the maximum allowable loss and RT is the risk threshold. It is assumed that MAL is in a universally accepted currency. Currency conversions would require a conversion module, which must be first integrated in the Internet infrastructure.

An added feature of the SM Server is that it may be configured to delete particular facts, when the SM client gives it a signal. This feature may be used to remove session-specific facts and stem the growth rate of the databases.

6.3 Re-evaluation

Re-evaluation is the process of re-examining the trust relationships based on new information gathered by the monitoring system. The monitoring system updates the information in the State Information Database, which is used by the analysis module in scenario-based analysis queries (Figure 6.5).

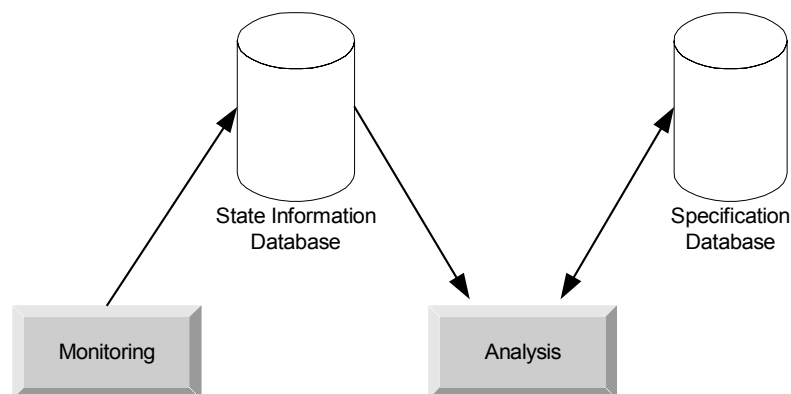


Figure 6.5: Monitoring and Analysis

After state information is added, the SM Server performs an analysis against the template of conflicts and ambiguities supplied with the SULTAN TMF. If a potential conflict is detected, a flag is set to alert the administrator. This is then added to his tasks to be done. Once the real problems are identified, the system uses the specifications to determine if the current trust relationships require adjustment, i.e. whether to continue the relationship (give another chance), discontinue the relationship, or discontinue the relationship and seek punitive actions against the betrayer. To illustrate a very simple (and abstract) scenario where re-evaluation may be used, a small example is used. The Specification Database has the following rules:

```
PDA: trust ( Morris, Symantec, definition_update(Morris, Computer), 100 )
← DefinitionState(Symantec) = "old";
MVer: recommend ( Morris, _KeyHolder, loadScript(_X), 50)
← trust+(Verisign, KeyHolder, _X);
```

Initially, the State Information Database contains no information pertinent to the above specifications. If the analysis question query([X], (trustee(X,P), trustor(Morris,P), actions(definition_update(_,_), P)), Answer), i.e. which entities do Morris trust to perform definition_update?, is asked, an answer of none will be returned because there is no attribute-value pair for DefinitionState(Symantec). Four days later, Morris executes the Norton Antivirus program to perform his weekly virus-scan. The program checks the date of the virus definition file and established that the file is old. It then sends the tuple (DefinitionState(Symantec), "old") to the SM Client and requests that Morris updates the definitions. Morris decides to use the TMF to help in the decision, he does this by using the consulting service (discussed in Chapter 7). Essentially a question similar to the analysis query presented above is asked. The system returns the fact that Symantec may be trusted.

6.4 Summary

In this chapter, the connection between experience, monitoring and re-evaluation was initially explained. Experience is the knowledge acquired as a result of observations of the outcomes of the interactions. Monitoring is the process of acquiring information about interactions. Re-evaluation is the process of re-examining the trust relationships based on new information gathered by the monitoring system. The representation of experience was highlighted and the four usage strategies presented, namely: optimistic, pessimistic, cautious and most-recently-used. The SULTAN Monitoring systems updates the state, risk and experience information, as

well as updating the Entity-Connections Database, the Action Dependency Database and the risk profiles. The chapter ended with an example that highlighted the re-evaluation process.

Chapter 7 SULTAN Trust Management

“The outcome, in the real world, of software system operation is inherently uncertain with the precise area of uncertainty also not knowable.”
- Lehman [158]

Trust management, as defined in Chapter 1, is defined as:

“the activity of collecting, encoding, 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.” [4, 5]

To enable the collection, encoding, analysis and presentation of evidence, tools are included in the SULTAN TMF [2] that support the following processes:

- **Trust specification** - the process of defining trust relationships in terms of the parties involved, and the context of the interaction. This is done using the Specification Editor.
- **Trust analysis** – the process of examining a set of trust relationship specifications to identify unwanted implicit relationships and possible conflicts of relationships. The Analysis Tool facilitates this process.
- **Trust monitoring** – the process of updating experience, risk and state information. This allows for the re-evaluation of the trust specifications based on this evidence, i.e. experience from interactions, new risk evaluation methods or changes in an entity’s credit ratings. This process is accomplished using the Trust Monitor.
- **Risk evaluation** – the process of estimating the risk involved in a transaction based on collected information or context-sensitive risk metrics. This is performed by the Risk Service.
- **Trust consultation** – the process of providing trust information to a user to enable more informed trust decision-making. The Trust Consultant is the SULTAN TMF Component that enables this functionality.

Figure 7.1 shows the current toolset and how these tools may be used by an organization. A process not mentioned above that is partially facilitated by the tools in the SULTAN TMF is Trust Establishment, which is the process of defining the protocols by which parties, wishing to

form a trust relationship, can negotiate and exchange evidence and credentials. The Establishment process involves discovering the credentials that need to be presented and transporting these credentials between the parties involved. The process of securely exchanging the (possibly sensitive) credentials between entities is not currently facilitated in the SULTAN toolset.

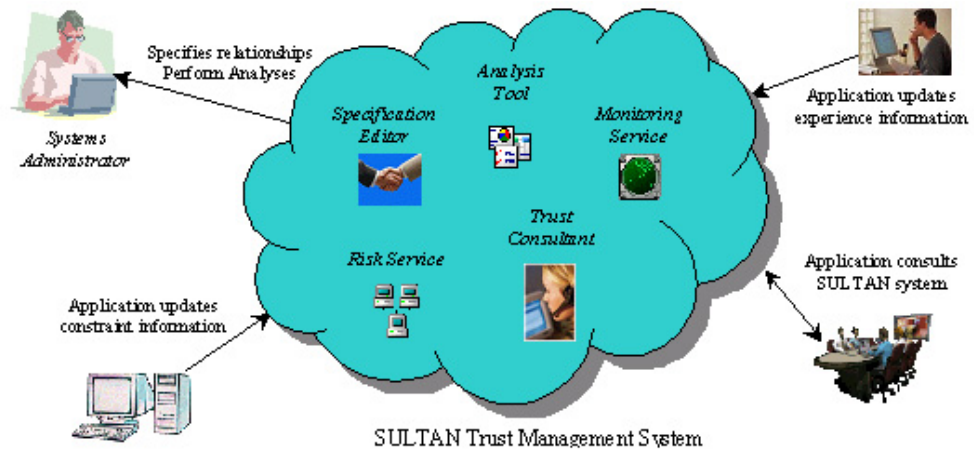


Figure 7.1: SULTAN Tools and their interactions to the external system

However, the SULTAN system can be used to identify the artifacts needed for a successful establishment negotiation. The constraints of a SULTAN specification may represent the credentials that need to be present for the initiation of a trust relationship. By performing a constraint satisfaction query (discussed in Chapter 4), the credentials required can be ascertained.

7.1 Trust Management Life Cycle

The SULTAN trust management life cycle (Figure 7.2) highlights the activities involved in setting up and maintaining a system that coordinates trust relationship information. The Setup Phase corresponds to the period when the system administrator is installing the SULTAN TMF. During this phase, she must perform a set of initialisation tasks, which are: 1) initialising the asset repository, 2) constructing the organizational chart, and 3) initialising the risk profile data. These tasks are supported through tools in the Specification Editor, which is discussed later in the chapter. After setup, the administrator specifies the trust relationships and performs some analysis (whether from the template provided by the SULTAN system or queries she has constructed). The bi-directional arrow between the specification/modification box in Figure 7.2 (module 1) represents data flow between the Specification Editor and the repositories of the

TMF. Analysis may lead to the specifications being revised to eliminate an unwanted property. It should be noted that modules 0, 1 and 2 are activities that can only be directly performed by the administrator and modules 3 and 4 are done by the computers in the domain. These computers are assumed to be proxies for the human operators and the applications they use. Information on system state, risk and experience are collected from these computers.

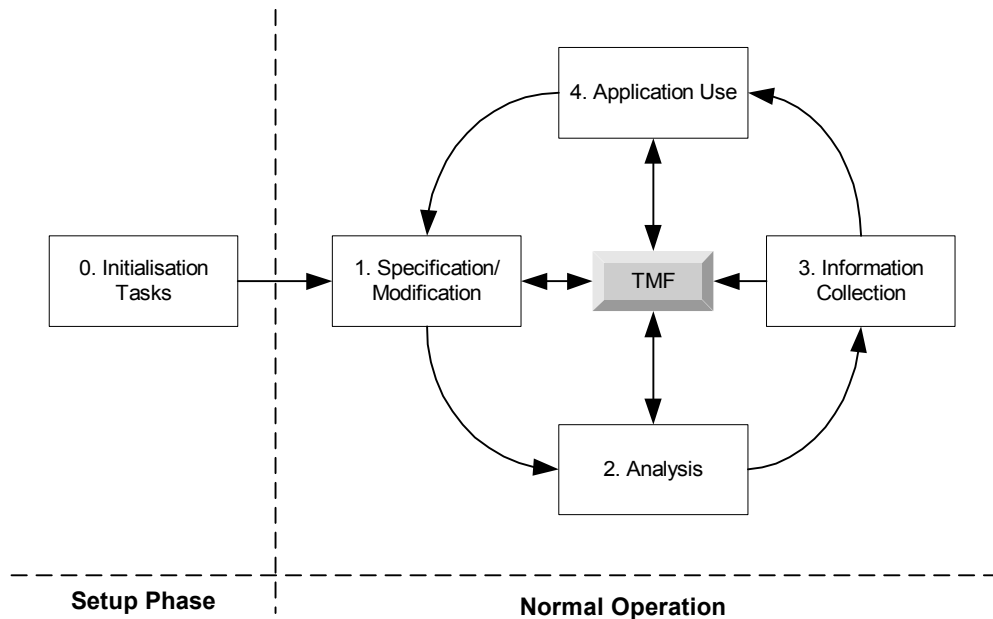


Figure 7.2: SULTAN Trust Management Life Cycle

The process of information collection leads to information being stored in the TMF and may trigger an analysis alarm. The relationships specified and information gathered from the monitoring service may be queried by applications about to engage in a transaction (module 4).

7.2 Basic Data Structures

The basic structures used in the SULTAN TMF can be placed in the following categories: specification-oriented, analysis-oriented, and risk calculation oriented. The specification-oriented databases are the Specification Database and the Entity-Connections Database, which represents the organizational chart for the firm. The analysis-oriented structures are the State Information Database, the SULTAN Analysis Model (SAM) and the predefined template of conflicts and ambiguities. The Risk Likelihood, Risk Profile, Action Dependency and Asset Databases are the risk calculation oriented databases.

7.2.1 Specification Server

The Specification Database stores the trust and recommend statements created by the administrator. A container object, called the Specification Server, encapsulates the database. The Specification Server abstracts away the implementation details of the database and facilitates the use of access control and security measures on the Specification Database.

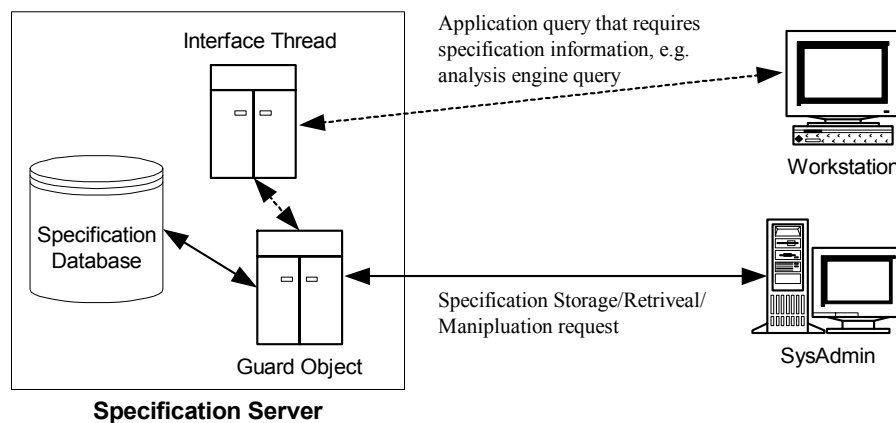


Figure 7.3: Specification Server

Figure 7.3 shows the organization of the Specification Server. Bi-directional broken lines represent the fact that requests can be made and answers returned. Solid lines represent data flow. The Guard Object (GO) is the primary interface to the database. The Interface Thread (IT) is the application programming interface (API) that allows programs to read and selectively write data. Essentially, the IT offers a subset of the methods available in the GO. This was done to ensure that applications do not have more access than is required to fulfil their tasks ('least privilege principle'). This design decision reduces the risk of the Specification Server being compromised by user applications. It should be noted that it is assumed that entities wishing to use the IT or GO must provide an authentication token to verify their identity.

7.2.2 Entity-Connections Server

As stated in Chapter 6, the Entity-Connections Database stores a representation of the organizational chart for a firm in a set of *isPartOf* tuples. Chapters 3 and 6 discuss the symbol representation of such a tuple and its interpretation. This Database is important to identify the initial entities of the system and the relations between them, which is useful in analysis.

As with the Specification Database, the Entity-Connections Database is encapsulated. The functions of the IT and GO in the Entity-Connections Server is similar to those in the

Specification Server. However, the IT in this context only accepts update requests from the SULTAN Trust Monitor. The Entity-Connections Server enables the application of transitivity on entity names when the administrator is analysing specifications. This will be illustrated in Chapter 9.

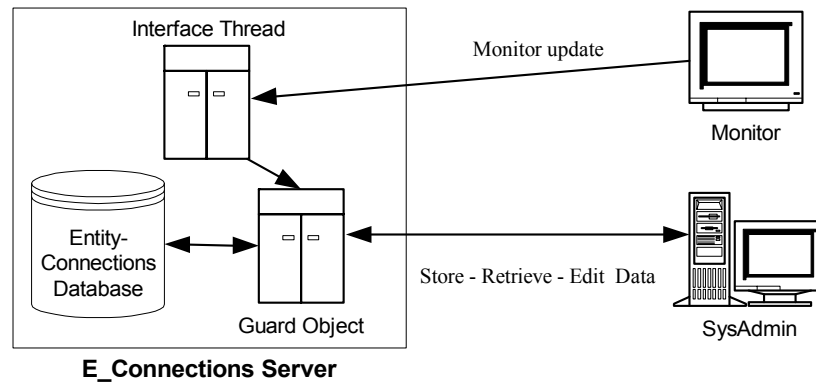


Figure 7.4: Entity-Connections Server

7.2.3 State Information Server

The State Information Database stores the risk, experience and state information for the system. This Database enables the scenario-based analysis and some of the SULTAN consulting services. Since the State Information Database currently has only three potential users, the SULTAN Trust Monitor, the SULTAN Analysis Tool or the SULTAN Consultant, the architecture is simpler than the ones previously discussed. There is no need for the differentiation between one interface for the system administrator and another for everyone else. Figure 7.5 shows the State Information Server and its interactions with the Analysis Tool, SULTAN Consultant and SULTAN Monitor. Note that the IT in the State Information Server allows: 1) the Consultant to ask the Database questions and to retrieve data from it, 2) the Monitor to write information to the Database, and 3) the Analysis Tool to store, retrieve and manipulate the State Information Database.

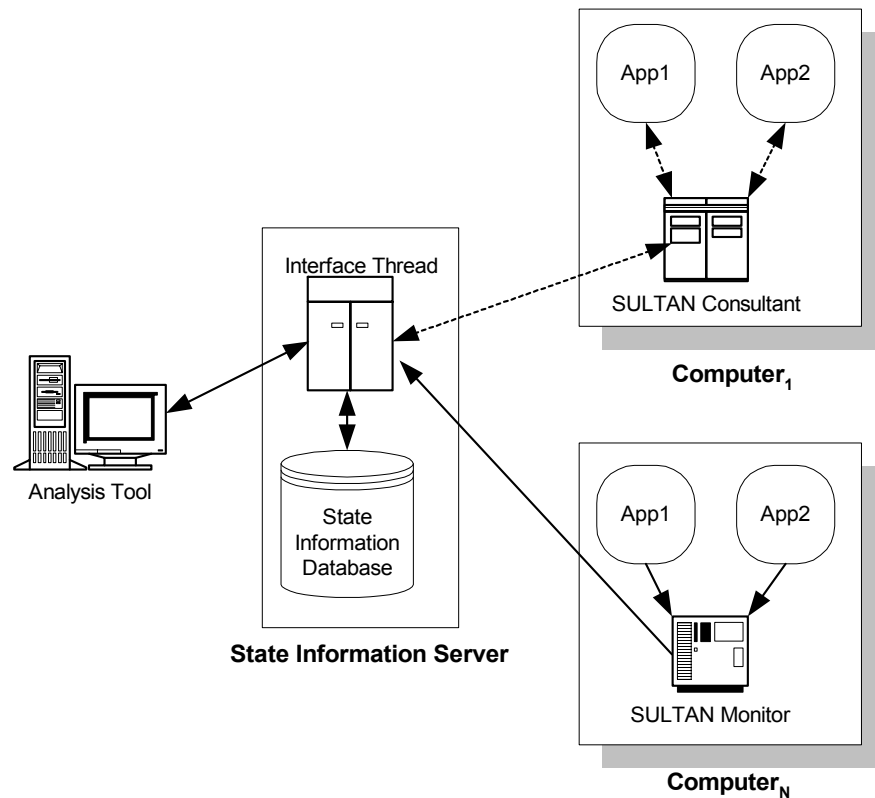


Figure 7.5: State Information Server

7.2.4 Risk Likelihood Server

The Risk Likelihood Database contains information on a set of risks, their probability of occurrence and a measure of confidence (i.e. uncertainty) in the probability estimate. This information is necessary for risk calculation. Figure 7.6 shows the architecture of the Risk-Likelihood Server, which encapsulates the Database.

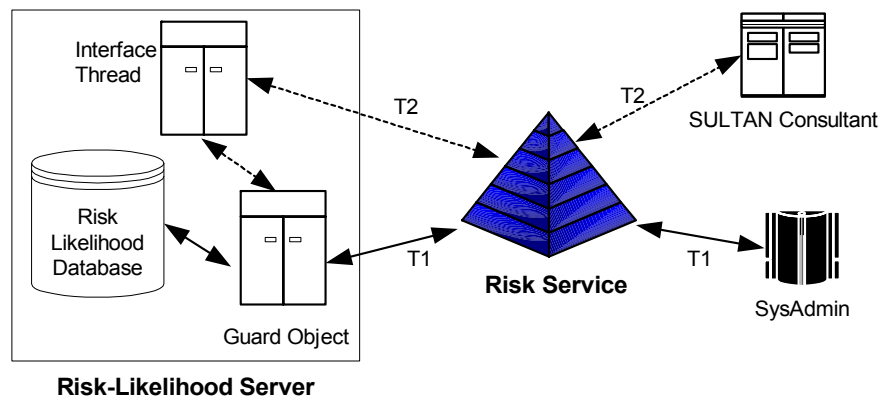


Figure 7.6: Risk-Likelihood Server

The Risk Service is the only entity that uses the Risk-Likelihood Server. However, the Risk Service receives two streams of traffic: one from the system administrator (T1) and one from the Consultant (T2). It is a part of the job of the Risk Service to determine the appropriate interface depending on the calling application. This is done with the help of authentication tokens.

7.2.5 The Other Risk Calculation Oriented Structures

The other databases used by the SULTAN TMF are the Risk Profile Database, the Dependency Database and the Asset Database, which are all used for risk calculation. They all have the same architecture as the Risk-Likelihood Server (Figure 7.6). The differences between the Risk Profile Database, the Dependency Database and the Asset Database lie in the information stored in their respective databases. The Risk Profile Database contains information on the entities in the system, their maximum allowable loss, their risk threshold and the context for which these numbers are valid (context is specified as an action or set of actions). Initially, the entity names (subject names) in this Database are populated with the entities defined by the system administrator in the Entity-Connections Database. The Dependency Database contains a list of subject-specific action dependencies. This means that information on a subject, the root action and the action it is dependent upon is stored. The Asset Database may be conceptually viewed as a sequence of tuples of the form:

(resource, ASSET_NAME, ASSET_CONTRIBUTION).

With a header tuple of the form (rTotal, Value). However, for the version of the toolset, the Asset Database is implemented as an Access database, with two tables.

7.2.6 The Other Analysis Oriented Structures

The other analysis-oriented structures are the SULTAN Analysis Model (SAM) and the template of queries. Both are simple text files, which contain Prolog definitions that enable queries to be performed.

7.2.7 Basic Data Structure Overview

Most of the basic structures are encapsulated in container objects, called Servers. This encapsulation allows the implementation details of the structure to be abstracted away and provides a mechanism by which access control can be enforced and the Databases' integrity can

be verified. Each structure interacts with a specific set of tools from the TMF. Figure 7.7 shows the connections between the tools and the structures.

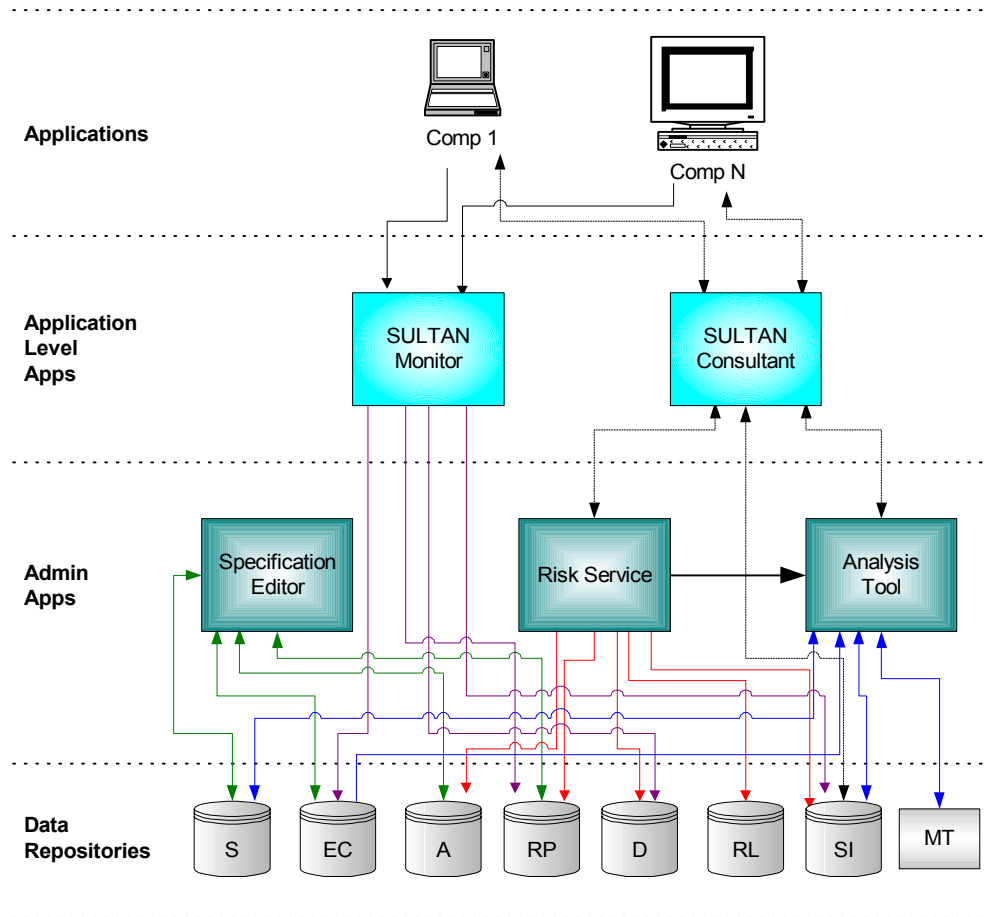


Figure 7.7: Data Structures, Tools and their connections

Abbreviation	Meaning
S	Specification Server
EC	Entity-Connections Server
A	Asset Server
RP	Risk Profile Server
D	Dependency Server
RL	Risk Likelihood Server
SI	State Information Server
MT	SULTAN Analysis Model & Template

Table 7.1: Abbreviations for Data Structure Chart

With the exception of the SULTAN Analysis Model and the SULTAN template of standard queries, all the databases have been implemented in Microsoft Access and all the GOs and ITs developed in Java.

7.3 Specification Editor

The Specification Editor is an Integrated Development Environment that ties together the processes of creating, modifying, compiling, storing, retrieving and translating specifications. The Editor also supports the tasks of creating, updating, storing and retrieving risk profile, entity-connections and asset information. The contents of the risk-likelihood database are also manipulated through this tool. Figure 7.8 shows the basic elements of the Specification Editor.

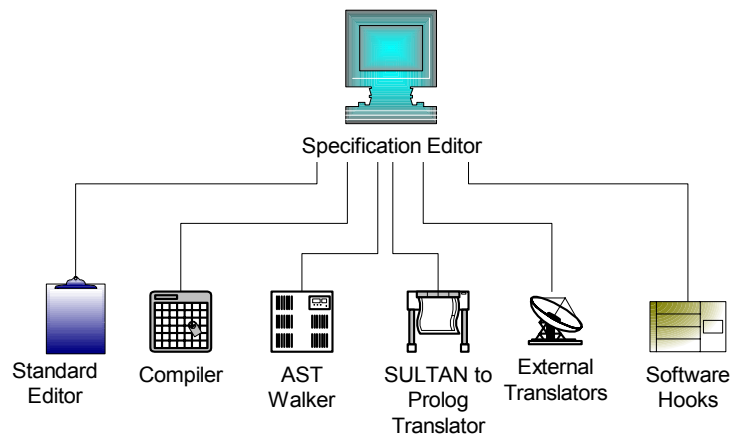


Figure 7.8: Components of the Specification Editor

In this section, each of the components shown in Figure 7.8 will be presented.

7.3.1 Standard Editor

The Standard Editor is a fully customisable, basic text editor designed specifically for manipulating SULTAN specifications. Standard text manipulations functions, such as copying, pasting, searching, replacing, shifting lines right, etc are supported. The Editor also supports syntax highlighting for reserved words of the SULTAN specification language. This is enabled by storing all the settings for the editor. Figure 7.9 shows a snapshot of the Editor, with the specifications from the example in Chapter 3. Associated with the Standard Editor is a context-sensitive mini-editor, which interfaces with the Risk-Likelihood, Asset, Entity-Connections and Risk Profile Servers. Figure 7.10 shows the mini-editor when used to manipulate the Entity-Connections Server. The mini-editor is run from the menu options in the Standard Editor. It should be noted that the Editor environment is stored in a series of settings files. Thus, all the features of the Editor, from the text size for the menu options to look and feel of the windows, may be reconfigured by the administrator through the Options menu choice in the Editor.

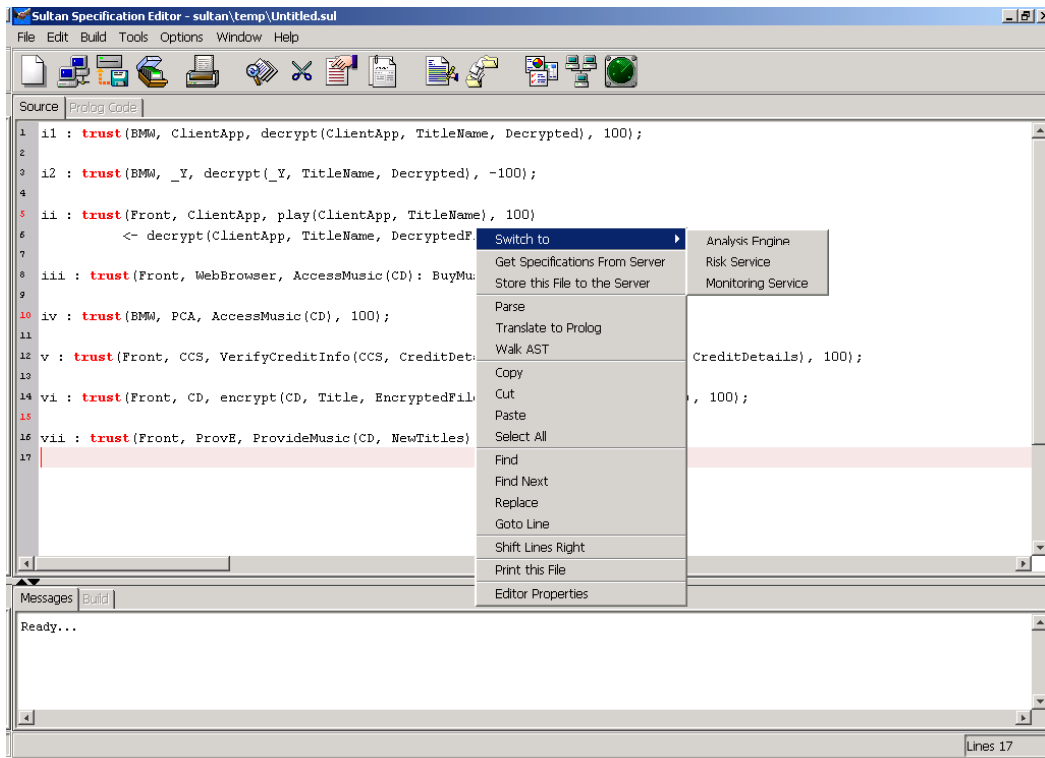


Figure 7.9: Snapshot of the Specification Editor

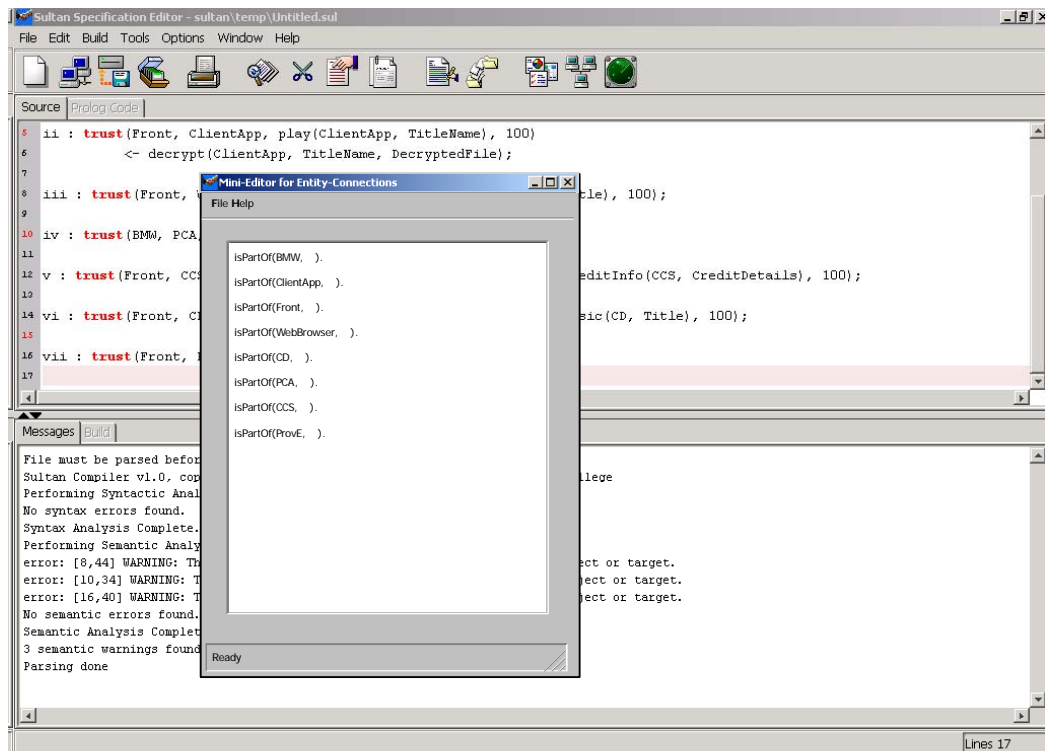


Figure 7.10: Mini-Editor used to update Entity-Connections Database

7.3.2 Compiler

The SULTAN Compiler ensures that code entered by the administrator adheres to the syntactic and semantic rules used to define valid SULTAN code. These rules are outlined in Chapter 3 and in the Appendices. Figure 7.11 illustrate the process involved in the compilation of SULTAN specifications.

The main modules of the compiler are based on a LALR(1) parser, which was generated by SableCC, an object-oriented Java parser generator [159]. The current compiler provides an intermediate representation, which is an abstract syntax tree with added label, and allows the addition of translators (abstract syntax tree walkers). Semantic checks enforce the ‘common sense’ rules discussed in Chapter 3 on the use of particulars features in the specification notation, e.g. a variable trust level must be a part of a constraint involving a comparison, action restrictions cannot be used in distrust statements, etc.

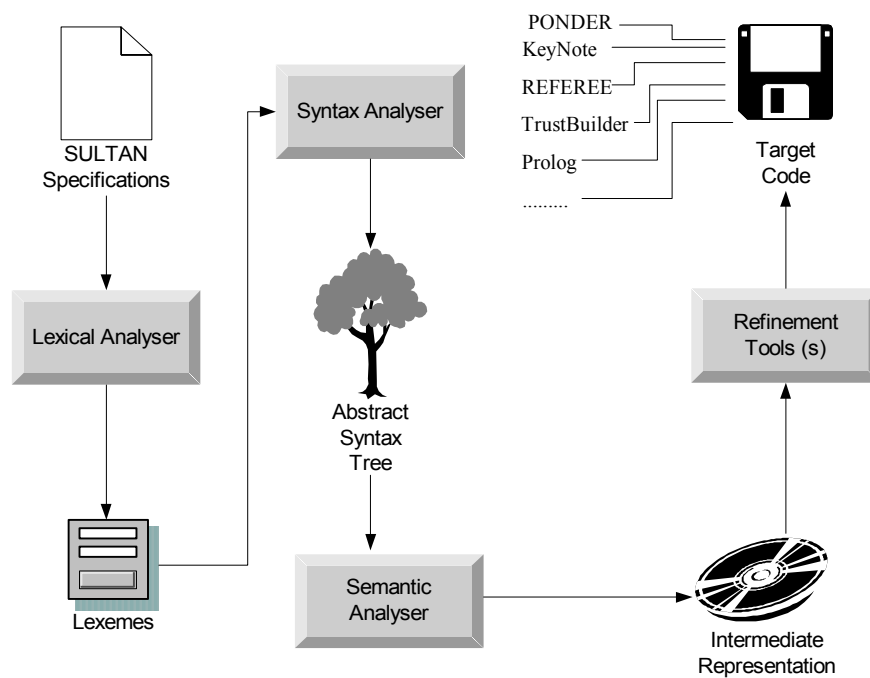


Figure 7.11: SULTAN Compiler Processes

Figure 7.12 shows a snapshot of the Editor, with the compilation results, for BMW (from Chapter 3).

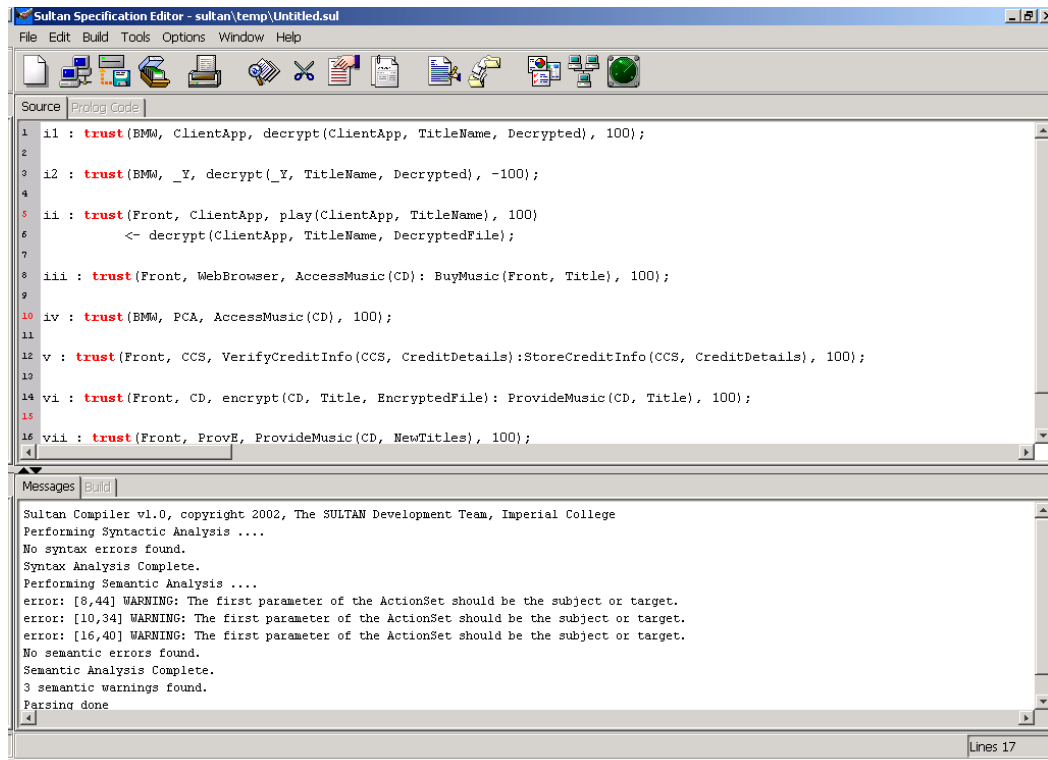


Figure 7.12: Compiled Specifications for BMW

From Figure 7.12, it is clear that some semantic warnings are not treated as critical; they are simply flagged. This is done in order to allow translation to proceed. However, it is advised that all errors and warnings be resolved as only a compiled set of specifications can be stored to the Specification Database.

7.3.3 AST Walker

The AST (Abstract Syntax Tree) Walker is a graphical representation of the abstract syntax tree for the specifications currently being manipulated by the administrator. Figure 7.13 shows the result of calling the AST Walker on the specifications for BMW. It is a required that specifications be compiled before the AST Walker may be run, because the compilation process produces the AST to be walked.

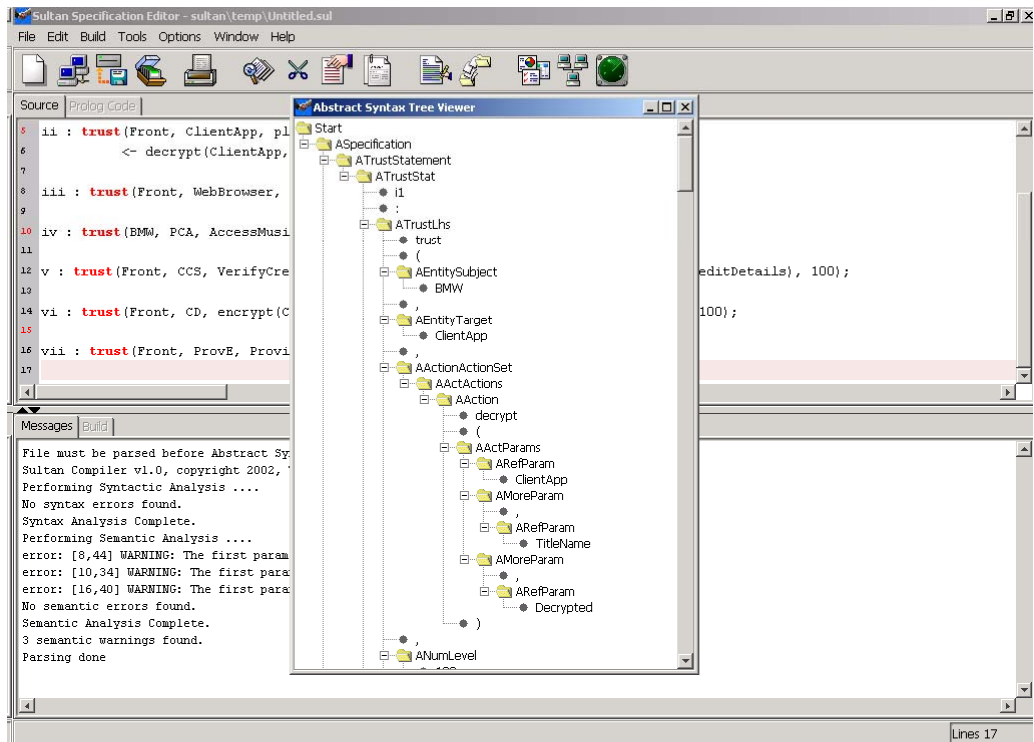


Figure 7.13: AST Walker run on BMW specifications

7.3.4 SULTAN to Prolog Translator

As proof that the refinement of SULTAN specifications to other languages is possible, a translator from SULTAN to Prolog is an integral part of the Specification Editor. Such a translator is also necessary to allow the specifications to be analysed by the Analysis Tool. Figure 7.14 shows the results of executing the translator on the specifications for BMW. The translation rules are outlined in Appendix C. As illustrated in Figure 7.11, the translator walks the AST and transforms the tree into the target code, which is Prolog in this case.

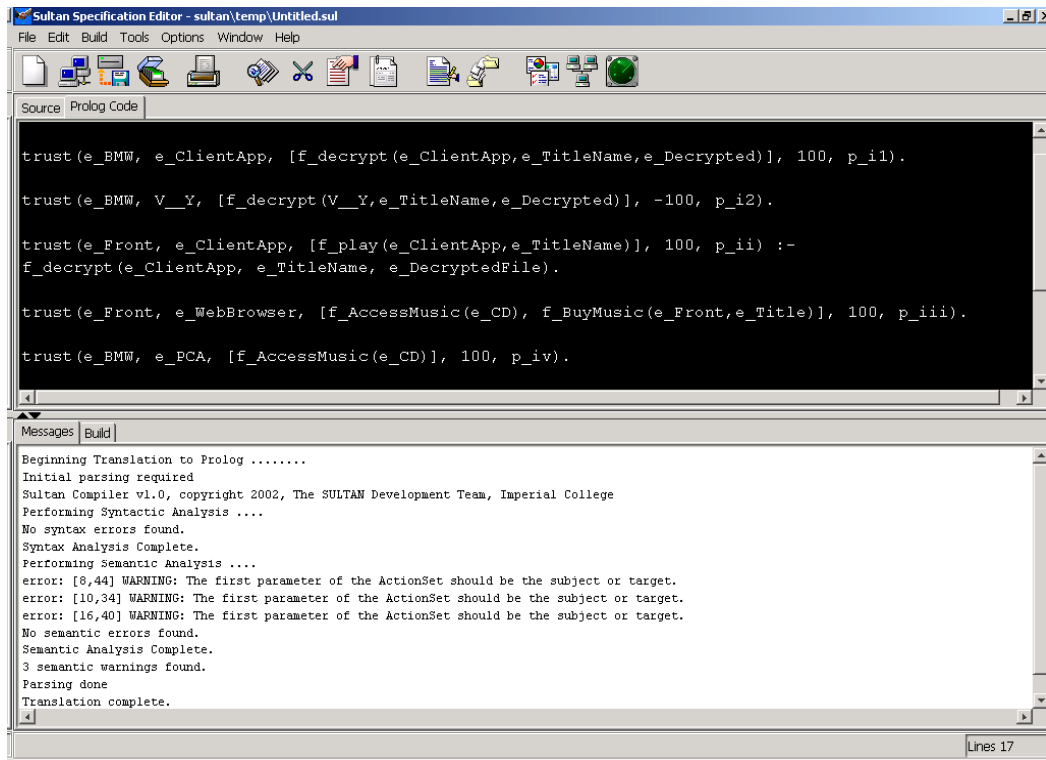


Figure 7.14: Using the SULTAN to Prolog translator on the BMW specifications

7.3.5 External Translators

External Translators can be integrated in the Specification Editor by adding them to a list of supported translators. This list is stored locally in the Editor settings files, which means that all translators added to the Editor will be accessible through the Editor on subsequent runs. Thus, once a translator (AST walker and transformer) has been created, it can be integrated into the Editor functionality.

Once translators have been created then the Editor provides a facility to specify the file and its location and have it integrated as a part of the Editor. The dialog used to perform this integration is shown in Figure 7.15. It should be noted that standard checks are performed on the file (a java source file) before it is integrated with the Specification Editor. These tests include checks on the file's existence, permissions, and it's use of the AST.

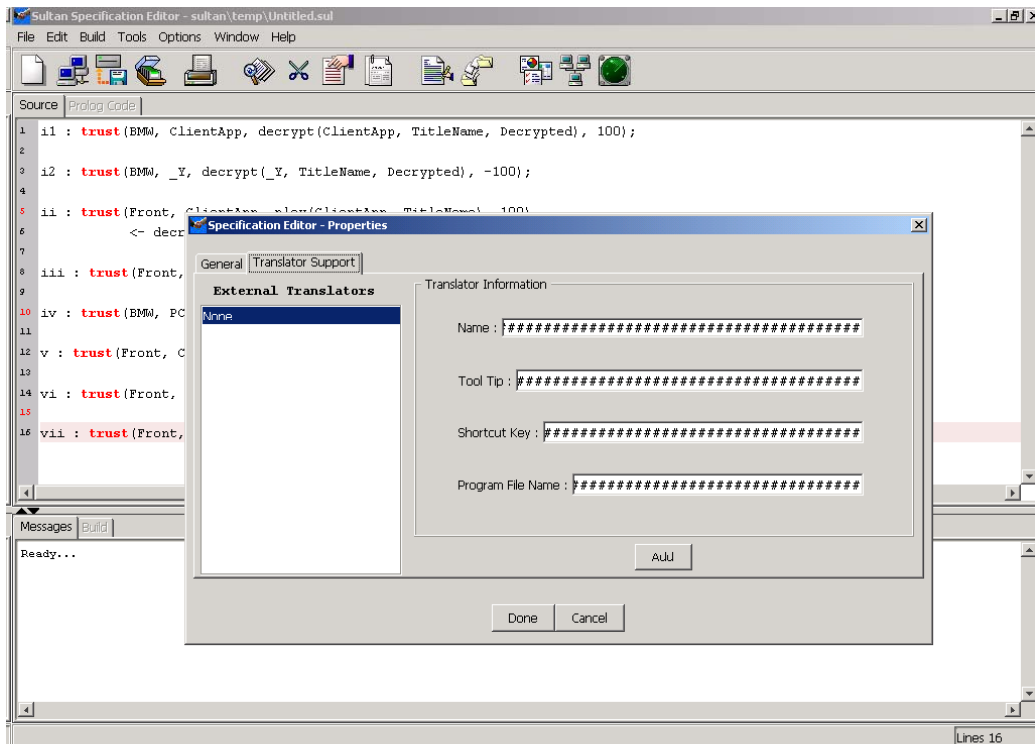


Figure 7.15: Adding an External Translator

7.3.6 Software Hooks

The Specification Editor has hooks into the Analysis Tool, the Risk Service and the mini-Editor. These hooks pass the state of the Specification Editor to the tool being switched to. For example, using the hook to the Mini-Editor for risk profiles, the Specification Editor can be asked to load the Mini-Editor with a template of risk profile records for each of the entities used in the specifications (Figure 7.16). Similarly, hooking into the Analysis Tool would mean that the specifications would be automatically translated and loaded into the Analysis Tool.

7.4 Analysis Tool

The Analysis Tool is the graphical front-end for the Analysis Engine, which is the component that interfaces with the Specification Server, State Information Server, the SICStus Prolog System, the SULTAN Analysis Model, the organization chart information and the template of conflicts and redundancies. Figure 7.17 further illustrates this point. The Analysis Engine encapsulates the SICStus Prolog System. This is done to secure the Prolog System from

(possibly intentional and malicious) manipulation. A Guard Object is defined in the Engine that negotiates access to the Prolog System.

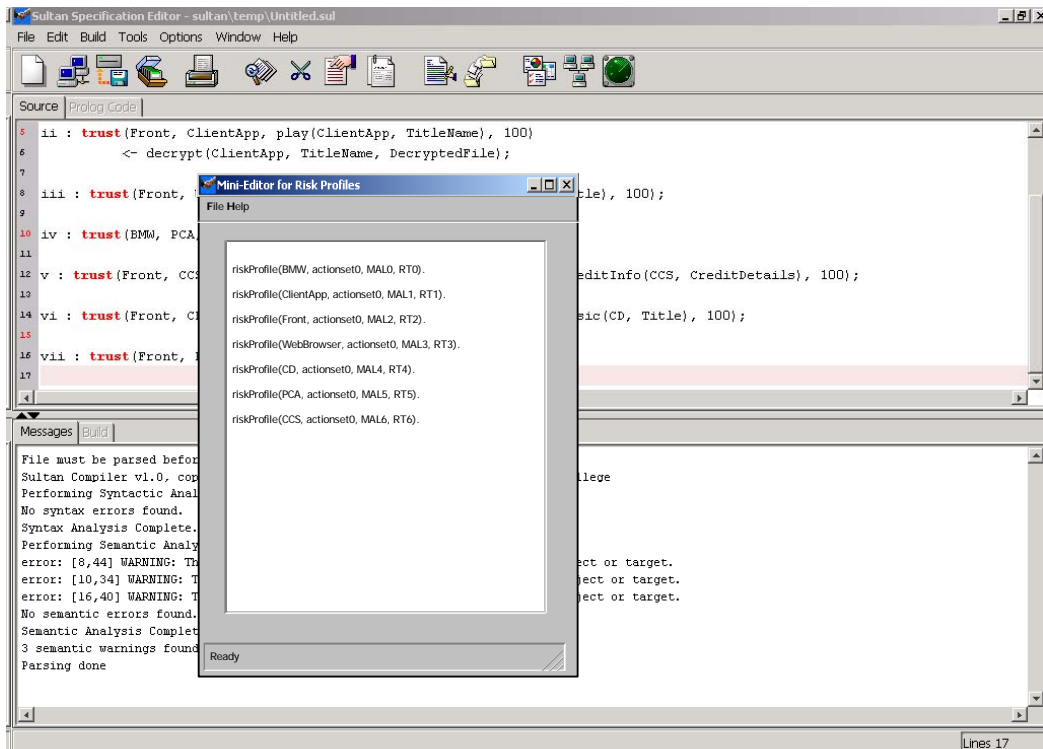


Figure 7.16: Software Hook to the Risk-Profile Mini-Editor

The interface used from the GO (coded in Java) to Prolog System is the Jasper Interface. Using Figure 7.17, a call to the Analysis Tool results in the GO retrieving the specifications from the Specification Server, loading the organisation chart information, getting the state information from the State Information Server, retrieving the SULTAN Analysis Model (which defines the reasoning mechanism), retrieving the SULTAN Analysis Template and loading all of this information into the Prolog System. Note that specifications, organisational chart information and state information are translated to Prolog before being loaded into the System. The bi-directional arrows between the GO and the SULTAN Analysis Template and the GO and the State Information Server indicate that the GO is able to change and store these two artefacts. This is done via the Analysis Tool. The exact circumstance where this may be necessary will be discussed further in this section.

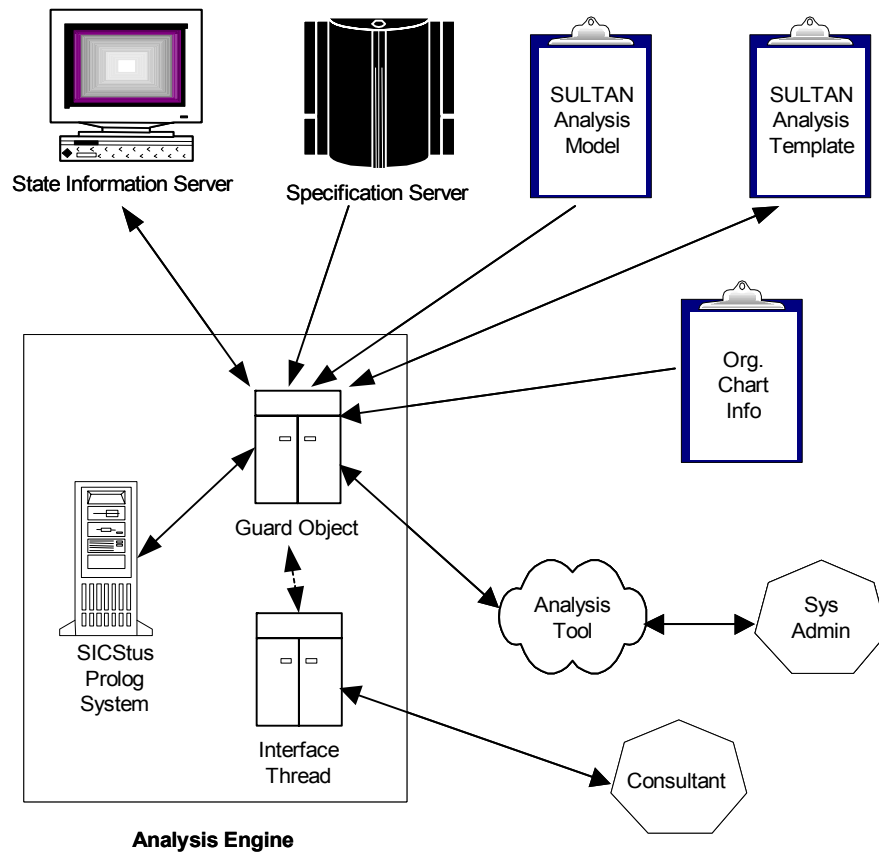


Figure 7.17: Analysis Tool – Analysis Engine Connection

The Analysis Tool provides a number of tools that try to make the process of analysis query construction easier, namely: the console, the loader, the viewer, query statement builder and the state manager. 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.

7.4.1 The Console

The console is the data input area, which is the bottom part of the Tool window. SULTAN Analysis query statements (adhering to the format discussed in Chapter 4) are entered into the console. These statements are executed at the administrator's request. Each request made to Prolog system is assumed to take a long time. Thus, each request is executed in a child execution thread and the administrator is free to enter new requests immediately. The requests are queued and the results are sent to the output area of the Analysis Tool in the order of receipt. All this is managed by an execution thread manager.

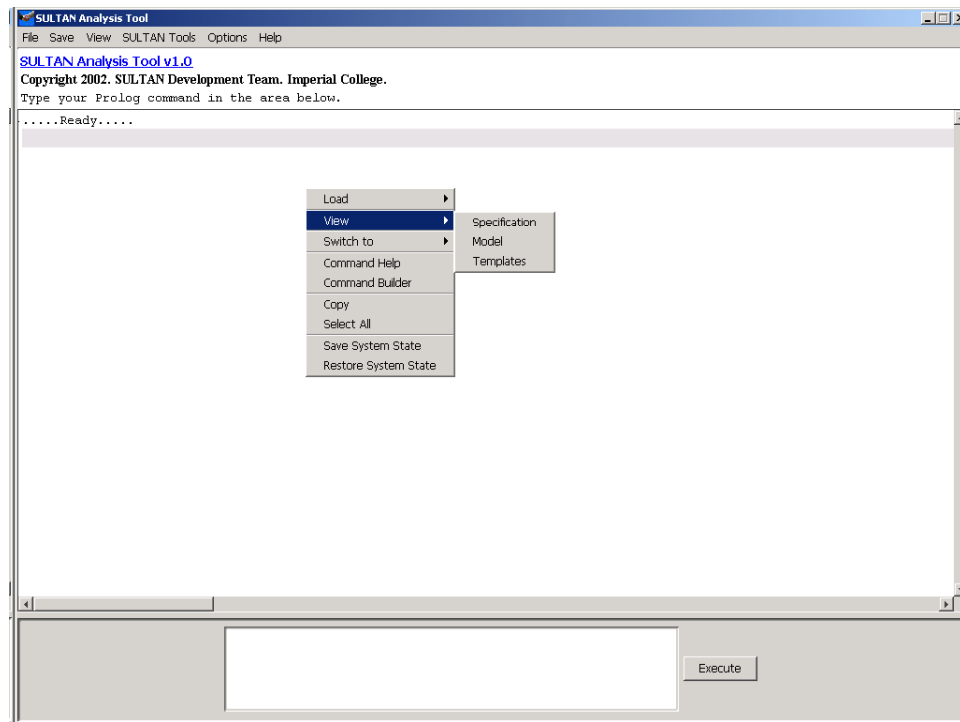


Figure 7.18: Snapshot of the Analysis Tool

7.4.2 The Loader

The loader allows the administrator to include a set of assertions from an already existing file. This feature is included to allow the re-use of previously created analysis queries and or scenarios. Note that the file must contain Prolog statements. An Open-Dialog box is the primary interface used to call the loader. Figure 7.19 shows the dialog used. It is important to mention that the Analysis Tool contains facilities for the storage of the history of queries entered in its current execution. In addition, the loader performs a simple syntax check to ensure that the file adheres to Prolog syntax.

7.4.3 The Viewer

The viewer is a simple text editor that allows the administrator to see the state information, the translated specifications and the SULTAN Analysis Model. It also allows the display and modification of the SULTAN Analysis Template. Figure 7.20 shows the viewer being used to look at the Template.

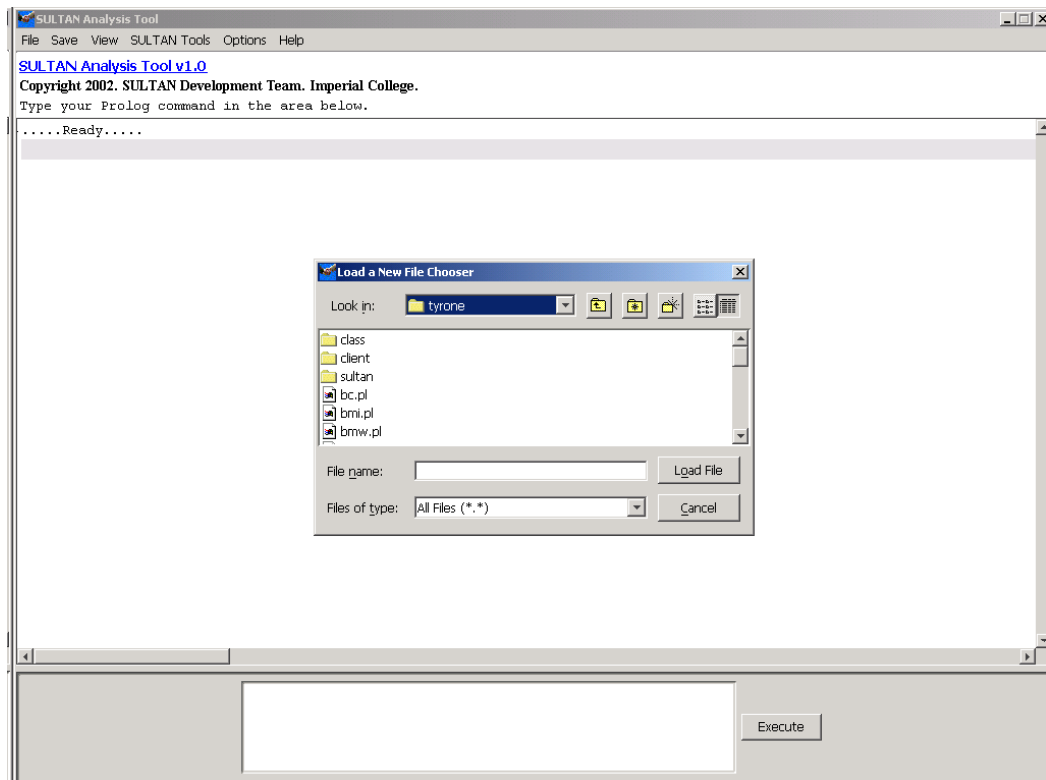
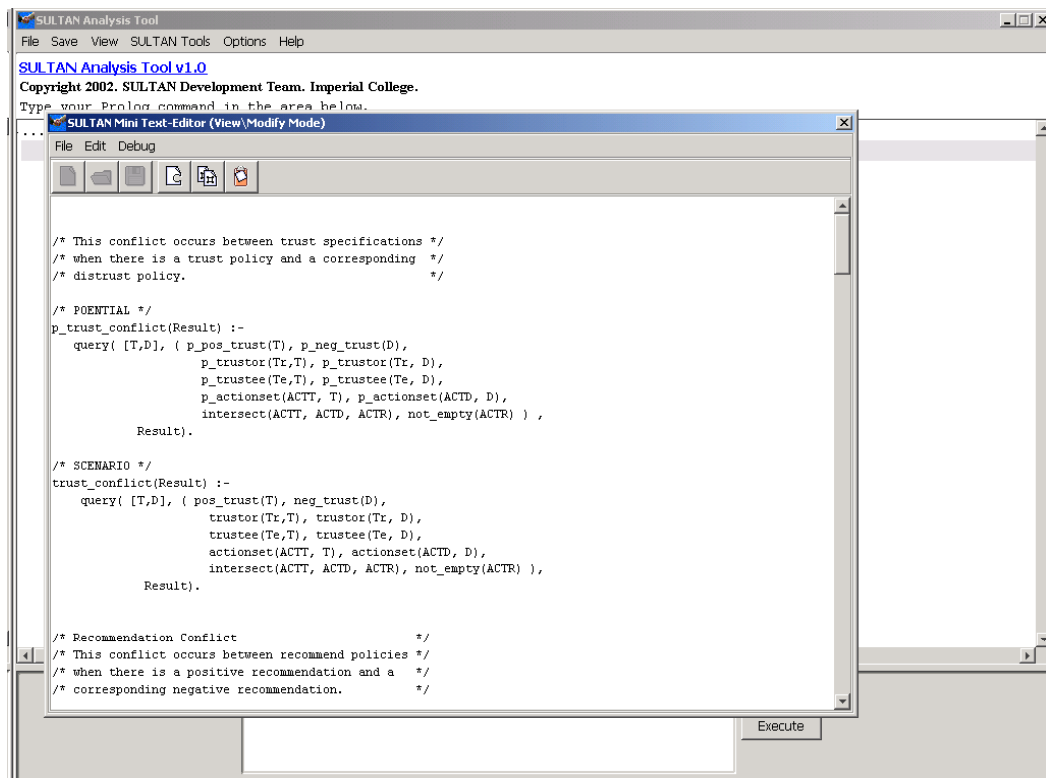


Figure 7.19: Loading a file



The administrator may use the viewer to update the Template and then use the Loader to redefine the common set of conflicts and ambiguities.

7.4.4 Query Statement Builder

Analysis queries may be complex. They may vary in difficulty within a particular query type (hard versus easy source analysis queries) and across query types (e.g. constraint satisfaction queries tend to be easier to formulate than some scenario analysis queries). To enable the easy construction of analysis queries the Analysis Tool contains a Query Statement Builder, which is a simple GUI that allows the query formulation. Figures 7.21 and 7.22 show different aspects of the Query Statement Builder. Figure 7.21 shows the types of queries accommodated. The Builder allows the creation of all five categories of queries, i.e. source queries, scenario queries, mixed queries (i.e. a mix of both source and scenario predicates), cycle detection queries and constraint discovery queries. Figure 7.22 shows the beginnings of a source analysis.

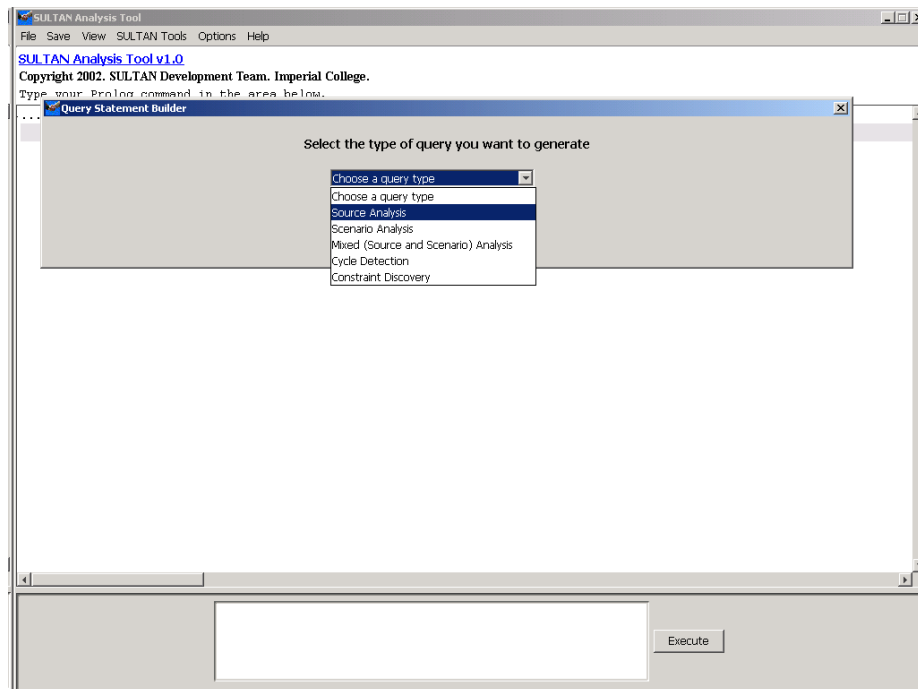


Figure 7.21: SULTAN Query Statement Builder (First Level)

7.4.5 State Manager

The state manager keeps a history of the queries entered into the Console and a record of the facts and rules stored in the memory of the Prolog system in the current run of the Analysis

Tool. At any time the query history or the assertions stored in the Prolog system can be saved to temporary files. This facility makes it possible for the administrator to continue a previous line of analysis or even combine several streams of previous analyses.

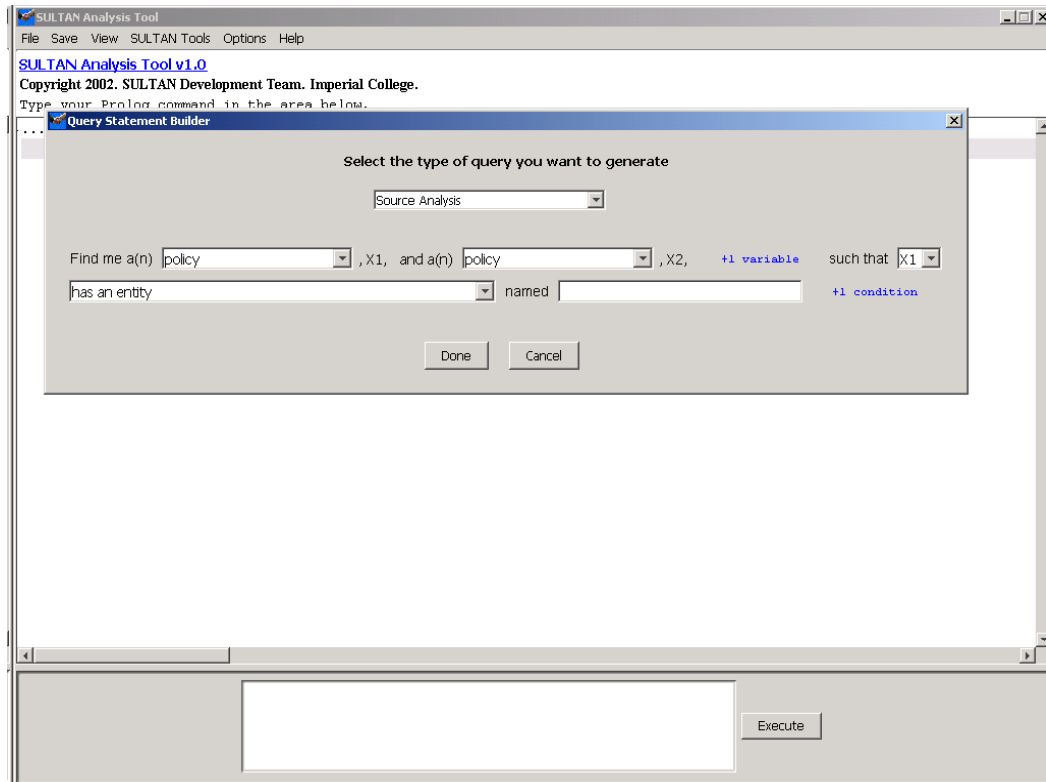


Figure 7.22: Source Analysis using the SULTAN Query Statement Builder

7.5 Trust Monitor

In Chapter 6, it was stated that the Trust Monitor is the tool that updates the entity connections database, the action dependency database, the risk profiles database, state, risk and experience information. It was also stated that the Monitor is designed on a client-server architecture. In this section, the technological details behind the architecture employed will be further discussed. Figure 7.23 represents a realistic illustration of the Trust Monitor architecture. From the discussion in Chapter 6, it is assumed that a SM Client is installed on each of the computers in the domain. The SM Client appears to the application to be a socket server, i.e. the SM Client has a socket open on port 33991. Thus, any application that can read and send information to a socket can use the SM Client. Behind the socket server interface presented to the application is a Java RMI (Remote Method Invocation) Client, which uses the methods defined in RMI Server in the SM Server. The use of sockets as the interface to applications

implies that the SM Client has to implement its own communication protocol and server access logic. Thus, the single uni-directional arrow between the application and the SM Client is actually an abstraction, which represents three local interactions (Figure 7.24). Why use both RMI and sockets? Sockets are used because it enables the SM Client to be used by any application. Using RMI as the application interface would imply that all the application using the SM Client must be written in Java, which is not a feasible assumption. RMI is used in the background because it offers an easy way to develop the SM Server without worrying about communication details, connection management or session management.

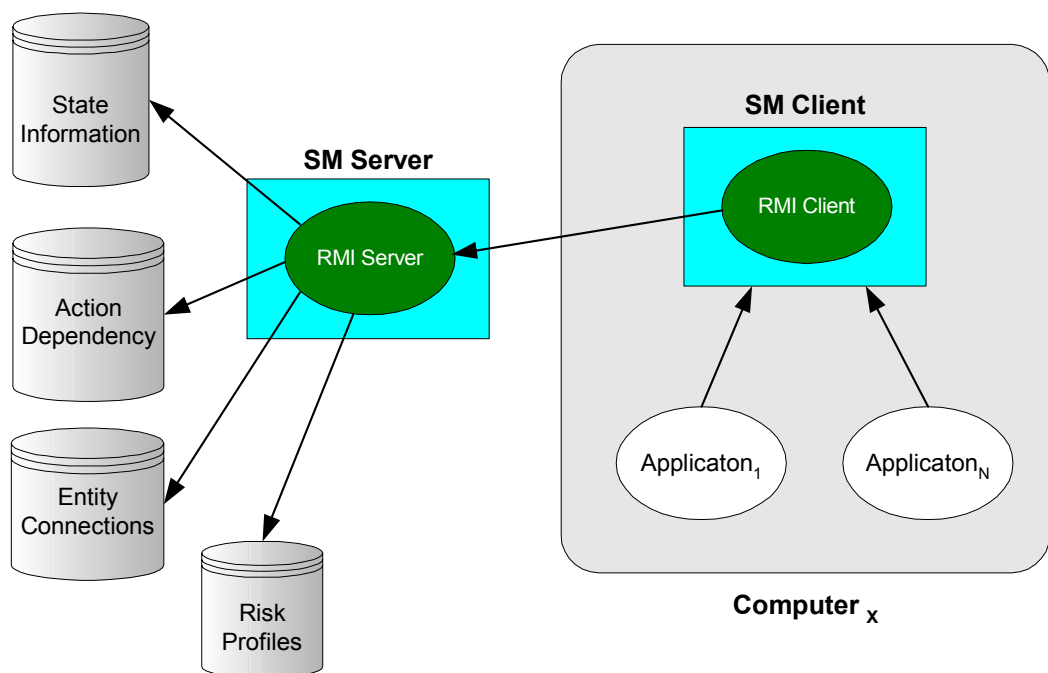


Figure 7.23: More detailed SULTAN Monitor Architecture

The SM Client socket server employs a simple communication protocol. This is illustrated in Figure 7.24.

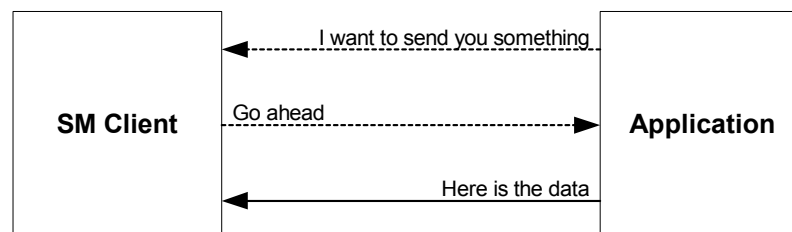


Figure 7.24: SM Client Socket Server Interactions

The process outlined in Figure 7.24 can be stated as follows:

- The application sends a packet (hello, Name), which means ‘Hello, I am application Name. Can I send you some info?’
- The SM Client receives this packet and checks its list of apps. If the app exists then the SM Client records the application id. If the app does not exist the SM Client creates an application id and records the id. Finally, the SM Client tells the application to send the data. (The id generation process is described in Chapter 6)
- The application sends the monitoring data.

Summarizing the discussion of Trust Monitoring in Chapter 6, the application can send monitoring data in the following formats:

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

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

(attribute, value) – provides state information

(**isPartOf**, app-id, UCI) – app-id is a part of UCI

(**depends**, action1, action2) – for action1 depends on action2

(**rprofile**, actions, MAL, RT) – MAL and RT are the values for subject and actions

When risk, experience or state information is added/updated, the SM Server calls the Analysis Tool to run the queries in the SULTAN Analysis Template. These queries are run in a separate thread so as not to significantly impact the process of information receipt by the SM Server. It is understood that these queries are performed on the information in the Specification Server and the State Information Server. If a potential conflict or ambiguity is detected then the SULTAN System sets a flag to alert the administrator the next time the Specification Editor or Analysis Tool or Risk Service is used.

7.6 Risk Service

As stated in Chapter 1, risk is a measure of the probability of a transaction failing. The job of the Risk Service is to provide such a measure, either based on stored risk information or on transaction risk factors (e.g. risk likelihood, etc.). Thus, the Risk Service has two purposes (presented in Chapter 5), which are to retrieve risk information and or to assess the risk for a particular context. The processes involved in risk information retrieval and risk assessment were covered in Chapter 5. In this section, the architecture of the Risk Service will be explained. Figure 7.25 is a pictorial representation of the Risk Service’s architecture.

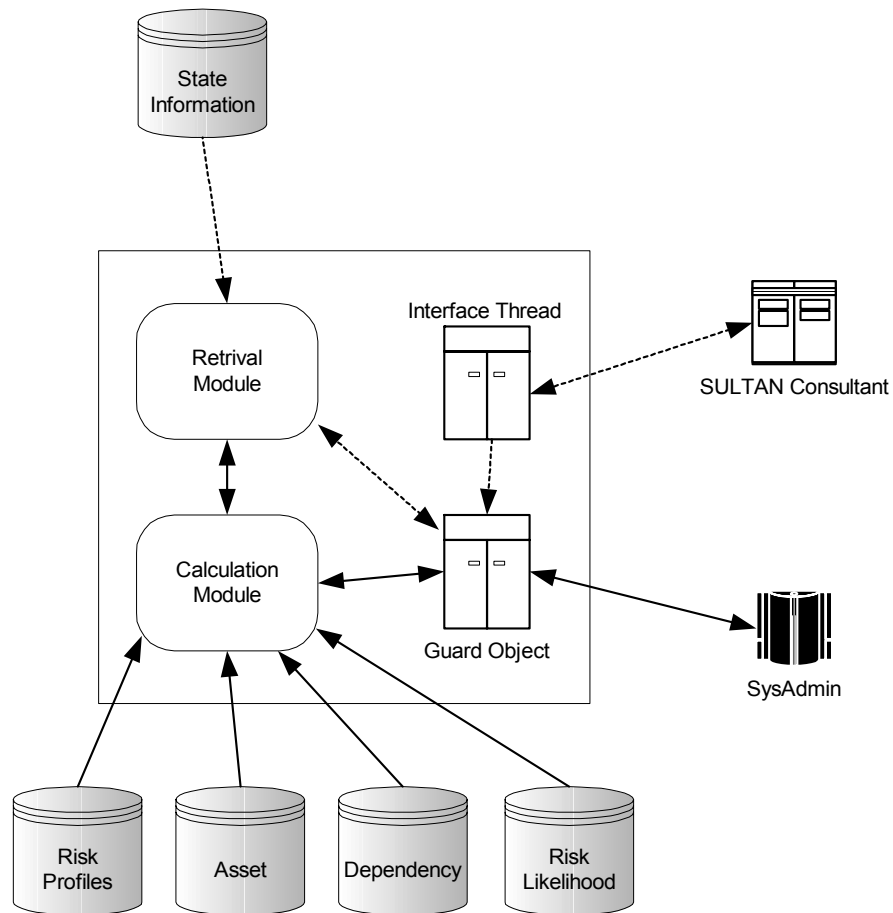


Figure 7.25: Architecture of the Risk Service

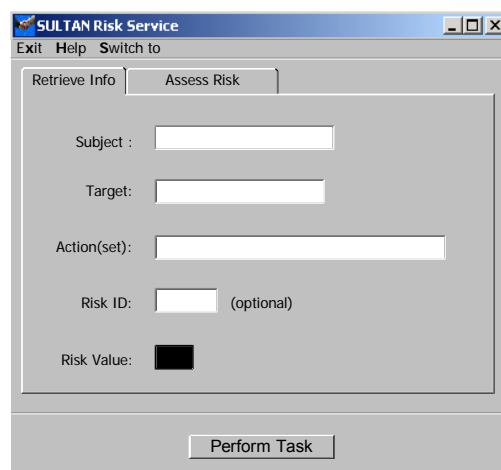


Figure 7.26: Snapshot of Admin's interface to Risk Service

The Consultant (discussed in the next section) allows ordinary users to ask questions concerning the risk involved in a transaction. The system administrator is provided with a simple interface (Figure 7.26), which allows him to examine the riskiness of certain contexts and possibly use

this information to refine the system's specifications. The admin's graphical interface to the Risk Service currently offers the basic functionality. For tasks such as calculating the expected loss, the SRS should be called at the command line with arguments.

7.7 Trust Consultant

The trust consultant performs an advice service. This function is normally described as 'recommendation provision' in other contemporary trust management systems. In the context of these systems, the term recommendation is normally used to imply the fact that the system recommends a course of action, but the decision is inevitably the user's. In the SULTAN TMF, the task of providing the user with information to enable decision-making is referred to as a trust consultation. To enable trust consultations, a trust consultant client must be resident on each of the firm's machine. These clients interact with the SULTAN Trust Consultant (STC) Server. Figure 7.27 illustrates the architecture employed to facilitate consultation.

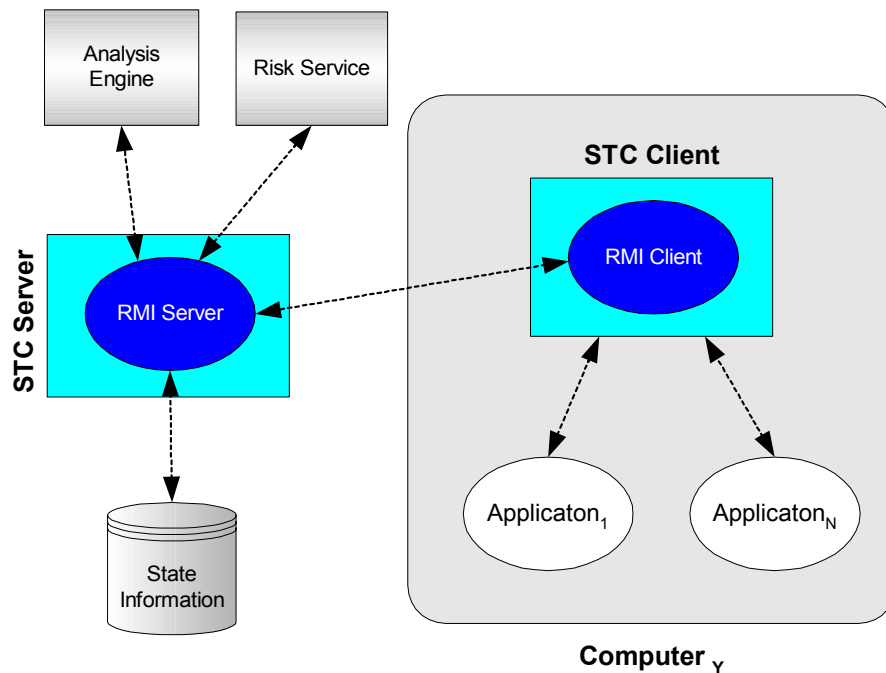


Figure 7.27: Architecture of the Sultan Trust Consultant (STC)

The STC's architecture is similar to that of the Trust Monitor. The STC Client is viewed as a socket server by applications (on port 33993). The Client interacts with the RMI Server in the STC Server, which interfaces with the Analysis Tool, Risk Service and State Information

Server. Applications go through the same three-step protocol process stated in the discussion on the Trust Monitor. The questions that can be asked by an application are:

1. Should I trust target, Y, to perform action(set), A?
This question sent in the format (*trust, Y, A*) to the STC Client.
2. Should I trust target, Y, to perform action(set), A, at level L?
This is stated in the format (*trust, Y, A, L*) to the STC Client.
3. Should I recommend target, Y, to perform action(set), A?
This is specified in the format (*recommend, Y, A*).
4. Should I recommend target, Y, to perform action(set), A, at level L?
This question is constructed using the form (*recommend, Y, A, L*).
5. What is the risk in engaging actions, A, with target, Y?
This is asked by sending a tuple of the form (*risk, A, Y*) to the STC Client.
6. What has my experience been with respect to target, Y, and actions, A?
This is stated in a form similar to (*exp, Y, A*).

The answer returned is dependent on the question asked. Questions 1 to 4 will return a tuple of the form (*BooleAnswer, Justification*), where *BooleAnswer* is *Yes* or *No*, and *Justification* is the partial explanation for the answer. For example, the question (*trust, ClientApp, play(ClientApp, TitleName)*), with the specifications for BMW in Chapter 3 and an empty State Information Database, will return (*No, decrypt(ClientApp, TitleName, DecryptedFile)*). Thus, the user is now aware that, based on the information in the system, *decrypt(ClientApp, TitleName, DecryptedFile)* must be proven before *ClientApp* trust should be trusted. If the State Information Database contained the fact that *decrypt(ClientApp, TitleName, DecryptedFile)* is true, then the answer returned would be (*Yes, trust(Front, ClientApp, play(ClientApp, TitleName), 100)*). This means that *ClientApp* should be trusted based on the trust specification that states that *Front* trusts *ClientApp* to perform *play(ClientApp, titleName)* at trust level 100.

For question 5, the answer returned is (*RiskValue, Justification*), where *Justification* is either the risk record(s) used to get the risk value or the word *calculated*. For example, an answer of the form (*100, risk(A,B,C)*) is interpreted as the risk value is 100 due to the risk record relating to A, B, C in the State Information Database. An answer of the form (*-50, calculated*) states that based on a risk calculation the risk value is -50. For question 6, the answer may be either (*ExpValue, Justification*) or (*0, none*). A result of (*0, none*) means that there is no experience information for the context specified. As with the answer to question 5, *Justification* is the record(s) used to derive the value.

7.8 Summary

This chapter presented SULTAN Trust Management. It was emphasised that trust management, in this context, is being viewed as:

“the activity of collecting, encoding, 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.” [4, 5]

This definition implies that trust must be specified, analysed and monitored; and that risk should be evaluated and trust information used in decision-making. The SULTAN TMF contains tools that support these trust management processes. The basic trust management lifecycle was presented and the core data structures used in the TMF were presented. Finally, the architecture, basic internal operations and use of each of the tools in the TMF were presented.

Chapter 8 Uses of the SULTAN TMF

"Nothing tends so much to the advancement of knowledge as the application of a new instrument."

- Sir Humphrey Davy [160]

The SULTAN TMF may be used both as a decision support tool to aid human managers or automated manager agents and to support on-line trust queries for policy decisions relating to access control or which security mechanisms to use. The components of the TMF may be used in a diverse range of tasks, e.g. the specification notation may be used to model a variety of situations (described in Appendix F). In this chapter, specific instances where the TMF may be used are presented.

8.1 Simulation Analysis

The Analysis Tool (presented in Chapter 7) provides an interface to the addition and retraction facilities of Prolog. Once the Tool is being used, the administrator has a local copy of the state information and specifications. He can insert trust/recommend rules, perform analysis queries, add state information and perform even more queries to ascertain the impact of his changes. To demonstrate how simulation may proceed, a very simple example using the BMW specifications from Chapter 3 will be used. The specifications for BMW are:

```
i1 : trust(BMW, ClientApp, decrypt(ClientApp, TitleName, Decrypted), 100);
i2 : trust(BMW, _Y, decrypt(_Y, TitleName, Decrypted), -100);
ii : trust(Front, ClientApp, play(ClientApp, TitleName), 100)
    ← decrypt(ClientApp, TitleName, DecryptedFile);
iii : trust(Front, WebBrowser, AccessMusic(CD): BuyMusic(Front, Title), 100);
iv : trust(BMW, PCA, AccessMusic(CD), 100);
v : trust(Front, CCS, VerifyCreditInfo(CCS, CreditDetails):
    StoreCreditInfo(CCS, CreditDetails), 100);
vi : trust(Front, CD, encrypt(CD, Title, EncryptedFile): ProvideMusic(CD, Title), 100);
vii : trust(Front, ProvE, ProvideMusic(CD, NewTitles), 100);
```

For this example, the Prolog translated representation will be used during analysis. Suppose the administrator wants to find out if there are any trust conflicts in the source code. He could use the `p_trust_conflict(X)` statement from the template.

```

SULTAN Analysis Tool
File Save View SULTAN Tools Options Help
SULTAN Analysis Tool v1.0
Copyright 2002. SULTAN Development Team. Imperial College.
Type your Prolog command in the area below.
.....Ready.....
Current Command : p_trust_conflict(X).
Results :
-----
[ {X=.(.(p_i1,.(p_i2,[])),[])} ]
-----

```

Figure 8.1: Source Trust Conflict for BMW

From Figure 8.1, there is a source trust conflict between rules *i1* and *i2*. Note that a feature of the Prolog engine used is that lists are returned in Lisp-like dotted pair notation. Suppose the administrator wants to see the effect of deleting *i2*.

```

SULTAN Analysis Tool
File Save View SULTAN Tools Options Help
SULTAN Analysis Tool v1.0
Copyright 2002. SULTAN Development Team. Imperial College.
Type your Prolog command in the area below.
.....Ready.....
Current Command : p_trust_conflict(X).
Results :
-----
[ {X=.(.(p_i1,.(p_i2,[])),[])} ]
-----
Current Command : retractall(trust(_,_,_p_i2)).
Results :
-----
[()]
-----
Current Command : p_trust_conflict(X).
Results :
-----
[ {X=[]} ]
-----

```

Figure 8.2: Deletion then Source Trust Conflict for BMW

In Figure 8.2, the administrator has issued the command to delete rule *i2*. Then he re-executes the query and sees that there are no more source trust conflicts. This is a simple demonstration of how the process of simulation takes place.

8.2 Using SULTAN with Ponder

Ponder is a language for specifying security and management policies, while SULTAN is a language for specifying and analysing trust relationships. The two frameworks may be connected by either:

- Using SULTAN in Ponder policies.
- Using Ponder as the target for the refinement of SULTAN rules.

8.2.1 Using SULTAN in Ponder policies

Ponder policies may use SULTAN either:

- To check that someone be trusted/recommended, or
- To update experience information.

SULTAN rules as constraints

To explain this concept, a simple example is used. It is assumed that the following is the specification being used:

```

inst auth+ steve_harr {
subject Steve;
target Andrea;
action FinanceAdvice;
when (trust+(Harry, Andrea, FinanceAdvice(Andrea)) or
        (recommend+(Barclays, Andrea, FinanceAdvice(Andrea) ));
}

```

In the above code segment (and all the others in this chapter), terms in bold represent Ponder reserved words and terms in bold and italicized represent SULTAN reserved words. The above policy states that subjects in the Steve domain are permitted to perform the action FinanceAdvice on target objects in the Andrea domain when entities in the Harry domain trust entities in the Andrea domain to perform FinanceAdvice(Andrea) or when entities in the Barclays domain recommend entities in the Andrea domain to be trusted to perform FinanceAdvice(). It should be noted that the *when* clause contains SULTAN trust constraints (in the SULTAN notation). This example only shows the use of the trust+ and recommend+ functions as constraints. The constraint could have been any trust or recommend constraint (including, trust- and recommend+). For enforcement of this policy, the Ponder system will make a call to the SULTAN Analysis Tool to query whether the particular constraint is true or false.

Ponder updating experience information

A Ponder policy specification may wish to update experience information as a result of the enforcement of some policy. When an action is completed, an event can be generated to represent the successful completion of this action. Let's assume that the Print_Monitor is obliged to increase the level of trust between Deandra and Diane once a print action is successfully completed. This may be specified in Ponder as:

```

inst oblig+ inc_d {
subject Print_Monitor;
target TrustMonitor;
on print_complete(Deandra, Diane);
do increase(exp, Deandra, Diane, print(_), 5);
}

```


The increase function generates a call to the SULTAN Monitor. For the remaining uses of the SULTAN TMF, the BMW example (presented in Chapter 3) is used to illustrate them.

8.2.2 Refinement of SULTAN rules to Ponder policies

The SULTAN system operates at a higher layer of abstraction than Ponder. Thus, SULTAN rules can be refined into Ponder policies. The refinement process involves translating a SULTAN specification into a variety of Ponder policies (obligation, authorisation, etc.). The refinement process will entail stipulating what should be done with the level of a SULTAN rule and determining how constraints and entities should be treated (translated verbatim or mapped to more meaningful low-level functions). The value of the first parameter of an action will also be used to determine the assignment of subject and target domains of a Ponder policy. The difficulty in refining a SULTAN rule to a Ponder policy lies in maintaining the meaning of the rule. Is it possible to specify a semantically equivalent Ponder policy of a SULTAN trust rule? Probably not, given that Ponder is closer to the implementation layer than SULTAN. The refined policy will be more specific (and more closely linked to access control). It should be noted that the Ponder equivalent might require additional statements to ensure that the translation is as close as possible in its intent. The steps involved in refining a SULTAN specification into Ponder policy specification are:

- Refine the principals (the subject, the target, the action set and the constraint set).
- Determine the policy type.
- Define the meaning to be associated with the level.
- Generate the necessary Ponder policies.

The aforementioned rules are purposefully high-level and indicate that automated translation of SULTAN rules to Ponder policies is not possible. This is due to the abstract nature of the SULTAN language. Abstract concepts will have to be translated and this translation has to be driven by a human. It should also be noted that the problem of refinement is a difficult one, and thus the problem of refining from SULTAN to Ponder will also be inherently difficult. However, to illustrate that refinement is possible an example is used.

```
// Simple Access Example
SimpleAccess: trust(Deandra, Darren, use_printer(Deandra_Printers), 10 );
```

Assuming that a trust level of 10 does not lead to the inclusion of any extra constraints in the Ponder policy, the refined Ponder policy would be:

```
// Simple Access Example
```

```
inst auth+ SimpleAccess { subject Darren; target Deandra/printers; action use_printer(); };
```

In the above example, it is assumed that Deandra_Printers is refined into Deandra/printers. Now, let's look at the refinement of two simple SULTAN distrust statements. The first states that WebServer distrusts External to perform edit(WebServer, WebPages). In the SULTAN notation, this may be represented as:

```
// Simple Distrust Example 1
SimpleDistrust: trust(WebServer, External, edit(WebServer, WebPages), -10 );
```

Given a similar assumption about the trust level as used in the previous example, the refined Ponder policy is:

```
// Simple Distrust Example 1
inst auth- SimpleDistrust { subject External; target WebServer, action edit(WebPages); }
```

Our second simple distrust example states that Ann distrusts BuggySw site for downloads. This is written as:

```
// Simple Distrust Example 2
SDE2: trust(Ann, BuggySw, download(BuggySw), -10 );
```

It is not possible to install authorisation policies on BuggySw, so this SULTAN statement should be modelled as a Ponder refrain policy. The refined policy should be written:

```
// Simple Distrust Example 2
inst refrain SDE2 {subject Ann/PC; target BuggySw; action download(); }
```

The interesting point about the above two examples is that the trustor corresponds to the target in the first and the subject in the second.

To show that refinement from SULTAN to Ponder is not a trivial task, more difficult examples will be discussed.

```
/* A Little More Difficult */
MGR: trust(AdminMan, PayrollMan, sign(Cheque), 10 ) ← associated(Cheque, Company).
```

Rule MGR states that AdminMan trusts PayrollMan at trust level 10 to perform sign(Cheque) if associated(Cheque, Company) is true. In translating this to Ponder, a statement that says that PayrollMan is authorised to perform sign on Cheque (the target – because the first parameter of an action name is the location of the action) if associated(Cheque, Company) is true is constructed. In Ponder, this would be:

```
/* A Little More Difficult */
inst auth+ MGR {
  subject PayrollMan; target Cheque;
  action sign; when associated(Cheque, Company); }
```

Suppose Dan is trusted to perform the print and configure functions at trust level 100 if DSEAdmin(Dan) is true. In the SULTAN notation, this is:

```
/* A More Difficult Example */
Diff : trust(DSE, Dan, print(PRINTERS):configure(PRINTERS), 100 ) ← DSEAdmin(Dan);
```

Applying the rules discussed above to perform the refinement may result in the following steps:

Step 1: *Refine the principals*

For this example, it is assumed that entity names are translated verbatim. However, if the SULTAN specification contains highly abstract principals, it is likely that each of these principals would have to be mapped to lower level principals by the system administrator.

Step 2: *Determine the policy type*

Intuitively, it is clear that this example requires an authorisation policy. However, if the example is sufficiently hard, then the patterns presented above may be used to determine the particular Ponder policies required. It should be noted that a mapping from SULTAN rule patterns to their corresponding Ponder equivalents has not yet been done.

Step 3: *Define the meaning to be associated with the level.*

For this example, a trust level below 50 necessitates that an entity be checked for an additional security credential and that a trust level above 50 means that this check is not done.

Step 4: *Generate the necessary Ponder policies.*

The final step is simply putting all the pieces together.

```
/* A More Difficult Example */
inst auth+ Diff { subject Dan; target PRINTERS;
  action print, configure;
  when DSEAdmin(Dan); }
```

Note that this example is still relatively very simple. Automatic generation will not always create one authorisation for each SULTAN trust rule. A network of policies (both authorization and obligatory) may often be required in refining a SULTAN rule. For complex examples, the refinement process becomes more difficult. For example, if there is a SULTAN rule that is modelling ‘delegation’ trust, then it might be refined into a Ponder delegation policy. Thus, the context of a rule is an important issue in the refinement process. An added problem is the treatment of SULTAN recommend rules. Currently there is no direct (or indirect) representation for SULTAN recommend rules in Ponder. To solve this problem, rules can be created that infer trust rules from recommend rules and refine the inferred trust rules. The process described above may be used to translate SULTAN into the lower-level trust notations.

8.3 Negotiation

The negotiation process between a client and a producer determines whether or not a business relationship should be set up. This process governs the exchange of credentials [134], bargaining over price and possibly quality of service. BMW might be approached by another content provider MusicStore. Initially, MusicStore would only have a low default trust level, so BMW would take a cautious approach. However, as part of the credential exchange, MusicStore provides a signed recommendation from PMW. This recommendation is inserted into BMW's trust spec and now there is a match with a trust specification based on a PMW recommendation, so the trust level of MusicStore is now medium.

8.4 Contract Evaluation

Eventually BMW has two potential contracts to choose from but decides it really only needs one more content supplier. When faced with a choice between a group of potential E-commerce partners, the SULTAN trust framework can be used to help in selecting the appropriate partner. AllMP3 has revealed that part of its content is outsourced from DodgyStores and BMW has experience information about DodgyStores. The manager of BMW queries the information store and Specification Server to give all recommendations, trust rules, experience and risk information relating to an entity and then inserts a new trust rule which effectively gives greater weight to bank recommendations than client recommendations.

8.5 Recommendation Formation

It is possible to explicitly specify recommendations in the SULTAN notation. However, humans may want to generate recommendations about an entity based on a current trust rating for that entity either within the same context or even with respect to a different context i.e. there are no existing trust rules for that specific context. For example, BMW gets a request for a recommendation of PMW as video content supplier but only has a trust rule related to music supply. BMW is not quite sure about PMW's ability to provide the data rate needed for video so only gives a low rating for a recommendation, which is inserted in the Specification Server.

8.6 Infrastructural Security

Trust-based decision-making can be used to configure the security of infrastructure. BMW sets up a contract with a local radio station to give unlimited access to all its music sources for a fixed monthly charge. A VPN connection is used to link the Radio station network and BMW, so the encryption can be disabled on sending music to the Radio station as the VPN provides a secure channel.

8.7 Access Control Decisions

The trust level of a trustee can be the basis of an authorisation decision for access to a trustor's resources or services by a trustee. A new customer of BMW is covered by the following rule, which allocates a default low trust level of 10.

```
DefCust: trust (BMW, _newCustomer, AccessMusic(ContentDatabase), 10);
```

A couple of Ponder authorisation policies can be used to give access to all music if the customer trust level is high but only to a subset if the trust level is low. A Ponder authorisation policy specifies access control for security. Positive authorisation policies specify the actions that a subject is permitted to perform on a target object, while negative authorisation policies specify the actions that a subject is forbidden from performing on the target object.

```
type auth+ Access ( domain SegmentofContentBase, string TrustValue){
  subject Client; target SegmentofContentBase; action AccessMusic();
  when (trust+(FrontEnd, ClientApp, AccessMusic(ContentDatabase), TrustValue )) };
inst auth+ AccessHigh = Access(/BMW/ContentBase, HighTrust);
inst auth+ AccessLow = Access(/BMW/ContentBase/Restricted, LowTrust);
```

This defines a Ponder policy type called Access with 2 parameters – the segment of the music to which access is to be permitted and a trust value. The constraint to the policy is in the form of a query to the SULTAN framework to determine whether the subject is trusted at the required level. Two policy instances are then created to cater for the high and low trust situations. This is an application of using SULTAN queries as Ponder constraints (to aid access control decisions).

8.8 Resource Allocation

BMW performs an audit of its records relating to customer use every six months. Based on the experience information on the customer and the trust level (and the changes in the trust level) of the customer, BMW decides whether to upgrade the customer's status and give the customer further discounts on all purchases.

8.9 Summary

This chapter presented typical instances where the SULTAN TMF may be used. The first instance discussed was simulation analysis, which involves the creation of 'What-if' scenarios. SULTAN rules may be used as constraints or as a source notation for refinement into lower frameworks. The components of the TMF may be used to help in the negotiation, contract evaluation and recommendation formation processes. The TMF may also help in the determination of infrastructure security, access control decisions and resource allocation.

Chapter 9 Case Study

*"A theory, ultimately, must be judged for its accord with reality."
Stanislaw Leshniewski (1886 - 1939), [62]*

This chapter describes a real business (based on [161]). However, names and figures have been modified in order to make the case study more generic and to not infringe on the company's copyright. The SULTAN TMF is incorporated into the study to demonstrate its applicability to Internet-based corporations.

9.1 Overview

ResWorld is an Internet-based distribution network for the hotel industry, which has its headquarters in the USA and offices in the United Kingdom and France. The client database consists of travellers, hotels and partners. Currently, ResWorld has a network that includes 50,000 properties, over 3,000 Web distribution partners, 80,000 travel agents and the global distribution systems Galileo, Sabre and Worldspan. The strategic objectives of ResWorld are:

- to generate additional revenue for hotels at a lower cost, through online marketing and distribution, and
- to provide consumers with a more efficient and effective shopping and booking experience.

9.2 The Players

A partner is a company that sees the benefit of establishing a symbiotic relationship with ResWorld. All partners receive a portion of the revenues generated by reservations booked online. Partners may connect to ResWorld through a range of methods, e.g. a simple link or full technical integration.

Member hotels may post content on ResWorld's website. This information includes detailed information and photos, which is distributed to partners or through their own Web site using the ResWorld network and technology.

Travellers, who are ResWorld members, may view travel web pages and or make online reservations. The company makes revenue every time a reservation is made through its system.

9.3 Processes and Infrastructure

Content for member hotels is stored in a central database, which is connected to ResWorld's network of partners. Hotels can update their content in real time. When this is done, the changes are automatically transmitted to all partners simultaneously. Reservation and database services are provided by UNIX machines and powered by Sybase database technology. Web services are provided on Windows NT machines and all transactions use the Secure Sockets Layer (SSL) protocol to allow for the encryption of any information that is transferred across the Internet. Digital client identification services are provided by VeriSign Inc. To manage the demands on ResWorld's system, a minimum of two web servers is present for each office, with more servers added as the load increases. High availability is enabled by using multiple parallel servers for each hardware component of the system. If server failure is detected, the load is shifted to another, using IP load distribution to route a user's request to the least busy server. These servers also allow repairs and upgrades while the system is operational.

The study detailed in this chapter will focus on ResWorld's American office, which will hereafter be referred to as ResAm. The American office consists of a team of managers, technical and administrative staff. The office also consists of web servers and database servers, which represents the bare minimum equipment listing for ResAm. Management is sub-divided into operational and strategic divisions, spearheaded by a General Manager. Technical staff is lead by the systems administrator and includes a coalition of database administrators, integration specialists and web services developers. Administrative staff includes administrative assistants and legal aides.

To illustrate the applicability of the TMF to Internet-based business, this chapter will outline the phases of the trust management lifecycle (Chapter 7) as it pertains to ResWorld.

9.4 Initialisation Tasks

The initialisation tasks are those that are performed by the systems administrator when the SULTAN system is being installed.

9.4.1 Organizational Chart Diagram

Figure 9.1 shows the organizational diagram that may be constructed from the information provided above.

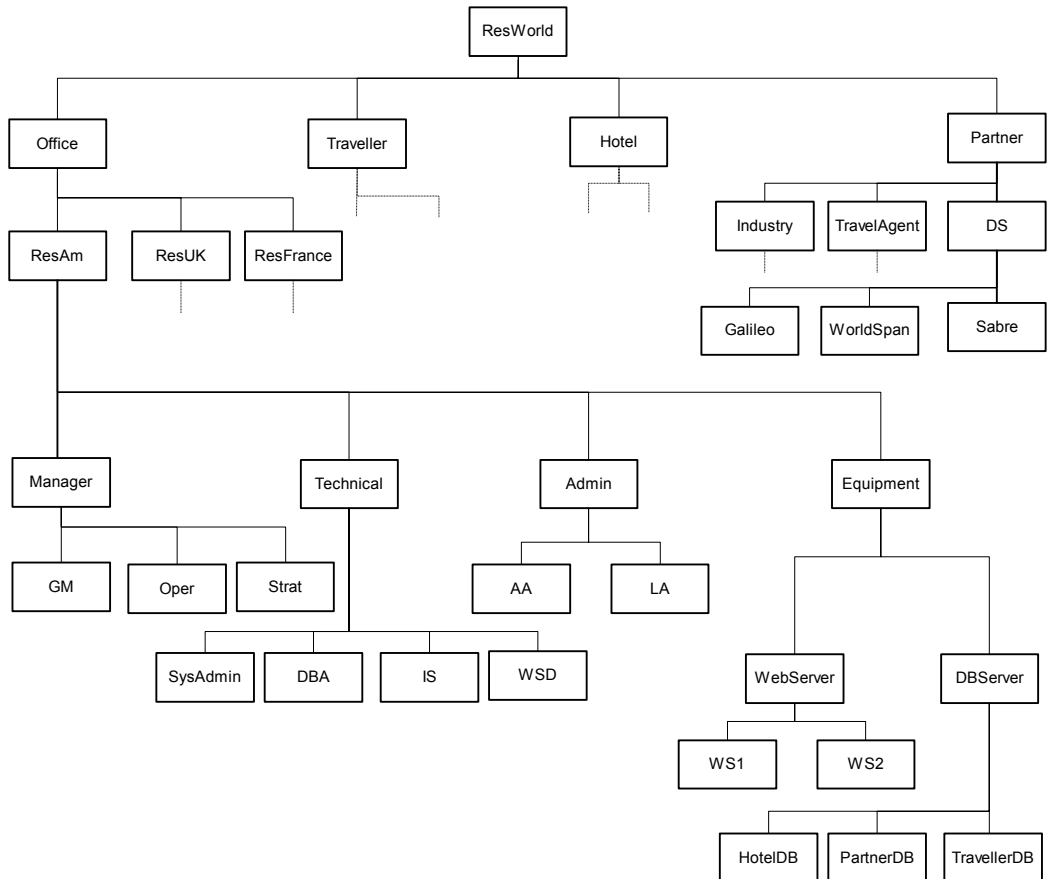


Figure 9.1: Organizational Chart for ResWorld

The symbols and their representations used in Figure 9.1 that are hard to decipher are presented in Table 9.1.

Symbol	Meaning
GM	General Manager
Oper	Operational Manager
Strat	Strategic Manager
DBA	Database Administrator
IS	Integration Specialist
WSD	Web Services Developer
AA	Administrative Assistant
LA	Legal Aide
WS1	Web Server 1
WS2	Web Server 2

Table 9.1: Key for ResWorld’s Organization Chart

The above diagram would lead to the construction of the following facts:

isPartOf(Office, ResWorld). isPartOf(Traveller, ResWorld). isPartOf(Hotel, ResWorld).
 isPartOf(Partner, ResWorld). isPartOf(ResAm, Office). isPartOf(ResUK, Office).
 isPartOf(ResFrance, Office). isPartOf(Manager, ResAm). isPartOf(Technical, ResAm).
 isPartOf(Admin, ResAm). isPartOf(Equipment, ResAm). isPartOf(GM, Manager).
 isPartOf(Oper, Manager). isPartOf(Strat, Manager). isPartOf(SysAdmin, Technical).
 isPartOf(DBA, Technical). isPartOf(IS, Technical). isPartOf(WSD, Technical).
 isPartOf(AA, Admin). isPartOf(LA, Admin). isPartOf(WebServer, Equipment).
 isPartOf(DBServer, Equipment). isPartOf(WS1, WebServer). isPartOf(WS2, WebServer).
 isPartOf(HotelDB, DBServer). isPartOf(PartnerDB, DBServer).
 isPartOf(TravellerDB, DBServer). isPartOf(Industry, Partner). isPartOf(TravelAgent, Partner).
 isPartOf(DS, Partner). isPartOf(Galileo, DS). isPartOf(WorldSpan, DS). isPartOf(Sabre, DS).
 isPartOf(Traveller, foreign). isPartOf(Hotel, foreign). isPartOf(Partner, foreign).

All this information is stored in the Entity-Connections database, via the Mini-Editor of the SULTAN Specification Editor.

9.4.2 Asset Repository Construction

The asset listing has been purposefully restricted for this study. Currently, there are five assets, i.e. the web servers and the databases. As the web servers merely provide an interface to the data and implement server access control logic, their value is determined from the cost of the machines and the software running on them. This is approximated to £4,000. The monetary value of the databases is surmised from the potential worth of the data in the databases. Since the database contains sensitive information, their initial value is assumed to be £12,000. This means that the total value of all the assets is £44,000 (£4,000 + £4,000 + £12,000 + £12,000 + £12,000). This information would necessitate the construction of an asset repository with the following information:

rTotal(44000). resource(WS1, 0.0909). resource(WS2, 0.0909). resource(HotelDB, 0.2727).
 resource(PartnerDB, 0.2727). resource(TravellerDB, 0.2728)

Note that the contribution of TravellerDB is rounded up to ensure that the proportions sum to one (to be consistent with the theory presented in Chapter 5).

9.4.3 Risk Profile Data

To specify risk profiles, the set of actions that is to be used in the trust and recommend specifications must be known. To ascertain these actions, the options available to the different

parties and the system administrator's comprehension of the functions of the members of the organizations are used. ResWorld's website states:

We offer the following services:

For travellers	For hotels	For Partners
Find, see and book a hotel room. Get the best discounts in your favourite destinations, including distinctive hotels, inns and resorts worldwide at www.PlacesToStay.com	Join ResWorld or upgrade your current ResWorld services to make your hotel bookable to millions of worldwide consumers and travel agents.	ResWorld offers leading Web sites, travel agencies and call centers the largest and most unique selection of hotels, bed & breakfasts and inns.

Technical Support is provided for any problem.

A quick scan of the services provided by ResWorld to its clients and the outline presented in earlier sections of this chapter reveal that the following actions will be required:

Action Signature	Interpretation
send_payment(Obj, Portion, ToP, TransId)	Sends payment, Portion, to ToP for transaction TransId.
view_info(Obj, Id)	Entity Id views information.
post(Obj, MemId, Cont)	Entity MemId posts content Cont.
update(Obj, MemId, Cont)	Entity MemId updates Cont.
search(Obj, MemId, ForText)	MemId searches Obj using ForText.
book(Obj, MemId)	MemId books a room.
send_info(Obj, MemId)	Obj sends new information to MemId.

Table 9.2: Initial set of actions for ResWorld

Since the perspective taken by the system administrator is from the point of view of the American Office (i.e. ResAm), the risk profiles will be defined for ResAm. Actions relating to bookings and the transmission of information and or monetary units are assigned higher Maximum Allowable Losses and lower Risk Thresholds. The following represents an initial, administrator-assigned risk profile repository:

rprofile(ResAm, send_payment(_O, _P, _T, _TID), 500, 0.15).

rprofile(ResAm, view_info(_O, _Id), 100, 0.50). rprofile(ResAm, post(_O, _Mid, _C), 200, 0.20).

rprofile(ResAm, update(_O, _Mid, _C), 200, 0.30).

rprofile(ResAm, search(_O, _Mid, _F, _R), 100, 0.65).

rprofile(ResAm, book(_O, _Mid), 600, 0.10).

rprofile(ResAm, send_info(_O, _Mid), 250, 0.30).

It should be noted that there isn't a rigid linear progression of activities. It is often the case that initialisation and specification occur simultaneously.

9.5 Specification

The trust relationships for ResAm can be ascertained from the description of the business, which was provided earlier in this chapter, and from the information garnered from executing the initialisation tasks. In this section, all the trust relationships involving ResAm will be outlined. These relationships are:

1. ResAm's partners trust ResAm to send a portion of a reservation revenue, once a booking is made.
2. ResAm's partners trust ResAm to send information once it has been posted or updated on ResAm's site.
3. ResAm trusts its partners to responsibly view the information in ResAm's databases. (the partnership integration agreement).
4. ResAm trusts its member hotels to post and update content.
5. ResAm trusts Traveller to responsibly search, view and make bookings through their web system.

In addition to the above trust relationships, the administrator knows that there are other relationships that relate to the people in the organization, namely:

6. Technical staff are trusted to configure and maintain the equipment. They are also trusted to produce status reports for the equipment.
7. Database Administrators are distrusted to configure and maintain web servers
8. Web Service Developers are distrusted to configure and maintain the database servers.
9. Administrative Assistants are trusted to check client information, once they have authorisation from a manager.
10. Managers are trusted to read equipment status reports and view equipment settings.

The above is a snapshot of (some of) the organizational trust relationships that exist in this application domain. These relationships necessitate the use of additional actions, namely:

Action Signature	Interpretation
view_sett(Obj, Id)	Id views the settings of Obj.
update_sett(Obj, Id, NewSet)	Id updates the settings for Obj with NewSet.
reset(Obj, Id)	Id resets the settings for Obj.
create_srep(Obj, Id)	Id creates a status report for Obj
read_srep(Obj, Id)	Id reads a status report for Obj
check(Obj, Id, ClientID)	Id checks the information entered by ClientID

Table 9.3: Additional actions for ResWorld

It is assumed that the risk profiles for these actions will be:

rprofile(ResAm, view_sett(_O, _ID), 150, 0.20).

rprofile(ResAm, update_sett(_O, _Id, _NS), 300, 0.10). rprofile(ResAm, reset(_O, _Id), 400, 0.10).

rprofile(ResAm, create_srep(_O, _Id), 50, 0.70). rprofile(ResAm, read_srep(_O, _Id), 100, 0.70).

rprofile(ResAm, check(_O, _Id, _C), 100, 0.50).

The SULTAN specifications correlating to the trust relationships broadly defined above are:

1. PPay : **trust**(Partner, ResAm, send_payment(WebServer, _Portion, Partner, _T_Id), 100)
 ← booking_made(_T_Id) & portion(Partner, _T_Id, _Portion);
 Partner trust ResAm to perform send_payment(WebServer, Portion, Partner, T_Id) at trust level 100 if the booking was made for transaction T_Id and Portion is the Partner share of the transaction.
2. PSend : **trust**(Partner, ResAm, send_info(WebServer, _P_Id), 100)
 ← is_security_token(_P_Id, Partner);
 Partner trust ResAm to perform send_info(WebServer, _P_Id) at trust level 100 if _P_Id is the security token for Partner. This token is assumed to be a credential.
3. PView : **trust**(ResAm, Partner, view_info(DBServer, _PartId), 50)
 ← is_security_token(_PartId, Partner);
 ResAm trusts Partner to perform view_info(DBServer, _PartId) at trust level 50 if _PartId is the security token for Partner.
4. HCont: **trust**(ResAm, Hotel, post(WebServer, _H_Id, _Cont) :
 update(WebServer, _H_Id, _Cont), 100)
 ← is_security_token(_H_Id, Hotel) & is_valid(_Cont, _H_Id);
 ResAm trusts Hotel to perform post(WebServer, _H_Id, _Cont) and update(WebServer, _H_Id, _Cont) at trust level 100 if is_security_token(_H_Id, Hotel) and is_valid(_Cont, _H_Id) are both true. Note that is_valid(_Cont, _H_Id) represents the fact that validity checks have been performed on _Cont to ensure that its integrity has not been compromised and that it was actually sent by _H_Id.
5. Table1 : **trust**(ResAm, Traveller, search(WebServer, _T_Id, _Text), 100)
 ← is_security_token(_T_Id, Traveller) & is_valid(_Text, T_Id);
 ResAm trusts Traveller to perform search(WebServer, _T_Id, _Text) at trust level 100 if is_security_token(_T_Id, Traveller) and is_valid(_Text, T_Id) are true.
 Table2 : **trust**(ResAm, Traveller, view_info(WebServer, _T_Id):book(WebServer, _T_Id), 100)
 ← is_security_token(_T_Id, Traveller);
 ResAm trusts Traveller to perform view_info(WebServer, _T_Id) and book(WebServer, _T_Id) at trust level 100 if is_security_token(_T_Id, Traveller) is true.
6. TechAble : **trust**(ResAm, Technical, view_sett(Equipment, _T_Id) :
 update_sett(Equipment, _T_Id, _NewSet) :
 reset(Equipment, _T_Id) : create_srep(Equipment, _T_Id), 100)
 ← is_security_token(_T_Id, Technical);
 ResAm trusts Technical to perform view_sett(Equipment, _T_Id), update_sett(Equipment, _T_Id, _NewSet), reset(Equipment, _T_Id) and create_srep(Equipment, _T_Id) at trust level 100 if is_security_token(_T_Id, Technical) is true.
7. DBANotAble : **trust**(ResAm, DBA, view_sett(WebServer, _D_Id) :
 update_sett(WebServer, _D_Id, _NewSet) :
 reset(WebServer, _D_Id) : create_srep(WebServer, _D_Id), -100);
 ResAm distrusts DBA to perform view_sett(WebServer, _D_Id), update_sett(WebServer, _D_Id, _NewSet), reset(WebServer, _D_Id) and create_srep(WebServer, _D_Id) at trust level -100.
8. WSDAble : **trust**(ResAm, WSD, view_sett(DBServer, _W_Id) :
 update_sett(DBServer, _W_Id, _NewSet) : reset(DBServer, _W_Id) :
 create_srep(DBServer, _W_Id), -100);
 ResAm distrusts WSD to perform view_sett(DBServer, _W_Id), update_sett(DBServer, _W_Id, _NewSet), reset(DBServer, _W_Id) and create_srep(DBServer, _W_Id) at trust level -100.

9. AACheck : **trust**(ResAm, AA, check(DBServer, _A_Id, _C_Id), 100)
 ← is_security_token(_A_Id, AA) & auth(_M_Id, Cid) &
 is_security_token(_M_Id, Manager);

ResAm trusts AA to perform check(DBServer, _A_Id, _C_Id) at trust level 100 if is_security_token(_A_Id, AA), auth(_M_Id, Cid) and is_security_token(_M_Id, Manager) are true.

10. MEquip : **trust**(ResAm, Manager, read_srep(Equipment, _M_Id) :
 view_sett(Equipment, _M_Id), 100)
 ← is_security_token(_M_Id, Manager);

ResAm trusts Manager to perform read_srep(Equipment, _M_Id) and view_sett(Equipment, _M_Id) at trust level 100 if is_security_token(_M_Id, Manager) is true.

Entities that are a part of the organization are linked to the entity names in the specifications by the entity-connections update facility of the SULTAN Monitor. For foreign entities (discussed in Section 3.4), a trust rule must be included to link these unknown parties to entities in our domain. An alternative would be to create default trust relationships for foreign entities and replace all entities that are a part of foreign with SULTAN variables. This would imply that these variables will be checked in the constraints. However, since VeriSign provides the client identification services, VeriSign certificates can be used to establish the client's role (i.e. Traveller, Hotel, Partner). The trust rule that allows this binding is:

11. ForeignBind : **trust**(ResAm, _Y, _A, _L)
 ← verified_as(_Y, VeriSign, _R) & isPartOf(_R, foreign) & trust(ResAm, _R, _A, _L);

ResAm trusts/distrusts an entity _Y to perform _A at trust level _L if _Y is verified by VeriSign as having a certificate for role _R and _R is a foreign entity and _R is involved in a trust/distrust rule involving ResAm, _R, _A and _L.

Figure 9.2 shows the compilation results of the specifications discussed above. The next generation of the SULTAN compiler will link the organizational chart into the compilation process. This will eliminate the semantic warnings generated in the results of Figure 9.2.

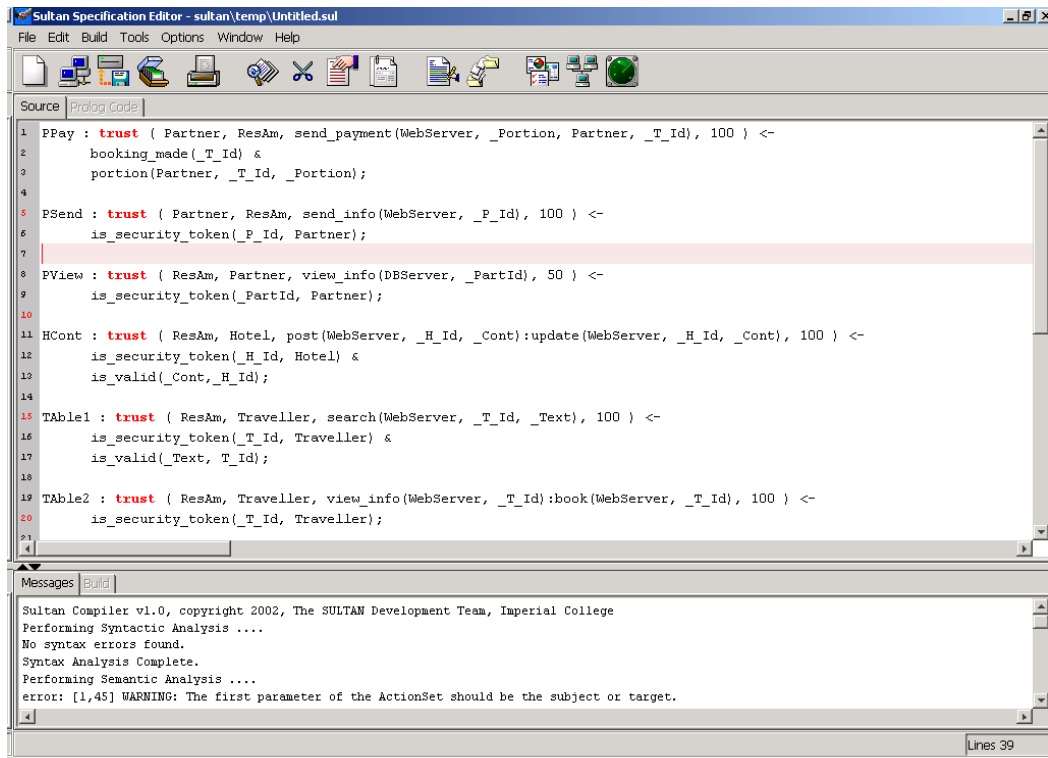


Figure 9.2: Compilation Results for ResWorld

9.6 Analysis

In this section, three analysis queries will be described. The Analysis Tool will be used to see the results of each query and the necessary adjustments to the specifications presented. As these are the questions that are asked at installation-time, they will be restricted to source analysis queries. As a part of the information collection discussion in the next section, scenario queries will be presented. Queries will be manipulating translated versions of the specifications, state and organisational chart information. Thus, this discussion will highlight queries in both the SULTAN form and their Prolog translated form.

The first query is a standard trust conflict. This will identify if there are trust and distrust relationships between two entities for the same context. Ordinarily, this query may be specified as:

$$\text{query}([T,D], (\text{p_pos_trust}(T), \text{p_neg_trust}(D), \text{p_trustor}(Tr,T), \text{p_trustor}(Tr,D), \\ \text{p_trustee}(Te,T), \text{p_trustee}(Te,D), \text{p_commonAS}(T,D)), \text{Result}).$$

Since this generic conflict is specified in the template, the template call that may be used is:

$$\text{p_trust_conflict}(\text{Result}).$$

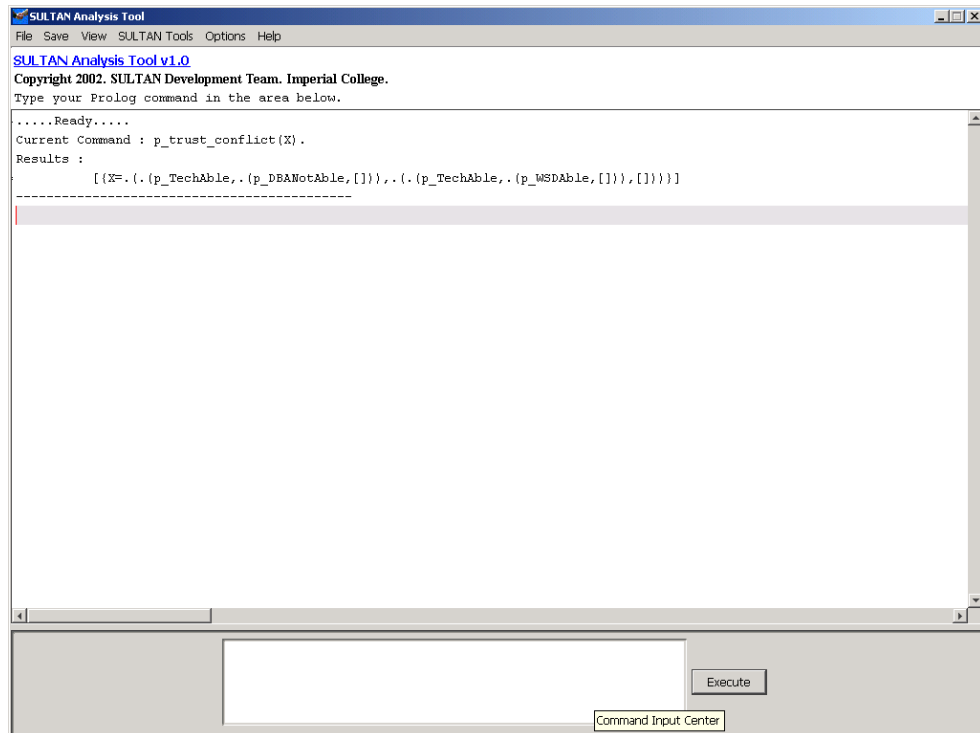


Figure 9.3: Analysis Result for Source Trust Conflict Query

From the specifications, it is noticed that technical staff is trusted to configure and maintain the firm's equipment, yet DBAs are not trusted with the Web Server and Web Service Developers are not trusted with the databases. The source conflict is evident in the results of Figure 9.3, which shows two conflicting pairs of relationships, namely 1) TechAble & DBANotAble, and 2) TechAble & WSDAble. To ensure that this conflict does not occur in actual scenarios, the administrator needs to simply modify the constraints of TechAble to include checks that ensure that the relationship does not hold for DBAs (with respect to the Web Servers) and Web Service Developers (with respect to the Databases). Such a change may be:

```
TechAble : trust(ResAm, Technical, view_sett(Equipment, _T_Id) :
            update_sett(Equipment, _T_Id, _NewSet) :
            reset(Equipment, _T_Id) : create_srep(Equipment, _T_Id), 100)
← is_security_token(_T_Id, Technical) &
  ( ( is_not(Technical, WSD) & is_not(Equipment, DBServer) ) |
    ( is_not(Technical, DBA), is_not(Equipment, WebServer) ) );
```

This inclusion will not change the result of the source trust conflict query, but will ensure that scenarios will not lead to this conflict. The next query that will be asked is a conflict of interest query. This time the administrator chooses to formulate his own conflict of interest query, namely:

```
query( [T1,T2, Te], ( p_pos_trust(T1), p_pos_trust(T2), T1 \== T2, p_trustee(Te,T1),
                    p_trustee(Te, T2), p_trustor(Tr1,T1), p_trustor(Tr2,T1),
```


$Tr1 \neq Tr2, p_commonAS(T1,T2)), Result).$

This particular query is looking for trust relationships, with the same trustee, different trustors and a common set of actions, i.e. which positive trust rules involve someone who is trusted by different people to perform the same task(s)?

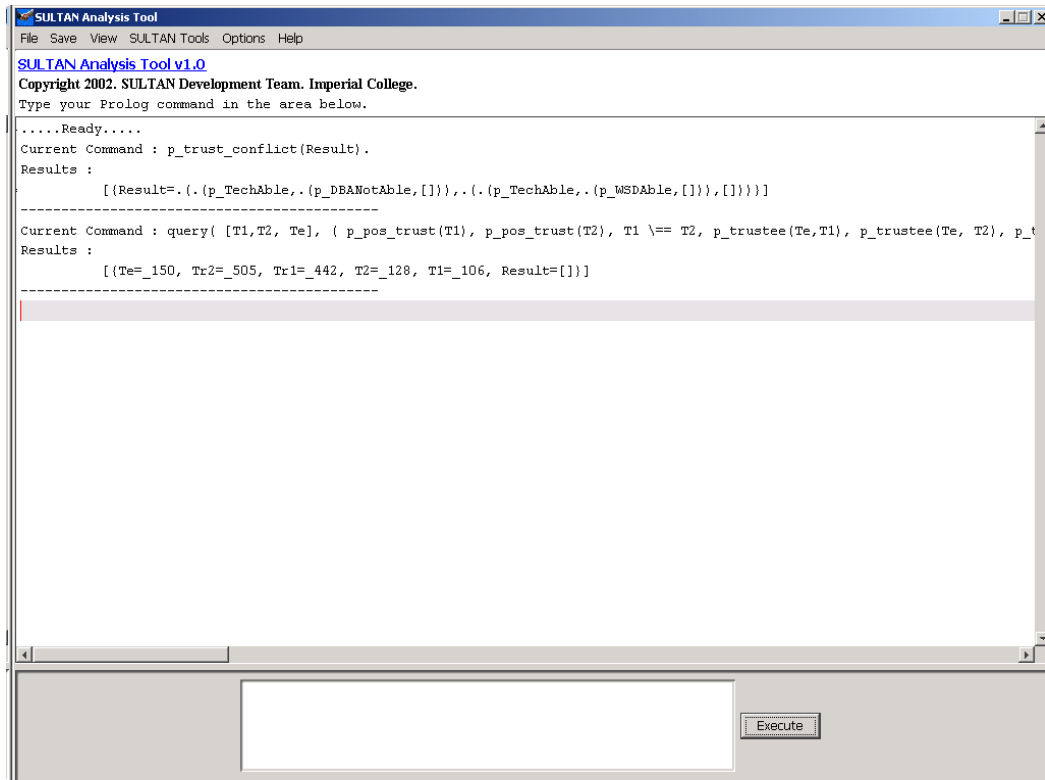


Figure 9.4: Analysis Result for Source Conflict of Interest Query

From Figure 9.4, it can be seen that the Result variable is empty. Thus, there are no source conflicts of interest. The rationale behind the answers of the other variables in the query are due to the Prolog unification algorithm and may provide some useful insight. For example, the fact that all of our auxiliary variables have been instantiated to Prolog variables suggests that a rule with variables for the trustor, trustee and rule-name were used to determine Result. Since the specification is relatively small, we know that the rule in question is ForeignBind. For a source query, ForeignBind will match every other rule in our specification (except PPay and PSend, which have a different trustor). However, the inequality test in the conflict of interest statement ensures that such variable-entity pairings do not appear in the result.

The final query to be examined in this section is a source separation of duties conflict. The administrator knows from earlier conversations with the General Manager (GM) that no one should be trusted to update the settings of the Web Servers and view information on the

Partners. The GM believes that disgruntled employees may be induced into committing corporate espionage for the right price. To specify such a query, the template may be used. The statement to be used would be:

```
p_separation_of_duties(Entity, [f_update_sett(e_WebServer, _)],
                        [f_view_info(e_PartnerDB, _)], Result).
```

The above would be translated to:

```
query( [T1,T2], ( p_pos_trust(T1), p_pos_trust(T2), T1 \== T2, p_trustee(Entity,T1),
                p_trustee(Entity, T2), p_actions([f_update_sett(e_WebServer, _)], T1),
                p_actions([f_view_info(e_PartnerDB, _)], T2) ), Result).
```

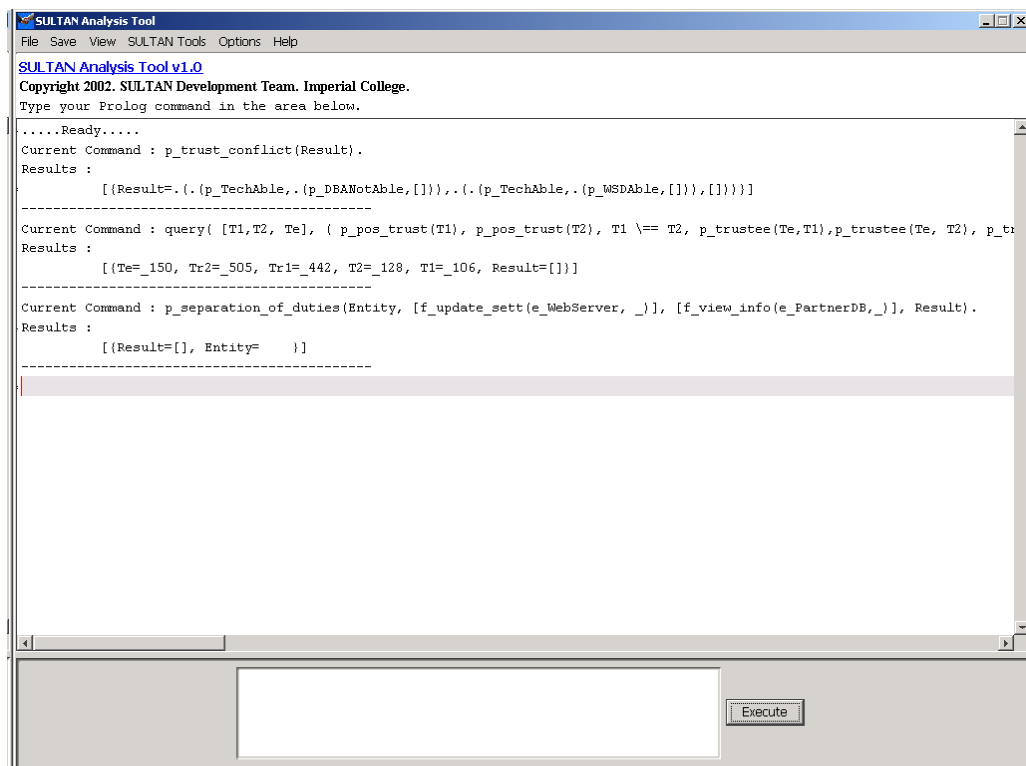


Figure 9.5: Analysis Result for Source Separation of Duties Query

From Figure 9.5, there is no separation of duties conflicts with respect to the source. This completes the initial discussion on trust analysis. Now, the focus moves to establishing the mechanisms necessary to collect information and how this information may be used. To illustrate the next two sections, which highlight the SULTAN Monitor and Consultant, the administrator's tools will be used. This is because the Monitor and Consultant connect to user applications via socket messages and have no graphical interface.

9.7 Information Collection

To enable the collection of information (i.e. relating to state, risk, experience, entity connections, action dependencies and risk profiles), the SULTAN Monitor (SM) client will be installed on all the machines in the domain. This implies that a SM Client must be present on all the machines used by ResWorld's staff and on the web and database servers. Each machine is installed with its associated entity name in the local store of the SM Client. For the purposes of this study, the processes involved in setting up the SM Client on WS1 will be discussed.

As stated in Chapter 6, the SULTAN Monitor is based on a passive design architecture. Thus, the administrator decides when information is to be sent. Since, he knows the constraints that need to be monitored, there is no problem in determining what is to be sent to the Monitor. After modifying the software running on WS1 to send constraint information to the SM Client, the following are the tasks performed:

- After the verification of a client, a `verified_as` fact and an `is_security_token` fact are sent to the Monitor.
- At the end of a SSL transaction, a `booking_made` and an experience record are generated.
- When hotel content is received, an `is_valid` record is sent to the Monitor.
- Generates an `is_registered` record for newly registered members.
- When a booking cancellation is performed, the associated `booking_made` record is deleted, a `booking_cancelled` record is created and a negative experience is noted.
- The SULTAN Monitor Server automatically signals the State Information Database to delete the associated `verified_as` fact when a member logs out.

Suppose that Jane is going on vacation in Macau and needs to book a room for seven nights. She has heard of ResWorld's reputation for providing quality, low-cost accommodation and decides to try them out. She enters her personal information, which generates an `is_registered(_JaneID, Traveller)`, and starts viewing the available hotels in Macau. At this point, a VeriSign certificate is issued to Jane's machine (and `verified_as` and `is_security_token` facts are sent to the Monitor). At this point, the State Information Database has the following information:

```
is_registered(C190123, Traveller).
verified_as(C190123, VeriSign, Traveller).
is_security_token(C190123, Traveller).
```

When each of these facts is sent to the Monitor, the template of conflicts and ambiguities is re-evaluated. Figure 9.6 shows the results of the re-evaluation process at the point when these three facts are in the State Information Database.

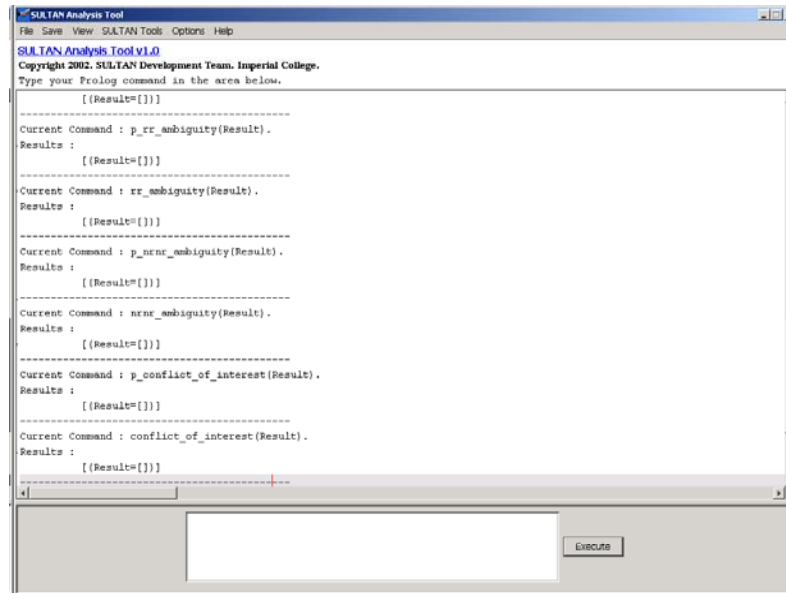


Figure 9.6: Re-evaluation Results

From Figure 9.6, it is clear that there are no conflicts or ambiguities arising from the addition of the state information. However, if the result of any of the queries being executed returns a non-empty set, then the re-evaluation flag is set (in the common configuration file for the administrator's tools) and the queries involved added to a list of potential problems file. This file and flag are reset when the administrator runs a tool and looks at the (potential) problems.

When Jane finds a room and makes a booking, a `booking_made` fact is sent to the Monitor and a positive experience record generated. When she logs out, the `verified_as` fact is retracted. After her success, Jane tells her two sisters about ResWorld. They coax her to let them book their vacation accommodation using Jane's account at ResWorld. They get their way and in the process generate two new booking and experience records. Due to unforeseen circumstances, Jane has to cancel her booking. Thus, a negative experience record is generated for the transaction. Thus, our State Information Database has the following contents:

```

is_registered(C190123, Traveller). is_security_token(C190123, Traveller).
booking_made(C190123_11903p1400_Macau).
experience(ResWorld, C190123, book(WS1_20, C190123_11903p1400_Macau), 100).
booking_made(C190123_12903a0900_Hawaii).
experience(ResWorld, C190123, book(WS1_20, C190123_12903a0900_Hawaii), 100).
booking_made(C190123_12903a0925_Florida).
experience(ResWorld, C190123, book(WS1_20, C190123_12903a0925_Florida), 100).
experience(ResWorld, C190123, book(WS1_20, C190123_11903p1400_Macau), -20).

```

A value of -20 is used for the experience value of the cancellation to demonstrate the relative insignificance of the cause of this record. It is assumed that a negative experience record that is due to a more serious cause, such as the failure to pay, would have a larger negative value. The focus moves on to how all this information may be used by the workers of ResWorld (or their software proxies).

9.8 Application Use

For each machine in the system to take advantage of the information in the SULTAN system, the SULTAN Consultant must be installed on every machine in ResWorld's organisation. The SULTAN Consultant is integrated into an application just as the SULTAN Monitor is. As stated in Chapter 7, the questions that can be asked are:

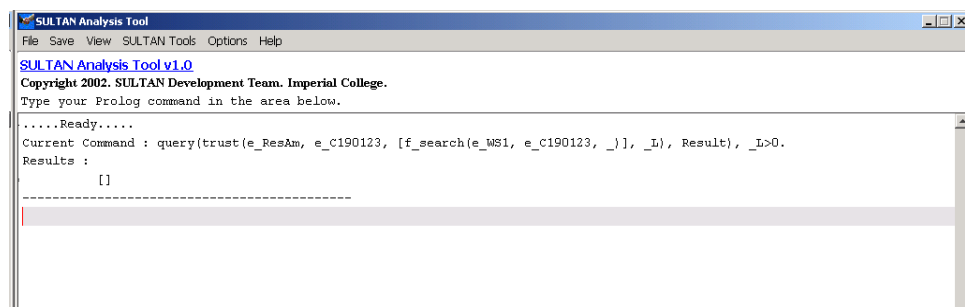
- Should I trust a target entity in a particular context?
- Should I recommend an entity to be trusted in a particular context?
- What is the risk involved in a transaction, w.r.t. target Y and action A?
- What is the experience w.r.t. target Y and action A?

In this section, a trust and experience consultant query will be discussed. The underlying query (made in the Analysis Tool) and the results will be presented. When Jane logs back into the system, the application may ask the question 'Should I trust her to search the web pages?' The application sends the following tuple to the STC client:

```
(trust, C190123, search(WS1, C190123, _))
```

This translates into the following question that is to be executed by the Analysis Tool:

```
query(trust(e_ResAm, e_C190123, [f_search(e_WS1, e_C190123, _)], _L), Result), _L>0.
```



The screenshot shows a window titled "SULTAN Analysis Tool" with a menu bar (File, Save, View, SULTAN Tools, Options, Help). Below the menu bar, it displays "SULTAN Analysis Tool v1.0" and "Copyright 2002. SULTAN Development Team. Imperial College." It prompts the user to "Type your Prolog command in the area below." The command entered is:


```
.....Ready.....
Current Command : query(trust(e_ResAm, e_C190123, [f_search(e_WS1, e_C190123, _)], _L), Result), _L>0.
Results :
[]
```

 The results area shows an empty list `[]`.

Figure 9.7: Translated Trust Consultation Query

This is a typical abduction question, which returns the set of predicates that need to be proven for the rule to be true. In this case, the result is empty, thus the consultant will return yes and the name of the rule to the calling application.

Suppose the web server is modified to use past experience to augment the process of deciding who is allowed access to the system. The policy could be that a negative cumulative experience with a trustee would override the VeriSign authentication tokens. Thus, if Jane decides to use ResWorld's system when this policy is in place, then the application may ask the question:

(exp, C190123, _)

This is translated to a special command that calculates the weighted average of the experience records, which is:

getExp(e_ResAm, e_C190123, V__, Value).

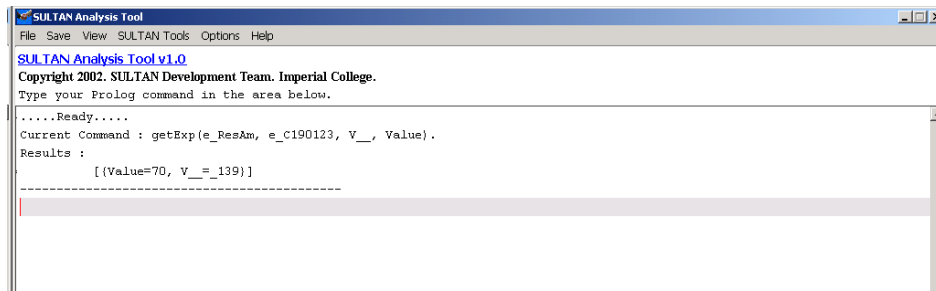


Figure 9.8: Translated Experience Consultation Query

From Figure 9.8, the experience rating for Jane over her period of use has an experience value of 70, which indicates that ResAm has had an overall positive experience. With this information and the VeriSign certificate, the web server can decide that Jane has earned the right to use the system once again. If for some reason, the overall experience w.r.t. Jane becomes negative, the web server logic may dictate that Jane has become (or is becoming) a bad customer and decide not to allow Jane to use the system. Thus, applications can use the SULTAN system to evaluate client's behaviour (i.e. interactions and outcomes) to determine whether they wish to continue doing business with them or whether they should look at limiting the business areas available to the client.

9.9 Summary

This chapter presented the case study of an Internet-based business, ResWorld, which provides online, real-time reservation system. ResWorld is primarily concerned with business-to-business operations, but also offers direct access to the consumer and a low-cost marketing platform for smaller hotels and resorts. The phases of the trust management lifecycle were applied to ResWorld and in each phase, it was shown how the SULTAN TMF can be incorporated and used. From the chapter's discussion, it is demonstrated that the current prototype can be useful to the administrator after he has performed the necessary configurations.

Chapter 10 Critical Evaluation

“It is only through evaluation that value exists: and without evaluation the nut of existence would be hollow.”

– Friedrich Nietzsche (1844–1900), German philosopher [162]

This chapter details the limitations and deficiencies of the SULTAN TMF and highlights areas of improvement. Firstly, the TMF is criticised in the context of related work.

10.1 Relationship to Related Work

The SULTAN Trust Management Framework emerged from work on trust models and trust management systems (Chapter 2). All these fields concentrate on very specific issues, which tend to be application-specific and security-biased. The SULTAN framework approaches trust management from an application-independent, security non-specific perspective.

Logic-based trust models have focused on authentication protocol specification (BAN Logic [46]), authentication rules (ASL [88]), modalities [89, 90], transmission reliability [91], agent communication message history [92] and subjective proposition beliefs [96-98]. The shared theme in these models, with the exception of Josang’s Subjective Logic, is the encoding of a scenario into a notation, defined by the creator of the formalism, and the use of logic reasoning to determine if the scenario adheres to particular (logical) properties. In these models, the analysis of a scenario to determine if a logical property holds is done by a human expert. With Josang’s Logic, the model is concerned solely with specifying and reasoning about Subjective Opinions. The specification and analysis components of the SULTAN TMF perform the same task executed by the majority of these trust formalisms. The SULTAN specification notation was not designed to facilitate behavioural models of trust. Thus, modal logics that model trusting behaviour are not modelled by the SULTAN TMF. The analysis facilities of the TMF improve upon those of current trust models, by allowing for reasoning about standard logical properties, as well as ad-hoc application domain-specific properties.

Computational and HCI-based trust models tend to focus on the evaluation and manipulation of trust values. Computational trust models (whether numerical or fuzzy) emphasis the calculation of the level of trust for a particular situation, while HCI-based models emphasis the assessment of trustworthiness in various interfaces and the mechanisms necessary to make computer-based

interfaces as trustworthy as non-computer-based ones. The usefulness of the methods of trust value calculation is governed by the underlying theory behind the calculation/assessment algorithms and how closely they tend to follow the intuitive mechanisms used by humans. The problems of trust value initialisation, calculation and combination are still open research issues. The SULTAN TMF models the value of trust by the trust level component of a SULTAN trust specification. The value is assumed to be a number assigned by the administrator or a variable that is put in as a placeholder. From examples and studies that have been used to demonstrate specification with the SULTAN system, it was uncovered that there are six commonly used trust levels, namely: HIGH-trust, MEDIUM-trust, LOW-trust, HIGH-distrust, MEDIUM-distrust and LOW-trust. The comparatively wide range of trust values supported by the SULTAN specification language allows the notation to be tailored for a diverse number of application areas (both current and future ones). Currently, the SULTAN specification notation is flexible enough to facilitate variable trust levels that result from trust value calculation. The SULTAN trust measure approximates the real trust level and is used as a baseline for comparisons to help in analysis and decision-making.

All the trust management solutions, with the exception of TrustBuilder, can be thought of as having a single conceptual design model, as shown in Figure 10.1. Contemporary trust management solutions consist of a set of context-specific data, a set of rules that constitute a policy with regards to access control or authentication and an evaluation mechanism that uses the question posed by an application, the policy and the facts in its local repository to generate and return an answer. The question is normally of the form, ‘Should I allow X given info Y?’ The evaluator checks to determine if X should be allowed given Y, the facts and the rules.

With public-key certificates, the facts would be the set of public keys in a local keyring. The rules would be the criteria used to trust new keys, i.e. I trust a key if it marginally-trusted by two meta-introducers. The evaluation mechanism is normally implemented in the software. The same approach can be observed with PICS, PolicyMaker, Keynote, REFEREE, Fidelis, SD3, IBM TEF, TCPA and Poblano. However, with the PolicyMaker-based systems, the evaluator (compliance checker) may make use of external data sources and or algorithms. Fig 10.1 is also representative of the specification and constraint satisfaction analysis components of the SULTAN TMF. The overall structure is similar to the process of an application querying the SULTAN consultant with a constraint satisfaction query. Contemporary solutions were designed with particular sub-problems of the trust management issue in mind, e.g. PolicyMaker addresses public-key authorisation, PICS tackles access control (web content filtering), etc.

Modifying these systems to function in other application domains, with different implementation paradigms, would require a lot of effort. However, the SULTAN TMF can be easily tailored to function in a variety of application domains. TrustBuilder is primarily concerned with secure credential exchange, which is important in enforcing trust establishment policy. As stated in Chapter 7, the SULTAN TMF can be used to aid the process of trust establishment.

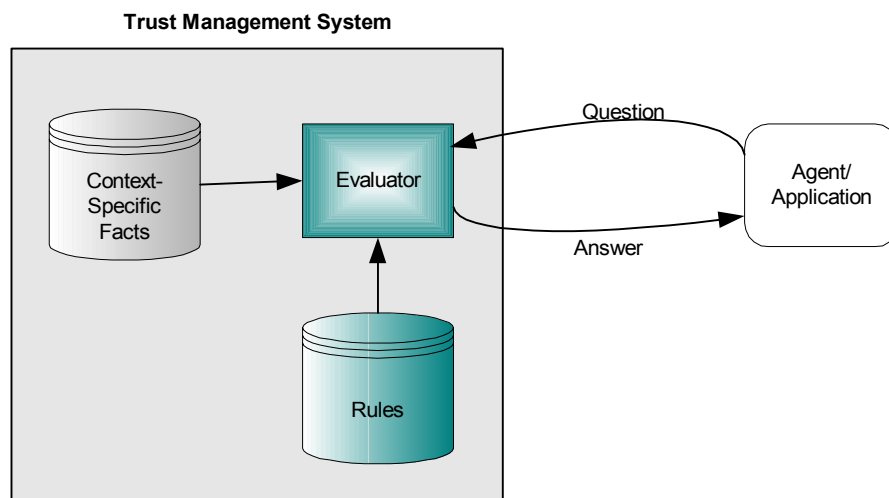


Figure 10.1: General Conceptual Structure of Contemporary Trust Management Solutions

As trust management is viewed, in this context, as an abstraction of security management, it may be pertinent to mention a security management framework that has evolved from trust management work and its relation to the SULTAN TMF. The StrongMan Architecture [163, 164] evolved out of the work done on PolicyMaker and KeyNote. The approach taken by the StrongMan system allows the use of multiple application-specific policy languages to specify security policies for particular applications. These languages map to a common layer, which is implemented in KeyNote. The disadvantages of this system are that 1) certain features that they want to uniformly provide in their architecture, e.g. delegation, must be specified at the lower-level (i.e. KeyNote), which makes specification more difficult, 2) KeyNote does not have facilities for the discovery of missing credentials, and 3) KeyNote has no support for negative assertions (c.f. IBM's TPL [57, 128]). These shortcomings indicate that KeyNote may not be a suitable choice for a common target layer. At an abstract level, the difference between StrongMan and SULTAN (apart from their areas of specialisation) is that SULTAN assumes trust management should be an appropriate source platform for refinement into security mechanisms, while Strongman assumes the converse.

The dominant connection between the SULTAN TMF and the majority of other trust models and trust management solutions is that SULTAN specifications can be refined into the trust policy language of these systems and that the analysis mechanisms supported by these systems may be modelled using the SULTAN Analysis Model. Thus, the SULTAN TMF provides a broader and more comprehensive approach to the trust management problem.

10.2 Evaluation of the Framework

In this section, the limitations and issues that need to be improved in the SULTAN TMF will be presented.

10.2.1 Specification Language Design

One of the more powerful features of the specification notation is the use of symbolic tags to represent entity names. This feature allows the specification of trust rules and recommendations about objects that represent any entity, ranging from public keys to applications to IP addresses. However, this feature may have unforeseen side effects when refinement is performed. For example, if it is known that *ProxyForAbby* is an entity name used in a specification and that the Monitor determines that *PublicKey1* is a part of *ProxyForAbby*, then care must be taken when determining the actions applicable to each refined entity. The semantic interpretation of the refined entities may imply that they are used differently. *ProxyForAbby* may refine into a computer peripheral and *PublicKey1* may refine into a public key. This may not ordinarily be a problem, but refinement tools need to consider this. To clarify the side effects that may occur, a study needs to be done on the implications of refining SULTAN specifications into a variety of different platforms. This would help to determine if untyped symbolic tags are too general an abstraction to be included in the specification language.

The current way that trust (and recommendation) levels are modelled in the SULTAN notation allows the specification of a large and diverse set of rules. This was illustrated in the mappings from SULTAN to other trust policy languages (Section 3.3). It was previously mentioned that the notation is able to handle scenarios where a trust evaluation module is set-up. The following example illustrates how this can be done:

```
CustVer: trust(Supplier, Customers, view_pages(Supplier), _X)  
← _X = trust_eval(Supplier, Customers, view_pages(Supplier));
```

In rule `CustVer`, `_X` is viewed as a trust value that has not been set. The method `trust_eval` represents the trust value mechanism, which will determine the value of the trust level. Once this method is defined and integrated in the framework, it will be possible to allow the (semi-) automated calculation of trust values. The addition of this method would make the specification process easier for the administrator. Of course, the inclusion of trust level calculation would mean that the analysis model would have to be modified. Currently, trust level calculation is a difficult problem, which still requires considerable research.

SULTAN specifications allow the encoding of a range of absolute trust requirements, which do not account for uncertainty or ignorance. This is not normally the case in real life, as trust value estimation is currently imprecise and uncertain. For example, an administrator may not know exactly what his belief should be. He may have a rough idea, but may be uncertain about the exact value. For this reason, there should ideally be a measure of his ignorance/uncertainty coupled to the trust value he assigns. Thus, the incorporation of uncertainty into the SULTAN specification language is a possible improvement, although assigning values to uncertainty can be as difficult as assigning values for trust.

Trust is a temporal concept. The framework facilitates this temporal dimension with the inclusion of the SULTAN Monitor and the issuing of time-stamps on some storage artefacts. However, the issue of the inclusion of an explicit temporal component in a SULTAN specification may enhance analysis. Temporal elements (time and date) could be inserted into the specification format for trust and recommendation rules. This addition would require that the specification language, the analysis model and the supporting tools be modified. The benefit of such a change would be the possibility of useful temporally driven analyses, e.g. over time period T days, starting from date D , which relationships were involved in one or more trust-distrust conflicts?

The trust and recommend statement are the core of the specification notation. Both constructs cover a wide range of scenarios that an administrator may currently want to specify. In the future, there may be a need for additional constructs, which may be either shorthand for complex SULTAN statement combinations or may represent related but different notions, e.g. reputation, delegation.

10.2.2 Analysis Model Design

The Analysis Model is an important component of the SULTAN TMF, which allows the formulation of a comprehensive set of analysis questions and enables reasoning on many different levels. Reasoning may be with respect to the program code, actual scenarios, cycles and missing credentials (constraint satisfaction). However, the addition of facilities for reasoning about experience and risk would further enhance the Analysis Model. For example, it may be useful for some administrators to know which clients have experience records with decreasing experience levels pertaining to a particular context or to know the exact stored history of experience levels for a customer. Currently, only the usage strategies (section 6.1.2) are supported in the SAM. Thus, more complex and diverse experience and risk-based reasoning facilities may be included to enhance the Analysis Model.

Cycle detection and resolution are important topics when dealing with analysis queries about the actual state of particular relationships. Cycle resolution is an intensive and complex problem. Though the Analysis Model includes a simple cycle resolution strategy, it is neither an efficient nor an optimal solution. The current strategy merely suppresses the cycle (if it exists) and ensures that analysis proceeds and always terminates. The real resolution occurs when some action is taken by the administrator to remove the cycle. Thus, the cycle resolution strategy needs to be improved. A possible improvement would be a strategy that does the following: 1) resolves the conflict temporarily, 2) flags the rules involved for the admin's attention, 3) allows the query to proceed, 4) reinterprets the query results if any of the rules involved in the cycle are to be returned to the user, and 5) reverses the resolution step to ensure that the constraints remain in the same state that the administrator entered. The biggest drawback of this strategy is the overhead involved in embedding these processes in each scenario-based query.

10.2.3 Architecture Design

The architecture for the SULTAN TMF is built on a simple model (Figure 10.2). A key architectural decision was to focus on building a framework that would support the core functionality of a trust management system, i.e. specification, analysis, monitoring, risk provision and calculation and consultation. The issues of the security and availability of the TMF components were considered secondary. There are standard ways of securing the databases, such as encrypting them to protect sensitive data, creating and validating hashes to ensure the integrity of the databases and replication could be used for improved availability, but

the inclusion of these issues in the current version of the prototype would have obscured the core functionality of the project.

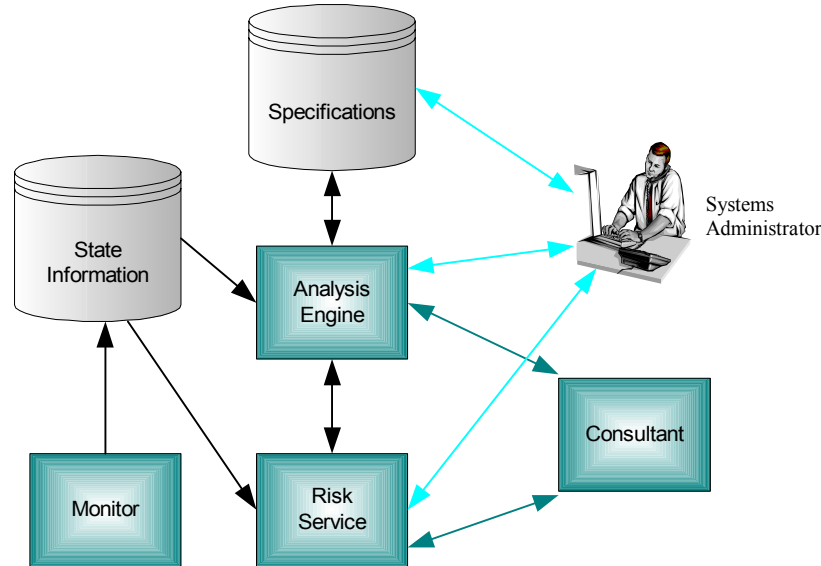


Figure 10.2: Basic Architecture of the SULTAN TMF

Any framework that assumes that the state of dynamic information must be monitored and stored will have scalability issues. In order to lessen the effect of state space explosion, subtle design decisions were taken in the construction of the SULTAN TMF. As this thesis discussed each aspect of the trust management framework, simple mechanisms were introduced to minimise the effect of the problem of scalability. For example, in the discussion of the SULTAN Specification Notation, entity names are made abstract and *isPartOf* facts are used to make a rule applicable to a range of entities, without explicitly specifying rules for each of these entities. This conserves the storage requirements for trust and recommend rules. In the description of the SULTAN Analysis Model, a client-server suite was designed over this SICStus Prolog engine to enable multiple threads to access it. This lessened the demands on the machine hosting the Analysis Tool, by reducing the number of Prolog interface objects used in any particular analysis session. A possible solution to the problem of a growing database of trust and recommend rules would be to segment the databases into namespaces and store each namespace in separate locations. This solution would require that the TMF include facilities to handle namespace management (i.e. location discovery, namespace unavailability, etc.), to ensure completeness and consistency of analysis results and to deal with conflicts.

In accordance with the simple design architecture, a simple protocol was used between the Consultant (and Monitor) and an application. The issues of ensuring information

confidentiality, dual authentication and non-repudiation were not addressed in the current protocol. The protocols could be enhanced to include security and dependability features if required.

Re-evaluation is a very important concept in the TMF. It allows the identification of conflicts and or ambiguities that may arise because of new state information. Currently, identified problems are flagged for the administrator's attention, which may work for an environment with bursty transactions, but may be inappropriate for a system with a high, consistent volume of transactions. Such a system may generate a large backlog of problems. If problems are not addressed within appropriate timescales, then the task of resolving these (possible) problems may seem formidable from the administrator's perspective. This may even affect the system's performance. An alternate means of resolving the problems need to be devised. The ideal solution to this issue would be the creation of an automated problem correction mechanism that allows the administrator to specify what action must be done when certain type of conflicts or ambiguities occur.

10.3 Evaluation of the Implementation

The primary function of the prototype was to evaluate the feasibility of implementing the SULTAN TMF. In developing this proof-of-concept, it was hoped that the implementation issues that arose would lead to the identification of the problems that remain unsolved and the issues to be addressed in future implementations.

The majority of the components of the SULTAN TMF were implemented in Java version 2. A significant part of the implementation is the Specification Editor, which incorporates a SULTAN Compiler, an Abstract Syntax Tree (AST) Walker, a SULTAN to Prolog translator and a mini-editor for entity connections, asset, risk likelihood and risk profile information. The Compiler and the AST Walker allows SULTAN specifications to be mapped to a variety of other representations. The Compiler was built using SableCC, a parser generator that created the lexical and syntactic analysers. SableCC enabled the relatively easy production of the front-end of the compiler. However, SableCC generates a large number of Java classes. A simple change in the SableCC grammar file to allow actionsets to include action restrictions resulted in the creation of six additional classes, as a result of the inclusion of two tokens in the grammar file. This implies that a considerable change in the specification notation, say to include additional constructs or temporal dimensions, would generate a considerably larger state space.

Although, this is unlikely to significantly affect the compilation process, it may make the refinement process more involved.

The Analysis Model forms a part of the backbone of the TMF. The versatility and flexibility of the Model is evidenced by the diverse range of questions that can be posed to it and the number of services that depend upon it. The Analysis Model and template file were implemented in SICStus Prolog and the databases are Microsoft Access. All analysis-related information (organization chart, state information, specification) is translated to Prolog. The Jasper interface used to bridge the gap between Java and Prolog provides a heavyweight Prolog object to the Java environment. Due to the memory requirements of the Prolog object, the Analysis Tool took an inordinately long time to respond to a query. Thus, the administrator had to wait a considerable time to enter new queries. This was partially resolved by creating an execution query thread management system, which coordinates the addition, output and removal of execution threads containing analysis queries. This allows input of analysis queries to proceed immediately after one has been sent for execution. The implementation of the Jasper interface made it necessary to define a client-server suite on top of the Jasper interface that would allow any thread to access the Prolog system and not just the thread that created the object. This suite also eliminated the problem of having each query generate a new Prolog engine. Nonetheless, the interface still requires a considerable amount of memory. Problems were initially encountered in maintaining the Analysis Tool and Specification Editor in memory together. These issues were resolved through creative memory management (i.e. by temporarily unloading parts of the Specification Editor). As a result of the issues discussed above, the Analysis Tool and Risk Service sometimes suffers time delays of more than 30 seconds. This normally occurred at the start of the session or when more than sixty-three processes attempted to concurrently access these tools. The scenarios that cause this delay involve asking the Prolog system a cross-section of queries. The Jasper interface is overloaded with the queries and starts monopolizing the available resources of the machine it is run on. In some cases, the Prolog system has crashed the system. However, the implementation clearly showed that the analysis defined in Chapter 4 was possible. The only issue is with the performance of the current Java-Prolog interface.

10.4 Summary

This chapter started with an evaluation of the SULTAN TMF in relation to related work. The obvious relation between the SULTAN TMF and the majority of other trust models and trust management solutions is that SULTAN specifications can be refined into the trust policy language of these systems and that the analysis mechanisms supported by some of these systems (e.g. SD3) can be modelled in the SULTAN Analysis Model. The chapter then presented a critique of the framework, highlighting the limitations and room for improvement in the specification language, analysis model and architecture. Although the SULTAN Trust Language seems powerful enough to specify many scenarios and the examples included in papers describing other notations, it could benefit from the inclusion of uncertainty, trust level calculation, an explicit temporal component in a specification, more primitive constructs and the examination of the effects on refinement of using symbolic tags as entity names. The Analysis Model could be improved by defining a more robust cycle resolution strategy and by allowing reasoning about risk and experience. The overall architecture would be enhanced if databases were segmented, replicated, securely stored and their integrity validated before use and the protocol between an application and the client portions of the SULTAN Monitor and SULTAN Trust Consultant were improved. Finally, an evaluation of the prototype for the SULTAN TMF was given.

Chapter 11 Conclusions

“Trust has become a critical feature for Internet applications because the economic viability of using the Internet as a business medium has been realized. Trust has become a focal point because of the complexity of the environment and the high level of interdependence, and thus reliance on the behaviours of Internet agents.”

- adapted from ‘*Electronic Commerce and the Concept of Trust*’ [165]

In this chapter, a summary of the work covered in this thesis will be presented. The contributions of this work will be stated and the future direction of this work will be given.

11.1 Review and Discussion

Relative to distributed computing, trust management is a new topic. The contemporary approach has been to focus on the security management aspects of the trust management problem, specifically the trust problems directly related to access control, authentication and authorisation. There is a plethora of trust management solutions being developed. Each is applicable only to a certain application domain, execution environment, or vendor. There is a lack of a vendor and application independent solution to the problem of trust management. The work presented here helps in that direction by providing a definition of a framework for trust management that could be used as the blueprint for future solutions.

This thesis presents a comprehensive survey of trust definitions, trust formalisms, trust management perspectives and trust management solutions. We defined a structure for classification of the literature in terms of trust contexts and also discussed the characteristic properties of a trust relationships. This work is motivated by the importance of trust to the future success of Internet Commerce and by the necessity for a trust management framework that focused on the trust management problem (and not a subset of the problem). The SULTAN TMF examined the trust management problem from an abstract level, incorporating a generalised view of trust relationships and recommendations, providing versatile and flexible analysis facilities, including the notions of risk and experience and factoring in the nature and characteristics of trust.

To provide a common background and define a shared lexicon, new definitions for the concepts of trust and distrust are formulated. Trust is:

“the quantified belief by a trustor with respect to the competence, honesty, security and dependability of a trustee within a specified context.”

Each of the important terms of the definition, i.e. quantified, trustor, competence, honesty, security, dependability, trustee, specified context, are explained further in Chapter 1. Distrust is:

“the quantified belief by a trustor that a trustee is incompetent, dishonest, not secure or not dependable within a specified context.”

The characteristics of the trust relationship are presented and some of the contributing factors to a trust relationship are identified, namely: risk and experience. Trust management is defined as:

“the activity of collecting, encoding, 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.”

All of this background information provide a valuable starting point for our work and may be beneficial to other researchers in the field. Trust management involves the acts of specifying trust relationships, analysing them to uncover new (and or wanted relationships or side-effects) and presenting evidence that can be used to make trust decisions. Evidence should be collected from the source of the interactions and should be used to allow the subject to adapt his trust requirements based on this (new information). Thus, the process of trust management should also include the monitoring and (re)-evaluation of the subject’s trust information.

The SULTAN specification notation captures the essence of the trust relationships and recommendations. Each trust relationship must have a trustor (subject), a trustee (target to be trusted), a context (a set of actions) and a level of trust. A recommendation has a recommendor (subject), a recommendee (target to be recommended), a context (a set of actions) a measure of the recommendor’s confidence in the recommendation. These basic concepts for trust, distrust, positive and negative recommendations seem to be powerful enough for most aspects of trust relationships. Through work done with Prof. Han Reichgelt, of Georgia State University and Dr. Audun Josang, of the Distributed Systems Technology Centre, it is believed that the notation encapsulates the basic components of a trust relationship and a recommendation. This belief is also confirmed by the use of these basic concepts in the design of a trust management model for Multi-Agent Systems [2].

The SULTAN Analysis Model (SAM) incorporates the visions of previous analysis models from the world of logical trust formalisms and extends them to increase its applicability across a spectrum of problem areas. The SAM facilitates both simulation and property analysis. Simulation analysis is concerned with asking ‘What-If’ questions. Property analysis involves checking whether specified properties hold on trust and recommendation rules and is concerned with the discovery of conflicts and redundancies. A conflict arises as a result of two assertions (trust or recommend) of different polarities (positive and negative), the same actions and referring to the same subject and target. A redundancy (or ambiguity) is defined as the state where two assertions, of the same type (trust or recommend), have the same subject, target, actions and levels and where the assertions are of the same polarity, but possess different values. The properties to be analysed can be with respect to the specification source (program reasoning) or with respect to examining trust relationships to identify scenarios of interest. When reasoning about scenarios, the issue of detecting cycles and the issue of the constraints that are still to be satisfied for a trust relationship to be valid arises. Thus, the SAM allows reasoning about source code, scenarios, cycles and constraint satisfaction (missing credentials). A template of common conflicts and ambiguities is provided for the administrator’s use. This is a vast improvement on the set of trust analysis models in existence.

There is no other trust management model that incorporates a well-defined risk evaluation mechanism. The SULTAN TMF includes a Risk Service (SRS), which offers a risk provision service and a risk assessment service. To place the SRS in context, a survey of current approaches to risk modelling is provided and a critique of their problems given. Risk provision involves retrieving risk information stored in the State Information Database, while risk assessment involves generating a risk value based on the TMF’s risk calculation algorithm. The calculation algorithm uses Josang’s Subjective Logic and incorporates ideas from expected loss models and risk threshold models in order to overcome the problems of risk identification, probability determination, loss evaluation, dependence handling for actionsets and risk profiling. Novel aspects of the risk assessment service include its application of the expected loss model to asset valuation and its handling of dependent actions.

Experience, trust monitoring and trust re-evaluation are relatively new topics in the field of trust management. A majority of trust management solutions assume that trust is a static concept and therefore does not require monitoring or (periodic) re-evaluation. This is not true in the real world, where trust is dynamic and changes daily. This thesis highlights the cyclic connection between the concepts of experience, monitoring and re-evaluation that is necessary for trust

evolution. A formulation of the basic components of an experience record is presented and the various types of experience usage strategies (i.e. optimistic, pessimistic, cautious, most-recently used) were given. The SULTAN Monitor collects information on the system's state, on risk and on dependencies and connections. Based on the new information gathered by the Monitor, the queries in the Analysis template are executed and any conflicts or ambiguities uncovered are flagged for the administrator's attention.

As a proof of concept, a set of tools, consisting of the Specification Editor, the Analysis Tool, the Risk Service, the Trust Monitor and the Trust Consultant, are presented. A basic trust management life cycle was developed and presented in conjunction with the tools and the TMF basic data structures. The Specification Editor includes a compiler for SULTAN specifications, an AST walker (to help in refinement to other notations), a SULTAN to Prolog translator (to illustrate that refinement is possible) and a Mini-Editor (for the initialisation tasks). The Analysis Tool is connected to a SICStus Prolog engine and utilises the SULTAN to Prolog translator to convert specifications, state information and entity connection information. To ease the effort required by the administrator, a template of conflicts and ambiguities is provided. There is also a Query Statement Builder Tool, which helps in the formulation of analysis queries. With the exception of the Risk Service, the other tools are strictly for use of the machine of ordinary users. They allow the user to send the system information and to query this information. The tools may be used for negotiation, contract evaluation, recommendation formation, infrastructure security configuration, access control decision-making and resource allocation.

Throughout this thesis, examples are used to describe the concepts under discussion. The objectives of the framework (Chapter 1) are to provide:

1. A clear, semantically well-defined, expressive specification notation.
2. A comprehensive analysis model.
3. A framework that assumes trust non-monotonicity.
4. A framework that facilitates trust decision-making.

The framework, as presented in chapters 3 to 7, fully satisfies each of these objectives. Interest in the work done on the SULTAN TMF has led to its ideas and concepts being used in many projects, ranging from an EU project on 'Trust and Contract Management for Secure, Dynamic Virtual Organisations' (TrustCoM) [166] to a project on 'Distributed Digital Rights Management and Security Model' [167] to a project on 'Trust-Based Self-Organized Routing

Protocols for Secure Ad Hoc networks' [168] to a project on 'Building a Formal Model of Trust for Dynamic Networks' [169].

11.2 Future Work

Some of the issues identified as future work originated from the discussion in Chapter 10. In this section, the most important of these issues is presented.

11.2.1 Specification Language

The idea of including uncertainty in SULTAN specifications may be an interesting one to trust researchers. Work has been done on merging Josang's Subjective Logic and the SULTAN TMF. Two possible ways are identified to connect the two models, namely:

1. To modify SULTAN specification syntax and Analysis Model to handle Subjective Opinions as trust and or recommendation levels.
2. To refine SULTAN specifications to Opinions about logical propositions.

Exploration of option 2 led to the extension of Subjective Logic to include an operator for handling conditional statements. The conditional inference operator [170] not only facilitates the direct refinement of SULTAN statements to Subjective Logic Opinions, but also provided Subjective Logic with a useful way of modelling classical logical reasoning mechanisms (modus ponens, modus tollens). However, this particular avenue represents a method of injecting an uncertainty measure into specifications after they have been written in SULTAN, which may not always be desirable. The issue of modifying components of the TMF to allow direct specification and reasoning about Opinions is a possible extension.

11.2.2 Analysis Model

The Analysis Model can be improved by expanding it to incorporate ad hoc reasoning about experience and risk. The particular type of analysis queries that should be accommodated needs to be investigated. Should usage strategies be included in this new Model? Should time-based reasoning about experience and risk records be allowed? The inclusion of this feature may also impact the architecture of the TMF, i.e. the Consultant would only need to interact with the Analysis Tool and not with the Risk Service.

11.2.3 Architecture

Partitioning the Specification Database into namespaces allows the system to be used for larger information stores. The failure of contemporary reasoning models for large-scale distributed systems has been the performance degradation that accompanied large source files, which grow at exponential rates. Segmentation, via namespaces, would increase the usability of databases as they increase in size. However, this solution would raise management issues, such as locating, combining and replicating segments and ensuring that analysis remains complete and consistent.

11.2.4 Implementation

The performance and memory requirements of the Prolog system need to be vastly improved. Possible replacements may be: Amzi-Prolog, B-Prolog or BinProlog, which have native interfaces to Java. An alternative direction may be to use a Prolog engine written in Java, such as jProlog, DGKS Prolog, JavaLog, JIProlog or MINERVA. However, the problem with this approach is that Prolog engines in Java tend to provide just the basic Prolog inference engine and methods for the addition and removal of facts/rules. This may not be a major problem, but it would mean that libraries that come as standard with most pure Prolog systems would have to be implemented in the Analysis Model. It would be interesting to evaluate the performance of each of these approaches to determine which would produce a more stable, lightweight Analysis system.

11.2.5 General

It is believed that the SULTAN Trust Management Framework has captured the core issues involved in trust and trust management. Although, the current implementation is heavyweight and may suffer from scalability issues, the general ideas may be transferred to the research fields of enabling Grid technology. The Distributed Systems Lab at Argonne National Laboratory has expressed interest in utilising the ideas from this project to incorporate into the Globus project. The core ideas of trust specification, trust analysis, trust monitoring, risk evaluation, experience modelling and trust consultation can be applied to Peer-to-Peer Computing, Ubiquitous Computing and Pervasive Computing. With this goal in mind, initial discussions are in progress to augment the 'Trusted Software Agents and Services for Pervasive Information Environments in the Home' project at the University of Southampton with the lessons learnt from work done on the SULTAN TMF.

11.3 Closing Remarks

Trust management is a relatively young and complex research field with many different emerging ideas and solutions. This has led to a situation where there is confusion, ambiguity, conflicts, misinterpretations, an absence of core principles and a lack of standard approaches towards and about research in this field. In this thesis, a presentation is given on the basics of trust and trust management. It is hoped that the information presented will help form a common core of knowledge that may be the starting point for new trust researchers. A framework is presented that incorporates risk, experience, trust relationship evolution and allows specification and analysis of trust statements and recommendations. The process of designing and implementing (a prototype for) this framework represents a significant challenge, which is successfully addressed in this thesis, and which will hopefully form the basis of other work in the field of trust management. The specification notation, analysis model, risk service and treatment of trust relationship evolution constitute the main contributions of this thesis.

Bibliography

1. Confucius, *The Analects of Confucius*, ed. A.T. Waley. 1989: Vintage Books. 256. ISBN: 0679722963.
2. Grandison, T. and M. Sloman. *Trust Management Tools for Internet Applications*. in *1st International Conference on Trust Management*. 2003. Heraklion, Crete, Greece: Springer. <http://www.doc.ic.ac.uk/~tgrand/>
3. Grandison, T. and M. Sloman, *A Survey of Trust in Internet Applications*. IEEE Communications Surveys and Tutorials, 2000. **4**(4). <http://www.comsoc.org/pubs/surveys/>, <http://www-dse.doc.ic.ac.uk/~tgrand/>
4. Grandison, T., *Trust Specification and Analysis for Internet Applications*. 2001. MPhil/PhD Report, Imperial College of Science, Technology and Medicine: London. <http://www.doc.ic.ac.uk/~tgrand>
5. 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: IEEE. <http://www.doc.ic.ac.uk/~mss/Papers/I3e2002.pdf>
6. Verissimo, P. and L. Rodrigues, *Distributed Systems for System Architects*. 2001. Kluwer Academic Publishers. ISBN: 0-7923-7266-2.
7. Blaze, M., J. Feigenbaum, P. Resnick and M. Strauss, *Managing Trust in an Information-Labeling System*. European Transactions on Telecommunications, 1997. **8**: p. 491-501. <http://www.si.umich.edu/~presnick/papers/bfrs/Paper.ps>
8. Blaze, M., J. Feigenbaum and J. Lacy, *Managing Trust in Medical Information Systems*. 1996. AT&T Research Labs. <http://citeseer.nj.nec.com/did/35925>
9. Harrington, S.J. and C.P. Ruppel, *Telecommuting: a test of trust, competing values, and relative advantage*. IEEE Transactions on Professional Communication, 1999. **42**(4): p. 223 - 239. <http://ieeexplore.ieee.org/iel5/47/17506/00807960.pdf>
10. Ordille, J.J. *When agents roam, who can you trust?* in *First Annual Conference on Emerging Technologies and Applications in Communications*. 1996. <http://ieeexplore.ieee.org/iel4/3740/10938/00502505.pdf>
11. Feigenbaum, J. and P. Lee. *Trust Management and Proof-Carrying Code in Secure Mobile Code Applications: Position Paper*. in *DARPA Workshop on 'Foundations for Secure Mobile Code'*. 1997. <http://www.research.att.com/~jfpubs/darpa-mobile.ps>
12. Wilhelm, U.G., S. Staamann and L. Buttyan. *On the problem of trust in mobile agent systems*. in *IEEE Symposium on Network And Distributed System Security*. 1998. San Diego, California. <http://citeseer.nj.nec.com/pdf/139207>, <http://icawww.epfl.ch/buttyan/publications/NDSS98.ps>
13. Ketchpel, S.P. and H. Garcia-Molina. *Making trust explicit in distributed commerce transactions*. in *16th International Conference on Distributed Computing Systems*. 1996. <http://ieeexplore.ieee.org/iel3/3771/11006/00507925.pdf>
14. Iacono, C.S. and S. Weisband. *Developing trust in virtual teams*. in *13th Hawaii International Conference on System Sciences*. 1997. <http://ieeexplore.ieee.org/iel4/5350/14595/00665615.pdf>

15. Holland, C.P. and A.G. Lockett. *Business trust and the formation of virtual organizations*. in *31st Annual Hawaii International Conference on System Sciences*. 1998. Hawaii. <http://ieeexplore.ieee.org/iel4/5217/14260/00654821.pdf>
16. Clark, T.H. and G.L. Ho. *Electronic intermediaries: trust building and market differentiation*. in *32nd Annual Hawaii International Conference on Systems Sciences*. 1999. Hawaii. <http://ieeexplore.ieee.org/iel5/6293/16785/00772939.pdf>
17. Jarvenpaa, S.L., N. Tractinsky, L. Saarinen and M. Vitale, *Consumer Trust in an Internet Store: A Cross-Cultural Validation*. *Journal of Computer-Mediated Communication*, 1999. **5**(2). <http://www.ascusc.org/jcmc/vol5/issue2/jarvenpaa.html>
18. Jiawen, S. and D.W. Manchala. *Trust vs. threats: recovery and survival in electronic commerce*. in *19th IEEE International Conference on Distributed Computing Systems*. 1999. <http://ieeexplore.ieee.org/iel5/6307/16865/00776513.pdf>
19. Jøsang, A. *Trust-based decision making for electronic transactions*. in *The 4th Nordic Workshop on Secure IT Systems (NORDSEC'99)*. 1999. Stockholm, Sweden: Stockholm University Report 99-005, 1999.
20. Su, J. and D. Manchala. *Trust vs. Threats: Recovery and Survival in Electronic Commerce*. in *19th International Conference on Distributed Computing Systems*. 1999.
21. Khare, R. and A. Rifkin, *Trust Management on the World Wide Web*. Peer-reviewed Journal on the Internet. **3**(6). <http://www.firstmonday.dk/issues/khare/index.html>
22. Swarup, V. and C. Schmidt. *Interoperating between Security Domains*. in *ECOOP (European Conference on Object-Oriented Programming) Workshop on Distributed Object Security*. 1998. Brussels, Belgium.
23. Jøsang, A. and S.J. Knapskog. *A metric for trusted systems*. in *21st National Security Conference*. 1998. <http://www.idt.ntnu.no/~ajos/papers.html>
24. Amoroso, E., et al. *Toward an approach to measuring software trust*. in *IEEE Computer Society Symposium on Research in Security and Privacy*. 1991. <http://ieeexplore.ieee.org/iel2/349/3628/00130788.pdf>
25. Manchala, D.W. *Trust metrics, models and protocols for electronic commerce transactions*. in *18th International Conference on Distributed Computing Systems*. 1998. <http://ieeexplore.ieee.org/iel4/5583/14954/00679731.pdf>
26. Damianou, N., N. Dulay, E. Lupu and M. Sloman. *The Ponder Specification Language*. in *Workshop on Policies for Distributed Systems and Networks*. 2001. HP Labs, Bristol: Springer-Verlag. <http://www-dse.doc.ic.ac.uk/~mss/MSSPubs.html>
27. Gelfond, M. and R. Watson, *Encyclopedia of Cognitive Science - Non-monotonic Logic (Article 62)*. 2001. <http://www.krlab.cs.ttu.edu/Papers/download/gw03.pdf>
28. Luhmann, N., *Trust and Power*. 1982, Chicester: John Wiley & Sons. ISBN: 0471997587.
29. Kini, A. and J. Choobineh. *Trust in Electronic Commerce: Definition and Theoretical Considerations*. in *31st Annual Hawaii International Conference on System Sciences*. 1998. Hawaii. <http://ieeexplore.ieee.org/iel4/5217/14270/00655251.pdf>
30. Jones, S., *TRUST-EC: requirements for Trust and Confidence in E-Commerce*. 1999. Technical Report. European Commission, Joint Research Centre.
31. Lewis, D. and A. Weigert, *Social Atomism, Holism and Trust*. *Sociological Quarterly*, 1985. **26**(4): p. 455-471.

32. Mayer, R.C. and J.H. Davis, *An Integrative Model of Organizational Trust*. Academy of Management Review, 1995. **20**(3): p. 709 - 734.
33. Zand, D.E., *Trust and Managerial Problem Solving*. Administrative Science Quarterly, 1972. **17**: p. 229 - 239.
34. Curral, S. and T. Judge, *Measuring Trust Between Organizational Boundary Role Persons*. Organizational Behaviour and Human Decision Processes, 1995. **65**: p. 601 - 620.
35. Mui, L., M. Mohtashemi and A. Halberstadt. *A Computational Model of Trust and Reputation for E-Businesses*. in *35th Annual Hawaii International Conference on System Sciences (HICSS'02)*. 2002. Big Island, Hawaii.
<http://dlib2.computer.org/conferen/hicss/1435/pdf/14350188.pdf?>,
http://www.albany.edu/~bmsi603/scholl/Mui_et_al_2002.pdf
36. Jones, A.J.I., *On the Concept of Trust*. <http://alfebiite.ee.ic.ac.uk/docs/papers/D1/ab-d1-jones-trust.pdf>
37. Anderson, R.J., *TCPA / Palladium Frequently Asked Questions Version 1.0*. 2002.
<http://www.cl.cam.ac.uk/~rja14/tpa-faq.html>
38. Adams, C. and S. Farrell, *RFC2510 - Internet X.509 Public Key Infrastructure Certificate Management Protocols*. 1999. <http://www.cis.ohio-state.edu/htbin/rfc/rfc2510.html>
39. 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>
40. Blaze, M., J. Feigenbaum and M. Strauss. *Compliance Checking in the PolicyMaker Trust Management System*. in *Financial Cryptography: Second International Conference*. 1998. Anguilla, British West Indies.: Springer-Verlag.
<http://www.crypto.com/papers/pmcomply.pdf>
41. 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>
42. Blaze, M., J. Feigenbaum, J. Ioannidis and A.D. Keromytis, *The Role of Trust Management in Distributed Systems Security*, in *Secure Internet Programming: Security Issues for Mobile and Distributed Objects*, Vitek and Jensen, Editors. 1999, Springer-Verlag. <http://www.crypto.com/papers/trustmgt.pdf>
43. Blaze, M., J. Ioannidis and A.D. Keromytis. *Trust Management and Network Layer Security Protocols*. in *Cambridge Protocols Workshop*. 1999. Cambridge.
<http://www.crypto.com/papers/networksec.pdf>
44. Blaze, M., *Using the KeyNote Trust Management System*. 1999, AT&T Research Labs.
<http://www.crypto.com/trustmgt/kn.html>
45. Blaze, M., J. Feigenbaum, I. J. and K. A., *RFC2704 - The KeyNote Trust Management System (version 2)*. 1999. <http://www.crypto.com/papers/rfc2704.txt>
46. Burrows, M., M. Abadi and R.M. Needham, *A Logic of Authentication*. ACM Transactions on Computer Systems, 1990. **8**(1): p. 18-36.
<http://citeseer.nj.nec.com/details/burrows90logic.html>

47. 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>
48. Chu, Y.-H., J. Feigenbaum, B. LaMacchia, P. Resnick and M. Strauss, *REFEREE: Trust Management for Web Applications*. 1997, AT&T Research Labs. <http://www.farcaster.com/papers/www6-referee/>
49. Chu, Y.-H., *Trust Management for the World Wide Web*. 1997, Massachusetts institute of Technology. <http://www.w3.org/1997/YanghuaChu/>
50. Compaq, Hewlett-Packard, IBM, Intel and Microsoft, *TCPA Design Philosophies and Concepts Version 1*. 2000. http://www.trustedcomputing.org/docs/designv1_0final.pdf
51. Compaq, Hewlett-Packard, IBM, Intel and Microsoft, *Building a Foundation of Trust in the PC*. 2000, The Trusted Computing Platform Alliance. <http://www.trustedpc.org>
52. Compaq, Hewlett-Packard, IBM, Intel and Microsoft, *TCPA PC Specific Implementation Version 1.00*. 2001. http://www.trustedcomputing.org/docs/TCPA_PCSpecificSpecification_v100.pdf
53. Compaq, Hewlett-Packard, IBM, Intel and Microsoft, *TCPA Main Specification version 1.1b*. 2002. http://www.trustedcomputing.org/docs/main%20v1_1b.pdf
54. Compaq, Hewlett-Packard, IBM, Intel and Microsoft, *TPM FAQ*. 2002, Trusted Computing Platform Alliance. http://www.trustedcomputing.org/docs/TPM_QA_071802.pdf
55. Evans, C., C.D.W. Feather, A. Hopmann, M. Presler-Marshall and P. Resnick, *PICSRules 1.1*. <http://www.w3.org/TR/REC-PICSRules>
56. Feigenbaum, J. *Overview of the AT&T Labs Trust Management Project: Position Paper*. in *Proceedings of the 1998 Cambridge University Workshop on Trust and Delegation*. 1998: Lecture Notes in Computer Science.
57. IBM, *IBM Trust Establishment Policy Language*. <http://www.haifa.il.ibm.com/projects/software/e-Business/TrustManager/PolicyLanguage.html>
58. 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>
59. Miller, J., P. Resnick and D. Singer, *PICS Rating Services and Rating Systems (and Their Machine Readable Descriptions) version 1.1*. <http://www.w3.org/TR/REC-PICS-services>
60. McKnight, H.D. and N. Chervany, *The Meanings of Trust*. <http://misrc.umn.edu/wpaper/WorkingPapers/9604.pdf>
61. Lamsal, P., *Understanding Trust and Security*. 2001. <http://citeseer.nj.nec.com/lamsal01understanding.html>
62. Gerck, E., *Toward Real-World Models of Trust*. 1998, E Gerck and MCG. <http://www.mcg.org.br/trustdef.htm>
63. Corritore, C.L., B. Kracher and S. Wiedenbeck, *An Overview of Trust: Working Document*. 2001. http://cobacourses.creighton.edu/trust/articles/trustpaper2-9-01_final.rtf

64. Marsh, S.P., *Formalising Trust as a Computational Concept*, in *Computing Science and Mathematics*. 1994, University of Stirling: Stirling, Scotland. p. 170.
<http://www.iit.nrc.ca/~steve/Publications.html>
65. Mayer, F.L. *A brief comparison of two different environmental guidelines for determining 'levels of trust' (computer security)*. in *Sixth Annual Computer Security Applications Conference*. 1990. <http://ieeexplore.ieee.org/iel2/319/3856/00143781.pdf>
66. Christianson, B. and W.S. Harbison. *Why Isn't Trust Transitive?* in *Security Protocols International Workshop*. 1996. University of Cambridge.
67. Abrams, M.D., *Trusted System Concepts*, in *Computers and Security*, M.V. Joyce, Editor. 1995. p. 45 - 56.
68. Abrams, M.D. and M.V. Joyce, *Trusted Computing Update*. *Computers and Security*, 1995. **14**(1): p. 57 - 68.
69. Abrams, M.D. and M.V. Joyce, *New Thinking About Information Technology Security*. *Computers and Security*, 1995. **14**(1): p. 69 - 81.
70. Dzubeck, F., *Application Service Providers: An old idea made new*. 1999.
<http://www.nwfusion.com/archive/1999b/0823dzubeck.html>
71. Morency, J., *Application Service Providers and e-business*. 1999.
<http://www.nwfusion.com/newsletters/nsm/0705nm1.html?nf>
72. Nix, M., *Entering the Application Service Provider market*.
<http://www.developer.ibm.com/library/articles/nixasp.html>
73. Krauskopf, T., J. Miller, P. Resnick and W. Treesee, *PICS Label Distribution Label Syntax and Communication Protocols Version 1.1*. <http://www.w3.org/TR/REC-PICS-labels>
74. *X.509 Certificates and Certificate Revocation Lists (CRLs)*, Sun Microsystems Inc.
<http://java.sun.com/products/jdk/1.2/docs/guide/security/cert3.html>
75. *An Introduction to Cryptography*, in *PGP 6.5.1 User's Guide*, Network Associates Inc. p. 11 - 36. <http://www.fi.pgpi.org/doc/pgpintro/>
76. Gerck E and M.-C. Group, *Overview of Certification Systems: X.509, CA, PGP and SKIP*. 1997, Meta-Certificate Group. <http://www.mcg.org.br/cert.htm>
77. Galvin, P., *Are you certifiable?*. 2000. http://www.sunworld.com/sunworldonline/swol-10-1997/f_swol-10-security.html
78. Gerck, E., *Certification: Extrinsic, Intrinsic and Combined*. 1997.
<http://mcg.org.br/cie.htm>
79. Rivest, R.L. *Can We Eliminate Certificate Revocation Lists?* in *Financial Cryptography*. 1998. <http://theory.lcs.mit.edu/~rivest/revocation.ps>
80. Povey, D., *Trust Management*. 1999. <http://security.dstc.edu.au/presentations/trust/>
81. Ding, Y. and H. Petersen, *A new approach for delegation using hierarchical delegation tokens*. 1995, University of Technology Chemnitz-Zwickau Department of Computer Science.
82. *Department of Defense: Trusted Computer System Evaluation Criteria*. 1983.
<http://ftp.std.com/obi/DOD/orange.book/>

83. Helvik, B.E., *Dependable Computer Systems and Communications Networks : Design and Evaluation*. 2001, Department of Telematics, NTNU Norwegian University of Science and Technology. <http://www.item.ntnu.no/~bjarne/Dependability-ICTBook.toc.pdf>
84. Galin, D., *Software Quality Metrics—From Theory to Implementation*. Software Quality Professional, 2003. **5**(3).
85. Kan, S.H., *Metrics and Models in Software Quality Engineering*. 1995. 344. 0201633396: 0201633396.
86. Littlewood, B. and L. Strigini. *Software Reliability and Dependability: a Roadmap*. in *22nd Int. Conf. on Software Engineering*. 2000. Limerick: ACM Press.
87. Abdul-Rahman, A. and S. Hailes. *Supporting Trust in Virtual Communities*. in *Hawaii International Conference on System Sciences 33*. 2000. Maui, Hawaii. <http://www.cs.cs.ucl.ac.uk/staff/F.AbdulRahman/docs/>
88. Jajodia, S., P. Samarati and V. Subrahmanian. *A Logical Language for Expressing Authorizations*. in *Security and Information Privacy*. 1997: IEEE. <http://ieeexplore.ieee.org/iel3/4693/13107/00601312.pdf>
89. Firozabadi, B.S. and M. Sergot. *Power and Permission in Security Systems*. in *7th International Workshop In Security Protocols*. 1999. Cambridge, UK: LNCS. Springer-Verlag.
90. Genesereth, M.R. and N.N. J., *Logical Foundations of Artificial Intelligence*. 1987, California, U.S.A.: Morgan Kaufmann Publishers Inc. 405. 0-934613-31-1: 0-934613-31-1.
91. 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.
92. Rangan, P.V. *An Axiomatic Basis of Trust in Distributed Systems*. in *Symposium on Security and Privacy*. 1988. Washington, DC: IEEE Computer Society Press.
93. Elgesem, D., *The Modal Logic of Agency*. *Journal of Philosophical Logic*, 1997. **2**(2): p. 1-46.
94. Kakas, A. and R. Miller, *A Simple Declarative Language for Describing Narratives with Actions*. *Journal of Logic Programming*, 1997(Special Issue on Reasoning About Actions).
95. Jøsang, A. *An Algebra for Assessing Trust in Certification Chains*. in *Network and Distributed Systems Security Symposium (NDSS)*. 1999. San Diego, California: The Internet Society.
96. 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>
97. Jøsang, A. *Artificial Reasoning with Subjective Logic*. in *2nd Australian Workshop on Commonsense Reasoning*. 1997. <http://www.idt.ntnu.no/~ajos/papers.html>
98. 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>

99. 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>
100. Jøsang, A., *A Logic for Uncertain Propositions*. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 2001. **9**(3): p. 1 - 30.
<http://security.dstc.edu.au/papers/>
101. Jøsang, A., *The Consensus Operator for Combining Beliefs*. *Artificial Intelligence Journal*, 2002. **1**(2): p. 157 - 170. <http://security.dstc.edu.au/papers/>
102. Friedman, B., P.H. Kahn Jr. and D.C. Howe, *Trust Online*. *Communications of the ACM*, 2000. **43**(12): p. 34 - 40.
103. Olson, J.S. and G. Olson, *i2i Trust in E-Commerce*. *Communications of the ACM*, 2000. **43**(12): p. 41 - 44.
104. Cassell, J. and T. Bickmore, *External Manifestations of Trustworthiness in the Interface*. *Communications of the ACM*, 2000. **43**(12): p. 50 - 56.
105. Schoder, D. and P.-L. Yin, *Building Firm Trust Online*. *Communications of the ACM*, 2000. **43**(12): p. 73 - 79.
106. Shneiderman, B., *Designing Trust into Online Experiences*. *Communications of the ACM*, 2000. **43**(12): p. 57 - 59.
107. Schneider, J., G. Korteum, D. Preuitt, S. Fickas and Z. Segall, *Auranet: Trust and Face-to-Face Interactions in a Wearable Community*. 2001, Technical Report, University of Oregon. <http://www.cs.uoregon.edu/research/wearables/Papers/auranet.pdf>
108. Schneider, J., G. Korteum, J. Jager, S. Fickas and Z. Segall. *Disseminating Trust Information in Wearable Communities*. in *2nd International Symposium on Handheld and Ubiquitous Computing (HUC2K)*. 2000. Bristol, England.
<http://www.cs.uoregon.edu/research/wearables/Papers/HUC2K.pdf>
109. Egger, F.N. *Affective Design of E-Commerce User Interfaces: How to Maximise Perceived Trustworthiness*. in *International Conference on Affective Human Factors Design*. 2001. The Oriental, Singapore: Asean Academic Press, London.
<http://www.ecommuse.com/research/publications/CAHD2001.htm>
110. Staples, S.D. and P. Artnasingham. *Trust: The Panacea of Virtual Management*. in *19th International Conference on Information Systems*. 1998. Helsinki, Finland: Association for Information Systems.
111. Egger, F.N. and B.d. Groot. *Developing a Model of Trust for Electronic Commerce: An Application to a Permissive Marketing Web Site*. in *9th International World Wide Web Conference*. 2000. Amsterdam, The Netherlands: Foretec Seminars IInc.
112. Kim, K. and B. Prabhakar. *Initial Trust, Perceived Risk and the Adoption of Internet Banking*. in *International Conference on Information Systems*. 2000. Brisbane, Australia. <http://www.nr.no/~abie/Papers/00RIP11.pdf>
113. Bos, N., J. Olson, D. Gergle, G. Olson and Z. Wright. *Effects of Four Computer-Mediated Communications Channels on Trust Development*. in *Computer Human Interaction (CHI)*. 2002. Minneapolis, Minnesota, USA: ACM.
114. Zheng, J., E. Veinott, N. Bos, J.S. Olson and G.M. Olson. *Trust without Touch: Jumpstarting long-distance trust with initial social activities*. in *Computer Human Interaction*. 2002. Minneapolis, Minnesota, USA: ACM.

115. Axelrod, R., *The Evolution of Cooperation*. 1985: Basic Books. ISBN: 0465021212.
116. Poundstone, W., *Prisoner's Dilemma: John Von Neumann, Game Theory and the Puzzle of the Bomb*. 1993: Anchor. 294. ISBN: 038541580X.
117. Riegelsberger, J., A.M. Sasse and J. McCarthy, *The Researcher's Dilemma: evaluating trust in computer-mediated communication*. International Journal of Human Computer Studies., 2002(Special Issue on Trust).
118. Patrick, A., *Privacy, Trust, Agents & Users: A Review of Human-Factors Issues Associated with Building Trustworthy Software Agents*. 2001.
<http://www.iit.nrc.ca/~patricka/agents/agents.pdf>
119. Riegelsberger, R. and M.A. Sasse. *Trustbuilders and trustbusters: The role of trust cues in interfaces to e-commerce applications*. in *1st IFIP Conference on E-commerce, E-business, E-government (I3E)*. 2001. Zurich.
http://www.cs.ucl.ac.uk/staff/jriegels/trustbuilders_and_trustbusters.htm
120. Jøsang, A. and N. Tran, *Trust Management for E-Commerce*. 2000.
<http://citeseer.nj.nec.com/375908.html>
121. Gunter, C.A. and T. Jim, *Policy-directed certificate retrieval*. Software: Practice & Experience, 2000. **30**(15): p. 1609 - 1640.
122. Gunter, C.A. and T. Jim. *Design of an application-level security infrastructure*. in *DIMACS Workshop on Design and Formal Verification of Security Protocols*. 1997. Piscataway, NJ, USA.
<http://dimacs.rutgers.edu/Workshops/Security/program2/jim/index.html>
123. Gunter, C.A. and T. Jim. *Generalized certificate revocation*. in *27th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. 2000. Boston, Massachusetts, USA: ACM Press.
124. Yao, W.T.-M. *Fidelis: A Policy-Driven Trust Management Framework*. in *1st International Conference on Trust Management*. 2003. Heraklion, Crete, Greece: Springer.
125. Yao, W.T.-M., K. Moody and J. Bacon. *A model of OASIS role-based access control and its support for active security*. in *Sixth ACM Symposium on Access Control Models and Technologies*. 2001. Chantilly, Va: ACM Press.
<http://citeseer.nj.nec.com/yao01model.html>
126. Hayton, R., J. Bacon and K. Moody. *OASIS: Access control in an open, distributed environment*. in *IEEE Symposium on Security and Privacy*. 1998. California, USA: IEEE Computer Society Press.
127. Bacon, J., K. Moody and W. Yao, *Access Control and Trust in the use of Widely Distributed Services*. Software - Practice and Experience, 2001. **33**: p. 375 - 394.
<http://citeseer.nj.nec.com/bacon01access.html>
128. IBM. *Access Control Meets Public Key Infrastructure, or: Assigning Roles to Strangers*. in *IEEE Symposium on Security and Privacy*. 2000.
<http://www.hrl.il.ibm.com/TrustEstablishment/paper.asp>
129. Winsborough, W.H., K.E. Seamons and V.E. Jones. *Negotiating Disclosure of Sensitive Credentials*. in *Second Conference on Security in Communication Networks*. 1999. Amalfi, Italy. <http://isrl.cs.byu.edu/pubs/TrustNegotiationFramework.pdf>

130. Seamons, K.E., M. Winslett and T. Yu. *Limiting the Disclosure of Access Control Policies During Automated Trust Negotiation*. in *Network and Distributed System Security Symposium*. 2001. San Diego, CA. <http://isrl.cs.byu.edu/pubs/ndss2001.pdf>
131. Winsborough, W.H., K.E. Seamons and V.E. Jones. *Automated Trust Negotiation*. in *DARPA Information Survivability Conference and Exposition*. 2000. Hilton Head, SC, USA. <http://isrl.cs.byu.edu/pubs/DISCEX2000.pdf>
132. Barlow, T., A. Hess and K.E. Seamons. *Trust Negotiation in Electronic Markets*. in *Eighth Research Symposium in Emerging Electronic Markets*. 2001. Maastricht, Netherlands. <http://isrl.cs.byu.edu/pubs/rseem2001.pdf>
133. Yu, T., M. Winslett and K.E. Seamons. *Interoperable Strategies in Automated Trust Negotiation*. in *ACM Conference on Computer and Communications Security (CCS)*. 2001. Philadelphia, Pennsylvania, USA: ACM. <http://isrl.cs.byu.edu/pubs/ccs2001.pdf>
134. Winsborough, W., K. Seamons and V. Jones, *Automated Trust Negotiation: Managing Disclosure of Sensitive Credentials*. 1999, Transarc.
135. Winsborough, W.H. and N. Li. *Towards Practical Automated Trust Negotiation*. in *Third International Workshop on Policies for Distributed Systems and Networks (POLICY)*. 2002. Monterey, California: IEEE Computer Society Press, Los Alamitos, California. http://crypto.stanford.edu/dc/papers/atn_policy02.pdf
136. Li, N., J.C. Mitchell and W.H. Winsborough. *Design of a Role-based Trust-Management Framework*. in *IEEE Symposium on Security and Privacy*. 2002. Berkeley, California. <http://cui.unige.ch/OSG/research/trust/li.pdf>
137. Shrobe, H., J. Doyle and P. Szolovits, *Active Trust Management for Autonomous Adaptive Survivable Systems*. 2000, DARPA Proposal, MIT, AI Lab and Lab for Computer Science. p. 27.
138. Kagal, L., S. Cost, T. Finin and Y. Peng. *A Framework for Distributed Trust Management*. in *Second Workshop on Norms and Institutions in MAS, Autonomous Agents*,. 2001. Montreal, Canada. <http://ccs.mit.edu/dell/aa2001/aa2001papers/paper4.pdf>
139. Maarof, M.A. and K. Krishna, *An Hybrid Trust Management Model For MAS Based Trading Society*. 2002. <http://citeseer.nj.nec.com/551840.html>
140. Witkowski, M., A. Artikis and J. Pitt. *Trust and Cooperation in a Trading Society of Objective Trust Based Agents*. in *Deception, Fraud and Trust in Agent Societies Workshop: Autonomous Agents*. 2000. Barcelona, Spain.
141. Simmons, G.J. *An introduction to the mathematics of trust in security protocols*. in *Computer Security Foundations Workshop VI*. 1993. <http://ieeexplore.ieee.org/iel2/466/6323/00246634.pdf>
142. Shand, B., N. Dimmock and J. Bacon. *Trust for Ubiquitous, Transparent Collaboration*. in *First IEEE International Conference on pervasive Computing and Communications*. 2003. Fort Worth, Texas: IEEE. <http://www.cl.cam.ac.uk/Research/SRG/opera/publications/Papers/percom03.pdf>
143. Viega, J., T. Kohno and B. Potter, *Trust (and Mistrust) in Secure Applications*. *Communications of the ACM*, 2001. **44**(2): p. 31 - 36.
144. Whitehead, A.N., *Science and the Modern World*, ed. T. Parsons. 1997: Simon & Schuster Adult Publishing Group. 212. 0684836394: 0684836394.

145. Carr, M.J., S.L. Konda, I. Monarch, F.C. Ulrich and C.F. Walker, *Taxonomy-based Risk Identification*. 1993, Carnegie Mellon University: Pittsburgh, Pennsylvania.
146. *Introduction to Risk Analysis*. 2001. <http://www.security-risk-analysis.com/intoduction.htm>
147. Boehm, B.W., *Software Risk Management: Principles and Practices*. 1991.
148. *Information Security Risk Management Guidelines*. 2000, Standards Australia: Strathfield.
149. *Risk Management - AS:NZS 4360:1999*. 1999.
150. *Software Risk Abatement*. 1988, United States Air Force.
151. Soo Hoo, K.J., *How Much is Enough? A Risk management Approach to Computer Security*. 2000, Working Paper, Consortium for Research on Information Security and Policy (CRISP), Stanford University. p. 88.
152. Shafer, G., *A Mathematical Theory of Evidence*. 1976, Princeton, N.J.: Princeton University Press.
153. *Advances in the Dempster-Shafer Theory of Evidence*, ed. R.R. Yager, J. Kacprzyk, and M. Fedrizzi. 1994: John Wiley & Sons. 0471552488: 0471552488.
154. Martin, D., *Managing B2B E-Commerce Risks*. 2001. <http://www.law.washington.edu/ABA-eADR/documentation/docs/geotrust.pdf>
155. Kuester, J.R. and L.E. Thompson, *Risks associated with restricting business method and E-Commerce Patents*. Georgia State University Law Review, 2001. **17**(657): p. 657 - 688. <http://www.ftc.gov/os/comments/intelpropertycomments/kuesterjeffreyr.pdf>
156. Kamthan, P., *E-Commerce on the WWW : A Matter of Trust*. 1999. <http://tech.irt.org/articles/js158/index.htm>
157. Hrebicek, O., *Computers and E-Commerce: Issues, Risks and Concerns*. 2002, Illionis Institute of Technology. <http://www.csam.iit.edu/~cs485/spring2002/presentations/Hrebicek/issues.html>
158. Lehman, M.M. *Uncertainty in Computer Application is Certain - Software Engineering as a Control*. in *Euro Comp*. 1990. Tel Aviv.: IEEE.
159. Gagnon, E., *SableCC: An Object-Oriented Compiler Framework*, in *School of Computer Science*. 1998, McGill University: Montreal.
160. Hager, T., in *Force of Nature*. 1995, Simon and Schuster: New York, USA. p. 86.
161. *The European Internet Report*. 1999. Industry Report, Morgan Stanley Dean Witter. p. 362. <http://www.morganstanley.com/institutional/techresearch/euroinet.html?page=research>
162. *The Writings of Friedrich Nietzsche*. <http://plato.stanford.edu/entries/nietzsche/>
163. Keromytis, A.D., *Scalable Security Policy Mechanisms*. 2001. Technical Report, University of Pennsylvania CIS Dept: Pennsylvania.
164. Keromytis, A.D., S. Ioannidis, M.B. Greenwald and J.M. Smith. *The STRONGMAN Architecture*. in *Third DARPA Information Survivability Conference and Exposition (DISCEX III)*. 2003. Washington, D.C. <http://www.cis.upenn.edu/~sotiris/papers/discex03.ps>

165. Keen, P.G.W., C. Ballance, S. Chan and S. Schrupp, *Electronic Commerce and the Concept of Trust*, in *Electronic Commerce Relationships: Trust by Design*, P. Ken, Editor. 1999, Prentice Hall PTR.
166. Dimitrakos, T., *TrustCoM - Enabling the on-demand creation and self-management of secure, dynamic and scaleable virtual organisations*.
http://eoi.cordis.lu/docs/int_32192.pdf
167. Abie, H., *Distributed Digital Rights management (DRM) and Security Model for Distributed Multimedia Systems*. 2003. <http://www.nr.no/~abie/TrustPolicy.htm>
168. Xiaoqi, L., *Trust Model based Self-Organized Routing Protocol for Secure Ad Hoc Networks*, ,PhD Term Paper.
<http://www.cse.cuhk.edu.hk/~lyu/student/phd/gigi/term2.pdf>
169. Carbone, M., M. Nielsen and V. Sassone. *A Formal Model for Trust in Dynamic Networks*. in *International Conference on Software Engineering and Formal Methods*. 2003. Brisbane, Australia.
170. Jøsang, A. and T. Grandison. *Conditional Inference in Subjective Logic*. in *6th ISIF International Conference of Information Fusion*. 2003. Cairns, Australia.

Appendix A Syntax Specification

The following is the grammar of the SULTAN specification notation, written using SableCC. It can be used to generate the lexical and syntactic parser of the SULTAN compiler.

```
/******  
* SULTAN - Trust Specification Language  
*  
* Author: SULTAN Implementation Group,  
*         Distributed Software Engineering Group,  
*         Department of Computing  
*         Imperial College  
*  
* Last Date Modified: Feb 18, 2002  
*  
* Grammar written in: SableCC  
*****/  
  
Package sultan.spec.compile;  
  
/******  
HELPERS  
*****/  
  
Helpers  
  
    all           =      [0 .. 127];  
  
    cr            =      13;  
    lf            =      10;  
    eol           =      cr | lf | cr lf;  
    tab           =      9;  
    non_eol       =      [all - [cr + lf]];  
    not_star      =      [all - '*'];  
    not_star_slash =      [not_star - '/'];  
    not_db        =      [all-""];  
  
    bracketed     =      "" not_db* "";  
  
    underscore    =      '_';  
    letter        =      ['A' .. 'Z'] | ['a' .. 'z'];  
    digit         =      ['0' .. '9'];  
  
    dash          =      '-';  
  
    l_comment     =      '/' non_eol* eol;  
    c_comment     =      '/'* not_star* '*'+ (not_star_slash not_star* '*'+)* '/';  
  
/******  
TOKENS  
*****/  
  
Tokens  
  
/****** White Spaces *****/  
comments = l_comment | c_comment;  
blanks   = ( eol | tab | ' ' )+;
```

```

/***** Main predicates *****/
trust      =      'trust';
recommend  =      'recommend';

everyone   =      'everyone';
ispartof   =      'isPartOf';
foreign    =      'foreign';
not        =      'not';
risk       =      'risk';
experience =      'experience';
trustcp    =      'trust+';
trustcn    =      'trust-';
reccp     =      'recommend+';
reccn     =      'recommend-';

/*****
reference   =      letter (underscore | letter | digit)*;
variable   =      underscore (underscore | letter | digit)*;
number     =      (dash? ['1' .. '9'] digit*);

/***** Punctuation Symbols Allowed *****/
colon      =      ':';
comma      =      ',';
openpr     =      '(';
closepr    =      ')';
implies    =      '<=';
semicolon  =      ';';
dquote     =      '"';

/***** Boolean Operators *****/
gt         =      '>';
lt         =      '<';
eq         =      '=';
neq        =      '!=';
leq        =      '<=';
geq        =      '>=';

/***** Logical Operators *****/
and        =      '&';
or         =      '|';

/*****
quoted     =      bracketed;

```

```

/*****
  IGNORED TOKENS
  *****/

```

```

Ignored Tokens
  blanks,
  comments;

```

```

/*****
  PRODUCTIONS
  *****/

```

```

Productions

  specification =      statement*;

```

```

statement      =
    {trust}      trust_stat |
    {recommend}  recommend_stat;

trust_stat     =
    reference colon trust_lhs option? semicolon;

recommend_stat =
    reference colon recommend_lhs option? semicolon;

trust_lhs      =
    trust openpr subject [c1]:comma target [c2]:comma action_set [c3]:comma
    level closepr;

recommend_lhs  =
    recommend openpr subject [c1]:comma target [c2]:comma action_set
    [c3]:comma level closepr;

subject        =
    {entity}     reference |
    {var}        variable;

target         =
    {entity}     reference |
    {var}        variable |
    {everyone}   everyone;

action_set     =
    {action}     actions more_action* |
    {var}        variable;

level          =
    {num}        number |
    {ref}        reference |
    {var}        variable;

actions        =
    {act}        action |
    {notact}     not openpr action more_act* closepr;

action         =
    reference openpr act_params+ closepr;

act_params     =
    param more_param*;

param          =
    {num}        number |
    {ref}        reference |
    {string}     strg |
    {var}        variable;

more_param     =
    comma param;

more_act       =
    colon action;

more_action    =
    colon actions;

option         =
    implies constraints;

```

```

constraints      =
    constraint more_constraint*;

constraint       =
    {trust}      trust_lhs |
    {recommend}  recommend_lhs |
    {trust_cp}   trust_pos |
    {trust_cn}   trust_neg |
    {rec_cp}     rec_pos |
    {rec_cn}     rec_neg |
    {func}       func |
    {ref}        reference |
    {expr}       expr |
    {risk_expr}  riskexpr |
    {exp_expr}   expexpr |
    {part_of}    ispartof_const;

trust_pos       =
    trustcp openpr subject [c1]:comma target [c2]:comma action_set closepr;

trust_neg       =
    trustcn openpr subject [c1]:comma target [c2]:comma action_set closepr;

rec_pos        =
    reccp openpr subject [c1]:comma target [c2]:comma action_set closepr;

rec_neg        =
    reccn openpr subject [c1]:comma target [c2]:comma action_set closepr;

func           =
    reference openpr parameters? closepr;

expr           =
    {func}      func op value |
    {var}       variable op value;

parameters     =
    parameter more_para*;

op             =
    {gt}        gt |
    {lt}        lt |
    {eq}        eq |
    {neq}       neq |
    {leq}       leq |
    {geq}       geq ;

value         =
    {num}       number |
    {ref}       reference |
    {string}    strg;

action_name    =
    {ref}       reference |
    {func}     func;

parameter     =
    {num}       number |
    {ref}       reference |
    {string}    strg |
    {var}       variable |

```

```

    {func} func;
more_para    =
    comma parameter;
strg        =
    quoted;
riskexpr    =
    risk openpr subject [c1]:comma target [c2]:comma action_set closepr
    op number;
expexpr     =
    experience openpr subject [c1]:comma target [c2]:comma action_set closepr
    op number;
ispartof_const    =
    ispartof openpr child comma parent closepr;
child        =
    {var} variable |
    {ent} reference;
parent       =
    {var} variable |
    {ent} reference |
    {for} foreign;
more_constraint =
    log_op constraint;
log_op       =
    {and} and |
    {or} or ;
```


Appendix B Missing Definitions

In this appendix, the basic definitions used for the syntax elements in the discussion of the SULTAN specification language in Chapter 3 are presented.

Building Blocks

The following are the basic definitions that will be used in Chapter 3's definition of the SULTAN notation:

```
letter      = 'A'..'Z' | 'a'..'z';
underscore = '-';
digit       = '0'..'9';
```

Comments

SULTAN comments can either be multiple-line or single-line. A multiple-line comment starts with the characters `/*` and terminates with the characters `*/`, while a single-line comment starts with `//` and terminates at the end of the line.

Keywords

The following words are reserved as keywords in the SULTAN specification language:

trust	recommend	everyone	not
risk	experience	trust+	trust-
recommend+	recommend-	isPartOf	foreign

Operators

The relational operators that can be used in SULTAN are:

>	<	=	!=
<=	>=		

The logical operators are:

&	
---	--

String Literals

String literals are any sequence of characters that are enclosed by the “ character.

Functions

Be aware that trust and recommend constraints have the same syntactic structure as ordinary function calls. Thus, the distinction is not made here.

function = reference '(' parameters? '');

Function Parameters

A function parameter list is a comma-delimited list of parameters, where a parameter can be a number, reference, string or function call.

parameter = variable | number | reference | string | function;
parameters = parameter (' parameter)*;

Appendix C Refining SULTAN to Trust Rules

This appendix outlines the transformation from SULTAN specifications to Prolog rules. The details of the SULTAN to Prolog translator are not presented. Only the input and output of the translation process is given. For this discussion, SICStus Prolog will be referred to simply as Prolog.

Translating the trust construct

The Prolog equivalent to the SULTAN trust construct has the following form:

```
trust(trustor, trustee, actions, level, policy) :- constraints.
```

where

- trustor and trustee are standard Prolog atoms or variables,
- actions is a list of function names,
- level is a number (It is assumed that all labels used in a SULTAN specification are converted before translation into Prolog),
- policy is the name of the policy (this must be a Prolog atom),
- constraints are the conditions that must be true for the rule to be true.

In order to provide a feel of a translated specification, we provide Prolog-compatible examples of some of the examples cited in Chapter 3. Note that the examples have not undergone the strict translation scheme because this is not necessary to demonstrate our point.

Examples:

```
trust(supplier, customers, [view_pages(supplier)], 100, customerver) :-  
  goodcredit(customers),  
  risk(supplier, customers, _, X),  
  X >= 2.
```

```
trust(jenny, realtor, [send_deals(realtor, jenny)], 100, realtor) :-  
  trust(jenny, tom, [provideinfo(jenny)], 50, _);  
  trust(tom, realtor, [send_deals(realtor, tom)], 50, _).
```

```
trust(morris, symantec, [do_definition_update(morris, computer)], 100, pda) :-  
  eq(definitionstate(symantec), "old").
```

Translating the recommend construct

A SULTAN recommend construct is translated to a Prolog statement of the following form:

```
recommend(recommendor, recommendee, actions, level, policy) :- constraints.
```

where

recommendor and recommendee are standard Prolog atoms or variables,
 actions is a list of function names,
 level is a number,
 policy is the name of the policy,
 constraints are the conditions that must be true for the rule to be true.

To provide a feel of the Prolog representation of a recommendation, Prolog-compatible versions of some of the examples cited in Chapter 3 will be presented.

Examples:

```
recommend(natwest, Client, [getcredit(Client, SwitichCard)], 100, credit) :-
  isClient(natWest, Client),
  isValidCard(natwest, SwitichCard).

recommend ( verisign, KeyHolder, loadScript(X), -50, veri) :-
  isCustomer(veriSign, KeyHolder),
  isUsedBy(verisSign, X),
  gt(outStandingBalance(veriSign, KeyHolder), 40);

recommend(ucl, openu, [do_research(openu)], -10, attpol) :-
  researchquality(openu, X),
  X =< 3.
```

Translating Policy Names

A policy name in SULTAN is defined as a sequence of letters, numbers and underscores (an underscore not being the first character). To allow for easy translation from SULTAN to Prolog and from Prolog back to Sultan, the SULTAN policy name is prefixed with the characters ‘p_’ to get the Prolog equivalent.

Examples:

<u>SULTAN Policy Name</u>	<u>Prolog Equivalent</u>
SupplierDemand	p_SupplierDemand
web_access_policy	p_web_access_policy
mobile_1200	p_mobile_1200

Translating Entity Names

As stated in Chapter 3, entity names can be either references or variables. For conversion into Prolog, references are prefixed with the characters ‘e_’ and variables are translated by prefixing ‘V_’. A variable that consists of only the underscore (an anonymous variable) undergoes the normal translation plus a four-digit counter is appended to it. This counter represents the relative position of this variable in the set of anonymous variables.

Examples:

<u>SULTAN Entity Name</u>	<u>Prolog Equivalent</u>
Supplier	e_Supplier
_Customer	V__Customer
_	V__0001

Translating Actions

A SULTAN action is translated by prefixing 'f_' to it. All actions have one or more arguments, which can be either a number, reference, string or function call. Numbers are converted verbatim. For strings, the opening and closing double quotation marks are replaced by single quotation marks. A reference is converted using the same method used to translate an entity.

Examples:

<u>SULTAN Function Name</u>	<u>Prolog Equivalent</u>
acquire(Tony)	f_acquire(e_Tony).
request(Info)	f_request(e_Info);
Control(_X, Router)	f_Control(V__X, e_Router)
load(page, "http")	f_load(e_page, 'http')

Standard Action Delimiter

The standard action delimiter for SULTAN is the colon. This is converted to a comma for its Prolog representation.

Examples:

<u>SULTAN Action Set</u>	<u>Prolog Equivalent</u>
startup(MyComputer):pause(MyComputer)	f_startup(e_MyComputer), f_pause(e_MyComputer)

The not function

When an action has a not surrounding it, this represents the restriction of the performance of these actions. To convert action sets that include the not function, the functions must be translated using the rules outlined in this section. The word not is kept in the Prolog version.

Examples:

<u>SULTAN Action Set</u>	<u>Prolog Equivalent</u>
not(acquire(tony):integrate(circuit))	not(f_acquire(e_tony), f_integrate(e_circuit))
not(Control(_X,Router))	not(f_Control(V__X, e_Router))

Action Sets

In Prolog, an action set must be enclosed in square brackets. The process is that the actions are translated and then placed within square brackets.

Example:

SULTAN

```
WebUserCheck: trust(WebServer, _User, access_se(WebServer):view_pages(WebServer):
    write_pages(WebServer):logintoWeb(WebServer, ID,PASS) , 10 )
← RealEstatePassport(_User);
```

Prolog

```
trust(e_WebServer, V__User, [ f_access_se(e_WebServer), f_view_pages(e_WebServer),
    f_write_pages(e_WebServer), f_logintoWeb(e_WebServer, e_ID,e_PASS) ], 10,
    p_WebUserCheck) :-
    f_RealEstatePassport(V__User).
```

Translating Levels

It is assumed that levels used in a Prolog representation of a SULTAN specification are numeric. If a SULTAN specification uses a label as its level, it is assumed that the label is translated to a number before the specification is translated to Prolog.

Translating Constraints

Constraints are the conditions that must be satisfied for a policy to be active. A policy may have a collection of constraints separated by & (logical and) and | (logical or). These delimiters are translated to , (logical *and* in Prolog) and ; (logical *or* in Prolog) respectively. A closer look is taken of the various types of constraints that a SULTAN policy can have and how each is translated to Prolog.

Translation of Trust Policy Constraints

When used as a constraint, a trust policy has the following form:

```
trust(trustor, trustee, actions, level)
```

To translate the above, the constituent elements (trustor, trustee, actions, level) are translated and then a placeholder is inserted for the policy name. Thus, the Prolog equivalent is:

```
trust(trustor, trustee, actions, level, _)
```

To translate the trust+ and trust-, the same procedure is followed and a level constraint is added. For example:

trust+(trustor, trustee, actions,)

becomes:

trust(trustor, trustee, actions, X, actions, _), X > 0

Translation of Recommend Policy Constraints

When used as a constraint, a recommend policy has the following form:

recommend(recommendor, recommendee, actions, level)

Its translation to Prolog involves translating the constituent elements (recommendor, recommendee, actions, level) and then inserting a placeholder for the policy name. Thus, the Prolog equivalent is:

recommend(recommendor, recommendee, actions, level, _)

To translate the recommend+ and recommend-, the same procedure is followed and a level constraint is added. For example:

recommend-(recommendor, recommendee, actions)

becomes:

recommend(recommendor, recommendee, actions, X, actions, _), X < 0

User-Defined Constraints

User-defined constraints are constraints that are specific to the user's domain. These constraints can either be function calls, object method calls or expressions. For user-defined expression constraints, the SULTAN operator symbols are translated according to the following table:

SULTAN Symbol	Prolog Predicate
>	gt(lhs, rhs)
<	lt(lhs, rhs)
>=	gteq(lhs, rhs)
<=	Lteq(lhs, rhs)
=	eq(lhs, rhs)
!=	neq(lhs, rhs)

It is assumed that both the left hand side (lhs) and right hand side (rhs) of the expression are translated using the rules previously described.

Examples:*SULTAN*

Law: trust(Client, ELawyers, advice(Client), 100)
 ← Accredited(ELawyers, USBar);

Prolog

trust(e_Client, e_ELawyers, [f_advice(e_Client)], 100, p_Law) :-
 f_Accredited(e_ELawyers, e_USBar).

SULTAN

Doc: recommend(BMA, EDoctor, sell_drugs_online(EDoctor), 100)
 ← certified(EDoctor, BMA);

Prolog

recommend(e_BMA, e_EDoctor, [f_sell_drugs_online(e_EDoctor)], 100, p_Doc) :-
 f_certified(e_EDoctor, e_BMA).

SULTAN

Site: trust(I, WebSites, load(I), -100)
 ← SiteSecurityLevel(WebSites) < 3;

Prolog

trust(e_I, e_WebSites, [f_load(e_I)], -100, p_Site) :-
 !t(f_SiteSecurityLevel(e_WebSites), 3);

Translation of Auxiliary Functions

All trustees and trustors are viewed as entitles. And there are two auxiliary functions defined on each of these entities that provide useful functionality when specifying trust specifications.

Risk

The SULTAN risk constraint has the following format:

risk(B, C, A)

The risk method returns an integer value and the method must be used in a Boolean expression. To translate it to Prolog, we assume the existence of a predicate that has an arity of 4. Then SULTAN terms B (an entity), C (an entity) and A (an action set) are translated using the scheme that have outlined. Thus, a risk constraints has the following form:

risk(B, C, A) op riskvalue

We translate such a constraint to:

risk(B,C, A, R), R op riskvalue

Examples:*SULTAN*

Contract: recommend(College, Sun, access_internal_web(College), -50)
 ← risk(College, Sun, access_internal_web(College)) <= 3;

Prolog

recommend(e_College, e_Sun, [f_access_internal_web(e_College)], -50, p_Contract) :-
 risk(e_College, e_Sun, [f_access_internal_web(e_College)], R), R <= 3.

SULTAN

Amaz: trust(Amazon, _AnyOne, buy_product(Amazon), -5)
 ← .risk(Amazon, _Anyone, _) > 10;

Prolog

trust(e_Amazon, V__AnyOne, [f_buy_product(e_Amazon)], -5, p_Amaz) :-
 risk(Amazon, V__Anyone, , V__0002, R), R > 10.

Experience

The SULTAN experience method has the following form:

experience(B, C, A)

This method returns an integer, which must also be used in an expression. As such, its translation process is exactly the same as that for a risk constraint. Thus, a experience constraint has the following form:

experience(B, C, A) op expvalue

Will be translated to:

experience(B, C, A, E), E op expvalue

It is assumed that B, C and A are translated beforehand.

Examples:*SULTAN*

Sup: recommend(EDistributor, EReseller, market(EReseller), 100)
 ← experience(EDistributor, EReseller, _) > 0;

Prolog

recommend(e_EDistributor, e_EReseller, [f_market(e_EReseller)], 100, p_Sup) :-
 experience(e_EDistributor, e_EReseller, V__0003, E), E > 0.

SULTAN

Rod: trust(Police, GSM, provide_info(GSM, Police), 100)
 ← experience(Police, GSM, provide_info(GSM, Police)) < 0;

Prolog

trust(e_Police, e_GSM, [f_provide_info(e_GSM, e_Police)], 100, p_Rod) :-
 experience(e_Police, e_GSM, [f_provide_info(e_GSM, e_Police)], E), E < 0.

Appendix D SULTAN Analysis Model

The SULTAN Analysis Model is a set of rules that enable reasoning about trust specifications and recommendations.

```
/******  
/* Author: Tyrone W.A. Grandison */  
/* Last Date Modified: Aug 10, 2002 */  
/* Purpose: This is a model that allows for the analysis of trust and recommend */  
/* rules. */  
/******  
  
/******  
/* Input Output Predicates */  
/* Input Output Predicates */  
/******  
  
/* Reads in input (Variables)-(Conditions) from Keyboard. */  
readin :-  
    write("\n\n"),  
    write('Please enter your query :\n'),  
    write(' - The format is \n'),  
    write(' (Variables)-(Conditions) - for a scenario or source query . \n '),  
    write(' A trust or recommend statement for an abduction query, and \n '),  
    write(' The word cycle or make_acyclic for cycle investigation and correction. \n '),  
    write(' or quit to exit. \n '),  
    read(X),  
    test(X).  
  
/* Processes User Input */  
test(quit) :- /* Exit UI */  
    write('\nGoodbye'),  
    fail.  
  
test((X)-S) :- /* Source or Scenario Query */  
    query(X, S, Answer),  
    write('Solution(s) : \n'),  
    pretty(Answer),  
    write('\n'),  
    readin.  
  
test(A) :- /* Abductive Query */  
    query(A, Result),  
    pretty(Result),  
    write('\n'),  
    readin.  
  
test(cycle) :- /* Cycle Detection  
    query(cycle, Result),  
    pretty(Result),  
    write('\n'),  
    readin.
```

```

test(make_acyclic) :- /* Cycle Resolution
    query(make_acyclic,
    error_wrapper(listing(trust), _),
    error_wrapper(listing(recommend), _),
    write("\n"),
    readin.

/* Formats the screen output */
pretty([]).

pretty([A|B]) :-
    write(' '),write(A),
    write("\n"), pretty(B).

/* Main Working Predicate */

query(A, Result) :- /* Abductive Query */
    ( (functor(A, trust, 5),A=trust(_,_._,P));
      (functor(A, recommend, 5),A=recommend(_,_._,P))
    ),
    ( \+ cycle(P,_),
      cycles(CycList),
      ( empty(CycList);
        (not_empty(CycList), write('Cycles Detected : '),
          write(CycList),write("\n"))
        ),
      abduce(A, Result)
    );
    ( cycle(P,Y),
      write('There is a cycle between rules '),
      write(P),write(' and '), write(Y),write("\n"),
      insert(error,[.: cycle, P,Y], Result)
    ).

query(cycle, Result) :- /* Cycle Detection */
    cycles(Result).

query(make_acyclic) :- /* Cycle Resolution */
    make_acyclic.

query(X, S, Answer) :- /* Source and or Scenario Query */
    findall(X, S, A),
    remove(A, Answer).

/*****
/*
/*                               */
/*                               */
/*                               */
/*                               */
/*****

/*****
/*   Querying Basic Types & Attributes   */
/*****

/* P is a SULTAN trust/recommend rule. */
p_policy(P) :-
    p_rec_pol(P);
    p_trust_pol(P).

```

```

/* P is a recommend rule. */
p_rec_pol(P) :-
    clause(recommend(_, _, _, P), _).

/* P is a trust rule. */
p_trust_pol(P) :-
    clause(trust(_, _, _, P), _).

/* P is a positive trust rule. */
p_pos_trust(P) :-
    clause(trust(_, _, _, L, P), _),
    integer(L), L > 0.

/* P is a negative trust rule.*/
p_neg_trust(P) :-
    clause(trust(_, _, _, L, P), _),
    integer(L), L < 0.

/* P is a positive recommend rule. */
p_pos_rec(P) :-
    clause(recommend(_, _, _, L, P), _),
    integer(L), L > 0.

/* P is a negative recommend rule. */
p_neg_rec(P) :-
    clause(recommend(_, _, _, L, P), _),
    integer(L), L < 0.

/* E is the entity in rule P. */
p_entity(E, P) :-
    p_subject(E, P) ;
    p_target(E, P).

/* E is the subject of rule P. */
p_subject(E,P) :-
    p_trustor(E,P);
    p_recommendor(E,P).

/* rules P1 and P2 have the same subject. */
p_sameSub(P1, P2) :-
    p_subject(E, P1),
    p_subject(E, P2).

/* E is the target of rule P. */
p_target(E,P) :-
    p_trustee(E,P);
    p_recommendeer(E,P).

/* E is the trustor of rule P. */
p_trustor(E, P) :-
    clause(trust(E, _, _, P), _).

p_trustor(E, P) :- /* E is a trustor of rule P through an isPartOf rule */
    clause(trust(M, _, _, P), _),
    partOf(E,M).

/* E is the trustee of rule P. */
p_trustee(E, P) :-
    clause(trust(_, E, _, P), _).

p_trustee(E, P) :- /* E is a trustee of rule P through an isPartOf rule */
    clause(trust(_, M, _, P), _), partOf(E,M).

```

```

/* E is the recommendor of rule P. */
p_recommendor(E, P) :-
    clause(recommend(E, _, _, P), _).

p_recommendor(E, P) :- /* E is a recommendor of rule P through an isPartOf rule */
    clause(recommend(M, _, _, P), _),
    partOf(E,M).

/* E is the recommendee of rule P. */
p_recommendee(E, P) :-
    clause(recommend(_, E, _, P), _).

p_recommendee(E, P) :- /* E is a recommendee of rule P through an isPartOf rule */
    clause(recommend(_, M, _, P), _),
    partOf(E,M).

/* L is the level associated with rule P. */
p_level(L, P) :-
    ( clause(trust(_, _, L, P), _) ;
      clause(recommend(_, _, L, P), _) ).

/* C is the set of constraints associated with rule P. */
p_constraints(C, P):-
    ( clause(trust(_, _, L, P), C) ;
      clause(recommend(_, _, L, P), C) ),
    not_empty_constraint(C).

/* A is the actionset associated with rule P. */
p_actionset(A, P) :-
    clause(trust(_, _, A, P), _) ;
    clause(recommend(_, _, A, P), _).

/* A is a subset of the actionset of rule P. */
p_actions(A, P) :-
    ( clause(trust(_, _, Action, P), _) ;
      clause(recommend(_, _, Action, P), _) ),
    subset(A, Action).

/* Entity E is trusted by exactly N other entities
   at level L to perform action(s) A.*/
p_trustedby(E, N, L, A, e) :-
    findall(T, (p_trustorsWithAS(E, A, L, T)), No),
    remove_dup(No, Pol),
    count(Pol, NoPol),
    N = NoPol.

/* Entity E is trusted by at least N other entities
   at level L to perform action(s) A. */
p_trustedby(E, N, L, A, a) :-
    findall(T, (p_trustorsWithAS(E, A, L, T)), No),
    remove_dup(No, Pol),
    count(Pol, NoPol),
    N =< NoPol.

/* C is the set of constraints for the rules
   that relate entities X and Y. */
p_constraints(X, Y, C) :-
    findall(P, (clause(trust(X, Y, _, _), P), not_empty_constraint(P) ), A),
    findall(Q, (clause(recommend(X, Y, _, _), Q), not_empty_constraint(Q) ), B),
    findall(R, (clause(trust(Y, X, _, _), R), not_empty_constraint(R) ), E),
    findall(S, (clause(recommend(Y, X, _, _), S), not_empty_constraint(S) ), D),
    union(A,B, H), union(H, E, G), union(G, D, L), remove_dup(L, C).

```

```
/* Finding constraints by looking through the organisational chart links */
```

```
p_constraints(X, Y, C) :-
    partOf(X,M),
    p_constraints(M,Y,C).
```

```
p_constraints(X, Y, C) :-
    partOf(Y,M),
    p_constraints(X,M,C).
```

```
p_constraints(X, Y, C) :-
    partOf(X,M),
    partOf(Y,N),
    p_constraints(M,N,C).
```

```
/******  
/* Defining SourceShortHand Predicates */  
/******
```

```
/* X and Y have the same subject */
```

```
p_commonSubj(X,Y) :-
    p_subject(E,X),
    p_subject(E,Y),
    X \== Y.
```

```
/* X and Y have the same subject E */
```

```
p_commonSubj(X,Y,E) :-
    p_subject(E,X),
    p_subject(E,Y),
    X \== Y.
```

```
/* X and Y have the same target */
```

```
p_commonTar(X,Y) :-
    p_target(E,X),
    p_target(E,Y),
    X \== Y.
```

```
/* X and Y have the same target E */
```

```
p_commonTar(X,Y,E) :-
    p_target(E,X),
    p_target(E,Y),
    X \== Y.
```

```
/* X and Y have the same trustor */
```

```
p_commonTrustor(X,Y) :-
    p_trustor(E,X),
    p_trustor(E,Y),
    X \== Y.
```

```
/* X and Y have the same trustor E */
```

```
p_commonTrustor(X,Y,E) :-
    p_trustor(E,X),
    p_trustor(E,Y),
    X \== Y.
```

```
/* X and Y have the same trustee */
```

```
p_commonTrustee(X,Y) :-
    p_trustee(E,X),
    p_trustee(E,Y),
    X \== Y.
```

```

/* X and Y have the same trustee E */
p_commonTrustee(X,Y,E) :-
    p_trustee(E,X),
    p_trustee(E,Y),
    X \== Y.

/* X and Y have the same recommendor */
p_commonRecommendor(X,Y) :-
    p_recommendor(E,X),
    p_recommendor(E,Y),
    X \== Y.

/* X and Y have the same recommendor E */
p_commonRecommendor(X,Y,E) :-
    p_recommendor(E,X),
    p_recommendor(E,Y),
    X \== Y.

/* X and Y have the same recommendee */
p_commonRecommendee(X,Y) :-
    p_recommendee(E,X),
    p_recommendee(E,Y),
    X \== Y.

/* X and Y have the same recommendee E */
p_commonRecommendee(X,Y,E) :-
    p_recommendee(E,X),
    p_recommendee(E,Y),
    X \== Y.

/* X and Y have the same level */
p_equalLevel(X,Y) :-
    p_level(L,X),
    p_level(L,Y),
    X \== Y.

/* X and Y have the same level L */
p_equalLevel(X,Y,L) :-
    p_level(L,X),
    p_level(L,Y),
    X \== Y.

/* X and Y have the same actionset */
p_commonAS(A, B) :-
    p_actionset(APR, A),
    p_actionset(ANR, B),
    intersect(APR, ANR, ACTR),
    not_empty(ACTR).

/* Computes Action Set Equality using Organisation Chart Info */
p_commonAS(A, B) :-
    p_commonAS(A, B, X),
    not_empty(X).

p_commonAS(A, B, Result) :-
    p_actionset(AA, A),
    p_actionset(AB, B),
    find_commonAS(AA, AB, R), not_empty(R), concat([], R, Result).

```

```

/*****
/*
/*          Actual Scenarios          */
/*
/*****

/*****
/*          Querying Types & Attributes          */
/*****

/* P is a SULTAN trust/recommend rule. */
policy(P) :-
    rec_pol(P) ;
    trust_pol(P).

/* P is a recommend rule. */
/* To avoid nasty run-time errors, scenario predicates were implemented
   in a way to eliminate excessives to constraints that may be undefined
   at the moment of execution */
rec_pol(P) :-
    clause(recommend(_ , _ , _ , P),C),
    empty_or_met(C).

/* P is a trust rule. */
trust_pol(P) :-
    clause(trust(_ , _ , _ , P),C),
    empty_or_met(C),
    look_for_cycle(P).

/* P is a positive trust rule. */
pos_trust(P) :-
    clause(trust(_ , _ , _ , L, P),C),
    integer(L), L > 0, empty_or_met(C),
    look_for_cycle(P).

/* P is a negative trust rule. */
neg_trust(P) :-
    clause(trust(_ , _ , _ , L, P),C),
    integer(L), L < 0, empty_or_met(C),
    look_for_cycle(P).

/* P is a positive recommend rule. */
pos_rec(P) :-
    clause(recommend(_ , _ , _ , L, P),C),
    integer(L), L > 0, empty_or_met(C),
    look_for_cycle(P).

/* P is a negative recommend rule. */
neg_rec(P) :-
    clause(recommend(_ , _ , _ , L, P),C),
    integer(L), L < 0, empty_or_met(C),
    look_for_cycle(P).

/* E is the entity in rule P. */
entity(E, P) :-
    subject(E, P) ;
    target(E, P).

/* E is the subject of rule P. */
subject(E,P) :-
    trustor(E,P);
    recommendor(E,P).

```



```

/* E is the target of rule P. */
target(E,P) :-
    trustee(E,P);
    recommendee(E,P).

/* E is the trustor of rule P.*/
trustor(E, P) :-
    clause(trust(E, _, _, P),C),
    empty_or_met(C), look_for_cycle(P).

trustor(E, P) :-
    clause(trust(M, _, _, P),C),
    partOf(E,M), empty_or_met(C), look_for_cycle(P).

/* E is the trustee of rule P. */
trustee(E, P) :-
    clause(trust(_, E, _, P),C),
    empty_or_met(C), look_for_cycle(P).

trustee(E, P) :- /* E is the trustee of rule P through through organisational chart links. */
    clause(trust(_, M, _, P),C),
    partOf(E,M),empty_or_met(C),
    look_for_cycle(P).

/* E is the recommendor of rule P. */
recommendor(E, P) :-
    clause(recommend(E, _, _, P),C),
    empty_or_met(C), look_for_cycle(P).

recommendor(E, P) :- /* E is the recommendor of rule P thru org. chart. */
    clause(recommend(M, _, _, P),C),
    partOf(E,M), empty_or_met(C), look_for_cycle(P).

/* E is the recommendee of rule P. */
recommendee(E, P) :-
    clause(recommend(_, E, _, P),C),
    empty_or_met(C), look_for_cycle(P).

recommendee(E, P) :- /* E is the recommendee of rule P thru org. chart. */
    clause(recommend(_, M, _, P),C),
    partOf(E,M), empty_or_met(C), look_for_cycle(P).

/* L is the level associated with rule P. */
level(L, P) :-
    ( clause(trust(_, _, L, P),C) ;
      clause(recommend(_, _, L, P),C)
    ),
    empty_or_met(C), look_for_cycle(P).

/* C is the set of constraints associated with rule P. */
constraints(C, P):-
    ( clause(trust(_, _, _, P), C) ;
      clause(recommend(_, _, _, P),C)
    ),
    empty_or_met(C), look_for_cycle(P).

/* A is the actionset associated with rule P. */
actionset(A, P) :-
    ( clause(trust(_,_,A,_,P),C);
      clause(recommend(_,_,A,_,P),C)
    ), empty_or_met(C), look_for_cycle(P).

```

```

/* A is a subset of the actionset of rule P. */
actions(A, P) :-
    ( clause(trust(_,_ ,Action,_ ,P),C) ;
      clause(recommend(_,_ ,Action,_ ,P),C)
    ),
    empty_or_met(C),
    subset(A, Action), look_for_cycle(P).

/* Entity E is trusted by exactly N other entities
   at level L to perform action(s) A. */
trustedby(E, N, L, A, e) :-
    findall(T, (trustorsWithAS(E, A, L, T)), No),
    remove_dup(No, Pol),
    count(Pol, NoPol),
    N = NoPol.

/* Entity E is trusted by at least N other entities
   at level L to perform action(s) A. */
trustedby(E, N, L, A, a) :-
    findall(T, (trustorsWithAS(E, A, L, T)), No),
    remove_dup(No, Pol),
    count(Pol, NoPol),
    N =< NoPol.

/* C is the set of constraints for the rules that
   relate entities X and Y.*/
constraints(X, Y, C) :-
    findall(P, (clause(trust(X,Y,_ ,_ ,_ ), P), empty_or_met(P) ), A),
    findall(Q, (clause(recommend(X,Y,_ ,_ ,_ ), Q),empty_or_met(Q) ), B),
    findall(R, (clause(trust(Y,X,_ ,_ ,_ ), R), empty_or_met(R) ), E),
    findall(S, (clause(recommend(Y,X,_ ,_ ,_ ), S), empty_or_met(S) ), D),
    union(A,B, H), union(H, E, G), union(G, D, L),
    remove_dup(L, C), look_for_cycle(P).

/* Looking through the organisational chart */
constraints(X, Y, C) :-
    partOf(X,M),
    constraints(M,Y,C).

constraints(X, Y, C) :-
    partOf(Y,M),
    constraints(X,M,C).

constraints(X, Y, C) :-
    partOf(X,M),
    partOf(Y,N),
    constraints(M,N,C).

/*****
/*          Defining Scenario ShortHand Predicates          */
*****/

/* X and Y have the same subject */
commonSubj(X,Y) :-
    subject(E,X),
    subject(E,Y),
    X \== Y.

/* X and Y have the same subject E */
commonSubj(X,Y,E) :-
    subject(E,X),
    subject(E,Y),
    X \== Y.

```

```
/* X and Y have the same target */
commonTar(X,Y) :-
    target(E,X),
    target(E,Y),
    X \== Y.
```

```
/* X and Y have the same target E */
commonTar(X,Y,E) :-
    target(E,X),
    target(E,Y),
    X \== Y.
```

```
/* X and Y have the same trustor */
commonTrustor(X,Y) :-
    trustor(E,X),
    trustor(E,Y),
    X \== Y.
```

```
/* X and Y have the same trustor E */
commonTrustor(X,Y,E) :-
    trustor(E,X),
    trustor(E,Y),
    X \== Y.
```

```
/* X and Y have the same trustee */
commonTrustee(X,Y) :-
    trustee(E,X),
    trustee(E,Y),
    X \== Y.
```

```
/* X and Y have the same trustee E */
commonTrustee(X,Y,E) :-
    trustee(E,X),
    trustee(E,Y),
    X \== Y.
```

```
/* X and Y have the same recommendor */
commonRecommendor(X,Y) :-
    recommendor(E,X),
    recommendor(E,Y),
    X \== Y.
```

```
/* X and Y have the same recommendor E */
commonRecommendor(X,Y,E) :-
    recommendor(E,X),
    recommendor(E,Y),
    X \== Y.
```

```
/* X and Y have the same recommendee */
commonRecommendee(X,Y) :-
    recommendee(E,X),
    recommendee(E,Y),
    X \== Y.
```

```
/* X and Y have the same recommendee E */
commonRecommendee(X,Y,E) :-
    recommendee(E,X),
    recommendee(E,Y),
    X \== Y.
```

```

/* X and Y have the same level */
equalLevel(X,Y) :-
    level(L,X),
    level(L,Y),
    X \== Y.

/* X and Y have the same level L */
equalLevel(X,Y,L) :-
    level(L,X),
    level(L,Y),
    X \== Y.

/* A and B have a common actionset */
commonAS(A, B) :-
    actionset(APR, A),
    actionset(ANR, B),
    intersect(APR, ANR, ACTR),
    not_empty(ACTR).

/* A and B have a common actionset (using part of rules) */
commonAS(A, B) :-
    commonAS(A, B, X),
    not_empty(X).

commonAS(A, B, Result) :-
    actionset(AA, A),
    actionset(AB, B),
    find_commonAS(AA, AB, R), concat([], R, Result).

/*****
/*          Identifying Cycles          */
*****/

cycles(Result) :-
    findall([A,B], cycle(A,B), Temp),
    remove(Temp, Result).

/* A cycle exists between rules PA and PB. */
cycle(PA, PB) :-
    exists(PA, CA), exists(PB, CB), PA \== PB,
    sys_con(CA, CAA), sys_con(CB, CBA),
    entails(CBA, PA),
    circle(CAA, PB).

/* There is a path from the set of constraints CoA to rule PB. */
circle(CoA, PB) :-
    first(CoA, Element, Rest),
    (
        ( Element = trust(Subject, Target, Actions, Level, Policy);
          Element = recommend(Subject, Target, Actions, Level, Policy)
        ),
        clause(Element, Con),
        not_empty_constraint(Con),
        convert_to_list(Con, Constr),
        ( entails(Constr, PB) ;
          (sys_con(Constr, CoH), not_empty(CoH), circle(CoH, PB)), !
        );
        ( not_empty(Rest), circle(Rest, PB) )
    ).

/* There exists a trust rule Policy that has a set of constraints called Constraints. */
exists(Policy, Constraints) :-

```

```

        clause( trust(_Sub, _Tar, _Act, _Lev, Policy), Con),
        convert_to_list(Con, Constraints).

/* There exists a recommend rule Policy that has a set of constraints called Constraints. */
exists(Policy, Constraints) :-
    clause( recommend(_Sub, _Tar, _Act, _Lev, Policy), Con),
    convert_to_list(Con, Constraints).

/* CoA is the set of trust and recommend rules present in CA. */
sys_con(CA, CoA) :-
    findall(X, (member(X, CA),(functor(X, trust, 5);
        functor(X, recommend, 5))), CoA).

/* The constraint set CBA contains the trust rule PA. */
entails(CBA, PA) :-
    clause(trust(Subject, Target, Actions, Level, PA), _),
    member(trust(Subject, Target, Actions, Level, PA),CBA).

/* The constraint set CBA contains the recommend rule PA. */
entails(CBA, PA) :-
    clause(recommend(Subject, Target, Actions, Level, PA), _),
    member(recommend(Subject, Target, Actions, Level, PA),CBA).

/*****
/*          Removing Cycles          */
*****/

/* Make the specification acyclic. */
make_acyclic :-
    cycles(Result),
    process_cycles(Result).

/* Processes the cycles in List. */
process_cycles(List) :-
    get_first_pair(List, Policy1, Policy2, Rest),
    correct_cycle(Policy1, Policy2),
    process_cycles(Rest).

process_cycles([]). /* base case */

/* Corrects a cycle between two trust rules Policy1 and Policy2. */
correct_cycle(Policy1, Policy2) :-
    clause(trust(Subject1,Target1,Actions1,Level1,Policy1), _),
    clause(trust(Subject2,Target2,Actions2,Level2,Policy2), Constr2),
    convert_to_list(Constr2, ConstrSet2),
    delete(trust(Subject1,Target1,Actions1,Level1,Policy1), ConstrSet2, ConstrSet3),
    insert(rule(Policy1), ConstrSet3, ConstrSet4),
    to_constraints(ConstrSet4, Con4),
    retractall(trust(Subject2,Target2,Actions2,Level2,Policy2)),
    assertz((:-trust(Subject2,Target2,Actions2,Level2,Policy2),Con4))).

/* Corrects a cycle between two recommend rules Policy1 and Policy2. */
correct_cycle(Policy1, Policy2) :-
    clause(recommend(Rr1,Re1,Actions1,Level1,Policy1), _),
    clause(recommend(Rr2,Re2,Actions2,Level2,Policy2), Constr2),
    convert_to_list(Constr2, ConstrSet2),
    delete(recommend(Rr1,Re1,Actions1,Level1,Policy1), ConstrSet2, ConstrSet3),
    insert(rule(Policy1), ConstrSet3, ConstrSet4),
    to_constraints(ConstrSet4, Con4),
    retractall(recommend(Rr2,Re2,Actions2,Level2,Policy2)),

```

```

    assertz((:- (recommend(Rr2, Re2, Actions2, Level2, Policy2), Con4))).

/* Corrects a cycle between two trust rule Policy1 and recommend rule Policy2.*/
correct_cycle(Policy1, Policy2) :-
    clause(trust(Subject1, Target1, Actions1, Level1, Policy1), _),
    clause(recommend(Rr2, Re2, Actions2, Level2, Policy2), Constr2),
    convert_to_list(Constr2, ConstrSet2),
    delete(trust(Subject1, Target1, Actions1, Level1, Policy1), ConstrSet2, ConstrSet3),
    insert(rule(Policy1), ConstrSet3, ConstrSet4),
    to_constraints(ConstrSet4, Con4),
    retractall(recommend(Rr2, Re2, Actions2, Level2, Policy2)),
    assertz((:- (recommend(Rr2, Re2, Actions2, Level2, Policy2), Con4))).

/*****
/*      Constraint Satisfaction      */
*****/

abduce(Head, Things) :-
    solve(Head, [], Things).

solve(Head, Interm, Things) :-
    findall(Body, clause(Head, Body), Ans),
    process_list(Ans, Interm, Thing),
    meets(Thing, [], Things).

process_list(Ans, Interm, Things) :-
    first(Ans, Top, Rest),
    processtop(Top, Interm, NewInterm),
    process_list(Rest, NewInterm, Things).

process_list([], Things, Things).

processtop(Top, Interm, NewInterm) :-
    \+ functor(Top, '!', _),
    process_predicate(Top, Interm, NewInterm).

processtop(Top, Interm, NewInterm) :-
    functor(Top, '!', _),
    convert_to_list(Top, T),
    process_list(T, Interm, NewInterm).

process_predicate(Top, Interm, NewInterm) :-
    (functor(Top, trust, 5); functor(Top, recommend, 5)),
    solve(Top, Interm, NewInterm).

process_predicate(Top, Interm, NewInterm) :-
    \+ (functor(Top, trust, 5); functor(Top, recommend, 5)),
    complement(Top, Neg),
    \+ contains(Interm, Neg),
    insert(Top, Interm, NewInterm).

meets([], F, F).

meets(L, I, F) :-
    first(L, H, T),
    asserta(exist(all)),
    on_exception(E, H, check(E, H)),
    (
    ( exist(all), concat(I, [], NI), retractall(exist(all)), meets(T, NI, F) );
    ( exist(Top), (Top \== all), retractall(exist(_)), insert(Top, I, NI) , meets(T, NI, F) )
    ).

```

```

check(E, Top) :-
    E = existence_error(.,.,.,.),
    retractall(exist(all)),
    asserta(exist(Top)).

getExp(Tr, Te, As, Value) :-
    findall(Ev, (experience(Tr,Te,As,Ev)), Eval),
    sum(Eval, Val),
    count(Eval, D), D > 0,
    Value is Val // D .

getOptimistExp(S, T, As, Value) :-
    findall(Ev, (experience(Tr,Te,As,Ev)), Eval),
    sort(Ev, SortedEv),
    append(., [Value], SortedEv).

getPessimExp(S, T, As, Value) :-
    findall(Ev, (experience(Tr,Te,As,Ev)), Eval),
    sort(Ev, [Value|_]).

/*****/
/*                                     */
/*      Auxiliary and Low Level Predicates      */
/*                                     */
/*****/

/* The set of constraints C is empty of has been satisfied */
empty_or_met(C) :-
    ( empty_constraint(C);
      ( not_empty_constraint(C),
        convert_to_list(C,CList),
        meets(CList,[],Met),
        empty(Met)
      )
    ).

/* Look for cycles, first ones relating to policy P */
look_for_cycle(P) :-
    ( \+ cycle(P,_),
      cycles(CycList),
      ( empty(CycList);
        (not_empty(CycList), write('Cycles Detected : '),
          write(CycList),write('\n'))
        )
      );
    ( cycle(P,Y),
      write('There is a cycle between rules '),
      write(P),write(' and '), write(Y),write('\n')
    ).

/* Auxiliary Predicate used to determine a trust relationship
with E as the trustee, A is contained in the actionset and
the level is L */
p_trustorsWithAS(E, A, L, T) :- /* Source version */
    p_trustee(E, P),
    p_level(L, p),
    p_actionset(Action, P),
    subset(A, Action),
    p_trustor(T,P),
    T \== E.

```

```

trustorsWithAS(E, A, L, T) :- /* Scenario Version */
    trustee(E, P),
    level(L, p),
    actionset(Action, P),
    subset(A, Action),
    trustor(T,P),
    T \== E.

/* Includes the organizational chart info in finding common
Action Sets */
find_commonAS([], _, []).

find_commonAS( [X|Y], [A|B], [Common|R] ) :-
    functor(X, Name, Num), functor(A, Name, Num),
    same_first_argument(X, A, Common),
    Common \== Name,
    N is Num-1, check_other_arguments(X, A, 2, N),
    find_commonAS(Y, B, R).

find_commonAS( [X|Y], [A|B], R ) :-
    functor(X, Name, Num), functor(A, Name, Num),
    same_first_argument(X, A, Common),
    Common == Name,
    find_commonAS(Y, B, R).

find_commonAS( [X|Y], [A|B], R ) :-
    functor(X, Name1, Num), functor(A, Name2, Num),
    Name1 \== Name2,
    (find_commonAS([X|Y], B, R);find_commonAS(X, [A|B], R)).

/* Ensuring that the first argument of an actionset match */
same_first_argument(F, S, F) :-
    arg(1, F, Ent1), arg(1, S, Ent2),
    partOf(Ent1, Ent2).

same_first_argument(F, S, S) :-
    arg(1, F, Ent1), arg(1, S, Ent2),
    partOf(Ent2, Ent1).

same_first_argument(F, S, X) :-
    arg(1, F, Ent1), arg(1, S, Ent2),
    \+ partOf(Ent1, Ent2), \+ partOf(Ent2,Ent2),
    functor(F, X, _).

/* Checking that the other arguments in the actionset match */
check_other_arguments(_, _, _, 0).

check_other_arguments(X, Y, Start, Num) :-
    arg(Start, X, XStart), arg(Start, Y, YStart),
    (
        (var(XStart));(var(YStart));
        (nonvar(XStart),nonvar(YStart), XStart==YStart)
    ),
    NewS is Start+1, NewNum is Num-1,
    check_other_arguments(X, Y, NewS, NewNum).

/* C is an empty constraint */
empty_constraint(C) :- =(C, true).

/* C is a non-empty constraint */

```



```
not_empty_constraint(C) :- \+ =(C, true).
```

```
/* B is the complement of A. */
complement(A, B) :- B = not(A).
complement(A, B) :- B = (\+(A)).
```

```
/* X is a member of the list, specified by the second parameter. */
member(X, [X|_]).
member(X, [_|Y]) :- member(X, Y).
```

```
/* X is not a member of the list, specified by the second parameter. */
non_member(X, [Y|T]) :- X \== Y, non_member(X, T).
non_member(_, []).
```

```
/* X is the head of the list specified by the first parameter (a list) and Y is the list
   minus the first element X. */
first([X|Y], X, Y).
```

```
/* The first parameter is a list that is a subset of the second parameter
   (which is also a list). */
subset([A|X], Y) :- member(A, Y), subset(X, Y).
subset([], _).
```

```
/* N is the number of elements in the list that specified by the first parameter. */
count([_|Y], N) :- count(Y, X), N is 1 + X.
count([], 0).
```

```
/* V is the sum of the integers in the list, the first parameter */
sum([], 0).
sum([X|Y], V) :-
    sum(Y, Z), V is Z + X.
```

```
/* The third parameter is the set union of the sets
   specified by the first and second parameters
   (which are also sets that are represented by lists). */
union([X|Y], Z, W) :- member(X, Z), union(Y, Z, W).
union([X|Y], Z, [X|W]) :- \+ member(X, Z), union(Y, Z, W).
union([], Z, Z).
```

```
/* Finding the intersection of two lists */
intersect([], _, []).
intersect([X|R], Y, [X|Z]) :- member(X, Y), !, intersect(R, Y, Z).
intersect([X|R], Y, Z) :- non_member(X, Y), !, intersect(R, Y, Z).
```

```
/* Constructing the permutations of a set */
permutation([], []).
permutation(L, [F|P]) :-
    select(F, L, R),
    permutation(R, P).
```

```
select(E, [E|T], T).
select(E, [H|T1], [H|T2]) :-
    select(E, T1, T2).
```

```
/* M is the set L minus the duplicate elements. */
remove_dup(L, M) :- rdup(L, [], M).
rdup([], A, A).
rdup([H|T], A, L) :- member(H, A), rdup(T, A, L).
rdup([H|T], A, L) :- rdup(T, [H|A], L).
```

```
/* The parameter passed to it is an empty list. */
empty([]).
```

```

/* A is not an empty list. */
not_empty(A) :- \+ empty(A).

/* Inserts the element E into the list L and produces the new list (the third parameter).*/
insert(E, L, [E|L]).

/* Element E is contained the list specified by the first parameter. */
contains([E|_], E).
contains([H|T], E) :- H \== E, contains(T,E).

/* Deletes element E from the list specified by the second parameter
and produces the new list (the third parameter). */
delete(E, [E|T], T).
delete(E, [H|T], [H|N]) :- delete(E, T, N).

/* The second and third parameters are the first pair of elements
from the list of pairs specified by the first parameter and
T is the remainder of the list. */
get_first_pair([[X, Y]| T], X, Y, T ).

/* Converts a list (the first parameter)
to a set of constraints (the second parameter). */
to_constraints([H|T], (H, Z)) :- to_constraints(T, Z).
to_constraints([X], (X)).

/* Converts a set of constraints (the first parameter)
into a list (the second parameter).*/
convert_to_list((X,Y),[X|Z]) :- !, convert_to_list(Y, Z).
convert_to_list(X,[X]).

/* Adds two lists together (the first and second paramters)
to produce a new concatenated list (the third parameter) */
concat([], L, L).
concat([X|L1], L2, [X|L3]) :- concat(L1, L2, L3).

/* Removes duplicates from a list of lists, L, and returns
the List R. Element order is the sublist is ignored.
Thus, if L is [[a,b],[b,a]], then R will be [[a,b]],
because [a,b] is considered equal to [b,a] */
remove(L, R) :-
    removeAux(L, [], R).

/* General Case - removing duplicates from list of lists */
removeAux(Start, Interm, Result) :-
    first(Start, Head, Tail),
    (
        ( \+ mem(Head, Interm)),
        insert(Head, Interm, NewInterm)
    );
    ( mem(Head, Interm),
      concat(Interm,[], NewInterm)
    ),
    removeAux(Tail, NewInterm, Result).

removeAux([],R,R).

mem(ListA, ListB) :-
    first(ListB, Head, _),
    \+ empty(Head),
    findall(A, (permutation(Head,A)), Result),
    member(ListA, Result).

```

```

mem(ListA, ListB) :-
    first(ListB, Head, Tail),
    \+ empty(Head),
    findall(A, (permutation(Head,A)), Result),
    \+ member(ListA, Result),
    mem(ListA, Tail).

mem(_, ListB) :-
    first(ListB, Head, _),
    empty(Head), fail.

/* Inclusion of Organisational Chart Info */
partOf(X,Y) :-
    on_exception(existence_error(_,_,_,_), isPartOf(X,Y), fail).

partOf(X,Y) :-
    on_exception(existence_error(_,_,_,_), isPartOf(T,Y), fail),
    partOf(X,T).

/* Predicates for translated expressions - Numbers */
gt(LHS,RHS) :-
    eq(LHS, A), RHS > A.

gt(LHS,RHS) :-
    eq(LHS, A), RHS @> A.

lt(LHS,RHS) :-
    eq(LHS, A), RHS < A.

lt(LHS,RHS) :-
    eq(LHS, A), RHS @< A.

gteq(LHS, RHS) :-
    eq(LHS, A), RHS >= A.

gteq(LHS, RHS) :-
    eq(LHS, A), RHS @>= A.

lteq(LHS, RHS) :-
    eq(LHS, A), RHS =< A.

lteq(LHS, RHS) :-
    eq(LHS, A), RHS @=< A.

neq(LHS, RHS) :-
    eq(LHS, A), RHS \= A.

neq(LHS, RHS) :-
    eq(LHS, A), RHS \== A.

/* Error Handling */
error_wrapper(Statement, E) :-
    on_exception(E, Statement, error_message(E, Statement)).

error_message(_,_).

```

Appendix E Template of Conflicts and Ambiguities

This appendix lists the definitions of the SULTAN Analysis Template.

```

/*****
/* Author: Tyrone W.A. Grandison */
/* Last Date Modified: Aug 10, 2002 */
/* Purpose: This is the SULTAN template, which contains a list of generic conflicts */
/* and ambiguities. */
*****/

/* Trust conflict - occurs when there is a trust rule and a distrust rule relating to the */
/* same trustor, trustee and actionset. */

/* Source Trust Conflict */
p_trust_conflict(Result) :-
    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_commonAS(T, D)
                ), Result).

/* Scenario Trust Conflict */
trust_conflict(Result) :-
    query( [T,D], ( pos_trust(T), neg_trust(D), trustor(Tr,T), trustor(Tr, D),
                  trustee(Te,T), trustee(Te, D), commonAS(T, D)
                ), Result).

/* Recommendation Conflict - occurs when there is a positive recommendation */
/* and a negative recommendation relating to same recommendor, recommendee */
/* and set of recommended actions. */

/* Source Recommendation Conflict */
p_rec_conflict(Result) :-
    query( [PR, NR], ( p_pos_rec(PR), p_neg_rec(NR), p_recommendor(Rr,PR),
                    p_recommendor(Rr, NR), p_recommendee(Re,PR),
                    p_recommendee(Re, NR), p_commonAS(PR, NR)
                  ), Result).

/* Scenario Recommendation Conflict */
rec_conflict(Result) :-
    query( [PR, NR], ( pos_rec(PR), neg_rec(NR), recommendor(Rr,PR),
                    recommendor(Rr, NR), recommendee(Re,PR),
                    recommendee(Re, NR), commonAS(PR, NR)
                  ), Result).

/* Trust-Recommend Conflict - occurs when a trust policy and a recommendation */
/* related to the same subject, target and actions. */

/* Source Positive Trust - Negative Recommendation Conflict*/
p_tr_conflict(Result) :-
    query( [T, NR], ( p_pos_trust(T), p_neg_rec(NR), p_subject(Rr,T),
                    p_subject(Rr, NR), p_target(Re,T), p_target(Re, NR),
                    p_commonAS(T, NR)
                  ), Result).

/* Scenario Positive Trust - Negative Recommendation Conflict*/

```

```

tr_conflict(Result) :-
  query( [T, NR], ( pos_trust(T), neg_rec(NR), subject(Rr,T), subject(Rr, NR),
                    target(Re,T), target(Re, NR), commonAS(T, NR)
                  ), Result).

/* Source Positive Recommendation - Distrust Conflict */
p_rd_conflict(Result) :-
  query( [PR, D], ( p_pos_rec(PR), p_neg_trust(D), p_subject(Rr,PR),
                    p_subject(Rr, D), p_target(Re,PR), p_target(Re, D),
                    p_commonAS(PR, D)
                  ), Result).

/* Scenario Positive Recommendation - Distrust Conflict */
rd_conflict(Result) :-
  query( [PR, D], ( pos_rec(PR), neg_trust(D), subject(Rr,PR), subject(Rr, D),
                    target(Re,PR), target(Re, D), commonAS(PR, D)
                  ), Result).

/* Trust Ambiguity - occurs when two trust rules relate to the same subject, target, */
/* actionset with the levels having the same polarity, but different values. */

/* Source Positive Trust Ambiguity */
p_tt_ambiguity(Result) :-
  query( [T1,T2], ( p_pos_trust(T1), p_pos_trust(T2), T1 \== T2, p_trustor(Tr,T1),
                    p_trustor(Tr, T2), p_trustee(Te,T1), p_trustee(Te, T2),
                    p_commonAS(T1, T2), p_level(L1, T1), p_level(L2, T2),
                    L1 \== L2
                  ), Result).

/* Scenario Positive Trust Ambiguity */
tt_ambiguity(Result) :-
  query( [T1,T2], ( pos_trust(T1), pos_trust(T2), T1 \== T2, trustor(Tr,T1),
                    trustor(Tr, T2), trustee(Te,T1), trustee(Te, T2),
                    commonAS(T1, T2), level(L1, T1), level(L2, T2), L1 \== L2
                  ), Result).

/* Source Negative Trust Ambiguity */
p_dd_ambiguity(Result) :-
  query( [T1,T2], ( p_neg_trust(T1), p_neg_trust(T2), T1 \== T2,
                    p_trustor(Tr,T1), p_trustor(Tr, T2), p_trustee(Te,T1),
                    p_trustee(Te, T2), p_commonAS(T1, T2), p_level(L1, T1),
                    p_level(L2, T2), L1 \== L2
                  ), Result).

/* Scenario Negative Trust Ambiguity */
dd_ambiguity(Result) :-
  query( [T1,T2], ( neg_trust(T1), neg_trust(T2), T1 \== T2, trustor(Tr,T1),
                    trustor(Tr, T2), trustee(Te,T1), trustee(Te, T2),
                    commonAS(T1, T2), level(L1, T1), level(L2, T2), L1 \== L2
                  ), Result).

/* Recommend Ambiguity - occurs when two recommendations related to the same */
/* subject, target, actionset with the levels having the same polarity, but different values. */

/* Source Positive Recommend Conflict */
p_rr_ambiguity(Result) :-
  query( [R1,R2], ( p_pos_rec(R1), p_pos_rec(R2), R1 \== R2, p_subject(Rr,R1),
                    p_subject(Rr, R2), p_target(Re,R1), p_target(Re, R2),
                    p_commonAS(R1, R2), p_level(L1, R1), p_level(L2, R2), L1 \== L2
                  ), Result).

```

/ Scenario Positive Recommend Conflict */*

```
rr_ambiguity(Result) :-
  query( [R1,R2], ( pos_rec(R1), pos_rec(R2), R1 \== R2,
                  subject(Tr,R1), subject(Tr, R2), target(Te,R1),
                  target(Te, R2), commonAS(R1, R2), level(L1, R1),
                  level(L2, R2), L1 \== L2
                ), Result).
```

/ Source Negative Recommend Conflict */*

```
p_nrrr_ambiguity(Result) :-
  query( [R1,R2], ( p_neg_rec(R1), p_neg_rec(R2), R1 \== R2,
                  p_subject(Rr,R1), p_subject(Rr, R2), p_target(Re,R1),
                  p_target(Re, R2), p_commonAS(R1, R2),
                  p_level(L1, R1), p_level(L2, R2), L1 \== L2
                ), Result).
```

/ Scenario Negative Recommend Conflict */*

```
nrrr_ambiguity(Result) :-
  query( [R1,R2], ( neg_rec(R1), neg_rec(R2), R1 \== R2,
                  subject(Tr,R1), subject(Tr, R2), target(Te,R1),
                  target(Te, R2), commonAS(R1, R2), level(L1, R1),
                  level(L2, R2), L1 \== L2
                ), Result).
```

/ Conflict of Interest - occurs when an entity is trusted by two (other) competing entities */*
/ with respect to the same actionset. */*

/ Source Conflict of Interest */*

```
p_conflict_of_interest(Result) :-
  query( [T1,T2], ( p_pos_trust(T1), p_pos_trust(T2), T1 \== T2,
                  p_trustee(Te,T1), p_trustee(Te, T2), p_commonAS(T1,T2)
                ), Result).
```

/ Scenario Conflict of Interest */*

```
conflict_of_interest(Result) :-
  query( [T1,T2], ( pos_trust(T1), pos_trust(T2), T1 \== T2,
                  trustee(Te,T1), trustee(Te, T2), commonAS(T1,T2)
                ), Result).
```

/ Source Conflict of Interest with Specific Action Set */*

```
p_conflict_of_interest(ActionSet, Result) :-
  query( [T1,T2], ( p_pos_trust(T1), p_pos_trust(T2), T1 \== T2,
                  p_trustee(Te,T1), p_trustee(Te, T2),
                  p_commonAS(T1, T2, ActionSet)
                ), Result).
```

/ Scenario Conflict of Interest with Specific Action Set */*

```
conflict_of_interest(ActionSet, Result) :-
  query( [T1,T2], ( pos_trust(T1), pos_trust(T2), T1 \== T2,
                  trustee(Te,T1), trustee(Te, T2),
                  commonAS(T1, T2, ActionSet)
                ), Result).
```

/ Separation of Duties Conflict - occurs when one entity is trusted to perform two or */*
/ more actions that conflict (conflict here refers to the adherence to organizational policy). */*

/ Source Separation of Duties */*

```
p_separation_of_duties(Entity, ActionSet1, ActionSet2, Result) :-
  query( [T1,T2], ( p_pos_trust(T1), p_pos_trust(T2), T1 \== T2,
                  p_trustee(Entity,T1), p_trustee(Entity, T2),
                  p_actions(ActionSet1, T1), p_actions(ActionSet2, T2)
                ), Result).
```

```
/* Scenario Separation of Duties */
separation_of_duties(Entity, ActionSet1, ActionSet2, Result) :-
    query( [T1,T2], ( pos_trust(T1), pos_trust(T2), T1 \== T2,
                    trustee(Entity,T1), trustee(Entity, T2),
                    actions(ActionSet1, T1), actions(ActionSet2, T2)
                    ), Result).
```

Appendix F SULTAN Specifications Modelling Contexts

The trust classification scheme (described in Chapter 2) is a useful and convenient taxonomy for the contexts in which trust is used. In this Appendix, we discuss how SULTAN may be used to model these contexts. It must be stated that there is no standard way to model this classification scheme and there may be examples that fall into one or more of the categories. This implies that some examples would require composition of the models highlighted below.

Access to Trustor Resources

In this context, the trustor trusts the trustee to access a resource or service that the trustor owns or controls.

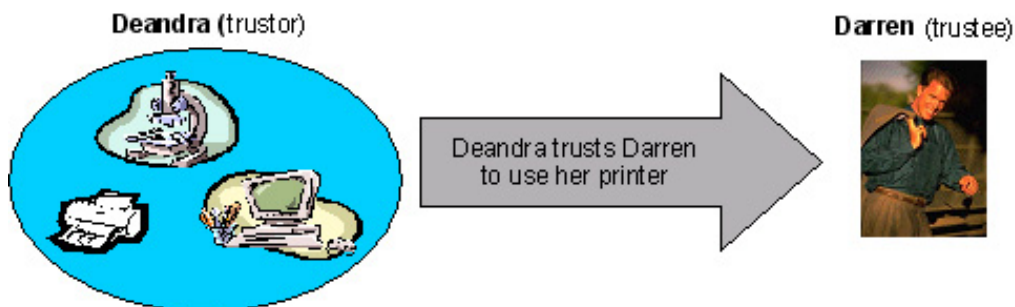


Figure F1: Access to Trustor Resources Trust

From Figure F1, it can be seen that Deandra (highly) trusts Darren to access her printer (`use_printer()`). In the SULTAN specification language, this can be written as:

```
ExA1: trust(Deandra, Darren, use_printer(Deandra), HighTrust );
```

Figure F1 highlights the fact that when the action is performed on the trustor, we can assume that a resource access trust is present.

Provision of Service by the Trustee

By definition, the trustor trusts that the trustee will provide a service to him. This service does not involve access to the trustor's resources. This may not be true of many services, which may require access to resources owned by the trustor.

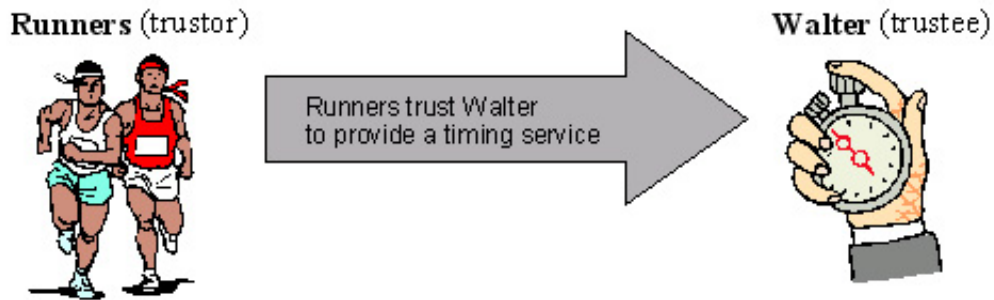


Figure F2: Provision of Service by the Trustee Trust

The example in Figure F2 can be expressed in SULTAN as:

ExA2: **trust**(Runners, Walter, time(Walter, Runners, Times), MediumTrust);

The first parameter of the action is the trustee and this signifies that the action is being performed on the trustee. This may be viewed as the trustee providing a service to the trustor. Another example is an Internet user (we will refer to her as Ann) who trusts a web-based service (this entity will be referred to as NewsServer) to deliver the New York Times. This fact may be represented as:

ExA3: **trust**(Ann, NewsServer, send(NewsServer, NYTimes, Ann), HighTrust);

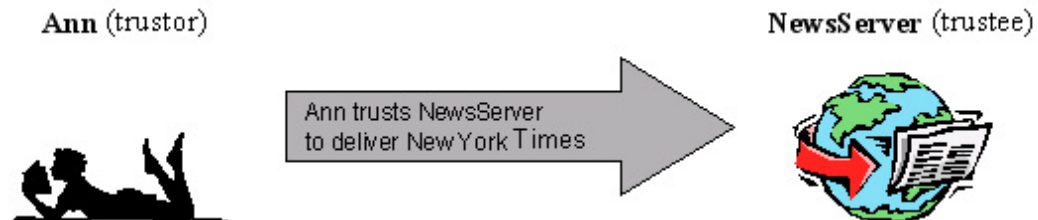


Figure F3: Another example of Provision of Service by the Trustee Trust

In reality, service providers often require access to the trustor's resources. Thus, there may be a few examples that do not follow this general trend. It should also be noted there are some scenarios that require both a resource of trustor resources trust and a provision of service by the trustee trust. For example, Ann may undoubtedly have to trust the NewsServer to access a particular file or set of files where her New York Times will be stored.

Certification

Trust used in the context of certification is the belief in the trustworthiness of an entity based on a third party certifying this entity (for a particular purpose). This third party will be referred to as a certificate authority. To represent certification, the following scenarios must be modelled:

1. An entity, $_X$, trusts any entity $_Y$ to perform $_Actions$ if the certificate authority ($_CA$) trusts $_Y$ to perform $_Actions$.

Cert1: **trust** ($_X, _Y, _Actions, _Arb$)
 \leftarrow **trust** ($_CA, _Y, _Actions, _Arb$) & $_Arb > 0$;

This states that $_X$ trusts $_Y$ at trust level $_Arb$ to perform $_Actions$ if $_CA$ trusts $_Y$ at trust level $_Arb$ to do $_Actions$ and if $_Arb$ is positive. Note that the trust levels need not be related. They have been arbitrarily assumed equal for this example.

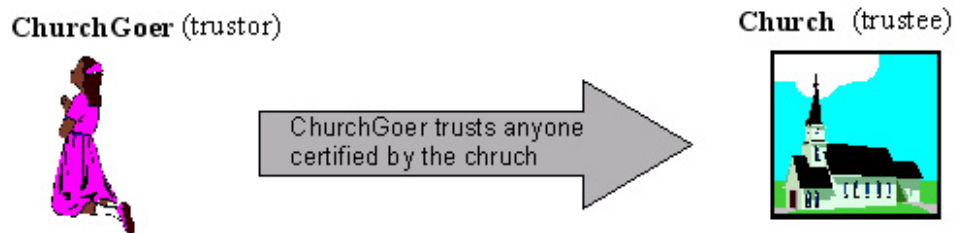


Figure F4: Certification Trust (Phase One)

Figure F4 shows an example of this scenario. ChurchGoer trusts anyone certified by Church. Expressed in the SULTAN notation, this is:

C1: **trust** (ChurchGoer, $_X, _Actions, HighTrust$)
 \leftarrow **trust** (Church, $_X, _Actions, _Arb$), $_Arb > 0$;

2. A certificate authority ($_CA$) certifying an entity, $_Y$, to perform a particular action

Cert2: **trust** ($_CA, _Y, _Actions, _Arb$) \leftarrow $_Arb > 0$;

This states that $_CA$ trusts $_Y$ at trust level $_Arb$ to perform $_Actions$.

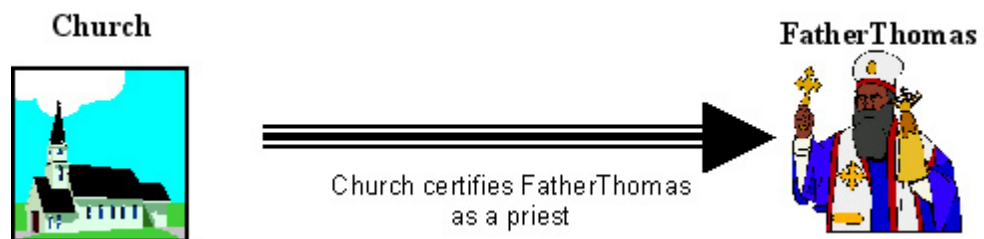


Figure F5: Certification Trust (Phase Two)

The above scenario can be specified as:

C2: **trust** (Church, FatherThomas, $_Actions, HighTrust$);

Delegation

Delegation is the process in which a trustor trusts the trustee to make decisions on its behalf, with respect to a resource or service the trustor owns or controls. In order to model delegation, the following be represented:

1. An entity, $_X$, delegates his decisions with respect to actions, $_DelActions$, to another entity, $_Z$.

```
Del1: trust ( $\_X$ ,  $\_Z$ ,  $\_DelActions$ , MediumTrust)
      ← trust+( $\_X$ ,  $\_Y$ ,  $\_DelActions$ ) &
        trust+( $\_Y$ ,  $\_Z$ ,  $\_DelActions$ );
```

This is a situation where $\exists a, b, c : W \bullet aTb \wedge bTc \Rightarrow aTc$ (the constraint property of the transitivity rule is true).

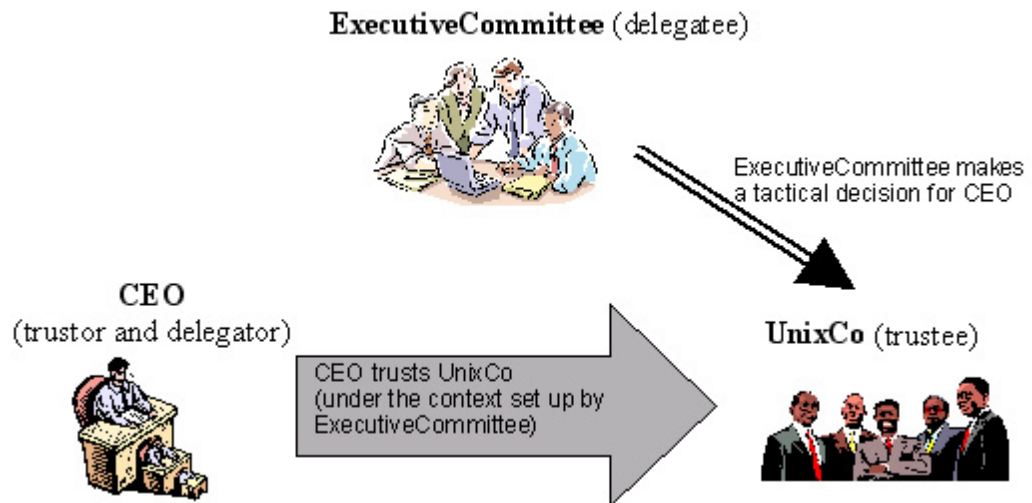


Figure F6: Delegation Trust

The above example is expressed as follows:

```
D1: trust (CEO, UnixCo,  $\_DelActions$ , MediumTrust)
      ← trust+(CEO, ExecutiveCommittee,  $\_DelActions$ ,) &
        trust+ (ExecutiveCommittee, UnixCo,  $\_DelActions$ );
```

For some scenarios, it may be a requirement to have the levels in the trust constraints explicitly specified as positive.

Infrastructure Trust

The trustor trusts the infrastructure that it is based upon. This is often referred to as ‘implicit trust’ (trust in one’s self). For a distributed environment, it cannot be assumed that all of a system’s components are automatically initially trusted. Such an assumption can lead to attackers taking advantage of security holes in one’s platform core. For example, using a backdoor in the operating system, even though the higher-level software has been secured.

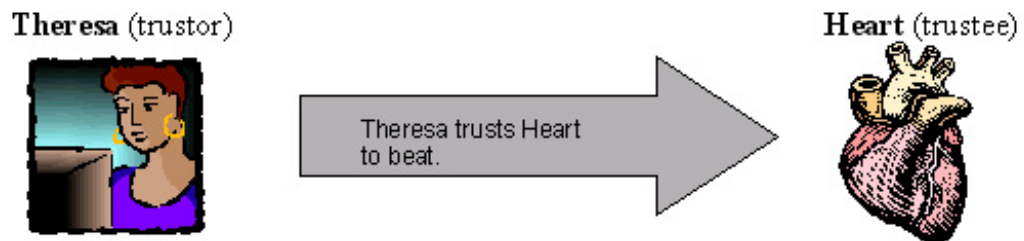


Figure F7: Infrastructure Trust

In SULTAN, the above example is stated as:

IT1: **trust** (Theresa, Heart, beat(Heart), HighTrust);

The modelling of *Infrastructure* Trust is similar to modelling *Provision of Service by the Trustee* trust. The only difference lies in the fact that the trustee is assumed to be a sub-domain (or in the domain/sub-domains) of the trustor (as illustrated in Figure F7). Another example is that of an ordinary computer, which will be called MyComputer. If MyComputer trusts the Windows Operating System (we will refer to this as Windows) to provide a time service, then if we know that Windows is a part of MyComputer then this represents Infrastructure Trust. We write this specification as:

ExG6a: **trust**(MyComputer, Windows, time(Windows), HighTrust);