

Electronic Thesis and Dissertation Repository

2-2-2011 12:00 AM

Architecture Supporting Computational Trust Formation

Chern Har Yew
University of Western Ontario

Supervisor
Hanan Lutfiyya
The University of Western Ontario

Graduate Program in Computer Science
A thesis submitted in partial fulfillment of the requirements for the degree in Doctor of
Philosophy
© Chern Har Yew 2011

Follow this and additional works at: <https://ir.lib.uwo.ca/etd>



Part of the [Systems Architecture Commons](#)

Recommended Citation

Yew, Chern Har, "Architecture Supporting Computational Trust Formation" (2011). *Electronic Thesis and Dissertation Repository*. 86.
<https://ir.lib.uwo.ca/etd/86>

This Dissertation/Thesis is brought to you for free and open access by Scholarship@Western. It has been accepted for inclusion in Electronic Thesis and Dissertation Repository by an authorized administrator of Scholarship@Western. For more information, please contact wlsadmin@uwo.ca.

.ARCHITECTURE SUPPORTING COMPUTATIONAL TRUST
FORMATION

(Spine title: Architecture Supporting Computational Trust Formation)

(Thesis format: Monograph)

by

Chern Har Yew

Graduate Program in Computer Science

A thesis submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy

The School of Graduate and Postdoctoral Studies
The University of Western Ontario
London, Ontario, Canada

© Chern Har Yew 2011

CERTIFICATE OF EXAMINATION

Supervisor

Examiners

Dr. Hanan Lutfiyya

Dr. Michael Bauer

Supervisory Committee

Dr. Robert Mercer

Dr.

Dr. Victoria Rubin

Dr.

Dr. Ferhat Khendek

The thesis by

Chern Har Yew

entitled:

**ARCHITECTURE SUPPORTING COMPUTATIONAL TRUST
FORMATION**

is accepted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Date

Chair of the Thesis Examination Board

Abstract

Trust is a concept that has been used in computing to support better decision making. For example, trust can be used in access control. Trust can also be used to support service selection. Although certain elements of trust such as reputation has gained widespread acceptance, a general model of trust has so far not seen widespread usage. This is due to the challenges of implementing a general trust model. In this thesis, a middleware based approach is proposed to address the implementation challenges.

The thesis proposes a general trust model known as computational trust. Computational trust is based on research in social psychology. An individual's computational trust is formed with the support of the proposed computational trust architecture. The architecture consists of a middleware and middleware clients. The middleware can be viewed as a representation of the individual that shares its knowledge with all the middleware clients. Each application uses its own middleware client to form computational trust for its decision making needs. Computational trust formation can be adapted to changing circumstances. The thesis also proposed algorithms for computational trust formation. Experiments, evaluations and scenarios are also presented to demonstrate the feasibility of the middleware based approach to computational trust formation.

Keywords

trust, experience, recommendation, reputation, signal, belief, architecture, middleware

Acknowledgments

I wish to express my sincere gratitude towards my supervisor, Dr. Hanan Lutfiyya. Her guidance, encouragement and friendship are instrumental in my completion of this work. Other professors that have helped me along the way include Dr. Michael Bauer and Dr. Mark Perry. As for fellow distributed systems research group colleagues, special thanks needs to be directed towards Brad Simmons (graduated), Raphael Mafita Bahati (graduated), Gaston Keller and Zainab Aljazzaf.

Table of Contents

CERTIFICATE OF EXAMINATION	ii
Abstract	iii
Acknowledgments.....	iv
Table of Contents	v
List of Tables	xi
List of Figures	xiii
List of Appendices	xviii
Chapter 1	1
1 Introduction	1
1.1 Current Use of Trust in Computing	1
1.2 Computational Trust	3
1.2.1 Foundational Research.....	3
1.2.2 Definition	4
1.2.3 Properties	6
1.2.4 Conceptual Model.....	9
1.2.5 Trust Formation	12
1.3 Application of Computational Trust	13
1.3.1 Movie Selection Scenario	13
1.3.2 Web Service Selection Scenario	14
1.4 Challenges to Computational Trust	17
1.5 Contributions of Thesis.....	18
1.6 Thesis Organization	19
1.7 Summary.....	20
Chapter 2.....	21

2	Trust Primer	21
2.1	Preconditions for Trust	21
2.2	Concepts Related to Trust.....	21
2.2.1	Untrust, Distrust and Mistrust.....	22
2.2.2	Trustworthiness.....	22
2.2.3	Contract.....	23
2.2.4	Confidence	23
2.3	Privacy	24
2.4	Risk	25
2.5	Decision Making.....	26
2.6	Summary	26
	Chapter 3.....	27
3	Literature Review.....	27
3.1	Survey of Trust and Reputation	27
3.1.1	The World Wide Web (WWW).....	27
3.1.2	Peer-to-Peer (P2P) Networks.....	30
3.1.3	Virtual Communities.....	34
3.1.4	Pervasive Computing Environments.....	38
3.2	Middleware Support for Computational Trust Formation.....	39
3.3	Survey of Trust Middleware	40
3.4	Summary	41
	Chapter 4.....	42
4	Computational Trust Architecture.....	42
4.1	Assumptions.....	42
4.1.1	Identity and Type	42

4.1.2	Computational Trust Formation.....	43
4.1.3	Evidence.....	44
4.1.4	Belief.....	45
4.1.5	Emotional Trust Formation.....	47
4.1.6	Factors Influencing Computational Trust Formation	48
4.1.7	Privacy	50
4.1.8	Architecture Deployment.....	50
4.1.9	Summary.....	50
4.2	Overview of Architecture	51
4.3	Information Flow of Architecture.....	52
4.4	Summary.....	53
Chapter 5	54
5	SCOUT.....	54
5.1	SCOUT Design.....	54
5.2	Evidence Gathering Service.....	55
5.2.1	Evidence Gatherer.....	57
5.2.2	Evidence Handler.....	57
5.2.3	Evidence Source Assessor	58
5.2.4	Evidence Gathering Manager	59
5.3	Belief Formation Service	63
5.3.1	Belief Engine	64
5.3.2	Belief Formation Manager.....	65
5.4	Emotional Trust Service	68
5.5	Summary.....	69
Chapter 6	71

6	Trust Calculator.....	71
6.1	Trust Calculator Design	71
6.2	Trust Calculation Plan.....	72
6.3	Trust Calculator	77
6.4	Summary.....	79
	Chapter 7.....	81
7	Algorithms	81
7.1	Belief Formation	81
7.1.1	Experience.....	82
7.1.2	Recommendation	84
7.1.3	Reputation.....	85
7.1.4	Signal	86
7.1.5	Summary.....	86
7.2	Evidence Source Assessment.....	87
7.2.1	Evidence Assessment.....	87
7.2.2	Evidence Source Trust Assessment	89
7.2.3	Summary.....	90
7.3	Summary.....	91
	Chapter 8.....	92
8	Implementation	92
8.1	SCOUT	92
8.2	Trust Calculator	93
8.2.1	Trust Calculation Planner	94
8.2.2	Trust Calculation Engine	95
8.3	Summary.....	96

Chapter 9.....	97
9 Experiments.....	97
9.1 Goals	97
9.2 Assumptions.....	98
9.3 Metrics	99
9.4 Experimental Testbed	99
9.4.1 Overview.....	100
9.4.2 Web Services	100
9.4.3 Evidence Sources	102
9.4.4 Web Service Selection Strategies	103
9.5 Experiments Based on Experience.....	107
9.5.1 Static Web Services	108
9.5.2 Fluctuating Web Services	110
9.5.3 Dynamic Web Services.....	112
9.5.4 Fluctuating-Dynamic Web Services	114
9.6 Experiments Based on Recommendation	116
9.6.1 Similar Recommenders.....	118
9.6.2 Dissimilar Recommenders	122
9.7 Experiments Based on Experience, Recommendation	127
9.8 Conclusions and Discussions.....	131
9.9 Summary	132
Chapter 10.....	134
10 Scenarios	134
10.1 Movie Scenario	134
10.1.1 Overview.....	134

10.1.2 Implementation	138
10.2 Web Service Scenario	138
10.2.1 Overview	138
10.2.2 Implementation	159
10.3 Summary	160
Chapter 11	161
11 Evaluation	161
11.1 Evaluation of SCOUT	161
11.1.1 Evolution	161
11.1.2 Computational Trust Properties	162
11.1.3 Computational Trust Challenges	163
11.2 Evaluation of Trust Calculator	164
11.2.1 Evolution	164
11.2.2 Computational Trust Properties	165
11.2.3 Computational Trust Challenges	165
11.3 Summary	166
Chapter 12	167
12 Conclusions and Future Directions	167
12.1 Conclusions	167
12.2 Future Work	169
References	172
Appendices	186
Curriculum Vitae	189

List of Tables

Table 1: Trust Types and Computational Trust Formation	49
Table 2: Assumptions of Computational Trust Architecture	50
Table 3: Algorithms for Belief, Aggregate Belief and Trust Calculation.....	53
Table 4: Example Aggregate Belief Filters	67
Table 5: Web Services (Experiments)	99
Table 6: Legend of Figure 39.....	107
Table 7: Result Summary (Static Web Services, Average: $window_{exp} = \infty$)	108
Table 8: Result Summary (Static Web Services, Average: $window_{exp} = 5$).....	109
Table 9: Result Summary (Fluctuating Web Services, Average: $window_{exp} = \infty$)	110
Table 10: Result Summary (Fluctuating Web Services: $prob_f = 0.2, duration_f = 2$, Average: $window_{exp} = \infty$)	111
Table 11: Result Summary (Fluctuating Web Services: $prob_f = 0.1, duration_f = 4$, Average: $window_{exp} = \infty$)	111
Table 12: Result Summary (Fluctuating Web Services, Average: $window_{exp} = 5$).....	112
Table 13: Result Summary (Fluctuating Web Services, Weighted Average: $\lambda = 1.443$)	112
Table 14: Result Summary (Dynamic Web Services, Average: $window_{exp} = \infty$).....	113
Table 15: Result Summary (Dynamic Web Services: $prob_d = 0.08$, Average: $window_{exp} = \infty$).....	113
Table 16: Result Summary (Dynamic Web Services, Average: $window_{exp} = 5$)	114

Table 17: Result Summary (Dynamic Web Services, Weighted Average: $\lambda = 1.443$).	114
Table 18: Result Summary (FD Web Services, Average: $window_{exp} = \infty$)	115
Table 19: Result Summary (FD Web Services, Average: $window_{exp} = 5$).....	116
Table 20: Result Summary (FD Web Services, Weighted Average: $\lambda = 1.443$).....	116
Table 21: Similar Recommenders (Experiments).....	118
Table 22: Dissimilar Recommenders (Experiments).....	123
Table 23: Syntax of SCOUT Policies	186

List of Figures

Figure 1: Relationships among Trust Dimensions.....	11
Figure 2: Computational Trust Formation	13
Figure 3: Cognitive Trust Formation (Movie)	14
Figure 4: Cognitive Trust Formation (Web Service).....	15
Figure 5: From Distrust to Trust [86]	22
Figure 6: eBay’s Feedback Forum.....	28
Figure 7: Computational Trust Formation (Architecture)	43
Figure 8: Cognitive Trust Formation from Aggregate Beliefs (Web Service).....	47
Figure 9: Hierarchy of Trustee Type and Identity	48
Figure 10: Computational Trust Architecture.....	52
Figure 11: Information Flow of Architecture	53
Figure 12: Evidence Gathering Service	56
Figure 13: EG-Policies (Identity).....	60
Figure 14: EG-Policy (Properties)	61
Figure 15: EG-Policy (Evidence Source Trust)	61
Figure 16: EG-Policy (Broadcast Strategy)	62
Figure 17: EG-Policy (Evidence Source Trust Strategy).....	62
Figure 18: EA-Policy	63

Figure 19: Belief Formation Service.....	64
Figure 20: BF-Policy.....	66
Figure 21: AF-Policy	67
Figure 22: Emotional Trust Service.....	68
Figure 23: ET-Policies	69
Figure 24: Structure of TcPlan.....	72
Figure 25: TcPlan (Movie).....	73
Figure 26: Tree of TcPlan (Movie).....	73
Figure 27: TcTemplate (Movie).....	74
Figure 28: TcPlan (Web Service)	76
Figure 29: Tree of TcPlan (Web Service).....	77
Figure 30: Trust Calculator.....	77
Figure 31: Trust Calculator's Execution.....	79
Figure 32: Experience Weights.....	83
Figure 33: Evidence Assessments.....	88
Figure 34: SCOUT Implementation.....	92
Figure 35: Factors-TcPlan Mapping (Movie).....	94
Figure 36: Factors-TcPlan Policy (Movie)	95
Figure 37: Experimental Testbed's Algorithm	100
Figure 38: Exploration Ratios.....	105

Figure 39: Exploration Probabilities	106
Figure 40: Mean Experience (Similar Recommenders, Average: $window_{exp} = \infty$)	119
Figure 41: Percentage of Positive Experiences (Similar Recommenders, Average: $window_{exp} = \infty$).....	120
Figure 42: Mean Experience (Similar Recommenders, $exploration = 14$, $window_{exp} =$ 5).....	121
Figure 43: Percentage of Positive Experiences (Similar Recommenders, $exploration =$ 14 , $window_{exp} = 5$)	121
Figure 44: Mean Experience (Similar Recommenders, $exploration = 14$, $\lambda = 1.443$)	122
Figure 45: Percentage of Positive Experiences (Similar Recommenders, $exploration =$ 14 , $\lambda = 1.443$).....	122
Figure 46: Mean Experience (Dissimilar Recommenders, Average: $window_{exp} = \infty$)	124
Figure 47: Percentage of Positive Experiences (Dissimilar Recommenders, Average: $window_{exp} = \infty$).....	124
Figure 48: Mean Experience (Dissimilar Recommenders, $exploration = 14$, $window_{exp} = 5$).....	125
Figure 49: Percentage of Positive Experiences (Dissimilar Recommenders, $exploration = 14$, $window_{exp} = 5$)	126
Figure 50: Mean Experience (Dissimilar Recommenders, $exploration = 14$, $\lambda = 1.443$)	126
Figure 51: Percentage of Positive Experiences (Dissimilar Recommenders, $exploration = 14$, $\lambda = 1.443$).....	127

Figure 52: Mean Experience (Trustor and Similar Recommenders, Average: <i>exploration</i> = 14, <i>window_{exp}</i> = ∞).....	129
Figure 53: Percentage of Positive Experiences (Trustor and Similar Recommenders, Average: <i>exploration</i> = 14, <i>window_{exp}</i> = ∞)	129
Figure 54: Mean Experience (Trustor and Dissimilar Recommenders, Average: <i>exploration</i> = 14, <i>window_{exp}</i> = ∞).....	130
Figure 55: Percentage of Positive Experiences (Trustor and Dissimilar Recommenders, Average: <i>exploration</i> = 14, <i>window_{exp}</i> = ∞)	131
Figure 56: BF-Policy (Movie)	136
Figure 57: EG-Policy (Movie).....	137
Figure 58: Computational Trust Formation (Movie).....	138
Figure 59: Factors-TcPlan Mappings (Web Service Selection)	140
Figure 60: TcPlan (Web Service Selection).....	142
Figure 61: Tree of TcPlan (Web Service Selection).....	143
Figure 62: BF-Policy (Competence Belief)	144
Figure 63: EG-Policy (Qualification Belief).....	145
Figure 64: BF-Policies (Accessibility Belief).....	147
Figure 65: EG-Policies (Availability Belief)	148
Figure 66: Computational Trust Formation (Web Service Selection).....	149
Figure 67: Factors-TcPlan Mappings (Web Service Negotiation)	151
Figure 68: TcPlan (Web Service Negotiation).....	152

Figure 69: Tree of TcPlan (Web Service Negotiation).....	153
Figure 70: BF-Policies (Dependability Belief).....	154
Figure 71: Computational Trust Formation (Web Service Negotiation).....	155
Figure 72: Web Service Management Policies.....	156
Figure 73: AF-Policies (Web Service).....	158
Figure 74: EA-Policy (Web Service).....	159
Figure 75: Web Service Discovery, Selection and Negotiation Web Service.....	160
Figure 76: Web Service Management Web Service.....	160
Figure 77: Trust Coordination Plan.....	165
Figure 78: Syntax of Aggregate Belief Filter.....	188
Figure 79: Syntax of Emotional Trust Filter.....	188

List of Appendices

Appendix A: Syntax of SCOUT Policies.....	186
Appendix B: Syntax of Subscription Filters	188

Chapter 1

1 Introduction

Decision making is about trying to make the best possible selection from a set of available choices. Decisions in real life are often made through satisficing (i.e., trying to make a good enough selection) as opposed to aiming for the optimal solution [119]. This is due to the fact that human beings are cognitive misers [42] that take advantage of mental shortcuts (cognitive heuristics) when faced with complex decisions. Note that the taking of these mental shortcuts is usually not out of laziness but due to the limited capacity of human beings to process uncertainty and incomplete information [42]. As a result, when faced with complex decisions, trust is often used as a mental shortcut for complexity reduction. This is by eliminating choices that are not trusted from consideration. Trust can be thought of as an alternative to rational prediction [75].

This chapter is organized as follow. The chapter starts with a discussion of the current use of trust in computing. This is followed by an introduction to computational trust. Two scenarios are provided as illustration of the use of computational trust in computing. The chapter ends with a discussion of the challenges to calculating computational trust and how this thesis contributes towards addressing these challenges.

1.1 Current Use of Trust in Computing

In computing, trust has its roots in computer security. Authentication and authorization are concepts commonly associated with trust [50]. In this thesis, this type of trust is known as *access trust*. The goal of access trust is to restrict access so as to protect computing resources from harm. One could think of an authorized user as a user that is trusted by the computing resource owner. The more a user is trusted, the more likely it would be afforded more rights.

A common form of access trust is *identity-based trust*. Trust is determined from the identity of the user (e.g. username and password). A weakness of identity-based trust is its assumption that identity is a trust predictor. This may be a valid assumption in

closed environments such as a corporate LAN where all the LAN participants have been vetted (i.e., being hired by human resources) beforehand. This is not a valid assumption in open environments such as the Internet. On the Internet, most of the users are strangers to each other. Knowing the identity of the stranger (if it is even available) is not sufficient in determining how much the stranger can be trusted.

In [16], the authors proposed that trust should be resolved without the use of identity. The paper introduced the term *trust management* as “a unified approach to specifying and interpreting security policies, credentials and relationships that allow direct authorization of security-critical actions” [16]. This view of trust is sometimes known as *rule-based trust* or *policy-based trust*. Several well-known trust management systems include PolicyMaker [16], KeyNote [15] and REFEREE [28]. In a trust management system, rules or policies are used to define the conditions in which rights are to be granted. This is usually based on the presence of digital certificates (credentials). An example is to only grant access to university computing resources if the user has a certificate proving that the user is a student of the university. More recently, this view of trust has been used to provide security for web services through WS-Security [97] and WS-Trust [96].

In [50] and [40], the authors identified a number of weaknesses to trust management. The weaknesses include:

- A trust management system is a binary yes/no system where a user is either trusted or not trusted. Trust in everyday social life however is much more complex. For example, John may trust both Alice and Bob but between the two, John may trust Alice more than Bob. Such trust dynamics are not represented in trust management systems.
- In trust management, certificate issuers are unconditionally trusted. If a university issues a certificate stating that a user is a student, it is automatically assumed that this is indeed the case. This is an unrealistic assumption. There may be a conflict of interest or factors such as money or even carelessness that could result in the publishing of inaccurate certificates.

- Trust is not static and may change over time. For example, good experiences from using a service may increase trust in the service while bad experiences do the opposite. Instead of updating trust immediately based on a trustee's change in behavior, a trust management system needs to wait for certificates to make any trust changes. As a result, the trust management system is completely dependent on the pace at which certificate issues updates and distributes certificates.

To address the weaknesses of trust management, a more general model of trust is needed. In this thesis, this general model of trust is known as *computational trust*.

1.2 Computational Trust

The first part of this section covers the research that forms the foundation of computational trust. Next, a definition for computational trust is proposed. This is followed by discussion on the properties of computational trust and the different dimensions of computational trust. Finally, computational trust formation is explained.

1.2.1 Foundational Research

Many different disciplines have studied the concept of trust. In biology, research has shown that the hormone Oxytocin [70] when administered can increase trust among humans. In [17], the author examined trust as seen in social psychology, philosophy, economics, contract law and market research. The author discovered that different disciplines tend to view trust differently. For example, social psychology tends to focus on trust as the fulfillment of expectations. In philosophy, trust is viewed as non-rational. In economics, trust is viewed as rational. In contract law, the focus is on trust as a complement to legal contracts. Finally in market research, the focus is on the role that trust plays in relationship based marketing.

Of the different disciplines, this thesis has adopted the view of trust in social psychology. Basically, trust can be divided into four areas of research [27]. Individual trust treats trust as a personality trait. Interpersonal trust is concerned with trust as it

relates to a *trustor* and a *trustee*. A trustor refers to a subject¹ that trusts others. A trustee is an entity² that is being trusted. Relational trust believes that trust is an emergent property of a relationship as opposed to a directed behavior. Societal trust is concerned with the role that trust plays in the proper functioning of society. Trust is often credited as the social capital that makes cooperation in a society possible [92].

Of the four research areas, the focus of this thesis is on interpersonal trust. As a real life example, consider the case of a buyer (trustor) trying to buy a car from a used car salesperson (trustee). There is no way for the buyer to know whether or not the salesperson has the buyer's best interest in mind. The salesperson could be trying to sell the buyer a lemon. The uncertainty and lack of complete information makes the car buying decision challenging. Trust can be used to aid in the buyer's decision making. A buyer can choose to only buy cars from a trusted used car salesperson. In a computing context, the example could be the buying of a car from a used car website.

Many real life decisions have computing equivalents. Thus interpersonal trust can be used in computing to aid the trustor in its decision making. In this thesis, computational trust is built on the research in interpersonal trust. This should allow computational trust to be used in not just computer security but also in general decision making such as web service selection.

1.2.2 Definition

Trust is a concept that is easily recognizable but surprisingly difficult to define. Many different definitions of trust have been proposed to address the different facets of trust. For example, the American Heritage Dictionary defines trust as “firm reliance on the integrity, ability, or character of a person or thing” [132]. Grandison and Sloman define trust as “the quantified belief by a trustor with respect to the competence, honesty,

¹ A subject could be a person, a group of people (e.g. organization) or a representative of a person (e.g. software agent).

² An entity could refer to both animate (e.g. person) and inanimate (e.g. movie) object.

security and dependability of a trustee within a specified context” [51]. Note that in both definitions, trust is defined through the listing of the characteristics of a trustee.

A different approach to defining trust is provided by Deutsch in [35]:

- An individual is confronted with an ambiguous path, a path that can lead to an event perceived to be beneficial (Va^+) or to an event perceived to be harmful (Va^-);
- He perceives that the occurrence of Va^+ or Va^- is contingent on the behavior of another person;
- He perceives the strength of Va^- to be greater than the strength of Va^+
- If he chooses to take an ambiguous path with such properties, the individual is said to have made a trusting choice; if he chooses not to take the path, he has made a distrustful choice;

In this definition, there is no listing of characteristics. This is also the case in [44] where Gambetta defines trust as “a particular level of the subjective probability with which an agent assesses that another agent or group of agents will perform a particular action, both before he can monitor such action (or independently of his capacity ever to be able to monitor it) and in a context in which it affects his own action”.

The definition of computational trust in this thesis is based on the definition proposed by Gambetta in [44]. The concept of roles as discussed in [24] is also incorporated into the trust definition. We define computational trust as follows:

Computational trust is a particular level of subjective assessment of whether a trustee will exhibit characteristics consistent with the role of the trustee, both before the trustor can monitor such characteristics (or independent of the trustor’s capacity ever to be able to monitor it) and in a context in which it affects the trustor’s own behavior.

When compared to the definition presented in [44], the term “probability” has been replaced with the term “assessment” to highlight the fact that trust can be calculated in ways other than probability theory. As the term “agent” has a different meaning in social science than in computer science, the more general terms “trustor” and “trustee” are used instead to avoid confusion. Also, a trustee can be an inanimate object. So “taking action” may not always be possible for certain trustees. The term “action” therefore has been replaced with the term “exhibit characteristics”. As for decision making, there are roles in which both the trustor and the trustee must play. For example the decision of buying services from a trustee. In this example, the trustor plays the role of a buyer while the trustee plays the role of a seller. Whether the trustor trusts the trustee or not is dependent on what the trustor believes to be the role of a “good seller”. If a trustee conforms to the role as defined by the trustor, then the trustor can be said to trust the trustee.

1.2.3 Properties

Trust can be characterized by a number of basic properties [1], [52]. These properties are also applicable to computational trust and are briefly described in this section.

1.2.3.1 Quantifiable and Comparable

There are many different ways in which trust can be represented. For example in [2], trust is represented as a binary value of trust or mistrust. Although a binary representation is easy to understand, it also has the weakness of lack of expressiveness. A simple way to provide more expressiveness is to represent trust as multi-valued. In [1] for example, four different trust degrees are defined. These are “Very Trustworthy”, “Trustworthy”, “Untrustworthy” and “Very Untrustworthy”. It is also possible to represent trust as a continuous variable as opposed to a discrete variable. Common representations include representing trust as a value between zero and one (often seen in cases when trust is treated as a probability) or between negative one and one. Although representing trust as a real number allows for fine-grained distinction, such a representation can make it difficult to discern the meaning behind the calculated trust value. For example, given two trust values 0.73 and 0.74, what is the cause for the trust differences? Is it a rounding

error or is there something fundamentally different about trust in the two trustees? A continuous representation makes discerning trust differences more challenging.

A trustor may feel uncertain about the result of its trust calculation. Some researchers choose to represent this uncertainty as separate from trust. Often times, this is known as reliability ([110], [59]) or confidence ([54], [128], [135]) in the calculated trust. Other researchers choose to integrate uncertainty into trust such as in [23] where trust is represented not as one value but as a range of possible trust values. Here, the larger the range, the less certain the trustor is with regards to the calculated trust value.

1.2.3.2 Subjective

Different trustors may have different views on whether a trustee can be trusted. The differences in trust are caused by the following two factors:

- *Incomplete information.* A trustor may not have access to the complete history of a trustee. Trust is often calculated based on available information. Different trustors may not have access to the same information.
- *Subjectiveness of trustor.* Different trustors may interpret the same information differently. For example, a more forgiving trustor may be willing to accept some failures from a trustee while a less forgiving trustor may not. Different trustors may also have different views on how much uncertainty is acceptable. Emotions like love and friendship could influence trust calculation as well. Finally, different trustors may view the role of a trustee differently. For example, a trustor may value security more than ease of use while this may not be the case for a different trustor. These differences in expectations may influence the level of trust that a trustor has for a trustee.

1.2.3.3 Multidimensional

A trustor may trust a trustee in one dimension but not in a different dimension. For example, a trustor may trust a dentist with his dental health but not trust the dentist's stock picks. A trustor may trust Amazon's EC2 Compute Cloud [8] but not trust its

Mechanical Turk service [9]. A trustor may trust a trustee with transactions that are less than a hundred dollars but not in cases when the transaction is worth a million dollars. Even the environment can play a role in a trustor's trust in a trustee. For example, an environment in which malicious sellers are actively being identified and banned can increase trust in all the sellers in the environment. The associations of a trustee for example, the fact that a trustee is acting on behalf of the government may have an impact on trust as well.

It is possible for trust in one dimension to influence trust in a different dimension. As an example, consider the case of general trust and specific trust [107]. An example of general trust is trust in Amazon [7]. An example of specific trust is trust in one of the services provided by Amazon. If Amazon decided to launch a new service, knowing nothing about the new service, the trustor's trust in Amazon may influence the trustor's initial trust in the new service.

1.2.3.4 Dynamic

A trustor's level of trust in a trustee may change over time. This change may be caused by the presence of new information or the retirement of old information that are no longer relevant. It could also be caused by the lack of new information on a trustee. With no new information, a trustee's behavior could have changed without the trustor knowing anything about it. This increases the trustor's uncertainty with regards to the calculated trust value. It is important for computational trust to be regularly reevaluated so that it can keep up with the dynamic nature of trust.

1.2.3.5 Reflexive, Non-Symmetrical and Non-Transitive

Trust is reflexive as a trustor always trusts itself. Though the level of trust a trustor has for itself may change over time (as trust is dynamic). Since trust is also multidimensional, there is more than one type of self-trust. For example, a trustor that does not trust itself to fix the water leak would hire a plumber to do the job. A trustor that does not trust itself to calculate trust would have to depend on others to calculate trust for it (trust delegation).

In general, trust is non-symmetrical. This means that knowing the level of trust a trustor A has for a trustee B does not help in determining B's level of trust in A. This non-symmetrical property follows from the property of trust being subjective. Therefore, a trust relationship between A and B can be represented as two one-way trusts.

In general, trust is non-transitive. However, under certain semantic constraints, it is possible for trust to be transitive. According to [64], transitivity requires that “the last edge in the path represents functional trust and that all other edges in the path represent referral trust, where the functional and the referral trust edges all have the same trust purpose”. The term “trust purpose” is used to refer to why the calculated trust is needed. Suppose the trustor has a trust purpose for finding a good car mechanic. Referral trust would be the trust in a trustee to be able to recommend a good car mechanic. Functional trust would be the trust in a trustee as being a good car mechanic. If A trusts B to recommend a good car mechanic and B recommends C, then through transitivity, A can trust C to be a good car mechanic. A different example of trust transitivity can be found in a public-key infrastructure (PKI). In PKI, trust transitivity is established through the existence of a chain of certificates (known as a certification path).

In general, trust is weakened or diluted by transitivity [64]. As such, in a chain of A, B and C, A's trust in C cannot be greater than B's functional trust in C.

1.2.4 Conceptual Model

A conceptual model is used to show the “relationships between factors that are believed to impact or lead to a target condition” [93]. A conceptual model should be implementation independent. The conceptual model of computational trust is based on the work of Lewis and Weigert [75]. The model views trust as consisting of three dimensions: *cognitive trust*, *emotional trust* and *behavioral trust*. For the rest of this section, each of the dimensions is described followed by a discussion of the relationships between the dimensions.

1.2.4.1 Cognitive Trust

Cognitive trust is trust based on reasoning. It is knowledge-driven [61] though the information that cognitive trust is based on can be unreliable and incomplete. Given the available information, a cognitive leap is needed to arrive at a trustor's cognitive trust in a trustee [75]. As predicting the future is often impossible, the cognitive leap though based on knowledge is still a leap of faith. Different trustors may have different comfort levels about how much information it needs before it is willing to make a cognitive trust prediction. This is one of the reasons for why trust is subjective.

1.2.4.2 Emotional Trust

Emotional trust or affective³ trust is trust based on feelings and affective bonds. It is non-cognitive and often cannot be justified by the available information. Examples of emotional trust includes trust based on faith, love, friendship, family, common values etc. It is also present in public trust such as trust in doctors, judges, politicians, etc. Due to the presence of emotion, a trustor would feel hurt or betrayed when the trustor realizes that the trustee should not have been trusted. This is seen when relationship ends or when someone who is highly respected such as a judge is caught in a scandal.

1.2.4.3 Behavioral Trust

Behavioral trust is trust as expressed through a trustor's behavior. Experiments such as Prisoner's Dilemma (PD) games [10] can be used to investigate this type of trust. In a PD game, a player can choose to either cooperate or defect. A player who chooses to cooperate can be viewed as having behavior trust in the opposing player while a player who chooses to defect is signaling the opposite. A strict behavioral interpretation of trust can be misleading as trust is not the only factor that could influence a trustor's behavior [75]. For example, a trustor may decide to take action not due to trust but due to pressure

³ In psychology, the term "affect" refers to feeling or emotion [4].

from its superior. An interpretation of this action as trust would lead to the drawing of the wrong conclusion.

1.2.4.4 Trust Relationships

Cognitive trust, emotional trust and behavioral trust are related to each other [75]. The web of relationships is graphically illustrated in Figure 1. To develop emotional trust, a cognitive base needs to be present as it is hard to develop emotions towards complete strangers. When reasoning about whether to trust, a trustor's emotions may influence how the available information are interpreted [72]. A trustor's behavior can be influenced by its cognitive trust and emotional trust. Behavioral trust exhibited by others may influence a trustor's cognitive trust and emotional trust.

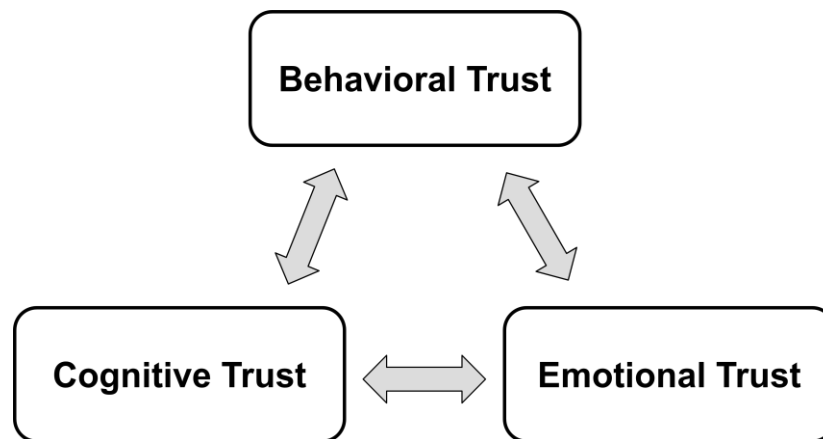


Figure 1: Relationships among Trust Dimensions

In [75], it was pointed out that trust in real life can be decomposed into cognitive trust and emotional trust. For example, ideological trust can be viewed as trust with high cognition and emotion. Faith is trust with high emotion but with a lack of cognition. For mundane everyday decisions such as the route to drive to work or what to have for lunch, trust typically consists of low cognition and low emotion. A more detailed list of examples can be found in [75].

1.2.5 Trust Formation

The conceptual model introduced in Section 1.2.4 forms the basis for computational trust formation. Cognitive trust can be viewed as a function of a trustor's *beliefs* in a trustee [25]. Belief can be defined as the “degree of conviction of the truth of something especially based on a consideration or examination of the evidence” [12]. *Evidence* is any information that can be used in the formation of belief value. An evidence creator is also known as the *evidence source*. There are many different types of evidence. The most common are the trustor's *experience* with a trustee, *recommendation*, *reputation* and *signal*. Experience with a trustee is the knowledge that the trustor gained after interacting with the trustee. Recommendation is experience that a trustor shares with others. For example, when a movie critic writes a review for a movie, the review is the critic's movie recommendation. Reputation is the consensus assessment of a trustee by members of a social network [94]. An example reputation is the Tomatometer of Rotten Tomatoes [106] that is used to keep track of the reputation of movies among movie reviewers. In the example, Rotten Tomatoes is the reputation system. The movie reviewers are members of Rotten Tomatoes' social network. Signal is information that a trustee volunteered to the trustor. For example, by volunteering information showing that the trustee has just passed an inspection (information that a trustor may not be aware), this increases the trustor's beliefs in a trustee.

The formation of emotional trust is based on the trustor's emotions towards the trustee. Behavior trust formation is based on the behaviors exhibited towards the trustee. A graphical illustration of computational trust formation is shown in Figure 2. As computational trust is subjective, it is the responsibility of the trustor to determine how the trust dimensions are to be combined to form computational trust. It is also the responsibility of the trustor to determine how the different trust dimensions influence each other.

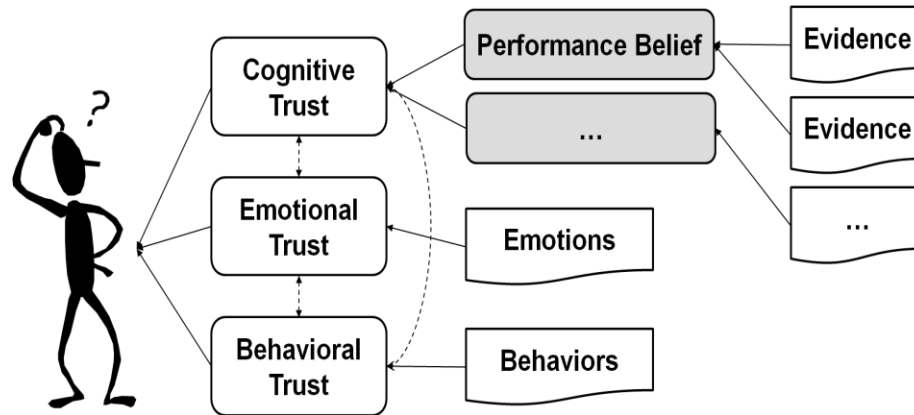


Figure 2: Computational Trust Formation

1.3 Application of Computational Trust

To demonstrate how computational trust can be used in computing, two scenarios are introduced in this section. The scenarios are movie selection and web service selection. The scenarios are used throughout this thesis to highlight different aspects of computational trust.

1.3.1 Movie Selection Scenario

In the movie scenario, the movie selector (trustor) is interested in selecting a movie (trustee) to watch based on the movie's quality. Reading a movie's description does not help in making quality determination. A movie description is typically advertisement from the movie studios. Advertisements can be assumed to be biased towards the movie. To obtain evidence on a movie's quality, a movie selector could obtain reputation values from movie review sites such as Rotten Tomatoes, IMDb (The Internet Movie Database) [60], Metacritic [91], etc. Due to differences in the underlying social network and differences in the reputation calculation algorithm used, different reputation systems may produce different reputation values. For example, the movie "Zombieland" as of October 6, 2010 has a score of 89% on Rotten Tomatoes, a 7.8 on IMDb and a 73 on Metacritic. The differences in reputation values could also be due to the reputation system being malicious or biased. As a result, this thesis has adopted the view that a trustor should have access to as many evidence sources as possible. This is to provide the trustor with the flexibility of selecting the evidence sources for evidence gathering.

An example cognitive trust calculation is shown in Figure 3. In the figure, the movie selector chooses to only contact Rotten Tomatoes and IMDb for the reputation of “Zombieland”. Basically, quality belief is calculated by averaging the normalized reputations. As for cognitive trust, it is calculated by being assigned the same value as the quality belief.

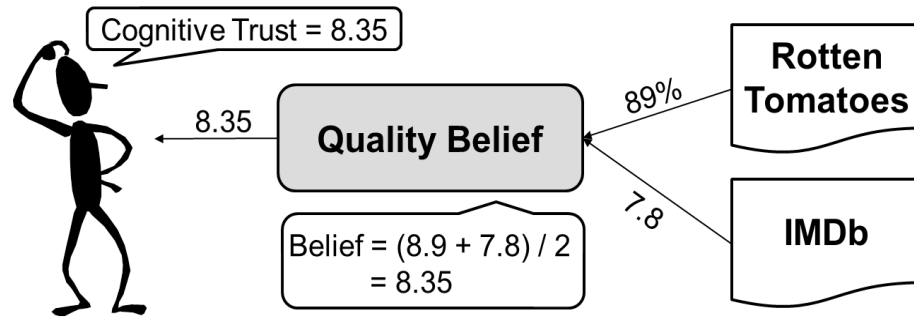


Figure 3: Cognitive Trust Formation (Movie)

In terms of emotional trust, the movie selector may enjoy horror films. Such emotion could influence whether the movie is emotionally trusted. With respect to behavior trust, the movie selector’s observations that a lot of his friends have seen “Zombieland” and liked it can cause the movie to be behaviorally trusted. These different trust dimensions are combined based on the movie selector’s subjective views to form the selector’s computational trust in a movie.

1.3.2 Web Service Selection Scenario

Typically web service selection starts with web service discovery. The return value of discovery is a set of web service instances $S = \{s_0, s_1, \dots, s_{m-1}\}$ that satisfies the consumer’s functional requirements. As for the consumer’s non-functional requirements, let $M = \{m_0, m_1, \dots, m_{n-1}\}$ denote the set of metrics representing the non-functional aspects. Associated with each web service instance is an offer $O_i = \{(m_0, v_{i0}), (m_1, v_{i1}), \dots, (m_{n-1}, v_{i(n-1)})\}$ where v_{ij} is the value that service s_i promises for metric m_j . Many web service selection strategies focus on selecting a web service instance based on offers. Such approaches are often risky. A web service may not keep its promise to provide the agreed upon offer. This could be due to the service having

exaggerated or lied about its capabilities (intentional) or the service being overloaded (unintentional). Trust can be used to assess the validity of an offer. Figure 4 graphically depicts an example of how cognitive trust in a web service is calculated based on beliefs: qualification, timeliness, accuracy and coverage.

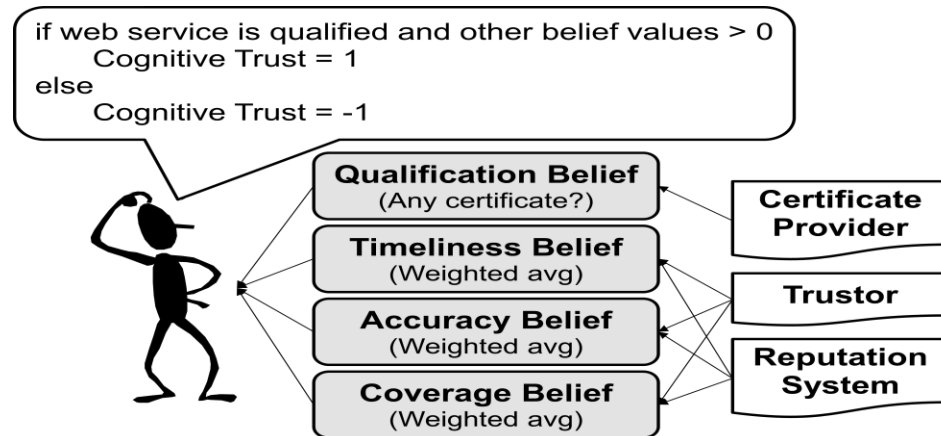


Figure 4: Cognitive Trust Formation (Web Service)

In terms of the qualification of the web service, this could be deduced from the qualification signals (certificates) issued by neutral third parties (certificate providers) that has validated the web service's qualification. As for the timeliness of the web service, i.e., whether computation required can complete on time or not, this belief is formed through the weighted average of the consumer's past experiences with the service and the service's reputation. The results produced by a web service could be evaluated by the consumer in terms of accuracy and coverage. The results are accurate if they are correct. The results have good coverage if the breadth of the solutions provided meets the trustor's expectations. As both accuracy and coverage cannot be obtained through monitoring, they are obtained by querying the consumer after service invocation. Accuracy belief and coverage belief are based on applying weighted average on the consumer's past experiences with the service and the service's reputation. Finally, cognitive trust is calculated from the four calculated beliefs. In this case, if a web service is believed to be qualified and the values calculated for timeliness belief, accuracy belief and coverage belief are all greater than zero, then cognitive trust is assigned a positive value of 1. Otherwise, cognitive trust is assigned the negative value of -1.

In the case of timeliness belief, accuracy belief and coverage belief, the beliefs are formed from evidence of multiple types. This is to account for the fact that evidence of different types has different properties. Between experience, recommendation and reputation, experience is considered the most important as it is created by the trustor [63]. It is assumed that the trustor is always non-malicious towards itself⁴. A trustor also always has its own best interest in mind. This may not be the case with recommendation created by recommender and reputation created by reputation system. Recommendation and reputation however can be used to guide the trustor in terms of which trustee to invoke to gain experience. Moreover, the number of past experiences that a trustor has with a trustee could be very low if the trustor had very few interactions with the trustee. The number may be too few to be representative of the trustee's behavior. In this case, recommendations and reputation may be valued more. Between recommendation and reputation, recommendation is usually valued more than reputation. With recommendation, the trustor determines the recommenders to be contacted. This is based on the assumption that the trustor is familiar with the behavior of the recommenders. Without familiarity, a trustor has to depend on reputation. With reputation, its calculation is often opaque to the trustor. This means that the trustor may not be able to determine the effectiveness of the gathered reputation. As for signals, third-party signals are valued more than signals generated by the trustee. This is due to the fact that a signal created by the trustee about itself can be self-serving.

As different evidence types have strengths and weaknesses, this thesis has adopted the view that having access to multiple evidence types is desirable in belief calculation. This is similar to views found in existing work such as [59] and [110]. For each evidence type, multiple evidence sources should be accessible. In the case of web services, reputation systems may include [78], [87] and [140].

In terms of emotional trust, this could be influenced by the branding of a web service. In fact, the use of branding to create an affective bond between a consumer and a

⁴ This does not mean that the trustor cannot act maliciously towards others.

brand (e.g. through shared values such as caring about the environment) is known as emotional branding [47]. With respect to behavior trust, a consumer is more likely to behaviorally trust a web service that is popular and used by many. The different trust dimensions are combined based on the consumer's subjective views to form the consumer's computational trust in a web service.

1.4 Challenges to Computational Trust

The idea of applying a general model of trust to computing was first proposed by Marsh [84] in 1994. Since then, certain elements of trust such as reputation has gained widespread acceptance (e.g. eBay [38] and Amazon). However, the concept of a general trust model has so far failed to gain traction. We believe that this is due to the inherently difficulty in implementing a general trust model. For computational trust, the challenges are as follows:

- *Lack of standards for discovering and accessing evidence.* This means that it is up to the application developer to implement all needed evidence discovery and evidence gathering protocols. This is inefficient as applications may implement the same protocols leading to code duplication. Also, if the trustor has preferences towards a certain set of evidence sources, this would need to be configured for each application individually. Moreover, if the trustor would like to use the evidence provided by an evidence source, it is not always possible as the protocols needed may not be supported by an application.
- *Lack of standards for representing evidence.* Evidence can be represented as a rating. A rating is the result of evidence evaluation where the result is represented as a position on a scale [101]. Ratings can come in different forms. For example, ratings used by Rotten Tomatoes are in the form of 0 to 100 percent. Ratings used by IMDb are in the form of 1 to 10. There are other forms of evidence representations as well. For example as a X.509 certificate or in the form of a text review. Some evidence may even be encrypted to ensure security and privacy. The lack of a standard evidence representation makes it challenging for an application to be able to understand what the different evidence are representing.

- *Evidence Filtering.* Not all the gathered evidence should be used in belief calculation. Evidence may be fabricated by malicious evidence sources to intentionally mislead the trustor. Evidence may be biased by the views of evidence sources. For example, a movie reviewer (evidence source) that is biased against horror films may give a horror film a worse rating than what a regular moviegoer would consider to be fair.

According to [63], there are currently two main approaches to evidence filtering. One approach is to simply exclude or give low weight to evidence that are outliers. This is based on the assumption that in any environment, the majority of the evidence are non-biased and non-malicious. In [137] for example, a beta distribution is computed of the evidence. Any evidence that is less than the 1% quantile or greater than the 99% quantile are filtered out. A different approach is for the trustor to identify those evidence sources that can be trusted to provide the trustor with quality evidence. When evidence is needed, only those trusted evidence sources would be contacted for evidence gathering. An example can be found in [30] where evidence source trust depends on whether the evidence source agrees with the trustor.

- *Computational Trust is subjective and multidimensional.* There are many different ways in which computational trust in a trustee can be calculated. Factors such as the trustor's emotion (e.g. feeling pessimistic), the trustor's preferences (e.g. valuing accuracy over coverage), the decision the trustor is trying to make (e.g. movie selection or web service selection), the importance of the decision, etc. could all have an influence on computational trust calculation.

1.5 Contributions of Thesis

Listed below are the contributions of this thesis:

- A general trust model known as computational trust. The model is based on work by Lewis and Weigert [75]. As a result, unlike most work on trust in computing, computational trust has a social psychological underpinning. The model is also different from other work in that it considers trust dimensions and treats evidence

based trust (cognitive trust and behavioral trust) and non-evidence-based trust (emotional trust) differently. When implementing the computational trust model, a number of adaptations have been made to Lewis and Weigert's model including the merging of cognitive and behavioral trust, deriving emotions from the trustor's knowledge of the trustee and the simplification of the relationships among the trust dimensions

- An architecture for supporting computational trust formation. The architecture consists of a middleware known as SCOUT (Services supporting COmpUtation of Trust). SCOUT consists of a set of web services that can be used by Trust Calculators for computational trust calculation. The calculated computational trust can be used to support decision making. The computational trust architecture is designed to address the challenges outlined in Section 1.4.
- Algorithms for computational trust formation.
- Experiments, evaluations and scenarios demonstrating the feasibility of the proposed architecture in supporting computational trust formation.

1.6 Thesis Organization

The remaining chapters of this thesis are organized as follows. Chapter 2 provides more background on trust. Chapter 3 reviews the trust literature. Chapter 4 provides a high level overview of the computational trust architecture. The components of the architecture are introduced in Chapter 5 and Chapter 6. The algorithms implemented in the architecture are described in Chapter 7. Our prototype implementation is described in Chapter 8. Experiments using the prototype are described in Chapter 9. The prototype is applied to two different scenarios in Chapter 10. We evaluated the architecture with respect to our stated goals in Chapter 11. Finally, the thesis ends with conclusions and future works in Chapter 12.

1.7 Summary

Trust is a concept has been investigated by many different disciplines including computing. There are a number of weaknesses to the current computing approach to trust. This could be addressed through the adoption of a more general trust model. In this thesis, the trust model is known as computational trust. It is based on research on interpersonal trust in the field of social psychology. A definition for computational trust is provided along with discussion of its properties, conceptual model, formation and application. The rest of this thesis focuses on addressing the challenges to calculating computational trust.

Chapter 2

2 Trust Primer

The focus of this chapter is on providing more background on trust. As computational trust is a type of trust, the discussions in this chapter also apply to computational trust. This chapter is divided into five sections. The first section describes the preconditions for trust. Next, several concepts that are related to trust are introduced and explained. This is followed by a discussion of the role that privacy plays in trust calculation. The fourth section of this chapter covers the relationship between trust and risk. The last section discusses how trust can be used in decision making.

2.1 Preconditions for Trust

Trust is not always needed for decision making. In [69], the authors identified three preconditions for trust. The preconditions are *dependence*, *uncertainty* and *vulnerability*. Dependence is about having to rely on others to fulfill a specific need. Uncertainty occurs when one is not one hundred percent certain about the outcome of a decision. As for vulnerability, it is the cost for making of a wrong decision. Uncertainty and vulnerability together are commonly known as risk [69]. There would be no need for trust if a decision poses no risk to the trustor [82].

In the movie selection scenario, the selector depends on the movie being of high quality. Without actually seeing the movie, there is always uncertainty about the actual movie quality. Finally, making the wrong movie selection could result in wasted money and time. As the scenario satisfied all three preconditions, trust can be used in the scenario to facilitate movie selection. Similar arguments can also be made for the web service selection scenario.

2.2 Concepts Related to Trust

In this section, several concepts are introduced with the goal of clarifying how these concepts are related to trust.

2.2.1 Untrust, Distrust and Mistrust

In [86], the authors introduced the concepts: *untrust*, *distrust* and *mistrust*. Mistrust is used to describe trust that turns out to be misplaced. Distrust is the negative form of trust where the trustee is believed to be actively working against the trustor. Finally, untrust is when there is trust in a trustee but not enough to overcome the perceived risk. A graphical illustration of the different concepts is shown in Figure 5. In the figure, the cooperation threshold is used to represent the perceived risk associated with a decision.

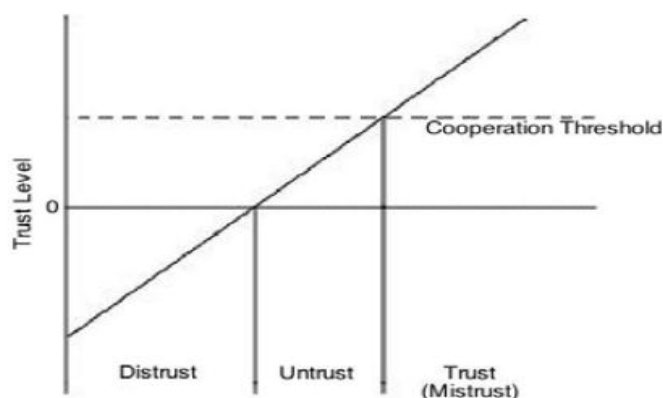


Figure 5: From Distrust to Trust [86]

In movie selection and web service selection, a selection is only made if trust in a movie or a web service is above the cooperation threshold. Distrust occurs if a movie is believed to be of low quality or if a web service is believed to be incompetent or malicious. Untrust occurs if trust exists but just not enough for a selection to be made. As for mistrust, this occurs after a selection when the trustor discovers that it should not have selected the movie or web service.

2.2.2 Trustworthiness

Trust and trustworthiness should not be used interchangeably [31], [45]. Trust is associated with the trustor and could influence the behavior of a trustor. Trustworthiness is a characteristic of the trustee. A trustee is perceived to be trustworthy. A trustor is more likely to trust a trustee that exhibits the characteristic of trustworthiness [122].

In movie selection, a movie directed by a competent director is perceived to be trustworthy. Whether a selector would trust such a movie is a completely separate question. A web service could behave in a trustworthy manner by trying its best to satisfy all consumer requests. Whether the displayed trustworthiness is enough for a consumer to trust the web service is subjective and consumer dependent.

2.2.3 Contract

A trustor and a trustee may decide to enter into a contract. By doing so, the trustor and trustee are delegating the responsibility for contract enforcement to some agreed upon authority. An authority could be the legal system or an impartial mediator. It is the authority's responsibility to assess whether there is any contract violation and if so to impose punishment. The introduction of a contract does not mean that there is no need for trust [25]. Trust in the authority to be able to enforce signed contracts underlies the entire rationale for contracts. A trustor also needs to trust that that the trustee would be deterred by the punishment. Otherwise, the trustee could choose to accept the punishment and violate the contract at will.

Contracts are not always available for all decision making. For example, there is usually no contract for movie watching. Contracts are more commonly associated with web services. A web service could offer a standardized contract to all its consumers. A web service could also choose to negotiate with each of its consumers to offer customized contracts to suit each of the consumer's needs.

2.2.4 Confidence

A trustor trusts a trustee to complete the assigned job. A trustor is confident that the trustee can complete the assigned job. In the previous examples, trust and confidence are used interchangeably. Different researchers have offered different views on how to distinguish between the two concepts. According to Luhmann in [82], confidence is the expectation of not being disappointed. For example, a trustor is confident that the light would turn on with the flip of a switch. Luhmann argues that trust also involves expectation except it presupposes a situation of risk. With trust, a trustor is given a choice while this is not the case with confidence. For example, whom a trustor should buy a used

car from is a trusting decision. This is not the view in [31] where trust is defined as “an attitude of confident expectation...”. In this definition, confidence is considered a part of trust which seems to contradict the view of Luhmann [86].

In [118], the authors argued that trust is only relevant in risky situations where familiarity with the trustee is low. Moreover, trust can only applied to a trustee that is a person or a person-like entity. Confidence on the other hand is based on a high level of familiarity and can be applied to just about anything. Carole Smith in [121] proposed an alternate view where trust is seen as a moral exercise of free will. With trust, there is uncertainty about the possible outcomes and a lack of objective information. With confidence, there is an explicitly established outcome where information is collected in an objective and scientific manner. For example, holding public servants accountable through performance reviews can increase public confidence in a public institution. The focus on performance measurements however can have a negative effect instilling distrust between employees and managers.

There is a distinct lack of agreement among different researchers on how to distinguish between trust and confidence. As the disagreement is mainly a naming issue (i.e., when should trust be used and when should confidence be used to describe a situation), we do not see how introducing the concept of confidence could benefit decision making. As a result, we ignore confidence and focus on trust exclusively in this thesis. Note that in some computer science literature, the term confidence is sometimes used to refer to confidence in the calculated trust. This specific usage of the term is unrelated to the discussion in this subsection and therefore the usage is acceptable.

2.3 Privacy

It is up to an evidence source to determine whether evidence should be disclosed to a trustor. Without evidence, cognitive trust cannot be calculated. Yet when evidence is disclosed, privacy may be lost. It is difficult to recover from privacy loss. As an example, consider a trustee’s identity. Identity is needed for trust calculation. For maximum privacy, a trustee could choose anonymity. However, it is almost impossible to calculate trust if a trustor does not know who the trustee is. As a result, trustees are often provided

with lesser privacy through pseudonymity. Even with pseudonymity, it can be difficult to hold a trustee accountable for its bad behavior if the trustee can change its pseudonym at will. So for pseudonymity to work, it needs to be difficult for a trustee to acquire more than one pseudonym.

In the field of trust negotiation [13], disclosure policies are used to implement conditional evidence disclosure. The goal is to provide privacy protection by only revealing evidence on an as needed basis. However as pointed out by Danah Boyd in [18], privacy is more than just control over evidence disclosure. It is also about providing the user with control over how its evidence is being distributed. There is a difference between making private evidence publicly accessible and having that evidence being widely publicized without the user's consent [18].

2.4 Risk

Trust is a concept that is inherently linked to risk [22], [65]. Risk involves uncertainty and vulnerability. In [73], risk is defined as the likelihood and severity of an accident. In [88], the authors argued that risk should consider more than just the accidents or negative outcomes. If a decision involves the possibility of both positive and negative outcomes, the aggregate level of risk (i.e., risks from all possible outcomes of a decision) should be different than if there is only the possibility of negative outcomes. This view of risk is the one that is adopted in this thesis. Risk is also considered to have the following properties: quantifiable, subjective, multidimensional and dynamic.

In [25], the authors argued that there are two types of risk. The first is the risk of failure. This is the gain the trustor could have had had it not make the wrong decision. The second is the risk of wasted effort. This is the investment of the trustor that had gone to waste. Example investments could be time or money. As for how trust and risk are related to each other, [45] proposed three possibilities:

- *Mediating relationship.* In this relationship, trust does not have a direct impact on decision making. Instead, trust is used to reduce the riskiness of a decision. It is risk that is responsible for determining whether a decision should proceed.

- *Moderating relationship.* In this relationship, risk is responsible for moderating the effect of trust on decision making. Basically when risk is high, trust has an influence on decision making. When risk is low, trust is considered irrelevant to decision making.
- *Threshold model.* In this relationship, trust and risk are both independently formed. A trustor would only implement a decision if its level of trust surpasses its threshold of risk.

Of the three relationships, the mediating relationship is reported as having the widest acceptance among ecommerce researchers [45]. As for the threshold model, although it is widely used in computing, surprisingly it has no ecommerce research underpinning [45].

2.5 Decision Making

There are many factors that could influence a trustor's decision making. Several example factors include company policies, the cost of each decision choices, trust in a trustee, loyalty to a trustee, etc. During decision making, a trustor may prefer a certain trustee but is held back by its lack of trust in the trustee. An example of this could be whether to use an unknown web service that is being offered at a great price point. If more trust is needed, one possible solution could be to demand more accountability from the trustee. An example could be to demand heavier punishment for contract violation. A different approach to solve this problem could be to simply reduce the risk associated with the decision. For example by buying insurance, a trustor could hedge against possible negative outcome thereby reducing the need for high level of trust in the trustee.

2.6 Summary

For trust to be useful in decision making, there needs to be dependence, uncertainty and vulnerability associated with the decision. The chapter introduced several concepts related to trust including untrust, distrust, mistrust, trustworthiness, contract and confidence. The concepts are explained and distinguished from trust. The role that privacy and risk plays in trust is also explained. Finally, the chapter ends with a discussion of trust's influence on decision making.

Chapter 3

3 Literature Review

The focus of this chapter is to distinguish our work from others in the trust literature. This chapter is divided into three sections. The first section is a survey of work on trust and reputation. Next, arguments for a middleware approach to computational trust formation are presented. The arguments are based on insights gained from the literature survey. Finally, the last part of this chapter covers work on trust middleware. None of the work on trust middleware covers all the elements that we have identified as critical to supporting computational trust formation.

3.1 Survey of Trust and Reputation




The literature survey on trust and reputation covers four different areas of computing. These are the World Wide Web, peer-to-peer environments, virtual communities (e.g. agent societies, grid computing, business networks, etc.) and pervasive environments (e.g. mobile and wearable computing, context-aware computing, etc.). In each area, we describe in detail how trust and reputation is used. Note that the survey is not intended to be complete. Instead, the focus is on presenting representative works in each of the computing areas. As there is no commonly accepted definitions for trust and reputation, footnotes are used to point out differences in definitions whenever necessary.

3.1.1 The World Wide Web (WWW)

There are many different uses for trust and reputation on the World Wide Web. For example, trust and reputation can be used to promote e-commerce. Such uses could be found in Amazon [7], Best Buy [14], [26], eBay [38], Epinions [39] and [83]. Trust and reputation can also be used for identifying quality information. Examples of such use include Advogato [3], PageRank [19], IMDb [60], Metacritic [91], Rotten Tomatoes [106], Slashdot [120], Stack Overflow [126] and Web of Trust [135]. In the field of Semantic Web, trust and reputation are used to determine how much an information source should be trusted [48] [104]. For the rest of this subsection, three works on the WWW are described in detail.

3.1.1.1 eBay

In eBay [38], a reputation system (feedback forum) is responsible for calculating the reputation of each eBay member. After each transaction, a buyer and seller are allowed to rate each other by leaving feedback. A feedback consists of both a rating (positive, neutral or negative) and a short comment. If a member is a buyer, it can also leave detailed ratings (scale of one to five stars) on the seller. The ratings are on criteria such as communication and shipping time. The reputation of a member is calculated as the difference between the positive and negative ratings left by eBay members (eBay's social network). To capture a member's recent behavior, the ratings from the last one, six and twelve months are summarized and presented as shown in Figure 6.

Recent Feedback Ratings (last 12 months)			
	1 month	6 months	12 months
 Positive	4284	16710	32847
 Neutral	48	295	561
 Negative	110	576	966





Detailed Seller Ratings (last 12 months)		
Criteria	Average rating	Number of ratings
Item as described		26352
Communication		26308
Shipping time		26307
Shipping and handling charges		26260

Figure 6: eBay's Feedback Forum

Although there is a number of weaknesses to eBay's reputation system such as its counter intuitiveness (member with higher reputation is not always better on eBay) and disproportionately positive feedbacks [102], the feedback forum has been shown to be effective in encouraging transactions [102]. In fact, an analysis done by [81] on the auctioning of collectible one-cent coins found that a seller's reputation have a statistically significant impact on the coin auction prices.

3.1.1.2 Web of Trust

Web of Trust (WOT) [135] is a browser add-on that is designed to promote safe browsing. WOT uses user ratings and carefully selected information sources (e.g. listings of phishing sites) to calculate the reputation of websites. A reputation in WOT is based on assessments of four different components: trustworthiness, vendor reliability, privacy and child safety. WOT also calculates its confidence in the calculated reputation.

Confidence is based on the amount of evidence and the quality of the evidence sources. A WOT user can also leave comments on a site. A comment could belong to one of seventeen categories: good site, entertaining, spam, useless, etc. In WOT, the reputation of a web subdomain actually contributes to the reputation of the parent web domain.

The reputations calculated by WOT are displayed as coloured icons next to search results and links in emails. For example, a website with excellent reputation is assigned a green icon while a site with poor reputation is assigned a red icon. A website could also obtain a WOT Trust Seal as proof of its trustworthiness. The seal can be obtained for a monthly fee. As of writing, WOT has rated over 28 million sites with over 10 million add-on downloads.

3.1.1.3 Object Scoring by Chen and Singh

In [26], it is proposed that the reputation of a rater (member of a social network) should be structured as a tree. The reputation tree has the rater's reputation as the root with each of its children being the reputation of the rater in different knowledge domains (subject areas). To build a reputation tree, the first step is to perform comment evaluation. A comment refers to the numeric rating giving to an object (product or service) along with the associated text review. After comment evaluation, the reputation of a rater in a specific knowledge domain can be calculated. This is based on all the evaluations in the specific knowledge domain. Finally, the reputation of a rater can be calculated based on the rater's reputation in different knowledge domains.

The score of an object is calculated as the weighted average of the object's received ratings. The weight of a rating is assigned based on the rater's reputation. If a rater's reputation in a specific knowledge domain is unavailable, the solution is to move up the reputation tree and use the rater's more general reputation instead. As for confidence in the object score, this is calculated based on the total number of received ratings, reputation of raters and degree of consistency among the ratings. During experimentation, the authors compared their work with Epinions [39] and discovered that raters with high reputation actually correspond well to those good raters selected by the site manager and users of Epinions.

3.1.2 Peer-to-Peer (P2P) Networks

Trust and reputation are used in P2P networks to improve resource selection. They also encourage cooperation by ensuring that good peers are rewarded while bad peers are punished. Some works in P2P include [2], [20], [32], [54], [55], [56], [68], [71], [76], [77], [90], [116], [123], [127], [130], [133], [134], [139] and [143]. In this subsection, six of those works are described in greater detail.

3.1.2.1 XREP

XREP [32] is a reputation⁵ sharing protocol that is designed for Gnutella-like systems. In XREP, each servent⁶ and each resource is assumed to have a unique identity. The XREP protocol consists of five phases: resource searching, resource selection and vote⁷ polling, vote evaluation, best servent check (confirm resource digest is valid from best servent) and resource downloading (with a confirmed resource digest, one could download resource from any servent). Votes take only two values: 1 for positive opinion and 0 for negative opinion.

A distinguishing feature of XREP is that it not only calculates the reputation⁸ of servents but also the reputation of resources as well. By considering both types of reputation, this allows XREP to solve the cold-start problem, i.e., the problem of the lack of reputation on newcomers. For a new resource, XREP proposes that the resource can establish its reputation through the reputation of the servent. As for a new servent, it can gain reputation by offering access to reputable resources.

⁵ Due to differences in definitions, “reputation” actually corresponds to “recommendation” in this thesis. One could also think of XREP as a protocol for how recommendations can be exchanged in Gnutella-like systems.

⁶ A servent is a node in a P2P network. It is both a trustee and an evidence source.

⁷ A “vote” is basically a recommendation represented as a rating.

⁸ Due to differences in definitions, “reputation” as used in this paragraph actually corresponds to “trust” in this thesis.

3.1.2.2 Bayesian Reputation System by Buchegger and Le Boudec

In [20], a reputation rating⁹ is defined as the opinion formed by node i about node j 's behavior in a P2P system. A reputation rating is updated based on a node's first-hand observation (experience) and first-hand observations shared by other nodes (recommendations). In the former case, a modified Bayesian approach is used to update the reputation rating. The modifications proposed in [20] include giving past evidence less weight and to forget old observations through decay over time. In the latter case, the reputation rating is updated by taking the received observations into account. As nodes may provide false reports, [20] proposes to only update the reputation rating if the reporter¹⁰ is considered trustworthy or if the reported observations is similar to the node's own observations. The trustworthiness of a reporter (trust rating¹¹) is calculated using the same modified Bayesian approach based on observation similarity.

[20] also proposed to classify the different P2P nodes based on the expected value of their beta distribution. For example, if a node i is willing to tolerate at most 25% misbehavior, then a node j will be classified as normal if its expected value is less than or equal to 0.25. Otherwise, node j will be classified as misbehaving. The same approach can also be used to classify the trustworthiness of reporters.

3.1.2.3 Fuzzy Trust by Griffiths, Chao and Younas

In [54], fuzzy logic is proposed as a way to calculate trust in P2P systems. The idea is for a trustor to keep track of its experiences interacting with other member peers in the dimensions of success, cost, quality and timeliness. Trust is then calculated from experiences using fuzzy inference rules. The input fuzzy term is experience which could have the value of negative big, negative medium, negative small, zero, positive small, positive medium and positive big. The output fuzzy term is trust which could have the

⁹ Due to differences in definitions, "reputation rating" actually corresponds to "trust" in this thesis.

¹⁰ A "reporter" is basically a recommender in this thesis.

¹¹ Unlike in [20], this thesis does not distinguish between trust in nodes and trust in reporters. So, "trust rating" actually corresponds to "trust" in this thesis.

value of high distrust, distrust, undistrust, untrust, trust and high trust. Confidence is also calculated for each experience dimension to ensure that there are sufficient experiences available to perform trust calculation. If the calculated confidence is below a certain threshold, then a default experience value (chosen based on the trustor's trusting disposition) is used as oppose to the trustor's actual experiences. The paper also points out that sometimes selecting a peer based solely on trust may not be enough. So, additional fuzzy inference rules could be introduced that combines trust with other decision factors (e.g. advertised cost).

3.1.2.4 DCRC and CORC

In P2P networks like Gnutella, cooperation among peers is essential to the searching and downloading of content. It is therefore important for there to be incentives to encourage cooperation among peers. In [55], the authors proposed a reputation based incentive that is tied to a peer's level of participation in the system. The paper proposed two different ways to calculate the reputation score¹² of a peer. These are the debit-credit reputation computation (DCRC) scheme and the credit-only reputation computation (CORC) scheme.

With DCRC, a peer does not calculate its own reputation score. Instead, this is outsourced to a reputation computation agent (RCA). A peer presents its actions within the P2P network to the RCA and in return gets back its new reputation score. There are three types of credits that can be claimed by a peer. These are the Query-Response Credit (QRC) for proof of having responded to a query message, the Upload Credit (UC) for proof of having served content to a peer and the Sharing Credit (SC) for proof of having been online and for having contents that could be shared with others. As for debits, the only type of debit supported is the Download Debit (DD) for proof of content download. Together, the debits and credits are used to compute a peer's reputation score.

¹² A "reputation score" is basically reputation that is calculated using either DCRC or CORC scheme.

CORC is functionally similar to DCRC with the only difference being that it does not support DD. There is incentive in CORC for a peer to stop contributing to the P2P network as soon as it has obtained a good reputation score. To prevent this from happening, the RCA time-stamps the calculated reputation score for expiration. Both DCRC and CORC have their respective strengths and weaknesses. A detailed comparison of the two reputation computation schemes is available in [55].

3.1.2.5 Anomaly Detection by Stakhanova, et al.

In a P2P network, trust can be used to determine whether a peer should accept or send traffic to a different peer. In [127], trust is calculated based on four types of actions: resource search, resource upload, resource download and traffic extensiveness. Each time an action is taken, it can be classified as good (successful outcome) or bad (fail outcome). The classification is based on anomaly detection. The idea is for the creation of a peer profile that establishes a peer's typical behavior in the P2P network. Periodically, a peer's online session data is analyzed using a one-class support vector machine (SVM) for any deviation from the peer profile. Once an anomaly is discovered, Chebyshev's rule is used to determine the impact the anomaly has on an action.

The trust score¹³ of a peer is calculated as the percentage of bad actions during a given time period. In [127], the authors proposed to reject all traffic from peers with trust score that is greater than the distrust threshold. Traffic from peers with trust score that is lower than the full trust threshold is always accepted. If a peer's trust score is between the two thresholds, then only part of the peer's traffic is accepted.

3.1.2.6 PET and M-CUBE

PET [77] is a trust model that is designed to encourage cooperation in P2P resource sharing. The trust model consists of a reputation model and a risk model. Reputation¹⁴ in

¹³ Due to differences in definitions, "trust score" actually corresponds to the inverse of "trust" in this thesis.

¹⁴ Due to differences in definitions, "reputation" in the paper actually corresponds to "trust" in this thesis.

PET represents the accumulative assessment of a peer's long-term behavior. Interaction-derived information (experience) and recommendations are used in reputation calculation. Risk¹⁵ in PET represents a peer's short-term behavior. Risk is calculated using interaction-derived information and is normalized to the worst case scenario. When calculating trust, weights are assigned to both reputation and risk. The paper recommends risk be assigned a high weight especially in dynamic systems.

M-CUBE (Multiple Currency based Economic Model) [76] is a decentralized trading scheme that utilizes the trust calculated by PET to enforce cooperative resource sharing. The idea is for each peer to issue its own currency and to set its own price for the offered resources. If a peer needs the resource offered by a different peer, the first step is to obtain the currency of the other peer. This can be done through a currency exchange protocol where initially the currency ratio is set to one to one. The currency ratio will self-adjust over time. This is to force a less trusted peer to pay more than a trusted peer. Once currency has been obtained, the peer could now use the currency to acquire the shared resource. If a peer refuses to share its resources, its trustworthiness will decrease making currency exchange more expensive over time. When the trustworthiness of a peer drops below a certain threshold, the peer would get banned from the P2P community.

3.1.3 Virtual Communities

A virtual community is in many ways similar to a real life community [1]. The role that trust and reputation plays in a physical community therefore are applicable to virtual communities as well. Some work on virtual communities includes the following:

- Agent societies: [24], [25], [53], [59], [84], [100], [110], [111], [114], [142] and [144]
- Grid computing: [11], [79], [98], [129] and [136]

¹⁵ The paper's definition of "risk" is different from the definition in this thesis. In fact, their definition is really just a different way to calculate trust.

- Web services: [5], [78], [87], [115] and [144]
- Others: [1], [108] and [147]

In this subsection, six representative works are described in detail.

3.1.3.1 Regret

The Regret system [110] assumes that the reputation¹⁶ of an agent is based on three dimensions: individual dimension, social dimension and ontological dimension. The individual dimension is concerned with direct interaction (experience). Sociograms (graphs representing social relations) are used to calculate reputation in the social dimension. In Regret, it is assumed that each agent has its own sociograms. These sociograms represent an agent's view of the competition, cooperation and trade within the agent society. Regret supports three types of social reputations: witness reputation, neighborhood reputation and system reputation. The ontological dimension is concerned with the creation of complex reputation. An example complex reputation could be an agent's reputation as a swindler that could be based on the agent's reputation to overcharge and its reputation for providing poor quality products.

Implementation wise, Regret is designed to be modular [109]. This means that an agent can pick and choose the types of reputation to be used in reputation calculation. For example, if one does not have any experiences with an agent, it could choose to calculate the agent's reputation based solely on social reputations. After some interactions with the agent, there is now enough experiences so that any future reputation calculation will include both individual and social dimensions.

3.1.3.2 FIRE

FIRE [59] is a trust and reputation model that is designed for open multi-agent systems (MAS). Specifically, trust in FIRE is calculated based on interaction trust, role-based trust, witness reputation and certified reputation. Interaction trust is calculated based on

¹⁶ Due to differences in definitions, "reputation" in the paper actually corresponds to "trust" in this thesis.

the direct experiences of an agent. Role-based trust uses rules to determine how much an agent can be trusted. Witness reputation is calculated by employing a referral based approach. Certified reputation is calculated based on certified references. By taking into account a variety of information, this means that even if some information sources are unavailable, trust can still be calculated. FIRE assumes that all agents are honest when exchanging information. The possibility of there being disinformation is considered to be future work.

3.1.3.3 Grid Computing by Azzedin and Maheswaran

In a grid computing system, trust [11] can be used to encourage resource sharing among grid domains (GDs). Associated with each GD are two virtual domains: the resource domain (RD) and the client domain (CD). It is assumed that a trust agent exists in each GD and is responsible for calculating the GD's trust in the other domains. As for the resources and clients within a GD, they automatically inherit the trust attributes of their RD and CD. Trust in a domain is calculated based on direct trust (experiences) and reputation (recommendations from other domains). For domain-to-domain resource sharing to occur, it is required for both GDs to have enough trust in each other (*Trust > Required Trust Level*).

3.1.3.4 Service Selection by Ali, Ludwig and Rana

A framework for web services discovery and selection is proposed in [5]. A trust relationship in [5] consists of three phases: unknown, volatile and mature. A trust relationship starts in the unknown phase. After some interactions, the relationship enters the volatile phase. After developing a deeper understanding of a web service, the relationship enters the more stable mature phase. A trust policy is used to capture a user's trust disposition. A policy can also be used to specify the conditions for transitioning from one trust phase to another.

Trust in [5] is calculated using QoE (Quality of Experience) and QoC (Quality of Compliance). QoE is concerned with how the service delivered is when compared to the user's expectations. QoC is concerned with whether the service delivered met the agreed

upon quality of service (QoS). As for trust calculation, it is based on Fuzzy Cognitive Maps (FCM). The paper proposed to only trust a web service if the subjective expected utility to trust is greater than the subjective expected utility to distrust.

3.1.3.5 The TuBE Trust Management System

The TuBE trust management system [108] is responsible for facilitating inter-enterprise collaborations. The Guard component of TuBE intercepts SOAP messages from collaborating partners and passes the messages on to the trust decision maker. The decision maker needs to decide whether a SOAP message should be allowed to proceed or not. To make a trust decision, the decision maker passes information on the trustee along the action that the trustee would like to take on to the data processing component. In return, the decision maker receives a risk analysis and a constraint set. The constraint set represents the acceptable risk given the current situation. By determining whether the risk estimates can fit within the risk constraints, a decision is made on whether the SOAP message should be blocked or not.

The data processing component of TuBE consists of four subcomponents. These are risk, importance, reputation¹⁷ and context evaluators. Experience data used by the reputation evaluator comes from the reputation management component. Reputation in TuBE is calculated using locally monitored experiences and experiences reported by other peers in the reputation network¹⁸.

3.1.3.6 Information Trustworthiness by Zuo and Panda

In [147], the authors presented a model for evaluating trust in information in a virtual organization (VO). The model is based on the concept of objects where an object is a piece of information on a topic or issue (e.g. unemployment rate). Different subjects may have different views. This could result in the creation of multiple object versions. Since

¹⁷ Due to differences in definitions, “reputation” in this paper actually corresponds to “trust” in this thesis.

¹⁸ This is basically recommendations from other peers.

new information can be created from existing information, it is possible to create a compound object version from object versions as well. How a compound object version is created can be represented using a version dependency tree.

There are many different ways to calculate trust in an object version. For example, trust can be calculated by multiplying the owner's trust in the object version and the trust the evaluator (trustor) places on the owner. If the calculated trust is not enough to satisfy the needs of the evaluator, the principle of object trust combination can be used to increase the evaluator's trust in the object version. The idea here is that if two object versions are similar but are created through different approaches (have dissimilar version dependency trees), then the average of the two object versions should be more trustworthy than the individual object versions. For example, if two subjects through completely different approaches arrived at the same conclusion with regards to the unemployment rate, then the "multiple-proofs" should increase the level of trust in the unemployment rate.

3.1.4 Pervasive Computing Environments

Works on trust and reputation in pervasive computing environments include [22], [23], [52], [80] and [112]. In this subsection, the discussion focuses on two such works.

3.1.4.1 The SECURE Project

The SECURE project [22] is concerned with trust-based security in pervasive computing environments. Trust and risk are viewed as having a mediating relationship in SECURE where risk is seen as dependent on trust and an outcome's intrinsic cost. SECURE also supports trust delegation where a trustor's trust in a trustee is just a reference to the trust calculated by someone else.

When an interaction request is made to the SECURE framework, it is the responsibility of the request analyzer to decide on whether the request is to be allowed or denied. The request analyzer makes its decision based on information provided by the entity recognition component, trust calculator and risk evaluator. The entity recognition component is responsible for recognizing new or previously encountered entities. The

trust calculator performs trust calculation based on information it obtained from the trust lifecycle management component. The trust lifecycle management component is responsible for the formation and evolution of trust based on information stored in the evidence store. The risk analyzer uses the information provided by the risk configuration component to calculate the potential risk of the interaction request. The risk configuration component updates risk information based on the data stored in the evidence store. The evidence store holds all the trust and risk-related data. This includes the experiences of the trustor that can be obtained through monitoring and recommendations that are obtained through recommendation gathering.

3.1.4.2 Trust-Based Admission Control by Grey et al.

In [52], a trust framework is used to enable access control in collaborative ad hoc applications (e.g. blackjack game). To gain admission to an ad hoc application, the user first needs to specify the member role (e.g. dealer) it is interested in joining as. At the global level, an application manager will ask each existing member to vote on the admission. Based on the voting results, the application manager will either accept or reject the admission request. At the local level, each member uses its own trust-based policies to determine whether or not to support the admission. As for trust, it is calculated by the Trust Formation System based on a user's past interactions (experiences).

3.2 Middleware Support for Computational Trust Formation

There are many applications for trust in computing. Depending on the application, trust may be calculated differently. For example in P2P, trust is used to encourage cooperation. Cooperation however is rarely an issue with web services. With web services the main concern is whether the demanded QoS would be provided. Due to differences in concern, trust calculation is different between P2P and web services. As a concrete example, comparisons could be made between [55] and [5]. In [55] (Section 3.1.2.4), the authors use trust to measure a peer's level of participation in the network. Trust calculation therefore involves a peer's level of upload, download, resources shared and response rate to queries. In [5] (Section 3.1.3.4), trust in a web service is calculated using the trustor's quality of experience and the web service's quality of compliance.

Although there are many different ways to calculate trust, there are still some commonalities shared across the different trust calculation approaches. These commonalities can be abstracted into a middleware. Applications that need trust calculated therefore can leverage the services provided by the middleware in its trust calculation. Specifically, this thesis has identified three different areas of computational trust calculation that can benefit from middleware support. The areas are evidence gathering, belief calculation and emotional trust calculation.

To calculate cognitive trust, an application needs access to quality evidence. As for how evidence are to be collected and how evidence quality is to be determined, this is an issue that can be delegated to a middleware. Belief calculation can also be delegated to the middleware. This is based on the view that a trustor's beliefs about a trustee should be consistent across different applications. Cognitive trust calculation has therefore been simplified to a problem of selecting the relevant beliefs and determining how the beliefs are to be combined to form cognitive trust. In terms of emotional trust, emotions are associated with the trustor. It therefore makes sense for the middleware to keep track of the trustor's emotional trust so that it maintains consistent across different applications.

3.3 Survey of Trust Middleware

In this section, a literature survey is provided on trust middleware. As far as we know, there is only one paper on trust middleware. It is the Personalized Trust Framework (PTF) by Huynh in [58]. Note that Huynh's paper does not cover all the issues discussed in this thesis. Specifically, the paper does not presuppose the existence of a conceptual model nor concern itself with evidence gathering. PTF only addresses the challenge of trust being subjective and multidimensional. A description of PTF is provided below.

The Personalized Trust Framework (PTF) [58] is concerned with how a user's trust evaluation process can be captured and replicated by computers. The way PTF works is for a document and its meta-data to be submitted to the Trust Manager. A document in PTF refers to any piece of information that needs to be evaluated. As for meta-data, this covers information such as document type, context type, originator, etc. The Trust Manager matches the meta-data with the user's trust profile (preferences) to

determine the trust model to be used in trust calculation. An implementation of the trust model, i.e., a trust engine is then initialized to perform trust calculation. If a specific trust representation is required by the user, a converter could also be instantiated to perform the necessary trust transformation before the calculated trust value is returned to the user. Auditing is also supported by PTF. This allows a user to examine how trust is calculated.

Implementation wise, an ontology has been developed and forms the basic building blocks of PTF. It is also the language that is used to write the trust profiles. A trust profile contains the rules that are used for trust engine selection. It also contains information on the concepts to be used in trust calculation. Rules in PTF are executed by the Jena Rule Engine [103]. Both trust engines and converters can be written using Jena rules or implemented as Java classes.

3.4 Summary

Trust and reputation are used differently in different areas of computing. In the literature survey, the focus is on four areas: WWW, P2P networks, virtual communities and pervasive computing environments. Based on insights gained from the literature survey, a case is made for a middleware based approach for supporting computational trust formation. PTF as a trust middleware does not address all of the identified trust challenges.

Chapter 4

4 Computational Trust Architecture

The focus of this chapter is to provide a high level overview of the computational trust architecture. The individual components of the architecture are described in detail in the next two chapters. The rest of this chapter is divided into three sections. The first section describes the assumptions made when designing the computational trust architecture. Next, an overview of the architecture is provided. The final section of this chapter describes how information flows through the architecture.

4.1 Assumptions

The computational trust architecture is designed with the following assumptions in mind.

4.1.1 Identity and Type

Trustors, trustees and evidence sources all need to have unique identities. As explained in Section 2.3, identities need to be verifiable and long lived. How this can be achieved is outside the scope of this thesis. Some possible references to learn more about identity includes X.509 [57], PGP [146] and Sybil attack [36], [74]. It is assumed in this thesis that any identity used in the proposed architecture has already been verified and can be trusted. If this is not the case, identity can be treated as evidence. Belief in a trustee's identity therefore forms the basis for computational trust formation.

Trustors, trustees and evidence sources also need to be categorizable into their respective types. For example, a trustee could be a movie in the fantasy genre. A trustee could be a web service providing cloud computing. It is assumed that an ontology and taxonomy is in place for performing the categorization. How this can be achieved is outside the scope of this thesis. Some possible references include UDDI [29] and RosettaNet [105].

4.1.2 Computational Trust Formation

Computational trust formation is introduced in Section 1.2.5. The computational trust architecture currently only supports a limited form of computational trust formation. More support has been identified as future work. Basically, the architecture makes a number of formation assumptions. The assumptions are the following:

- *The relationship between cognitive trust and emotional trust is not explicitly supported by the architecture.* Capturing the impact that a trustor's emotions have on its beliefs in a trustee is a difficult problem. So is the capturing of how beliefs can impact a trustor's emotions. Further compounding the problem is the fact that the relationship is circular where emotions can influence beliefs which again can influence emotions. As a result, this relationship is currently not explicitly supported by the architecture. Instead, any changes in cognitive trust and emotional trust due to their relationship would necessitate the trustor making manual adjustments to cognitive trust and emotional trust.
- *Behavior trust is not explicitly represented in the architecture.* If a trustor observes other trustors' behaviors towards a trustee, such behavioral knowledge can be treated as evidence. The evidence can be used in cognitive trust calculation, thereby removing the need to explicitly represent behavior trust.

Figure 7 shows the resulting computational trust formation where computational trust is calculated from cognitive trust and emotional trust.

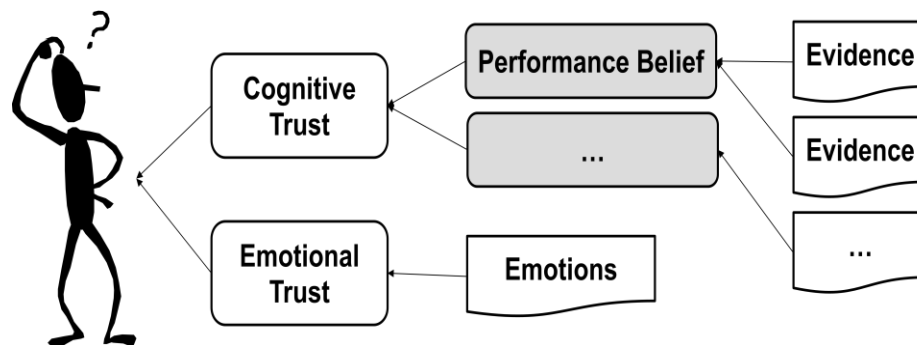


Figure 7: Computational Trust Formation (Architecture)

4.1.3 Evidence

There are many different types of evidence. The computational trust architecture currently supports four different evidence types:

- *Experience*. Experience is the knowledge that the trustor gained about the trustee. There are two ways that a trustor can gain experience with a trustee. One is to directly interact with the trustee. Another is to directly observe the trustee interacting with another trustor. In the first approach, monitoring is used to record the trustee's behaviors during an interaction. Besides monitoring, a trustor could also be asked for its view on a completed interaction. In the second approach, if a trustee is aware of the presence of observers, it may behave differently (e.g. perform better) than when it is not being observed. Therefore when calculating belief from experience, it is important to take into account the means used in experience gathering.
- *Recommendation*. When a trustor shares its experience with others, the shared experience is known as recommendation. All gathered recommendations should be filtered. This is to avoid using recommendations from biased or malicious recommenders in belief calculation.

An endorsement is a type of recommendation. When a movie advertises that it is recommended by movie critics, this can be seen as the critics endorsing a movie. With recommendation, the burden is on the trustor to discover the available recommendations. With endorsement, the burden is shifted to the trustee. It is the responsibility of the trustee to provide the trustor with all of its endorsements. As an endorsement could be neutral or negative, it is in the trustee's interest to throw away these non-positive endorsements. Therefore, endorsements are often biased towards the trustee. This can be seen in movie advertisements where all mentioned reviews are positive. Due to the inherent bias, care should be taken when using endorsements in belief calculation.

- *Reputation.* Reputation is the consensus assessment of a trustee by members of a social network. All gathered reputations should be filtered. This is to avoid using reputations from biased or malicious reputation systems in belief calculation.
- *Signal.* In economics, signaling [124] is used by a party to provide information about itself. The provided information can be used by the interacting parties to overcome the challenge of information asymmetry. In decision making, a trustee could send signals to the trustor explaining why it should be believed.

4.1.4 Belief

There are many different types of beliefs. To better organize the different beliefs, the computational trust architecture aggregates beliefs into *aggregate belief*. An aggregate belief is calculated from its constituent beliefs. The computational trust architecture hides the mappings from beliefs to aggregate belief from the applications that need computational trust calculated. By only exposing aggregate beliefs, this makes it easier for an application to specify the aggregate beliefs needed for cognitive trust calculation. The computational trust architecture currently supports eight different aggregate beliefs:

- *Accessibility belief.* This is belief in a trustee as being reachable when needed. Example constituent beliefs include availability belief, latency belief, etc.
- *Competence belief.* This is belief in a trustee as being qualified for what is expected of the trustee. Example constituent beliefs include qualification belief, popularity belief (i.e., inferring competence from the trustee being popular), etc.
- *Dependability belief.* This is belief in a trustee as being reliable and has the trustor's best interest in mind. A dependable trustee is one who would not betray the trustor. An example betrayal could be the violation of a signed service level agreement (SLA). Example constituent beliefs include compliance belief (i.e., level of compliance with SLA), popularity belief, etc.
- *Identity belief.* This is belief in a trustee as being who it says it is. Example constituent beliefs include personal knowledge belief (i.e., having knowledge that

only the trustee would know such as password), behavioral belief (i.e., behaving in a way that is similar to the trustee), etc.

- *Performance belief.* This is belief in a trustee as being able to carry out what is expected of the trustee. Example constituent beliefs include timeliness belief, response time belief, etc. Competence belief and performance belief are not the same. Competence is concern with capability. For example, having graduated from the Computer Science program is proof of competence. Performance is concern with usage of the capability. For example, being able to apply what is learned in the program to solve computing problems is proof of performance. A competent trustee could choose to perform well or to perform poorly. An incompetent trustee however by definition cannot perform well.
- *Privacy belief.* This is belief in a trustee as being in compliance with the trustor's policies with respect to the trustor's information. Example constituent beliefs include information retention belief, information sharing belief, etc.
- *Quality belief.* This is belief in the excellence of a trustee. Quality belief is used in cases when the trustee is an inanimate object. It is used in place of performance belief. For example, a trustor forms its belief in the quality of a novel. It does not form its belief in the performance of a novel. Example constituent beliefs include accuracy belief, bias belief (i.e., is the trustee bias?), etc.
- *Security belief.* This is belief in a trustee as being able to protect the interest of the trustor from harm. Example constituent beliefs include auditability belief, integrity belief (includes data and transactional integrity), etc.

There are many possible mappings from belief to aggregate belief. For example, a positive mapping would be availability belief that has a positive influence on the aggregate belief accessibility. A negative mapping would be bias belief that has a negative influence on the aggregate belief quality. A belief could also be a constituent belief in multiple aggregate beliefs. For example, popularity belief has a positive influence on both competence and dependability beliefs.

In terms of the scenarios, in movie selection, quality belief is already an aggregate belief; therefore cognitive trust formation remains the same as in Figure 3 of Section 1.3.1. As for web service selection, the cognitive trust formation needs to be updated with the incorporation of aggregate beliefs. The resulting cognitive trust formation is shown in Figure 8. In the figure, competence belief, performance belief and quality belief are accessible to the consumer. How these aggregate beliefs are formed (from constituent beliefs or from evidence directly) is hidden from the consumer.

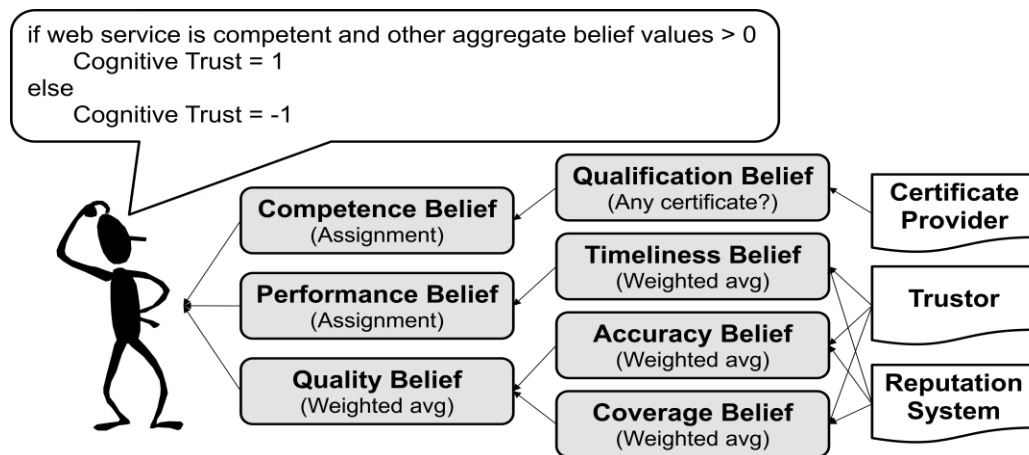


Figure 8: Cognitive Trust Formation from Aggregate Beliefs (Web Service)

4.1.5 Emotional Trust Formation

The computational trust architecture does not support the formation of emotional trust from the trustor's emotions directly. This is due to the difficulty in monitoring the trustor's emotions. Instead, emotional trust is formed through the recognition of the trustee. For example, based on the trustee's type and identity, the trustor may be predisposed to emotionally trust or distrust the trustee. An example is shown in Figure 9 where trustee types and identities are organized into a hierarchy. Each node in the hierarchy is assigned an emotional trust value. If there is no emotional trust value set for a trustee, the emotional trust value of its closest parent is used instead. For example, there is no emotional trust value set for the horror film "Zombieland" in Figure 9. Therefore, the trustor's emotional trust value in horror movies is used as an approximation.

Another example is to create a hierarchy based on the trustee's relationship with the trustor. Example relationships could include family, friend, acquaintance and stranger. Multiple hierarchies can be used together to form the trustor's emotional trust in a trustee. An example of the use of multiple hierarchies is for emotional trust to be formed based on the sum of the emotional trust from each hierarchy. Another example is for emotional trust to be based on the maximum emotional trust value obtained from all the hierarchy.

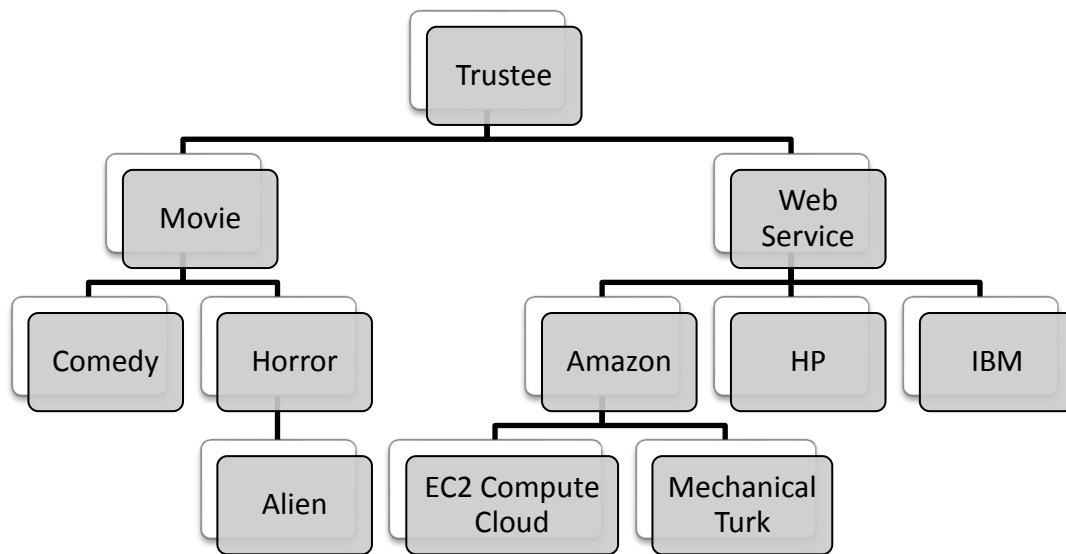


Figure 9: Hierarchy of Trustee Type and Identity

4.1.6 Factors Influencing Computational Trust Formation

There are many factors that influence computational trust formation. The computational trust architecture grouped the factors into decision, trustor and trustee.

4.1.6.1 Decision Factors

Decision factors are factors associated with a decision. An example is decision type where the trustor calculates computational trust differently for movie selection and web service selection. Another example is the importance of a decision where more evidence may be demanded if a decision is important. Table 1 shows an example of how trust types can influence computational trust formation. Other example decision factors include decision cost, decision environment, etc.

Table 1: Trust Types and Computational Trust Formation

Types of Trust	Computational Trust Formation
Faith	$compTrust = emoTrust$
Emotion	$compTrust = 0.7 \cdot emoTrust + 0.3 \cdot cogTrust$
Cognition and Emotion	$compTrust = 0.5 \cdot emoTrust + 0.5 \cdot cogTrust$
Cognition	$compTrust = 0.3 \cdot emoTrust + 0.7 \cdot cogTrust$
Rational	$compTrust = cogTrust$

4.1.6.2 Trustor Factors

Trustor factors are factors associated with the trustor. A change in a trustor factor may influence computational trust formation of many decisions. Trustor factors include:

- *Trust Disposition.* A trustor could be an optimist, a realist or a pessimist in its trust disposition [85]. An optimist may demand less evidence in belief calculation while the opposite is the case for a pessimist. A realist's evidence demands falls somewhere in between an optimist and a pessimist.
- *Trust Preference.* A trustor's trust preferences could influence the algorithms used for computational trust calculation (e.g. Bayesian vs. average). It could also influence how evidence are gathered (e.g. prefer one evidence source over another) and how beliefs in a trustee are formed (e.g. Bayesian vs. average).
- *Culture.* The culture upbringing of a trustor could influence computational trust formation. For example, the presence of credit card symbols had been shown to have a bigger impact on trust in Latin American and Brazil than in the US [131]. Basically, different cultures may interpret the same evidence (a credit card signal in this case) differently. Moreover, different cultures may instill in a trustor different norms and values. For example, business relationships in Japan are more personal than in more "legalistic" cultures such as the US [17]. Trustors living in cultures with a high degree of power inequality (high power distance) or cultures that are less tolerant of change (high uncertainty avoidance) have been shown to have a higher use of evidence from personal sources (e.g. friends) and a lower use of evidence from impersonal sources (e.g. Consumer Reports) [34].

4.1.6.3 Trustee Factors

Trustee factors are factors associated with the trustee. An example is the identity of the trustee where computational trust is calculated differently depending on whether the trustor recognizes the trustee. If a trustee is recognizable, it can be calculated using emotion. Otherwise, it is calculated using cognition. Other trustee factors include trustee type, trustee's relationship with the trustor, etc.

4.1.7 Privacy

As explained in Section 2.3, privacy plays an important role in computational trust formation. Due to the complexity of the issues involved, privacy is considered outside the scope of this thesis. Some possible references to learn more about privacy includes [13], [18], [49] and [113]. This is an important area that should be addressed in future work.

4.1.8 Architecture Deployment

Computational trust is subjective as detailed in Section 1.2.3.2. To accommodate the trustor's subjective views, it is assumed that each trustor has its own computational trust architecture deployment.

4.1.9 Summary

A summary of all the discussed assumptions is shown in Table 2.

Table 2: Assumptions of Computational Trust Architecture

	Assumptions
Identity and Type	<ul style="list-style-type: none"> • Identity is unique and long lived • Identity has already been verified through means outside the scope of this thesis • Every trustor, trustee and evidence source has a type property whose assignment is outside the scope of this thesis
Computational trust formation	<ul style="list-style-type: none"> • Computational trust is calculated from cognitive trust and emotional trust • Behavioral trust is treated as a form of cognitive trust • The relationship between cognitive trust and emotional trust is not represented in the proposed architecture
Evidence	<ul style="list-style-type: none"> • Evidence types: Experience, recommendation, reputation and signal

Belief	<ul style="list-style-type: none"> • Aggregate belief is calculated from constituent beliefs • Aggregate beliefs: Accessibility, competence, dependability identity, performance, privacy, quality and security
Emotional trust formation	<ul style="list-style-type: none"> • Emotional trust is calculated from properties of a trustee • The properties can be organized into a hierarchy
Factors influencing computational trust formation	<ul style="list-style-type: none"> • Decision factors • Trustor factors: Disposition, preference and culture • Trustee factors
Privacy	<ul style="list-style-type: none"> • Outside the scope of this thesis
Architecture deployment	<ul style="list-style-type: none"> • Each trustor deploys its own architecture

4.2 Overview of Architecture

A graphical illustration of the computational trust architecture is shown in Figure 10. The architecture consists of three components: SCOUT (Services supporting COMpUtation of Trust), Evidence Repository and Trust Calculator. SCOUT is a middleware that provides three different web services: Evidence Gathering Service (EGS), Belief Formation Service (BFS) and Emotional Trust Service (ETS). EGS is responsible for the discovery and gathering of evidence. Any gathered evidence is normalized (i.e., mapped to standard evidence representation) before being stored in the Evidence Repository. The Evidence Repository is the storage for all the gathered evidence. BFS uses the evidence stored in the Evidence Repository for belief and aggregate belief calculation. ETS is responsible for calculating the trustor's emotional trust in a trustee. An application could contact EGS directly for evidence gathering. It could also contact BFS or ETS to obtain the trustor's aggregate belief or emotional trust. An application could also subscribe to BFS or ETS to be informed of any changes to the trustor's aggregate belief or emotional trust.

The Trust Calculator is a client that can be used to access the SCOUT services. It calculates computational trust based on the belief values and emotional trust values calculated by SCOUT. An application could either query or subscribe to the computational trust calculated by the Trust Calculator.

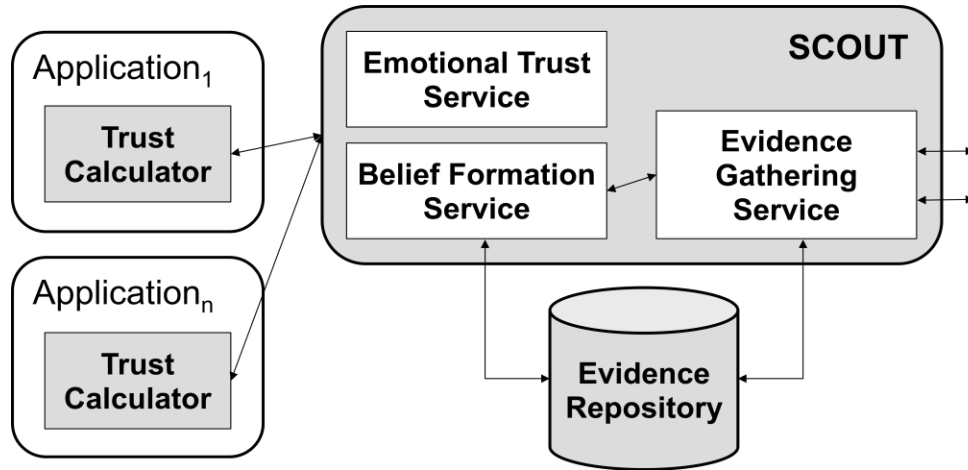


Figure 10: Computational Trust Architecture

4.3 Information Flow of Architecture

The computational trust architecture can be viewed from the perspective of the flow of information. Such a perspective is graphically illustrated in Figure 11. In the figure, computational trust formation is described as a three step process. The process starts with a request for a computational trust calculation. This results in evidence gathering. Evidence may be found locally. For example by parsing the log file of the trustor's interaction with a trustee. Evidence may also be found in the open environments. For example by requesting reputation from an Internet-based reputation system. In both cases, the gathered evidence are normalized before being stored in an Evidence Repository. The next step is the calculation of the trustor's beliefs and aggregate beliefs in a trustee based on the stored evidence. Finally, the last step is to perform the trust calculation. This involves the calculation of cognitive trust from aggregate beliefs and emotional trust from the trustor's knowledge about the trustee. Computational trust is calculated from cognitive and emotional trust. There are many different algorithms that can be used in belief, aggregate belief and trust calculations. The different algorithms used in the literature are summarized in Table 3.

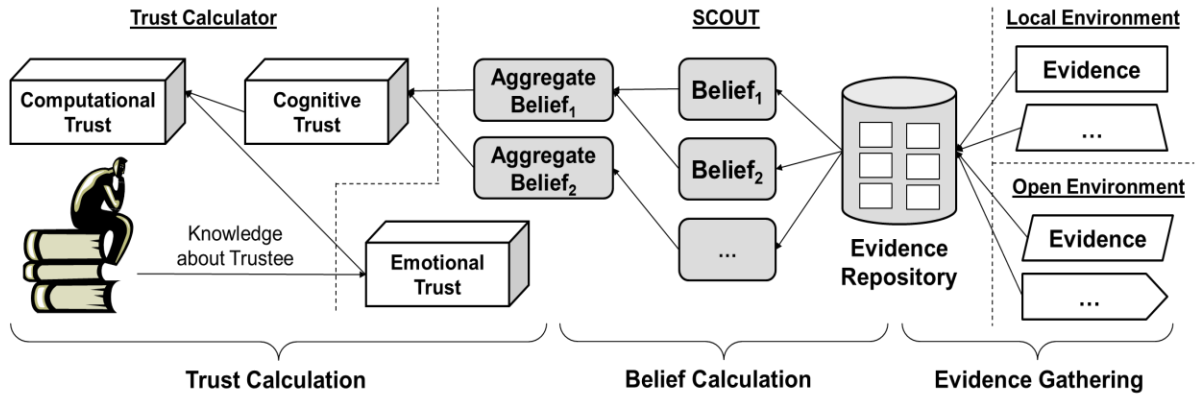


Figure 11: Information Flow of Architecture

Table 3: Algorithms for Belief, Aggregate Belief and Trust Calculation

Categories	Examples
Arithmetic	<ul style="list-style-type: none"> • Summation: [38], [55]... • Average: [7], [14]... • Weighted average: [11], [39], [91], [135]... • Others: [23], [53], [71], [84]...
Probability Theory	<ul style="list-style-type: none"> • Bayesian approach: [20], [43], [62], [80]... • Belief theory: [64], [142]... • Others: [2], [114]...
Fuzzy Logic	<ul style="list-style-type: none"> • [5], [41], [54], [145]...
Others	<ul style="list-style-type: none"> • [59], [68], [110], [117]...

4.4 Summary

The computational trust architecture consists of a SCOUT middleware, an Evidence Repository and Trust Calculators that are used by applications to perform computational trust calculation. When designing the architecture, a number of assumptions were made. The assumptions include identity, computational trust formation, evidence, belief, emotional trust formation, factors influencing computational trust formation, privacy and architecture deployment. To provide a different perspective on how the computational trust architecture operates, the chapter also describes how information flows through the architecture. The information flow helps illustrate how information changes as it passes through the computational trust architecture.

Chapter 5

5 SCOUT

The focus of this chapter is on introducing the SCOUT middleware. This chapter is divided into four sections. The first section provides a high level overview of the middleware design. The next three sections focus on the introduction of each of the SCOUT services in detail.

5.1 SCOUT Design

SCOUT is designed with the following properties in mind:

- *Modularity*. SCOUT is designed to support evidence gathering, belief calculation and emotional trust calculation. To achieve modularity, all three functionalities are implemented as web services that can operate independently of each other.
- *Extensibility*. There are many algorithms and protocols that can be used to implement the different SCOUT functionalities. To achieve extensibility, a plug-in approach has been adopted for cases when there is no best way to implement a specific functionality.
- *Adaptability*. Computational trust is subjective and multidimensional. The SCOUT services therefore should be adaptable to meet the needs of computational trust formation. To achieve this, a policy based approach has been adopted. A policy refers to “a rule that defines a choice in the behavior of a system” [33]. Policies are used in SCOUT to determine how each service responds to the trustor’s queries or subscriptions. For example, a policy could specify that a specific algorithm is always to be used in calculating quality belief in movies as this is the trustor’s preferences. Another example could be that if a decision is important, this may entail the gathering of more evidence, the consideration of more evidence types and the use of a higher threshold for evidence filtering. All these requirements can be presented as policies for important decisions. As every trustor is different, policies are selected as it is

challenging to program for adaptability ahead of time. Moreover, a trustor may change over time. Policies allow for any change to be captured without requiring the coding and compilation of the SCOUT services.

5.2 Evidence Gathering Service

The Evidence Gathering Service (EGS) is responsible for gathering the evidence needed for belief calculation. A graphical illustration of EGS is shown in Figure 12. In the figure, the Evidence Gathering Manager is responsible for processing any request for evidence gathering. This is accomplished through the exposed *gatherEvidence* method that has as input the following three parameters:

- *Trustee*. Information about the trustee can be represented as attribute-value pairs. There are two attribute-value pairs that must be present for each trustee: the trustee's identity and the trustee's type. For example, the online retailer Amazon has an identity of "www.amazon.com" and type of "OnlineRetailer". Amazon's EC2 Service has an identity of "aws.amazon.com/ec2/" and type of "WebService/ComputeCloud". In this case, since EC2 is both a web service and a compute cloud, both are specified with the more general type specified first and the types separated by a slash. Other attributes could include the trustee's relationship with the trustor, signals that the trustee is willing to provide, etc.
- *Belief*. Evidence could be gathered for aggregate belief calculation or belief calculation. As aggregate belief is just a special type of belief, EGS does not differentiate between aggregate belief and belief. Instead, both are treated the same when it comes to evidence gathering.
- *Hints*. This is an optional parameter that is represented as attribute-value pairs. It is used to provide hints to EGS with regards to how evidence gathering should be changed. Some example hints may include the importance of decision, the evidence types to be used, etc.

The input parameters are used in policies to determine how evidence are to be gathered. This may involve the invocation of one or more Evidence Gatherers for evidence

discovery and gathering. The gathered evidence are then handed over to the Evidence Handler for processing. After processing, the last step is for the evidence to be stored in the Evidence Repository.

As for evidence filtering, EGS has adopted the trust-based approach (see evidence filtering bullet point in Section 1.4 for a review). This is due to the fact that the outlier-approach depends on the gathered evidence exhibiting certain statistical properties [63]. The assumption that outliers should be filtered out is not always a valid assumption. A trustor may have views that are more similar to the outliers than the majority of evidence. In the trust-based approach, trust is used for identifying and avoiding of distrusted evidence sources. To calculate evidence source trust, EGS first needs to identify whether an evidence source can provide evidence of high quality. Feedbacks provided by evidence gathering requesters can be used for assessing the quality of evidence. Feedbacks can be provided to EGS by calling the *provideFeedback* method of the Evidence Gathering Manager. This method takes two input parameters: *trustee* and *belief feedbacks*. The two input parameters are used in policies to determine the Evidence Source Assessor to be invoked for assessing evidence quality and for calculating evidence source trust. The calculated evidence source trust can be taken into account during evidence gathering. For the rest of this section, each of the components of EGS is described in detail.

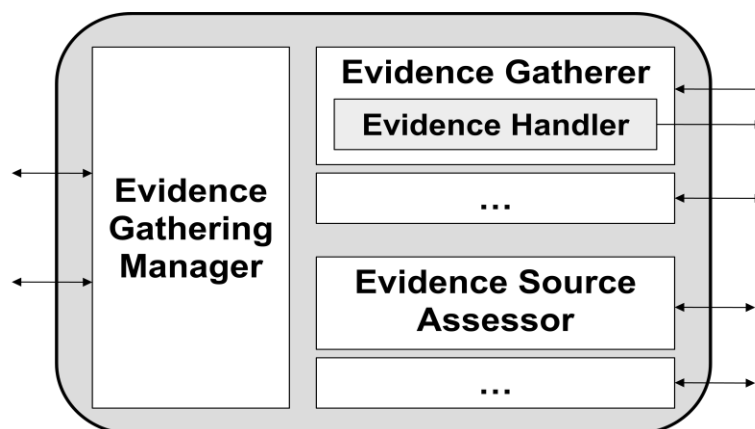


Figure 12: Evidence Gathering Service

5.2.1 Evidence Gatherer

There is currently no standard for evidence discovery and evidence gathering. As a result, to gather evidence from multiple evidence sources, different discovery and gathering protocols have to be implemented. Moreover, not all evidence sources provide evidence in formats that are machine friendly. For example Rotten Tomatoes does not provide an API for accessing the reputation of movies. The only way to access a movie's reputation is to access the movie page and parse the html file to extract the movie's reputation. This in turn makes evidence gathering challenging.

In EGS, each Evidence Gatherer is responsible for one evidence discovery and gathering protocol. An Evidence Gatherer may be responsible for a single evidence source (e.g. IMDb) or multiple evidence sources (e.g. locating reputation systems through a registry). Evidence Gatherers are implemented as plug-ins to EGS. This allows an Evidence Gatherer to be implemented by a third party (e.g. the evidence source) and used in multiple SCOUT deployments. This in turn reduces the burden on application developers as they no longer have to write and support different evidence discovery and gathering protocols.

Finally, all Evidence Gatherers deployed in EGS are registered with a registry. The registry is used by the Evidence Gathering Manager (Section 5.2.4) to discover the available Evidence Gatherers for evidence gathering.

5.2.2 Evidence Handler

In EGS, Evidence Handler is responsible for mapping the gathered evidence to a standard representation. For example, all evidence could be mapped to an interval of $[-1, 1]$ where 1 is the best and -1 is the worst. By having a single standard evidence representation, this simplifies evidence interpretation. No longer would an evidence user have to worry about what a piece of evidence is representing.

Besides evidence mapping, the Evidence Handler is also responsible for the metadata needed for evidence interpretation. The following metadata are supported by the Evidence Handler: timestamp of evidence creation, timestamp of evidence gathered,

identity of evidence source, identity of trustee and belief. The metadata could be provided by the evidence source (e.g. timestamp of evidence creation), generated by the Evidence Handler itself (e.g. timestamp of evidence gathered) or obtained from the Evidence Gatherer (e.g. identity of evidence source, identity of trustee and belief). The mapped evidence along with their metadata are stored into the Evidence Repository by the Evidence Handler.

Evidence Handlers are implemented as plug-ins to EGS. This allows EGS to be extended to support new evidence representations. As with Evidence Gatherers, all deployed Evidence Handlers are registered with a registry. The registry is used by the Evidence Gatherers to discover the available Evidence Handlers. The selection of an Evidence Handler is based on evidence type and evidence representation.

5.2.3 Evidence Source Assessor

There are many ways in which an evidence source can be assessed. For example, evidence source trust¹⁹ in [30] is represented as a triplet: (*servent_id*, *num_agree*, *num_disagree*). For each servent²⁰, the trustor records the number of times the servent's recommendations agrees or disagrees with the trustor's feedback. In a recommendation²¹, a servent either votes for or against another servent as a content provider. After a successful download, any servent that voted for the content provider would have its *num_agree* increase by one. Those servents that voted against the content provider would have their *num_disagree* increase by one. If a download failed, any servent that voted for the content provider would have its *num_disagree* increase by one. Those servents that voted against the content provider would have their *num_agree* increase by one. Based on the calculated evidence source trust (i.e., *num_agree* vs. *num_disagree*), a decision can now be made on whether to contact a servent for recommendation. Other evidence source

¹⁹ Known as “credibility” in [30].

²⁰ A servent is a node in a P2P network. It is both a trustee and an evidence source.

²¹ Known as “vote” in [30].

assessment algorithms include [1], [11], [80] and [142]. In this thesis, an evidence source assessment algorithm is introduced in Section 7.2.

In EGS, each Evidence Source Assessor is responsible for implementing its own evidence source assessment algorithm. The Evidence Source Assessors are implemented as plug-ins. This allows EGS to be easily extended to support new evidence source assessment algorithms. The Evidence Source Assessors are also provided with access to a scheduler. The scheduler can be used to schedule when trust in an evidence source is to be reevaluated. This is to provide an evidence source with enough time to prove its trustworthiness as oppose to having evidence source trust calculated after every feedback. As with Evidence Gatherers, all deployed Evidence Source Assessors are registered with a registry. The registry is used by the Evidence Gathering Manager (Section 5.2.4) to discover the available Evidence Source Assessors.

5.2.4 Evidence Gathering Manager

The Evidence Gathering Manager (EGM) employs policies to determine how evidence is to be gathered and how evidence sources are to be assessed. It is the responsibility of the trustor to deploy the needed policies. Policies are in the form of **if condition then action** and are the focus for the rest of this section.

5.2.4.1 Evidence Gathering Policy

The Evidence Gathering Manager uses evidence gathering policies (EG-Policies) to determine how the evidence needed for belief calculation is to be gathered. The syntax for EG-Policies is discussed in Appendix A. An EG-Policy may state that specific evidence gatherers are to be used for evidence gathering. An example is shown in Figure 13. In the figure, “Trustee”, “Belief” and “Hint” are obtained through the *gatherEvidence* method. If it is hinted at that the decision is of low importance, evidence gatherers with identity of “EG1” (lines 1-7) and “EG2” (lines 9-16) are invoked for evidence gathering. For decision of high importance, “EG1” (lines 1-7) and “EG3” (lines 18-25) are invoked instead. A possible reason for the EG-Policies in Figure 13 is that some evidence sources may be more costly than others. A trustor may only be willing to use the costlier evidence sources (i.e., invoking “EG3”) if it is hinted at that the decision is of high importance.

```

1  when
2    trustee: Trustee( type == "WebService/Type1" )
3    belief: Belief( type == "Timeliness" )
4  then
5    gatherer = Registry.lookup("EG1");
6    gatherer.gatherEvidence(trustee, belief);
7  end
8
9  when
10   trustee: Trustee( type == "WebService/Type1" )
11   belief: Belief( type == "Timeliness" )
12   Hint( name == "Importance" && value == "Low" )
13 then
14   gatherer = Registry.lookup("EG2");
15   gatherer.gatherEvidence(trustee, belief);
16 end
17
18 when
19   trustee: Trustee( type == "WebService/Type1" )
20   belief: Belief( type == "Timeliness" )
21   Hint( name == "Importance" && value == "High" )
22 then
23   gatherer = Registry.lookup("EG3");
24   gatherer.gatherEvidence(trustee, belief);
25 end

```

Figure 13: EG-Policies (Identity)

It is not always the case that a trustor would know in advance which evidence sources should be contacted for evidence gathering. In such cases, the better approach is to let EGM handle the Evidence Gatherers to be invoked. Such an example is shown in Figure 14. In the figure, the registry is used to look up all the Evidence Gatherers with properties that satisfy the trustor's evidence gathering requirements (line 8). Since no evidence type is specified, the policy assumes that reputation is to be gathered. A different approach would be to throw an exception if evidence type is not specified. After the Evidence Gatherers are found, they are invoked for evidence gathering. An advantage of this property-based approach is that if a new evidence gatherer is deployed, the evidence gatherer would be included in evidence gathering if it has the necessary properties. This is not the case with the identity-based approach where the EG-Policies need to be updated before the new Evidence Gatherer can be used in evidence gathering.

```

1  when
2    trustee: Trustee( type == "WebService/Type1" )
3    belief: Belief( type == "Timeliness" )
4  then
5    properties = [ "TrusteeType" : trustee.getType(),
6                  "Belief" : belief.getType(),
7                  "EvidenceType" : "Reputation" ];
8    gatherers = Registry.lookup(properties);
9
10   for (gatherer: gatherers)
11     gatherer.gatherEvidence(trustee, belief);
12 end

```

Figure 14: EG-Policy (Properties)

```

1  when
2    trustee: Trustee( type == "WebService/Type1" )
3    belief: Belief( type == "Timeliness" )
4  then
5    properties = [ "TrusteeType" : trustee.getType(),
6                  "Belief" : belief.getType(),
7                  "EvidenceType" : "Reputation" ];
8    gatherers = Registry.lookup(properties);
9
10   sortByEvidenceourceTrust(gatherers);
11
12   for (int i=0; i<3; i++) {
13     gatherer = gatherers.get(i);
14
15     if (gatherer.getEvidenceourceTrust() > 0)
16       gatherer.gatherEvidence(trustee, belief);
17     else
18       break;
19   }
20 end

```

Figure 15: EG-Policy (Evidence Source Trust)

A weakness of the EG-Policies presented in Figure 13 and Figure 14 is that the policies do not take evidence source trust into account during evidence gathering. The policies implicitly assume that all the evidence sources can be trusted. This is often not a valid assumption. The use of evidence source trust in evidence gathering is demonstrated in Figure 15. In the figure, it is assumed that each Evidence Gatherer has access to a single evidence source. As such, the Evidence Gatherers can be sorted based on evidence source trust (line 10). After sorting, the three Evidence Gatherers that have

access to the most trusted evidence sources are invoked for evidence gathering. The invocation is under the condition that an Evidence Gatherer may be skipped if the corresponding evidence source trust is less than or equal to zero.

As seen in Figure 15, the EG-Policy turns out to be rather complex. This is not considering the case of an Evidence Gatherer having access to multiple evidence sources. To simplify the use of evidence source trust in an EG-Policy, an abstraction known as *strategy* has been implemented. A strategy is an encapsulation of all the steps that need to be taken to perform evidence gathering. EGM currently supports two strategies: the broadcast strategy and the evidence source trust strategy. The broadcast strategy involves the invocation of all matched Evidence Gatherers in evidence gathering. An example is shown in Figure 16. The example is functionally the same as the EG-Policy shown in Figure 14. The evidence source trust strategy performs evidence gathering based on evidence source trust. An example is shown in Figure 17. The example is functionally the same as the EG-Policy shown in Figure 15. The only difference is that the example actually works with Evidence Gatherers that have access to multiple evidence sources.

```

1  when
2    trustee: Trustee( type == "WebService/Type1" )
3    belief: Belief( type == "Timeliness" )
4  then
5    strategy = new BroadcastStrategy();
6    strategy.set("EvidenceType", "Reputation");
7    strategy.execute(trustee, belief);
8  end

```

Figure 16: EG-Policy (Broadcast Strategy)

```

1  when
2    trustee: Trustee( type == "WebService/Type1" )
3    belief: Belief( type == "Timeliness" )
4  then
5    strategy = new EvidenceSourceTrustStrategy();
6    strategy.set("EvidenceType", "Reputation");
7    strategy.set("NumOfEvidence", 3);
8    strategy.set("EvidenceSourceTrustThreshold", 0);
9    strategy.execute(trustee, belief);
10 end

```

Figure 17: EG-Policy (Evidence Source Trust Strategy)

5.2.4.2 Evidence Source Assessment Policy

The Evidence Gathering Manager uses evidence source assessment policies (EA-Policies) to determine the Evidence Source Assessor that is responsible for processing the incoming feedback. The syntax for EA-Policies is discussed in Appendix A. An example EA-Policy is shown in Figure 18. In the figure, an Evidence Source Assessor is selected based on its identity “EA1”. Next, the assessor is configured using the *set* method (lines 6-7). For example, an “EvidenceWindow” of 60 restricts assessment to evidence that have been created in the last 60 minutes. If creation timestamp is unavailable, the timestamp associated with evidence gathering is used instead. There is no point in assessing outdated evidence. After the completion of configuration, the assessor is invoked to calculate evidence source trust. Depending on the evidence source assessment algorithm implemented, an Evidence Source Assessor may choose to not calculate evidence source trust right away. Instead, a scheduler can be used by the assessor to schedule evidence source trust to be calculated periodically.

```

1  when
2    trustee: Trustee( type == "WebService/Type1" )
3    feedback: BeliefFeedback( type == "Timeliness" )
4  then
5    assessor = Registry.lookup("EA1");
6    assessor.set("EvidenceWindow", 60);
7    ...
8    assessor.assessEvidenceource(trustee, feedback);
9  end

```

Figure 18: EA-Policy

5.3 Belief Formation Service

The Belief Formation Service (BFS) is responsible for calculating the trustor’s aggregate beliefs from the available evidence. A graphical illustration of BFS is shown in Figure 19. In the figure, the Belief Formation Manager is responsible for processing any request for the trustor’s aggregate belief. This is accomplished through the exposed *getAggregateBelief* method that has as input three parameters: *trustee*, *aggregate belief* and *hints*. These input parameters are used in policies to determine how aggregate belief

is to be calculated. This may involve the invocation of one or more Belief Engines to perform belief calculation.

A trustor can also subscribe to the aggregate beliefs calculated by BFS. This is accomplished through *subscribeAggregateBelief* method exposed by the Belief Formation Manager. The method has the same input parameters as *getAggregateBelief*. The return value of the method is a unique identifier: *subscriptionId*. The *subscriptionId* identifier is used in the *unsubscribeAggregateBelief* method to unsubscribe from an existing subscription. Aggregate belief subscription is the responsibility of the Subscription Manager.

BFS also exposes a *provideFeedback* method that has as input three parameters: *trustee*, *aggregate belief feedbacks* and *hints*. The input parameters are used in policies to determine how an aggregate belief feedback can be mapped to constituent belief feedbacks. The *provideFeedback* method of EGS is then invoked with the *trustee* and *belief feedbacks*. For the rest of this section, each of the components of BFS is described in detail.

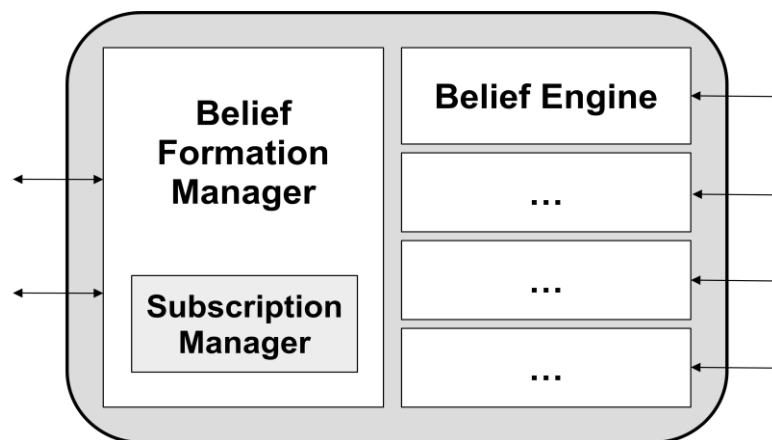


Figure 19: Belief Formation Service

5.3.1 Belief Engine

There are many ways to calculate a trustor's belief in a trustee (Table 3 of Section 4.3). In BFS, belief calculation is the responsibility of the Belief Engines. Basically, each Belief Engine is responsible for implementing its own belief formation algorithm. Evidence

needed for belief calculation can be retrieved from the Evidence Repository. If the evidence available in the repository are not enough for belief calculation to proceed, EGS could be invoked to perform additional evidence gathering.

Belief Engines are implemented as plug-ins to BFS. This allows BFS to be extended to support new belief formation algorithms. All Belief Engines deployed in BFS are registered with a registry. The registry is used by the Belief Formation Manager (Section 5.3.2) to discover the available Belief Engines for belief calculation.

5.3.2 Belief Formation Manager

The Belief Formation Manager (BFM) employs policies to determine how aggregate belief in a trustee is calculated and how feedback to aggregate belief is mapped to feedback to constituent beliefs. It is the responsibility of the trustor to deploy the needed policies. The rest of this section is divided into three parts. The first part introduces the belief formation policies. The second part describes aggregate belief subscription. The last part introduces the aggregate belief feedback policies.

5.3.2.1 Belief Formation Policy

The Belief Formation Manager uses belief formation policies (BF-Policies) to determine how beliefs and aggregate beliefs are to be calculated. The syntax for BF-Policies is discussed in Appendix A. An example BF-Policy is shown in Figure 20. In the figure, “Trustee”, “AggregateBelief” and “Hint” are obtained through either the *getAggregateBelief* method or *subscribeAggregateBelief* method. The trustor’s performance belief in a web service is calculated based on the trustor’s experiences. To calculate performance belief, its constituent belief timeliness needs to be calculated (see Figure 8 in Section 4.1.4 for reference). To calculate timeliness belief, a Belief Engine identified as “BE1” is selected (line 6). The belief formation algorithm implemented by the Belief Engine is then configured (lines 7-8) based on the trustor’s preferences. The configuration step is optional as a trustor may be satisfied with the defaults of the Belief Engine. Documentation associated with the Belief Engine should detail the defaults of the belief formation algorithm along with what can and cannot be configured. The last step is to invoke the Belief Engine (line 9) to calculate timeliness belief. The calculated belief

value is returned as the belief value of aggregate belief performance. If an aggregate belief has multiple constituent beliefs, the constituent beliefs can all be calculated by different Belief Engines. The constituent beliefs can also be calculated by the same Belief Engine invoked multiple times.

```

1  when
2    trustee: Trustee( type == "WebService/Type1" )
3    aggBelief: AggregateBelief( type == "Performance" )
4    hint: Hint( name == "EvidenceType" && value == "Experience" )
5  then
6    beliefEngine = Registry.lookup("BE1");
7    beliefEngine.set("ExperienceWindow", 30);
8    ...
9    beliefEngine.calculateBelief(trustee, new Belief("Timeliness"));
10 end

```

Figure 20: BF-Policy

A different example could be to employ different formation algorithms based on the trustor's trust disposition. A pessimistic trustor could invoke multiple Belief Engines and take the minimum of the calculated beliefs as its belief in the trustee. For an optimistic trustor, the maximum belief value could be selected. For a realistic trustor, its belief in a trustee could be based on the average of the beliefs calculated by the Belief Engines.

5.3.2.2 Aggregate Belief Subscription

Upon invocation of the *subscribeAggregateBelief* method, the EGM creates a new subscription and adds it to the Subscription Manager. Periodically (as defined by the trustor), the Subscription Manager examines each of its subscriptions to determine whether it is time for an aggregate belief to be reevaluated. If so, the aggregate belief is calculated based on the deployed BF-Policies and returned to the subscriber. When a subscription is no longer needed by the subscriber (i.e., invocation of *unsubscribeAggregateBelief* method), EGM deletes the subscription from the Subscription Manager. As a subscriber may not be interested in all the changes in the calculated aggregated belief, the Subscription Manager supports the registration of a filter to filter out those aggregate belief changes the trustor is not interested in knowing. The

syntax for aggregate belief filter is discussed in Appendix A. An aggregate belief filter could be specified as a hint with the name of “AggregateBeliefFilter”. Several example types of filters supported by the Subscription Manager are shown in Table 4.

Table 4: Example Aggregate Belief Filters

Aggregate Belief Filter	Description
aggBelief.belief < 0	Only notify subscriber if aggregate belief has dropped below zero
aggBelief.belief != lastAggBelief.belief	Only notify subscriber if aggregate belief has changed from last published aggregate belief
Math.abs(aggBelief.belief – lastAggBelief.belief) > 0.2	Only notify subscriber if the change in aggregate belief from the last published aggregate belief is greater than 0.2

5.3.2.3 Aggregate Belief Feedback Policy

The Belief Formation Manager uses aggregate belief feedback policies (AF-Policies) to determine how feedback to aggregate belief is mapped to feedback to beliefs. The syntax for AF-Policies is discussed in Appendix A. An example AF-Policy is shown in Figure 21. In the figure, feedback to aggregate belief performance is mapped to feedback to timeliness belief. Since there is only one constituent belief, timeliness belief is simply assigned the timestamp and feedback of performance belief. As EGS expects a list of belief feedbacks as input, the square brackets ([]) are used to create a list. The *provideFeedback* method of EGS is then invoked with the belief feedbacks (line 8).

```

1  when
2    trustee: Trustee( type == "WebService/Type1" )
3    aggFeedback: AggregateBeliefFeedback( type == "Performance" )
4  then
5    feedback = new BeliefFeedback("Timeliness",
6                                  aggFeedback.getTimestamp(),
7                                  aggFeedback.getFeedback());
8    EGS.provideFeedback(trustee, [feedback]);
9  end

```

Figure 21: AF-Policy

If an aggregate belief is formed from multiple constituent beliefs, then more complex mappings are needed. For example, a quality belief feedback of 1.0 can be

mapped to feedback of 1.0 to accuracy belief and coverage belief. A quality belief feedback of 0.5 can be mapped to feedback of 0.75 for accuracy belief but feedback of 0.25 for coverage belief. The mappings are based on the idea of assigning meaning to the provided aggregate belief feedback. The mappings can be defined as different AF-Policies. For example, there is an AF-Policy for when quality belief feedback is 1.0 and another for when quality belief feedback is 0.5.

5.4 Emotional Trust Service

The Emotional Trust Service (ETS) is responsible for calculating the trustor's emotional trust in a trustee. A graphical illustration of ETS is shown in Figure 22. In the figure, the Emotional Trust Manager (ETM) is responsible for processing any request for the trustor's emotional trust. This is accomplished through the exposed *getEmotionalTrust* method that has as input one parameter: *trustee*. The input parameter is used in policies to determine the trustor's emotional trust in the trustee. For emotional trust subscription, ETM exposes two methods: *subscribeEmotionalTrust* and *unsubscribeEmotionalTrust*. Both methods function in the exact same way as in the case of BFS. The syntax for emotional trust filter is discussed in Appendix A.

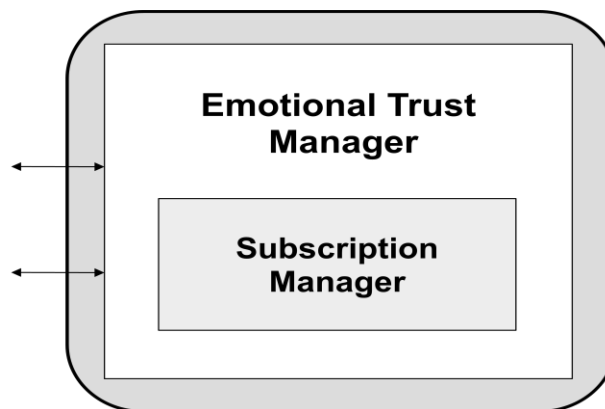


Figure 22: Emotional Trust Service

Emotional trust formation is based on the establishment of a hierarchy (Section 4.1.5). The hierarchy can be constructed using emotional trust policies (ET-Policies). The syntax for ET-Policies is discussed in Appendix A. An example of ET-Policies is shown in Figure 23. In the figure, a hierarchy has been established for horror movies. By using

“salience”, the trustor can prioritize which policy should fire when there are multiple conditional matches. For example, if the trustee is the horror movie “Zombieland”, then the first three policies (lines 1-6, 8-13 and 15-20) in Figure 23 could fire. This is due to the fact that the first policy (lines 1-6) matches any trustee; the second policy (lines 8-13) matches any trustee that is of type movie while the third policy (lines 15-20) matches any trustee that is of type horror movie. As only one policy can fire, the policy with the highest “salience” would fire, thereby returning the emotional trust value of 0.8. ETS currently only supports one hierarchy. Support for multiple hierarchies will be part of our future work.

```

1  salience 0
2  when
3    trustee: Trustee( )
4  then
5    0
6  end
7
8  salience 10
9  when
10   trustee: Trustee( type matches "Movie.*" )
11 then
12   0.5
13 end
14
15 salience 20
16 when
17   trustee: Trustee( type matches "Movie/Horror.*" )
18 then
19   0.8
20 end
21
22 salience 30
23 when
24   trustee: Trustee( type matches "Movie/Horror.*" && id == "Alien" )
25 then
26   1.0
27 end

```

Figure 23: ET-Policies

5.5 Summary

SCOUT is a middleware designed to support computational trust formation. SCOUT currently consists of three services: Evidence Gathering Service (EGS), Belief Formation

Service (BFS) and Emotional Trust Service (ETS). EGS is responsible for evidence gathering. BFS is responsible for aggregate belief formation. ETS is responsible for emotional trust formation. All three services are designed with modularity, extensibility and adaptability in mind.

Chapter 6

6 Trust Calculator

The focus of this chapter is on introducing the Trust Calculator. This chapter is divided into three sections. The first section provides a high level overview of the Trust Calculator design. The second section explains how computational trust calculation can be described using a trust calculation plan. The final section of this chapter covers the Trust Calculator. The Trust Calculator calculates computational trust based on the supplied trust calculation plan.

6.1 Trust Calculator Design

The Trust Calculator is designed with the following properties in mind:

- *Adaptability.* There are many factors that could influence computational trust calculation (see Section 4.1.6 for reference). To capture the different ways to calculate computational trust, the Trust Calculator uses trust calculation plans (TcPlans). A TcPlan is basically a description of how computational trust is to be calculated. The Trust Calculator associates with each set of factors a TcPlan and switches TcPlans as the factors change.
- *Ease of Use.* A TcPlan provides a high level abstraction of how computational trust calculation is implemented. By separating the design of computational trust formation from its implementation, even non-developers such as domain experts can participate in determining how computational trust is formed.
- *Reusability.* Different TcPlans may share similar computational trust formation algorithms. To achieve reusability, these algorithms are implemented as nodes that can be referenced in TcPlans. The nodes can also be packaged into libraries. A library can either be home grown or obtained from third parties. By leveraging the libraries when implementing computational trust calculation, a developer could save on development time and effort by not having to implement all the algorithms from scratch.

6.2 Trust Calculation Plan

Trust calculation is a form of inductive reasoning [1]. The reasoning can be modeled as the invocation of a tree. Computational trust calculation therefore is represented as a tree in a TcPlan. The structure of a TcPlan is shown in Figure 24. A TcPlan is divided into two segments: *nodeDefinition* and *trustCalculation*. In *nodeDefinition* (lines 3-5), the nodes of the tree (i.e., the algorithms needed for computational trust calculation) are defined. The *trustCalculation* segment (lines 6-9) consists of two parts. The *trigger* part (line 7) specifies when computational trust calculation is to take place. If no *trigger* is specified, the assumption is for computational trust calculation to take place now. The *tree* part (line 8) specifies the aggregate beliefs and emotional trust to be obtained from SCOUT. It also specifies the tree's construction (i.e., how the nodes are to be applied to aggregate beliefs and emotional trust to form computational trust).

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <tcplan>
3    <nodeDefinition>
4      ...
5    </nodeDefinition>
6    <trustCalculation>
7      <trigger> ... </trigger>
8      <tree> ... </tree>
9    </trustCalculation>
10 </tcplan>

```

Figure 24: Structure of TcPlan

An example TcPlan is shown in Figure 25. In the figure, computational trust is calculated for movie selection. Four nodes are defined in the *nodeDefinition* segment (lines 3-14). Each node is given an identifier through node *id* that is referenced in tree construction. The *class* attribute specifies the implementation of a node. “AggregateBeliefQuery” (lines 4-6) and “EmoTrustQuery” (line 7) are used to query SCOUT for aggregate belief and emotional trust respectively. In the case of “AggregateBeliefQuery”, any non-“aggregateBelief” parameter would be treated as *hints* to the *getAggregateBelief* method of BFS (Figure 28 shows some possible *hints*). “AggregateBeliefToTrust” (line 8) maps the trustor’s aggregate belief to cognitive trust. “WeightedAvg” (lines 9-13) weights all of its inputs based on the “weights” parameter.

In the *tree* part of the *trustCalculation* segment (lines 15-22), an XML element with child elements represents a node with child nodes. Based on the elements, a tree can be constructed. The tree is graphically represented in Figure 26. In the figure, nodes that interact with the SCOUT middleware are represented as squares. All other nodes are represented as circles.

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <tcplan>
3    <nodeDefinition>
4      <node id="qualityBelief" class="AggregateBeliefQuery">
5        <parameter name="aggregateBelief" type="string">Quality</parameter>
6      </node>
7      <node id="emoTrust" class="EmoTrustQuery"/>
8      <node id="cognitiveTrust" class="AggregateBeliefToTrust"/>
9      <node id="computationalTrust" class="WeightedAvg">
10       <parameters name="weights" type="double">
11         <parameter>0.7</parameter><parameter>0.3</parameter>
12       </parameters>
13     </node>
14   </nodeDefinition>
15   <trustCalculation>
16     <tree>
17       <computationalTrust>
18         <cognitiveTrust><qualityBelief/></cognitiveTrust>
19         <emoTrust/>
20       </computationalTrust>
21     </tree>
22   </trustCalculation>
23 </tcplan>

```

Figure 25: TcPlan (Movie)

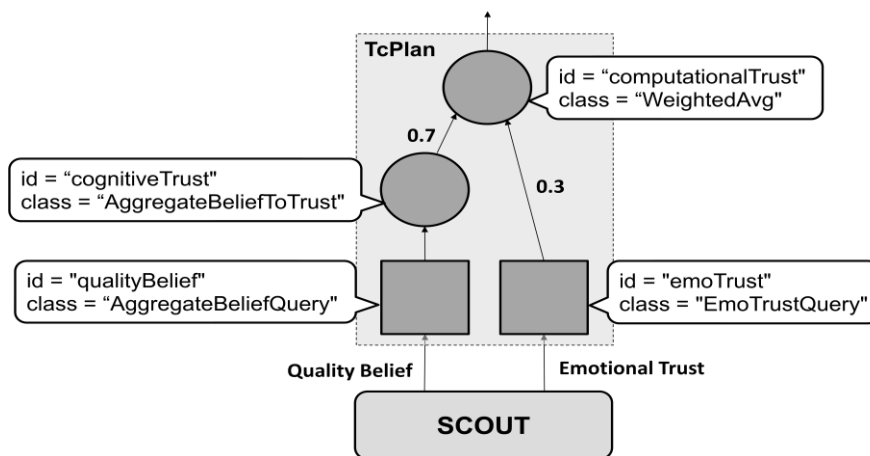


Figure 26: Tree of TcPlan (Movie)

In Figure 25, computational trust is calculated with cognitive trust being assigned a weight of 0.7 and emotional trust being assigned a weight of 0.3. Different movie selectors may have different views on the appropriate weight to assign to cognitive trust and emotional trust. As a result, when constructing a TcPlan, it is not always possible to fill in the weights a priori. A Trust Calculation Template (TcTemplate) is used to allow portions of the TcPlan to be filled in when appropriate. An example TcTemplate is shown in Figure 27. Figure 27 is basically Figure 25 with the weights replaced by variables. The variables are identified by being surrounded by curly brackets (lines 8-9). Also, comments “BEGIN: TcTemplate” (line 2) and “END: TcTemplate” (line 15) are included to identify the TcPlan as a TcTemplate. By assigning values to the variables, the parameterization allows for the creation of a TcPlan from a TcTemplate. Variables in a TcTemplate are commonly associated with information that are subjective (e.g. weights) or decision dependent (e.g. decision importance). This information cannot be known until the application is configured or during decision making.

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <!-- BEGIN: TcTemplate -->
3  <tcplan>
4    <nodeDefinition>
5      ...
6      <node id="computationalTrust" class="WeightedAvg">
7        <parameters name="weights" type="double">
8          <parameter>{COG_WEIGHT}</parameter>
9          <parameter>{EMO_WEIGHT}</parameter>
10       </parameters>
11     </node>
12   </nodeDefinition>
13   ...
14 </tcplan>
15 <!-- END: TcTemplate -->

```

Figure 27: TcTemplate (Movie)

A different TcPlan example is shown in Figure 28. In the figure, computational trust is subscribed and used in web service selection. A “timer” of class “Timer” is defined in the *nodeDefinition* segment (lines 4-7). The “timer” is set to trigger computational trust calculation immediately with zero delay (line 5). After which, calculation is set to trigger every 60 minutes (line 6). Besides triggering calculation by

time, the Trust Calculator also supports triggering through SCOUT subscription. For example, by subscribing to emotional trust or aggregate belief, a TcPlan could trigger computational trust calculations only if there has been a change in the emotional trust or aggregate belief obtained from SCOUT.

In the figure, cognitive trust is calculated from competence belief, performance belief and quality belief. The “expression” parameter of the “cognitiveTrust” node (lines 30-32) is associated with conditions for “aBelief1”, “aBelief2” and “aBelief3”. The numbering used on “aBelief” refers to the order of the elements in the *tree* part of *trustCalculation* segment (lines 44-46). Therefore, “aBelief1” refers to aggregate belief competence; “aBelief2” refers to aggregate belief performance, “aBelief3” refers to aggregate belief quality. All three conditions need to be evaluated to “True” for the cognitive trust value to be set to 1 (i.e., cognitively trusted). Otherwise, cognitive trust value is set to -1 (i.e., cognitively distrusted). As the conditions are all “AND” together, the “cognitiveTrust” node can short circuit the evaluation process if one of the conditions is evaluated to false. Aggregate beliefs are obtained from SCOUT through “AggregateBeliefQuery”. *Hints* are provided to BFS in terms of “importance” (lines 10, 18, 26) and “evidenceType” (lines 14-17, 22-25).

In terms of computational trust formation, the “cause” parameter of the “computationalTrust” node (line 35) is used to determine the calculated computational trust value. If the value obtained from the node’s left child (i.e., “emoTrust”) has a value that is greater than zero ($\text{trust} > 0$), then computational trust is assigned the value of the node’s right child (i.e., “cognitiveTrust”). Otherwise, the returned computational trust value is -1 (i.e., distrusted). Since emotional trust is evaluated first, the “computationalTrust” node does not need to invoke the “cognitiveTrust” node if emotional trust is not greater than zero. This is another demonstration of how computational trust formation can be short circuited. The tree constructed from the TcPlan is graphically represented in Figure 29.

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <tcplan>
3    <nodeDefinition>
4      <node id="timer" class="Timer">
5        <parameter name="delay" type="integer">0</parameter>
6        <parameter name="period" type="integer">60</parameter>
7      </node>
8      <node id="competenceBelief" class="AggregateBeliefQuery">
9        <parameter name="aggregateBelief" type="string">Competence</parameter>
10       <parameter name="importance" type="string">High</parameter>
11     </node>
12     <node id="performanceBelief" class="AggregateBeliefQuery">
13       <parameter name="aggregateBelief" type="string">Performance</parameter>
14       <parameters name="evidenceType" type="string">
15         <parameter>Experience</parameter>
16         <parameter>Reputation</parameter>
17       </parameters>
18       <parameter name="importance" type="string">High</parameter>
19     </node>
20     <node id="qualityBelief" class="AggregateBeliefQuery">
21       <parameter name="aggregateBelief" type="string">Quality</parameter>
22       <parameters name="evidenceType" type="string">
23         <parameter>Experience</parameter>
24         <parameter>Reputation</parameter>
25       </parameters>
26       <parameter name="importance" type="string">High</parameter>
27     </node>
28     <node id="emoTrust" class="EmoTrustQuery"/>
29     <node id="cognitiveTrust" class="ConditionalTrust">
30       <parameter name="expression" type="string">
31         aBelief1 == 1 && aBelief2 > 0 && aBelief3 > 0
32       </parameter>
33     </node>
34     <node id="computationalTrust" class="CausalTrust">
35       <parameter name="cause" type="string">trust > 0</parameter>
36     </node>
37   </nodeDefinition>
38   <trustCalculation>
39     <trigger><timer/></trigger>
40     <tree>
41       <computationalTrust>
42         <emoTrust/>
43         <cognitiveTrust>
44           <competenceBelief/>
45           <performanceBelief/>
46           <qualityBelief/>
47         </cognitiveTrust>
48       </computationalTrust>
49     </tree>
50   </trustCalculation>
51 </tcplan>

```

Figure 28: TcPlan (Web Service)

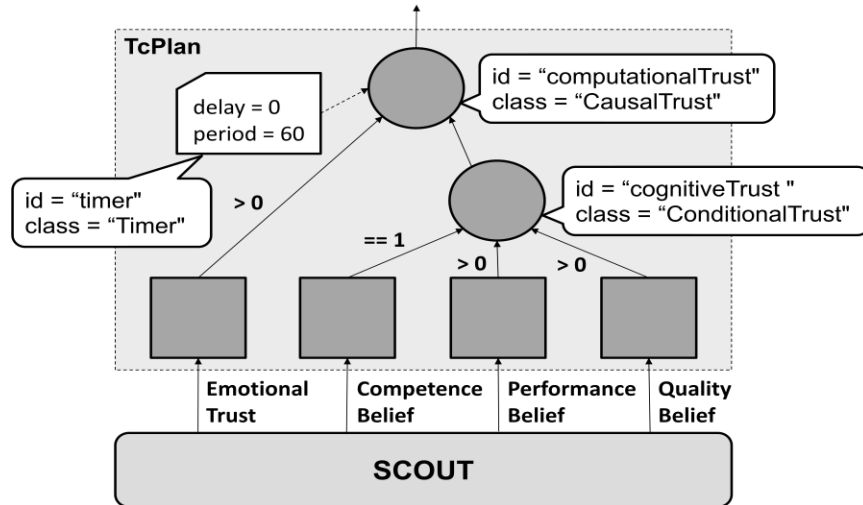


Figure 29: Tree of TcPlan (Web Service)

6.3 Trust Calculator

A Trust Calculator is responsible for computational trust formation. Each application has its own Trust Calculator that can be customized to satisfy the application's computational trust needs. A graphical illustration of a Trust Calculator is shown in Figure 30. The Trust Calculator consists of two components: the Trust Calculation Planner (TcPlanner) and the Trust Calculation Engine (TcEngine). The TcPlanner is responsible for TcPlan selection based on existing factors. The TcEngine is responsible for the execution of the selected TcPlan.

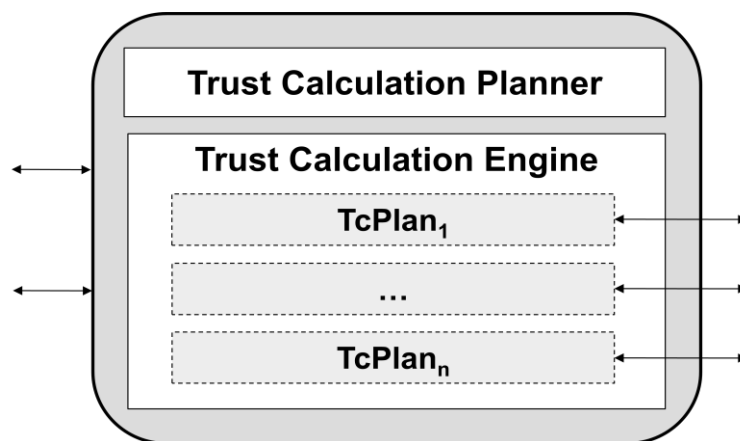


Figure 30: Trust Calculator

The *calculateTrust* method of the Trust Calculator has as input two parameters: *trustee* and *decision factors*. The parameter *trustee* consists of trustee factors. An *application factors* parameter has also been defined and can be supplied to the Trust Calculator through the *setApplicationFactors* method. Unlike *trustee* and *decision factors*, *application factors* are not discarded after computational trust formation. As a result, *application factors* can be reused across decisions. Trustor factors are *application factors* since they generally do not change across decisions. Trustee factors and decision factors can also be *application factors*. An example is decision type where an application that only makes one type of decision can specify decision type as an application factor thereby no longer needing to provide decision type for every *calculateTrust* method invocation. All three factor types are represented as attribute-value pairs. The TcPlanner treats all the factors as the same during TcPlan selection. If there is any attribute conflict (e.g. decision type being provided through both *decision factors* and *application factors*), *application factors* are overwritten by *trustee* and *decision factors*.

The TcPlanner uses the factors to determine the TcPlan to be selected for execution. For example, a decision factor could be importance. There may be different TcPlans associated with different levels of importance. Thus there may be a TcPlan associated with a decision of high importance and another TcPlan associated with a decision of low importance. The factors could also influence the hints to be passed to BFS. For example, a TcPlan corresponding to a decision of high importance may cause a hint to be passed to BFS that reflects the importance of the decision.

If the selected TcPlan is a TcTemplate, the variables in the TcTemplate need to be parameterized by the TcPlanner. It is the responsibility of the application developer to define the mapping from factors to TcPlans (see Section 8.2.1 for examples). After TcPlan selection, the next step is for the TcEngine to execute the TcPlan. This is by instantiating a calculation tree from the TcPlan. By invoking the tree, the TcEngine executes computational trust formation. The final step is for the Trust Calculator to return the calculated computational trust to the application. The entire process is graphically illustrated in Figure 31.

The *subscribeTrust* method of the Trust Calculator has as input three parameters: *trustee*, *decision factors* and *trust listener*. The *trust listener* is notified of any calculated computational trust value. The return value of the method is a unique identifier: *subscriptionId*. The *subscriptionId* identifier is used in the *unsubscribeTrust* method to unsubscribe from an existing subscription. For a subscription to succeed, the corresponding TcPlan needs to have a *trigger* defined. There is no reason for computational trust subscription if a TcPlan only needs to be invoked once. The TcEngine automatically ignores any *trigger* definition in the case of *calculateTrust*. As a result, the same TcPlan can be used for both computational trust query and subscription.

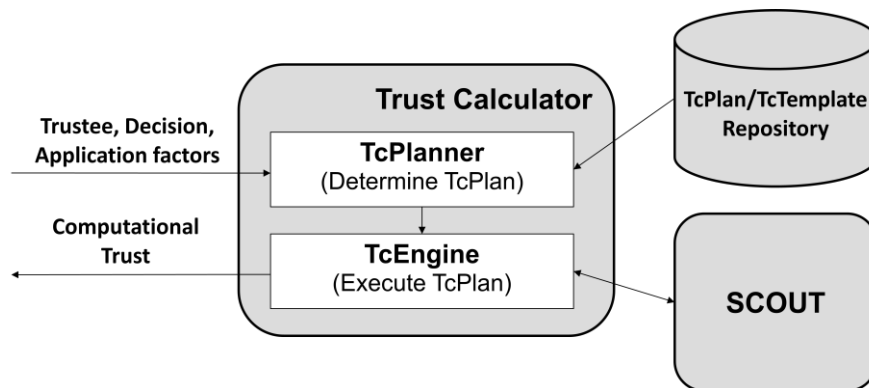


Figure 31: Trust Calculator's Execution

Although in most cases having access to computational trust is enough for making trust-based decisions. There are exceptions when an application may be interested in how computational trust is formed. For example, if the computational trusts calculated for two trustees are the same, examining the underlying calculations may help determine which of the two trustees should be selected. To provide this information to the application, the TcEngine logs the output of each of the nodes in the TcPlan. The log is then returned to the application as part of the calculated computational trust.

6.4 Summary

The Trust Calculator is responsible for computational trust calculation. It consists of two components: the Trust Calculation Planner (TcPlanner) and the Trust Calculation Engine (TcEngine). The TcPlanner takes the trustee, decision factors and application factors into

account when performing TcPlan selection. A TcPlan is basically a description of how computational trust is to be calculated. The selected TcPlan is passed to the TcEngine for execution. The calculated computational trust value is returned to the requesting application. In this thesis, it is assumed that each application has its own Trust Calculator. The Trust Calculator is designed with adaptability, ease of use and reusability in mind.

Chapter 7

7 Algorithms

The focus of this chapter is on introducing the algorithms used in the computational trust architecture. This chapter is divided into two sections. The first section discusses an approach for calculating belief from evidence. The second section explains how evidence source trust can be calculated.

7.1 Belief Formation

There are many ways to calculate belief from evidence (see Table 3 in Section 4.3 for a summary of the algorithms used in the literature). A belief formation algorithm is introduced in this section. The algorithm is applicable to both belief and aggregate belief. The equations used in belief formation are as follows:

$$belief = \frac{w_{exp} \cdot belief_{exp} + w_{rec} \cdot belief_{rec} + w_{rep} \cdot belief_{rep} + w_{sig} \cdot belief_{sig}}{w_{exp} + w_{rec} + w_{rep} + w_{sig}} \quad (1)$$

$$reliability = \frac{w_{exp} \cdot reliability_{exp} + w_{rec} \cdot reliability_{rec} + w_{rep} \cdot reliability_{rep} + w_{sig} \cdot reliability_{sig}}{w_{exp} + w_{rec} + w_{rep} + w_{sig}} \quad (2)$$

In equation (1), $belief_{exp}$ is belief calculated from the trustor's experiences. The values of $belief_{rec}$, $belief_{rep}$ and $belief_{sig}$ are calculated from recommendations, reputations and signals respectively. Since different evidence types have different properties, these properties need to be taken into account when evaluating beliefs calculated from the evidence types. Therefore, a trustor's belief in a trustee is calculated as the weighted average of beliefs calculated from different evidence types. A possible weight assignment is $w_{exp} \geq w_{rec} \geq w_{rep} \geq w_{sig}$.

In equation (2), the reliability of $belief$ is calculated. In this thesis, reliability is an evaluation of the quality of the underlying evidence used in belief calculation. Like belief, reliability is calculated as weighted average of reliabilities calculated from the evidence types. The calculated $belief$ and $reliability$ can be used in cognitive trust calculation.

If there is no evidence of a specific type, its weights in equations (1) and (2) can be set to zero. The equations can be used in a BF-Policy where *belief* and *reliability* are the values returned by the Belief Formation Service (BFS). Alternatively, the equations can be a node in the Trust Calculator. The return values of BFS are then *belief_{exp}*, *belief_{rec}*, *belief_{rep}* and *belief_{sig}*. Both approaches are viable as demonstrated in Section 10.2.1.1. The calculation of *belief_{exp}*, *belief_{rec}*, *belief_{rep}* and *belief_{sig}* are the responsibility of the Belief Engines. The algorithms used are the focus of the rest of this section.

7.1.1 Experience

The equations used for calculating belief from experiences are as follows:

$$belief_{exp} = \frac{\sum_i w_i \cdot exp_i}{\sum_i w_i} \quad \text{where} \quad w_i = e^{-\left(\frac{\Delta t(exp_i)}{\lambda}\right)} \quad (3)$$

$$reliability_{exp} = \begin{cases} w_t \cdot w_{max} + (1 - w_t) \cdot \frac{numExp}{threshold_{exp}}, & numExp < threshold_{exp} \\ w_t \cdot w_{max} + (1 - w_t) & , numExp \geq threshold_{exp} \end{cases} \quad (4)$$

Equation (3) is based on the equations proposed in Regret [110] and FIRE [59]. Belief is calculated as the weighted average of the trustor's experiences. Each experience exp_i is assigned a weight of w_i that is based on the age of the experience. Newer experiences are assigned greater weights since these experiences are more likely to reflect the trustee's current behavior. In weight calculation, $\Delta t(exp_i)$ is the time difference between the current time and the creation time of the experience. The *recency scaling factor* or λ determines how much $\Delta t(exp_i)$ influences w_i . The relationship between $\Delta t(exp_i)$ and w_i for different values of λ are shown in Figure 32. As seen in the figure, w_i decreases as $\Delta t(exp_i)$ increases. As λ increases, the rate of decrease of w_i slows down. For $\lambda = 7.213$, $w_i = 0.500$ when $\Delta t(exp_i) = 5$ but for $\lambda = 14.427$, $w_i = 0.500$ when $\Delta t(exp_i) = 10$ and for $\lambda = 28.854$, $w_i = 0.500$ when $\Delta t(exp_i) = 20$. Increases in values of λ reduces the calculated belief's sensitivity to changes in the trustee's behaviors. As anomalies do happen, it may be desirable to not punish a trustee too severely for a single misbehavior.

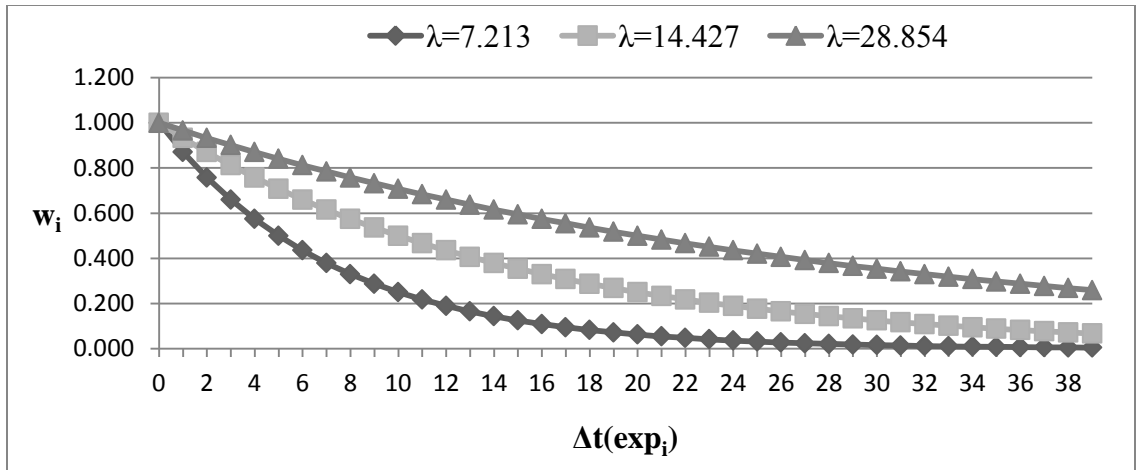


Figure 32: Experience Weights

In equation (4), the reliability of $belief_{exp}$ is calculated based on two factors. The first factor is the age of the experiences. If belief is calculated from old experiences, the calculated belief may be unreliable. As w_i in equation (3) already takes into account the age of an experience, a simple solution is to sum all the weights as an approximation of the age of the experiences [59]. The problem with this approach is that the summed weight is influenced by not just w_i but also the number of experiences. Having more old experiences does not imply higher reliability. To address this problem, only the weight of the newest experience is used as an approximation of the age of the experiences. The newest experience is assigned the maximum weight and is represented in equation (4) as w_{max} . This is a reasonable approximation as the newest experience is also the experience that has the most influence on the calculated $belief_{exp}$.

The second factor that influences reliability is the number of experiences used in belief calculation. As explained in [110], experiences are needed for the trustor to become familiar with the trustee. As familiarity with the trustee increases, so should reliability in the calculated belief. However, once familiarity is established having more experiences should have no impact on belief reliability. Therefore, $numExp$ is used to represent the number of experiences used in belief calculation. The value of $threshold_{exp}$ represents the number of experiences needed to establish familiarity. In

equation (4), if familiarity is established, the factor is assigned the value of one. Otherwise, the factor is represented as the ratio of $numExp$ and $threshold_{exp}$.

Finally, $reliability_{exp}$ is calculated as the weighted average of experience age and experience count factors. The assignment of the weight w_t is based on the trustor's preferences. Old experience that are no longer relevant can be filtered out using an *experience window* ($window_{exp}$). For example, a one year experience window implies that all experiences that are older than one year are to be filtered out and cannot be used in belief formation.

7.1.2 Recommendation

The equations used for calculating belief from recommendations are as follows:

$$belief_{rec} = \frac{\sum_i trust_i \cdot rec_i}{\sum_i trust_i} \quad (5)$$

$$reliability_{rec} = \begin{cases} \frac{\sum_i (trust_i)^2}{\sum_i trust_i} - w_c \cdot \left(1 - \frac{numRec}{threshold_{rec}}\right), & numRec < threshold_{rec} \\ \frac{\sum_i (trust_i)^2}{\sum_i trust_i}, & numRec \geq threshold_{rec} \end{cases} \quad (6)$$

In equation (5), belief is calculated as the weighted average of the gathered recommendations. Each recommendation rec_i is assigned a weight based on the trustor's level of trust in the recommendation's recommender ($trust_i$) [20], [22], [26]. This is so that more trusted recommenders have a larger influence on $belief_{rec}$ than less trusted recommenders.

In equation (6), the reliability of $belief_{rec}$ is calculated based on two factors. The first factor is the trustor's level of trust in the recommenders. The more recommendations are gathered from trusted recommenders, the higher should be the reliability of the calculated belief. This factor is calculated by applying weighted average to recommender trust. The weights ($trust_i$) used are the same as those in equation (5). The second factor is the number of recommendations used in belief calculation. As a recommender is not the trustor, even the most trusted recommender could on occasion deviate from the trustor. Therefore having recommendations from more than one recommender is

desirable and can help increase the reliability of the calculated belief. The calculation of this factor is similar to the calculation of the experience count factor in Section 7.1.1.

Finally, $reliability_{rec}$ is calculated by subtracting the inverse of recommendation count factor from the recommender trust factor. As the recommendation count factor decreases, this should negatively impact $reliability_{rec}$. The magnitude of the impact could be adjusted using the weight w_c which is based on the trustor's preferences.

7.1.3 Reputation

The equations used for calculating belief from reputations are as follows:

$$belief_{rep} = \frac{\sum_i w_i \cdot rep_i}{\sum_i w_i} \quad \text{where} \quad (7)$$

$$w_i = \begin{cases} trust_i \cdot \frac{numFdk_i}{threshold_{fdk}} & , numFdk_i < threshold_{fdk} \\ trust_i & , numFdk_i \geq threshold_{fdk} \end{cases}$$

$$reliability_{rep} = \begin{cases} \frac{\sum_i (w_i)^2}{\sum_i w_i} - w_c \cdot \left(1 - \frac{numRep}{threshold_{rep}}\right) & , numRep < threshold_{rep} \\ \frac{\sum_i (w_i)^2}{\sum_i w_i} & , numRep \geq threshold_{rep} \end{cases} \quad (8)$$

In equation (7), belief is calculated as the weighted average of the gathered reputations. Each reputation rep_i is assigned a weight w_i that is calculated based on two factors. The first factor is the trustor's level of trust in the reputation's reputation system ($trust_i$). A reputation should be given more weight if it is created by a more trusted reputation system. The second factor is the number of feedbacks used by the reputation system in its reputation calculation. If a reputation is calculated from very few feedbacks, it is less reliable and should be given less weight. The number of feedbacks used in reputation calculation is a metric that can readily be found in web-based reputation systems. For example, Amazon provides information on the number of reviews available for each of its products. The same information is also found at Best Buy, eBay, Rotten Tomatoes, IMDb, etc. The calculation of this factor is based on the same approach used for calculating experience count factor and recommendation count factor. Finally, w_i is calculated as the product of reputation system trust factor and feedback count factor.

In equation (8), the reliability of $belief_{rep}$ is calculated using the same approach used for calculating $reliability_{rec}$ in Section 7.1.2. The value of $reliability_{rep}$ is calculated by subtracting the inverse of reputation count factor from the weight factor.

7.1.4 Signal

A signal can be mapped to a numeric domain. An example signal is an Extended Verification (EV) certificate [21] that is used as confirmation of a website's identity. The signal could be mapped to the domain of -1, 0 and 1. If a trustee does not have an EV certificate, this could be mapped to 0. If the EV certificate failed to be validated, this could be mapped to -1. Otherwise, the mapping would be to 1. After mapping, the signals can now be used in belief formation. The equations used for calculating belief from signals are as follows:

$$belief_{sig} = \frac{\sum_i trust_i \cdot sig_i}{\sum_i trust_i} \quad (9)$$

$$reliability_{sig} = \begin{cases} \frac{\sum_i (trust_i)^2}{\sum_i trust_i} - w_c \cdot \left(1 - \frac{numSig}{threshold_{sig}}\right) & , numSig < threshold_{sig} \\ \frac{\sum_i (trust_i)^2}{\sum_i trust_i} & , numSig \geq threshold_{sig} \end{cases} \quad (10)$$

In equation (9), belief is calculated as the weighted average of the supplied signals. The calculation uses the same approach used for calculating $belief_{rec}$ in Section 7.1.2. In equation (10), the reliability of $belief_{sig}$ is calculated using the same approach used for calculating $reliability_{rec}$ in Section 7.1.2. The value of $reliability_{sig}$ is calculated by subtracting the inverse of signal count factor from the signaler trust factor.

7.1.5 Summary

Several algorithms have been proposed for belief formation. For belief formation from experiences, the *recency scaling factor* λ is used to influence the weight assigned to each experience. As λ increases, the calculated belief becomes less sensitive to the age of the experiences. The weight w_t is used to determine whether age of experiences or amount of experiences (influenced by subjective $threshold_{exp}$) should be the main determinant

for belief reliability. For trustees that change behaviors frequently, smaller λ and larger w_t is preferable. The opposite would be the case for trustees that seldom change.

As for belief formation from recommendations, recommender trust can be used to weigh each of the gathered recommendation. The subjective weight w_c is used to determine the influence that the amount of recommendations (influenced by subjective *threshold_{rec}*) have on belief reliability. The same concept is also applicable to belief formation from reputations. The only difference is that reputation is weighted by not just reputation system trust but also by the number of feedbacks used in reputation calculation. Finally, belief formation from signals is calculated using the same approach as that for recommendations. The beliefs calculated from different evidence types can be aggregated using weighted average with the weight assignments dependent on the properties of evidence types.

7.2 Evidence Source Assessment

There are many ways to calculate evidence source trust as discussed in Section 5.2.3. An evidence source assessment algorithm is introduced in this section. The algorithm can be viewed as consisting of two stages: evidence assessment and evidence source trust assessment. Both stages are described in detail in the rest of this section.

7.2.1 Evidence Assessment

There are many ways to perform evidence assessment. In [127], Chebyshev's rule [89] is used. In [6], [67] and [115], assessment is based on compliance level. The compliance level approach has been adopted in this section. The approach calls for the evaluation of the gathered evidence with respect to some evidence quality standard. The standard used in this section is the trustor's feedback. A trustor provides feedback to the Evidence Gathering Service (EGS) after it has interacted with the trustee. To be in compliance therefore means that the gathered evidence should be similar to the trustor's feedback. The equation used to perform evidence assessment is as follows:

$$assessment_{ij} = 1 - \frac{1}{similarity} \cdot |fdk_i - ev_j| \quad (11)$$

In equation (11), $assessment_{ij}$ has an interval of $[-1, 1]$. Any calculated $assessment_{ij}$ that is less than -1 is mapped to -1 instead. Similarity is represented as the absolute difference between the feedback fdk_i and the evidence ev_j . If the absolute difference is zero, then the evidence is assigned the maximum assessment of one. Increases in the absolute difference should have a negative impact on the calculated assessment. The level of impact can be adjusted through $similarity$. The relationships between $|fdk_i - ev_j|$ and $assessment_{ij}$ for different $similarity$ values are shown in Figure 33. In the figure, as $similarity$ increases, the rate of decrease of $assessment_{ij}$ slows down. Changes in $similarity$ therefore can be used to adjust how sensitive EGS should be to evidence noncompliance.

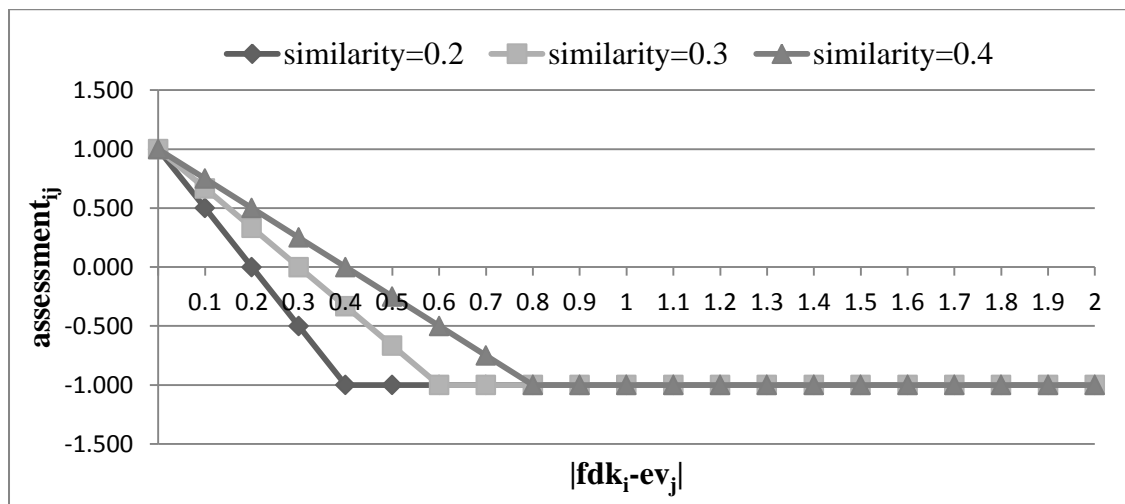


Figure 33: Evidence Assessments

Using equation (11), a single feedback can be used to assess the qualities of multiple evidence. However, not all gathered evidence should be considered in evidence assessment. As a trustee's behavior may change over time, it is unreasonable to expect old evidence to be in compliance with the trustor's feedback. Therefore, an *evidence window* ($window_{ev}$) has been defined that can be used to limit the evidence considered during evidence assessment. As an example, an evidence window of 60

minutes would limit the evidence to those that have creation timestamp (or gathered timestamp if creation timestamp is unavailable) that is within the past hour of the feedback's creation.

An evidence could have multiple assessments as it could be within the $window_{ev}$ of multiple feedbacks. If so, these assessments need to be aggregated to form the trustor's overall assessment of the evidence. The equation used for calculating overall evidence assessment is as follows:

$$assessment_j = \frac{\sum_i w_{ij} \cdot assessment_{ij}}{\sum_i w_{ij}} \quad \text{where} \quad w_{ij} = 1 - \frac{\Delta t(ev_j) - \Delta t(fdki)}{window_{ev}} \quad (12)$$

In equation (12), the overall assessment is calculated as the weighted average of the individual assessments. Each assessment $assessment_{ij}$ is assigned a weight based on the difference between feedback creation time and evidence creation time (or evidence gathered time if evidence creation time is unavailable). The time difference is represented as the difference in evidence age ($\Delta t(ev_j)$) and feedback age ($\Delta t(fdki)$). The resulting difference is normalized using $window_{ev}$. The inverse of which is the weight assigned to an assessment. If an evidence is created or gathered close to when the trustor interacts with the trustee (i.e., when feedback is created), it is more likely to be correct than an older evidence. As a result, the assessment of this evidence should be given more weight than assessment of older evidence that has a higher likelihood of being wrong.

7.2.2 Evidence Source Trust Assessment

Evidence source trust can be viewed as a form of computational trust. As a result, evidence source trust can be calculated from cognitive trust and emotional trust. In terms of cognitive trust, the overall evidence assessments calculated using equation (12) in Section 7.2.1 can be viewed as the trustor's experiences with the evidence sources. Equations (3) and (4) in Section 7.1.1 therefore can be used to calculate the trustor's quality belief in the evidence provided by an evidence source. With belief calculated, the last step is the calculation of cognitive trust. The equation for cognitive trust calculation is as follows:

$$cogTrust_k = belief_k - w_{reliability} \cdot (1 - reliability_k) \quad (13)$$

In equation (13), cognitive trust in an evidence source k is calculated as the difference between belief and the inverse of reliability. As belief reliability decreases, this should have a negative effect on the calculated cognitive trust. The magnitude of the effect can be adjusted through $w_{reliability}$. For example if $w_{reliability} = 0$, the calculated cognitive trust would be solely based on the calculated belief.

In terms of emotional trust, it can be calculated based on the trustor's recognition of the evidence source. As for computational trust, its calculation is as follows:

$$compTrust_k = w_{ct} \cdot cogTrust_k + (1 - w_{ct}) \cdot emoTrust_k \quad (14)$$

In equation (14), computational trust is calculated as the weighted average of cognitive trust and emotional trust. The assignment of the weight w_{ct} is based on the trustor's preferences. Some possible values are shown in Table 1 of Section 4.1.6.1.

7.2.3 Summary

An algorithm for calculating evidence source trust is proposed in this section. The algorithm assesses the gathered evidence by comparing the evidence to the trustor's feedback. If the evidence is similar to the trustor's feedback, a positive assessment would be assigned to the evidence. Otherwise a negative assessment would be assigned. Similarity's influence on the calculated assessment can be adjusted through *similarity*. For an optimistic trustor, the value assigned to *similarity* could be larger than that for a pessimistic trustor that views any dissimilarity from feedback with suspicion. The window $window_{ev}$ is used to limit the evidence to be considered during assessment. The value for $window_{ev}$ could be based on the rate at which the trustees change behaviors. As evidence can be reused, they may be multiple feedbacks for each piece of evidence. An equation therefore has been proposed on how evidence assessment can take into account multiple feedbacks.

Finally, the assessments are treated as experiences from evidence gathering. This in turn allows for the calculation of cognitive trust with subjective weight $w_{reliability}$ being used to determine the influence that belief reliability has on cognitive trust. As for the calculation of computational trust in an evidence source, the subjective weight to be placed on cognitive trust and emotional trust is determined by w_{ct} .

7.3 Summary

Two algorithms have been proposed in this chapter. The first algorithm is used to calculate belief from evidence of different types. The second algorithm calculates evidence source trust in two stages: evidence assessment and evidence source trust assessment.

Chapter 8

8 Implementation

The focus of this chapter is on describing the implementation of the computational trust architecture. This chapter is divided into two sections. The first section introduces the prototype implementation of SCOUT. The second section introduces the prototype implementation of the Trust Calculator.

8.1 SCOUT

SCOUT (Figure 34) is a web application that is implemented in Java EE 5. The implementation is deployed on GlassFish [46], an open source application server. SCOUT consists of three services: Evidence Gathering Service (EGS), Belief Formation Service (BFS) and Emotional Trust Service (ETS). EGS consists of Evidence Gathering Manager (EGM), Policy Engine, Evidence Gatherers, Evidence Handlers, Evidence Source Assessors and Job Scheduler. BFS consists of Belief Formation Manager (BFM), Subscription Manager, Policy Engine and Belief Engines. ETS consists of Emotional Trust Manager (ETM), Subscription Manager and Policy Engine. Implementation wise, the SCOUT services shared a number of SCOUT components.

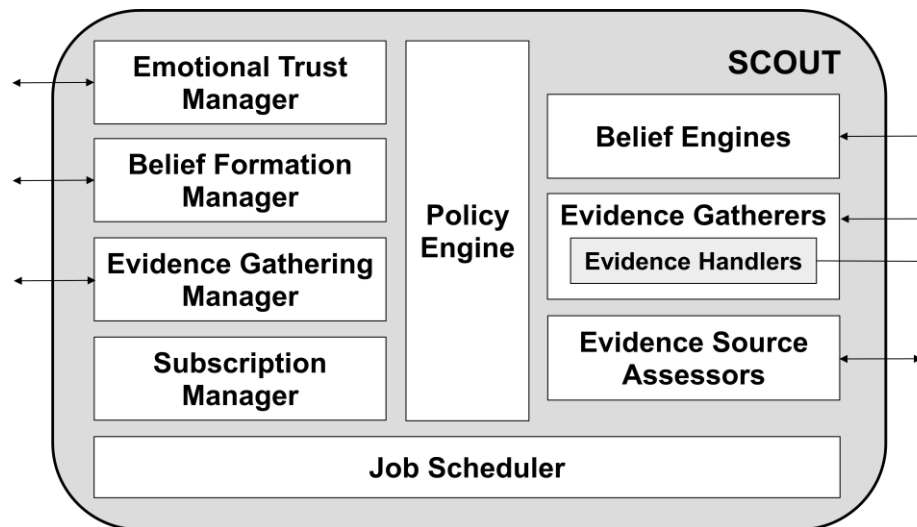


Figure 34: SCOUT Implementation

EGM, BFM and ETM are implemented as EJBs (Enterprise Java Beans) with web service frontends. All three managers depend on the Policy Engine for interacting with the deployed plug-ins (Evidence Gatherers, Evidence Source Assessors and Belief Engines). The Policy Engine is an EJB that uses the Drools rule engine [37] for policy processing. The policies are stored in a Policy Repository. It is the responsibility of the managers to retrieve and deploy the policies to the Policy Engine. As for the different plug-ins, they are all implemented as EJBs with communication with the Policy Engine being based on JMS (Java Message Service). With JMS, messages are delivered by specifying the destination of a message (as oppose to communicating using an EJB reference) which in our implementation is based on identities of the plug-ins and the Policy Engine.

The Subscription Manager is implemented as an EJB. It uses the Timer Service offered by Glassfish to perform periodic subscriptions re-evaluation. As for the specification and processing of aggregate belief filters and emotional trust filters, these are based on the MVEL expression language [95]. The filtered aggregate beliefs and emotional trusts are stored in a JMS message queue. A subscriber can retrieve its subscribed information using its *subscriptionId*.

The Job Scheduler is implemented as a Servlet. It is based on the open source job scheduling service Quartz [99]. The Job Scheduler is used by the Evidence Source Assessors to trigger periodic evidence source trust calculation. It is also used to perform Evidence Repository maintenance. As an example, evidence stored in the Evidence Repository are periodically examined. If the evidence is outdated, they are removed from the Evidence Repository.

8.2 Trust Calculator

The Trust Calculator is implemented as a Java SE 6 library. The implementation of the Trust Calculation Planner (TcPlanner) and the Trust Calculation Engine (TcEngine) is described in the rest of this section.

8.2.1 Trust Calculation Planner

The TcPlanner is implemented as a Java interface. It consists of a single method *formulateTcPlan* that has as input a single parameter *factors*. The return value of the method is the formulated TcPlan. An application developer could implement its own TcPlanner based on the TcPlanner interface. An application developer could also use one of the two implementations included in the Trust Calculator library for TcPlan selection.

The first TcPlanner implementation in the library is the *XMLTcPlanner*. The planner reads an XML document that describes the mappings from factors to TcPlan. An example document is shown in Figure 35. In the figure, if the decision type is movie selection (line 5), then the TcTemplate named “MovieTrust” is retrieved and parameterized with the supplied parameters (lines 7-13). The parameterization is based on the Jtpl template engine [66]. If the mapping is to a TcPlan, then the “tctemplate” tag is replaced by the “tcplan” tag. *XMLTcPlanner* offers a simple way to perform factors to TcPlan mapping. If more complex mappings are needed, the *PolicyTcPlanner* in the library can be used instead. The *PolicyTcPlanner* performs TcPlan selection based on policies. The implementation uses the Drools rule engine [37] for policy processing. An example policy is shown in Figure 36. The example is the same as Figure 35 except the mapping is in policy form.

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <mappings>
3    <mapping>
4      <factors>
5        <factor name="DecisionType">MovieSelection</factor>
6      </factors>
7      <tctemplate>
8        <name>MovieTrust</name>
9        <parameters>
10       <parameter name="COG_WEIGHT">0.7</parameter>
11       <parameter name="EMO_WEIGHT">0.3</parameter>
12     </parameters>
13   </tctemplate>
14 </mapping>
15 </mappings>

```

Figure 35: Factors-TcPlan Mapping (Movie)

```

1  when
2    Factor( name=="DecisionType" && value == "MovieSelection" )
3  then
4    template = new TcTemplate("MovieTrust");
5    template.set("COG_WEIGHT", 0.7);
6    template.set("EMO_WEIGHT", 0.7);
7    template.parse();
8  end

```

Figure 36: Factors-TcPlan Policy (Movie)

8.2.2 Trust Calculation Engine

The TcEngine performs computational trust calculation by parsing the TcPlan supplied by the TcPlanner. First the nodes defined in the *nodeDefinition* segment of the TcPlan are parsed and instantiated. The nodes all belong to one of three types: belief-belief, belief-trust and trust-trust. A belief-belief node takes aggregate belief as input and produces aggregate belief as output. An example belief-belief node is the *AggregateBeliefQuery* node that retrieves aggregate belief from SCOUT. A belief-trust node takes aggregate belief as input and produces trust as output. An example belief-trust node is the *AggregateBeliefToTrust* node that calculates cognitive trust from aggregate belief. A trust-trust node takes trust as input and produces trust as output. An example trust-trust node is *EmoTrustQuery* node that retrieves trust from SCOUT. If a node needed for computational trust calculation is not available in the Trust Calculator library, an application developer can choose to implement the node by extending one of the three basic node types.

With the nodes instantiated, the next step is to consult the *tree* part of the *trustCalculation* segment. For each element and its children, the corresponding nodes are retrieved and their parent-children relationships established. By performing breadth-first traversal, a tree can be constructed. The last step is to invoke the root node of the tree to perform computational trust calculation. It is the responsibility of each parent node to determine which of its child nodes should be invoked to continue the calculation.

As for computational trust subscription, besides the creation of a calculation tree, a tree also needs to be created for triggering the computational trust calculation. This

involves the parsing of the *trigger* part of the *trustCalculation* segment. The approach taken is exactly the same as for calculation tree creation. After both trees are created, the next step is for the TcEngine to invoke all the leaf nodes of the trigger tree. Associated with each trigger node are conditions that have to be satisfied before its parent node can be triggered. This would continue on until the root node of the tree has been triggered. This causes the invocation of the root node of the calculation tree to perform computational trust calculation. If a node needed for triggering computational trust calculation is not available in the Trust Calculator library, an application developer can choose to implement the node by extending the basic trigger node type.

8.3 Summary

SCOUT is implemented as a web application. The components of SCOUT are shared by all three SCOUT services. As an example, the Policy Engine is shared by all three SCOUT managers. The managers interact with the SCOUT plug-ins through the deployed SCOUT policies. As for the Trust Calculator, it is implemented as a library that can be used to perform computational trust calculation. In terms of the TcPlanner, two implementations have been showcased. In terms of the TcEngine, the focus is on how a TcPlan can be parsed and instantiated for computational trust query and subscription.

Chapter 9

9 Experiments

The focus of this chapter is on presenting the experimental results. This chapter is divided into eight sections. The first section discusses the goals of the experiments. The second section examines the assumptions made in the experiments. Next, the metrics for measuring the effectiveness of computational trust are introduced. This is followed by an introduction to the experimental testbed. In section five, the experiments based on experiences are explained. Next, the experiments based on recommendations are discussed. This is followed by discussion of experiments based on both experiences and recommendations. The last section of this chapter examines the conclusions that could be drawn from the experiments.

9.1 Goals

The experiments in this chapter are designed to show the following:

- *Evidence source type and behavior plays a key role in computational trust formation.* An evidence source could be of type: trustor, recommender, reputation system or signal provider. An evidence source could behave in an honest or malicious manner. The experiments are designed to evaluate how the effectiveness of computational trust changes with changes in the availability of evidence source type and evidence source behavior.
- *Effectiveness of different computational trust formation algorithms.* Specifically, the averaging of evidence is used as a base case. The comparison is between the use of a window to discard old evidence ($window_{exp}$ on page 84 of Section 7.1.1) and the use of weighted average (equation (3) of Section 7.1.1 including varying *recency scaling factor* or λ) to give newer evidence more weight. The goal is to investigate which approach is more effective. Thresholds (e.g. $threshold_{exp}$) are not considered in the experiments. We consider the varying of belief reliability to be part of future work.

9.2 Assumptions

The experiments in this chapter are designed with the following assumptions in mind:

- *Computational trust is evaluated in a web service selection scenario.* There are many different approaches to evaluate computational trust. In this chapter, computational trust is evaluated based on its effectiveness in supporting web service selection.
- *Computational trust formation.* Emotional trust evaluation is considered future work. Instead, the focus of the experiments is on cognitive trust. Basically, cognitive trust and computational trust are treated as the same in the experiments. As for cognitive trust formation, it is assumed that cognitive trust is formed from the aggregate belief performance and that the aggregate belief is formed from either experiences or recommendations. For simplicity sake, it is assumed that the calculated belief reliability has no influence on cognitive trust. These assumptions are setup to limit the number of variables in the experiments. The assumptions should be reduced or even eliminated in future work. Also, it is assumed that experience, recommendation, aggregate belief, cognitive trust, computational trust and feedback are all in the interval of $[-1, 1]$.
- *Most web services provide average usage experiences.* The web services are configured as in Table 5 where most of the web services (40%) provides usage experiences of 0 while few of the web services (10%) provides extremely high (0.8) or extremely low (-0.8) usage experiences. The setup is chosen as it is a reflection of the belief that extreme usage experiences are rare while average usage experiences are common in everyday life. This is just one possible web services setup. Other setups such as when extremes are more common than average or when all usage experiences are equally likely are part of future work. It is expected that different setups may lead to different experimental outcomes.

Table 5: Web Services (Experiments)

Number of Web Services	Usage experience
1	0.8
2	0.4
4	0
2	-0.4
1	-0.8

- *Most evidence sources are honest.* It is assumed that most evidence sources would not change its evidence just to mislead the trustor. The setup is chosen the belief that most honesty is more prevalent in everyday life. Other setups such as most evidence sources being dishonest are considered future work. It is expected that different setups may lead to different experimental outcomes.

9.3 Metrics

The effectiveness of computational trust is evaluated using the following two metrics

- *Mean experience (exp).* After a web service is selected, the trustor could invoke the web service to gain experience. The mean experience is calculated by averaging all of the gained experiences. This is to evaluate the effectiveness of computational trust in terms of its contribution to the trustor's overall experiences.
- *Percentage of positive experiences (+exp).* A trustor should avoid invoking web services that provide below average usage experiences (i.e., $exp_{usage} < 0$). The percentage of the trustor's experiences that are greater than or equal to zero can be used to evaluate the effectiveness of computational trust in terms of its ability to minimize the trustor's negative experiences.

9.4 Experimental Testbed

The experiments in this chapter are conducted using an experimental testbed that is described in this section. This is followed by a discussion of the different factors that can be configured in the testbed.

9.4.1 Overview

The experimental testbed was developed using the Java programming language. It implements the algorithm presented in Figure 37. In the setup phase (lines 1-4), the web services, evidence sources and selector are configured. The selector implements an algorithm for performing web service selection. A timer is also initialized to keep track of time in the testbed.

With the completion of the setup phase, the testbed could begin web service selections (lines 6-19). For all of the experiments in this chapter, *repeatCount* is set to 100 (line 6) and *selectionCount* is set to 50 (line 7). The testbed therefore performs 50 web service selections. Afterwards, *exp* and *+exp* are calculated. This process is repeated 100 times in order to reduce the impact that randomness has on the experiments. After 100 experimental runs, the calculated *exp* and *+exp* are averaged to form the *exp* and *+exp* of an experiment.

```

1  configure web_services
2  configure evidence_sources
3  configure selector
4  initialize timer
5
6  for i = 1 to repeatCount
7      for j = 1 to selectionCount
8          web_service = selector.select(web_services)
9          experience = web_service.invoke()
10
11             EGS.provideFeedback(web_service, experience)
12             Evidence_Repository.store(web_serivce, experience)
13
14             increment timer
15         end
16
17     reset selector, timer
18     cleanup Evidence_Respository
19 end

```

Figure 37: Experimental Testbed's Algorithm

9.4.2 Web Services

The testbed can be configured with four different types of web services:

- *Static web service.* A static web service always provides its default usage experience to the trustor. For example, a web service with $exp_{usage} = 0.4$ always provide experience of 0.4 when invoked.
- *Fluctuating web service.* A fluctuating web service provides its default usage experience to the trustor. Occasionally, the web service may provide usage experience that deviates from the default. The fluctuation is usually temporary and lasts for a short period of time. An example of a fluctuating web service is an online retailer whose performance fluctuates during boxing week sales due to its servers being overloaded. The fluctuation of a web service is represented using two parameters. The parameter $prob_f$ represents the likelihood of fluctuation. The parameter $duration_f$ represents the maximum duration of fluctuation. Duration is measured in time units generated by the testbed's timer. The magnitude and duration of the fluctuation is randomly generated with a uniform distribution. For example, a web service with $exp_{usage} = 0.4$, $prob_f = 0.1$ and $duration_f = 2$ has a 10% chance of fluctuation. During fluctuation, exp_{usage} is assigned a randomly generated value. This represents the magnitude of the change. The duration of the fluctuation is randomly generated to be less than or equal to $duration_f$. By the end of the duration, exp_{usage} is reset. In the example, exp_{usage} is reset to 0.4.
- *Dynamic web service.* A dynamic web service is a static web service whose default usage experience changes over time. For example, the installation of new servers may permanently improve the usage experience of the trustor. As a web service's popularity increases, the increased traffic may permanently deteriorate the usage experience of the trustor. The dynamisms of a web service is represented by the $prob_d$ parameter. The parameter shows the likelihood of a default usage experience change. For each change, a new default usage experience is randomly generated with a uniform distribution. The web service's usage experience is then updated by 0.1 every time unit to move towards the new default usage experience. For example, a web service with $exp_{usage} = 0.4$ and

$prob_d = 0.04$ has a 4% chance of dynamism. During dynamism, a new default usage experience is generated. Assume that the generated value is 0.7. For the next three time units, the usage experience provided by the web service becomes 0.5, 0.6 and 0.7 respectively. After arriving at 0.7 (i.e., the new default usage experience), the usage experience remains at 0.7 until the next dynamism.

- *Fluctuating-Dynamic (FD) web service.* A FD web service has the properties of both fluctuating web service and dynamic web service. The usage experience provided by a FD web service is represented using three parameters: $prob_f$, $duration_f$ and $prob_d$. For example, a web service with $exp_{usage} = 0.4$, $prob_f = 0.1$, $duration_f = 2$ and $prob_d = 0.04$ has a 4% chance of dynamism. During dynamism, the web service's usage experience is updated as in the case of a dynamic web service. Otherwise, it needs to be determined whether there is the 10% chance of fluctuation. During fluctuation, the web service's usage experience is updated as in the case of a fluctuating web service. Otherwise, the usage experience provided by the web service would remain the same at 0.4. For simplicity sake, it is assumed that a FD web service cannot be both fluctuating and dynamic at the same time.

9.4.3 Evidence Sources

The testbed can be configured with four different types of evidence sources:

- *Trustor.* The trustor provides its experiences with a web service to be used in performance belief formation.
- *Honest recommender.* An honest recommender provides its own experiences as recommendation to be used in performance belief formation. As a recommender may have different experiences than the trustor, the *deviation* parameter is used to represent the difference between the trustor's usage experience and recommendation. The value of *deviation* is randomly generated with a uniform distribution within a specified range. For example if $deviation = [-0.1, 0.1]$,

then any recommendation provided by the honest recommender would deviate by at most 0.1 from the trustor's usage experience.

- *Malicious recommender.* The goal of a malicious recommender is to mislead the trustor. Therefore, recommendation provided by a malicious recommender is the inverse of the trustor's usage experience. For example, if the trustor's usage experience is 0.5, then the recommendation provided would be -0.5.
- *Oscillating recommender.* An oscillating recommender changes its recommendation from honest to malicious and vice versa at regular intervals. The interval is determined by the $duration_o$ parameter. For example with $duration_o = 5$, the oscillating recommender would provide honest recommendations for 5 time units. The recommender would then switch to providing malicious recommendations for 5 time units. The transition between honest to malicious and vice versa would continue to occur every 5 time units.

9.4.4 Web Service Selection Strategies

There are many different ways to perform web service selection. The simplest strategy is the *random strategy* where a web service is selected at random. Web services could also be selected based on computational trust. An example is to always select the web service that is most trusted. This strategy is known as the *max-trust strategy*. An assumption inherent in this strategy is that computational trust can be calculated for each of the web services. If the trustor only has access to its own experiences, then it is possible that computational trust cannot be calculated for those web services that the trustor does not have experiences with. When faced with this challenge, exploration may be needed. Exploration and several other terms are defined as follows:

- *Unknown web service:* If a web service has never been invoked by the trustor, the web service is unknown to the trustor. Therefore, computational trust cannot be calculated for the web service.

- *Known web service*: If a web service is known to the trustor, then the trustor has experience with the web service. Therefore, computational trust can be calculated for the web service.
- *Exploration*: Exploration is the random selection of an unknown web service for invocation. Exploration allows the trustor to gain experience with an unknown web service thereby the transitioning the unknown to a known web service.
- *Exploitation*: Exploitation utilizes the max-trust strategy to select from the known web services.

Web service selection between unknown and known web services can be viewed as a tradeoff between exploration and exploitation. By choosing exploration, a trustor could determine whether an unknown web service could be trusted. By choosing exploitation, the trustor could take advantage of the calculated computational trust for known web services. There are different heuristics for determining when to explore and when to exploit. The testbed supports two strategies: *ϵ -greedy exploration strategy* and *Boltzmann exploration strategy*. Both strategies are introduced in the rest of this section.

9.4.4.1 ϵ -greedy Exploration Strategy

The ϵ -greedy exploration strategy uses the calculated exploration ratio to determine when to explore and when to exploit. By generating a random number between 0 and 1, the random number could be compared to the exploration ratio. If the random number is less than the exploration ratio, selection is based on exploration. Otherwise, the selection is based on exploitation. The equation for calculating exploration ratio is the following:

$$explorationRatio = \frac{1}{1+(time \cdot dr)} \quad (15)$$

In equation (15), time is a determinant for the value of the exploration ratio. When $time = 0$, the $explorationRatio = 1$. As the time unit value increases, the exploration ratio decreases. The strategy is based on the observation that at $time = 0$, the web services are all unknown, therefore the trustor should explore. As the time unit value increases and as more and more web services become known, the strategy should

transition to mainly exploitation. The parameter dr determines how much $time$ influences $explorationRatio$. The relationship between $explorationRatio$ and $time$ for different values of dr are shown in Figure 38. In the figure, higher values of dr speeds up the rate at which $explorationRatio$ decreases over $time$.

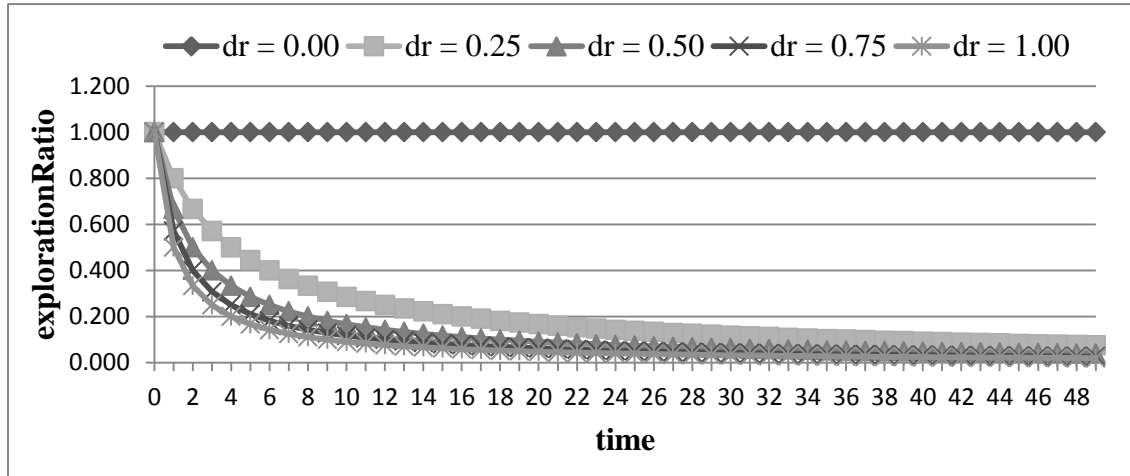


Figure 38: Exploration Ratios

9.4.4.2 Boltzmann Exploration Strategy

The Boltzmann exploration strategy uses the calculated exploration probability to determine when to explore and when to exploit. By generating a random number between 0 and 1, the random number could be compared to the exploration probability. If the random number is less than the exploration probability, selection is based on exploration. Otherwise, the selection is based on exploitation. The equations used for calculating exploration probability are the following:

$$explorationProbability = \frac{e^{\left(\frac{exp_{exploration}}{temperature}\right)}}{e^{\left(\frac{exp_{exploitation}}{temperature}\right)} + e^{\left(\frac{exp_{exploration}}{temperature}\right)}} \quad (16)$$

$$temperature = \begin{cases} 1.0 & , time = 0 \\ temp_f + dr \cdot (temperature - temp_f) & , time > 0 \end{cases} \quad (17)$$

In equation (16), exploration probability is influenced by two factors: the experience the trustor would gain from exploration ($exp_{exploration}$) and the experience the trustor would

gain from exploitation ($exp_{exploitation}$). The $exp_{exploration}$ is estimated by averaging all of the trustor's past experiences from exploration. The $exp_{exploitation}$ is estimated from the computational trust calculated for the most trusted web service. If the calculated $exp_{exploration} > exp_{exploitation}$, then the strategy prefers exploration. If the calculated $exp_{exploration} < exp_{exploitation}$, then the strategy prefers exploitation. If the calculated $exp_{exploration} = exp_{exploitation}$, then the strategy has no preference.

The influence that $exp_{exploration}$ and $exp_{exploitation}$ have on exploration probability is determined by the *temperature* parameter. Temperature is calculated using equation (17) and is updated after every time unit value increment. The calculated temperature decreases as the time unit value increases. The rate of decrease is determined by the parameter dr . The parameter $temp_f$ determines the minimum value of temperature. As *temperature* is used as a divisor in equation (16), $temp_f \neq 0$. The relationship between *explorationProbability* and *time* for different values of $exp_{exploration}$, $exp_{exploitation}$, dr and $temp_f$ are shown in Figure 39. The legend of the figure is shown in Table 6.

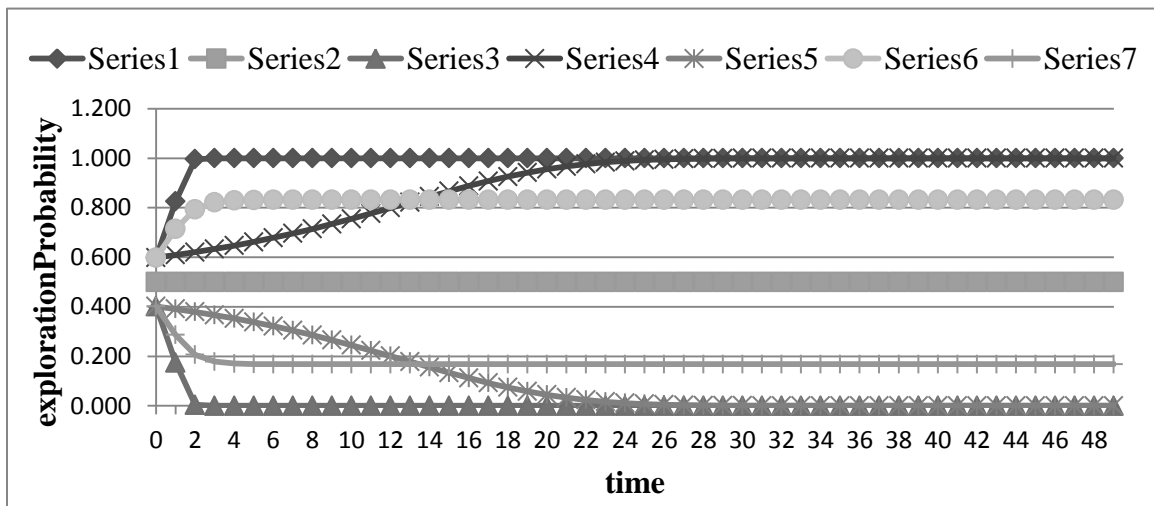


Figure 39: Exploration Probabilities

Table 6: Legend of Figure 39

	$exp_{exploration}$	$exp_{exploitation}$	dr	$temp_f$
Series1	0.00	-0.40	0.25	0.01
Series2	0.00	0.00	0.25 / 0.90	0.01 / 0.25
Series3	0.00	0.40	0.25	0.01
Series4	0.00	-0.40	0.90	0.01
Series5	0.00	0.40	0.90	0.01
Series6	0.00	-0.40	0.25	0.25
Series7	0.00	0.40	0.25	0.25

In Figure 39, if $exp_{exploration} = exp_{exploitation}$ as is the case with *Series2*, then exploration and exploitation are equally likely ($explorationProbability = 0.5$) since both would return a usage experience of 0. As dr increases, this slows down the rate of change of $explorationProbability$ as shown in *Series4* and *Series5*. Higher values of $temp_f$ results in a smaller range of values for $explorationProbability$ as shown in *Series6* and *Series7*.

9.5 Experiments Based on Experience

In this section, the testbed is configured with the trustor as the only evidence source. In each of the subsections, the experiments conducted are based on a different web service type. The web service selection strategies are configured as follows:

- Random strategy
- ϵ -greedy exploration strategy
 - dr : 0, 0.25, 0.5, 0.75, 1
- Boltzmann exploration strategy
 - $exp_{exploration}$: - 0.1, 0, 0.25, 0.5, 0.75
 - dr : 0, 0.25, 0.5, 0.75, 0.9, 0.95, 0.99
 - $temp_f$: 0.01, 0.1, 0.25, 0.75

The max-trust strategy is not used in the experiments as computational trust cannot be calculated for unknown web services. The computational trust formation algorithms are configured as follows:

- Average
 - $window_{exp}: \infty$
- Average
 - $window_{exp}: 5, 10, 15, 20, 25$
- Weighted average
 - $\lambda: 1.443, 5.771, 11.542, 17.312$

9.5.1 Static Web Services

All the experiments in this subsection are based on static web services. The section is divided into two parts. The first part describes the averaging of all the trustor's usage experiences. The second part explores the filtering of usage experiences using experience window and the assignment of weights based on usage experience age.

9.5.1.1 Average

Table 7 shows a summary of the experimental results from web service selection. As a web service selection strategy could have numerous configurations, the table only shows the minimum and maximum values for all configurations of a strategy. The columns in the table show the values for the minimum mean experience, the maximum mean experience, the minimum percentage of positive experiences and the maximum percentage of positive experiences.

Table 7: Result Summary (Static Web Services, Average: $window_{exp} = \infty$)

	exp_{min}	exp_{max}	$+exp_{min}$	$+exp_{max}$
Random	-0.007	-0.007	69.26%	69.26%
ϵ-greedy	0.399	0.640	94.00%	97.26%
Boltzmann	0.223	0.722	94.00%	99.34%

The experimental results show that the random strategy performs significantly worse than the ϵ -greedy exploration strategy and the Boltzmann exploration strategy. This demonstrates the effectiveness of computational trust calculated from experiences. Between ϵ -greedy and Boltzmann, the experimental results are not as clear cut. Both strategies produce a high percentage of positive experiences. As for mean experience, ϵ -greedy has the better exp_{min} value while Boltzmann has the better exp_{max} value. This is due to the fact that Boltzmann is a more complicated strategy that when configured properly could be better than ϵ -greedy (i.e., exp_{max}) but when misconfigured could be worse than ϵ -greedy (i.e., exp_{min}). As an example, a good configuration is when $exp_{exploration} = 0.5$, $dr = 0$ and $temp_f = 0.01$. With this configuration, exploration could continue until the web service with $exp_{usage} = 0.8$ is found. After that, only exploitation could take place which leads to exp_{max} of Boltzmann. An example of a bad configuration is when $exp_{exploration} = -0.1$, $dr = 0$ and $temp_f = 0.01$. With this configuration, as long there is a web service to exploit, exploitation would be conducted. As a result, after the first exploration, all that would happen is the exploitation of the first explored web service. This is undesirable and would lead to exp_{min} of Boltzmann.

9.5.1.2 Experience Window and Weights

When experience window is applied to computational trust formation, a decrease in window size has been shown to have a negative impact on web service selection. An example is shown in Table 8. With all of its old experiences discarded, a known web service may be treated as an unknown web service. This means that the known web service may be selected during exploration. As a web service's usage experiences never changes, exploring a web service that has already been explored would only lead to worse usage experiences.

Table 8: Result Summary (Static Web Services, Average: $window_{exp} = 5$)

	exp_{min}	exp_{max}	$+exp_{min}$	$+exp_{max}$
ϵ-greedy	-0.006	0.464	69.36%	97.20%
Boltzmann	0.223	0.699	81.20%	99.34%

Weights when applied to computational trust formation should have no effect on the experimental results. This is due the fact that usage experiences from a web service never changes. Therefore, the observations made in Section 9.5.1.1 are applicable to weighted average.

9.5.2 Fluctuating Web Services

All the experiments in this subsection are based on fluctuating web services. Unless otherwise specified, the web services are assumed to have $prob_f = 0.1$ and $duration_f = 2$. This is a pessimistic assumption as it assumes a 10% chance of service fluctuation. The section is divided into two parts. The first part describes the averaging of all the trustor’s usage experiences. The second part explores the filtering of usage experiences using experience window and the assignment of weights based on usage experience age.

9.5.2.1 Average

Table 9 shows a summary of the experimental results from web service selection. The results demonstrated that computational trust calculated from experiences is effective in improving the selection of fluctuating web services. However when compared to Table 7, the values for the metrics are lower. This is due to the fact that the usage experiences obtained during fluctuation are not representative of the web service’s default usage experience. As a result, if a fluctuation is in the positive direction, this may cause the trustor to overestimate a web service. If a fluctuation is in the negative direction, this may cause the trustor to avoid a web service that should have been selected for invocation.

Table 9: Result Summary (Fluctuating Web Services, Average: $window_{exp} = \infty$)

	exp_{min}	exp_{max}	$+exp_{min}$	$+exp_{max}$
Random	-0.014	-0.014	66.54%	66.54%
ϵ-greedy	0.361	0.514	88.72%	91.20%
Boltzmann	0.284	0.567	88.42%	92.84%

To examine how changes to $prob_f$ and $duration_f$ influences web service selection, two additional experiments have been conducted. Representative results are

seen in Table 10 and Table 11. Table 10 shows the experimental results when there is an increase in $prob_f$. Table 11 shows the experimental results when there is an increase in $duration_f$. In both cases, the experimental results are lower than those in Table 9. This is due to the fact that an increase in $prob_f$ would cause more fluctuation while an increase in $duration_f$ would cause longer fluctuation.

Table 10: Result Summary (Fluctuating Web Services: $prob_f = 0.2$, $duration_f = 2$, Average: $window_{exp} = \infty$)

	exp_{min}	exp_{max}	$+exp_{min}$	$+exp_{max}$
Random	-0.002	-0.002	65.38%	65.38%
ϵ-greedy	0.311	0.402	84.60%	86.44%
Boltzmann	0.254	0.455	83.82%	88.14%

Table 11: Result Summary (Fluctuating Web Services: $prob_f = 0.1$, $duration_f = 4$, Average: $window_{exp} = \infty$)

	exp_{min}	exp_{max}	$+exp_{min}$	$+exp_{max}$
Random	-0.009	-0.009	65.78%	65.78%
ϵ-greedy	0.313	0.433	86.32%	89.08%
Boltzmann	0.220	0.480	86.38%	91.08%

9.5.2.2 Experience Window and Weights

When experience window is applied to computational trust formation, a decrease in window size has been shown to have a negative impact on web service selection. An example is shown in Table 12. In the table, the values of the different metrics all are lower than in Table 9 with the only exception being the exp_{min} of Boltzmann. In this particular case, the value in Table 9 is due to a bad configuration which led to a lack of exploration. With a smaller experience window, the calculated computational trust becomes more sensitive to fluctuation which in turn led to more exploration. The increase in exploration led to improvement in exp_{min} . As for the reasoning for the negative impact on web service selection, this is due to the fact that fluctuations are rare and that the exploration of already explored web services in general is not worth the risk.

Table 12: Result Summary (Fluctuating Web Services, Average: $window_{exp} = 5$)

	exp_{min}	exp_{max}	$+exp_{min}$	$+exp_{max}$
ϵ-greedy	0.002	0.367	67.80%	90.80%
Boltzmann	0.310	0.464	81.82%	92.18%

When weights are applied to computational trust formation, this has been shown to have a negative impact on mean experience but a positive impact on the percentage of positive experiences. An example is shown in Table 13. By incorporating weights, computational trust becomes more sensitive to usage experience changes. The increase in sensitivity may cause more mistrusting during fluctuation which in turn causes the drop in values of the exp metric. However, fluctuation could last for more than one time unit. If the fluctuation produces a negative usage experience, the increase sensitivity could cause the trustor to switch to a different web service. The trustor therefore would not have to experience the rest of the negative usage experiences. This in turn improves the values of the $+exp$ metric.

Table 13: Result Summary (Fluctuating Web Services, Weighted Average: $\lambda = 1.443$)

	exp_{min}	exp_{max}	$+exp_{min}$	$+exp_{max}$
ϵ-greedy	0.284	0.407	89.40%	91.46%
Boltzmann	0.235	0.403	89.24%	93.02%

9.5.3 Dynamic Web Services

All the experiments in this subsection are based on dynamic web services. Unless otherwise specified, the web services are assumed to have $prob_d = 0.04$. The $prob_d$ is chosen to be less than $prob_f$ as it is assumed that dynamic behavior changes occur less frequently than fluctuations. The section is divided into two parts. The first part describes the averaging of all the trustor's usage experiences. The second part explores the filtering of usage experiences using experience window and the assignment of weights based on usage experience age.

9.5.3.1 Average

Table 14 shows a summary of the experimental results from web service selection. The results demonstrated that computational trust calculated from experiences is effective in improving the selection of dynamic web services. However when compared to Table 7, the values for the metrics are lower. This is due to the fact that changes in a web service's default usage experience could cause the calculated computational trust to be outdated by the time of web service selection.

Table 14: Result Summary (Dynamic Web Services, Average: $window_{exp} = \infty$)

	exp_{min}	exp_{max}	$+exp_{min}$	$+exp_{max}$
Random	-0.018	-0.018	58.00%	58.00%
ϵ-greedy	0.318	0.416	85.54%	87.36%
Boltzmann	0.234	0.493	84.62%	90.60%

To examine how changes to $prob_d$ influences web service selection, an experiment is conducted with $prob_d$ increased to 0.08. The experimental results are shown in Table 15. The results show that the increase in dynamism could have negative impact on web service selection.

Table 15: Result Summary (Dynamic Web Services: $prob_d = 0.08$, Average: $window_{exp} = \infty$)

	exp_{min}	exp_{max}	$+exp_{min}$	$+exp_{max}$
Random	0.005	0.005	56.98%	56.98%
ϵ-greedy	0.267	0.336	80.80%	82.32%
Boltzmann	0.212	0.404	77.94%	86.00%

9.5.3.2 Experience Window and Weights

When experience window is applied to computational trust formation, a decrease in window size has been shown to have a positive impact on web service selection. An example is shown in Table 16. In the table, the values of the different metrics all performed better than Table 14 with the exceptions being exp_{min} and $+exp_{min}$ of ϵ -

greedy. In this particular case, the parameter dr of ϵ -greedy is set to 0, therefore preference would always be given to exploration. With $window_{exp} = 5$, this means that there is always unknown web services to explore (due to discarding of old experiences). As a result, the strategy would always explore and never exploit leading to lower experimental results. As for the reasoning for the positive impact on web service selection, this is due to the fact that a web service's default usage experience may change over time. As a result, it is worthwhile to re-explore web services to see if anything has changed from the last exploration.

Table 16: Result Summary (Dynamic Web Services, Average: $window_{exp} = 5$)

	exp_{min}	exp_{max}	$+exp_{min}$	$+exp_{max}$
ϵ-greedy	-0.007	0.422	58.72%	92.74%
Boltzmann	0.295	0.532	86.28%	95.54%

When weights are applied to computational trust formation, this has been shown to have a positive impact on both mean experience and percentage of positive experiences. An example is shown in Table 17. By incorporating weights, computational trust becomes more sensitive to usage experience changes. The increase in sensitivity allows the trustor to more quickly switch web services when faced with dynamism.

Table 17: Result Summary (Dynamic Web Services, Weighted Average: $\lambda = 1.443$)

	exp_{min}	exp_{max}	$+exp_{min}$	$+exp_{max}$
ϵ-greedy	0.380	0.463	91.72%	94.74%
Boltzmann	0.317	0.547	91.64%	96.42%

9.5.4 Fluctuating-Dynamic Web Services

All the experiments in this subsection are based on FD web services. Unless otherwise specified, the web services are assumed to have $prob_f = 0.1$, $duration_f = 2$ and $prob_d = 0.04$. This assumption is based on the parameter assumptions in Section 9.5.2 and Section 0. The section is divided into two parts. The first part describes the averaging of all the trustor's usage experiences. The second part explores the filtering of usage

experiences using experience window and the assignment of weights based on usage experience age.

9.5.4.1 Average

Table 18 shows a summary of the experimental results from web service selection. The results demonstrated that computational trust calculated from experiences is effective in improving the selection of FD web services. However when compared to Table 7, the values for the metrics are lower. This is due to the fact that fluctuation and dynamism both can negatively impact web service selection.

Table 18: Result Summary (FD Web Services, Average: $window_{exp} = \infty$)

	exp_{min}	exp_{max}	$+exp_{min}$	$+exp_{max}$
Random	-0.009	-0.009	58.40%	58.40%
ϵ-greedy	0.250	0.345	80.86%	81.96%
Boltzmann	0.211	0.397	79.48%	85.52%

9.5.4.2 Experience Window and Weights

When experience window is applied to computational trust formation, a decrease in window size has been shown to have a positive impact on web service selection. An example is shown in Table 19. In the table, the values of the different metrics all performed better than Table 18 with the exceptions being exp_{min} and $+exp_{min}$ of ϵ -greedy and $+exp_{min}$ of Boltzmann. In the case of ϵ -greedy, the reasoning is the same as in Section 0. In the case of Boltzmann, the bad configuration causes the strategy to perform too much exploration and not enough exploitation (21 explorations and 29 exploitations). As for the reasoning for the positive impact on web service selection, this is due to the fact that exploration of already explored web services is worthwhile. This is the case with dynamism as shown in Section 0. Although the impact is negative with fluctuation, the negative impact is not big enough to overcome the positive impact of avoiding dynamism.

Table 19: Result Summary (FD Web Services, Average: $window_{exp} = 5$)

	exp_{min}	exp_{max}	$+exp_{min}$	$+exp_{max}$
ϵ-greedy	-0.004	0.350	59.04%	86..44%
Boltzmann	0.277	0.411	77.34%	88.44%

When weights are applied to computational trust formation, this has been shown to have a positive impact on both mean experience and percentage of positive experiences. An example is shown in Table 20. By incorporating weights, computational trust becomes more sensitive to usage experience changes. The increase in sensitivity allows the trustor to more quickly switch web services when faced with dynamism. Although this may lead to unnecessary switches in the case of fluctuation (it is better to wait out the fluctuation that switch web services), the negative impact is not big enough to overcome the positive impact of avoiding dynamism.

Table 20: Result Summary (FD Web Services, Weighted Average: $\lambda = 1.443$)

	exp_{min}	exp_{max}	$+exp_{min}$	$+exp_{max}$
ϵ-greedy	0.288	0.373	86.10%	87.94%
Boltzmann	0.294	0.414	85.52%	89.20%

9.6 Experiments Based on Recommendation

In this section, the testbed is configured with recommender as the only type of evidence source. In each of the subsections, the recommenders are configured differently. The testbed is also configured with static web services. There is no need to experiment with fluctuating, dynamic or fluctuating-dynamic web services. This is due to the fact that recommendation is calculated based on deviation from the trustor's usage experience. As usage experience changes with different web service type, the change would also be reflected in the recommendations. As a result, any observations made concerning a static web service should be applicable to the other web service types as well. This is a weakness of our recommendation definition that should be addressed in future work. In terms of web service selection, only the max-trust strategy is evaluated. This is due to the fact that recommendations are available for each of the web services.

In terms of computational trust formation, it is based on equation (5) of Section 7.1.2. The formation is configured as follows:

- *exploration*: 0, 1, 2, 3, 14
- *exploitation*: 1, 2, 3, 4, 5, 6
- Evidence source assessment (Section 7.2)
 - Evidence assessment
 - *similarity*: 0.2
 - w_{ev} : 1
 - Evidence source trust assessment
 - Cognitive trust formation
 - Average
 - w_{exp} : ∞
 - Average
 - w_{exp} : 5
 - Weighted average
 - λ : 1.443
 - $w_{reliability}$: 0
 - w_{ct} : 1

In equation (5), recommender trust is used as weights for each of the gathered recommendations. As a trustor may not have evidence source trust in all of the recommenders, exploration of recommenders (similar to exploration of web services) is needed. For each recommendation gathering, recommendations are gathered from both unknown recommenders and known recommenders. For example, if *exploration* = 1 and *exploitation* = 2, 1 unknown recommender and 2 known recommenders are contacted for recommendation gathering. The recommendations gathered from unknown recommenders are not used in performance belief formation. The recommendations however are used in evidence source assessment. This in turn allows evidence source

trust to be calculated for unknown recommenders without their recommendations influencing computational trust formation.

As for the formation of evidence source trust, since it is a form of computational trust, it is assumed to be based solely on cognitive trust without considering belief reliability. For simplicity sake, it is assumed that recommendations would be gathered at every time unit. Moreover, the feedback provided would only be used to assess the recommendation gathered in the previous time unit. The calculated recommendation assessments are used in evidence source trust formation. The formation is based on either average, average with experience window or weighted average.

9.6.1 Similar Recommenders

The recommenders in this subsection are configured as in Table 21. In the table, there are four different variations of honest recommenders. The variations are based on either disposition or magnitude. A recommender could either be more optimistic (*deviation* = $[0, 0.2]$) or more pessimistic (*deviation* = $[-0.2, 0]$) than the trustor. The magnitude of deviation could be either 0.1 (*deviation* = $[-0.1, 0.1]$) or 0.2 (*deviation* = $[-0.2, 0.2]$). As *similarity* = 0.2 in the experiments configuration (Section 0), this means that there would be no negative assessment for any recommendation obtained from honest recommenders. These honest recommenders can be thought of as being “similar” to the trustor. In terms of the “bad” recommenders, 2 malicious and 2 oscillating recommenders (minority of recommenders) have been configured.

Table 21: Similar Recommenders (Experiments)

Number of Recommenders	Recommender Type
3	Honest (<i>deviation</i> = $[-0.1, 0.1]$)
3	Honest (<i>deviation</i> = $[-0.2, 0.2]$)
2	Honest (<i>deviation</i> = $[0, 0.2]$)
2	Honest (<i>deviation</i> = $[-0.2, 0]$)
2	Malicious
1	Oscillating (<i>duration_o</i> = 5)
1	Oscillating (<i>duration_o</i> = 10)

As for the rest of this subsection, it is divided into two parts. The first part describes the calculation of evidence source trust through the averaging of all the recommendation assessments. The second part explores the filtering of recommendation assessments using experience window and the assignment of weights based on recommendation age.

9.6.1.1 Average

The relationship between *exp* and *exploitation* for different values of *exploration* are shown in Figure 40. The relationship between *+exp* and *exploitation* for different values of *exploration* are shown in Figure 41. By comparing the experimental results of the max-trust strategy with the experimental results of the random strategy (Table 7 of Section 9.5.1.1), the results demonstrated that computational trust calculated from recommendations (similar recommenders) is effective in improving the selection of static web services.

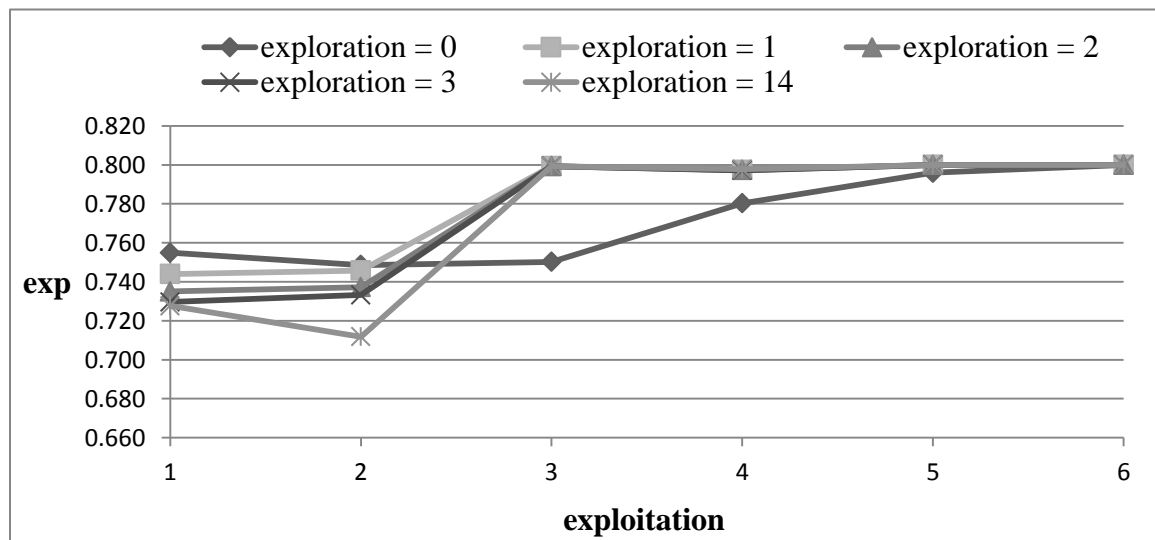


Figure 40: Mean Experience (Similar Recommenders, Average: $window_{exp} = \infty$)

The figures also suggest that an increase in exploitation has a positive effect on mean experience and percentage of positive experiences. This is due to the fact that most recommenders are honest and similar to the trustor. An increase in exploitation allows the gathered honest recommendations to overwhelm the recommendations provided by the

“bad” recommenders. The only exception in the figures is when *exploitation* = 2. In this case, the overwhelm scenario is impossible when a “good” recommendation and a “bad” recommendation cancel each other out. As for exploration, the figures suggested that exploration is always preferable over no exploration. However, the impact of increase in exploration is small. This is due to the fact that most recommenders are honest and similar to the trustor. Therefore, similar recommenders can be easily located even with *exploration* = 1.

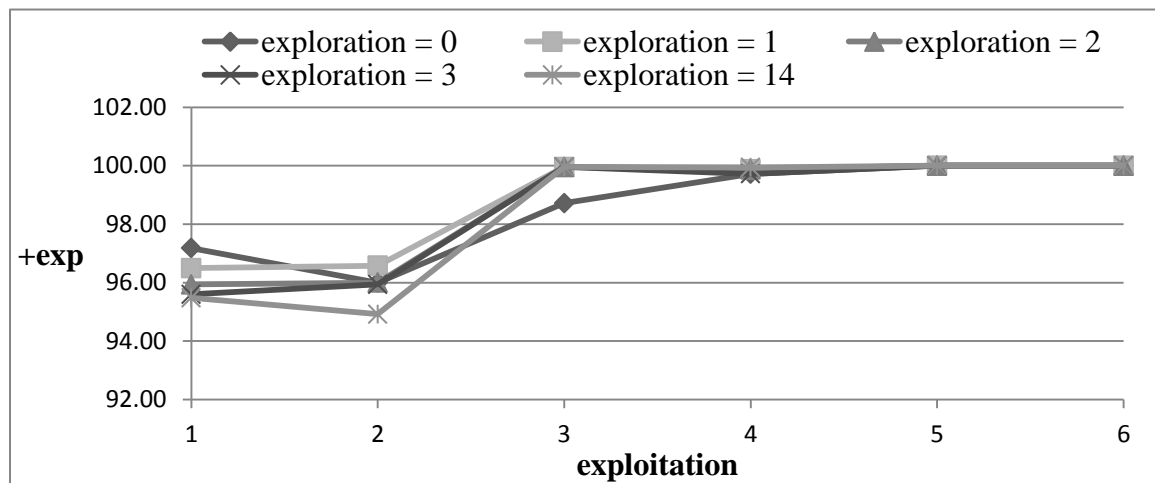


Figure 41: Percentage of Positive Experiences (Similar Recommenders, Average: $window_{exp} = \infty$)

9.6.1.2 Experience Window and Weights

When experience window is applied to evidence source trust assessment, a decrease in window size has been shown to have a negative impact on web service selection. The negative impact only applies to small number of exploitations. This is due to the fact that the positive impact of exploitation is able to overwhelm the negative impact of smaller experience window. An example is shown in Figure 42 and Figure 43. The negative impact of experience window is due to the fact that for the majority of recommenders, there is no need to reevaluate evidence source trust.

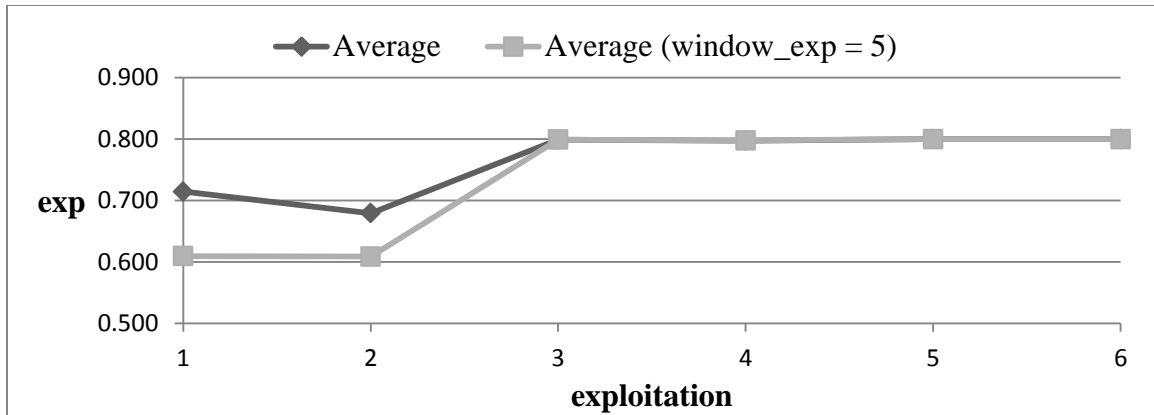


Figure 42: Mean Experience (Similar Recommenders, $exploration = 14$, $window_{exp} = 5$)

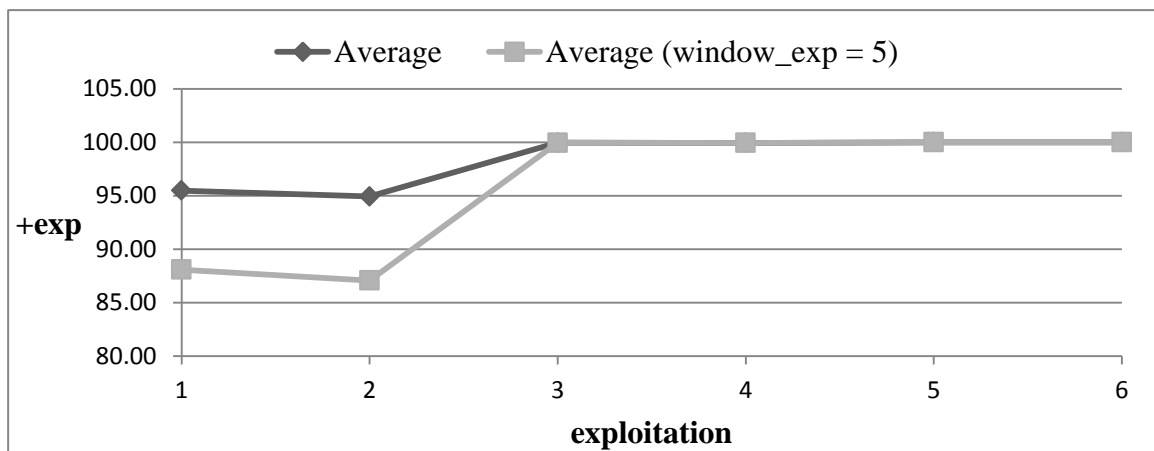


Figure 43: Percentage of Positive Experiences (Similar Recommenders, $exploration = 14$, $window_{exp} = 5$)

When weights are applied to evidence source trust assessment, this has been shown to have a positive impact on both mean experience and percentage of positive experiences. The positive impact only applies to web service selection with small number of exploitation. This is due to the fact that the positive impact of exploitation is able to overwhelm the positive impact of weighted average. An example is shown in Figure 44 and Figure 45. By incorporating weights, evidence source trust becomes more sensitive to assessment changes. The increase in sensitivity allows the trustor to more quickly filter out oscillating recommenders.

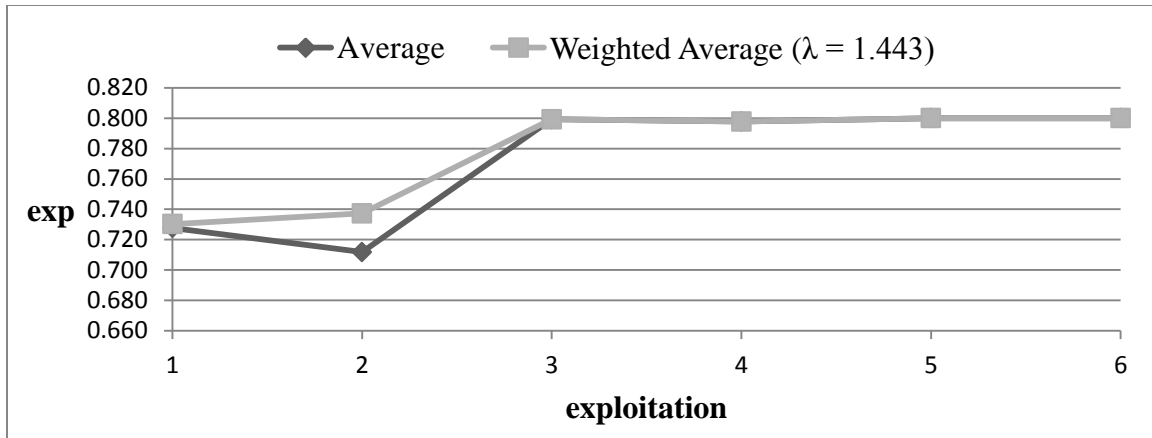


Figure 44: Mean Experience (Similar Recommenders, $exploration = 14$, $\lambda = 1.443$)

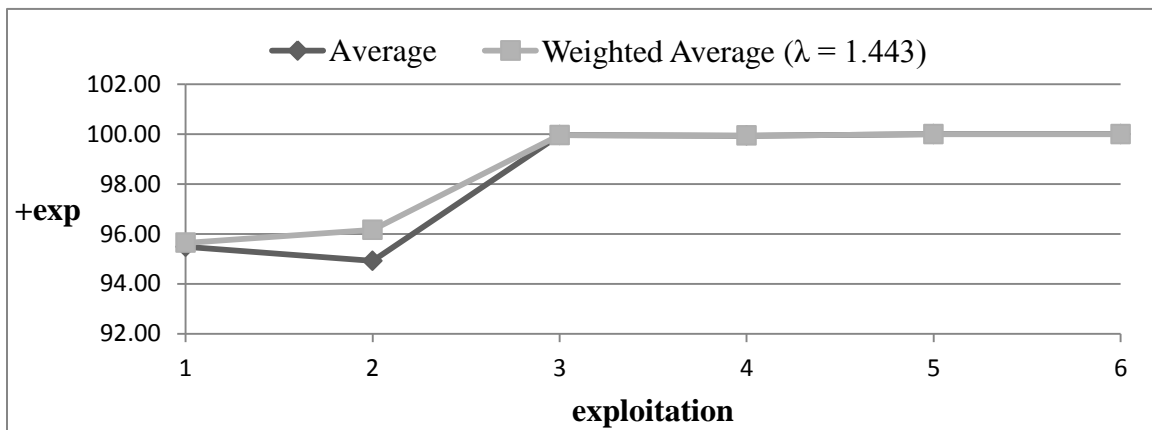


Figure 45: Percentage of Positive Experiences (Similar Recommenders, $exploration = 14$, $\lambda = 1.443$)

9.6.2 Dissimilar Recommenders

The recommenders in this subsection are configured as in Table 22. In the table, four new variations of honest recommender have been introduced. All four variations have *deviation* that are greater than *similarity*. This means that the recommendations gathered from these recommenders could potentially be assessed negatively. These honest recommenders can be thought of as being “dissimilar” to the trustor. Moreover, there are more dissimilar recommenders than similar recommenders.

Table 22: Dissimilar Recommenders (Experiments)

Number of Recommenders	Recommender Type
1	Honest (<i>deviation</i> = [-0.1, 0.1])
1	Honest (<i>deviation</i> = [0, 0.2])
1	Honest (<i>deviation</i> = [-0.2, 0])
3	Honest (<i>deviation</i> = [-0.9, 0.9])
2	Honest (<i>deviation</i> = [-1.0, 1.0])
1	Honest (<i>deviation</i> = [-0.8, 1.0])
1	Honest (<i>deviation</i> = [-1.0, 0.8])
2	Malicious
1	Oscillating (<i>duration_o</i> = 5)
1	Oscillating (<i>duration_o</i> = 10)

9.6.2.1 Average

The relationship between *exp* and *exploitation* for different values of *exploration* are shown in Figure 46. The relationship between *+exp* and *exploitation* for different values of *exploration* are shown in Figure 47. By comparing the experimental results of the max-trust strategy with the experimental results of the random strategy (Table 7 of Section 9.5.1.1), the results demonstrated that the computational trust calculated from recommendations (dissimilar recommenders) is effective in improving the selection of static web services. When the results are compared to Figure 40 and Figure 41 however, the introduction of dissimilar recommenders is shown to have a negative impact on web service selection.

Figure 46 suggests that an increase in exploitation has a mixed effect on mean experience. This is due to the fact that a dissimilar recommender has the potential for leading or misleading the trustor. As a result, an increase in exploitation could potentially improve or deteriorate mean experience. Figure 46 suggests that an increase in exploitation has a positive effect on percentage of positive experiences. This is due to the fact that the recommendations provided by dissimilar recommenders generally are of the same sign as that of a web service's usage experience. For example, if a web service has $exp_{usage} = 0.8$, for recommendation to be negative (i.e., of different sign), a dissimilar

recommender needs to have $deviation < -0.8$. As deviation is randomly generated, this is going to rarely occur when compared to the case of the recommendation remaining positive. The same reasoning could also apply to $exp_{usage} = -0.8$ and to lesser extents $exp_{usage} = 0.4$ and $exp_{usage} = -0.4$. As a result, even when recommendations from dissimilar recommenders dominate, the calculated computational trust is still likely to result in the selection of a web service with positive usage experience.

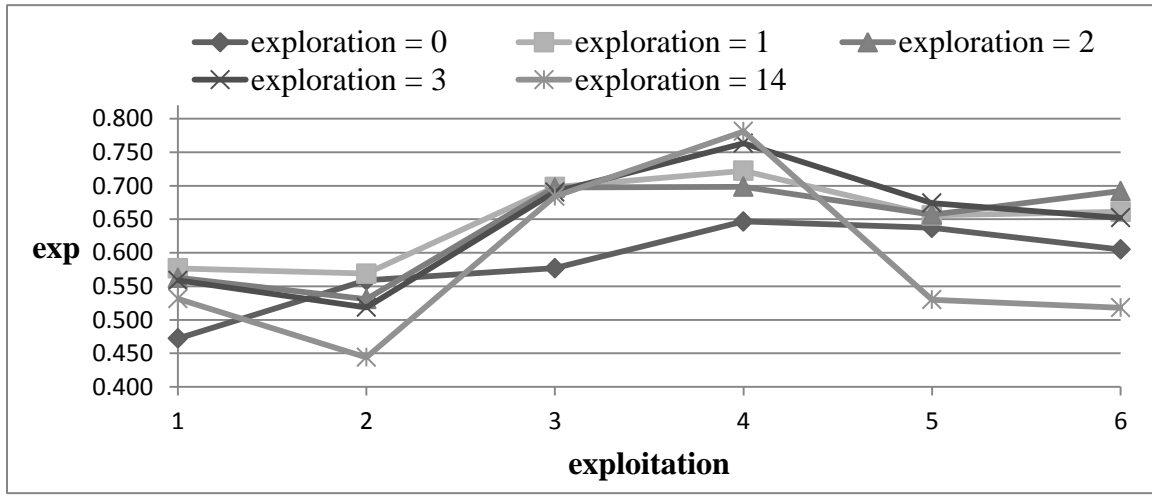


Figure 46: Mean Experience (Dissimilar Recommenders, Average: $window_{exp} = \infty$)

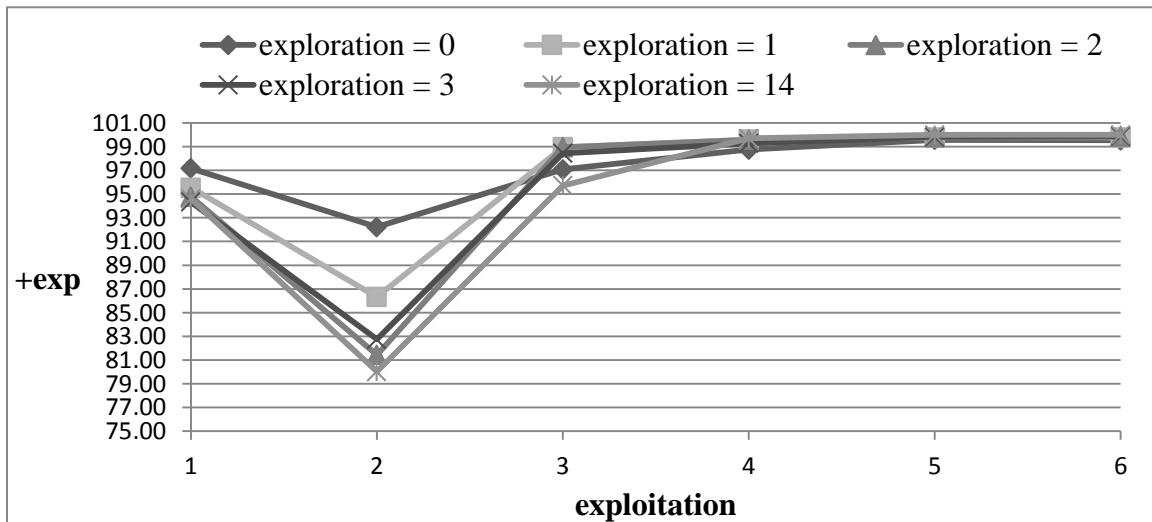


Figure 47: Percentage of Positive Experiences (Dissimilar Recommenders, Average: $window_{exp} = \infty$)

Figure 46 and Figure 47 suggest that an increase in exploration has a mixed effect on mean experience and percentage of positive experiences. Given that dissimilar recommenders are the majority and that they could either lead or mislead the trustor, more exploration therefore could either improve or deteriorate web service selection.

9.6.2.2 Experience Window and Weights

When experience window is applied to evidence source trust assessment, a decrease in window size has been shown to have a negative impact on web service selection. The negative impact only applies to web service selection with small number of exploitations. This is due to the fact that the mixed impact of exploitation is able to overwhelm the negative impact of smaller experience window. An example is shown in Figure 48 and Figure 49. The negative impact of experience window is due to the fact that for the majority of recommenders (similar, dissimilar and malicious recommenders), there is no need to reevaluate evidence source trust.

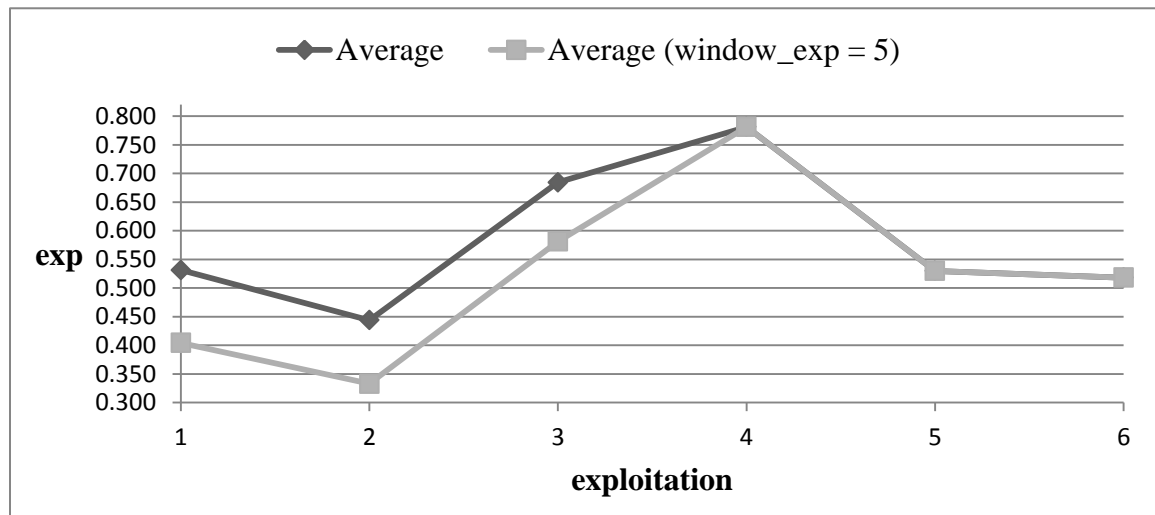


Figure 48: Mean Experience (Dissimilar Recommenders, $exploration = 14$, $window_{exp} = 5$)

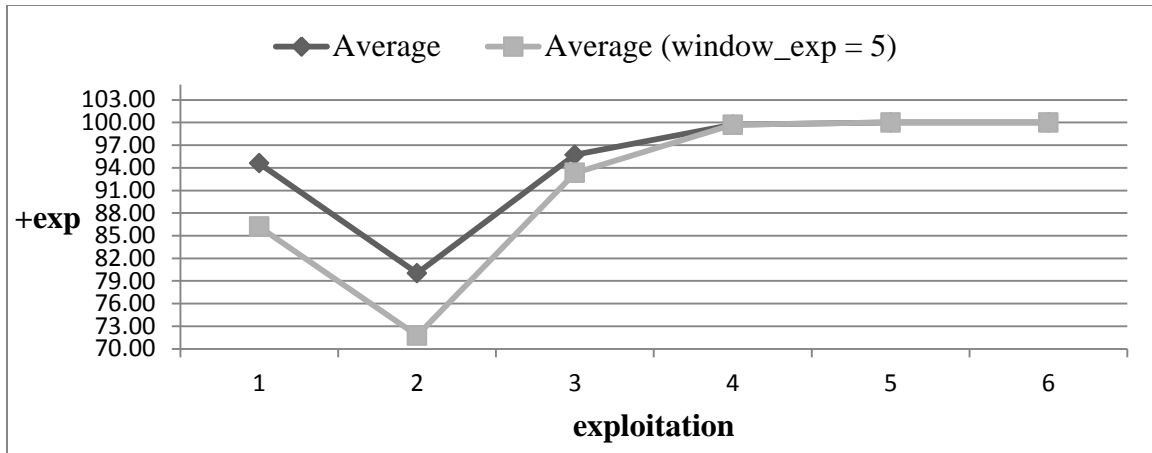


Figure 49: Percentage of Positive Experiences (Dissimilar Recommenders, $exploration = 14$, $window_{exp} = 5$)

When weights are applied to evidence source trust assessment, this has been shown to have a positive impact on both mean experience and percentage of positive experiences. The positive impact only applies to web service selection with a small number of exploitations. This is due to the fact that the mixed impact of exploitation is able to overwhelm the positive impact of weighted average. An example is shown in Figure 50 and Figure 51. By incorporating weights, evidence source trust becomes more sensitive to assessment changes. The increase in sensitivity allows the trustor to more quickly filter out dissimilar recommenders and oscillating recommenders.

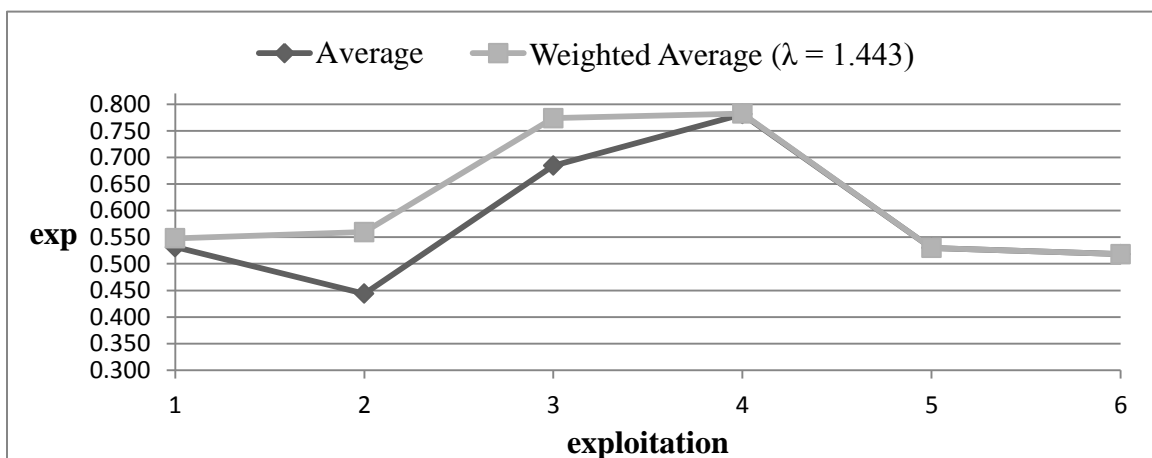


Figure 50: Mean Experience (Dissimilar Recommenders, $exploration = 14$, $\lambda = 1.443$)

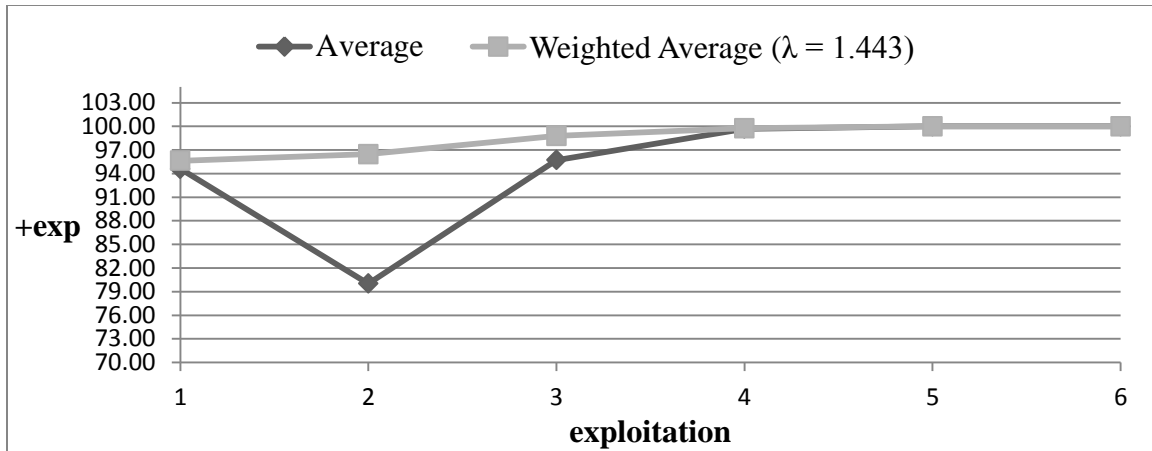


Figure 51: Percentage of Positive Experiences (Dissimilar Recommenders, $exploration = 14$, $\lambda = 1.443$)

9.7 Experiments Based on Experience, Recommendation

In this section, the testbed is configured with two types of evidence sources: trustor and recommender. In each of the subsections, the evidence sources are configured differently. The testbed is also configured with static web services. As the experiments are designed to evaluate whether experience and recommendation complement each other in computational trust formation, any observations made concerning static web services should be applicable to the other web service types as well. In terms of web service selection, only the max-trust strategy is evaluated. This is due to the fact that exploration is not needed due to the existence of recommendations.

Computational trust formation is based on applying weighted average to experience-based performance belief and recommendation-based performance belief. Experience-based performance belief is assigned a weight of 0.7. Recommendation-based performance belief is assigned a weight of 0.3. The weight assignment is based on the observation that the trustor would always have its own best interest in mind while that may not be the case with recommenders. If the trustor does not have experiences with a web service, computational trust formation would be based solely on recommendation-based performance belief. In terms of belief formation, experience-based performance belief is based on the averaging of all the trustor's usage experiences. As for

recommendation-based belief, evidence source trust formation is based on the averaging of all the calculated recommendation assessments. The experimental results obtained from evidence averaging should be applicable to cases of experience window and weighted average.

9.7.1.1 Trustor and Similar Recommenders

An example of the relationship between *exp* and *exploitation* for experience, recommendation and experience and recommendation are shown in Figure 52. An example of the relationship between *+exp* and *exploitation* for experience, recommendation and experience and recommendation are shown in Figure 53.

In the figures, experience is based on exp_{max} and $+exp_{max}$ of Boltzmann in Table 7 of Section 9.5.1.1. This is the best experimental results based on experience. Recommendation is based on Figure 40 and Figure 41 with $exploration = 14$. The experimental results in the figures demonstrated that in most cases computational trust calculated from experiences and recommendations (similar recommenders) can improve on the experimental results of computational trust calculated from either experiences or recommendations (similar recommenders) in the selection of static web service. The only exception is when $exploitation = 1$ in Figure 53. In this case, if the recommender is an oscillating recommender, it could cause the trustor to try out web services with negative usage experiences.

As for the reasoning for the improvement over experience-based computational trust, this is due to the fact that exploration is no longer random but instead is directed by the calculated recommendation-based performance belief. The improvement over recommendation-based computational trust is due to the fact that experience-based performance belief is given more weight and in some cases can help mitigate when recommendation-based performance belief is dominated by misleading recommendations.

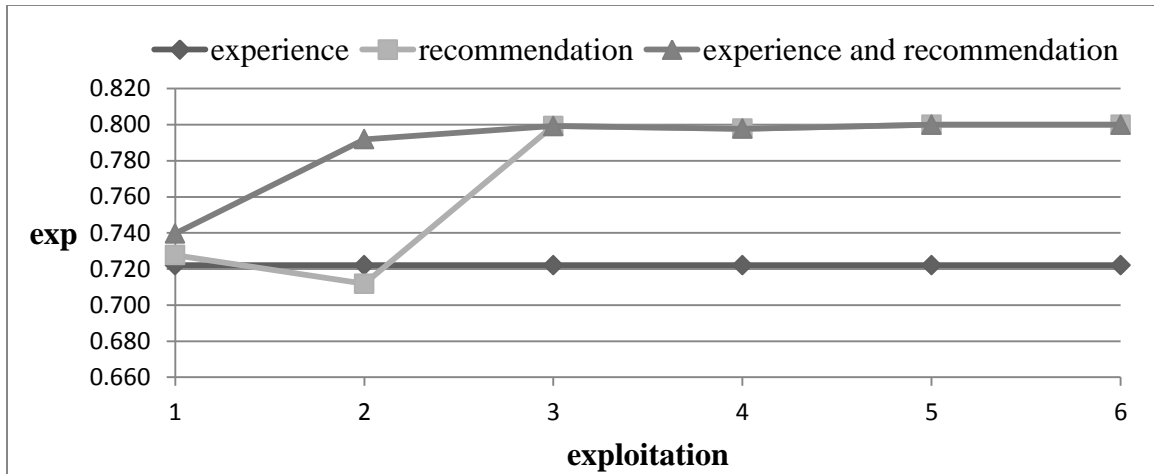


Figure 52: Mean Experience (Trustor and Similar Recommenders, Average: $exploration = 14, window_{exp} = \infty$)

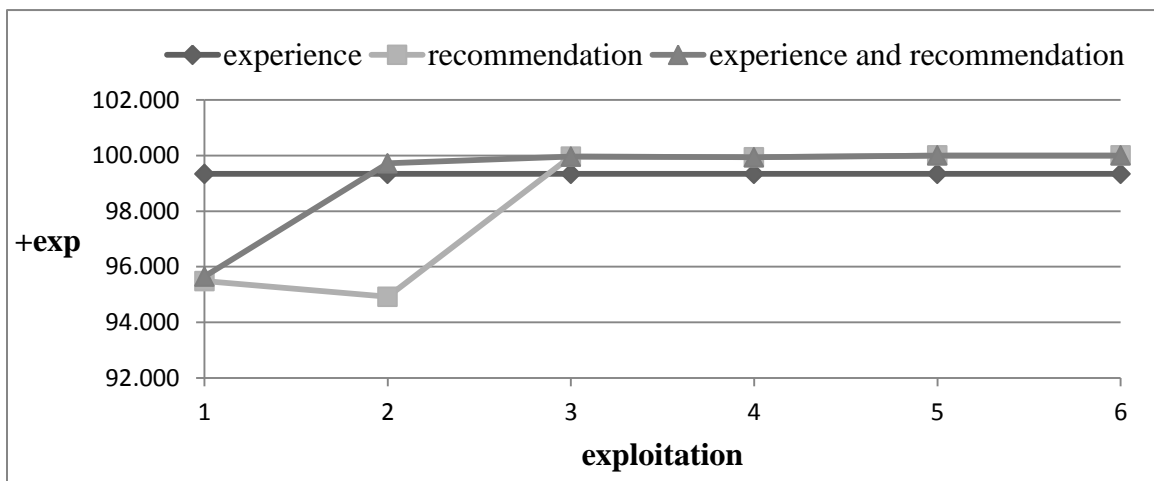


Figure 53: Percentage of Positive Experiences (Trustor and Similar Recommenders, Average: $exploration = 14, window_{exp} = \infty$)

9.7.1.2 Trustor and Dissimilar Recommenders

An example of the relationship between exp and $exploitation$ for experience, recommendation and experience and recommendation are shown in Figure 54. In the figure, the experimental results demonstrated that for small number of exploitation, computational trust calculated from experiences and recommendations (dissimilar recommenders) can improve on the experimental results of computational trust calculated

from either experiences or recommendations (dissimilar recommenders) in the selection of static web service. However, as exploitation increases, recommendation-based performance belief starts to coalesce around a single web service. Since the web service could be the recommendation of dissimilar recommenders, the mean experience ends up being lower than when computational trust is calculated from experiences.

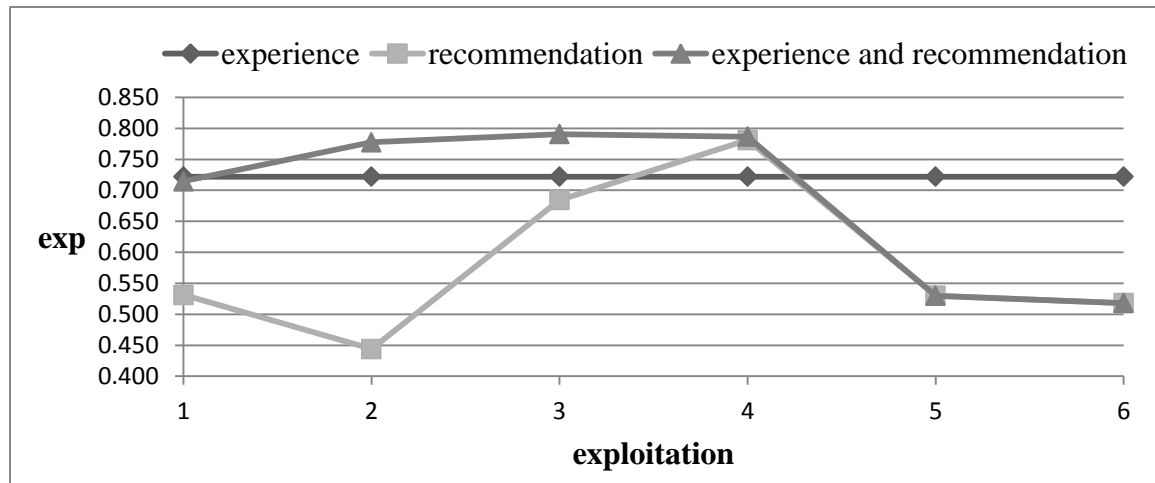


Figure 54: Mean Experience (Trustor and Dissimilar Recommenders, Average: $exploration = 14, window_{exp} = \infty$)

An example of the relationship between $+exp$ and $exploitation$ for experience, recommendation and experience and recommendation are shown in Figure 55. In the figure, the experimental results demonstrated that in most cases computational trust calculated from experiences and recommendations (dissimilar recommenders) can improve on the experimental results of computational trust calculated from either experiences or recommendations (dissimilar recommenders) in the selection of static web service. The only exception is when $exploitation = 1$. In this case, if the recommender is an oscillating recommender, it could cause the trustor to try out web services with negative usage experiences. As the recommendations provided by recommenders are usually of the same sign as that of the recommended web service's usage experience, experience-based computational trust is improved upon due to the trustor seldom having to invoke web service with negative usage experience. As for recommendation-based computational trust, it is improved upon due to the mitigating effect of experience-based

performance belief on recommendation-based performance belief that is dominated by misleading recommendations.

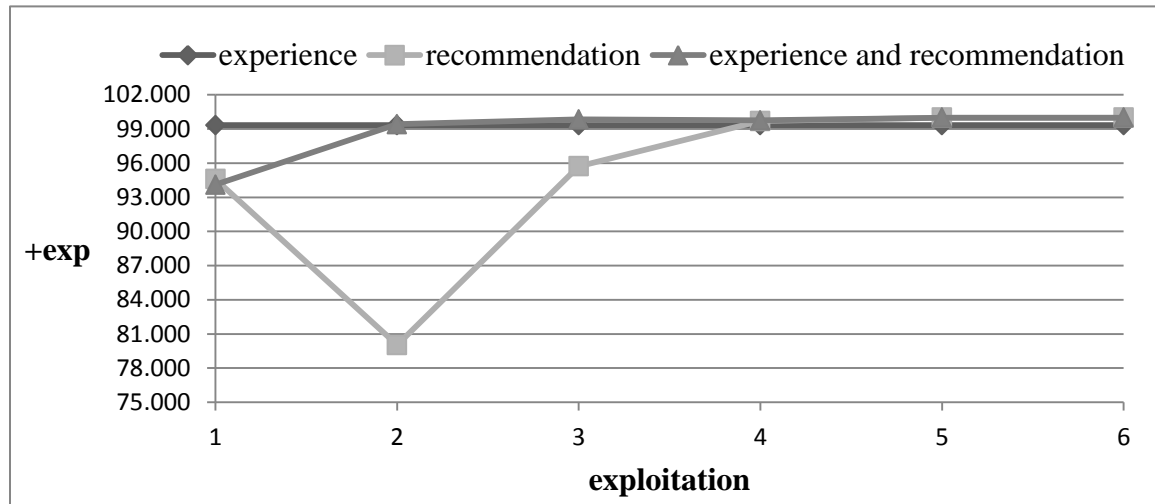


Figure 55: Percentage of Positive Experiences (Trustor and Dissimilar Recommenders, Average: $exploration = 14$, $window_{exp} = \infty$)

9.8 Conclusions and Discussions

The experiments have demonstrated the following:

- *Evidence source type and behavior plays a key role in computational trust formation.* With only access to experiences, the trustor has to depend on exploration during web service selection. Too much exploration or too much exploitation is both undesirable. The optimal amount of exploration and exploitation is dependent on the available web services. With only access to recommendations, the trustor's performance in web service selection is dependent on the composition of the recommenders. If most of the recommenders are similar to the trustor, recommender exploration and exploitation is desirable. This is not the case when most of the recommenders are dissimilar to the trustor. Although experience and recommendation when used together in most cases can improve web service selection. However, performing exploration based solely on recommendations as in Section 9.7 may not be the best possible solution. Some randomness may be needed to prevent the trustor from being misled by

recommendations and end up only exploiting a single web service. As evidence plays an important role in computational trust formation, middleware support for evidence gathering is provided in the computational trust architecture in the form of the Evidence Gathering Service.

- *Effectiveness of different computational trust formation algorithms.* The experiments showed that weighted average is a relatively effective approach to computational trust formation. Weighted average works well with web services that are static, dynamic and fluctuating-dynamic. During evidence source trust formation, weighted average is also an improvement over average. The only case for which weighted average does not work well is with fluctuating web services. Since fluctuating web services only fluctuate for a small amount of time, the best strategy is to wait out the fluctuation as opposed to switching to a different web service. There is a tradeoff between switching web services too early in cases of fluctuation vs. switching web services too late in cases of dynamism. This is an area that requires further investigation.

As for experience window, the experiments showed that it can improve web service selection in cases of dynamic and fluctuating-dynamic web services. However, experience window when applied to average also performs worse in all the other cases. Therefore, experience window should not be used for short term evidence filtering. Instead, experience window should be in long term filtering to limit the amount of evidence to be considered during belief formation. Identifying the appropriate window to use is an area that requires further investigation.

9.9 Summary

The experiments in this chapter are designed with two goals in mind. The first goal is to evaluate the role that evidence source availability plays in computational trust formation. The second goal is to investigate how different computational trust formation algorithms performed. Both goals are examined through a series of experiments. The experiments are conducted using an experimental testbed. Factors including web services, evidence sources and web service selection strategies are varied during the experiments. The

experimental results have demonstrated the importance of evidence source availability to the quality of the formed computational trust. The results have also demonstrated that weighted average is a relatively better algorithm than average and average with experience window.

Chapter 10

10 Scenarios

The focus of this chapter is to demonstrate the ability of the computational trust architecture (SCOUT and Trust Calculator) to support computational trust formation in different scenarios. This chapter is divided into two sections. The first section describes a movie scenario. The second section describes a web service scenario.

10.1 Movie Scenario

The first part of this section provides a general overview of the movie scenario. The scenario's implementation is described in the second part of this section.

10.1.1 Overview

In the movie scenario, the movie selector uses an application to view the available movies and to rank the movies based on computational trust. Before ranking, the application first needs to instantiate the Trust Calculator. The instantiated Trust Calculator can be configured by calling the *setApplicationFactors* method (Section 6.3) with the following input parameters:

- *application factors*
 - name = "DecisionType", value = "MovieSelection"

The method invocation informs the Trust Calculator that it is calculating computational trust for movie selection. Although "DecisionType" is a decision factor, it is passed to the Trust Calculator as an application factor. This is due to the fact that "DecisionType" does not change across decisions. There is no point in passing "DecisionType" for every computational trust formation. To calculate computational trust in a movie, the application calls the *calculateTrust* method (Section 6.3) of the Trust Calculator. An example invocation could have the following input parameters:

- *trustee*
 - id = "Zombieland"

- type = “Movie/Horror”
- *decision factors* = null

The invocation would cause the Trust Calculator to invoke the TcPlanner.

The *XMLTcPlanner* uses the mappings in Figure 35 of Section 8.2.1 to select the TcPlan needed for computational trust calculation. The TcPlanner retrieves the TcTemplate named “MovieSelection” in Figure 27 of Section 6.2. The TcPlanner then parameterizes the TcTemplate with the parameters in the mappings. The resulting TcPlan is shown in Figure 25 of Section 6.2. The tree constructed from the TcPlan is graphically represented in Figure 26 of Section 6.2. Computational trust is calculated as the weighted average of cognitive trust and emotional trust. Cognitive trust is calculated based on the selector’s quality belief. The formed TcPlan is returned to the Trust Calculator.

After TcPlan formation, the Trust Calculator invokes the TcEngine to execute the TcPlan. The selector’s quality belief in a movie can be obtained by calling the *getAggregateBelief* method of BFS with the following input parameters:

- *trustee*
 - id = “Zombieland”
 - type = “Movie/Horror”
- *aggregate belief*
 - type = “Quality”
- *hints* = null

The BF-Policy responsible for calculating quality belief in a movie is shown in Figure 56. In the figure, the Belief Engine identified as “SCOUT-Reputation” is selected to perform belief calculation (line 5). “SCOUT-Reputation” implements the belief formation algorithm described in Section 7.1.3. The configuration of the Belief Engine (lines 6-8) is mapped to the algorithm as follows:

- FeedbackThreshold \rightarrow *threshold_{fdk}*

- ReputationCountWeight $\rightarrow w_c$
- ReputationThreshold $\rightarrow threshold_{rep}$

The configuration values are determined by the movie selector during BF-Policy creation. All SCOUT policies need to be created before computational trust calculation. The last step in the BF-Policy is to invoke the Belief Engine to perform belief calculation.

```

1  when
2      trustee: Trustee( type matches "Movie.*" )
3      aggBelief: AggregateBelief( type == "Quality" )
4  then
5      beliefEngine = Registry.lookup("SCOUT-Reputation");
6      beliefEngine.set("FeedbackThreshold", 10);
7      beliefEngine.set("ReputationCountWeight", 0.3);
8      beliefEngine.set("ReputationThreshold", 2);
9
10     belief = new Belief(aggBelief.getType());
11     beliefEngine.calculateBelief(trustee, belief);
12 end

```

Figure 56: BF-Policy (Movie)

The Belief Engine calculates the selector’s quality belief in a movie based on the reputations stored in the Evidence Repository. To maximize belief reliability, if the number of reputations available is less than the “ReputationThreshold” in Figure 56, the Belief Engine invokes the *gatherEvidence* method of EGS with the following input parameters to gather more reputation values:

- *trustee*
 - id = “Zombieland”
 - type = “Movie/Horror”
- *belief*
 - type = “Quality”
- *hints*
 - name = “EvidenceType”, value = “Reputation”

The EG-Policy responsible for the gathering of quality reputations of a movie is shown in Figure 57. In the figure, the “BroadcastStrategy” is invoked. This means that all Evidence Gatherers that can gather movie’s quality reputations are invoked to perform reputation gathering. The gathered reputations are mapped to the interval of [-1, 1] by the Evidence Handlers. The mapped reputations are stored in the Evidence Repository to be used in belief calculation.

```

1  when
2    trustee: Trustee( type matches "Movie.*" )
3    belief: Belief( type == "Quality" )
4    hint: Hint( name == "EvidenceType" && value == "Reputation" )
5  then
6    strategy = new BroadcastStrategy();
7    strategy.set("EvidenceType", hint.getValue());
8    strategy.execute(trustee, belief);
9  end

```

Figure 57: EG-Policy (Movie)

The calculated quality belief is returned to the Trust Calculator. The TcEngine then maps the quality belief to the selector’s cognitive trust in the movie. Emotional trust is obtained by calling the *getEmotionalTrust* method of ETS with the following parameter:

- *trustee*
 - id = “Zombieland”
 - type = “Movie/Horror”

The ET-Policies responsible for emotional trust of movies are shown in Figure 23 of Section 5.4. After emotional trust is obtained, it is returned to the Trust Calculator. The TcEngine then calculates computational trust by applying weighted average to the calculated cognitive trust and emotional trust. The last step is for the calculated computational trust to be returned to the application to be used in movie ranking. The computational trust formation process is graphically illustrated in Figure 58. The highest ranked movie (i.e., most trusted movie) is then selected by the movie selector.

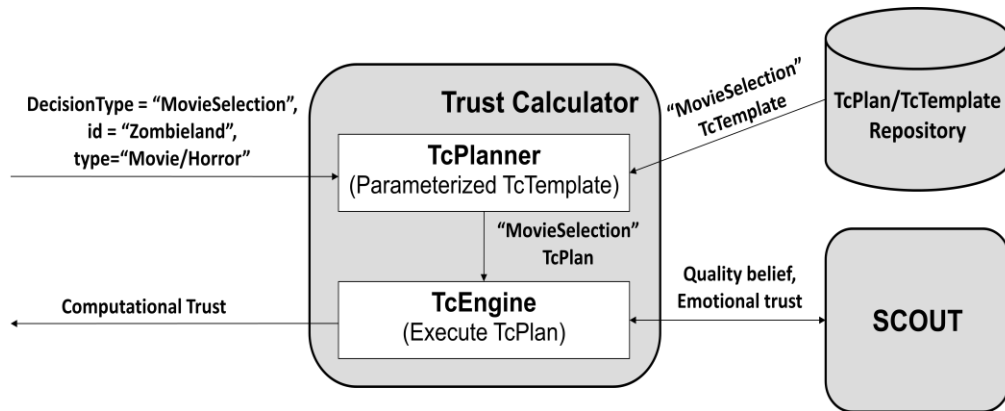


Figure 58: Computational Trust Formation (Movie)

10.1.2 Implementation

The movie ranking application was developed using the Java programming language. Reputations needed for quality belief calculation are obtained from Rotten Tomatoes and IMDb. For simplicity sake, the reputations are looked up and stored in a database. The reputations are stored in the same format as the original website. When EGS needs to perform reputation gathering, it accesses the database to obtain a movie's reputation. An Evidence Gatherer and Handler pair has been implemented for accessing the reputations calculated by Rotten Tomatoes. A different Evidence Gatherer and Handler pair is used to access the reputations calculated by IMDb.

10.2 Web Service Scenario

The first part of this section provides a general overview of the web service scenario. The scenario's implementation is described in the second part of this section.

10.2.1 Overview

The web service scenario described in this subsection is a more complex version of the scenario described in Section 1.3.2. The scenario consists of three decisions: web service selection, negotiation and management. Each decision along how its associated computational trust is calculated is introduced in the rest of this subsection.

10.2.1.1 Web Service Selection

A web service discovery, selection and negotiation web service (DSN) has been implemented. An application could invoke the *findWebService* method of DSN to obtain a web service. The method has as input three parameters: *web service type*, *web service classifications* and *web service importance*. An example invocation could have the following input parameters:

- *web service type* = “Type1”
- *web service classifications* = [“Consumer”, “Enterprise”]
- *web service importance* = “Low”

Basically, the invocation would cause DSN to find a web service of “Type1”. The web service could be a consumer level web service or an enterprise level web service. The invocation is of low importance to the trustor. The importance level is used to determine computational trust formation.

After web service discovery based on *web service type* and *web service classifications*, the next step for DSN is to calculate computational trust for each of the discovered web services. After the instantiation of the Trust Calculator, it is configured by calling the *setApplicationFactors* method with the following input parameters:

- *application factors*
 - name = “DecisionType”, value = “WebServiceSelection”

The method invocation informs the Trust Calculator that it is calculating computational trust for web service selection. To calculate computational trust in a web service, the DSN calls the *calculateTrust* method of the Trust Calculator. An example invocation could have the following input parameters:

- *trustee*
 - id = “www.webservice.com”
 - type = “WebService/Type1”

- *decision factors*
 - name = “Importance”, value = “Low”

Information about the *trustee* is obtained during web service discovery. As for *decision factors*, the decision’s importance is obtained from *web service importance* supplied to the *findWebService* method. The invocation would cause the Trust Calculator to invoke the TcPlanner.

The *XMLTcPlanner* uses the mappings in Figure 59 to select the TcPlan needed for computational trust calculation. In the figure, there are two possible TcPlans. There is a TcPlan for “Importance” is “Low” and another for “Importance” is “High”. In the case of low importance (lines 3-12), the TcPlan “WS-SelectionTrust-Low” is returned to the Trust Calculator. In the case of high importance (lines 13-22), the TcPlan “WS-SelectionTrust-High” is returned to the Trust Calculator.

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <mappings>
3    <mapping>
4      <factors>
5        <factor name="DecisionType">WebServiceSelection</factor>
6        <factor name="Type">WebService/Type1</factor>
7        <factor name="Importance">Low</factor>
8      </factors>
9      <tcplan>
10       <name>WS-SelectionTrust-Low</name>
11     </tcplan>
12  </mapping>
13  <mapping>
14    <factors>
15      <factor name="DecisionType">WebServiceSelection</factor>
16      <factor name="Type">WebService/Type1</factor>
17      <factor name="Importance">High</factor>
18    </factors>
19    <tcplan>
20     <name> WS-SelectionTrust-High</name>
21   </tcplan>
22 </mapping>
23   ...
24 </mappings>

```

Figure 59: Factors-TcPlan Mappings (Web Service Selection)

The “WS-SelectionTrust-Low” TcPlan is shown in Figure 60. In the figure, computational trust is based solely on the calculated cognitive trust. Since emotional trust is not considered, the calculated computational trust can be viewed as “rational”. As both cognitive trust and computational trust have the same values, the TcPlan can be simplified by returning the calculated cognitive trust to DSN (lines 39-47).

In the figure, cognitive trust calculation is the responsibility of the “cognitiveTrust” node (line 35). Cognitive trust is calculated from the aggregate belief value of “aggregateBelief”. The “aggregateBelief” node (lines 30-34) has a “cause” parameter ($aBelief > 0 \ \&\& \ reliability > 0.5$) that when evaluated to true based on the aggregate belief value of its left child (i.e., “competenceBelief”) would return the aggregate belief value of its right child (“avgAggregateBelief”). Otherwise, it would return $belief = -1$ and $reliability = 0$ (i.e., disbelief). This aggregate belief value would be mapped by the “cognitiveTrust” node to $trust = -1$ (i.e., distrust). The “avgAggregateBelief” node (lines 24-29) calculates aggregate belief by averaging the calculated accessibility belief and competence belief. Competence, accessibility and performance beliefs are all obtained from SCOUT (lines 4-7, 8-15, 16-23). All three nodes have a parameter “importance” that is set to “Low” (lines 6, 14, 22) since the TcPlan is selected when a decision is of low importance. Accessibility belief and performance belief are also calculated based on experience and reputation (lines 10-13, 18-21). The parameters “importance” and “evidenceType” are *hints* to the *getAggregateBelief* method of BFS since any non-“aggregateBelief” parameter of “AggregateBeliefQuery” is treated as *hints*. The tree constructed from this TcPlan is graphically represented in Figure 61. As for “WS-SelectionTrust-High”, its difference from “WS-SelectionTrust-Low” is that its “importance” parameter is set to “High”. Also, recommendations are used in accessibility belief and performance belief calculation.


```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <tcplan>
3    <nodeDefinition>
4      <node id="competenceBelief" class="AggregateBeliefQuery">
5        <parameter name="aggregateBelief" type="string">Competence</parameter>
6        <parameter name="importance" type="string">Low</parameter>
7      </node>
8      <node id="accessibilityBelief" class="AggregateBeliefQuery">
9        <parameter name="aggregateBelief" type="string">Accessibility</parameter>
10       <parameters name="evidenceType" type="string">
11         <parameter>Experience</parameter>
12         <parameter>Reputation</parameter>
13       </parameters>
14       <parameter name="importance" type="string">Low</parameter>
15     </node>
16     <node id="performanceBelief" class="AggregateBeliefQuery">
17       <parameter name="aggregateBelief" type="string">Performance</parameter>
18       <parameters name="evidenceType" type="string">
19         <parameter>Experience</parameter>
20         <parameter>Reputation</parameter>
21       </parameters>
22       <parameter name="importance" type="string">Low</parameter>
23     </node>
24     <node id="avgAggregateBelief" class="AggregateBeliefWeightedAvg">
25       <parameters name="weights" type="double">
26         <parameter>0.5</parameter>
27         <parameter>0.5</parameter>
28       </parameters>
29     </node>
30     <node id="aggregateBelief" class="CausalAggregateBelief">
31       <parameter name="cause" type="string">
32         aBelief > 0 && reliability > 0.5
33       </parameter>
34     </node>
35     <node id="cognitiveTrust" class="AggregateBeliefToTrust"/>
36   </nodeDefinition>
37   <trustCalculation>
38     <tree>
39       <cognitiveTrust>
40         <aggregateBelief>
41           <competenceBelief/>
42           <avgAggregateBelief>
43             <accessibilityBelief/>
44             <performanceBelief/>
45           </avgAggregateBelief>
46         </aggregateBelief>
47       </cognitiveTrust>
48     </tree>
49   </trustCalculation>
50 </tcplan>

```

Figure 60: TcPlan (Web Service Selection)

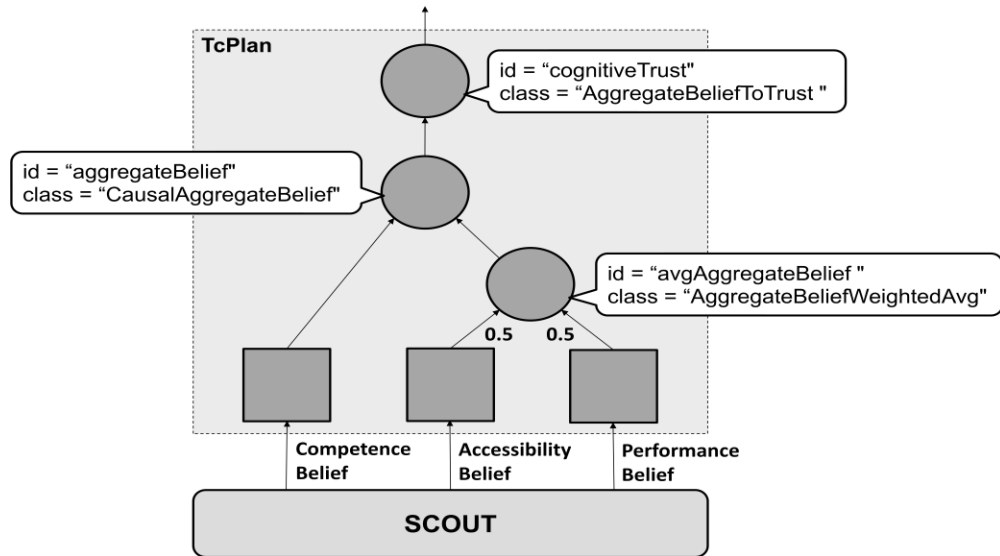


Figure 61: Tree of TcPlan (Web Service Selection)

After TcPlan formation, the Trust Calculator invokes the TcEngine to execute the TcPlan. The consumer's competence belief in a web service is obtained by calling the *getAggregateBelief* method of BFS with the following input parameters:

- *trustee*
 - id = "www.webservice.com"
 - type = "WebService/Type1"
- *aggregate belief*
 - type = "Competence"
- *hints*
 - name = "Importance", value = "Low"

The BF-Policy responsible for calculating competence belief in a web service is shown in Figure 62. In the figure, importance is not considered in belief calculation. The aggregate belief competence is calculated from qualification belief. Qualification belief is calculated from qualification signals (implemented as certificates). The Belief Engine identified as "SCOUT-Signal" is selected for belief calculation (line 5). "SCOUT-Signal"

implements the belief formation algorithm described in Section 7.1.4. The configuration of the Belief Engine (lines 6-7) is mapped to the algorithm as follows:

- SignalCountWeight $\rightarrow w_c$
- SignalThreshold $\rightarrow threshold_{sig}$

The last step in the BF-Policy is to invoke the Belief Engine to perform belief calculation.

```

1  when
2    trustee: Trustee( type == "WebService/Type1" )
3    aggBelief: AggregateBelief( type == "Competence" )
4  then
5    beliefEngine = Registry.lookup("SCOUT-Signal");
6    beliefEngine.set("SignalCountWeight", 0);
7    beliefEngine.set("SignalThreshold", 1);
8
9    beliefEngine.calculateBelief(trustee, new Belief("Qualification"));
10 end

```

Figure 62: BF-Policy (Competence Belief)

To maximize belief reliability, if the number of signals available in the Evidence Repository is less than the “SignalThreshold” in Figure 62 (i.e., if there are no signals), the Belief Engine invokes the *gatherEvidence* method of EGS with the following input parameters to gather more signals:

- *trustee*
 - id = “www.webservice.com”
 - type = “WebService/Type1”
- *belief*
 - type = “Qualification”
- *hints*
 - name = “EvidenceType”, value = “Signal”

The EG-Policy responsible for the gathering of qualification signals of a web service is shown in Figure 63. In the figure, the Evidence Gatherer identified as

“QualificationGatherer” is invoked to perform qualification signals gathering (lines 6-7). The gathered signals are mapped to the interval of [-1, 1] by the Evidence Handler. The mapped signals are stored in the Evidence Repository to be used in belief calculation.

```

1  when
2    trustee: Trustee( type == "WebService/Type1" )
3    belief: Belief( type == "Qualification" )
4    Hint( name == "EvidenceType" && value == "Signal" )
5  then
6    gatherer = Registry.lookup("QualificationGatherer");
7    gatherer.gatherEvidence(trustee, belief);
8  end

```

Figure 63: EG-Policy (Qualification Belief)

Accessibility belief is obtained by calling the *getAggregateBelief* method of BFS with the following input parameters:

- *trustee*
 - id = “www.webservice.com”
 - type = “WebService/Type1”
- *aggregate belief*
 - type = “Accessibility”
- *hints*
 - name = “EvidenceType”, value = “Experience, Reputation”
 - name = “Importance”, value = “Low”

The BF-Policies needed for belief calculation are shown in Figure 64. In the figure, there is a BF-Policy for low importance (lines 1-36) and a BF-Policy for high importance (lines 38-45). In the case of low importance, the aggregate belief accessibility is calculated as the weighted average of the availability beliefs. Since the “EvidenceType” hint is set to “Experience” and “Reputation”, “SCOUT-Experience” (lines 11-17) and “SCOUT-Reputation” (lines 27-33) are configured and invoked to calculate availability belief based on experience and reputation. “SCOUT-Experience”, “SCOUT-Recommendation” and “SCOUT-Reputation” implements the belief formation algorithm described in

Section 7.1.1, Section 7.1.2 and Section 7.1.3. The availability beliefs are all assigned different weights (lines 16, 24, 32). The calculated accessibility belief is returned to the Trust Calculator. In the case of high importance, the belief calculation is similar to that of low importance. The difference is that the Belief Engines are configured differently. For example, “RecommendationThreshold” (line 21) is set to 3 for low importance but it is set to 6 for high importance. This is so that for high importance, more recommendations are taken into account during availability belief formation.

In the case of experience, the *gatherEvidence* method of EGS is always invoked to gather all of the consumer’s latest experiences. The input parameters are as follows:

- *trustee*
 - id = “www.webservice.com”
 - type = “WebService/Type1”

- *belief*
 - type = “Availability”

- *hints*
 - name = “EvidenceType”, value = “Experience”

The EG-Policy responsible for the gathering of availability experiences of a web service is shown in Figure 65 (lines 1-9). In the figure, the experiences are gathered using the broadcast strategy. The gathered experiences are mapped to the interval of [-1, 1] by the Evidence Handler. The mapped experiences are stored in the Evidence Repository to be used in belief calculation.

```

1  when
2    trustee: Trustee( type == "WebService/Type1" )
3    aggBelief: AggregateBelief( type == "Accessibility" )
4    hint: Hint( name == "EvidenceType" )
5    Hint( name == "Importance" && value == "Low" )
6  then
7    belief = new Belief("Availability");
8    evidenceTypes = hint.getValue();
9    aggBeliefCalculator = new WeightedAvg();
10
11   if (evidenceTypes.contains("Experience")) {
12     beliefEngine = Registry.lookup("SCOUT-Experience");
13     beliefEngine.set("ExperienceWindow", 30);
14     ...
15     expBelief = beliefEngine.calculateBelief(trustee, belief);
16     aggBeliefCalculator.setBelief(0.5, expBelief);
17   }
18
19   if (evidenceTypes.contains("Recommendation")) {
20     beliefEngine = Registry.lookup("SCOUT-Recommendation");
21     beliefEngine.set("RecommendationThreshold", 3);
22     ...
23     recBelief = beliefEngine.calculateBelief(trustee, belief);
24     aggBeliefCalculator.setBelief(0.3, recBelief);
25   }
26
27   if (evidenceTypes.contains("Reputation")) {
28     beliefEngine = Registry.lookup("SCOUT-Reputation");
29     beliefEngine.set("FeedbackThreshold", 10);
30     ...
31     repBelief = beliefEngine.calculateBelief(trustee, belief);
32     aggBeliefCalculator.setBelief(0.2, repBelief);
33   }
34
35   aggBeliefCalculator.calculateAggBelief();
36 end
37
38 when
39   trustee: Trustee( type == "WebService/Type1" )
40   aggBelief: AggregateBelief( type == "Accessibility" )
41   hint: Hint( name == "EvidenceType" )
42   Hint( name == "Importance" && value == "High" )
43 then
44   ...
45 end

```

Figure 64: BF-Policies (Accessibility Belief)

```

1  when
2    trustee: Trustee( type == "WebService/Type1" )
3    belief: Belief( type == "Availability" )
4    hint: Hint( name == "EvidenceType" && value == "Experience" )
5  then
6    strategy = new BroadcastStrategy();
7    strategy.set("EvidenceType", hint.getValue());
8    strategy.execute(trustee, belief);
9  end
10
11 when
12 trustee: Trustee( type == "WebService/Type1" )
13 belief: Belief( type == "Availability" )
14 hint1: Hint( name == "EvidenceType" && value == "Recommendation" )
15 hint2: Hint( name == "NumOfEvidence" )
16 then
17 strategy = new EvidenceourceTrustStrategy();
18 strategy.set("EvidenceType", hint1.getValue());
19 strategy.set("NumOfEvidence", hint2.getValue());
20 strategy.set("EvidenceourceTrustThreshold", 0);
21
22 strategy.execute(trustee, belief);
23 end
24
25 when
26 trustee: Trustee( type == "WebService/Type1" )
27 belief: Belief( type == "Availability" )
28 hint1: Hint( name == "EvidenceType" && value == "Reputation" )
29 hint2: Hint( name == "NumOfEvidence" )
30 then
31 strategy = new EvidenceourceTrustStrategy();
32 strategy.set("EvidenceType", hint1.getValue());
33 strategy.set("NumOfEvidence", hint2.getValue());
34 strategy.set("EvidenceourceTrustThreshold", 0);
35
36 strategy.execute(trustee, belief);
37 end

```

Figure 65: EG-Policies (Availability Belief)

In the case of recommendation, the *gatherEvidence* method of EGS is invoked if the number of recommendations available in the Evidence Repository is less than the “RecommendationThreshold” in Figure 64. This is to maximize belief reliability. The input parameters to the *gatherEvidence* method are as follows:

- *trustee*
 - id = “www.webservice.com”
 - type = “WebService/Type1”

- *belief*
 - type = “Availability”

- *hints*
 - name = “EvidenceType”, value = “Recommendation”
 - name = “NumOfEvidence”, value = “3”

The hint “NumOfEvidence” is used to inform EGS of the number of recommendations to be gathered. The EG-Policy responsible for the gathering of availability recommendations of a web service is shown in Figure 65 (lines 11-23). In the figure, the recommendations are gathered using the evidence source trust strategy. The gathered recommendations are mapped to the interval of [-1, 1] by the Evidence Handler. The mapped recommendations are stored in the Evidence Repository to be used in belief calculation. In the case of reputation, it is gathered using the same approach as that for recommendation. Its EG-Policy is shown in Figure 65 (lines 25-37).

The evidence gathering and belief formation of performance belief uses the same approach as that for accessibility belief. Basically, aggregate belief performance is calculated as the weighted average of timeliness beliefs. The calculated performance belief is returned to the Trust Calculator. After TcPlan execution, the calculated computational trust is returned to DSN. The computational trust formation process is graphically illustrated in Figure 66.

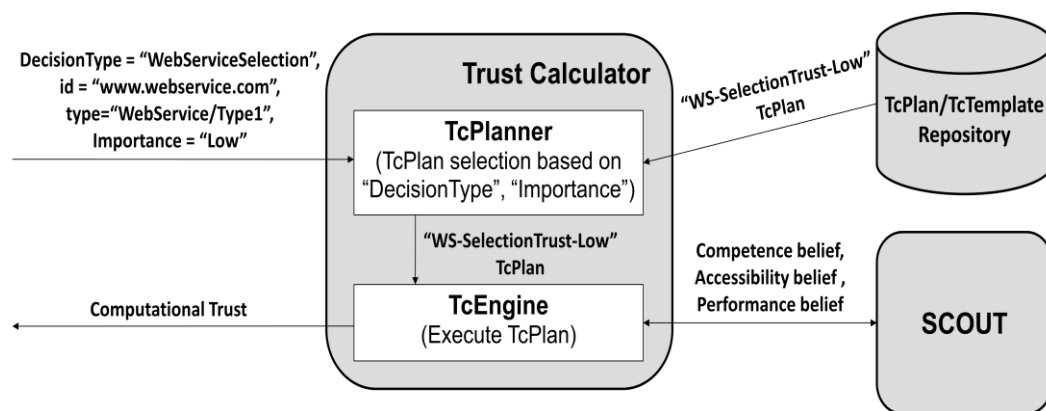


Figure 66: Computational Trust Formation (Web Service Selection)

For each discovered web service, the utility to be gained from invoking the web service is calculated as follows:

$$utility_i = (compTrust_i + 1) \cdot utility_{expectation} \quad (18)$$

In equation (18), $utility_{expectation}$ is the utility that the web service consumer would gain if the web service meets the consumer's expectations. A consumer based its expectation on the web service's classification. For example, a consumer may have higher expectation for an enterprise level web service than a consumer level web service. The different expectations result in different utility gained from web service invocation. If the consumer's computational trust in web service i is calculated to be -1, then zero utility would be gained from invoking the web service. If the computational trust is calculated as 0, then the web service is expected to meet the consumer's expectations. Higher computational trust should lead to higher utility. After utility is calculated, the next step is filter the candidate web services based on utility. This is by selecting all the web services with $utility_i \geq threshold_{utility}$ where $threshold_{utility}$ is a threshold determined by the web service consumer.

10.2.1.2 Web Service Negotiation

After DSN has performed utility-based web service selection, the next step is for the negotiation of contracts with the remaining web services. To determine the terms of negotiation, computational trust needs to be calculated. The Trust Calculator is reconfigured by calling the *setApplicationFactors* method with the following input parameters:

- *application factors*
 - name = "DecisionType", value = "WebServiceNegotiation"

The method invocation informs the Trust Calculator that it is calculating computational trust for web service negotiation. To calculate computational trust in a web service, the *calculateTrust* method of the Trust Calculator is invoked with the following parameters:

- *trustee*
 - id = “www.webservice.com”
 - type = “WebService/Type1”
- *decision factors*
 - name = “Importance”, value = “Low”

The invocation would cause the Trust Calculator to invoke the TcPlanner.

The *XMLTcPlanner* uses the mappings in Figure 67 to select the TcPlan needed for computational trust calculation. Figure 59 and Figure 67 together forms the factors-TcPlan mappings for DSN. In the figure, there are two possible TcPlans. There is a TcPlan for “Importance” is “Low” and another for “Importance” is “High”. In the case of low importance (lines 3-12), the TcPlan “WS-NegotiationTrust-Low” is returned to the Trust Calculator. In the case of high importance (lines 13-22), the TcPlan “WS-NegotiationTrust-High” is returned to the Trust Calculator.

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <mappings>
3    ...
4    <mapping>
5      <factors>
6        <factor name="DecisionType">WebServiceNegotiation</factor>
7        <factor name="Type">WebService/Type1</factor>
8        <factor name="Importance">Low</factor>
9      </factors>
10     <tcplan><name>WS-NegotiationTrust-Low</name></tcplan>
11   </mapping>
12   <mapping>
13     <factors>
14       <factor name="DecisionType">WebServiceNegotiation</factor>
15       <factor name="Type">WebService/Type1</factor>
16       <factor name="Importance">High</factor>
17     </factors>
18     <tcplan><name> WS-NegotiationTrust-High</name></tcplan>
19   </mapping>
20 </mappings>

```

Figure 67: Factors-TcPlan Mappings (Web Service Negotiation)

The “WS-NegotiationTrust-Low” TcPlan is shown in Figure 68. In the figure, computational trust is based solely on the calculated cognitive trust. Cognitive trust is

calculated from the web service consumer's dependability belief. Dependability belief is calculated from experience and reputation. The tree constructed from this TcPlan is graphically represented in Figure 69. Unlike web service selection, weighted average is applied to aggregate beliefs (line 25) at the Trust Calculator as oppose to at the BFS. This is just to illustrate a different way that aggregate belief can be calculated. The approach in this section allows the application to determine the weight on experience-based aggregate belief and reputation-based aggregate belief. In web service selection, the determination is left up to the web service consumer. Both approaches are equally viable and are supported by the computational trust architecture. As for "WS-NegotiationTrust-High", its difference from "WS-NegotiationTrust-Low" is that its "importance" parameter is set to "High". Also, recommendations are used in dependability belief calculation.

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <tcplan>
3    <nodeDefinition>
4      <node id="expBelief" class="AggregateBeliefQuery">
5        <parameter name="aggregateBelief" type="string">Dependability</parameter>
6        <parameter name="evidenceType" type="string">Experience</parameter>
7        <parameter name="importance" type="string">Low</parameter>
8      </node>
9      <node id="repBelief" class="AggregateBeliefQuery">
10       <parameter name="aggregateBelief" type="string">Dependability</parameter>
11       <parameter name="evidenceType" type="string">Reputation</parameter>
12       <parameter name="importance" type="string">Low</parameter>
13     </node>
14     <node id="dependabilityBelief" class="AggregateBeliefWeightedAvg">
15       <parameters name="weights" type="double">
16         <parameter>0.7</parameter>
17         <parameter>0.3</parameter>
18       </parameters>
19     </node>
20     <node id="cognitiveTrust" class="AggregateBeliefToTrust"/>
21   </nodeDefinition>
22   <trustCalculation>
23     <tree>
24       <cognitiveTrust>
25         <dependabilityBelief><expBelief/><repBelief/></dependabilityBelief>
26       </cognitiveTrust>
27     </tree>
28   </trustCalculation>
29 </tcplan>

```

Figure 68: TcPlan (Web Service Negotiation)

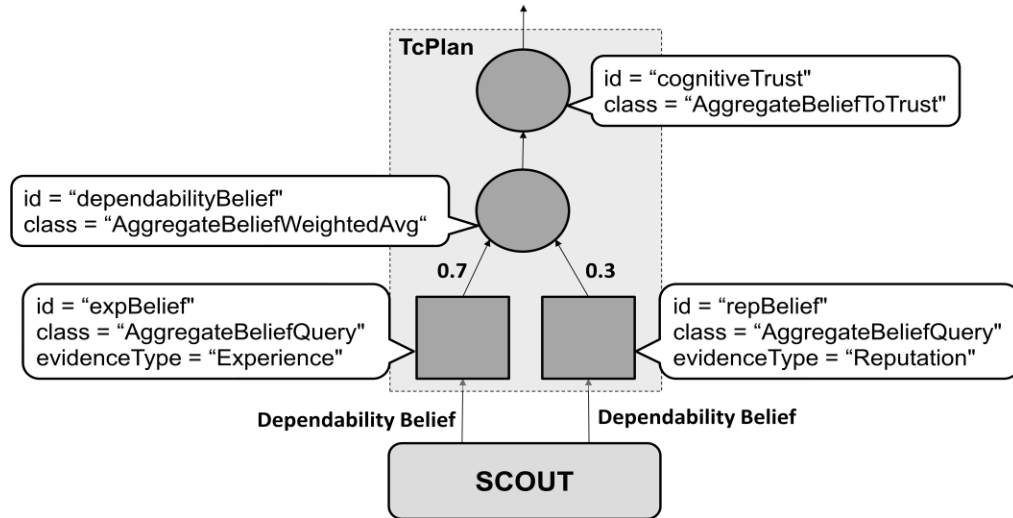


Figure 69: Tree of TcPlan (Web Service Negotiation)

After TcPlan formation, the Trust Calculator invokes the TcEngine to execute the TcPlan. The web service consumer's dependability belief in a web service based on experience is obtained by calling the *getAggregateBelief* method of BFS with the following input parameters:

- *trustee*
 - id = "www.webservice.com"
 - type = "WebService/Type1"
- *aggregate belief*
 - type = "Dependability"
- *hints*
 - name = "EvidenceType", value = "Experience"
 - name = "Importance", value = "Low"

The BF-Policies responsible for calculating dependability beliefs in a web service are shown in Figure 70. There needs to be BF-Policies for belief formation based on experience (lines 1-20), recommendation (lines 22-38) and reputation (not shown due to space limitation). For each evidence type, there is a BF-Policy for low importance (lines 1-11, 22-29) and a BF-Policy for high importance (lines 13-20, 31-38). The aggregate

belief dependability is calculated from compliance belief. The formation of compliance belief from experience, recommendation and reputation is based on the Belief Engines “SCOUT-Experience”, “SCOUT-Recommendation” and “SCOUT-Reputation”. Basically, the belief formation is similar to the case with web service selection except with different evidence.

```

1  when
2    trustee: Trustee( type == "WebService/Type1" )
3    aggBelief: AggregateBelief( type == "Dependability" )
4    Hint( name == "EvidenceType" && value == "Experience" )
5    Hint( name == "Importance" && value == "Low" )
6  then
7    beliefEngine = Registry.lookup("SCOUT-Experience");
8    beliefEngine.set("ExperienceWindow", 30);
9    ...
10   beliefEngine.calculateBelief(trustee, new Belief("Compliance"));
11 end
12
13 when
14   trustee: Trustee( type == "WebService/Type1" )
15   aggBelief: AggregateBelief( type == "Dependability" )
16   Hint( name == "EvidenceType" && value == "Experience" )
17   Hint( name == "Importance" && value == "High" )
18 then
19   ...
20 end
21
22 when
23   trustee: Trustee( type == "WebService/Type1" )
24   aggBelief: AggregateBelief( type == "Dependability" )
25   Hint( name == "EvidenceType" && value == "Recommendation" )
26   Hint( name == "Importance" && value == "Low" )
27 then
28   ...
29 end
30
31 when
32   trustee: Trustee( type == "WebService/Type1" )
33   aggBelief: AggregateBelief( type == "Dependability" )
34   Hint( name == "EvidenceType" && value == "Recommendation" )
35   Hint( name == "Importance" && value == "High" )
36 then
37   ...
38 end
39
40   ...

```

Figure 70: BF-Policies (Dependability Belief)

As for evidence gathering, the approach taken is similar to that of web service selection. The deployed EG-Policies are similar to those listed in Figure 65 except for the different evidence that is being gathered.

By obtaining experience-based dependability belief and reputation-based dependability belief from SCOUT, computational trust can now be calculated. After TcPlan execution, the calculated computational trust is returned to DSN. The computational trust formation process is graphically illustrated in Figure 71. Computational trust could help determine the penalties to be demanded for contract violation. Basically, the lower the calculated computational trust, the higher should be the penalties as the web service is believed to be undependability. The actual steps for contract negotiation are outside the scope of this thesis. Of all the web services where contract negotiation ends up being successful, DSN would select the web service with the highest utility to be returned to the application.

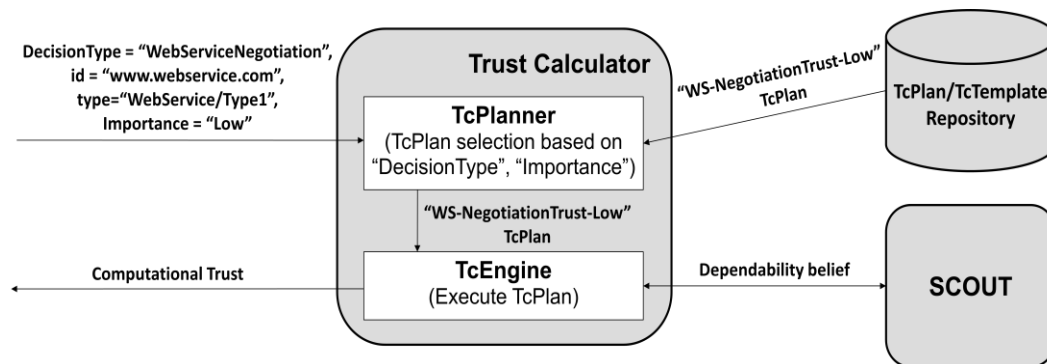


Figure 71: Computational Trust Formation (Web Service Negotiation)

10.2.1.3 Web Service Management

A web service management web service (WSM) has been implemented. An application could invoke the *startManagement* method of WSM to manage a web service. The method has as input two parameters: *web service* and *web service importance*. An example invocation could have the following input parameters:

- *web service*
 - id = "www.webservice.com"

- type = “WebService/Type1”
- *web service importance* = “Low”

Information about the *web service* is obtained from DSN. The return value of *startManagement* is a unique identifier: *managementId*. The *managementId* identifier is used in the *endManagement* method to end an existing web service management.

Upon invocation of *startManagement*, WSM would instantiate a management agent (mAgent) to manage the provided web service. The management policies to be deployed into the mAgent can be selected by taking computational trust into account. For example, computational trust can be calculated using the TcPlan of Figure 68. For web services with low computational trust, management policies can be used to keep the web service consumer inform of changes to the provided QoS. These management policies are not needed in cases when the web service is computationally trusted.

Computational trust can also be used within a management policy. An example is shown in Figure 72. In the figure, whenever there is an SLA violation, computational trust would be calculated (lines 1-6). An email is sent if computational trust has fallen below zero (lines 8-12). Upon invocation of the *endManagement* method, the WSM would look up the corresponding mAgent, stop its execution and destroy the mAgent.

```

1  when
2    Violation( )
3  then
4    trust = trustCalculator.calculateTrust(trustee);
5    insert(new Trust(trustee, trust));
6  end
7
8  when
9    trust: Trust( value < 0 )
10 then
11   email("chyew@csd.uwo.ca", "Computational Trust < 0");
12 end

```

Figure 72: Web Service Management Policies

With the end of web service invocation, the last step is for the application to request feedbacks from the web service consumer with regards to the invoked web

service. The application would request feedbacks concerning a web service's competence, accessibility, performance and dependability. A consumer could choose to provide feedback to only some of the aggregate beliefs. The obtained feedbacks are mapped to the interval of $[-1, 1]$. Next, the *provideFeedback* method of BFS is invoked. An example invocation could have the following input parameters:

- *trustee*
 - id = "www.webservice.com"
 - type = "WebService/Type1"
- *aggregate belief feedbacks*
 - type = "Accessibility", timestamp = "2010-10-06 15:10:00", feedback = 0
 - type = "Performance", timestamp = "2010-10-06 15:10:00", feedback = 0
- *hints*
 - name = "Importance", value = "Low"

The AF-Policies responsible for mapping aggregate belief feedbacks to belief feedbacks are shown in Figure 73. In the figure, importance is not considered since the aggregate belief to belief mappings are the same irrespective of importance. In the case of feedback to aggregate belief performance, this would trigger the policy at lines 21-29. The feedback to performance belief is mapped to feedback to timeliness belief (lines 25-27) with timeliness belief being assigned the timestamp and feedback of performance belief. After mapping, the *provideFeedback* method of EGS is invoked (line 28) with the timeliness belief feedback.

The EA-Policy responsible for evidence source assessment is shown in Figure 74. In the figure, the Evidence Source Assessor identified as "SCOUT-Assessor" is selected to calculate evidence source trust (line 5). "SCOUT-Assessor" implements the evidence source assessment algorithm described in Section 7.2. It is responsible for all the belief feedbacks to web service of type one and for all the gathered evidence types. The assessor is configured (lines 6-7) and finally invoked (line 8) to calculate evidence source

trust in recommenders, reputation systems and signal providers. The same approach is also taken for feedback to aggregate belief accessibility.

```

1  when
2    trustee: Trustee( type == "WebService/Type1" )
3    aggFeedback: AggregateBeliefFeedback( type == "Competence" )
4  then
5    feedback = new BeliefFeedback("Qualification",
6                                   aggFeedback.getTimestamp(),
7                                   aggFeedback.getFeedback());
8    EGS.provideFeedback(trustee, [feedback]);
9  end
10
11 when
12  trustee: Trustee( type == "WebService/Type1" )
13  aggFeedback: AggregateBeliefFeedback( type == "Accessibility" )
14 then
15  feedback = new BeliefFeedback("Availability",
16                                 aggFeedback.getTimestamp(),
17                                 aggFeedback.getFeedback());
18  EGS.provideFeedback(trustee, [feedback]);
19 end
20
21 when
22  trustee: Trustee( type == "WebService/Type1" )
23  aggFeedback: AggregateBeliefFeedback( type == "Performance" )
24 then
25  feedback = new BeliefFeedback("Timeliness",
26                                 aggFeedback.getTimestamp(),
27                                 aggFeedback.getFeedback());
28  EGS.provideFeedback(trustee, [feedback]);
29 end
30
31 when
32  trustee: Trustee( type == "WebService/Type1" )
33  aggFeedback: AggregateBeliefFeedback( type == "Dependability" )
34 then
35  feedback = new BeliefFeedback("Compliance",
36                                 aggFeedback.getTimestamp(),
37                                 aggFeedback.getFeedback());
38  EGS.provideFeedback(trustee, [feedback]);
39 end

```

Figure 73: AF-Policies (Web Service)

```

1  when
2      trustee: Trustee( type == "WebService/Type1" )
3      feedback: BeliefFeedback( )
4  then
5      assessor = Registry.lookup("SCOUT-Assessor");
6      assessor.set("EvidenceWindow", 60);
7      ...
8      assessor.assessEvidenceource(trustee, feedback);
9  end

```

Figure 74: EA-Policy (Web Service)

10.2.2 Implementation

A graphical illustration of DSN is shown in Figure 75. In the figure, DSN consists of three different types of plug-ins. The WS-Discoverer plug-in is responsible for web service discovery. An example of which can be used for discovering the web services stored in a WSO2 Governance Registry [138]. A WS-Selector plug-in is responsible for web service selection. An example plug-in could implement the algorithm described in Section 10.2.1.1. A WS-Negotiator plug-in is responsible for web service negotiation. A stub plug-in is currently used as a substitute for web service negotiation. The plug-ins are all registered with a registry. The DSN Manager uses the registry to discover the available plug-ins. It employs policies to coordinate the invocation of different plug-ins. The policies are deployed to the Drools rule engine [37] within the DSN Manager.

A graphical illustration of WSM is shown in Figure 76. In the figure, the WSM Manager is responsible for the selection of management policies. It is also responsible for the instantiation and destruction of mAgents. Each mAgent has its own Drools rule engine to process management policies. A mAgent may interact with the Monitoring Service to setup the monitoring of a web service. The Monitoring Service is implemented as a stub that provides events that trigger the management policies.

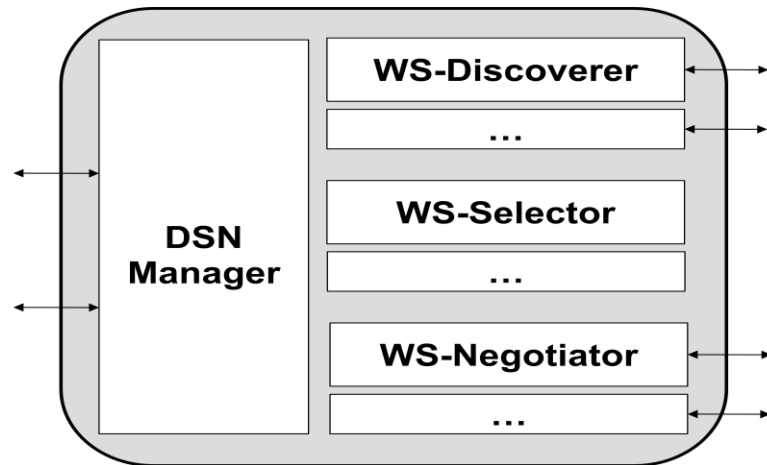


Figure 75: Web Service Discovery, Selection and Negotiation Web Service

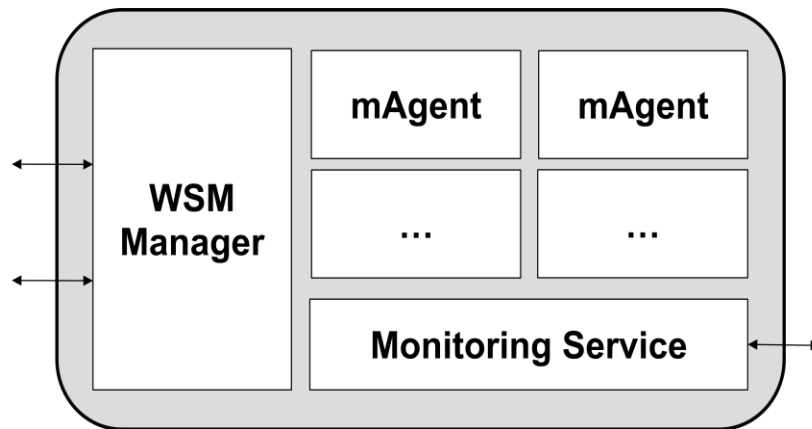


Figure 76: Web Service Management Web Service

10.3 Summary

Two scenarios are presented in this chapter. The scenarios are used to demonstrate how computational trust can be calculated with the support of the computational trust architecture. The first scenario is a movie selection scenario where computational trust is calculated from cognitive trust and emotional trust. The second scenario is a web service scenario where computational trust is calculated from cognitive trust. The calculated computational trust is used in web service selection, negotiation and management.

Chapter 11

11 Evaluation

The focus of this chapter is on evaluating the design of the computational trust architecture. This chapter is divided into two sections. The first section covers the evaluation of SCOUT. The second section covers the evaluation of the Trust Calculator.

11.1 Evaluation of SCOUT

SCOUT is evaluated from three different perspectives. The first perspective examines SCOUT through its evolution. The second perspective examines how SCOUT supports the computational trust properties. The last perspective examines how SCOUT addresses the challenges to computational trust.

11.1.1 Evolution

The design of SCOUT has gone through numerous revisions. Initially, SCOUT consisted of numerous Trust Manager plug-ins. Each Trust Manager is responsible for its own evidence gathering and trust calculation. For example, there is a Trust Manager for calculating trust based on experiences and a Trust Manager for calculating trust based on recommendations. As trust is subjective, each trustor is responsible for deploying its own Trust Managers. The weakness of this design is that evidence gathering and trust calculation are tightly coupled. As a result, a new Trust Manager has to be created for each evidence discovery and gathering protocol and trust formation algorithm combination. The design does not encourage code reuse. Moreover, it also introduces numerous challenges to SCOUT's maintenance.

To address the coupling weakness, SCOUT was redesigned with Evidence Gatherers responsible for evidence gathering and Trust Managers responsible for trust calculation. With this more modular design, evidence gathering and trust formation are now able to evolve independently of each other though a weakness still remains in that there is no research that backs up the trust calculation. As a result, the cognitive trust view from social psychology has been adopted. In cognitive trust formation, cognitive

trust is calculated from beliefs and beliefs calculated from evidence. SCOUT in turn was redesigned to be responsible for belief calculation and evidence gathering. As for cognitive trust calculation, it is the responsibility of the applications. Basically, SCOUT is a web service that provides beliefs to the applications.

A weakness of encapsulating both evidence gathering and belief calculation in a web service is that evidence gathering is not accessible to the applications. Some applications may already support cognitive trust calculation and could leverage SCOUT for more evidence. To address this weakness, the SCOUT service was redesigned into two services: the Evidence Gathering Service (EGS) and Belief Formation Service (BFS). Later, the Emotional Trust Service (ETS) is added to SCOUT to account for trust calculation that is not evidence-based. This three service design is the design discussed in Chapter 4 and graphically illustrated in Figure 10 of Section 0.

As for the adaptation of the SCOUT services to changing computational trust formation needs, initially the different factors (Section 4.1.6) were implemented as input parameters that can be configured by the trustor. A weakness of this approach is that there are many factors. Moreover, different trustors may have different views on when adaptation is needed along with how adaptation should proceed. The result is a system that is complex and challenging to configure. With the adoption of policies however, the entire process can be simplified. All the factors that SCOUT may be interested in are passed in as hints. The hints are then used in policies for web service adaptation. Basically, the trustor could start out with simple policies. If the policies are not adequate, more advanced policies could be created. Adding, updating and removing policies as the need arises allows for adaptation without having to sacrifice ease of use.

11.1.2 Computational Trust Properties

SCOUT is designed with the computational trust properties listed in Section 1.2.3 in mind. SCOUT supports computational trust that is quantifiable, comparable, subjective, multidimensional and dynamic. As for computational trust being reflexive, non-symmetrical and non-transitive, these are properties that SCOUT supports through the plug-ins. Example algorithms that have these properties are discussed in Chapter 7.

11.1.3 Computational Trust Challenges

The challenges to computational trust are discussed in Section 1.4. The role that SCOUT plays in addressing these challenges is as follows:

- *Lack of standards for discovering and accessing evidence.* To address this challenge, EGS has adopted a plug-in approach to evidence gathering. This allows EGS to be extended with new evidence discovery and gathering protocols. As EGS is a middleware service, any changes to evidence gathering (e.g. changes to EG-Policies or Evidence Gatherers) is transparent to the trustor's applications and BFS. Basically, developers no longer have to support a variety of evidence discovery and gathering protocols. This also cuts down on the amount of code duplication due to the implementation of the same protocols.
- *Lack of standards for representing evidence.* To address this challenge, EGS has adopted a plug-in approach to handle the variety of evidence representations. This allows EGS to be extended to support new evidence representations. The different evidence representations are all mapped to a standard representation. With only a single evidence representation, this simplifies evidence interpretation. Applications and BFS no longer have to account for the different evidence representations. This also cuts down on the amount of code duplication.
- *Evidence Filtering.* To address this challenge, EGS has adopted the trust-based approach to evidence filtering. The Evidence Source Assessors are implemented as plug-ins. This allows EGS to be extended to support new evidence source assessment algorithms. By moving evidence filtering from the application end to the middleware end, this allows evidence filtering to be conducted transparently during evidence gathering.
- *Computational Trust is subjective and multidimensional.* To address this challenge, the SCOUT services need to support subjectivity and multidimensionality. Subjectivity is supported through the deployed policies: EG-Policies, BF-Policies and ET-Policies. The policies allow the trustor to

subjectively influence evidence gathering, belief formation and emotional trust formation. Multidimensionality is supported through policies and aggregate beliefs. Cognitive trust can be calculated from different combinations of aggregate beliefs. The formation of aggregate beliefs could be influenced by the deployed policies.

11.2 Evaluation of Trust Calculator

The Trust Calculator is evaluated from three different perspectives. The first perspective examines the Trust Calculator through its evolution. The second perspective examines how the Trust Calculator supports the computational trust properties. The last perspective examines how the Trust Calculator addresses the challenges to computational trust.

11.2.1 Evolution

The Trust Calculator was originally a component of SCOUT. It was responsible for aggregating the trusts calculated by the different Trust Managers to form trust in the trustee. As there are many different ways for trust to be calculated, it is the responsibility of the application to describe its trust calculation needs to the Trust Calculator. The description is expressed as a Trust Coordination Plan. An example plan is shown in Figure 77. In the figure, the Trust Calculator first loads and names all of the components needed for trust calculation (lines 1-5). For example, the class “ExperienceManagerProxy” is loaded and given the name “ExpManager”. Next, the Experience Manager is invoked to calculate trust based on compliance experiences (line 7). The Recommendation Manager is then invoked to calculate trust based on compliance recommendations (line 8). The calculated trusts are then aggregated using the weighted average function (line 10). The last step in the Trust Coordination Plan is for the calculated trust to be returned to the requesting application (line 11).

There are a number of weaknesses to the SCOUT based Trust Calculator. These weaknesses have been addressed in the Trust Calculator introduced in Chapter 6. For example, the language for describing trust calculation has been simplified. Instead of a language that uses dollar sign (\$) to denote variables, an at sign (@) to denote information that needs to be filled in and square brackets ([]) for list creation, trust

calculation is structured as a tree that is described in XML. This in turn simplifies the specification of computational trust formation. Moreover, instead of calculating trust from other calculated trusts in an ad hoc manner, computational trust formation has a solid socio-psychological underpinning and is based on cognitive trust and emotional trust. The final major change is the move of the Trust Calculator from middleware end to the application end. This is to capture the fact that although the applications share the same emotional trust and beliefs in a trustee, computational trust formation is still adaptable to each application's decision making needs.

```

1  load {
2    ExpManager: "ExperienceManagerProxy";
3    RecManager: "RecommendationManagerProxy";
4    Math: "AggregationAlgorithms";
5  }
6  {
7    $expTrust = ExpManager.calculateTrust("@Provider", "Compliance");
8    $recTrust = RecManager.calculateTrust("@Provider", "Compliance");
9
10   $trust = Math.weightedAvg([$expTrust, $recTrust]);
11   return $trust;
12 }

```

Figure 77: Trust Coordination Plan

11.2.2 Computational Trust Properties

The Trust Calculator is designed with the computational trust properties listed in Section 1.2.3 in mind. The Trust Calculator supports computational trust that is quantifiable, comparable, subjective, multidimensional and dynamic. As for computational trust being reflexive, non-symmetrical and non-transitive, these are properties that are supported through the computational trust formation algorithms (i.e., nodes).

11.2.3 Computational Trust Challenges

The Trust Calculator addresses the challenge of *computational trust being subjective and multidimensional*. The Trust Calculator supports TcTemplates that allows subjective information concerning computational trust formation to be filled in by the trustor. As for multidimensionality, this is supported through the Trust Calculation Planner that maps the different factors that influence computational trust formation to TcPlans.

11.3 Summary

The computational trust architecture proposed in Chapter 4 is evaluated in this chapter. The evaluation is conducted from three different perspectives. The first perspective focuses on the evolution of the proposed architecture. The second perspective examines the architecture in terms of its support of the computational trust properties. The final perspective addresses how the proposed architecture meets the computational trust challenges.

Chapter 12

12 Conclusions and Future Directions

This chapter is divided into two sections. In the first section, the contributions of the thesis are summarized. In the second section, the focus is on possible future directions that the computational trust research could take.

12.1 Conclusions

Trust has been shown to play an important role in everyday decision making. In [84], Marsh showed that trust can also be applied to decision making in computing. In this thesis, a trust model known as computational trust is introduced. The model is based on the work of Lewis and Weigert [75]. The computational trust model calls for the formation of computational trust from cognitive trust and emotional trust. Cognitive trust is formed from the trustor's beliefs in a trustee. Belief can be calculated based on the available evidence. This thesis has identified eight different types of aggregate beliefs and four different types of evidence. As for emotional trust, it is based on recognition of the trustee.

The implementation of the computational trust model can be challenging. Four different challenges have been identified. The challenges are the lack of standards for discovering and accessing evidence, the lack of standards for representing evidence, evidence filtering and computational trust being subjective and multidimensional. To address these challenges, a computational trust architecture is proposed. The architecture consists of a SCOUT middleware, Evidence Repository and Trust Calculators. The SCOUT middleware can be thought of as the personification of the trustor. It provides information about the trustor that is shared by all of the trustor's applications. As for the Evidence Repository, it is a storage for all of the available evidence concerning the trustees. The Trust Calculator is associated with an application. It is responsible for the formation computational trust that meets the decision making needs of the application.

SCOUT consists of three services: Evidence Gathering Service (EGS), Belief Formation Service (BFS) and Emotional Trust Service (ETS). EGS is responsible for evidence gathering. The gathered evidence are normalized before being stored in the Evidence Repository. The evidence stored in the repository are accessible to all of the trustor's applications. EGS also calculates evidence source trust for evidence filtering. By taking into account the trustor's feedbacks, EGS could learn to avoid evidence from untrusted or distrusted evidence sources. As for BFS, it is responsible for belief formation. The evidence needed for belief formation can be retrieved from the Evidence Repository. If there are not enough evidence, BFS could contact EGS for evidence gathering. As for ETS, it is responsible for emotional trust formation. Emotional trust is determined by the trustee's position in a hierarchy.

A Trust Calculator is responsible for computational trust formation. The aggregate beliefs and emotional trusts needed for computational trust formation can be obtained from the SCOUT middleware. A TcPlan can be created to describe the formation of computational trust. A TcPlan is written in XML. It is structured as a tree with tree nodes representing operations on aggregate beliefs and trusts. The TcPlanner of the Trust Calculator is responsible for mapping the factors that could influence computational trust formation to a TcPlan. If the TcPlan has information missing, it is known as a TcTemplate. The TcTemplate needs to be parameterized before being used in computational trust formation. The TcEngine of the Trust Calculator is responsible for the instantiation and execution of the selected TcPlan. The calculated computational trust is then returned to the application.

The proposed computational trust architecture has been implemented. Algorithms have also been developed for belief formation and evidence source assessment. Experiments have also been conducted to demonstrate the importance of evidence source availability and how different algorithms impact the performance of computational trust. The computational trust architecture has been evaluated. Finally, a movie scenario and a web service scenario are presented. The scenarios are used to demonstrate how the computational trust architecture can support computational trust formation. In conclusion, this thesis has introduced a middleware approach to computational trust formation. The

approach should help address the challenges of implementing a computational trust model.

12.2 Future Work

There are a number of areas in the proposed architecture that can be improved with more work. The areas include:

- *Evidence filtering.* There are many different ways to perform evidence filtering. The computational trust architecture currently only supports the formation of evidence source trust from feedbacks. Other evidence filtering approaches may include the use of recommendations, reputations and signals to calculate evidence source trust. Works such as [125] and [141] layer multiple evidence filtering approaches together to perform evidence filtering. These alternative approaches should be supported in the computational trust architecture.
- *Privacy.* As pointed out in Section 2.3, privacy should be taken into account during evidence gathering and distribution. How a trustor's privacy concerns should be expressed and integrated into SCOUT is an area open to future work.
- *SCOUT policies.* A policy-based approach has been adopted to deal with computational trust being subjective and multidimensional. One weakness of the policy-based approach is that the specification of policies could still be too challenging for a trustor. A possible solution is to have default policies for different trustee types. For example, there would be default policies for movies and different types of web services. These policies can be downloaded and deployed to satisfy common decision making needs. Tools could also be made available to help in the editing of SCOUT policies. This is an area that requires more investigation and may involve other research areas such as human-computer interaction.
- *Multidimensionality of trust.* Trust in one dimension can influence trust in a different dimension. For example, cognitive or emotional trust in a movie could be influenced by cognitive or emotional trust in the movie's director and actors.

The relationship between dimensions is currently not explicitly represented in the proposed architecture. Whether support is needed for representing the relationships is an area that requires further exploration.

- *Emotional trust.* Currently, emotional trust policies are created by the trustor. More investigations should be conducted on whether it is possible to automate some of the emotional trust assignments. This could involve access to the trustor's social graph or the adoption of research in areas such as context-awareness and emotional recognition.
- *Conceptual model.* In Lewis and Weigert's conceptual model, cognitive, emotional and behavior trust all influence each other. The influences have been simplified in the computational trust architecture. It would be interesting to investigate whether some of the simplifications can be removed to create a more realistic computational trust model.
- *TcPlan.* A GUI editor could be created to simplify TcPlan creation. Instead of passing XML to the TcEngine for parsing, future work could include TcPlan compilation to ensure faster computational trust calculation.
- *Middleware management.* The trustor is currently tasked with the management of the middleware (e.g. configuration the middleware or deploying policies to the middleware). The development of an agent that stands as a representative of the trustor in middleware management is an area that requires further investigation.
- *Experiments.* More experiments should be conducted to better understand computational trust formation. Specifically, the assumptions made in Chapter 9 should be removed to create more realistic experiments. Metrics such as computational cost and storage cost of the proposed algorithms should also be measured. The societal benefit of using computational trust is another metric worth exploring. Finally, experts in other disciplines such as psychologists and sociologists may need to be consulted to determine the best way to validate

emotional trust, to capture a trustor's subjective views and to compare computational trust with the human notion of trust.

- *Evaluation.* To gain more insight into decision making based on computational trust, the computational trust architecture could be evaluated using more scenarios. Example scenarios may include access control and crowdsourcing. Applicability to more scenarios could help demonstrate the genericity of the proposed architecture.
- *Integration.* For computational trust to be widely used, it should be integrated into the trustor's everyday workflow. More research needs to be done to examine how the proposed architecture can be integrated into either a business or consumer oriented workflow.

References

- [1] A. Abdul-Rahman and S. Hailes, "Supporting trust in virtual communities," in *Proceedings of the 33rd Annual Hawaii International Conference on System Sciences*, vol. 6, 2000, p. 9.
- [2] K. Aberer and Z. Despotovic, "Managing trust in a peer-2-peer information system," in *Proceedings of the 10th International Conference on Information and Knowledge Management*, 2001, pp. 317-324.
- [3] Advogato. [Online]. <http://www.advogato.org> (accessed: February 7, 2010).
- [4] affect. Dictionary.com Unabridged. [Online]. <http://dictionary.reference.com/browse/affect> (accessed: February 3, 2010).
- [5] A. Ali, S. Ludwig, and O. Rana, "A cognitive trust-based approach for web service discovery and selection," in *3rd IEEE European Conference on Web Services*, 2005, pp. 38-49.
- [6] A. Ali, O. Rana, and D. Walker, "WS-QoC: Measuring quality of service compliance," in *Proceeding of the 2nd International Conference on Service Oriented Computing (ICSOC04)*, New York, 2004, pp. 24-33.
- [7] Amazon. [Online]. <http://www.amazon.com> (accessed: February 7, 2010).
- [8] Amazon Elastic Compute Cloud. [Online]. <http://aws.amazon.com/ec2> (accessed: February 7, 2010).
- [9] Amazon Mechanical Turk. [Online]. <http://aws.amazon.com/mturk> (accessed: February 7, 2010).
- [10] R. Axelrod, *The evolution of cooperation*. New York: Basic Books, 1985.
- [11] F. Azzedin and M. Maheswaran, "Evolving and managing trust in grid computing systems," in *IEEE Canadian Conference on Electrical Computer Engineering*, 2002, pp. 1424-1429.

- [12] belief. Merriam-Webster's Dictionary of Law. [Online]. <http://dictionary.reference.com/browse/belief> (accessed: February 3, 2010).
- [13] E. Bertino, E. Ferrari, and A. Squicciarini, "Trust negotiations: concepts, systems, and languages," *Computing in Science and Engineering*, vol. 6, no. 4, pp. 27-34, 2004.
- [14] Best Buy. [Online]. <http://www.bestbuy.com> (accessed: February 7, 2010).
- [15] M. Blaze, J. Feigenbaum, and A.D. Keromytis, "KeyNote: Trust management for public-key infrastructures," *Lecture Notes in Computer Science*, vol. 1550, pp. 59-63, 1999.
- [16] M. Blaze, J. Feigenbaum, and J. Lacy, "Decentralized trust management," in *IEEE Conference on Security and Privacy*, Oakland, 1996, pp. 164-173.
- [17] K. Blomqvist, "The many faces of trust," *Scandinavian Journal of Management*, vol. 13, no. 3, pp. 271-286, 1997.
- [18] D. Boyd, "Making sense of privacy and publicity," in *South by SouthWest (SXSW)*, Austin, 2010.
- [19] S. Brin and L. Page, "The anatomy of a large-scale hypertextual web search engine," *Computer networks and ISDN systems*, vol. 30, no. 1-7, pp. 107-117, 1998.
- [20] S. Buchegger and J. Y. Le Boudec, "A Robust Reputation System for Peer-to-Peer and Mobile Ad-hoc Networks," in *Proceeding of the 2nd Workshop on Economics of Peer-to-Peer Systems (P2PEcon)*, Cambridge, 2004, pp. 150-155.
- [21] CA/Browser Forum. [Online]. <http://www.cabforum.org> (accessed: February 7, 2010).
- [22] V. Cahill et al., "Using trust for secure collaboration in uncertain environments," *IEEE Pervasive Computing*, vol. 2, no. 3, pp. 52-61, 2003.
- [23] L. Capra, "Engineering human trust in mobile system collaborations," *ACM*

SIGSOFT Software Engineering Notes, vol. 29, no. 6, pp. 116-125, 2004.

- [24] J. Carter, E. Bitting, and A. Ghorbani, "Reputation Formalization for an Information-Sharing Multi-Agent System," *Computational Intelligence*, vol. 18, no. 4, pp. 515-534, November 2002.
- [25] C. Castelfranchi and R. Falcone, "Principles of trust for MAS: cognitive anatomy, social importance, and quantification," in *ICMAS '98: Proceedings of the 3rd International Conference on Multi Agent Systems*, Paris, 1998, pp. 72-79.
- [26] M. Chen and J. Singh, "Computing and using reputations for internet ratings," in *Proceedings of the 3rd ACM conference on Electronic Commerce*, New York, 2001, pp. 154-162.
- [27] K. Chopra and W. Wallace, "Trust in electronic environments," in *Proceedings of the 36th Annual Hawaii International Conference on System Sciences*, 2003, pp. 331-340.
- [28] Y. Chu, J. Feigenbaum, B. Lamacchia, P. Resnick, and M. Strauss, "REFEREE: trust management for Web applications," *Computer Networks and ISDN Systems*, vol. 29, no. 8-13, pp. 953-964, 1997.
- [29] L. Clement, A. Hately, C. Riegen, and T. Rogers. UDDI Version 3.0.2. [Online]. http://www.uddi.org/pubs/uddi_v3.htm (accessed: February 4, 2011).
- [30] F. Cornelli, E. Damiani, S. di Vimercati, S. Paraboschi, and P. Samarati, "Choosing reputable servants in a P2P network," in *Proceedings of the 11th International Conference on World Wide Web*, New York, 2002, pp. 376-386.
- [31] C. Corritore, B. Kracher, and S. Wiedenbeck, "On-line trust: concepts, evolving themes, a model," *International Journal of Human-Computer Studies*, vol. 58, no. 6, pp. 737-758, 2003.
- [32] E. Damiani, S. Paraboschi, P. Samarati, and F. Violante, "A reputation-based approach for choosing reliable resources in peer-to-peer networks," in *Proceedings of the 9th ACM Conference on Computer and Communications*

Security, 2002, pp. 207-216.

- [33] N. Damianou, N. Dulay, E. Lupu, and M. Sloman, "The ponder policy specification language," in *Proceedings of the International Workshop on Policies for Distributed Systems and Networks*, 2001, pp. 18–38.
- [34] N. Dawar, P. Parker, and L. Price, "A cross-cultural study of interpersonal information exchange.," *Journal of International Business Studies*, vol. 27, no. 3, pp. 497-516, 1996.
- [35] M. Deutsch, "Cooperation and trust: Some theoretical notes," *Nebraska Symposium on Motivation*, pp. 275-319, 1962.
- [36] J. Douceur, "The Sybil Attack," in: *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS '01)*, London, 2002, pp. 251-260.
- [37] Drooms. [Online]. <http://www.jboss.org/drooms> (accessed: May 16, 2010).
- [38] eBay. [Online]. <http://www.ebay.com> (accessed: February 7, 2010).
- [39] Epinions. [Online]. <http://www.epinions.com> (accessed: February 7, 2010).
- [40] S. Etalle, J. den Hartog, and S. Marsh, "Trust and punishment," in *Proceedings of the 1st International Conference on Autonomic Computing and Communication Systems*, Rome, 2007, pp. 251-260.
- [41] R. Falcone, G. Pezzulo, and C. Castelfranchi, "A fuzzy approach to a belief-based trust computation," *Trust, reputation, and security: theories and practice*, pp. 55-60, 1999.
- [42] S. Fiske and S. Taylor, *Social Cognition*, 2nd ed.: McGraw-Hill Publishing, 1991.
- [43] C. Fung, J. Zhang, I. Aib, and R. Boutaba, "Robust and scalable trust management for collaborative intrusion detection," in *IFIP/IEEE International Symposium on Integrated Network Management*, 2009, pp. 33-40.

- [44] D. Gambetta, "Can We Trust Trust?," *Trust: Making and Breaking Cooperative Relations, electronic edition*, pp. 213-237, 2000.
- [45] D. Gefen, V. Srinivasan Rao, and N. Tractinsky, "The conceptualization of trust, risk and their relationship in electronic commerce: the need for clarifications," in *Proceedings of the 36th Annual International Conference on System Sciences*, Hawaii, 2003, pp. 192-201.
- [46] GlassFish. [Online]. <https://glassfish.dev.java.net/> (accessed: May 16, 2010).
- [47] M. Gobe, *Emotional branding: the new paradigm for connecting brands to people*, 1st ed.: Allworth Press, 2001.
- [48] J. Golbeck, B. Parsia, and J. Hendler, "Trust networks on the semantic web," *Cooperative Information Agents VII*, vol. 1, no. 1, pp. 238-249, 2003.
- [49] P. Gowder, "Review of The Transparent Society by David Brin, Data Smog by David Shenk," *Harvard Journal of Law & Technology*, vol. 2995, no. 2, pp. 513-532, 1999.
- [50] T Grandison and M Sloman, "A survey of trust in internet applications," *IEEE Communications Surveys and Tutorials*, vol. 3, no. 4, pp. 2-16, 2000.
- [51] T. Grandison and M. Sloman, "Specifying and analysing trust for internet applications," in *Proceedings of the 2nd IFIP Conference on E-Commerce, E-Business, E-Government (I3E 2002)*, Lisbon, 2002, pp. 145-157.
- [52] E. Gray, C. Jensen, S. Weber, J. Seigneur, and C. Yong, "Towards a framework for assessing trust-based admission control in collaborative ad hoc applications," Department of Computer Science, Trinity College, Dublin, Tech Report 2002.
- [53] N. Griffiths, "Task delegation using experience-based multi-dimensional trust," in *Proceedings of the 4th International Joint Conference on Autonomous Agents and Multiagent Systems*, New York, 2005, pp. 489-496.
- [54] N. Griffiths and M. Younas, "Fuzzy trust for peer-to-peer systems," in

Proceedings of the 26th IEEE International Conference on Distributed Computing Systems Workshops, 2006, pp. 73-79.

- [55] M. Gupta, P. Judge, and M. Ammar, "A reputation system for peer-to-peer networks," in *Proceedings of the 13th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, 2003, pp. 144-152.
- [56] R. Gupta and A. Somani, "Reputation management framework and its use as currency in large-scale peer-to-peer networks," in *Proceedings of the 4th International Conference on Peer-to-Peer Computing*, 2004, pp. 124-132.
- [57] R. Housley, W. Polk, W. Ford, and D. Solo, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," RFC 2002.
- [58] T. Huynh, "A personalized framework for trust assessment," in *Proceedings of the 2009 ACM Symposium on Applied Computing*, New York, 2009, pp. 1302-1307.
- [59] T. Huynh, N. Jennings, and N. Shadbolt, "An integrated trust and reputation model for open multi-agent systems," *Autonomous Agents and Multi-Agent Systems*, vol. 13, no. 2, pp. 119-154, 2006.
- [60] IMDb. [Online]. <http://www.imdb.com> (accessed: February 7, 2010).
- [61] D. Johnson and K. Grayson, "Cognitive and affective trust in service relationships," *Journal of Business Research*, vol. 58, no. 4, pp. 500-507, 2005.
- [62] A. Jøsang and R. Ismail, "The beta reputation system," in *Proceedings of the 15th Bled Electronic Commerce Conference*, 2002, pp. 324-337.
- [63] A. Jøsang, R. Ismail, and C. Boyd, "A survey of trust and reputation systems for online service provision," *Decision Support Systems*, vol. 43, no. 2, pp. 618-644, 2007.
- [64] A. Jøsang and S. Pope, "Semantic constraints for trust transitivity," in

Proceedings of the 2nd Asia-Pacific Conference on Conceptual Modelling, Newcastle, 2005, pp. 68-77.

- [65] A. Jøsang and S. Presti, "Analyzing the relationship between risk and trust," in *Proceedings of the 2nd International Conference on Trust Management*, Oxford, 2004, pp. 135-145.
- [66] Jtpl. [Online]. <http://jtpl.sourceforge.net/> (accessed: May 16, 2010).
- [67] S. Kalepu, S. Krishnaswamy, and S. Loke, "Reputation = f(user ranking, compliance, verity)," in *IEEE International Conference on Web Services*, 2004, pp. 200-207.
- [68] S. Kamvar, M. Schlosser, and H. Garcia-Molina, "The eigentrust algorithm for reputation management in P2P networks," in *Proceedings of the 12th International Conference on World Wide Web*, New York, 2003, pp. 640-651.
- [69] K. Kelton, K. Fleischmann, and W. Wallace, "Trust in Digital Information," *Journal of the American Society for Information Science*, vol. 59, no. 3, pp. 363-374, 2008.
- [70] M. Kosfeld, M. Heinrichs, P. Zak, U. Fischbacher, and E. Fehr, "Oxytocin increases trust in humans," *Nature*, vol. 435, no. 7042, pp. 673-676, 2005.
- [71] E. Kotsovinos and A. Williams, "BambooTrust: practical scalable trust management for global public computing," in *Proceedings of the 2006 ACM Symposium on Applied computing*, 2006, pp. 1893-1897.
- [72] B. Lahno, "On the emotional character of trust," *Ethical Theory and Moral Practice*, vol. 4, no. 2, pp. 171-189, 2001.
- [73] N. Leveson, *Safeware: system safety and computers*. New York: Addison-Wesley Professional, 1995.
- [74] B. Levine, C. Shields, and N. Margolin, "A Survey of Solutions to the Sybil Attack," University of Massachusetts Amherst, Amherst, Tech Report 2006.

- [75] J. Lewis and A. Weigert, "Trust as a social reality," *Social Forces*, vol. 63, no. 4, pp. 967 - 985, 1985.
- [76] Z. Liang and W. Shi, "Enforcing cooperative resource sharing in untrusted P2P computing environments," *Mobile Networks and Applications*, vol. 10, no. 6, pp. 971-983, 2005.
- [77] Z. Liang and W. Shi, "PET: A PErsonalized Trust Model with Reputation and Risk Evaluation for P2P Resource Sharing," in *Proceedings of the 38th Annual Hawaii International Conference on System Sciences*, 2005, pp. 201-210.
- [78] N. Limam and R. Boutaba, "QoS and reputation-aware service selection," in *IEEE Network Operations and Management Symposium*, 2008, pp. 403-410.
- [79] C. Lin, V. Varadharajan, Y. Wang, and V. Pruthi, "Enhancing grid security with trust management," in *IEEE International Conference on Services Computing*, 2004, pp. 303-310.
- [80] J. Liu and V. Issarny, "An incentive compatible reputation mechanism for ubiquitous computing environments," *International Journal of Information Security*, vol. 6, no. 5, pp. 297-311, 2007.
- [81] D. Lucking-Reiley, D. Bryan, N.i Prasad, and D. Reeves, "Pennies From Ebay: the Determinants of Price in Online Auctions," *Journal of Industrial Economics*, vol. 55, no. 2, pp. 223-233, 2007.
- [82] N. Luhmann, "Familiarity, Confidence, Trust: Problems and Alternatives," in *Trust: Making and breaking cooperative relations.:* Basil Blackwell, 2000, pp. 94-107.
- [83] D. Manchala, "E-commerce trust metrics and models," *IEEE Internet Computing*, vol. 4, no. 2, pp. 36-44, 2000.
- [84] S. Marsh, "Formalising trust as a computational concept," University of Stirling, Department of Computer Science and Mathematics, PhD Thesis 1994.

- [85] S. Marsh, "Optimism and pessimism in trust," in *Proceedings of the Ibero-American Conference on Artificial Intelligence*, 1994.
- [86] S. Marsh and M. Dibben, "Trust, untrust, distrust and mistrust - an exploration of the dark(er) side," in *Proceedings of the 3rd iTrust International Conference*, vol. 3477, Paris, 2005, pp. 17-33.
- [87] E Michael Maximilien and Munindar P Singh, "Conceptual model of web service reputation," *ACM Special Interest Group on Management of Data (SIGMOD) Record*, vol. 31, no. 4, pp. 36-41, 2002.
- [88] R. Mayer, J. Davis, and F. Schoorman, "An integrative model of organizational trust," *The Academy of Management Review*, vol. 20, no. 3, pp. 709-734, 1995.
- [89] J. McClave and F. Dietrich, *Statistics*, 4th ed., 1988.
- [90] L. Mekouar, Y. Iraqi, and R. Boutaba, "Peer-to-peer's most wanted: malicious peers," *Computer Networks*, vol. 50, no. 4, pp. 545-562, 2006.
- [91] Metacritic. [Online]. <http://www.metacritic.com> (accessed: February 7, 2010).
- [92] B Misztal, *Trust in Modern Societies: The Search for the Bases of Social Order.: Polity*, 1996.
- [93] conceptual model. Dictionary.com's 21st Century Lexicon. [Online]. http://dictionary.reference.com/browse/conceptual_model (accessed: February 3, 2010).
- [94] L. Mui, M. Mohtashemi, and A. Halberstadt, "Notions of reputation in multi-agents systems: a review," in *Proceedings of the 1st International Joint Conference on Autonomous Agents and Multiagent Systems*, New York, 2002, pp. 280-287.
- [95] MVEL. [Online]. <http://mvel.codehaus.org/> (accessed: May 16, 2010).
- [96] A. Nadalin, M. Goodner, M. Gudgin, A. Barbir, and H. Granqvist, "WS-Trust 1.3," *OASIS Standard*, 2007.

- [97] A. Nadalin, C. Kaler, R. Monzillo, and P. Hallam-Baker, "Web Services Security: SOAP Message Security 1.1," *OASIS Standard*, 2006.
- [98] E. Papalilo and B. Freisleben, "Towards a flexible trust model for grid environments," in *Proceedings of the 1st International Conference of Grid Services Engineering and Management*, Erfurt, 2004, pp. 35–65.
- [99] Quartz. [Online]. <http://www.quartz-scheduler.org/> (accessed: May 16, 2010).
- [100] S. Ramchurn, N. Jennings, C. Sierra, and L. Godo, "Devising a trust model for multi-agent interactions using confidence and reputation," *Applied Artificial Intelligence*, vol. 18, no. 9-10, pp. 833-852, 2004.
- [101] rating. (2004) The American Heritage® Dictionary of the English Language, Fourth Edition. [Online]. <http://dictionary.reference.com/browse/rating> (accessed: May 25, 2010).
- [102] P. Resnick and R. Zeckhauser, "Trust among strangers in Internet transactions: Empirical analysis of eBay's reputation system," *Advances in Applied Microeconomics: A Research Annual*, vol. 11, pp. 127-157, 2002.
- [103] D. Reynolds, "Jena 2 inference support," HP Labs, Technical Report 2007.
- [104] M. Richardson, R. Agrawal, and P. Domingos, "Trust management for the semantic web," in *Proceedings of the 2nd International Semantic Web Conference*, 2003, pp. 351-368.
- [105] RosettaNet. [Online]. <http://www.rosettanet.org> (accessed: February 4, 2011).
- [106] Rotten Tomatoes. [Online]. <http://www.rottentomatoes.com> (accessed: February 7, 2010).
- [107] J. B. Rotter, "Generalized expectancies for interpersonal trust," *American Psychologist*, vol. 26, no. 5, pp. 443-452, 1971.
- [108] S. Ruohomaa, L. Viljanen, and L. Kutvonen, "Guarding enterprise collaborations with trust decisions - the TuBE approach," in *Proceedings of the 1st International*

Workshop on Interoperability Solutions to Trust, Security, Policies and QoS for Enhanced Enterprise Systems, 2006.

- [109] J. Sabater, "Evaluating the ReGreT system," *Applied Artificial Intelligence*, vol. 18, no. 9-10, pp. 797-813, 2004.
- [110] J. Sabater and C. Sierra, "Reputation and social network analysis in multi-agent systems," in *Proceedings of the 1st International Joint Conference on Autonomous Agents and Multiagent Systems*, 2002, pp. 475-482.
- [111] M. Schillo, P. Funk, and M. Rovatsos, "Using trust for detecting deceitful agents in artificial societies," *Applied Artificial Intelligence*, vol. 14, no. 8, pp. 825-848, 2000.
- [112] J. Schneider, G. Kortuem, J. Jager, S. Fickas, and Z. Segall, "Disseminating trust information in wearable communities," *Personal Technologies*, vol. 4, no. 4, pp. 245-248, 2000.
- [113] J. Seigneur and C. Jensen, "Trading privacy for trust," *Lecture notes in computer science*, vol. 2995, pp. 93-107, 2004.
- [114] S. Sen and N. Sajja, "Robustness of reputation-based trust: boolean case," in *Proceedings of the 1st International Joint Conference on Autonomous Agents and Multiagent Systems*, 2002, pp. 288-293.
- [115] W. Sherchan, S. Krishnaswamy, and S. Loke, "Relevant past performance for selecting web services," in *Proceedings of the 5th International Conference on Quality Software*, Melbourne, 2005, pp. 439-445.
- [116] R. Sherwood, S. Lee, and B. Bhattacharjee, "Cooperative peer groups in NICE," *Computer Networks*, vol. 50, no. 4, pp. 523-544, 2006.
- [117] V. Shmatikov and C. Talcott, "Reputation-based trust management," *Journal of Computer Security*, vol. 13, no. 1, pp. 167-190, 2005.
- [118] M. Siegrist, H. Gutscher, and T. Earle, "Perception of risk: the influence of

- general trust, and general confidence," *Journal of Risk Research*, vol. 8, no. 2, pp. 145-156, 2005.
- [119] H. Simon, *The Sciences of the Artificial*, 3rd ed. Cambridge: The MIT Press, 1996.
- [120] Slashdot. [Online]. <http://slashdot.org> (accessed: February 7, 2010).
- [121] C. Smith, "Trust and confidence: possibilities for social work in 'high modernity'," *British Journal of Social Work*, vol. 31, no. 2, pp. 287-305, 2001.
- [122] R. Solomon and F. Fernando, *Building trust in business, politics, relationships, and life*. New York: Oxford University Press, 2001.
- [123] S. Song, K. Hwang, R. Zhou, and Y. Kwok, "Trusted P2P transactions with fuzzy reputation aggregation," *IEEE Internet Computing*, vol. 9, no. 6, pp. 24-34, 2005.
- [124] M. Spence, "Job market signaling," *The Quarterly Journal of Economics*, vol. 87, no. 3, pp. 355-374, August 1973.
- [125] M. Srivatsa, L. Xiong, and L. Liu, "TrustGuard: countering vulnerabilities in reputation management for decentralized overlay networks," in *Proceedings of the 14th international conference on World Wide Web*, 2005, pp. 431-440.
- [126] Stack Overflow. [Online]. <http://stackoverflow.com> (accessed: February 7, 2010).
- [127] N. Stakhanova, S. Basu, J. Wong, and O. Stakhanov, "Trust framework for p2p networks using peer-profile based anomaly technique," in *ICDCS Workshops*, 2005, pp. 203-209.
- [128] W. Teacy, J. Patel, N. Jennings, and M. Luck, "Travos: trust and reputation in the context of inaccurate information sources," *Autonomous Agents and Multi-Agent Systems*, vol. 12, no. 2, pp. 183-198, 2006.
- [129] W. Teacy, J. Patel, N. Jennings, and M. Luck, "Travos: Trust and reputation in the context of inaccurate information sources," *Autonomous Agents and Multi-Agent Systems*, vol. 12, no. 2, pp. 183-198, 2006.

- [130] H. Tran, M. Hitchens, V. Varadharajan, and P. Watters, "A trust based access control framework for P2P file-sharing systems," in *Proceedings of the 38th Annual Hawaii International Conference on System Sciences*, 2005, pp. 302-311.
- [131] "Trust in the wired americas," Cheskin Research, 2000.
- [132] trust, *The American Heritage Dictionary of the English Language*, Fourth Edition, 2004, <http://dictionary.reference.com/browse/trust>. (accessed: February 3, 2010).
- [133] K. Walsh and E. Sirer, "Experience with an object reputation system for peer-to-peer filesharing," in *Proceedings of the 3rd Conference on Networked Systems Design & Implementation*, 2006, pp. 1-14.
- [134] Y. Wang and J. Vassileva, "Bayesian network trust model in peer-to-peer networks," in *Proceedings of the 2nd International Workshop on Agents and Peer-to-Peer Computing*, Berlin, 2004, pp. 23-34.
- [135] Web of Trust (WOT). [Online]. <http://www.mywot.com> (accessed: May 19, 2010).
- [136] J. Weng, C. Miao, and A. Goh, "A robust reputation system for the grid," in *Proceedings of the 6th IEEE International Symposium on Cluster Computing and the Grid*, 2006, pp. 548-551.
- [137] A. Whitby, A. Jøsang, and J. Indulska, "Filtering out unfair ratings in bayesian reputation systems," in *Proceedings of the 3rd International Joint Conference on Autonomous Agenst and Multi Agent Systems*, Rome, 2004, pp. 106-117.
- [138] WSO2 Governance Registry. [Online]. <http://wso2.com/products/governance-registry> (accessed: May 16, 2010).
- [139] L. Xiong and L. Liu, "Peertrust: supporting reputation-based trust for peer-to-peer electronic communities," *IEEE transactions on Knowledge and Data Engineering*, vol. 16, no. 7, pp. 843-857, 2004.

- [140] Z. Xu, P. Martin, W. Powley, and F. Zulkernine, "Reputation-enhanced qos-based web services discovery," in *IEEE International Conference on Web Services (ICWS)*, Salt Lake City, 2007, pp. 249–256.
- [141] Y. Yang, Y. Sun, S. Kay, and Q. Yang, "Defending online reputation systems against collaborative unfair raters through signal modeling and trust," in *Proceedings of the 2009 ACM symposium on Applied Computing*, New York, 2009, pp. 1308-1315.
- [142] B. Yu and M. Singh, "An evidential model of distributed reputation management," in *Proceedings of the 1st International Joint Conference on Autonomous Agents and Multiagent Systems*, New York, 2002, pp. 294-301.
- [143] B. Yu, M. Singh, and K. Sycara, "Developing trust in large-scale peer-to-peer systems," in *IEEE 1st Symposium on Multi-Agent Security*, 2004, pp. 1-10.
- [144] G. Zacharia, A. Moukas, and P. Maes, "Collaborative reputation mechanisms for electronic marketplaces," *Decision Support Systems*, vol. 29, no. 4, pp. 371-388, 2000.
- [145] D. Zhu and M. Mutka, "Promoting cooperation among strangers to access internet services from an ad hoc network," *Pervasive and Mobile Computing*, vol. 1, no. 2, pp. 213-236, 2005.
- [146] P. Zimmermann, *The Official PGP User's Guide*. Cambridge, USA: MIT Press, 1995.
- [147] Y. Zuo and B. Panda, "Information trustworthiness evaluation based on trust combination," in *ACM Symposium on Applied Computing*, 2006, pp. 1880-1885.

Appendices

Appendix A: Syntax of SCOUT Policies

The syntax for SCOUT policies is shown in Table 23. Everything in bold is a language keyword. Everything in bold-italic is automatically inserted by the architecture. Everything in italics can be set by the policy author. Square brackets ([]) are used to indicate optional elements while braces ({ }) are used to indicate zero or more elements.

As an example, consider EG-Policy, the computational trust architecture automatically inserted the package and import statements to ensure that the policies are placed in the right package and that basic classes are available to the policy author. Although neglected in the discussion in the thesis, each policy should be assigned a policy name. Optionally, a priority could be assigned to a policy. The default *salience* is set to zero if no priority is assigned. The conditional part (i.e., *when* part) of EG-Policy must consist of at least *Trustee* and *Belief*. *Hint* however is optional. It is possible for there to be one or more *Hint*. Finally, *RHS* consists of actions to be taken if a policy is triggered. Any legal Java statement is acceptable though in some cases libraries may need to be imported. The same interpretation also applies to EA-Policy, BF-Policy, AF-Policy and ET-Policy.

Table 23: Syntax of SCOUT Policies

SCOUT Policies	Syntax
Evidence Gathering Policy (EG-Policy)	<pre> <i>package ca.uwo.evidence.egPolicies</i> <i>import ca.uwo.evidence.*</i> policy <i>policyName</i> [salience <i>priority</i>] when <i>trustee-ref: Trustee(expression)</i> <i>belief-ref: Belief(expression)</i> { <i>hint-ref: Hint(expression)</i> } then <i>RHS</i> end </pre>

Evidence Source Assessment Policy (EA-Policy)	<pre> package ca.uwo.evidence.eaPolicies import ca.uwo.evidence.* policy policyName [salience priority] when trustee-ref: Trustee(expression) feedback-ref: BeliefFeedback(expression) then RHS end </pre>
Belief Formation Policy (BF-Policy)	<pre> package ca.uwo.belief.bfPolicies import ca.uwo.belief.* policy policyName [salience priority] when trustee-ref: Trustee(expression) aggBelief-ref: AggregateBelief(expression) { hint-ref: Hint(expression) } then RHS end </pre>
Aggregate Belief Feedback Policy (AF-Policy)	<pre> package ca.uwo.belief.afPolicies import ca.uwo.belief.* policy policyName [salience priority] when trustee-ref: Trustee(expression) aggFeedback-ref: AggregateBeliefFeedback(expression) { hint-ref: Hint(expression) } then RHS end </pre>
Emotional Trust Policy (ET-Policy)	<pre> package ca.uwo.trust.etPolicies import ca.uwo.trust.* policy policyName [salience priority] when trustee-ref: Trustee(expression) then RHS end </pre>

Appendix B: Syntax of Subscription Filters

There are two types of filters that can be installed in the Subscription Manager: aggregate belief filter and emotional trust filter. An aggregate belief filter can be used to filter the aggregate belief calculated by Belief Formation Service. The syntax for specifying an aggregate belief filter is shown in Figure 78. When evaluated, an aggregate belief filter should return a Boolean value of “True” or “False”. If “True”, the calculated aggregate belief is returned to the subscriber. Otherwise, the calculated aggregate belief is filtered out. In terms of functions, any function in Java that does not require “import” can be used in an aggregate belief filter. The same also applies to emotional trust filter. Its syntax is shown in Figure 79.

```

1 filter:    condition ((" &&" | "|" ) condition)*
2 condition: expression | expression operator expression
3 expression: function(expression) | "aggBelief.belief" | "aggBelief.reliability" |
4             "lastAggBelief.belief" | "lastAggBelief.reliability" | integer | double | boolean
5 operator:  ">" | ">=" | "<" | "<=" | "==" | "!="

```

Figure 78: Syntax of Aggregate Belief Filter

```

1 filter:    condition ((" &&" | "|" ) condition)*
2 condition: expression | expression operator expression
3 expression: function(expression) | "emoTrust" | "lastEmoTrust" | integer | double |
4             boolean
5 operator:  ">" | ">=" | "<" | "<=" | "==" | "!="

```

Figure 79: Syntax of Emotional Trust Filter

Curriculum Vitae

Name: Chern Har Yew

Post-secondary Education and Degrees: The University of Western Ontario
London, Ontario, Canada
1998-2002 B.Sc.

The University of Western Ontario
London, Ontario, Canada
2002-2005 M.Sc.

The University of Western Ontario
London, Ontario, Canada
2005-2010 Ph.D.

Honours and Awards: International Graduate Student Scholarship (IGSS)
2005-2010

Related Work Experience

Teaching Assistant
The University of Western Ontario
2002-2008

Instructor
The University of Western Ontario
2005

Publications:

C. Yew, V. Tasic and H. Lutfiyya, "On integrating trust into business-driven management of web services and their compositions," in *Proceedings of the 3rd international workshop on Business-driven IT Management (BDIM)*, Salvador, pp. 102-104, 2008.

C. Yew and H. Lutfiyya, "Introduction to MAGNATE: Gathering and Analysis of Trust Evidence", *Proceedings of the 5th international workshop on Business-driven IT Management (BDIM)*, Osaka, pp. 165-166, 2010

C. Yew and H. Lutfiyya, "Middleware-Based Approach to Supporting Trust-Based Service Selection", in *Proceedings of International Symposium on Integrated Network Management (IM)*, Dublin, 2011 (to be published)