Development and Automatic Monitoring of Trust-Aware Service-Based Software

by

Mohammad Gias Uddin

A thesis submitted to the
Department of Electrical and Computer Engineering
in conformity with the requirements for
the degree of Master of Science (Engineering)

Queen's University
Kingston, Ontario, Canada
February 2008

Abstract

Service-based software can be exploited by potentially untrustworthy service requestors while providing services. Given that, it is important to identify, analyze the trust relationships between service providers and requestors, and incorporate them into the service-based software. Treating trust as a nonfunctional requirement (NFR) during software development and monitoring allows clarifying these relationships and measuring the trustworthiness of service requestors. This analysis is facilitated by incorporating trust scenarios and trust models into the software. A trust scenario describes a trust relationship between interested parties based on a specific context. A trust model provides trust equations to measure the trustworthiness of service requestors based on the analysis of service-based interactions. Although much research has been devoted to monitor service quality, to date, no approach has been proposed to develop and automatically monitor service providing software from trust perspectives.

In this thesis, we propose a trust-aware service-based software development framework which utilizes our proposed Unified Modeling Language (UML) extension called UMLtrust (UML for trust scenarios) to specify the trust scenarios of a service provider and incorporates our developed trust model called CAT (Context-Aware Trust) into the software to calculate the trustworthiness of service requestors. The trust scenarios are converted to trust rules to monitor service-based interactions. A service requestor is penalized for the violation of a trust rule and rewarded when no rule is violated. The trustworthiness of the requestor is then calculated (using the equations of CAT) based on the current request, outcomes of previous requests, and recommendations from other service providers. A trust-based service granting algorithm is presented to decide whether a service requestor should be granted the requested service. A trust monitoring architecture is presented which is assumed to reside in each service provider. The monitor uses trust rules from UMLtrust specifications and trust equations from CAT to analyze service-based interactions. The incorporation of the monitor into a provider makes it trust-aware. A trust monitoring algorithm is provided to analyze interactions and make decisions at run-time. A prototype of a file sharing service-based grid is implemented to evaluate the applicability of our framework that confirms the effectiveness of the framework.

Acknowledgements

To almighty Allah, so merciful and so blessing. To my supervisor, Dr. Mohammad Zulkernine, not only for the guidance and supervision of this research work, but also for everything else which has helped me to grow as a person. To Dr. Iqbal, for useful advices on trust model.

To my two brothers and six sisters for their earnest care, exceptional support, and unfathomable love for me.

To the members of QRST lab, and Chanchal Roy and Banani Roy, for all the necessary and unnecessary discussions and comments.

To Dr. Kalam Shahed and his family, for being extremely supportive of me and becoming a family to me over the course of time.

And finally, to the two most special women in my life: First, to my mother, without whose love and affection, I would not and could not be the person who I am right now. Second, to Anonna, for loving me, being with me and keeping faith on me during my hardest and most frustrating moments.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Service-based software provide services to service requestors, where services are offered as software functionalities to achieve goals, perform tasks, and utilize resources [3–6]. Due to the dynamic nature of service-based systems, it may not be always effective to use *'hard security'* mechanisms (*e.g.,* access control [14]) to protect services from malicious and unwanted incidents [27]. For example, a malicious service requestor may try to illegally access others' resources. Such a security breach can be detected using intrusion detection [12, 13] and can be avoided using an access control mechanism. Although the requestor can be denied access to resources by such mechanisms, it can be punished for its misbehavior or rewarded for the proper usage of services using some trust mechanisms.

Compared to the traditional *'hard security'* mechanisms, trust is considered as *'soft security'* [15]. "Trust (or, symmetrically, distrust) is 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" [10]. A trusting agent is called a trustor entity, and a trusted agent is called a trustee entity. A 'context' is a situation which influences the construction of a trust relationship between a trustor and a trustee. For example, a provider builds trust relationships with a requestor based on the analysis of service-based interactions[1]. In this case, the services can be considered as contexts [11]. A trust-aware provider analyzes service-based interactions from trust perspectives and makes decisions based on the trustworthiness of requestors [8, 9]. Therefore, a trust-aware provider could be developed by addressing its corresponding trust concerns and incorporating suitable trust equations in the software to calculate the trustworthiness of requestors. Moreover, to facilitate the adaptive nature of trust, the software could monitor service-based interactions with requestors from trust perspectives and make trust-based decisions at run-time.

The traditional approach to service-based software development often concentrates on the functional requirements of provided services by assuming that the potential requestors using the service will behave in a trustworthy manner [16]. However, today's increasingly networked systems prove this concept futile. Therefore, given the various relationships with potential requestors, a provider software needs to be built by treating trust as a non-functional requirement (NFR) for clarifying these relationships and reasoning about system trustworthiness [17, 18]. The specification of trust relationships provides a set of trust scenarios of the system. A trust scenario binds interested parties together based on a specific context. For example, a trust

[1]For brevity, we use provider to denote service provider software and requestor to denote service requestor. We further denote software as a system, a service provider as a server, and a service requestor as a client or a service user.

relationship is formed between a provider and requestor based on a provided service, which the requestor may exploit to harm the provider or other users in the system. In the case of exploitation of the service, the provider software can consider the relationship with the requestor less trustworthy. This particular scenario is encompassed by a trust scenario which thus describes a trust relationship based on the possible exploitation of a service. Informally, "I trust you because of your good interaction records with me, or I may trust you despite your bad interaction records with me, but good interaction records with others".

Compared to the specification of functional requirements (FR), there are very few tools and methodologies to support trust requirements. Moreover, most of the trust specification methodologies only focus on authentication-based certification and overlook the impact of trust scenarios from a system perspective [19–24]. The Unified Modeling Language (UML) [25] is used for "specifying, visualizing, constructing, and documenting artifacts of software systems, as well as for business modeling and other non-software systems" [25]. The advantage of UML is that it can be extended to any particular domain. Although UML provides diagrams for specifying the FR, it does not provide that much support to specify trust requirements.

A provider software analyzes service-based interactions based on trust scenarios. However, it needs a trust model to calculate the trustworthiness of requestors according to the analysis. A trust model in a provider encompasses some trust equations to quantify the trustworthiness of requestors. A provider can use the calculated trustworthiness in making decisions for service-based interactions which thus protect the software from potentially untrustworthy requestors [27]. However, the type of protection we are concerned with cannot be addressed completely using cryptography and secret key sharing techniques that are used to encrypt messages between communicating entities [28]. We need trust model incorporated in each service providing software [30]. A number of such trust models can be adopted for service-based software [9, 11, 15, 31–34, 36–50]. However, these models suffer from one or more of the following shortcomings. First, most of them are not risk-aware and do not judge interactions from risk perspectives. Second, some of them do not address the dynamic aspect of trust that trust decreases over time without any further interaction. Third, most of them do not propose any mechanism to detect unreliable recommendations. Fourth, almost all of them treat all recommendations equally, when recommendations from closer and known entities would ideally be given more preference. Fifth, most of them consider general trust, overlooking the multi-faceted nature of trust.

Although trust models have been extensively researched for service-based software, a monitoring approach is not presently available to automatically analyze service-based interactions from trust perspectives and make trust-based decisions at run-time. However, a trust monitoring architecture should be incorporated in each trust-aware service provider entity to automate the trust-aware execution of the system.

1.2 Objective and Scope

The objective of this thesis is to provide a framework to develop and monitor trust-aware service-based software. The framework uses trust scenario specifications for the development of trust-aware provider software and incorporates trust equations into the software. We use trust as an NFR to develop trust-aware service-based software. We consider that potential requestors using the services may become untrustworthy, and therefore, interactions with them need to be monitored to analyze the interactions based on the trust scenarios and the trust equations. The major objectives of this thesis are threefold:

1. A trust-aware service provider should be developed by considering the corresponding trust concerns. We focus on the specification of trust scenarios. Since standard UML is not suitable for specifying trust scenarios, we extend UML.

2. A trust model for service-based software should facilitate the calculation of the trustworthiness of requestor based on service-based interaction analysis. We develop a trust model by addressing the shortcomings of existing trust models.

3. A trust monitoring approach for service-based software should utilize both the trust scenarios and the trust model to monitor the trustworthiness of requestors and to make trust-based decisions at run-time. We develop a trust monitoring architecture, which is assumed to reside in each service provider entity.

1.3 Overview

In this thesis, we focus on the development of trust-aware service-based software by considering that the service providers operate in an imperfect environment where requestors can be malicious and unreliable. The development is facilitated by a trust-aware service-based software development framework which specifies trust scenarios, generates trust rules, and incorporates trust equations into each provider. Moreover, to support the adaptability of trust, we propose a trust monitoring approach to analyze service-based interactions, calculate trustworthiness of requestors and make trust-based decisions at run-time.

To develop trust-aware service-based software, we propose a trust-aware software development framework. The framework allows developers to specify trust scenarios for provider software using the stages of a software development life cycle (SDLC). To specify trust scenarios, we propose a UML profile called UMLtrust (UML for trust scenarios) [1] that specializes UML notations to the domain of trust. The framework shows the gradual development of trust rules from the specified trust scenarios. These trust rules are used for monitoring purposes. The framework facilitates the incorporation of trust equations into the software to quantity and measure trust.

To incorporate trust calculation in total software development, we present a trust model called CAT (Context-Aware Trust) [2] which is assumed to reside in each service provider. The trust model utilizes trust rules and calculates the trustworthiness of requestors. Moreover, CAT exemplifies some well-known interaction-based trust properties including risk-awareness and context-awareness. The developed service-based software is thus able to analyze interactions from trust perspectives and calculate the trustworthiness of requestors. A trust-based service granting algorithm is provided to decide whether to grant services to requestors based on their trustworthiness.

To facilitate the monitoring of the trustworthiness in service-based interactions, we propose a trust monitoring approach by presenting a trust monitoring architecture which is assumed to reside in each provider. The approach uses the trust rules to monitor service-based interactions and employs the trust model to measure the trustworthiness of requestors. We propose a trust monitoring algorithm to analyze and calculate the trustworthiness of requestors and make decisions about service-based interactions at run-time. We implement and successfully evaluate the framework.

1.4 Contributions

The major contributions of this thesis are summarized as follows.

- A trust-aware service-based software development framework is proposed. The framework uses our proposed UML profile called UMLtrust (UML for trust scenarios) to specify trust scenarios [1].

- A trust model called CAT (Context-Aware Trust) is provided for service-based software. The outcomes of service-based interactions based on the trust scenarios are employed by CAT in some calculation equations to measure the trustworthiness of requestors [2].

- A trust monitoring approach based on the trust scenarios and the trust model is presented to analyze service-based interactions and facilitate trust-based decision making at run-time

1.5 Thesis Organization

The remainder of the thesis is organized as follows. Chapter 2 includes the background information and work related to this thesis. The background information provides an overview of service-based software, UML, trust in computing systems, and some well-known interaction-based trust properties which CAT follows. The related work provides an in-depth survey on trust in system development. Moreover, the related work covers UML extensions for trust specifications, different trust models, trust-based interaction monitoring, and service monitoring.

Chapter 3 presents the trust-aware software development framework, providing a detailed description of UMLtrust. The applicability of UMLtrust is illustrated using a trust scenario from file sharing applications.

Chapter 4 covers the underlying trust model (CAT) used in the framework. Moreover, a trust-based service granting algorithm is presented which employs CAT to decide whether a requestor is trustworthy enough to grant service. The model and the algorithm are illustrated using examples from a file sharing grid.

Chapter 5 provides the trust monitoring approach for service-based software by presenting a trust monitoring architecture and a trust monitoring algorithm to analyze service-based interactions and make decisions at run-time. The monitoring approach is elaborated by using examples from a file sharing grid.

Chapter 6 discusses the implementation details of our prototype system developed to evaluate our approach.

Chapter 7 presents the details on the case study we have performed to evaluate UMLtrust, CAT and the trust monitoring approach described in the earlier chapters.

Chapter 8 concludes the work by summarizing the thesis, identifying limitations, and providing a number of future research directions.

Chapter 2

Background and Related Work

This chapter provides background information and discusses work related to this thesis. Section 2.1 covers the background information by providing a brief overview of service-based software, Unified Modeling Language (UML), trust in computing system, and some well-known properties of interaction-based trust. The related work are summarized, compared and contrasted in Section 2.2.

2.1 Background

A brief overview of service-based software is provided in Section 2.1.1. Since we use UML to represent trust scenarios, UML is reviewed in Section 2.1.2. Section 2.1.3 discusses trust in computing systems. Since we focus on interaction-based trust in this thesis, Section 2.1.4 explores some interaction-based trust properties that are deemed necessary when developing an interaction-based trust model for service-based software.

2.1.1 Service-Based Software

The term ‘Service-Based Software’ was introduced in the early years of this decade to emphasize that next generation systems will be composed of services from a number of service providers [5]. In practice, services are the system functionalities offered to the end users (*i.e.,* requestors) following system requirements. An architecture of service-based software has three major parts: the service providers, the service requestors or clients, and the registries [4]. The service providers publish their services on registries, from which requestors identify and invoke them. Providers describe their services and the service requirements, and deploy their services in the system architecture where requestors invoke them. Providers interact with each other when necessary. This kind of infrastructure thus leads to a system of highly interacting entities.

Basically, in service-oriented computing (SOC) [3], service-based interactions are described and analyzed, emphasizing a service-oriented architecture (SOA) [4], where the providers and the requestors interact with each other through the services. Both the providers and the requestors operate independently in such open systems. This raises the issue of trust between providers and requestors. Requestors need to choose the most competent and trustworthy provider, but since the providers are engaged in dynamic service provision in such a highly interactive system, they also need to maintain secure and trustworthy interactions with the requestors [6, 7]. Since both the providers and the requestors are independent parties, this leads to a decentralized architecture. Because of this decentralization, service-based systems do not have any centralized trust computation mechanisms. The interacting parties, therefore, need to determine the trustworthiness of each other by themselves [4].

2.1.2 Unified Modeling Language (UML)

UML is used to model the abstract view of software in all stages of a software development life cycle (SDLC). The graphical notation of UML helps in better understanding the software and eases the communication between developers and customers. Moreover, UML diagrams can be mapped directly to object-oriented programming languages which helps in code generation.

UML 2.0 proposes 13 diagrams which are categorized as either structural or behavioral. Structural diagrams represent a static view of a software, where behavioral diagrams show the behavior of the software by representing communication between different software modules. Class, object, component, and package diagrams are all structural diagrams. Behavioral diagrams are of two types. General behavioral diagrams to show the general behavior of the software and interaction diagrams to show communication between different parts of the system. General behavioral diagrams include use-case, activity, and state-machine diagrams, while interaction diagrams include sequence, interaction overview, timing, and communication diagrams. In UML-trust, we use class and package diagrams to provide a structural view of service-based software based on different trust scenarios and use-case and state machine diagrams to show the behavior of the system by representing the trust scenarios. Moreover, we use basic interaction diagrams to show the interaction and communication between service providers and requestors in a service-based system.

Since UML is mainly used to represent software functional attributes, an extension of UML is necessary to represent the non-functional requirements (NFR) of the system. UML can be extended by generating domain-specific stereotypes and tagged values, where a stereotype is a specialized UML element to add information or constraints to a UML element and a tagged value adds specific notes to represent the UML element in that particular domain. The extension is performed either in the meta-model or the meta-meta-model of UML. The first approach provides tagged values and stereotypes on existing UML meta-models to define domain specific elements, while the second approach creates, deletes or modifies current UML syntax and semantics to instantiate new UML elements. An example of the first approach is UMLintr [81], while Rational Rose Real Time [76] adopts the second approach. We follow the first approach for UMLtrust.

2.1.3 Trust in Computing Systems

The concept of trust comes from the human notion of belief and disbelief [82]. Since human interpretation of trust may take many forms, the adaptation of trust to computing systems has remained tricky and elusive to date [19, 94]. In information systems, trust is often related to security [28], where an entity is (dis)trusted if it is (in)secure. Since the interacting entities in a given information system depend on each other for goals to be achieved, tasks to be performed, and resources to be utilized, the dependance naturally leads to some situations where entities could form trust relationships with each other [17]. This notion of situation or 'context' complies to the situational trust of McKnight and Chervany [92]. For example, software providing specific services builds trust relationships with potential requestors based on service usage. The services denote the specific situations or contexts based on which the entities interact with each other [7]. Grandison [93] recognizes the important aspects or requirements to build such trust-based service systems. The aspects are trust technology, business drivers, legislative framework, and social framework. Trust technologies include the system development artifacts of a service-based system, encompassing trust-based specifications, trust models having trust calculations, and trust monitoring. Business drivers focus on proper insurance and investment models focusing on the needs of users. Legislative framework implies the applicability of laws in preserving trust, while social framework builds on social interaction with requestors. In this thesis, we focus on trust technology by encompassing trust scenario specifications, a trust model, and a trust monitoring approach in the technological structure of service-based software. By incorporating trust technologies into the software, we make it trust-aware.

The specification of trust relationships in a system environment encompasses the quality attributes that need to be analyzed, including the non-functional requirements (NFR) of the system such as security and reliability [17]. A trust relationship specified in a formalism provides a trust scenario. A trust scenario thus combines interested parties together based on a context and analyzes the trust requirements that are necessary to build a trust relationship [18]. In service-based software, trust requirements can include the vulnerabilities and exploitations that can arise due to the usage of the services. Therefore, a trust scenario in service-based software includes the provider and requestor based on the provided service and the possible trust concerns.

Besides identifying and specifying the trust scenarios of a system, the trust relationships need to be formalized to quantify and calculate trust [94]. The calculation of trust is necessary to determine the extent of trust one party has for another in the computing system. Based on the calculation of trust, interested parties are able to analyze each other in specific contexts which leads to dynamic decision making [28].

The calculation of trust is facilitated by the development of trust models which analyze the trustworthiness of the potential stakeholders and assign quantified trust values to the stakeholders. The analysis is based on the specified trust scenarios. Moreover, the entities can recommend each other based on the analyzed trust values: a recommendation is a trust value an entity $E3$ provides to an entity $E1$ specifying how much it trusts a third entity $E2$ for a specific context. A trust model thus includes proper calculation schemes to analyze, quantify the trustworthiness of the potential stakeholders and make recommendations based on the analysis.

The monitoring of trust includes the trust scenarios and the trust model in the system environment. This inclusion helps in analyzing the trustworthiness of the entities at run-time for dynamic decision making.

2.1.4 Properties of Interaction-Based Trust

Since the analysis of trust in service-based software requires the analysis of service-based interactions, we focus on developing an interaction-based trust model. We develop the trust model following some well-known interaction-based trust properties specified in [11,34,37,38,44,45,82] and listed below; "P" standing for property.

P1. Context-awareness (CA): Trust is context or service dependent. Trust of entity $E1$ on entity $E2$ for context c_i at time t ($T(E1, E2, c_i, t)$) does not imply the same trust for another context c_j.

$$T(E1, E2, c_i, t) \nRightarrow T(E1, E2, c_j, t), \ where\ i \neq j$$

P2. Rule-oriented (RO): Trust is condition or rule-dependent, and so an interaction I can have a number of trust rules (r) to analyze its outcomes. The total outcome O of I is a function of all the outcomes as measured by the corresponding trust-rules $(r_1, r_2, \ldots, r_n)$.

$$O(I) = f(O(r_1), O(r_2), \ldots, O(r_i), \ldots, O(r_n))$$

P3. Risk-awareness (RA): For each interaction, the trust rules capture the possible risks associated with it. The higher the risk of an outcome, the higher is the trust level assigned to the corresponding trust rule.

P4. Recommendation-based (RB): An entity may need to take recommendations from other entities to (i) know about an unknown trustee, (ii) confirm belief on the trustee, and (iii) make decisions based on not only self observation, but also from others' observations. The total trust of $E1$ on $E2$ can be a function of the direct trust $(T_D(E1, E2, c_i, t))$ and the total recommendation trust of $E1$ on $E2$ $(R(E1, E2, c_i, t))$, where the total recommendation trust is calculated from the recommendations of other entities to $E1$ on $E2$. However, an entity can discard recommendations in calculating total trust for particular contexts which makes the following condition not applicable to all situations[1].

$$\exists c_i (T(E1, E2, c_i, t) = f(T_D(E1, E2, c_i, t), R(E1, E2, c_i, t)))$$

P5. Recommendation-filtration (RF): An entity can impose a certain accuracy threshold to differentiate between reliable and unreliable recommendations. The

[1] $\exists$ denotes that the corresponding condition can be applicable to some situations (*i.e.,* partial applicability), while $\forall$ denotes the universal applicability of a condition.

accuracy of a recommendation can be determined by comparing it against the corresponding direct trust, obtained by the analysis of direct interactions with the corresponding trustee (*i.e.,* recommended entity) [35]. A recommendation from a recommender should be discarded, if the previous recommendations from the recommenders for this context are below the accuracy threshold.

P6. Semi-transitive (ST): *"Trust cannot be trivially propagated"* [37] which means that the trust of an entity can only be known to others if the entity wants to propagate it to others. If $E1$ trusts $E2$ and $E2$ trusts $E3$, that cannot be concluded as $E1$ will trust $E3$. However, if $E2$ recommends $E3$ to $E1$, then there may be a partial or no transitive trust relationship between $E1$ and $E3$.

P7. Non-symmetric (NS): For any context c_i, trust of $E1$ on $E2$ at time t does not imply the same trust of $E2$ of $E1$ for c_i.

$$\forall c_i(T(E1, E2, c_i, t) \nRightarrow T(E2, E1, c_i, t))$$

P8. Dynamic (D): The trust value of $E1$ on $E2$ changes over time due to new interactions and new recommendations.

P9. Time-based Ageing (TA): Without any further interaction, the trust value of $E1$ on $E2$ decreases over time.

$$\forall c_i(T(E1, E2, c_i, t) > T(E1, E2, c_i, t + \Delta t))$$

P10. Path-based Ageing (PA): Recommendations from nearer and known entities are given more importance than recommendations from entities which are unknown or further away.

2.2 Related Work

In this section, we discuss some of the work related to this thesis. The discussion is divided into five subsections. The use of trust concerns in system development is elaborated in Section 2.2.1. Section 2.2.2 explores some UML extensions related to our proposed UML profile, UMLtrust. Section 2.2.3 compares and contrasts our developed trust model, CAT, with different trust models. Section 2.2.4 provides some interaction monitoring approaches based on trust perspectives from the literature, while Section 2.2.5 discusses the state-of-the-art of service monitoring in service-based software.

2.2.1 Trust in System Development

Trust has been incorporated into system design from three directions: specification of trust policy to provide access to system resources [19–23], generation of trust certificates to authorize someone for a specific purpose [24], and inclusion of trust as a non-functional requirement (NFR) in the system design phase by developing trust scenarios [17,18]. A trust policy specifies constraints for a particular access provision, where the access might provide the trustee the possibility of invoking certain services or utilizing certain resources. A trust certificate claims that the certified entity is trusted for the stated purpose. As opposed to the policies and certificates, a trust scenario looks at the quality attributes of entities and provides a way to monitor those attributes in terms of specifications. A trust scenario is unique among other non-functional scenarios (*e.g.*, intrusion scenarios), because it encloses a comprehensive domain of quality attributes such as security, reliability, competence and honesty.

Using trust as a non-functional requirement for system development helps in identifying as well as summarizing the quality requirements of a system, which thus helps in building the system in a trust-aware manner [17].

A number of trust-policy languages have been proposed including SULTAN (Simple Universal Logic-oriented Trust Analysis Notation) [19], PolicyMaker, KeyNote [20, 21], PICS (Platform for Internet Control Selection) [22], REFEREE (Rule-controlled Environment for Evaluation of Rules and Everything Else) [23], and TPL (Trust Policy Language) [24]. SULTAN focuses on trust management for internet applications by building trust relationships between entities. PolicyMaker is the pioneer of trust-based policy specification, focusing on an entity's public key and binding it to credentials which allow the entity a specific system service. KeyNote is a successor of PolicyMaker with an enhancement in the verification of such policies. PICS is used to label images in online web sites for the purpose of preventing children from watching pornography, and REFEREE uses PICS to specify access control policies. TPL extends role-based access control (RBAC) to make decisions based on the evaluation of existing certificates about an entity which may come from trusted third parties.

Although all these approaches mention access control policies for restricting access to certain system environments, they actually do not guide the development of systems from trust perspectives. For example, they do not consider whether a system meets user requirements or whether a system uses trust in its own development phase. Yu and Liu [17] address these issues at the requirement level of system development by using trust as a non-functional requirement, where trust is a combination of all or some quality attributes of a system. They illustrate their approach by describing the behavior of a system in the presence of attack, and examine required defenses from trust perspectives. Horkoff *et al.* [18] extend their work in the trusted computing base, by claiming that trust can be included in the technological framework of a system at both the software and the hardware level. We follow the approach of [17, 18] by capturing security as a system trust requirement; we consider an entity as secure if it is safe under malicious activities or attacks.

Table 2.1 compares UMLtrust with the related work, where the first column lists the related work, the second column mentions the different software development phases used in the corresponding work, the third column identifies the specific domain where the work is applicable, the fourth column presents the trust language used by the work, and the last column points out the trust scenarios encompassed by the work. In the second column (*i.e.,* Development phase), 'requirement' denotes identification and elicitation of software requirements, 'specification' denotes the modeling of system artifacts based on the elicited requirements, and 'evaluation' denotes the testing of the system in the deployment phase based on the elicited requirements and specifications.

2.2.2 UML for Specifying Non-Functional Requirements

The use of UML for the purpose of trust scenario specification has not been addressed in the literature. However, Górski *et al.* [72] use UML stereotypes to represent trust cases. A trust case influences a trustor's level of trust by providing evidence in the form of claims, facts and assumptions. The Claim Definition Language (CDL) is used to specify claims. While they use UML stereotypes and CDL to specify trust case, we use stereotypes and tagged values in use cases and class diagrams to specify trust scenarios as part of system development requirement. Moreover, compared to their usage of claims and facts to influence the trust level of an entity, we use trust

Table 2.1: Trust in system development

Work	**Development phase**	**domain**	**Language**	**Trust-concern**
[19]	requirement, specification	internet systems	SULTAN	trust policy
[20]	specification	application	PolicyMaker	trust policy, public key
[21]	specification	application	KeyNote	trust policy, public key
[22]	specification	web documents	PICS	trust label, trust policy
[23]	specification	web documents	REFEREE	trust label, trust policy
[24]	evaluation	RBAC	TPL	trust certification
[17]	requirement	information system	$i*$	trust scenario
[18]	requirement	trusted computing	$i*$	trust-scenario
Our Work	requirement, specification, evaluation	information system	UMLtrust	trust scenario

scenarios to monitor interactions and gather evidences. Therefore, while a trust case is a set of claims and facts to reason about a trust relationship, a trust scenario is a description of a situation necessary to construct a trust relationship between the potential stakeholders of the system.

Trust and risk are very much related, as risk identifies the probability of the unwanted happening. Vraalsen *et al.* [74] use the CORAS language for security threat modeling. CORAS is an extended UML 2.0 profile, which uses use case diagrams to model threats and risks in the form of unwanted or malicious behavior. While CORAS works in the system requirement phase, UMLtrust focuses on the system design and specification phase.

UMLQoS, a UML profile for Quality of Service (QoS) [73], specifies fault tolerance and criticality attributes, including constraints to specify policies, different performance attributes such as efficiency, throughput, and scalability. While UMLQoS covers a comprehensive extension of UML to model software quality attributes, it does not support trust scenario specifications. While UMLtrust denotes an entity expectation, and belief to construct a trust relationship based on a situation, UMLQoS describes the quality attributes that may be included in a trust scenario.

UMLintr [81] is a UML profile for specifying intrusion scenarios by using stereotypes and tagged values. While UMLintr specifies intrusion scenarios, UMLtrust specifies trust scenarios. A trust scenario may consider an intrusion scenario as part of its scenario. For example, an intrusion attempt from an entity may be considered accidental if it happens just once. However, if the attempt happens multiple times, the entity is surely an attacker and should not be trusted.

UMLsec [75] uses stereotypes and tagged values to define the security requirements of a system, and employs Object Constraint Language (OCL) [26] to validate tagged values for any security violation. UMLtrust models trust scenarios to impose certain conditions on the system functional requirements. For example, a user can assume that a security requirement will work properly based on his conditions. These conditions are expressed in UMLtrust, whereas the security requirements are expressed in UMLsec.

secureUML [14] models role-based access control (RBAC) by viewing a system as a set of users, roles, and resources. It is a generic security policy language from a security point of view. Although they impose certain conditions on specific roles using RBAC, they do not consider the scenarios, when the conditions are violated. For example, an entity assigned to do a simple user role can attempt to gain the privilege of an administrator. Since UMLtrust attempts to deal with these kinds of scenario, we consider that it complements secureUML by adding certain trust requirements to the system functionalities. In this way, trust can be preserved among the entities of the system.

Table 2.2 provides a brief comparison of UMLtrust with the above mentioned UML extensions, where the first column lists the work, the second column gives the name of the work, and the third column mentions the different non-functional requirements (NFR) encompassed by the work.

2.2.3 Trust Models

Trust models have been introduced to (i) analyze the trustworthiness of potential stakeholders and (ii) make trust-based decision. The development of trust models has been addressed from two angles: centralized and decentralized. While centralized

Table 2.2: UML extensions for specifying NFR

Work	**Name**	**Specified non-functional requirement(s)**
[72]	Trust case	evidence, claim
[73]	UMLQoS	fault-tolerance, criticality attributes
[74]	CORAS	security-threat modeling, risk-modeling
[81]	UMLintr	intrusion detection
[75]	UMLsec	security policy validation, encryption
[14]	secureUML	RBAC
Our Work	UMLtrust	trust scenario

trust models focus on building a central trust and reputation server, decentralized trust models aim to make each entity trust-aware. We discuss some related centralized and decentralized trust models in the next subsections.

Centralized Trust Models

In the eBay trust and reputation model [82], users can provide feedback ratings after each interaction. The model is not context-aware, so big and small transactions are all treated equally. SPORAS [29] proposes a solution to this problem by calculating and assigning trust based on rules. However, it discourages newcomers by providing very low initial values. In the BambooTrust [42] reputation system, users can query about another user. The query requests are handled in parallel machines to provide fast reply. Beta model [32] uses statistical representation of trust and reputation for e-commerce systems based on beta probability density functions. The advantages of using a central trust model are twofold: First, trust information can be retrieved from one single trusted server, and so entities in the system can avoid processing trust-related analysis by themselves. Second, the information of an entity being compromised can be propagated to other entities easily by the central trust server. The disadvantages of having a central trust model are threefold. First, the central trust server can put all the entities in the system in danger once compromised. Second, it is hard to implement a central trust server in a decentralized service-based system, since service-based software operate in an open environment, where any entity can join and leave anytime. Third, the central trust server may delay beyond an acceptable limit in providing replies to queries for a highly interacting service-based system.

Decentralized Trust Models

These models focus on two features. First, choosing the most trustworthy provider (*i.e.*, requestors assess the trustworthiness of the provider), and second, deciding whether a requestor is trustworthy (*i.e.*, providers assess the trustworthiness of the requestor).

FIRE [31] selects service providers based on previous interaction records and recommendations from other entities. Although FIRE uses interactions to gather trust, it does not rely on rules specifically to monitor it. Moreover, FIRE is not risk-aware and does not provide any ageing parameter.

Billhardt *et al.* [39] propose a trust model for selecting a provider in service-oriented computing. Similar to them, we propose a context-similarity parameter, however our parameter uses word-based similarity calculation while they use organization-level service hierarchy. Toivonen *et al.* [50] propose an ontology-based context similarity method. They use network ontology to calculate the similarity between two nodes, whereas we use keywords which are used to describe and distinguish the services (*i.e.*, contexts). Therefore, we do not require any hierarchical ontology-based node structure.

TRAVOS [36] is a trust and reputation mechanism based on direct interactions and recommendations adding the ability to identify malicious recommendations. Compared to them, while we propose the measurement of recommendation accuracy, we cannot conclude the malicious nature of recommenders from the analysis. However, they neither apply rules to analyze interactions, nor use time-based and path-based parameters. A broker based decentralized reputation system is proposed in [43], which penalizes entities for giving false recommendations; unlike this work, we do not penalize recommenders for false recommendations.

Certain trust [40] separates humans from software entities and provides mapping between them for dynamic decision making. Humans are capable of stating their certainty about a trust opinion, which its corresponding software representation uses for decision making and recommendation purposes. Fuzzy notations are used by Sherchan *et al.* [41] to reason about trust while choosing a provider. Bayesian trust models are used by [44] and B-trust [33] to denote the context-awareness nature of trust. Combining ideas from each, we use context-aware, rule-based trust reasoning in CAT.

In REGRET [34], users can rate each other after each interaction. It proposes a time-based ageing parameter. However, the parameter takes only the ratio of current and previous times and does not consider the system environment. For example, entities in an almost static system will interact less, which results in a high ratio between two interaction times.

An automatic trust prediction model is used in [48] to select service providers, where trustworthiness is measured from the advertised entity attributes and monitored entity attributes. Trust and recommendations are formally defined and analyzed in [15, 37, 38] by incorporating belief, disbelief and uncertainty in each interaction.

These are mostly theoretic trust models, the implementation aspects of which for practical systems need to be examined.

Trust-based decision making for a service request on the server side is mostly used to provide access to system resources. Dimmock *et al.* [46] propose trust based access control (TBAC) for open systems by defining policies. The server uses trust and recommendation values in its authorization policies to infer the trustworthiness of the clients before providing them services. They provide a grammar of the policy specification and employ risk assessment in the decision making, which helps in estimating the probable outcome of an action. Similarly, Chakraborty and Ray [54] propose TrustBAC, a trust-based access control model. This model provides the detailed structure of permission messages which include user name, trust values and assigned role. This model also provides a trust calculation scheme. While both of these work look at the access control mechanism from authorization perspectives, we focus on service granting based on the monitoring of service-based interactions. We propose a trust monitoring architecture and a trust model to support our work, whereas neither of the above propose any monitoring architecture. TRBAC [47] proposes access control policies to server resources based on trust reasoning. While both of the TRBAC [47] and TrustBAC [54] are risk-aware, unlike our system, they do not use time-based ageing, service similarity, and recommendation filtration.

Lin and Varadharajan [45] integrate trust with risk management to grant authorization, considering the utility of an outcome in the decision to grant access. Haque and Ahamed [11] propose OFTM, a formal trust model for pervasive computing to handle service requests in mutual collaboration. We adapt path-based ageing from [11], proposing a different calculation scheme for the path-based ageing parameter. However, [11] does not use time-based ageing, context-similarity, or recommendation filtration.

Finally, PeerTrust [49] is a context and risk-aware trust model that operates on the online rating mechanisms for e-commerce transactions, such as on eBay; unfortunately it does not offer any of the other desirable properties of interaction-based trust.

Table 2.3 compares and contrasts the above discussed work, where the first column lists the work and the rest of the columns compare and contrast the work based on the interaction based trust properties introduced in Section 2.1.4. Our trust model, CAT, satisfies all seven of these desirable properties from Section 2.1.4, while no other model satisfies more than four except TBAC, which covers the five most common (RO, CA, RA, RB and RF). Besides our model, only OFTM satisfies path-based ageing, and our model is unique in its implementation of time-based ageing.

CAT follows the rule-orientedness property (*i.e.,* RO) by following some trust rules for service-based interaction analysis; the context-awareness (*i.e.,* CA) property by calculating the trustworthiness of requestors based on provided services; the risk-awareness property (*i.e.,* RA) by encompassing trust concerns on software exploitation and misuses in the corresponding trust rules; the recommendation-based (*i.e.,* RB) property by using recommendations in the trust calculation; the time-based ageing (*i.e.,* TA) property by decreasing trust over time without further interactions; the path-based ageing (*i.e.,* PA) property by providing more emphasis on recommendations from nearer entities; and the recommendation-filtration (*i.e.,* RF) property by differentiating between reliable and unreliable recommendations. Moreover, CAT is

Table 2.3: Property-based placing of different trust models

Model	RO	CA	RA	RB	TA	PA	C/D	RF
FIRE [31]	Y	Y	Y	Y	N	N	D	N
FTMAS [37]	N	N	N	Y	N	N	D	N
TDAS [38]	N	Y	N	Y	N	N	D	Y
TSOC [39]	N	Y	N	Y	N	N	D	N
CertainTrust [40]	N	Y	Y	Y	N	N	D	N
Fuzzy [41]	N	Y	Y	Y	N	N	D	Y
BambooTrust [42]	N	Y	Y	N	N	N	C	N
SPORAS [29]	N	N	N	Y	N	N	C	N
REGRET [34]	N	Y	N	Y	N	N	D	N
ICRM [43]	N	N	N	Y	N	N	D	Y
TRAVOS [36]	N	N	N	Y	N	N	D	Y
TRP2P [44]	N	Y	N	Y	N	N	D	Y
TBRM [45]	N	Y	Y	Y	N	N	D	N
TRBAC [47]	Y	Y	Y	Y	N	N	D	N
TBAC [46]	Y	Y	Y	Y	N	N	D	Y
TrustBAC [54]	Y	Y	Y	Y	N	N	D	N
ATP [48]	N	N	N	N	N	N	D	N
OFTM [11]	N	Y	Y	Y	N	Y	D	N
EMDR [15]	N	N	N	Y	N	N	D	Y
PeerTrust [49]	N	Y	N	Y	N	N	D	Y
Beta [32]	N	Y	N	Y	N	N	C	N
B-trust [33]	N	Y	Y	Y	N	N	D	N
our work, CAT	Y	Y	Y	Y	Y	Y	D	Y

The trust properties follow the definitions provided in Section 2.1.4.
RO = Rule-Oriented, CA = Context-Aware
RA = Risk-Aware, RB = Recommendation-Based
TA = Time-Based Ageing, PA = Path-Based Ageing
C/D = Centralized/Decentralized, RF = Recommendation-Filtration

a decentralized trust model (*i.e.,* C/D), as it resides in each service provider. Therefore, from the standpoint of broad practical implementation of interaction-based trust properties, our model is a reasonably better choice for service-based software.

The other three properties of interaction-based trust, *i.e.,* semi-transitiveness, non-symmetricness, and dynamism (see properties P6, P7 and P8 respectively in Section 2.1.4) are common-sense theoretical properties of a trust relation that are likely provided by most or all of the above models.

2.2.4 Interaction Monitoring

In our work, we consider automatic monitoring of service-based interactions from trust perspectives. The monitoring of the interactions facilitates analyzing trustworthiness of requestors and making run-time decisions. However, this type of monitoring we are concerned with has not been addressed so far. Nevertheless, our trust monitoring approach is motivated by some work related to the monitoring of interactions.

English *et al.* [27] propose a monitoring architecture for a self-protective ubiquitous system. The system analyzes interactions, determines the trustworthiness of potential stakeholders, and is capable of making decisions based on the analyzed trustworthiness and the probable risks associated with the corresponding outcomes. Although their approach is close to our approach, we differ from them in a number of ways. First, they do not use trust rules for monitoring purposes but rather assume that some types of events are untrustworthy and if the events take place, the interactions are untrustworthy. Second, they do not propose a trust model compatible to their system, although the trust model applicable to their monitor needs to clearly denote the relationship between risks and outcomes. Third, they do not propose any clear guidance on how the decisions can be taken using the monitor, and they do not consider recommendations for the decision making and trust analysis. We address the above mentioned issues in our trust monitoring approach using the trust rules generated from the specified trust scenarios and the developed trust model in our trust monitoring architecture. Moreover, we propose a trust-based service granting algorithm and a trust monitoring algorithm to make decisions at run-time.

Etalle and Winsborough [51] propose a formal monitoring approach to detect violation of integrity constraints in trust-based authorization systems. They consider that some of the participants of a policy may become untrustworthy, and so their role in providing some credentials or certificates should be considered carefully. While they focus on policy management for authorization perspectives, we focus on rule-based interaction analysis to grant services.

Sandhu and Zhang [53] incorporate an access control mechanism in the Trusted Computing (TC) module. The enforcement of the policies are handled by a trust reference monitor which has direct access to the hardware to store secret key and other sensitive information in the trusted computing platform. The monitor enforces the execution of policies, which may provide one entity (application) access to the storage of another entity (application). While they impose access control at the hardware level, we work at the software level to impose service granting to requestors based on trust-based interaction monitoring.

Ryutov *et al.* [8] propose an adaptive trust negotiation and access control framework for open grid systems, where the actions of entities are monitored. Based on the monitoring the entities are assigned a suspicious level obtained by using the difference between the actual and estimated outcome from an action. This is related to a direct trust calculation. Moreover, entities can issue a warning if some serious security related suspicion is found during monitoring. This is related to recommendation trust calculation. The provisioning of services and the access to resources are based on policies, which use the suspicious level for dynamic decision making. Unfortunately, this framework focuses on policy management and overlooks the monitoring of actions.

Rajbhandari *et al.* [9] propose a fact-based monitoring framework for service-based software. The facts are written in Extensible Markup Language (XML) [88] as rules for monitoring purposes. The facts are not clearly specified by them, but rather are based on the interaction results between participating entities. While they use facts, we use trust rules. Moreover, we specify the trust rules based on our proposed UML profile. Nevertheless, they use the monitor to analyze the trustworthiness based on a generalized trust calculation, while trust in our system is context-aware.

2.2.5 Service Monitoring in Service-Based Software

In this section, we discuss work related to service monitoring in service-based software. The purpose of this discussion is to show the differences between our trust-based service monitoring approach and the existing work.

Jurca *et al.* [62] propose monitoring of quality of service (QoS) in service-level agreements (SLAs) based on user feedback. The SLAs are the service requirements that the service provider is bound to provide to requestors. Requestors are provided payment based on their feedback, and good and effective feedback is rewarded. However, there is no punishment for providing wrong and malicious feedback. Based on this feedback, the services are rated. Therefore, this is a fully manual monitoring of the system with little guarantee of honest feedback. We propose automatic monitoring of trust concerns (*e.g.,* security) of the services, and we monitor the trustworthiness of requestors, while they monitor the trustworthiness of providers.

Skene *et al.* [52] consider that the service constraints between the provider and requestors are monitorable by some trusted third party. The entities use the monitored value to depend on each other, and if the monitored value falls outside the dependence range, the value is discarded. We however, consider that the trust monitor should reside in each service provider entity since a service-based system is inherently decentralized. Unlike them, we penalize requestors, when they act maliciously. Therefore, we focus on the safety of services provided as opposed to the mutual suitability that is the focus of [52].

Zhang *et al.* [55] propose an accountability framework for service-based software. Accountability is defined as the monitoring and controlling of quality and performance issues of each service. They consider a chain of services, where invoking a service may trigger the invocation of related services. Dedicated agents monitor the services and report when some violations are detected. Based on the monitored values, a central accountability authority decides on corrective measures to resolve the problem. Compared to their centralized authority for accountability monitoring, we concentrate on a decentralized trust monitoring approach for service-based interaction monitoring. Moreover, we focus on analyzing the trustworthiness of requestors based on the services provided, while they focus on measuring the quality of the services.

Rochford *et al.* [56] propose and implement a grid-based service monitoring architecture. A remote monitor component is deployed to each service provider. In addition, a central monitor is deployed in the main operational center. The remote monitors are configured for monitoring based on the instructions from a central database residing in the central monitor. Based on submitted information by the remote monitors to the central monitor, decisions are taken manually or automatically. Similar to them, we use distributed monitors to check the safety of providers, but without relying on a centralized control monitor to handle the information; our system is completely decentralized.

Spanoudakis and Mahbub [57] propose a requirements monitoring framework for service-based software. The requirements form the behavioral properties of the services and the systems, composed of web services. A central monitor looks at the specified requirements for possible aberrations in the deployed services. Robinson [16] proposes monitoring the requirements of services. Monitors local to the services raise exceptions when service contracts are violated in the service invocation. The exceptions are sent to the global requirement monitor. Requestors can subscribe to the requirement monitor to know about the exceptions. While both of [57] and [16] concentrate on requirement monitoring, we concentrate on specification-based trust monitoring. Moreover, we do not consider any central monitor.

Letia *et al.* [58] use Z-specification to specify services. The operation of the services are specified from two directions: timeliness in providing output, and type checking in providing input. Monitors are deployed in the system to monitor the specifications run-time. Yan *et al.* [60] transform web service behaviors in discrete event systems (DES) to monitor the network of the web services. The focus is to monitor errors in the deployed services. Compared to [60], we focus on the monitoring of abnormal system states in terms of service execution and from trust perspectives. Unlike them, we use the monitored values to determine the trustworthiness of requestors.

Mao *et al.* [61] propose a grid resource monitoring architecture, where a grid is a collection of providers and requestors. They monitor the status of the resources to allocate them to different grid machines. Our system comes much closer to monitoring the grid machines to make sure that requestors are worthy of access to the resources, nearly the opposite of the focus of [61].

Baresi *et al.* [63] propose two types of monitoring for service-based software. The first type of monitoring considers that the developers insert monitoring code into the services, from which the monitors collect data. In the second type of monitoring, requestors annotate the services before invocation, and the monitor transforms the annotations into some form of assertions which are monitored at run-time. Compared to them, we consider service level contracts in the provided services to monitor and limit their invocations by untrustworthy requestors.

Sahai *et al.* [64] propose an external monitoring framework for internet-based services which are instrumented in providing the necessary monitoring data. An external monitor takes the monitoring data as input and provides the analysis. The focus of this work is to encourage the development of the monitor outside the basic system. However, the effectiveness of their monitor relies on the data received from the instrumented system assuming that the participating entities are all trustworthy. This approach is therefore, not suitable to open service-based systems, where both the providers and requestors can become malicious or unreliable.

Peng *et al.* [65] implement a grid monitoring architecture for a grid market place. Services are provided by different providers and are deployed in service containers. Local sensors are used to collect monitoring data from the service execution. The data from the local sensors are fed to the global monitor. The grid market is composed of a number of service providers and service containers. While they use monitoring information to analyze services, we use the information to determine the trustworthiness of requestors; moreover, we do not use any global monitor.

2.3 Summary

In this chapter, we discuss service-based software and identify why ensuring trust is important for service-based interactions. Since we model software trust scenarios using our UML extension called UMLtrust, we provide a brief overview on standard diagrams and extension techniques of UML. The role of trust and how the enforcement of trust can be achieved in computing systems is elaborated. Since preserving trust in service-based interactions requires maintaining the properties of interaction-based trust, we explore some well-known properties of interaction-based trust which our proposed trust model CAT follows. We then present related work on the use of trust in system development and compare UMLtrust with them. Moreover, some related UML extensions for specifying NFR are discussed, then compared and contrasted with UMLtrust. A number of trust models are elaborated, compared and contrasted based on the interaction-based trust properties with our proposed trust model, CAT. We then present related work on interaction-based and service-based monitoring. We summarize that our monitoring approach is different from other work as our monitor resides in each service provider, automatically analyzes service-based interactions from trust perspectives and makes decisions on requestors at run-time, while most of the work focus on centralized service quality monitoring or trust-based policy management and overlook the automatic trust-based analysis of interactions at run-time.

Chapter 3

A Trust-Aware Service-Based Software Development Framework

In this chapter, we present a trust-aware service-based software development framework. The overview of the framework is presented in Section 3.1. The framework uses an extension of UML called UMLtrust (UML for trust scenarios) for specifying trust scenarios, a detailed description of which is presented in Section 3.2. The applicability of UMLtrust is demonstrated using an example from file sharing applications in Section 3.3.

3.1 The Framework Overview

Our trust-aware service-based software development framework facilitates the development of trust rules and trust models using the stages of a software development life cycle (SDLC). In this way, the framework allows the incorporation of trust rules and trust models into the software to make it trust-aware.

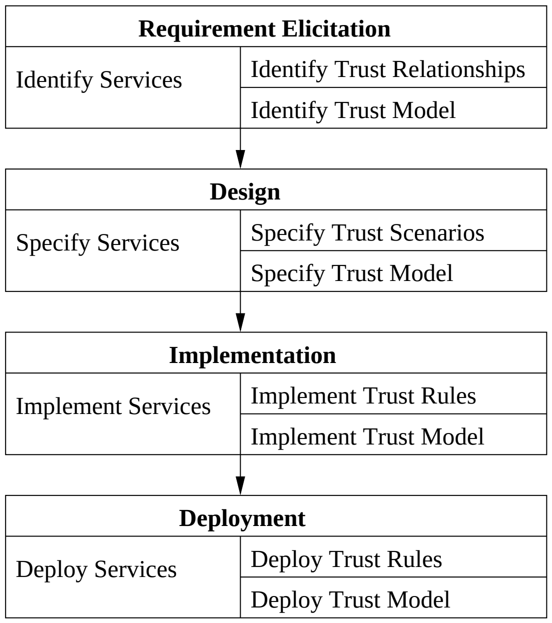


Figure 3.1: The trust-aware software development framework

Figure 3.1 presents the framework, where the left-hand side shows the traditional development stages of service-based software, and the right-hand side shows the gradual development of its trust scenarios and trust models. In the requirement elicitation stage, both software services and trust scenarios are elicited and trust models are identified. In the design phase, both the services and the trust scenarios are modeled using different UML diagrams. In this stage, the trust computation schemes are specified as equations in mathematical notation. In the implementation phase, the system and the trust scenarios are implemented, and the trust model is implemented in the system environment, binding the trust rules and the trust models. Both the services and the above mentioned trust mechanisms are deployed in the software deployment phase. We discuss the development of trust scenarios and trust models in the following paragraphs.

The specification of the trust scenarios (see Section 3.2) allows the generation of trust rules which can be used for monitoring purpose once the system is deployed. The trust model provides trust equations to measure the trustworthiness of requestors and to make trust-based decisions (see Chapter 4). The development of the trust model allows the developers to incorporate trust computation into the total system. The incorporated trust model works in conjunction with the trust rules to monitor service-based interactions (see Chapter 5).

This framework allows software developers to use UML for modeling both the system behavior and the related trust scenarios of the system simultaneously. This approach to system development is useful for several reasons. First, the specification of the trust scenarios forces the consideration of trust early in the design process, which can be used as a driving force to guide potential design requirements [17]. Second, the usage of the same modeling language (*i.e.*, UML) avoids introducing a trust specification language only for trust purposes. Third, the development of trust scenarios can be incorporated in the traditional SDLC so that their development will not create additional ambiguity and complexity, when the system is deployed. Fourth, the trust scenarios can be used to generate trust rules, which can later be used to monitor the trustworthiness of requestors.

To identify possible trust scenarios of a system, developers need to determine the services provided by the system and the specific roles of different requestors. For each service, the specific details need to be analyzed which should include exploration of some well-known questions [17]. First, what is expected from a service? How it can be achieved? Second, what is the desired outcome of using the service? Third, how the service can be exploited by untrustworthy requestors? Fourth, what is the outcome in terms of an exploitation? This sort of analysis identifies the possible trust requirements of each service, from which different trust scenarios are elicited. Developers draw use case diagrams to represent different trust scenarios and use class diagrams to model the static attributes of the scenarios. They identify possible system states of the trust scenarios using state machine diagrams. The class diagrams and the state machine diagrams can be used to implement the trust scenarios.

To identify the related trust model of the system, developers need to determine the domain of the software. For example, service-based software are usually deployed in an open and decentralized environment. Therefore, developers need to identify possible ways of interactions in such an environment. Based on the analysis, developers may identify that the system needs direct interaction analysis as well as recommendations from other providers to determine the trustworthiness of requestors. This happens in the requirement elicitation phase, and based on the findings, the trust model is specified by generating the corresponding equations using suitable mathematical notation. The trust model must be able to use the trust scenarios for the service-based interaction analysis from trust perspectives. In the implementation phase, the trust model is implemented, incorporating the trust rules. In the deployment phase, the developed trust rules and the trust models are deployed along with the software. The trust rules and the trust models can be implemented and deployed in a trust monitoring architecture. The incorporation of a trust monitoring architecture into each service provider thus makes the service-based software trust-aware. The generation of the trust rules using the framework can be performed even after the system is deployed, if new trust concerns arise.

We provide a detailed description of UMLtrust in the next section. We present CAT, our developed trust model for service-based software in Chapter 4. The trust rules and the trust model are implemented in a trust monitoring architecture which is presented in Chapter 5.

3.2 UMLtrust: A UML profile for Trust Scenario Specification

The UML diagrams used in this framework are use-case, class, and state-machine diagrams. The use-case diagrams represent trust scenarios by focusing on the customer or managerial point of view. The class diagrams specify the static attributes and the structure of the trust scenarios, while the state machine diagrams present their dynamic aspects. In addition, the package diagrams store similar types of trust scenarios. We name these various diagrams trust-use-case diagrams, trust-class diagrams, trust-state-machine diagrams, and trust-package diagrams respectively.

Stereotypes and tagged values

The stereotypes and tagged values in UMLtrust specialize UML to the trust domain. These notations help developers avoid attaching many notes and constraints to regular UML diagrams (*i.e.*, to denote the trust requirements for different system functional requirements). The stereotypes of UMLtrust are provided in Table 3.1, where the first column shows the table index of the stereotypes listed in the second column, the third column mentions the base classes of corresponding stereotypes, and the fourth column describes the stereotypes. The first twelve (index 1-12) stereotypes are used in trust-use-case diagrams, where the first six stereotypes represent the actors and four different use-cases between them, and the `Connector` type stereotypes (index 7-12)

Table 3.1: UMLtrust stereotypes

Index	**Stereotype**	**Base Class**	**Description**
1	$<< trustor >>$	Actor	trusting actor
2	$<< trustee >>$	Actor	trusted actor
3	$<< trust\ service >>$	Use case	provided service
4	$<< trust - resource >>$	Use Case	shared resource
5	$<< trust - cert >>$	Use Case	trusted property in certificate
6	$<< trust - infr >>$	Use Case	trusted base infrastructure
7	$<< trusts >>$	Connector	trusting behavior
8	$<< owns >>$	Connector	certificate ownership
9	$<< holds >>$	Connector	holding/controlling a resource
10	$<< provides >>$	Connector	providing a service
11	$<< uses >>$	Connector	actions performed by trustee
12	$<< exploit >>$	Connector	
13	$<< trustor >>$	Class	trustor class
14	$<< trustee >>$	Class	trustee class
15	$<< trust - concern >>$	Class	concern related to a trust-context of a trust relationship
16	$<< trust\ rules >>$	Package	package to store trust-rules

are used to imply the impact of different connections in the trust-use-case diagrams. The next three stereotypes in the table (index 13-15) are used in the trust-class diagrams, and the last stereotype (index 16) is used for trust-package diagrams.

The tagged values used in UMLtrust are provided in Table 3.2, where the first column lists the tagged values, the second column places the tagged values in their corresponding classes, and the third column describes them. The tagged values are used in trust-class diagrams.

Trust-use-cases

Trust-use-cases are very helpful to represent trust relationships, are easy to understand, and thus quite important. Developers can use the trust-use-case diagrams

Table 3.2: UMLtrust tagged values

Tagged Value	**Class**	**Description**
goal	$<< trustor >>$	The goal to be achieved from a trust relationship.
min-trust-level	$<< trustor >>$	Minimum level of trust needed to enforce to construct a trust relationship (*e.g.*, LOW, MEDIUM, HIGH).
req-attr	$<< trustor >>$	The required quality expected from a trust relationship (*e.g.*, competence, security, honesty).
level	$<< trust - concern >>$	The direction of a trust relationship from $<< trustor >>$ to $<< trustee >>$ (*e.g.*, u-to-u for user to user, u-to-s for user to system, s-to-u for system to user).
type	$<< trust - concern >>$	The specific type (*i.e.*, package) of a trust concern.
method	$<< trustee >>$	Methods used in a trust relationship (*e.g.*, provides, exploit).

to identify potential areas where trust aspects should be injected into a system. In UMLtrust, trust-use-cases are the basis of all the diagrams. Each of the trust-use-cases consists of a $<< trustor >>$ actor, a $<< trustee >>$ actor, a use-case, and the connection between them specifies a trust relationship.

As the requestors and providers in a service-based system depend on each other for goals to be achieved, tasks to be performed, and resources to be utilized, the relationships between them are categorized into four forms [19] shown in rows three through six of Table 3.1. The four categories are described in the next paragraphs.

$<< trust\ service >>$: A $<< trustor >>$ trusts a $<< trustee >>$ to invoke services provided by it ($<< trustor >>$) in a trustworthy manner (*e.g.*, without exploiting the services).

$<< trust - resource >>$: A $<< trustor >>$ trusts a $<< trustee >>$ to use resources that it owns or controls. For example, a software execution environment or an application service can be treated as $<< trust - resource >>$.

$<< trust - cert >>$: A $<< trustor >>$ may trust a $<< trustee >>$ based on a certification provided by a third party. In a software system, this certification can be treated as the reputation or the valid user-ID of the $<< trustee >>$.

$<< trust - infr >>$: A $<< trustor >>$ has to trust some base infrastructure which the system offers (*e.g.*, the hardware used by $<< trustor >>$, or the operating system on which the system is running).

In a use-case diagram, a use-case encompasses an activity between the corresponding actors. For example, a trustor and a trustee in a service-based software forms a trust relationship based on the provided service (*i.e.,* $<< trust\ service >>$), which is a system functionality. However, the last three categories of trust relationships (*i.e.,* $<< trust - resource >>$, $<< trust - cert >>$, $<< trust - infr >>$) do not represent specific activities, and so cannot be treated as standard use-cases. To facilitate their usage in trust-use-case diagrams, we extend use-case diagrams. Figure 3.2 provides the iconic representations of $<< trust - cert >>$, $<< trust - resource >>$, and $<< trust - infr >>$. The iconic representations thus (i) incorporates them in the trust-use-case diagrams, and (ii) designates their difference from the actors and other use-cases. UML 2.0 supports the use of such iconic representations in use-case diagrams, an example of which is CORAS [74].

The `Connector` type stereotypes designate different trust relationships in the trust-use-case diagrams. The trusting behavior of $<< trustee >>$ for a service, resource, certification, or infrastructure is denoted by $<< trusts >>$. $<< owns >>$ implies the ownership of a particular certification, and $<< holds >>$ represents

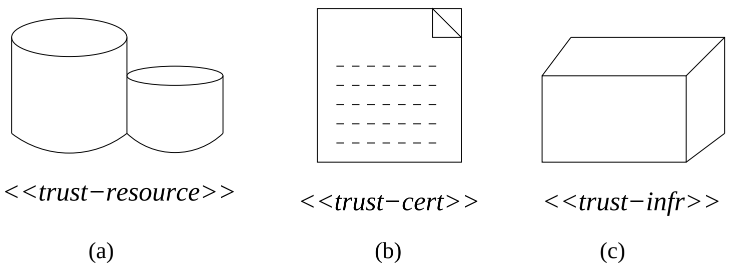


Figure 3.2: Representative icons for stereotypes (a) $<< trust - resource >>$, (b) $<< trust - cert >>$ (c) $<< trust - infr >>$

holding or controlling a resource, while the provision of a service is represented by $<< provides >>$. $<< uses >>$ specifies the use of a specific methodology to achieve something (*e.g.*, achieving a certification). $<< exploit >>$ designates a risk behavior of a trust-use-case which the $<< trustee >>$ uses to deceive or harm the $<< trustor >>$.

Trust-classes

After capturing the trust scenarios in the trust-use-case diagrams, the next step is to bind their static nature in the trust-class diagrams. Three types of stereotyped classes are used: $<< trustor >>$, $<< trustee >>$, and $<< trust - concern >>$.

$<< trustor >>$ represents the trusting entity which uses tagged values to imply some trust related information that is necessary to build a trust relationship. The tagged values for $<< trustor >>$ class are `goal`, `min-trust-level`, and `req-attr`. `goal` is used to identify the specific goal the $<< trustor >>$ wants to achieve from a trust relationship. `min-trust-level` suggests the minimum level of trust that is required in the trust calculation for this trust scenario. `req-attr` specifies the trust attribute that should be present to build the trust relationship (*e.g.*, competent to perform a task, or secure to protect something).

The $<< trustee >>$ class represents the trusted entity, which has one tagged value, `method`. This tagged value is used to identify the intention of the $<< trustee >>$ while building a trust relationship.

The $<< trust - concern >>$ class encapsulates the specific concern on which a trust relationship should be based. This class uses specific attributes which need to be monitored, specified as two tagged values: `level` is used to identify the type of the trust relationship that is being addressed by this class (*e.g.*, `u-to-u` is for trust relationship between two requestors or users, `u-to-s` and `s-to-u` are for trust relationship between the users and the system itself, where `u-to-s` specifies that the user is the trustor and the system is the trustee); `type` is used to categorize the $<< trust - concern >>$ class to a specific context, identifying the package of the corresponding trust rules.

Trust-packages

There is only one stereotyped package in UMLtrust, $<< trust - rules >>$. Different systems have different types of trust rules. Therefore, the $<< trust - rules >>$ package should be categorized based on a system. For example, for a system with M categories of trust rules, there will be M types of $<< trust - rules >>$ packages, each with different names. This helps in storing the trust rules in the packages based on their similarity.

Trust-state-machines

Trust-state machines are very helpful to explore the dynamic nature of trust-scenarios. Depending on the states of a particular scenario, it can be determined whether trust is violated or not. We use the original state-machine diagrams to represent the trust-scenarios, with two types of final states in our state-machine diagram. Trust is monitored at run-time, when interactions take place between entities. Depending on the outcome of an interaction, the final state can be successful or unsuccessful. A successful final state is denoted as $I(S)$, defined as an interaction where no trust-violation is occurred. An unsuccessful final state is denoted as $I(U)$, when a violation of trust is detected.

3.3 An Example Trust Scenario

In this section, we use an example trust scenario from file sharing applications [77–79] to illustrate the applicability of UMLtrust. Service requestors use file servers mainly for sharing resources (*i.e.*, files) with each other. File sharing applications are becoming more and more popular due to the need of collaborative groupware and scientific projects, where the sharing of information (files) is very crucial [66]. It is obvious that not all the requestors are benevolent, and so they may exploit the file sharing services for their own purposes. Therefore, trust concerns should be incorporated in the development of such a file sharing server to analyze the trustworthiness of requestors. Based on the analysis of service-based interactions, servers can decide which requestors should be granted services.

A file sharing server provides four types of basic services to requestors: file uploading, searching, opening, and downloading. An untrustworthy requestor can become malicious while uploading files. A well known activity is to upload large files beyond the acceptable limit of the server [77–79]. The purpose of this misbehavior is twofold. First, the uploading of large files will waste file server space and may limit the space

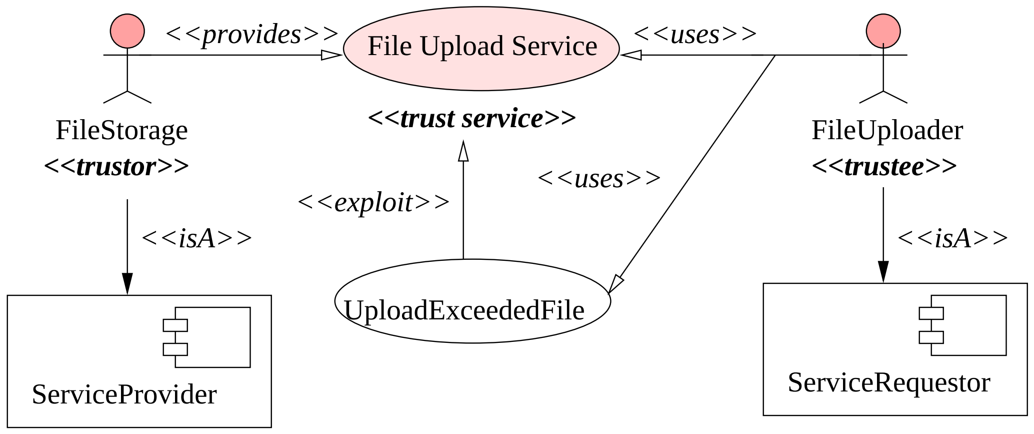


Figure 3.3: The trust-use-case diagram for the file excess trust scenario

for other requestors. Second, the requestor may try to upload a file of such a huge size that it will absorb all the space on the server and thus make the file uploading service completely unavailable to others. The file service provider can control this misbehavior by simply checking the file size while it is being uploaded. The checking can be facilitated by our file excess trust scenario as mentioned below.

Figure 3.3 presents the trust-use-case diagram representing the file excess trust scenario corresponding to the malicious requestor action described above. `FileUploader` is the service requestor, and `FileStorage` is the service provider. The provider trusts the requestor by providing it the file upload service, and the requestor invokes (<< *uses* >>) it. However, the requestor can upload excessive files (<< *exploit* >>) by using the `UploadExceedFile` activity.

Figure 3.4 provides the trust class diagram for the file excess trust scenario. The `goal` of the << *trustor* >> is to ensure safe uploading by preventing the upload of an excessively large file. The `req-attr` is security, while the `min-trust-level` is set to `MEDIUM`[1]. The << *trustee* >> exploits the service by uploading large files beyond

[1]The value of `min-trust-level` is capitalized to clarify that it should be treated as a numeric

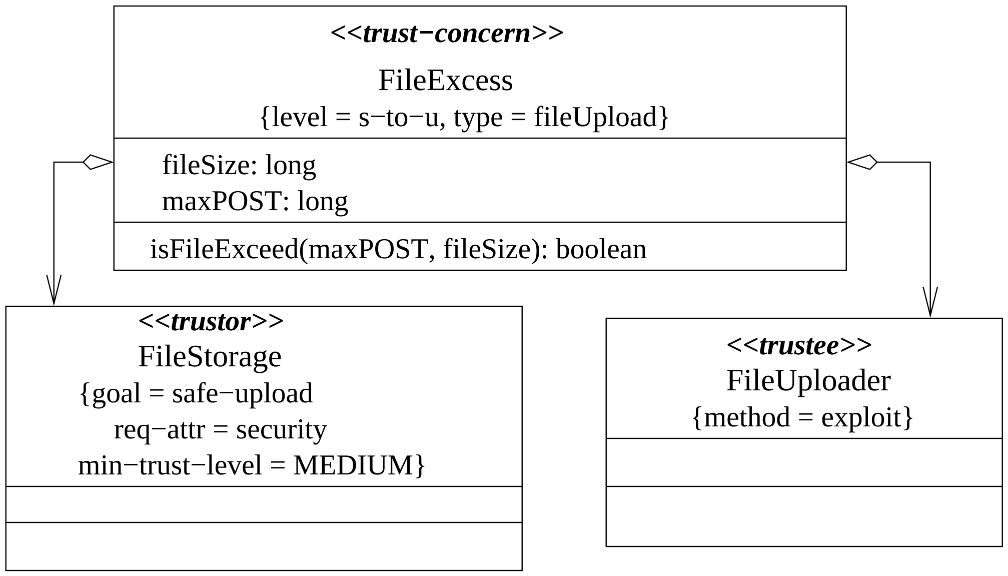


Figure 3.4: The trust-class diagram for the file excess trust scenario

the acceptable limit of the server. The $<< trust - concern >>$ is `FileExcess`, which has `level` set to s-to-u for a system to user level trust relationship. The `type` of the trust scenario is file uploading (`fileUpload`) because the context of the trust relationship is the file uploading service. `maxPOST` denotes the maximum acceptable file size the server will receive, while `fileSize` contains the size of the uploaded file. `isFileExceed(...)` is used to check whether the file being uploaded is less than or equal to `maxPOST`.

Figure 3.5 captures the trust-state diagram representing the file excess trust scenario. At first, the user requests a file upload to the file server (*req* state). Upon the granting of the file uploading service, the requestor uploads the file (*i.e., Upload* state). While uploading the file, the server uses the `isFileExceed(...)` function to check the uploaded file size. The function returns true if the file size is beyond

value. For example, the server can penalize a requestor based on the violation of the corresponding `FileExcess` trust rule, in which case this specification suggests that the requestor should be penalized with at least a `MEDIUM` trust value.

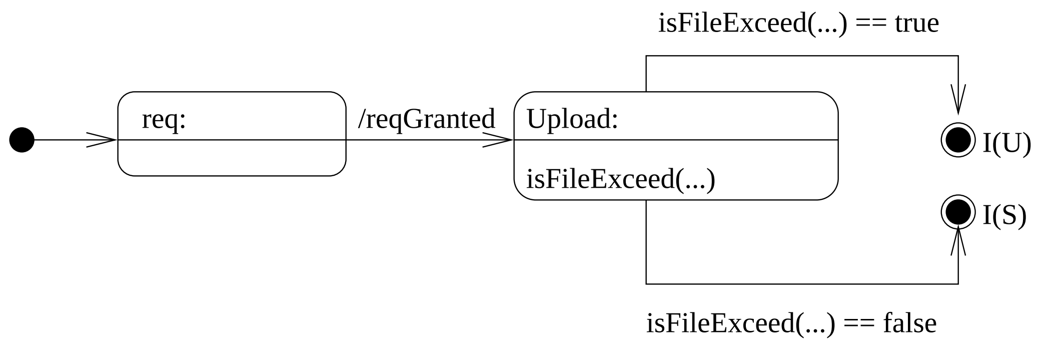


Figure 3.5: The trust-state-machine diagram for the file excess trust scenario

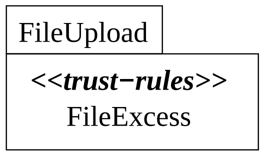
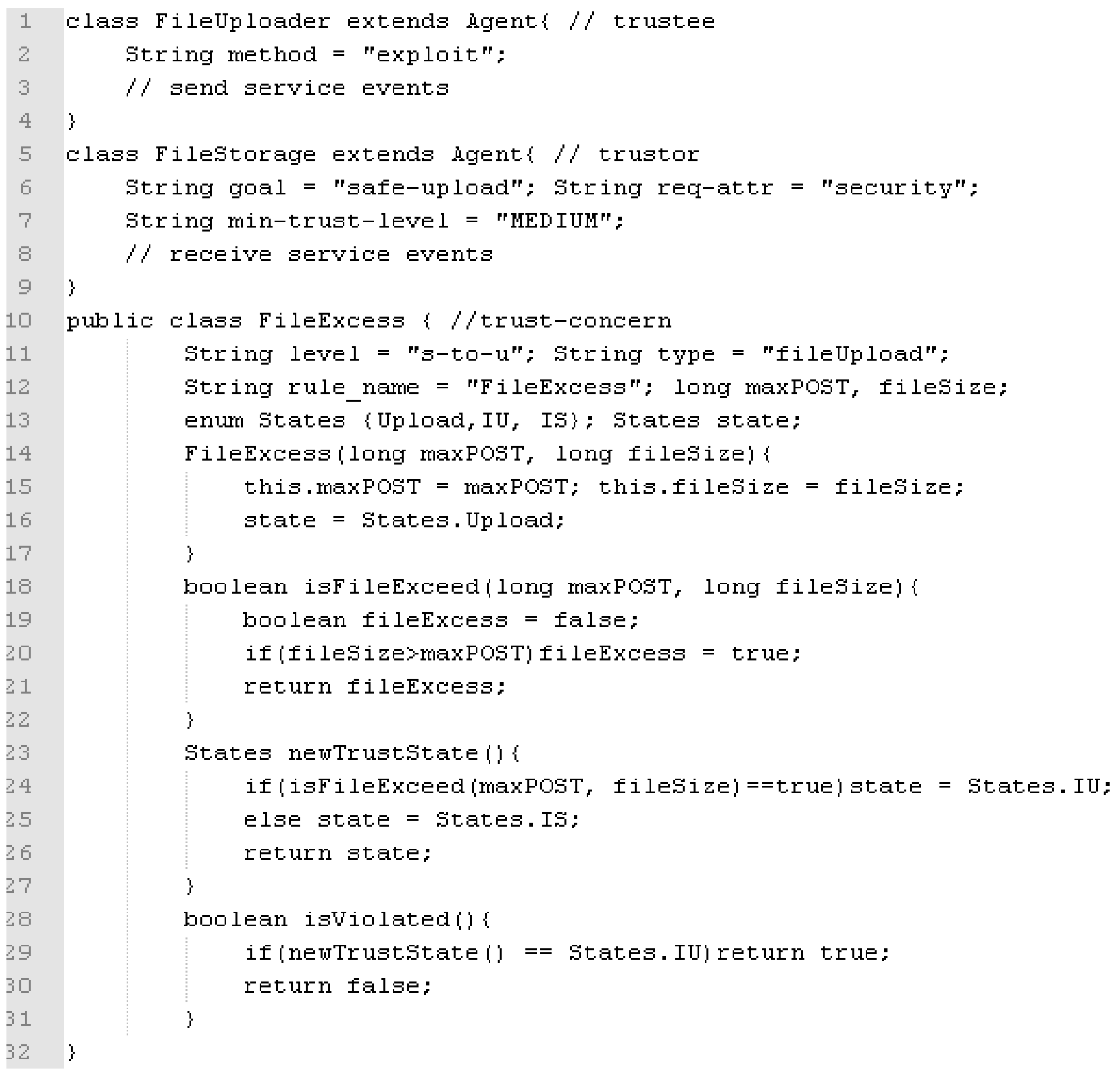


Figure 3.6: The trust-package diagram for the `FileExcess` trust rule

the acceptable limit, whereupon the interaction reaches $I(U)$, *i.e.,* unsuccessful. The interaction is successful if the the file size does not exceed the server regulations.

The corresponding trust rule is called `FileExcess` which is stored in a trust-package called `FileUpload` as shown in Figure 3.6. The `FileUpload` trust-package also stores any other trust scenarios related to file uploading.

We have manually converted the file excess trust scenario to Java code in the implementation phase of the framework which provides the corresponding `FileExcess` trust rule as shown in Figure 3.7. Both the provider and the requestor are implemented as agents, where an agent is an independent entity in a decentralized environment. The `method` in the `FileUploader` is set to "exploit" (Line 2). The trustee then sends the file uploading event to the `FileStorage`, which has set `goal`, `req-attr`, and `min-trust-level` according to the specification (Lines 6-7). The creation, sending and receiving of events are discussed in details in Section 6.3. The class name of this



```
class FileUploader extends Agent{ // trustee
    String method = "exploit";
    // send service events
}
class FileStorage extends Agent{ // trustor
    String goal = "safe-upload"; String req-attr = "security";
    String min-trust-level = "MEDIUM";
    // receive service events
}
public class FileExcess { //trust-concern
        String level = "s-to-u"; String type = "fileUpload";
        String rule_name = "FileExcess"; long maxPOST, fileSize;
        enum States {Upload,IU, IS}; States state;
        FileExcess(long maxPOST, long fileSize){
            this.maxPOST = maxPOST; this.fileSize = fileSize;
            state = States.Upload;
        }
        boolean isFileExceed(long maxPOST, long fileSize){
            boolean fileExcess = false;
            if(fileSize>maxPOST)fileExcess = true;
            return fileExcess;
        }
        States newTrustState(){
            if(isFileExceed(maxPOST, fileSize)==true)state = States.IU;
            else state = States.IS;
            return state;
        }
        boolean isViolated(){
            if(newTrustState() == States.IU)return true;
            return false;
        }
}
```

Figure 3.7: The `FileExcess` trust rule

trust rule is `FileExcess`, where the `level` and `type` are set according to the specification (Line 11), and local variables are initialized (Line 12). The working states for this trust rule are takes as $Upload$, IU, and IS (Line 13), since the transition from the req state has already been occurred by the time the trust rule code is entered (recall Figure 3.5). The `maxPOST`, `fileSize`, and the initial state are initialized (Lines 15-16). The `isFileExceed(...)` returns true if the `fileSize` exceeds the maximum allowed limit (Lines 18-22). The `newTrustState()` function returns the corresponding state (Lines 23-27). The final state is IU, *i.e.,* interaction unsuccessful, if `isFileExceed(...)` returns true. The function `isViolated()` then returns true by clarifying that the trust rule is violated (Lines 28-31).

We have implemented our prototype file service system in Jade 3.5 [83], where each of the trustors and the trustees are implemented as agents. Jade is written in Java and is a standard platform for agent-based system development. A brief overview on Jade is provided in Section 6.2. To test this trust rule, we construct a prototype file server and a requestor. The requestor can send service events to the file server regarding file uploading. In our prototype implementation, the requestor first sends a request for file upload to the server. If the server accepts this request, the client sends the file using an event $\{sr_1$, sp_1, $UploadDocFiles$, $docFileName - doc - 200 - fileContents$, $sr1300089544370$, $t_{session}\}$. Here, sr_1 is the ID of the client (*i.e.,* $<< trustee >>$), sp_1 is the ID of the file sharing server (*i.e.,* $<< trustor >>$), $UploadDocFile$ is the name of the service (*i.e.,* $<< trust - service >>$), $docFileName - doc - 200 - afileContents$ contains the parameters of this service, $sr1300089544370$ is the event ID generated by the system for this interaction, and $t_{session}$ is the time of this event. In $docFileName - doc - 200 - afileContents$, $docFileName$ is the name of the file, *doc* is the type of the file, 200 is the size of the file in megabytes (MB), and $afileContents$ is the contents of the file. The maximum acceptable size for a doc file in sp_1 is 100 MB[2]. The size of the uploaded size is greater than the acceptable size. Therefore, the sr_1 violates the `FileExcess` trust rule.

3.4 Summary

In this section, we present a trust-aware service-based software development framework. The framework uses our proposed UML profile called UMLtrust to specify trust scenarios and our proposed trust model called CAT (Context-Aware Trust) to incorporate trust calculation into total system development. The details of UMLtrust are provided and explained using an example from a file sharing application. The implementation of the trust scenario demonstrates the corresponding trust rule which is used for monitoring the corresponding service-based interactions with requestors from trust perspectives. Based on the analysis of interactions during monitoring, the trustworthiness of the requestors are calculated. The next chapter presents the detailed trust calculation schemes of our proposed trust model, CAT.

[2]This maximum file size is chosen arbitrarily. In practice, the limit in file size varies for different file sharing servers.

Chapter 4

A Trust Model for Service-Based Software

In this chapter, we present our proposed trust model, CAT (Context-Aware Trust). The trust model provides trust equations to determine the trustworthiness of requestors. The model uses trust rules specified in UMLtrust to analyze the service-based interactions with requestors. Based on the analysis, the model uses the underlying equations to calculate trustworthiness. The detailed calculation schemes of CAT are presented in Section 4.1. A trust-based service granting algorithm is outlined in Section 4.2 which uses CAT to decide whether a client should be provided with the service requested. The model and the algorithm are elaborated using examples from a file sharing grid in Section 4.3.

4.1 CAT: The Trust Model

CAT (Context-Aware Trust) is based on service-based interactions between entities. Services are offered by service provider entities, while the services are invoked by service requestors. An interaction can have a number of associated trust rules to analyze the interaction at run-time. A service requestor is penalized for any trust rule violation and awarded when no such violation occurs. Moreover, an alert is generated when the violation of a trust rule is detected in the interaction.

Direct trust is the quantified satisfaction of a trustor on a trustee for a provided service at a particular time based on the analysis of interactions with the trustee: direct trust is updated after each interaction. A provider acts as a recommender to other providers, passing on its satisfaction on a trustee in a quantified form. These recommendations are collected and compared against the outcome of the corresponding interactions. This comparison facilitates the measurement of recommendation accuracy which can be used to differentiate between reliable and unreliable recommendations.

The symbols used in CAT are summarized in Table 4.1, where the first column provides the symbols, the second column describes the symbols, and the third column mentions the numeric ranges of corresponding symbols. For example, in the first row, $(0.5, 1]$ denotes that the value of I_b should be greater than 0.5 but less than or equal to 1, while in the second row $[0, 0.5)$ mentions that the value of I_d should not be less than 0 but less than 0.5. For the sake of simplicity, we denote both the service provider and the requestor as entities (E). However, the trustor and the recommender entities are always service providers, and the trustee entities are always clients (*i.e.,* service requestors). A context (c_i) always denotes a service.

Table 4.1: Symbols used in CAT

Symbol	**Description**	**Range**
$I_b(E1, E2, c_i, t)$	Belief of $E1$ on $E2$ about context c_i at time t from interaction I	(0.5,1].
$I_d(E1, E2, c_,t)$	Disbelief of $E1$ on $E2$ about c_i at t from interaction I	[0,.5).
$\mu(E1, E2, c_i, t)$	Confidence of $E1$ on $E2$ about c_i at t	[0,1].
$T_D(E1, E2, c_i, t)$	Direct trust of $E1$ on $E2$ about c_i at t	[0,1].
$R_D(E3, E1, E2, c_i, t)$	Direct recommendation of $E3$ to $E1$ on $E2$ about c_i at t	[0,1].
$R_I(E3, E1, E2, c_i, t)$	Indirect recommendation of $E3$ to $E1$ on $E2$ about c_i at t	[0,1].
$A(E3, E1, E2, c_i, t)$	Accuracy of $E3$ in providing direct/indirect recommendation to $E1$ on $E2$ about c_i at t	[0,1].
$\theta(E3, E1, E2, c_i, t)$	Recommendation trust of $E2$ as measured by $E1$ about c_i based on the direct/indirect recommendation from $E3$	[0,1].
$R(E1, E2, c_i, t)$	Total recommendation trust calculated by $E1$ on $E2$ about c_i at time t	[0,1].
$T(E1, E2, c_i, t)$	Total trust of $E1$ on $E2$ about c_i at time t	[0,1].
$\gamma(t, t_r, c_i)$	Time-based ageing of trust about c_i from old time, t to new time t_r	[0,1].
$\vartheta(M, \Lambda, c_i)$	Path-based ageing of trust about c_i where M is the visited path length and Λ is the maximum allowed visiting path length	[0,1].
$\Re(c_i, c_j)$	Similarity between context c_i and c_j	[0,1].

In Table 4.1, I_b and I_d represent the level of belief and disbelief obtained from the analysis of a service-based interaction (*i.e.,* about a context) at a particular time respectively, while μ provides the confidence or satisfaction obtained by combining I_b and I_d (see Section 4.1.1). Each entity records its direct trust level (T_D) on interacted requestors (see Section 4.1.2), and a list of direct recommendations (R_D) and indirect recommendations (R_I) from other entities (see Section 4.1.5). The accuracy of a recommendation is denoted by A, while θ calculates a recommendation trust on a requestor based on a recommendation (see Section 4.1.5). The total recommendation trust R includes all the recommendation trusts calculated by θ (see Section 4.1.6), while total trust T calculates the trust from the direct trust and the total recommendation trust (see Section 4.1.7). γ is the time-based ageing parameter to decrease trust values over time without further interactions, ϑ is the path-based ageing parameter to give more importance on recommendations from nearer entities, while $\Re$ provides the similarity between two contexts in quantified form (see Sections 4.1.4, 4.1.5, and 4.1.3 respectively).

We provide the detailed calculation schemes of CAT in the next few subsections.

4.1.1 Interaction Trust: Calculating Confidence

'Confidence' is a quantified satisfaction obtained by a trustor on a trustee from the analysis of a service-based interaction with the trustee at a particular time. The calculation of trust begins whenever an interaction takes place, its outcome being used to calculate the confidence in the interaction. The calculation of confidence is thus the first step of the direct trust calculation (see Section 4.1.2). Each interaction

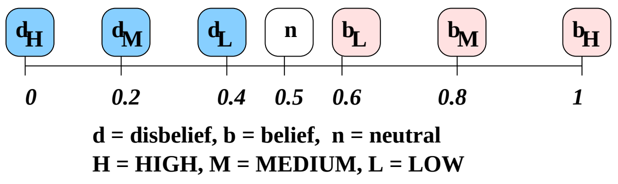

d_H
d_M
d_L
n
b_L
b_M
b_H
0
0.2
0.4
0.5
0.6
0.8
1
d = disbelief, b = belief, n = neutral
H = HIGH, M = MEDIUM, L = LOW

Figure 4.1: Trust values used in CAT

has certain trust rule(s) to compare against the outcome of the interaction at runtime, each of the which is assigned a particular trust value. For example, a trust rule can be assigned a high (H), medium (M), or low (L) trust value. The satisfaction of each trust rule awards the trustee a certain level of belief, while the violation of each trust rule penalizes the trustee with a certain level of disbelief. Figure 4.1 provides the trust values used in the model. The highest belief is assigned a value of 1, medium belief 0.8, low belief 0.6, while the highest disbelief score is 0, medium disbelief is 0.2, and low disbelief is 0.4. A neutral value of 0.5 denotes no belief and no disbelief[1].

The total belief (I_b) of trustor $E1$ on trustee $E2$ for context c_i at time t from interaction I is calculated using Equation (4.1), where $B(rn)$ contains the belief value of the trust rule indexed as rn, n_b is the total number of trust rules with belief outcomes. Similarly, total disbelief I_d (range $[0, 1]$) is calculated using Equation (4.2), where $D(rn)$ contains the disbelief value of the trust rule indexed as rn, and n_d is the total number of trust rules with disbelief outcomes. The confidence (μ) of $E1$ on $E2$ from I about context c_i at time t is calculated from I_b and I_d using Equation (4.3). The calculation of μ thus expresses the level of confidence $E1$ gathers on $E2$ from a particular interaction I at an interaction time t.

[1]The high, medium, low belief or disbelief values and the neutral value correspond to the trust values used by Haque and Ahamed [11]. We use these values since they fit well with our calculation range ($[0, 1]$).

$$I_b(E1, E2, c_i, t) = \frac{b(E1, E2, c_i, t)}{n_b},\ where\ b(E1, E2, c_i, t) = \sum_{rn=0}^{n_b} B(rn). \tag{4.1}$$

$$I_d(E1, E2, c_i, t) = \frac{d(E1, E2, c_i, t)}{n_d},\ where\ d(E1, E2, c_i, t) = \sum_{rn=0}^{n_d} D(rn). \tag{4.2}$$

$$\mu(E1, E2, c_i, t) = w_b I_b(E1, E2, c_i, t) + w_d I_d(E1, E2, c_i, t),\ where\ w_b + w_d = 1. \tag{4.3}$$

In Equation (4.3), w_b and w_d (range $[0, 1]$) are weights assigned to I_b and I_d respectively. In situations when the trustors want to emphasize the interactions with trustees despite their misbehavior, they can give more importance to w_b. By putting more weight on w_b, trustors can ensure that trustees can have additional chances up to certain limit, even though they misbehaved in the past. For example, if we provide values as $w_b = 0.8,\ and\ w_d = 02$, then we consider that the satisfactory interactions with a trustee (calculated by I_b) are given four times more importance than the unsatisfactory interactions (calculated by I_d), and so the trustee may have additional chances even after it has misbehaved earlier.

The calculation of confidence uses the trust rules to analyze the trustworthiness of entities. Based on the graveness of the trust violation, the importance value of trust rules differ. For example, a trust rule considering a highly malicious outcome is provided with a high importance. Therefore, the violation of this trust rule will penalize the trustee with the highest disbelief.

The use of trust rules in the trust calculation makes the trust model rule-oriented (see property P2 of Section 2.1.4), and these calculations are performed for each of the provided services which makes CAT context-aware (see property P1 of Section 2.1.4). Moreover, each trust rule is assigned a particular importance value based on its probable risk outcome. This satisfies the risk-awareness property of interaction-based trust (see property P3 of Section 2.1.4).

4.1.2 Calculating Direct Trust

Direct trust (T_D) of an entity $E1$ on $E2$ for context c_i at time t is calculated based on the analysis of the corresponding interactions between $E1$ and $E2$. The value of T_D changes after each interaction based on the outcome of the interaction. This satisfies the dynamism of trust (see property P8 of Section 2.1.4). The calculation of direct trust is the most important, as it is based on self-observation and does not depend on recommendations. The calculation of a direct trust T_D of $E1$ on $E2$ for c_i at time t has two options. The first option calculates T_D using the confidence (μ) gained from the analysis of interactions related to c_i, while the second option calculates T_D based on another context c_j, which is similar to c_i to some extent.

The first option of direct trust calculation is performed using Equation (4.4), where δ (range $[0, 1]$) is the weighting factor. When δ is given less weight (*e.g.*, for a fast changing environment), the latest confidence is preferred more than previous confidences. Upon the calculation of confidence μ for a context c_i at time t, the previous direct trust ($T_D(E1, E2, c_i, t_o)$) is retrieved from the corresponding interaction records, and then new direct trust is calculated using Equation (4.4).

$$T_D(E1, E2, c_i, t) = \delta T_D(E1, E2, c_i, t_o) + (1 - \delta)\mu(E1, E2, c_i, t). \tag{4.4}$$

The second option of direct trust calculation is used only when a decision needs to be made for $E2$ about c_i and there is no record of direct interaction for that context. Equation (4.5) handles this problem by calculating direct trust T_D of $E1$ on $E2$ for context c_i at time t using $\Re(c_i, c_j)$ (range $[0, 1]$), a context-similarity parameter which measures the similarity between context c_i and c_j, and $T_D(E1, E2, c_j, t)$ which provides the calculated direct trust of $E1$ on $E2$ for context c_j at time t. The context-similarity parameter is discussed further in Section 4.1.3.

$$T_D(E1, E2, c_i, t) = T_D(E1, E2, c_j, t)\Re(c_i, c_j). \tag{4.5}$$

Note that in Equations (4.4), and (4.5), t represents the current time index, and t_o represents the time of the most recent direct trust calculation stored for a particular entity pair and context.

Without further interaction, direct trust decreases over time. For example, if $E1$ calculates direct trust T_D on $E2$ for c_i at time t_o and the recent time is t, then the recent direct trust $T_{D_r}(E1, E2, c_i, t)$ should decrease without further interaction due to the time-based ageing property (see property P9 of Section 2.1.4). Equation (4.6) calculates the recent direct trust of $E1$ on $E2$ about c_i at time t, where we assume that the direct trust T_D of $E1$ on $E2$ about c_i at time t_o is obtained from the corresponding interaction records (*i.e.,* the T_D previously calculated using Equation (4.4) or Equation (4.5)). For example, suppose we have a record of direct trust at time

$t_o = 100$ but at time $t = 200$, we need the direct trust from the record. To obtain the recent direct trust, we use Equation (4.6) on the direct trust T_D found at time $t_o = 100$. The calculation of recent direct trust is performed using a time-based ageing parameter $\gamma(t_o, t, c_i)$ (range $[0, 1]$). The time-based ageing parameter is elaborated further in Section 4.1.4.

$$T_{D_r}(E1, E2, c_i, t) = T_D(E1, E2, c_i, t_o)\gamma(t_o, t, c_i). \tag{4.6}$$

Equation (4.7) calculates the generalized trust of $E1$ on $E2$ for all the available contexts, where $\rho(c_i)$ (range $[0, 1]$) is the priority of context c_i, and n is the total number of available contexts. The generalized trust is thus a priority-weighted mean of all direct trusts. For example, if each of the n contexts has the priority of 1, then the generalized trust of $E1$ on $E2$ provides an average of all the direct trusts obtained for all available contexts. The priority is necessary for real world situations, where a trustor does not consider all the contexts equally.

$$T_D(E1, E2, t) = \frac{\displaystyle\sum_{i=0}^{n} \rho(c_i) T_D(E1, E2, c_i, t)}{\displaystyle\sum_{i=0}^{n} \rho(c_i)}. \tag{4.7}$$

4.1.3 Context-Similarity Parameter ($\Re(c_i, c_j)$)

Recall that, the context-similarity parameter is necessary, when no previous interaction value is available in the record database of a trustor ($E1$) for a trustee ($E2$) regarding context c_i. We assume that every context has some keywords to describe it.

We let $\kappa(c_i)$ denote the total set of keywords describing context c_i. Then a similarity metric can be based on the ratio of keywords two contexts (*e.g.,* c_i and c_j) have in common.

Equation (4.8) provides the definition of context-similarity parameter, where the numerator takes total number of similar keywords that two contexts c_i and c_j have in common (*i.e.,* the union of all keywords for c_i and c_j), while the denominator takes total number of distinct keywords used to describe c_i and c_j (*i.e.,* the intersection of all keywords for c_i and c_j).

$$\Re(c_i, c_j) = \frac{length\_of[\kappa(c_i) \cap \kappa(c_j)]}{length\_of[\kappa(c_i) \cup \kappa(c_j)]}. \tag{4.8}$$

For example, suppose a file-server application has three types of services (*i.e.*, contexts): `UploadPDFFile` with keywords {`write, pdf, file`}, `UploadDocFile` with keywords {`write, doc, file`}, `Login` with keywords {`userName, passWD`}. Therefore, for `Login` and `UploadPDFFile` the numerator is 0 and the denominator is 5, giving the value of $\Re(c_i, c_j)$ as 0 (*i.e.*, the contexts are not similar at all). However, for `UploadPDFFile` and `UploadDocFile` the numerator is 2, denominator is 4, and so the value of the parameter is 0.50 (*i.e.*, if $E1$ trusts $E2$ for uploading `doc` files, it can reasonably trust $E2$ for uploading `PDF` files). This parameter helps in making decisions in similar situations, so as to decrease the reliance on recommendations and to rely on self-observation more.

The notion of context-similarity is first provided by Toivonen *et al.* [50] and then Billhardt *et al.* [39], where the first ones consider a context-tree, with the similarity between contexts is calculated as the distance between them based on the root node of the tree. However, the problem with the above approach is the formation of the context-tree, which might not be present for all the contexts in the system. However, if we can construct such a tree based on our service-keywords, our parameter will produce the same result. Billhardt *et al.* [39] considers a chain of services in a organizational structure, where the invocation of one service may lead to another service. We however, do not consider such scenario. Since we apply keywords to describe each service, our approach is applicable to services in general, and it eliminates the need for any such context-tree or organization-level service hierarchy. Although our approach of context-similarity can be adapted for a hierarchical structure, our parameter will produce different result than [39] because of the difference in calculation schemes.

4.1.4 Time-Based Ageing Parameter ($\gamma(t, t_r, c_i)$)

This parameter is used to reduce a trust value over time when no further interaction takes place. It is calculated using Equation (4.9), where Υ (range $[0, 1]$) is a time-based ageing factor. The closer Υ is to 1, the lower the value of γ, and the value of previous direct trust decreases with the decrease in γ. For essentially static environments, where very few number of interactions take place within a particular time period, the value of Υ might be chosen close to 0.

$$\gamma(t_o, t, c_i) = 1 - \frac{(t - t_o)\Upsilon}{t}. \tag{4.9}$$

In Equation (4.9), t_o is the last time of interaction between entity $E1$ and $E2$ for c_i, and t is the current time, when a trust calculation on $E2$ for c_i is required. $(t - t_o)/t$ provides the difference between t and t_o in terms of t, where the division by t restrains the ratio in the range $[0, 1]$.

For example, suppose a service-based system with very few number of interactions taking place per second (*e.g.,* 10 to 100), the value of Υ is chosen as 0.2. Let us assume that the direct trust on $E2$ for context c_i was calculated by $E1$ using either Equation (4.4) or Equation (4.5) at time $t_o = 100$th second as 0.78, and we need to decide on $E2$ for c_i at time $t = 50000$th second. We further assume that, $E2$ did not interact with $E1$ at all between t_o and t. We then use Equation (4.6) on top of the previously calculated direct trust to calculate the recent direct trust T_{D_r}, which becomes 0.624. The values of different constants are chosen arbitrarily and merely for describing the equations and corresponding operations. In a real-world system, the values should be chosen carefully based on observations.

The notion of time-based ageing is first introduced by Sabater and Sierra in their trust model called REGRET [34], calculated as $f(t_o, t) = \frac{t_o}{t}$. The problem with this parameter is that it only takes the ratio of current time and old time, and does not give preference to the system environment. For example, in a slowly interacting system, entities do not interact frequently, and so this parameter may drastically reduce the recent direct trust value. To address this shortcoming, we use our time-based ageing factor (Υ), where the value of Υ is chosen based on the nature of the system environment. Unlike them, we are taking the difference between the old time and the recent time, so we subtract our ratio from 1.

4.1.5 Calculating Recommendation and Recommendation Accuracy

We consider two types of recommendations: direct and indirect. Recommendations obtained from immediate neighbors [11] are considered to be direct, while the recommendations obtained from further afield are considered indirect. Figure 4.2 describes the two types of recommendations: the recommendation from $E5$ on $E2$ to $E1$ is indirect as it comes through $E4$ (*i.e.*, path-length = 3), the recommendation from $E3$ on $E2$ to $E1$ is direct (*i.e.*, path-length = 2). We consider the number of visited nodes (*i.e.,* vertices) in a recommendation network to calculate a path-length, where a recommendation network consists of all the providers in a service-based system. A recommendation path is indirect, if it has path-length more than 2.

An entity $E3$ provides direct recommendation (R_D) to $E1$ for $E2$ about context c_i at time t using Equation (4.10). η (range $[1, 0]$) is the weighting factor an entity imposes on its direct trust calculated at time t_o for the purpose of recommendation. A recommendation value R_D is at most equal to the corresponding direct trust value in our system.

$$R_D(E3, E1, E2, c_i, t) = \eta T_D(E3, E2, c_i, t_o). \tag{4.10}$$

We do not consider context-similarity parameter in providing recommendations, and so a recommender in our system only recommends a trustee for context c_i if the trustee has previously interacted with the recommender for the same context c_i. The reason for not choosing context-similarity is that this parameter only provides an assumed trust value derived from a similar context. Since entities in our system judge the recommendations from each other very carefully (*i.e.,* through the calculation of recommendation accuracy), they may want to avoid any more uncertainty in providing or receiving a recommendation.

The indirect recommendation (R_I) from entity $E5$ to $E1$ on $E2$ about c_i at time t is calculated using Equation (4.11), where $\vartheta(M, \Lambda, c_i)$ is the path-based ageing parameter. $\vartheta(M, \Lambda, c_i)$ is adapted from [11], but it is calculated in a different manner using Equation (4.12). M is the visited path-length in a recommendation path, Λ is the

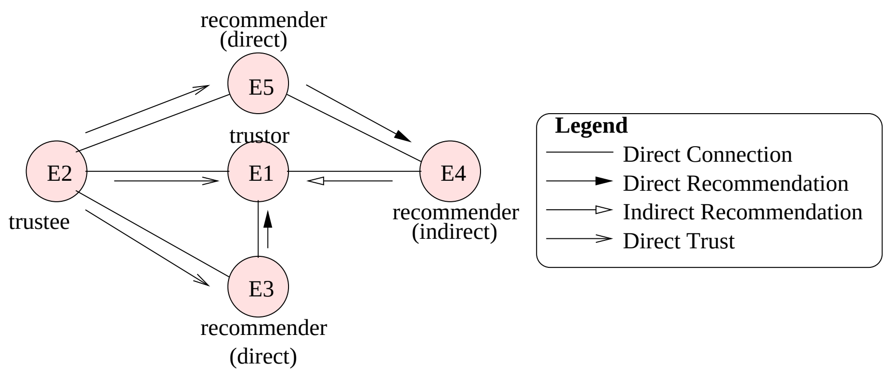


Figure 4.2: Direct and indirect recommendations in CAT

maximum allowed path-length for this recommendation path, and Ψ is the distance-based ageing factor. Using path-based ageing parameter, indirect recommendations are given less weight than direct recommendations.

$$R_I(E5, E1, E2, c_i, t) = R_D(E5, E1, E2, c_i, t)\vartheta(M, \Lambda, c_i). \tag{4.11}$$

$$\vartheta(M, \Lambda, c_i) = 1 - \frac{(M-2)\Psi}{\Lambda}, \ where\ M \geqslant 2. \tag{4.12}$$

The value of Λ can be changed based on preferences. For example, in a highly uncertain service-based system, where a service provider has no way of knowing about other recommenders, it can consider recommendations only from a number of recommenders whom it or its neighbors know properly. Moreover, it takes considerable time to get a recommendation reply if the maximum allowable path length is pretty long (as identified in Section 7.4). Therefore, if a service provider wants to rely more on the recommenders closer to it as well as wants to avoid delay in providing services it can set a low value for Λ. The use of a path-based ageing parameter in the calculation of indirect recommendations satisfies the path-based ageing property of interaction-based trust (see property P10 of Section 2.1.4).

The original parameter proposed by [11] is $1 - \frac{(M-1)\Psi}{10}$. The problem with the original parameter is that it has 10 as denominator, and therefore, at some point it may become negative. For example, for M = 16, $\Psi = 0.9$, the value of the parameter is -0.35. We address this problem by using maximum allowed path length Λ instead of 10. Moreover, since they consider the edges in a recommendation network to calculate path-length as opposed to the vertices, the subtraction in their numerator is by 1, while it is by 2 in our numerator.

The accuracy (A) of $E3$ to $E1$ in providing a direct recommendation for $E2$ about context c_i is calculated using Equation (4.13), where $\Delta R_D(E3, E1, E2, c_i, t)$ calculates the absolute difference between the provided recommendation and the calculated direct trust (see Equation (4.14)).

$$A_{R_D}(E3, E1, E2, c_i, t) = 1 - \Delta R_D(E3, E1, E2, c_i, t). \tag{4.13}$$

$$\Delta R_D(E3, E1, E2, c_i, t) = |R_D(E3, E1, E2, c_i, t) - T_D(E1, E2, c_i, t)|. \tag{4.14}$$

Each trustor keeps an accuracy table (AT), where it updates the accuracy of every recommendation after the analysis of the corresponding interaction with a trustee. The accuracy of a direct recommender $E3$ to $E1$ about $E2$ regarding context c_i at time t in the AT is denoted by $AT_{R_D}(E3, E1, E2, c_i, t)$. The update in the AT at time t is performed using Equation (4.15) by considering previous recommendation accuracy ($AT_{R_D}(E3, E1, E2, c_i, t_o)$) at time t_o and new recommendation accuracy ($A_{R_D}(E3, E1, E2, c_i, t)$) at time t. ζ (range $[1, 0]$) weights the importance of the previous accuracy and the current accuracy of a recommender. This is necessary, since an entity may be completely wrong at some point. However, this cannot be used to conclude on all of its recommendations.

$$AT_{R_D}(E3, E1, E2, c_i, t) = \zeta AT_{R_D}(E3, E1, E2, c_i, t_o) + (1 - \zeta)A_{R_D}(E3, E1, E2, c_i, t). \tag{4.15}$$

Using Equations (4.13), and (4.15), unreliable recommendations can be detected. A recommender is considered most reliable if it has accuracy 1 and most unreliable if it has accuracy 0. Based on the measured accuracy, an entity can impose a certain accuracy threshold to discard recommendations. This satisfies the recommendation filtration property (see property P5 of Section 2.1.4).

The calculation of recommendation accuracy in Equations (4.13), (4.14), and (4.15) closely follows the calculation of Azzedin and Maheswaran [35], and like our Equation (4.14), they also measure the difference between direct trust and provided recommendations. However, they ([35]) do not differentiate between direct and indirect recommendations in their calculations. Moreover, Equations (4.13), and (4.15) are context-aware in our system, as opposed to the generalized forms used in their model.

It should be noted here that the accuracy of a recommendation does not necessarily indicate the trustworthiness of the recommender, as the measurement of an accuracy can be affected by the path-based ageing factor. Indirect recommendations are given less weight, and so an indirect recommender can be deemed unreliable even though it has provided a reliable recommendation. For example, suppose a scenario where the recommendation to $E1$ for $E2$ from a recommender $E9$ is provided as 0.78, where the visited path-length M is 9, with maximum allowable path-length Λ is 10. We further assume that the value of Ψ in $E1$ is chosen as 0.2, which thus calculates the indirect recommendation as 0.67. Suppose, after the corresponding interaction with the trustee, the trustor calculates the direct trust as 0.78, which thus provides the recommendation accuracy of $E9$ as 0.89 for that particular interaction, whereas the recommendation accuracy was supposed to be 1. Although this measurement of recommendation does not fully reflect the reliability of recommender $E9$, it helps to some extent to measure it and use it in the calculations afterwards.

Equations (4.13), (4.14), and (4.15) are also used for calculating the accuracy of indirect recommendations by replacing R_D by R_I. The calculation of recommendation points out that an entity can be trusted based on the provided recommendations to a certain extent. This satisfies the semi-transitive and recommendation-based trust properties (see properties P4 and P6 of Section 2.1.4).

4.1.6 Calculating Total Recommendation Trust

A direct/indirect recommendation trust (θ) is calculated by binding a direct/indirect recommendation with its previous accuracy. The direct recommendation trust as measured by $E1$ on $E2$ based on the recommendation provided by $E3$ for context c_i at time t is calculated by Equation (4.16). Similarly, indirect recommendation trust is calculated using Equation (4.17).

$$\theta_{R_D}(E3, E1, E2, c_i, t) = R_D(E3, E1, E2, c_i, t)AT_{R_D}(E3, E1, E2, c_i, t). \tag{4.16}$$

$$\theta_{R_I}(E5, E1, E2, c_i, t) = R_I(E5, E1, E2, c_i, t)AT_{R_I}(E5, E1, E2, c_i, t). \tag{4.17}$$

The total recommendation trust of $E1$ on $E2$ about context c_i is calculated using Equation (4.18), where N_D is the total number of direct recommenders and N_I is the total number of indirect recommenders. The generalized recommendation trust of $E1$ on $E2$ at time t is measured by Equation (4.19), where $\rho(c_i)$ is the priority of c_i and n is the total number of contexts.

$$R(E1, E2, c_i, t) = \frac{\sum_{e=0}^{N_D} \theta_{R_D}(e, E1, E2, c_i, t) + \sum_{e=0}^{N_I} \theta_{R_I}(e, E1, E2, c_i, t)}{N_D + N_I}. \tag{4.18}$$

$$R(E1, E2, t) = \frac{\sum_{i=0}^{n} \rho(c_i) R(E1, E2, c_i, t)}{\sum_{i=0}^{m} \rho(c_i)}. \tag{4.19}$$

4.1.7 Calculating Total Trust

After obtaining the direct trust and the total recommendation trust on $E2$, $E1$ calculates total trust (T) on $E2$ for context c_i using Equation (4.20). The calculation is performed based on recent direct trust T_{D_r} obtained by Equation (4.6), and total recommendation trust obtained from Equation (4.18). The calculation of T_{D_r} is assisted by the calculations of belief, disbelief, confidence and previous direct trust, while the calculation of R is assisted by the calculations of direct, indirect recommendations and their accuracy measurements as described earlier. α (range $[0, 1]$) is the self-confidence level. An entity can increase α to rely on self-observation more, and in the limit can discard all recommendations by using $\alpha = 1$.

$$T(E1, E2, c_i, t) = \alpha T_{D_r}(E1, E2, c_i, t) + (1 - \alpha) R(E1, E2, c_i, t). \tag{4.20}$$

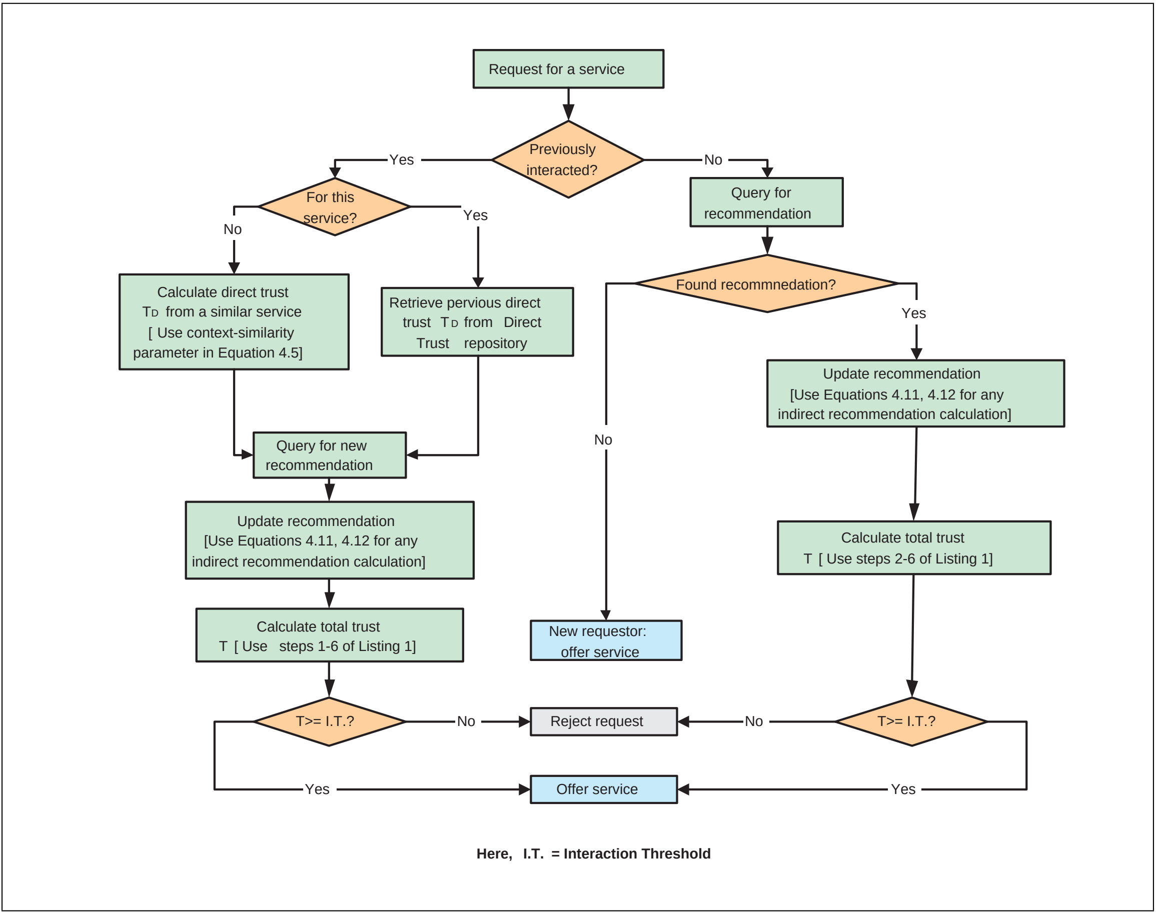
4.2 A Trust-Based Service Granting Algorithm

In this section, we present a trust-based service granting algorithm which uses CAT to make trust-based dynamic decision on behalf of a service provider (sp). Based on a request for a service (s) from a service requestor (sr), the algorithm provides a decision on whether the requestor will be granted the service or not. We assume that the time of this service request is t. A flowchart representation of the trust-based service granting algorithm is provided in Algorithm 1 which operates in two distinct conditions:

- The requestor has interacted with the provider previously, and therefore, is known to the provider.
- The requestor has not interacted with the provider before, and therefore, is unknown to the provider.

Each of the providers in our service-based system stores direct trust in a `Direct Trust` repository, recommendations from other entities in a `Recommendations` repository, and recommendation accuracy in a `Recom-Accuracy` repository.

The provider uses CAT to make decisions, checking whether the requestor has interacted previously for the requested service. This is determined by searching the `Direct Trust` repository for available interaction record(s) of the requestor. If the requestor has a previous interaction record for the service, the direct trust value on the requestor about the service is loaded from the `Direct Trust` repository. However, in case of no direct interaction with the requestor for that particular service, the provider uses the context-similarity parameter (Equation (4.8)) in Equation (4.5)


Request for a service
Previously interacted?
Yes
No
For this service?
Query for recommendation
No
Yes
Found recommnedation?
Calculate direct trust TD from a similar service [Use context-similarity parameter in Equation 4.5]
Retrieve pervious direct trust TD from Direct Trust repository
Yes
Update recommendation [Use Equations 4.11, 4.12 for any indirect recommendation calculation]
No
Query for new recommendation
Update recommendation [Use Equations 4.11, 4.12 for any indirect recommendation calculation]
Calculate total trust T [Use steps 2-6 of Listing 1]
Calculate total trust T [Use steps 1-6 of Listing 1]
New requestor: offer service
T>= I.T.?
No
Reject request
No
T>= I.T.?
Yes
Offer service
Yes
Here, I.T. = Interaction Threshold

Algorithm 1: Trust-based service granting

to obtain the direct trust value on the requestor for the service (*i.e.*, the second option of calculating T_D). Then the provider makes queries for recommendations about the requestor for the service. Upon the receipt of all recommendations, the provider updates the `Recommendations` repository, using Equation (4.11) and Equation (4.12) for any indirect recommendation. Finally, the total trust T is calculated by following all the steps in Listing 4.1, where the accuracy threshold τ_A is set up by the provider. The accuracy threshold can vary for different service providers. The service is offered to the requestor if T is not less than the interaction-threshold $(I.T.)$ for the service. The interaction threshold is set up by the sp and can vary for different

Listing 4.1 Total trust calculation in Algorithm 1

```
1. Use Equation (4.6) to obtain recent direct trust, $T_{D_r}(sp, sr, s, t)$.
2. Load recommendations about $sr$ for $s$ from the Recommendations repository.
3. Load accuracy of each recommender from the Recom-Accuracy repository.
4. Discard recommendations with accuracy level less than accuracy-threshold, $\tau_A$.
5. Calculate total recommendation trust $R(sp, sr, s, t)$ using Equations (4.16), (4.17), and (4.18).
6. Calculate total trust $T(sp, sr, s, t)$ using Equation (4.20).
```

service providers.

When the provider finds no record for the requestor in its `Direct Trust` repository, it queries for recommendations about the requestor for the service. If no recommendations are found, provider determines that the requestor is an absolutely new entity to the system. It then assigns the requestor the neutral value 0.5 and offers service. However, the entity is not considered completely new if some recommendations are found. In this case, the `Recommendations` repository is updated upon the arrival of all recommendations. Then the total trust T is calculated using Steps $2-6$ of Listing 4.1, and the service is provided if warranted ($i.e.,\ T \geqslant I.T.$).

After the service is granted to the requestor, an interaction session for this service is initiated between the provider and the requestor, during which time the interaction events are analyzed using the corresponding trust rules. After the analysis of each interaction based on the trust rules, the provider calculates μ and updates T_D in the `Direct Trust` repository using Equations (4.1), (4.2), (4.3), (4.4). The provider then calculates the corresponding recommendation accuracies using Equations (4.13), (4.14), (4.15), updating the `Recom-Accuracy` repository. The analysis of interactions, calculating, and updating of direct trust and recommendation accuracy are performed in the trust monitoring phase, which is described in the next chapter (Chapter 5).

4.3 An Illustrative Example

In this section, we present examples from file sharing applications to calculate trust using CAT and make trust-based automatic decisions using the trust-based service granting algorithm. We retain the same file sharing applications from Section 3.3 and adapt those into a file storage grid [46, 67]. In a grid structure [68–70], there are a number of servers to provide services. Such a file sharing grid can be used for collaborative groupware projects, or most importantly for scientific purposes [66]. This kind of grid structure is distributed and publicly accessible, and therefore, suitable for a large amount of resource sharing among entities. While requestor entities decide to access a particular server for their purposes, file servers also need to reason about the trustworthiness of requestors before granting them access to the server to share resources (*i.e.,* file upload, open, and search). Servers can employ certain trust rules for file storage such as `FileExcess` (uploaded files should have some acceptable size), `FileHarmful` (requestors should not upload any file containing malicious scripts), `FileSpam` (requestors should not upload insignificant files to the server to waste server space)[2]. Due to the gravity of the corresponding outcome, `FileHarmful` can be given high importance and `FileExcess` and `FileSpam` can be considered of medium and low importance respectively. Each of the file servers consult with other servers in the grid for recommendations, uses CAT as their trust model and employs the trust-based service granting algorithm to decide about service requests.

Figure 4.3 presents different cases of the trust algorithm with respect to the file sharing grid. A client is a service requestor (sr), the requested service (s) is `UploadDocFile`, and the file sharing server is the service provider (sp). We assume

[2]The last two trust rules are discussed in Section 7.2.1.

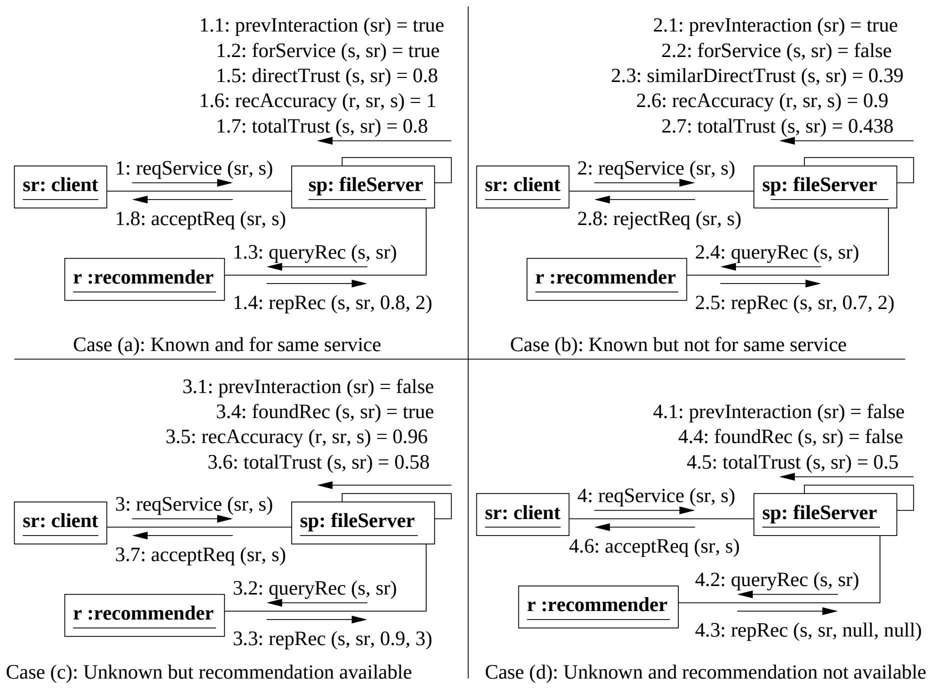

1.1: prevInteraction (sr) = true
1.2: forService (s, sr) = true
1.5: directTrust (s, sr) = 0.8
1.6: recAccuracy (r, sr, s) = 1
1.7: totalTrust (s, sr) = 0.8
sr: client
1: reqService (sr, s)
sp: fileServer
1.8: acceptReq (sr, s)
1.3: queryRec (s, sr)
r :recommender
1.4: repRec (s, sr, 0.8, 2)
Case (a): Known and for same service
2.1: prevInteraction (sr) = true
2.2: forService (s, sr) = false
2.3: similarDirectTrust (s, sr) = 0.39
2.6: recAccuracy (r, sr, s) = 0.9
2.7: totalTrust (s, sr) = 0.438
sr: client
2: reqService (sr, s)
sp: fileServer
2.8: rejectReq (sr, s)
2.4: queryRec (s, sr)
r :recommender
2.5: repRec (s, sr, 0.7, 2)
Case (b): Known but not for same service
3.1: prevInteraction (sr) = false
3.4: foundRec (s, sr) = true
3.5: recAccuracy (r, sr, s) = 0.96
3.6: totalTrust (s, sr) = 0.58
sr: client
3: reqService (sr, s)
sp: fileServer
3.7: acceptReq (sr, s)
3.2: queryRec (s, sr)
r :recommender
3.3: repRec (s, sr, 0.9, 3)
Case (c): Unknown but recommendation available
4.1: prevInteraction (sr) = false
4.4: foundRec (s, sr) = false
4.5: totalTrust (s, sr) = 0.5
sr: client
4: reqService (sr, s)
sp: fileServer
4.6: acceptReq (sr, s)
4.2: queryRec (s, sr)
r :recommender
4.3: repRec (s, sr, null, null)
Case (d): Unknown and recommendation not available

Figure 4.3: Interaction diagrams elaborating the different cases of the trust-based service granting algorithm

that *sp* uses 0.52 as the interaction threshold $(I.T.)$[3]. We discuss the following four cases that will occur in the presented algorithm: (a) *sr* has previously uploaded `Doc` files to *sp*; (b) *sr* has not uploaded docs to *sp*, but has uploaded other files (*e.g.,* `PDF`) previously; (c) *sr* has not uploaded or interacted with *sp* at all previously, but has uploaded files to other file servers in the gird; (d) *sr* has not previously interacted with any server in the grid. The four cases are elaborated in the next paragraphs.

[3]The value of $I.T.$ is chosen arbitrarily to describe the system and the its operations based on the proposed algorithm. However, in real world applications, the value of $I.T.$ should be chosen based on analysis and careful judgement.

Case (a): Known For The Same Service

This case is elaborated in Figure 4.3 (a). Upon the request for service *s* from *sr*, *sp* finds that *sr* has previously interacted with *sp* (1, 1.1). Moreover, *sr* has uploaded `Doc` files to *sp* (1.2). *sp* then makes queries for recommendations and gets a reply from a recommender (r) (1.3, 1.4), where 0.8 is the recommendation value from r, and 2 is the recommendation path-length. Therefore, this a direct recommendation. The direct trust of *sp* on *sr* for *s* is 0.8 after applying time-based ageing on the direct trust retrieved from the `Direct Trust` repository (1.5). The recommendation accuracy of r to *sp* is 1 for *s* and *sr* (1.6). *sp* uses its self-confidence $\alpha = 0.8$, and calculates total trust T using Equation (4.20), which is 0.8 (1.7). Since this value is greater than $I.T.$, the service request is accepted (1.8).

Case (b): Known But Not For The Same Service

This case is presented in Figure 4.3 (b) and is similar to Case (a), except that *sp* has no previous interaction record for *s* with *sr* (2.1, 2,2). However, *sp* finds that *sr* has uploaded `PDF` files previously. Applying Equation (4.5), *sp* obtains a similar direct trust value for *s* and applies time-based ageing on it (2.3). *sp* obtains a direct recommendation 0.7 from r with recommendation accuracy 0.9 (2.4, 2.5, 2.6). *sp* uses $\alpha = 0.8$ and calculates T using Equation (4.20) getting a value of 0.438 which is less than $I.T.$; therefore, the request is rejected (2.7, 2.8).

Case (c): Unknown But Recommendations Available

Figure 4.3 (c) elaborates this case, where *sr* is unknown to *sp* (3.1), but *sp* finds an indirect recommendation of 0.9 from r, where the visited path-length is 3 with accuracy 0.96 for s (3.2, 3.3, 3.4, 3.5). sp uses the maximum allowed recommendation path $\Lambda = 5$, with $\Psi = 0.2$. Since sr is unknown to sp, it uses a low self-confidence value, $\alpha = 0.3$ in its total trust calculation. The total trust of sr thus becomes 0.58 which is slightly greater than $I.T.$; therefore, the request is granted (3.6, 3.7).

Case (d): Unknown And Recommendations Not Available

This case is presented in Figure 4.3 (d), and is almost similar to Case (c), but sp does not receive any recommendation for sr about s (4.4). Therefore, sp concludes that sr is a completely new client to the whole system. sp assigns a neutral trust value of 0.5, does not examine $I.T.$, and grants the service request (4.5, 4.6).

4.4 Summary

In this chapter, we present a trust model called CAT (Context-Aware Trust) for calculating the trustworthiness of requestors in service-based software. CAT works in conjunction with trust rules to analyze service-based interactions from trust perspectives. Direct trust is obtained from this analysis. A time-based ageing and a context-similarity parameter is introduced in the direct trust calculation. Direct and indirect recommendations are collected and compared against direct trust to measure accuracy which helps in differentiating between reliable and unreliable recommendations. The total trust is calculated considering both the direct trust and the total recommendation trust. A trust-based service granting algorithm is presented which uses CAT to determine whether to grant a service to a requestor. The trust model and the algorithm are elaborated using examples from a file sharing grid. The analysis of service-based interactions, and the calculation of direct trust are facilitated by employing a trust monitoring architecture, discussed in the next chapter. The implementation and experimental evaluation of the trust model in conjunction with the trust monitor is provided in Chapter 6 and Chapter 7 respectively.

Chapter 5

A Trust Monitoring Approach for Service-Based Software

In this chapter, we present a trust monitoring architecture and its operations for service-based software. The trust monitor is assumed to reside in each service provider entity to monitor service-based interactions and make trust-based decisions at run-time. We provide an overview of the monitoring approach in Section 5.1 and then present the monitoring architecture in Section 5.2. A trust monitoring algorithm is provided in Section 5.3, which is used to analyze interactions, calculate and update the trustworthiness of requestors and accuracy of recommenders, and make trust-based automatic decisions. Section 5.4 illustrates the monitoring approach using examples from a file sharing grid.

5.1 Trust Monitoring Overview

The monitoring of trust is performed by providers to analyze service-based interactions with requestors. A provider interacts with requestors to provide services and with other providers in the system to exchange recommendations. Entities exchange recommendations to convey their trust of requestors. The providers analyze interactions with requestors based on trust rules and calculate the trustworthiness of the requestors based on CAT. Direct trust is calculated from this analysis, and alerts are reported if violations of trust rules are detected in the interaction. Based on the analysis of interactions, service-based dynamic decisions are obtained at run-time.

Figure 5.1 provides an overview of the trust monitoring approach, where the target system is any provider. An interaction is initiated when a requestor requests a service. Upon granting the service to the requestor, a service session is initiated between the provider and the requestor. The events received from requestors by the Main Module of a provider are called $InteractionInputs$. The provider contains a Trust Monitor to analyze interactions with requestors which are forwarded to it by the Main Module. For a service request, the Trust Monitor provides a *ServiceDecision* on whether to grant the service or not. Upon the granting of services, the Trust Monitor analyzes interaction events related to the corresponding session with the requestor based on trust scenario specifications used as trust rules at run-time. Based on this analysis, the trust monitor provides another *ServiceDecision* specifying whether the interaction is successful or not. The Main Module sends replies in the form of $InteractionOutputs$ to the requestors according to the *ServiceDecision*. The requestor is penalized with a distrust value if any trust rule is violated in one of the interaction events, while it is awarded a trust value if no such violation occurs. This facilitates the calculation of the

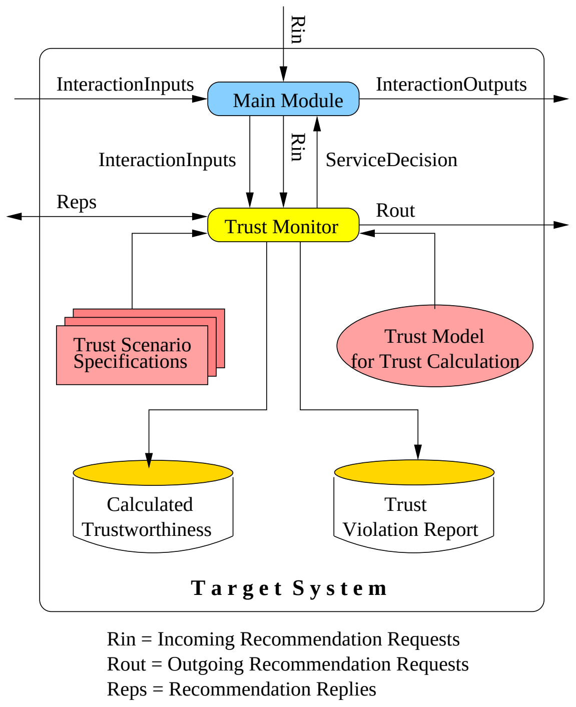

Rin
InteractionInputs
Main Module
InteractionOutputs
InteractionInputs
Rin
ServiceDecision
Reps
Trust Monitor
Rout
Trust Scenario Specifications
Trust Model for Trust Calculation
Calculated Trustworthiness
Trust Violation Report
Target System
Rin = Incoming Recommendation Requests
Rout = Outgoing Recommendation Requests
Reps = Recommendation Replies

Figure 5.1: Overview of the trust monitoring approach

trustworthiness of requestors based on trust calculation schemes from our trust model, CAT. The Main Module receives incoming recommendation requests (*Rin*) from other providers and forwards those to the Trust Monitor which can send recommendation requests to others through *Rout*. Moreover, The Trust Monitor receives or sends recommendation replies through *Reps*. The accuracies of incoming recommendations are obtained from the analysis of interactions. All the calculated trust values are stored in the repositories of the Trust Monitor. Moreover, alert reports are generated and logged if any violation of a trust rule is detected.

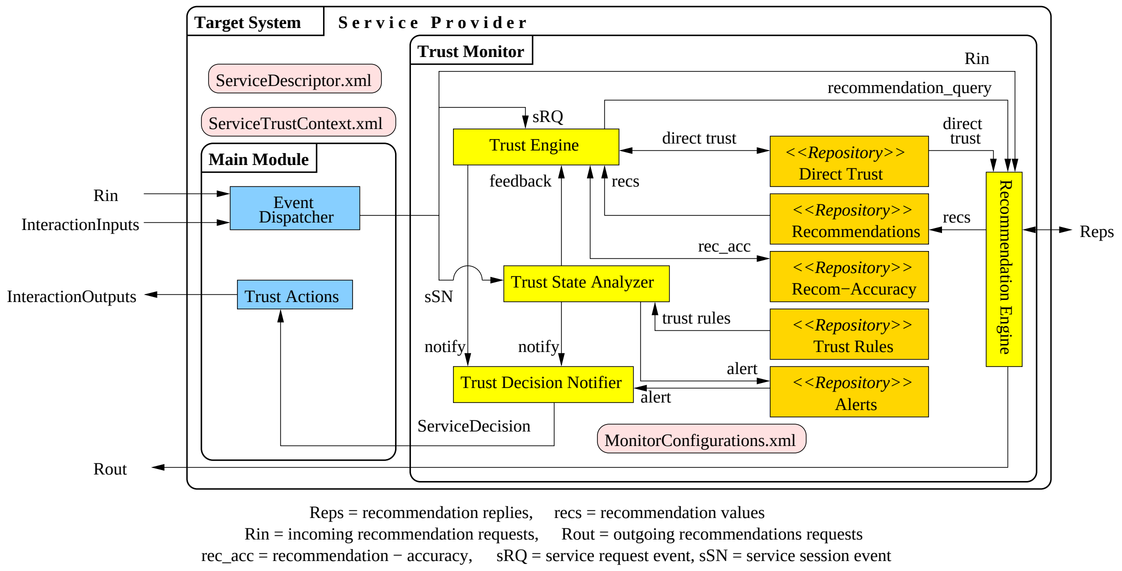


Figure 5.2: The trust monitoring architecture in service providing software

5.2 A Trust Monitoring Architecture

The trust monitoring architecture is composed of a number of modules to analyze and calculate trustworthiness and make trust-based run-time decisions. The architecture is presented in Figure 5.2. $InteractionInputs$ are service request events (sRQ) or service session events (sSN). Upon the granting of a service to a requestor by a provider, a service session is initiated, during which the requestor and the provider exchange information related to the granted service. The events related to the session are called service session events (sSN). The Event Dispatcher of the Main Module takes the sRQ and the sSN as primary inputs. A provider can make a recommendation request (rRQ) to other providers about the requestor and receive recommendation replies (rRP) from other providers. The secondary inputs to the Event Dispatcher are the recommendation requests from other service providers through Rin.

The sRQs are forwarded to the Trust Engine, and sSNs to the Trust State

Analyzer of the Trust Monitor. The *Rin*s are forwarded to the Recommendation Engine, from where *rRQ*s are sent as *Rout* to other providers. The replies to recommendation requests (*rRP*) are received and sent by the Recommendation Engine through *Reps*. The Trust Engine provides decisions on *sRQ*s, the Trust State Analyzer checks *sSN*s against possible trust rules and provides decisions based on the analysis. The Trust Decision Notifier forwards decisions from the Trust Engine and the Trust State Analyzer to the Trust Actions of the Main Module, from which replies to requestors are provided as *sRP*s through *InteractionOutputs*. Every provider has a `ServiceDescriptor.xml` file to describe provided services and a `ServiceTrustContext.xml` file to designate corresponding trust rules.

A snippet of `ServiceDescriptor.xml` is provided in Figure 5.3 (see Appendix A.1 for the complete file). The target system is a provider with ID sp_1, offering file sharing services (*i.e.,* file upload, search, open, and download) to requestors. One of the provided services is `UploadDocFile` to upload documents to the server. The required parameters (*i.e.,* service-param) are `fileName`, `fileSize`, `fileType`, and `fileContents`. sp_1 employs constraints on this service which requestors need to follow while uploading documents. The constraints are specified in the ServiceConstraints tag, such as `fileSize.maxPOST` which limits the maximum file size in the server, lest malicious users upload files of very large size to waste server space, possibly making the upload service unavailable to other users. The trust rules follow the ServiceConstraints.

A snippet of `ServiceTrustContext.xml` is presented in Figure 5.4 (see Appendix A.2 for the complete file). The interaction threshold is used to denote the minimum trust value necessary for a requestor to be offered the service; in this case,

```
<TargetSystem name = "ServiceProvider" id = "sp1">
  <ProvidedServices>
    <ProvidedService provides = "UploadDocFile"
      service-params = "fileName, fileSize, fileType, fileContents">
    </ProvidedService>
    ...
  </ProvidedServices>
  <ServiceConstraints>
    <ServiceConstraint service = "UploadDocFile" fileSize.maxPOST = "100MB">
    </ServiceConstraint>
    ...
  </ServiceConstraints>
</TargetSystem>
```

Figure 5.3: A snippet of `ServiceDescriptor.xml`

if the requestor has a total trust value greater than or equal to 0.52, the service will be granted to the service requestor and the requestor will be able to upload doc files the server[1]. The trust rules to analyze the `UploadDocFile` service are `FileExcess` and `UploadCompletion`. The `FileExcess` trust rule checks whether the uploaded file meets the server maximum file size constraint (*i.e.,* `fileSize.maxPOST`). If this trust rule is violated, the requestor is penalized by a disbelief value of medium as denoted by the category and importance respectively. The `UploadCompletion` trust rule checks for the successful completion of the service. This trust rule is not violated if requestors upload files maintaining all the service constraint; that is, if the `FileExcess` trust rule is not violated. If this trust rule is not violated, the requestor is awarded a belief value of high as denoted by the category and importance respectively.

The Main Module has two parts: Event Dispatcher and Trust Actions. All the incoming events are received by the Event Dispatcher and forwarded to different modules of the Trust Monitor. The incoming events to the Event Dispatcher are of

[1]We have chosen the threshold arbitrarily to describe the monitor. However, in real world applications, the interaction threshold should be chosen carefully based on experience. For example, for a highly critical service, the interaction threshold may be chosen high, while for a less critical system it can be low.

```
<ServiceTrustContexts>
    <Service name = "UploadDocFile" interaction-threshold= "0.52">
        <trust-rules>
            <trust-rule name= "FileExcess" importance= "MEDIUM" category= "disbelief">
            </trust-rule>
            <trust-rule name= "UploadCompletion" importance= "HIGH" category= "belief">
            </trust-rule>
        </trust-rules>
    </Service>
    ...
</ServiceTrustContexts>
```

Figure 5.4: A snippet of `ServiceTrustContext.xml`

three types: service requests (sRQ), service sessions (sSN), and recommendation requests (rRQ). Upon receipt of an event, the module delegates an sRQ to the Trust Engine, an sSN to the trust state analyzer, and an rRQ to the Recommendation Engine of the Trust Monitor. The Trust Actions module provides service replies to requestors according to the $ServiceDecision$s obtained from the Trust Decision Notifier of the Trust Monitor. For example, based on an sRQ, the module can offer or reject services, or based on an sSN, the module can terminate an unsatisfactory interaction.

The Trust Monitor analyzes interactions, calculates trustworthiness and makes decisions. It has four basic submodules: the Trust State Analyzer, the Trust Engine, the Recommendation Engine, and the Trust Decision Notifier. The modules are discussed in the following subsections.

5.2.1 Trust State Analyzer

This module analyzes service session events (sSNs) using trust rules from the `Trust Rules` repository. Upon the arrival of an sSN event, this module checks the event outcome against all possible trust rules. Based on the result of the check, the module notifies the Trust Engine about the confidence (μ) it has gained from the interaction, calculated using Equations (4.1), (4.2), (4.3). Whenever a trust rule is violated, the Trust State Analyzer generates an alert in the `Alerts` repository. An alert has the form $\{sr, s, r, sID, t_{alert}\}$, where sr is the requestor, s is the requested service, r is the trust rule which is violated, sID is the ID of the session in which the violation is detected, and t_{alert} is the detection time. If a trust rule is violated the corresponding interaction is determined as unsatisfactory; otherwise, it is considered as satisfactory. The Trust Decision Notifier is notified of this interaction status.

5.2.2 Trust Engine

This module performs two tasks. First, based on the feedback on confidence (μ) from the Trust State Analyzer, it calculates and updates direct trust and the corresponding recommendation accuracies. Second, it calculates total trust using direct trust, recommendations and recommendation accuracies to provide decisions on service requests (sRQ).

Whenever the Trust State Analyzer provides confidence from an interaction, direct trust is calculated, updated, and stored in the `Direct Trust` repository. The previous direct trust value is retrieved from the repository, updated based on the new confidence value and then stored into the repository. The direct trust T_D from an interaction is calculated using Equation (4.4) (recall the first option of direct trust calculation in Section 4.1.2) The value of T_D thus changes after each interaction based on the outcome of the interaction. After the calculation of T_D from the corresponding interaction, the Trust Engine compares T_D with corresponding recommendations (*i.e.,* the recommendations from other providers on the requestor about the requested service). This comparison results in a recommendation accuracy according to Equations (4.13) and (4.14). The Trust Engine then retrieves the previous recommendation accuracy of each recommender from its `Recom-Accuracy` repository, updates the recommendation accuracies using Equation (4.15), and then stores the new values in the repository.

The Trust Engine makes the trust-based service granting decision for an sRQ by calculating the total trust value. The service is granted if the total trust is at least equal to the interaction threshold of the requested service, otherwise the request is rejected (recall the interaction threshold in Figure 5.4 and Section 4.2). The Trust Decision Notifier is informed of this decision. Upon the arrival of an sRQ from a requestor, the Trust Engine asks the Recommendation Engine to query for new recommendations. Recommendations from unreliable recommenders are discarded in the total trust calculation. The recommendation trust on the requestor about the requested service is calculated using Equations (4.16), (4.17), (4.18). The total trust is calculated using Equation (4.20). The Trust Engine follows the trust-based service granting algorithm (Algorithm 1, Chapter 4) for this purpose.

5.2.3 Recommendation Engine

All the incoming recommendation requests (rRQ) are delegated to this module. This module provides an rRP in response to an rRQ using Equation (4.10), receives all rRPs from other providers, and sends rRQs to other providers. The recommendation values from incoming rRPs are stored in the `Recommendations` repository. If a recommendation has path-length more than 2, the recommendation is considered as indirect and calculated using Equations (4.11) and (4.12) (recall Section 4.1.5).

```
<RecommenderList>
    <Recommenders>
        <Recommender-id>sp2</Recommender-id>
        ...
    </Recommenders>
</RecommenderList>
<Constants>
    <EquationConstants>
        <EquationConstant = "Confidence" wb = "0.8" wd="0.2"></EquationConstant>
        ...
    </EquationConstants>
</Constants>
```

Figure 5.5: A snippet of `MonitorConfigurations.xml`

5.2.4 Trust Decision Notifier

This module provides the Trust Actions module the *ServiceDecicion* it obtains from the Trust Engine and Trust State Analyzer. A *ServiceDecision* is constructed as $\{sr, s, Accept, t\}$, if the request for service s from requestor sr is accepted at time t, or as $\{sr, s, Reject, t\}$ if it is rejected. A *ServiceDecision* is constructed as $\{sr, s, Unsatisfactory, FileExcess, sID, t\}$ if the sr makes the interaction unsatisfactory by violating the `FileExcess` trust rule in the session sID at t, or as $\{sr, s, Satisfactory, sID, t\}$ if the interaction was successful without violating any trust rules.

The `MonitorConfigurations.xml` file is used to denote the list of recommenders and to denote the constant values used in the trust equations. A snippet of the `MonitorConfigurations.xml` is provided in Figure 5.5 (see Appendix A.3 for the complete file). The RecommenderList tag shows the list of recommenders to whom sp_1 asks for recommendations, such as sp_2. The Constants tag includes the constant values used in the trust calculations[2]. For example, the value of w_b in Equation (4.3)

[2]The values of different constants used in the thesis are merely for describing the approach. These values can vary depending on situations.

is 0.8.

5.3 A Trust Monitoring Algorithm

Algorithm 2 is used to analyze interactions, calculate trust, and make decisions on service session events (sSN). The inputs are trust rules and an sSN. Based on the analysis of the sSN, the trust values are updated, and if a trust rule is violated by the sSN, an alert is generated. The algorithm operates in four phases: initialization of variables, analysis of sSN, update in trust values, and notifications based on the analysis. The four phases are described in the next paragraphs.

a. Initializations

The trust rules are loaded in the `TRStore` from the `Trust Rules` repository (Lines 1-3). The service requestor ID (rID), service name (s), service parameters (sp), session ID (sID), and session time (t) are retrieved from the sSN (Line 4). The service constraints (sc) for s are loaded from the `ServiceDescriptor.xml` (Line 5), and the possible trust rule names (TN) for s are loaded from the `ServiceTrustContext.xml` (Line 6). b and d calculate the total belief and total disbelief obtained from an sSN respectively. n_b and n_d count the total number of trust rules with belief and disbelief outcomes respectively. The `TrustStatus` denotes whether any trust is violated in the sSN. Before analyzing the event, b, d, n_b, and n_d are initialized to 0, and the `TrustStatus` is initialized to $Satisfactory$ (Line 7).

```
input  : A set of trust rules ($R$), ServiceSession event ($sSN$).
output: Update in trust values after each interaction. If a trust-rule is violated, an alert ($a$) is generated.
foreach trust-rules $r \epsilon R$ in the Trust Rules repository do
    TRStore:= TRStore $\cup r$ ;
end
Get serviceRequestor $rID$, serviceName $s$, serviceParams $sp$, session id $sID$, sessionTime $t$ from $sSN$ ;
Load serviceConstraints $sc$ based on $s$ from ServiceDescriptor.xml;
Load possible trust-rule names $TN$ based on $s$ from ServiceTrustContext.xml;
$b := 0$, $d := 0$, $n_b := 0$, $n_d := 0$, TrustStatus:= $Satisfactory$;
for $i = 0; i < TN.length; i++$ do
    if $TN[i].category$ = "disbelief" then
        $n_d = n_d + 1$;
        if $TN[i]$.isViolated() $= true$ then
            Generate alert $a$ using $rID$, $TN[i]$, $s$, $sID$, and $t$ ;
            Log alert $a$ in the Alerts repository;
            TrustStatus:= $Unsatisfactory$;
            $d := d + (disbelief)TN[i].importance$;
        end
    end
    else if $TN[i].category$ = "belief" then
        $n_b = n_b + 1$;
        if $TN[i]$.isViolated() $= false$ then
            $b := b + (belief)TN[i].importance$;
        end
    end
    if $i = TN.length - 1$ then
        Calculate confidence $\mu$ using $b$, $d$, $n_b$, and $n_d$;
        NotifyTrustEngine:= $true$;
        TrustDecisionNotifier:= $true$;
    end
end
if NotifyTrustEngine $= true$ then
    Get previous direct trust from the Direct Trust repository;
    Update direct trust for $rID$ on $s$ at time $t$ ;
    Store direct trust in the Direct Trust repository;
    Get all previous recommendations related to $rID$ and $s$ from the Recommendations repository;
    Calculate recommendation accuracy for $rID$ on $s$ at time $t$ ;
    Get all previous recommendation-accuracy related to $rID$ and $s$ from Recom-Accuracy repository;
    Update recommendation-accuracy for $rID$ on $s$ at time $t$ ;
    Store recommendation-accuracy in the Recom-Accuracy repository;
end
if TrustDecisionNotifier $= true$ then
    if TrustStatus $= Satisfactory$ then
        Construct ServiceDecision using TrustStatus and $rID$, $s$, $t$ ;
    else
        Construct ServiceDecision using TrustStatus and $a$ ;
    end
    Notify Trust Actions of the ServiceDecision ;
end
function boolean isViolated()
if $sp$ in $sSN$ violate the $sc$ related to $TN[i]$ then Return $true$;
else Return $false$;
```

Algorithm 2: Interaction-based trust monitoring

b. Trust-Based Event Analysis

The analysis of sSN takes place in the Trust State Analyzer module. sSN is checked against the trust rules TN to analyze the trustworthiness of rID on s (Lines 8-29). A trust rule in TN is denoted as $TN[i]$, where i is the index of the trust rule.

If a trust rule $TN[i]$ has a disbelief category, n_d is increased by 1, and the sSN is checked against $TN[i]$ to see whether the sSN violates it (Line 9-11); if it does, an alert a is generated and logged into the `Alerts` repository (Lines 12-13). Moreover, `Trust Status` is updated to $Unsatisfactory$ since a trust violation has been detected (Line 14). d is updated by using the importance of $TN[i]$ (*i.e.,* `HIGH`, `MEDIUM` or `LOW`) (Line 15). For example, if $TN[i]$ has the importance `MEDIUM` and the category is disbelief, then the corresponding trust value added to d is 0.2 (since medium disbelief has the value 0.2 according to Figure 4.1). If $TN[i]$ has a belief category, the sSN is checked against it with an increase in n_b (Lines 18-23). If the sSN does not violate $TN[i]$, b is updated accordingly.

If all the trust rules in TN have been checked, the confidence (μ) is calculated using Equation (4.3) (Lines 24-25). Then the Trust Engine is notified by providing μ (Line 26). Moreover, the Trust Decision Notifier is notified by sending the `Trust Status` (Line 27).

c. Trust Value Update

The trust values are updated in the Trust Engine upon the receipt of the confidence value (Lines 30-39). The previous direct trust is retrieved from the `Direct Trust` repository (Lines 31), and T_D is calculated using Equation (4.4) and stored in the `Direct Trust` repository (Lines 32-33). All the recommendations related to s

and rID are retrieved from the `Recommendations` repository (Line 34), any recommendation accuracy is calculated using Equations (4.13), (4.14) and updated using Equation (4.15) (Lines 35-37), and the accuracy is stored in the `Recom-Accuracy` repository (Line 38).

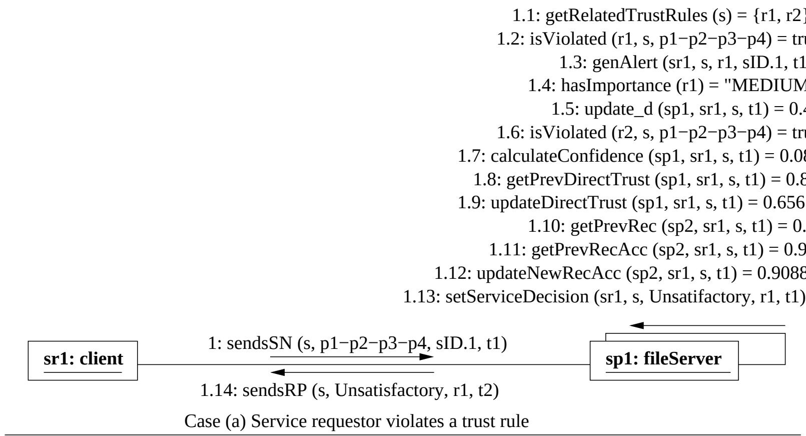
d. Trust Decision Notification

A `ServiceDecision` is constructed by the Trust Decision Notifier based on the information it gets (Lines 40-47). The `ServiceDecision` is constructed using `TrustStatus` and either rID, s and the session time t (Lines 41-42), or the corresponding alert from the `Alerts` repository (Lines 43-44), depending on whether any trust rule is violated. The Trust Actions module is notified of the `ServiceDecision` (Line 46).

The function `isViolated()` checks whether the sSN violates the trust-rule $TN[i]$ (Lines 48-50). The service parameters (sp) are checked against the service constraints (sc) related to $TN[i]$. If the sp do not match the sc, the sSN has violated $TN[i]$, and the function returns true (Line 49). Otherwise, it returns false, since the sSN has not violated $TN[i]$ (Line 50).

5.4 An Illustrative Example

In this section, we illustrate the trust monitoring approach by adapting file sharing applications of Section 3.3 to a file storage grid of Section 4.3. Recall that, in a file storage grid structure [46,67–70] there are a number of servers (*i.e.,* service providers) to provide file sharing services, and the users (*i.e.,* service requestors) share resources with each other by using the services provided. Such a file sharing grid can be useful


1.1: getRelatedTrustRules (s) = {r1, r2}
1.2: isViolated (r1, s, p1−p2−p3−p4) = true
1.3: genAlert (sr1, s, r1, sID.1, t1)
1.4: hasImportance (r1) = "MEDIUM"
1.5: update_d (sp1, sr1, s, t1) = 0.4
1.6: isViolated (r2, s, p1−p2−p3−p4) = true
1.7: calculateConfidence (sp1, sr1, s, t1) = 0.08
1.8: getPrevDirectTrust (sp1, sr1, s, t1) = 0.8
1.9: updateDirectTrust (sp1, sr1, s, t1) = 0.656
1.10: getPrevRec (sp2, sr1, s, t1) = 0.6
1.11: getPrevRecAcc (sp2, sr1, s, t1) = 0.9
1.12: updateNewRecAcc (sp2, sr1, s, t1) = 0.9088
1.13: setServiceDecision (sr1, s, Unsatifactory, r1, t1)
sr1: client
1: sendsSN (s, p1−p2−p3−p4, sID.1, t1)
sp1: fileServer
1.14: sendsRP (s, Unsatisfactory, r1, t2)
Case (a) Service requestor violates a trust rule

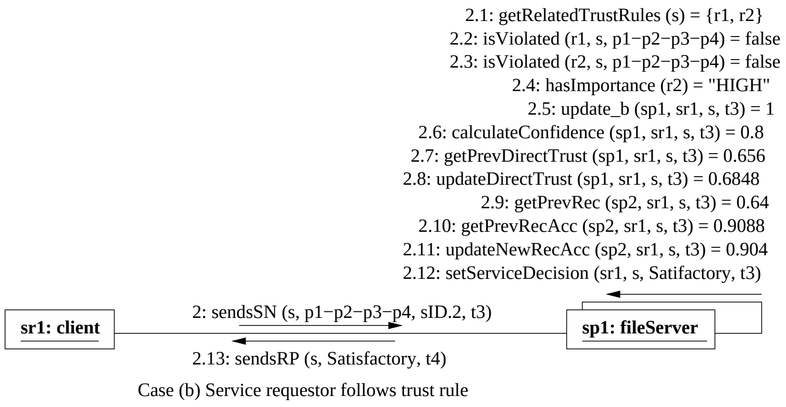

2.1: getRelatedTrustRules (s) = {r1, r2}
2.2: isViolated (r1, s, p1−p2−p3−p4) = false
2.3: isViolated (r2, s, p1−p2−p3−p4) = false
2.4: hasImportance (r2) = "HIGH"
2.5: update_b (sp1, sr1, s, t3) = 1
2.6: calculateConfidence (sp1, sr1, s, t3) = 0.8
2.7: getPrevDirectTrust (sp1, sr1, s, t3) = 0.656
2.8: updateDirectTrust (sp1, sr1, s, t3) = 0.6848
2.9: getPrevRec (sp2, sr1, s, t3) = 0.64
2.10: getPrevRecAcc (sp2, sr1, s, t3) = 0.9088
2.11: updateNewRecAcc (sp2, sr1, s, t3) = 0.904
2.12: setServiceDecision (sr1, s, Satifactory, t3)
sr1: client
2: sendsSN (s, p1−p2−p3−p4, sID.2, t3)
sp1: fileServer
2.13: sendsRP (s, Satisfactory, t4)
Case (b) Service requestor follows trust rule

Figure 5.6: Interaction diagrams elaborating different cases of interaction-based trust monitoring

for collaborative groupware and scientific projects [66]. While the servers provide services to the users, they also need to protect themselves from potentially untrustworthy users who may become malicious while uploading files. The servers need to analyze the service-based interactions with the users based on trust perspectives, determine the trustworthiness of requestors based on the analysis, and also make run-time decisions [71]. This can be facilitated by incorporating our trust monitoring architecture into each service provider of the file storage grid.

The service provider in our example is sp_1 (recall Figure 5.3), which provides the

`UploadDocFile` service to requestors. An exploitation could take place by uploading files of very large size which are beyond the acceptable limit of the file server. This trust violation can be detected by the `FileExcess` trust rule (recall Figure 5.4). The uploading interaction with a requestor becomes successful if the requestor's uploaded file follows the constraints. This can be detected using the `UploadCompletion` trust rule, which in this case is not violated, unless a `FileExcess` trust violation takes place. The `FileExcess` trust rule is specified using UMLtrust (recall Section 3.3). The trust equations are adopted from our trust model, CAT (recall Chapter 4) as mentioned in Section 5.2. The trust monitoring algorithm is employed by the provider sp_1 to analyze the sSN related to the `UploadDocFile` service. Figure 5.6 presents two different cases of the trust monitoring algorithm based on the file sharing grid with requestor sr_1 and requested service s being `UploadDocFile`. The two cases are: (a) sr_1 violates a trust rule, and (b) sr_1 does not violate any trust rule. The two cases are elaborated in the next paragraphs.

Case (a): Service Requestor Violates A Trust Rule

Upon granting of an sRQ for s (*i.e.,* `UploadDocFile`) to sr_1 from sp_1, the sr_1 sends an sSN in the form `sendsSN(s,p1-p2-p3-p4,sID.1,t1)` (see Figure 5.6 (a)). `p1-p2-p3-p4` denotes the service parameters containing `fname-doc-200-fileContents`. `sID.1` is the session ID, and t_1 is the time of the session. In the service parameters, `fname` is `fileName`, `doc` is the `fileType`, `200` is the uploaded `fileSize` in megabytes (MB), and `fileContents` stores the contents of the uploaded file. This event is received by the Event Dispatcher and then sent to the Trust State Analyzer. The related trust rules are retrieved to analyze the sSN (1.1). According to `ServiceTrustContext.xml`, there are two trust rules for this sSN, where `r1` is `FileExcess`, and `r2` is `UploadCompletion`. `r1` has a disbelief category, while `r2` has a belief category. Then violations of the trust rules are checked, and belief, disbelief, and confidence scores are updated according to the trust monitoring algorithm. In this case, the sSN violates `r1` by sending a file beyond the acceptable limit of the server, and `r2` is violated as the sSN does not complete satisfactorily. At the end, the direct trust is recalculated, and stored in the `Direct Trust` repository (1.2 to 1.9). Moreover, the Trust Engine retrieves the previous recommendations related for sr_1 on s from the `Recommendations` repository (1.10), and recommendation accuracy related to s and sr_1 from the `Recom-Accuracy` repository (1.11). sp_2 is the only recommender in the `Recommendations` repository of sp_1 for sr_1 on s. The new recommendation accuracy of sp_2 for sr_1 on s at time t_1 is calculated, and then stored in the `Recom-Accuracy` repository (1.12). The Trust Decision Notifier is notified of the generated alert for sSN. It retrieves the alert from the `Alerts` repository, constructs a $ServiceDecision$ using the alert and sends it to Trust Actions (1.13) which sends a service reply (sRP) to sr_1 (1.14).

Case (b): Service Requestor Follows Trust Rule

In this case where no violation occurs, an essentially identical set of steps are followed as can be seen by the detailed breakdown in Figure 5.6(b). Note because in 2.2 and 2.3, no rules are violated, no alert is generated as it was in Case (a). The Trust Decision Notifier is informed that the analysis is complete and no alert has been generated, so it constructs the $ServiceDecision$ as $\{sr_1,\ s,\ Satisfactory,\ t_3\}$; the Trust Actions module sends an sRP accordingly.

5.5 Summary

In this chapter, we present a trust monitoring approach for service-based software by describing a trust monitoring architecture which resides in each trust-aware service provider. The trust monitoring approach employs trust rules according to trust scenario specifications and trust equations of a trust model to measure and detect the trustworthiness of requestors in service-based interactions. The incorporation of the trust monitor makes total system trust-aware. We present a trust monitoring algorithm to monitor interactions, calculate the trustworthiness of requestors, and make trust-based dynamic decisions at run-time. The trust monitoring approach is illustrated using examples from a file sharing grid, where providers offer file uploading services to requestors. We have implemented the trust monitoring architecture in a prototype file-storage grid structure, the details of which are provided in the next chapter. In Chapter 7, we provide the experimental evaluation of the trust monitoring approach.

Chapter 6

Prototype Implementation

In this chapter, we provide a brief implementation overview of the prototype service-based system we have developed. The prototype is a service-based grid system, with a number of file sharing servers and clients. An overview of the development environment is provided in Section 6.1. The implementation is performed on Jade platform, introduced in Section 6.2. The implementation details of the trust monitoring architecture are provided in Section 6.3, which incorporates the calculation equations of CAT. Section 6.4 discusses the construction of different events in the prototype.

6.1 Development Environment

A file sharing grid offers file sharing services to its users, where the services are provided by the file sharing servers in the system. A prototype file sharing server takes service request and the related session events from other service requestors, analyze the events automatically at run-time and decide accordingly. The service requestors can choose any service providers in the system, as well as the service providers can also interact with other providers in the system. The summary of the development environment is as follows:

- Operating System: Windows XP. The prototype runs on a Pentium 1.886 GHz Dell machine with 1 GB RAM.

- Development Languages: Java, XML. Java is a standard language for developing multi-threaded distributed environment. XML is widely used to define and configure service-based software.

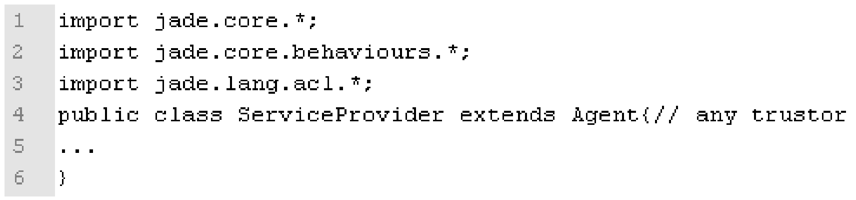
- Development Platform: Jade 3.5 [84], Eclipse IDE 3.2 [85]. Jade is a well known Java-based agent development environment written entirely in Java, where agents are software entities capable of interacting in decentralized systems. Eclipse is an excellent IDE for Java.

- Database: MySQL 5.0 [86]. MySQL is an open source database.

6.2 Jade Overview

Jade [83, 84] provides Java libraries and an execution environment to develop agent-based systems. The agents can be servers or clients or both at the same time. For example, a service provider entity can be developed as an agent by following the code snippet in Figure 6.1, where the first three lines import the necessary Jade libraries. The fourth line declares the 'ServiceProvider' class as an agent, where the agent is the common superclass for all user defined software agents. Similarly, the service requestor entities are also implemented as agents.

Jade has been created to facilitate the development of agent-based systems, where the entities (*i.e.,* the software agents) can interact with each other to achieve the



```
import jade.core.*;
import jade.core.behaviours.*;
import jade.lang.acl.*;
public class ServiceProvider extends Agent{// any trustor
...
}
```

Figure 6.1: Creating a service provider agent in a Jade platform

functionality of the entire system. The entities can provide services to others or can invoke services. In Jade, the entities interact with each other by sending or receiving messages (*i.e.,* events). Jade is fully implemented in Java, and therefore, standard Java libraries can also be used while using Jade platform. Jade is compliant with the IEEE Foundation for Intelligent Physical Agents (FIPA) [59] which develops software standard specifications for agent and agent-based systems. Jade has been developed to assist in the development of complex and large-scale systems so that developers can focus on the development of distributed systems composed of independent entities while avoiding the details of the standard functionality common to such systems. Since our prototype is a file sharing grid consisting of a number of service providers and requestors, we chose Jade as our working platform.

6.3 Implementing the Trust Monitoring Architecture

In this section, we provide details on how different modules of the trust monitoring architecture are implemented in our prototype system. The trust monitor employs the trust rules and the trust model for trust-based analysis and decision making on service-based interactions. The trust rules are stored in the `Trust Rules` repository of the trust monitoring architecture. The details of the trust rules will be discussed in Section 7.2.

Since the services providers and requestors act automatically, the different working modules are implemented in them as special kinds of threads in Java called 'behaviours' in Jade [87]. By extending the basic properties of a behaviour the entity can engage in event creation, sending and receiving, described in the next subsection. The behaviours are the means of executing parallelism in Jade platform.

The Event Dispatcher module is developed by implementing it as a CyclicBehaviour. This behaviour stays alive as long as an entity (*i.e.,* a provider) lives in the system. For example, the Event Dispatcher needs to receive events sent to it by requestors and other providers in the system at any time, which is why it is implemented as a CyclicBehaviour.

All the submodules in the Trust Monitor (*i.e.,* Trust State Analyzer, Trust Engine, Recommendation Engine, Trust Decision Notifier) and Trust Actions in the Main Module are implemented as SimpleBehaviours. This behaviour stays alive until a certain specific condition is satisfied. For example, the Trust State Analyzer needs to be activated when an sSN event takes place. After the analysis of the event, the module notifies the Trust Engine and Trust Decision Notifier and then terminates.

The `Direct Trust`, `Recommendations`, `Recom-Accuracy`, and `Alerts` repositories are implemented as tables in a MySQL 5.0 database. The `Direct Trust` repository has four fields $\{sr, s, T_D, t\}$, where sr is the service requestor which has interacted with the service provider for service s at time t. Based on the interaction analysis, T_D provides the trustworthiness of sr for s at time t. For example, $\{sr_1, UploadDocFile,$ 0.78, 1000$\}$ implies that the sr_1 client interacted with the service provider based on the service $UploadDocFile$ at system time 1000, and the direct trust value after the interaction is 0.78.

The `Recommendations` table has six fields $\{recID, sr, s, rV, M, t\}$, where $recID$ is the ID of the recommender which provided a recommendation on service requestor sr for service s at time t. The recommendation value is rV and the maximum visited path length is M. If M is greater than 2, then the recommendation is indirect.

The `Recom-Accuray` table has five fields $\{recID, sr, s, A, t\}$, where $recID$ is the recommender ID with recommendation accuracy A at time t on service requestor s for service s.

The `Alerts` table has five fields $\{sr, s, rn, sID, t\}$, where sr is the service requestor which violates the trust rule rn related to service s at time t, and sID is the ID of the session of this particular interaction.

Apart from these repositories, each service provider has another repository called `Similar` for trust calculations. This table has three fields $\{s, s_{similar}, sVal\}$, where s is the service which is similar to the service $s_{simialr}$, and the similarity measurement is denoted by $sVal$. The similarity calculation are performed based on the Equation (4.8). Since the context-similarity parameter is reflexive, the first two entries in the `Similar` table (*i.e.,* s and $s_{similar}$) can be used interchangeably. This repository is optional to a service provider, and so we do not include it in the trust monitoring architecture.

The Trust State Analyzer of the trust monitor uses trust rules from the `Trust Rules` repository for monitoring purposes and calculates confidence from the analysis. The calculation of confidence is performed by implementing Equations (4.1), (4.2), and (4.3) in the Trust State Analyzer, which writes trust rule violation reports in the

`Alerts` table. The Trust Engine uses the confidence from the Trust State Analyzer to calculate direct trust and recommendation accuracy. The calculation is performed by retrieving values from the `Direct Trust` and `Recom-Accuracy` tables and then updating them. Moreover, this module calculates total trust to facilitate decision making on service request events (sRQ). The implemented calculation equations in this module are Equations (4.4), (4.5), (4.6), (4.9), (4.13), (4.14), (4.15), (4.16), (4.17), (4.18), (4.20).

The Recommendation Engine receives recommendation requests forwarded by the Event Dispatcher and sends recommendation replies to other providers using Equation (4.10). Based on the instructions from the Trust Engine for new recommendation queries, the Recommendation Engine sends recommendation requests to other service providers by following the recommender list in `MonitorConfigurations.xml` file. Upon the receipt of a recommendation, this module stores the recommendation values to the `Recommendations` repository. Moreover, before storing the recommendations, it calculates the indirect recommendations using Equations (4.11) and (4.12).

The trust monitoring algorithm (Algorithm 2 of Chapter 5) is implemented across the Trust State Analyzer, the Trust Engine, and the Trust Decision Notifier. The trust-based service granting algorithm (Algorithm 1 of Chapter 4) is implemented across the Trust Engine, the Recommendation Engine and Trust Decision Notifier.

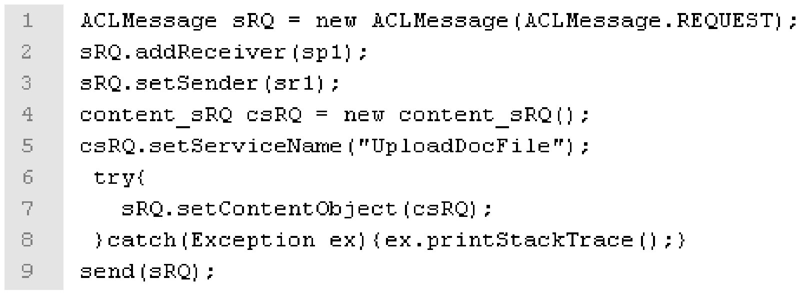
We did not use either Equation (4.7) or Equation (4.19) in our implementation, since the first equation calculates generalized direct trust and the second equation calculates generalized recommendation trust, and we do not use any of the generalized calculations in our trust-based algorithms. Moreover, we calculate the similarity between two services using Equation (4.8) and then store the similarity value in the

`Similariy` table; this calculation is performed off-line.

6.4 Generation of Events

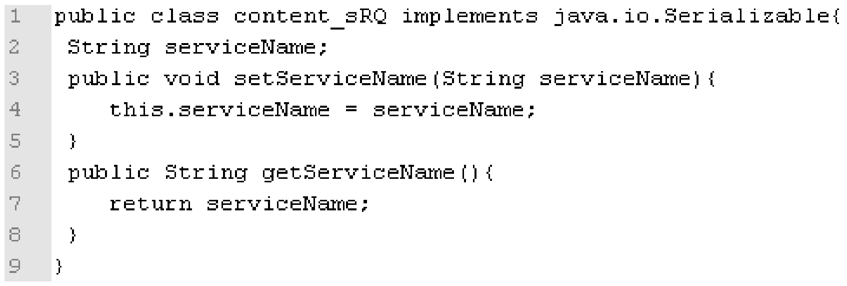
Since interaction is the only way to monitor the trustworthiness of entities, the creation, sending and receiving of events are crucial to our proposed system. This is accomplished through the exchange of messages in the Jade platform, where the messages are called events. The messages are specified in an agent communication language (ACL) [83]. Using ACLMessage in a Jade platform, entities can construct an event. However, the ACLMessage format for different events provides limited fields which sometimes are not sufficient for constructing an event. To add different attributes to the messages, the notion of ontology is used [87]. We discuss the use of ACLMessage and ontology as follows. Let us consider an sRQ event from service requestor sr_1 to service provider sp_1 for service $UploadDocFile$. The event is constructed by following the code snippet given in Figure 6.2, where the message is an `ACLMessage` of type $REQUEST$ (Line 1). An instance of the message is created as sRQ, and the receiver and sender are added to it (Line 2-3). The `content_sRQ` is a specific java class to add additional information to the message (Line 5). This class is considered as ontology for sRQ, as it adds the additional information of service name (*i.e.,* `UploadDocFile`) to the message (Line 6-8). Finally, the message is sent (Line 9).

A code snippet of `content_sRQ` class is provided in Figure 6.3, which implements the Java serialization interface (Line 1). By using the serializable interface, the data encapsulated into the corresponding class can be sent over the network of software entities. The service name is assigned by using the `setServiceName(...)` method



```
1  ACLMessage sRQ = new ACLMessage(ACLMessage.REQUEST);
2  sRQ.addReceiver(sp1);
3  sRQ.setSender(sr1);
4  content_sRQ csRQ = new content_sRQ();
5  csRQ.setServiceName("UploadDocFile");
6   try{
7     sRQ.setContentObject(csRQ);
8   }catch(Exception ex){ex.printStackTrace();}
9  send(sRQ);
```

Figure 6.2: Creation of an *sRQ* event using ACLMessage and Ontology



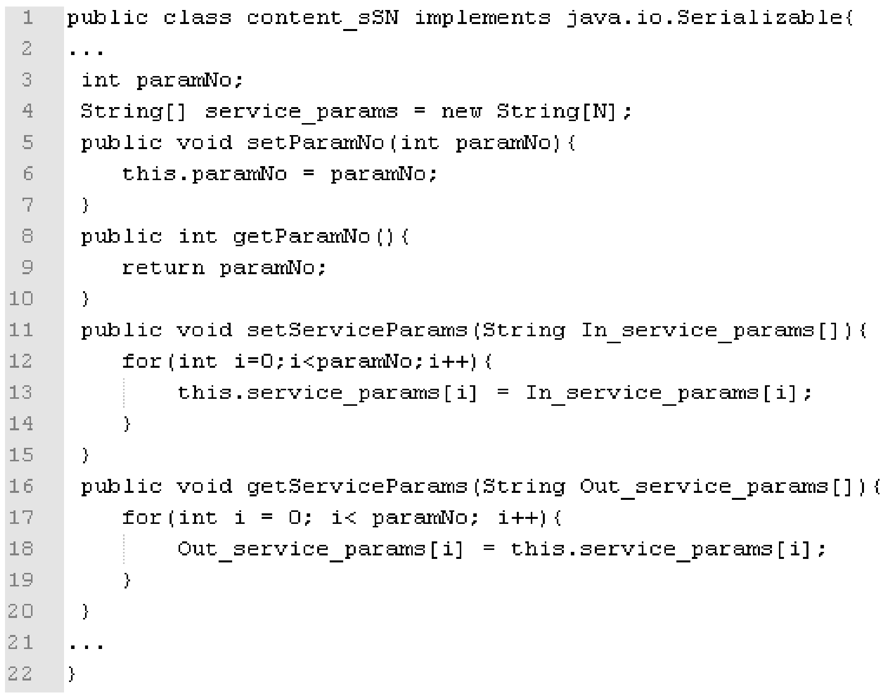
```
1  public class content_sRQ implements java.io.Serializable{
2   String serviceName;
3   public void setServiceName(String serviceName){
4      this.serviceName = serviceName;
5   }
6   public String getServiceName(){
7      return serviceName;
8   }
9  }
```

Figure 6.3: The `content_sRQ` ontology class

(Lines 3-5). At the receiving end, the service provider captures this information by using the `getServiceName()` method (Lines 6-8).

Similarly, we created all other events. For example, the service parameters of the sSN event are captured in its corresponding ontology based on the code snippet shown in Figure 6.4, where the number of parameters are set by the the service requestor using `setParamNo(...)`, which can be retrieved by the service provider using `getParamNo()` (Lines 3-10). The service requestor sets the service parameters using the `setServiceParams` by converting all data to a string format (Lines 11-15). The service provider retrieves the service parameters using the `getServiceParams(...)` method (Lines 16-20).

In our implementation, we use different types of ACLMessage for creating different events. The messages types for different events are presented in Table 6.1, where the

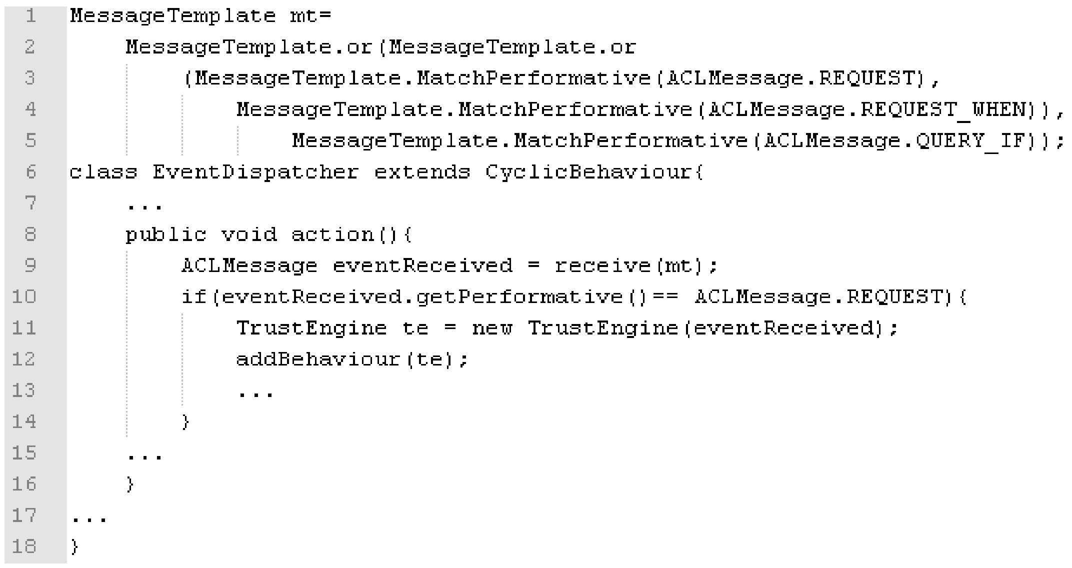


```
public class content_sSN implements java.io.Serializable{
...
 int paramNo;
 String[] service_params = new String[N];
 public void setParamNo(int paramNo){
    this.paramNo = paramNo;
 }
 public int getParamNo(){
    return paramNo;
 }
 public void setServiceParams(String In_service_params[]){
    for(int i=0;i<paramNo;i++){
        this.service_params[i] = In_service_params[i];
    }
 }
 public void getServiceParams(String Out_service_params[]){
    for(int i = 0; i< paramNo; i++){
        Out_service_params[i] = this.service_params[i];
    }
 }
...
}
```

Figure 6.4: The partial code of the ontology class for service session event (sSN)

first column specifies the particular event, the second column lists the ACLMessage type(s) used for the event and the third column describes the event based on the message type.

The sending of events follow the same pattern as described earlier. The events are received in the Event Dispatcher following the code snippets shown in Figure 6.5, where the `MessageTemplate` is used to capture all the different types of events the `Event Dispatcher` can receive (Line 1). The 'or' in the `MessageTemplate` is used to imply that the `EventDispatcher` will accept all incoming messages of types $REQUEST$, $REQUEST\_WHEN$, and $QUERY\_IF$ (Lines 2-5). The EventDispather extends the `CyclicBehaviour` so that it can receive events any time from other entities in the system (Line 8). All the actions related to a behviour are executed in the `action()` method (Lines 8-16). The event (*i.e.,* ACLMessage) is received



```
1   MessageTemplate mt=
2       MessageTemplate.or(MessageTemplate.or
3           (MessageTemplate.MatchPerformative(ACLMessage.REQUEST),
4               MessageTemplate.MatchPerformative(ACLMessage.REQUEST_WHEN)),
5                   MessageTemplate.MatchPerformative(ACLMessage.QUERY_IF));
6   class EventDispatcher extends CyclicBehaviour{
7       ...
8       public void action(){
9           ACLMessage eventReceived = receive(mt);
10          if(eventReceived.getPerformative()== ACLMessage.REQUEST){
11              TrustEngine te = new TrustEngine(eventReceived);
12              addBehaviour(te);
13              ...
14          }
15      ...
16      }
17  ...
18  }
```

Figure 6.5: The receiving of events in the Event Dispatcher module

by `receive(mt)` command, where `mt` contains all the acceptable message types (Line 9). The Event Dispatcher transfers the event to the Trust Engine if the event is a type of request (*i.e., sRQ* event) (Lines 10-12).

The *sRQ* event has the form $\{sr, sp, s, t_{sreq}\}$, where *sr* is the service requestor ID, *sp* is the service provider ID, *s* is the service name, and t_{sreq} is the service request time. The *sSN* event has the form $\{sr, sp, s, service\_params, sID, t_{session}\}$, where *service_params* denotes the parameters for service *s*, *sID* is the ID of the initiated session between the *sr* and the *sp*, and $t_{session}$ is the time of the particular session. The *sRP* event has the form $\{sr, sp, s, service\_status, t_{srep}\}$, where *service_status* is the status of the service requested by *sr* or offered to *sr*, and t_{srep} is the service reply time. The *sp* provides the *service_status* based on the trust-based analysis (*i.e., ServiceDecision*) from the Trust Monitor. An *rRQ* event has the form $\{sp, sr, s, rq, rp, \Lambda, t_{rreq}\}$, where *rq* is the ID of recommendation requestor, *rp* is the ID to whom *rq* requests for the recommendation, Λ is the maximum allowed path length,

Table 6.1: Different ACLMessage types used for different events

Event	**ACLMessage Type**	**Description**
sRQ	$REQUEST$	The service request event generated by a requestor.
sSN	$REQUEST\_WHEN$	The service session event when the service request is granted.
rRQ	$QUERY\_IF$	The query for a recommendation.
sRP	$ACCEPT$, $REFUSE$	The accept message when the service request is granted, and the refuse when rejected.
rRP	$QUERY\_REF$	The reply to a recommendation request.

and t_{rreq} is the recommendation request time. An rRP has the form $\{sp$, sr, s, rq, rp, rv, M, $t_{rrep}\}$, where the rv is the recommendation value provided by rp to rq at time t_{rrep}, and M is the visited path length. Λ and M are needed in rRQ and rRP to check for indirect recommendations.

6.5 Summary

In this chapter, we briefly discuss the implementation of our developed prototype file sharing grid. We describe our development environment, where we implemented and evaluated our system. The Jade platform is used on top of Java to build the prototype, an overview of which is provided. The trust monitoring architecture is implemented following different Jade-specific threads to support parallelism in its execution. Jade is a standard platform for such complex system development. The database of the system is chosen as MySQL because of its widespread applicability. Each of the service providers in the prototype are implemented using their corresponding trust rules based on UMLtrust specifications and trust equations based on CAT. We discuss the incorporation of trust equations with the implementation of the architecture. The entities in our system communicate through the receiving and sending of events. We have shown the construction and structure of the five types events that we have used in our prototype. The trust scenarios specified in UMLtrust are evaluated along with our system, the details of which are provided in the next chapter. Moreover, the evaluation of the trust modeling and monitoring are also elaborated in the next chapter.

Chapter 7

Case Study

In this chapter, we provide details on the case study conducted to evaluate our approach. Our case study has three steps: the specification of trust scenarios using UMLtrust to show its wide range of applicability, the analysis of the trust modeling based on CAT, and the trust monitoring approach in different situations, and the performance overhead of using the trust monitor. We describe the objectives of this case study in Section 7.1. We then specify a number of trust scenarios using UML-trust and generate trust rules from them in Section 7.2. The experimental evaluation of trust modeling and monitoring is presented in Section 7.3. Finally, Section 7.4 provides a performance overhead analysis of using our implementation of the trust monitor.

7.1 Evaluation Objectives

The major objective of this case study is to analyze the applicability of our development framework of Section 3.1 to support the development and automatic monitoring of trust-aware service-based software. We analyze the applicability of UMLtrust to show its wide range applicability. The trust equations, incorporated to a trust-aware provider are evaluated based on different scenarios to show their effectiveness to calculate the trustworthiness of service requestors. The final objective is to see the performance overhead of the prototype thus developed to indicate its real-world adaptation.

To evaluate UMLtrust, we specify a number of trust scenarios from file sharing applications, since our developed prototype is a file sharing grid. We describe, specify, and implement the scenarios, and discuss how we tested those in our system (see Section 7.2).

We analyze the different cases of our two algorithms provided in Section 4.2 (Algorithm 1) and Section 5.3 (Algorithm 2) to show the applicability of our trust model and trust monitoring architecture. We discuss the evaluation environment along with each cases and share the experimental results (see Section 7.3).

We demonstrate the performance of the system while analyzing interactions from trust perspectives and measure the corresponding overheads in Section 7.4.

7.2 Specifying Trust Scenarios using UMLtrust

A file sharing server offers four basic services to its clients: file uploading, searching, opening, and downloading. We analyze the services from trust perspectives, and elicit

Table 7.1: Elicited trust scenarios for a file sharing server [77–79]

Trust scenarios	**Description**
File Excess (see Section 3.3)	Requestors may upload files beyond the limit of the server and thus make the upload service unavailable for others.
File Spamming (see Section 6.1.1)	Requestors may upload illegal and insignificant files to waste storage space on the server.
File Harmful (see Section 6.1.1)	Requestors may upload files containing malicious scripts which can harm other users.
Illegal Access Attempt (see Section 6.1.2)	Requestors may try to access others' personal files in the resource database by manipulating the file search service.
Remote File Inclusion (see Section 6.1.3)	Requestors may manipulate the file open service to open malicious files remotely and to execute them on the server.

trust requirements using the documentation available for file sharing applications [77–79]. File download takes place on the client side, and since we are only considering service provider-side, we use the file upload, search and open related trust scenarios. The selected trust concerns are provided in Table 7.1; the first column provides the names of the trust scenarios, and the second column describes them. The selected trust concerns are the most common concerns in file sharing. The specification of trust scenarios and their implementation as trust rules are performed manually and off-line, while trust rules are used on-line to monitor service-based interactions.

We have already described the first scenario (Trust Scenario 1: File Excess) in Section 3.3. We present the modeling of the rest of the scenarios using UMLtrust and their implementation as trust rules for our developed prototype in the next few subsections. Section 7.2.1 encompasses trust scenarios related to file uploading, while Section 7.2.2 covers a trust scenario for file search, and Section 7.2.3 specifies a trust scenario for the file open service.

7.2.1 Trust Violation in File Upload Service

In this section, we specify the file spamming and file harmful trust scenarios and generate trust rules from the specifications.

File Spamming Trust Scenario

Untrustworthy clients may have malicious intentions while using the file uploading service of the file-server, attempting to spam the server by uploading illegal and invalid files for the purpose of (i) wasting storage space of a server and (ii) clogging the server bandwidth. The client can run a spam script to upload all kinds of files [79]. The server can control this activity by putting certain constraints on file upload service.

Figure 7.1 presents the trust-use-case diagram representing the file spamming scenario. `FileUploader` is the service requestor (*i.e.,* client), and `FileStorage` is the service provider (*i.e.,* server). The `FileStorage` trusts the `FileUploader` by providing it with the file upload service, and the client invokes ($<< uses >>$) it. However, the client can upload unwanted contents ($<< exploit >>$) by using the `UploadSpamFile` activity.

The trust-class diagrams for the file spamming scenario are provided in Figure 7.2, where the `goal` of the $<< trustor >>$ is to ensure `safe-upload`. It requires that the interaction with the $<< trustee >>$ is secure with `min-trust-level` of `LOW`, as spamming of insignificant files may not be the greatest risk to the server. The $<< trustee >>$ attempts to violate the trust of the $<< trustor >>$ by exploiting the file upload service, and can succeed in file spamming when the $<< trustor >>$ does not check the uploaded files.

The $<< trust - concern >>$ is `FileSpam`. File spamming happens when the

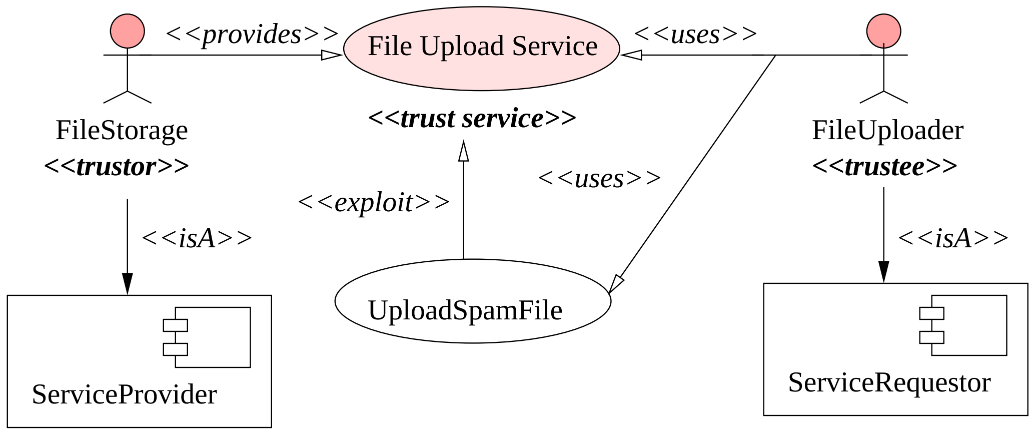


Figure 7.1: The trust-use-case diagram for the file spamming trust scenario

$<< trustee >>$ uploads files that contain unsupported characters in the filenames based on `invalidChars` (*e.g.,* "&, -,#,!, ;"), or are unsupported in their types by the file server according to `validFileTypes` (*e.g.,* for a document file server of doc and txt files, a movie file is a spam). These attributes are defined in the $<< trust - concern >>$ class, where the `level` is `s-to-u` for server to user trust, and `type` is `fileUpload`. The `chkValidFile(...)` function checks the file names. The `supportedFileType(...)` function checks whether the file type meets the server requirements.

Figure 7.3 captures the states necessary to represent the file spamming scenario. In the first state, the client requests uploading (req state). Upon the granting of the request, the client uploads the file ($Upload$ state). Once the file is uploaded, the functions `chkValidFile(...)` and `supportedFileType(...)` are called. The first function returns true if the file is valid, and the second function returns true if the file is supported by the server. Trust is violated if the file is found to be spam (*i.e.,* not valid or not supported), at which point the unsuccessful final state is reached ($I(U)$). The interaction is denoted as successful ($I(S)$) when both functions identify

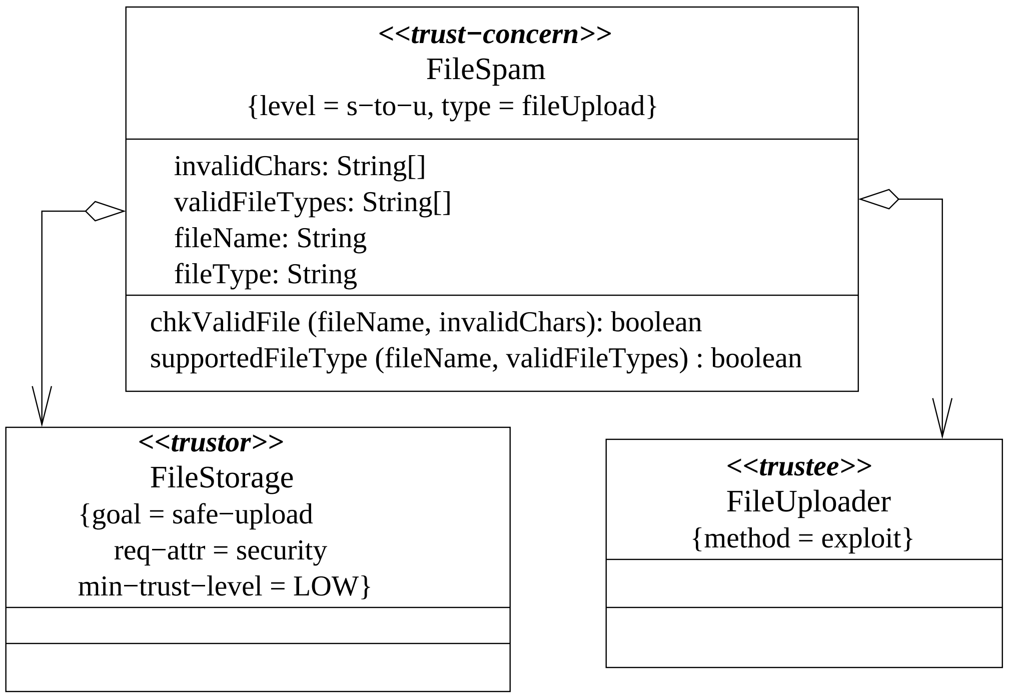


Figure 7.2: The trust-class diagram for the file spamming trust scenario

the uploaded file as valid.

The code generated for the `FileSpam` trust rule is provided in Figure 7.4. We do not show the implementation code for the $<<$ *trustor* $>>$ and $<<$ *trustee* $>>$ since they follow the same pattern of the `FileExcess` trust rule provided in Figure 3.7.

Lines 2 through 7 perform important initializations: `level` and `type` are set based on the specification (Line 2), the rule name is assigned (Line 3), and `invalidChars` and `validFileTypes` are specified (Lines 4-5). The transition from the *req* state has already occurred by the time the trust rule code is entered, so the possible states are defined as *Upload*, *IU*, and *IS* (Line 7). The incoming `fileName` and `fileType` are retrieved during class initialization (Lines 8-10). The `chkValidFile(...)` function looks for any invalid chars in the file name and returns false if it finds any (Lines 11-17), and function `supportedFileType(...)` returns true if the file is of a valid

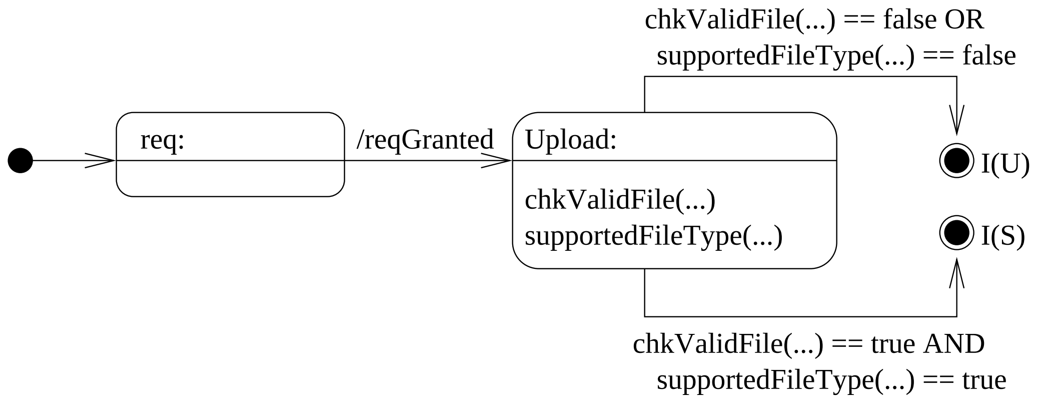

chkValidFile(...) == false OR supportedFileType(...) == false
req:
/reqGranted
Upload:
chkValidFile(...)
supportedFileType(...)
I(U)
I(S)
chkValidFile(...) == true AND supportedFileType(...) == true

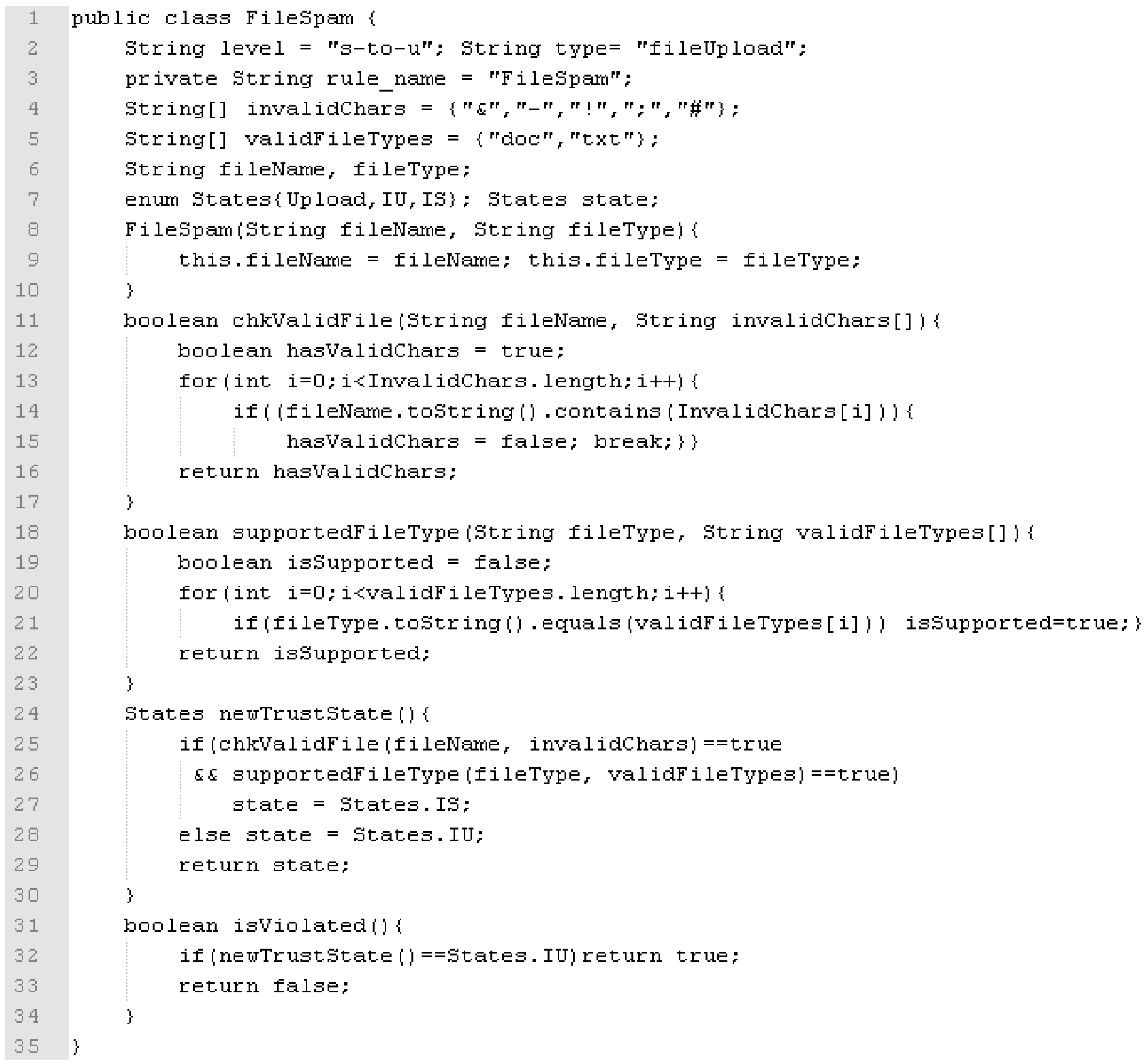
Figure 7.3: The trust-state-machine diagram for the file spamming trust scenario

type (Lines 18-23). `newTrustState()` provides the final state based on the analysis (Lines 24-30), and `isViolated()` returns true if the final state is IU (Lines 31-34).

To test this trust rule, we constructed a service session event (sSN) as $\{sr_1$, sp_1, $UploadDocFile$, $aFilen!a\#me - doc - 100 - fileContents$, sID, $t_{session}\}$, where sr_1 is the client, sp_1 is the file server, $UploadDocFile$ is the service, $aFilen!a\#me-doc-100-fileContents$ contains the service parameters, sID is the service session id, and $t_{session}$ is the session time. The first service parameter (*i.e., $aFilen!a\#me$*) is the file name, and the second one (*i.e., doc*) is the file type. These two parameters are passed to the `FileSpam` trust rule, which returns true as trust is violated in the file name.

File Harmful Trust Scenario

A malicious user can exploit a file uploading service by uploading harmful files to the server, where the file can contain malicious scripts embedded in it. Ideally, the uploaded file should contain no active scripts in the uploaded file, which could harm other users. For example, the uploaded file may contain the script "$<$ *script* $>$ *document.location* $=$ '*http* : //*evil.org/get?info* =' +*document.cookie* $<$ /*script* $>$"



```
public class FileSpam {
    String level = "s-to-u"; String type= "fileUpload";
    private String rule_name = "FileSpam";
    String[] invalidChars = {"&","-","!",";","#"};
    String[] validFileTypes = {"doc","txt"};
    String fileName, fileType;
    enum States{Upload,IU,IS}; States state;
    FileSpam(String fileName, String fileType){
        this.fileName = fileName; this.fileType = fileType;
    }
    boolean chkValidFile(String fileName, String invalidChars[]){
        boolean hasValidChars = true;
        for(int i=0;i<InvalidChars.length;i++){
            if((fileName.toString().contains(InvalidChars[i])){
                hasValidChars = false; break;}}
        return hasValidChars;
    }
    boolean supportedFileType(String fileType, String validFileTypes[]){
        boolean isSupported = false;
        for(int i=0;i<validFileTypes.length;i++){
            if(fileType.toString().equals(validFileTypes[i])) isSupported=true;}
        return isSupported;
    }
    States newTrustState(){
        if(chkValidFile(fileName, invalidChars)==true
         && supportedFileType(fileType, validFileTypes)==true)
            state = States.IS;
        else state = States.IU;
        return state;
    }
    boolean isViolated(){
        if(newTrustState()==States.IU)return true;
        return false;
    }
}
```

Figure 7.4: The `FileSpam` trust rule

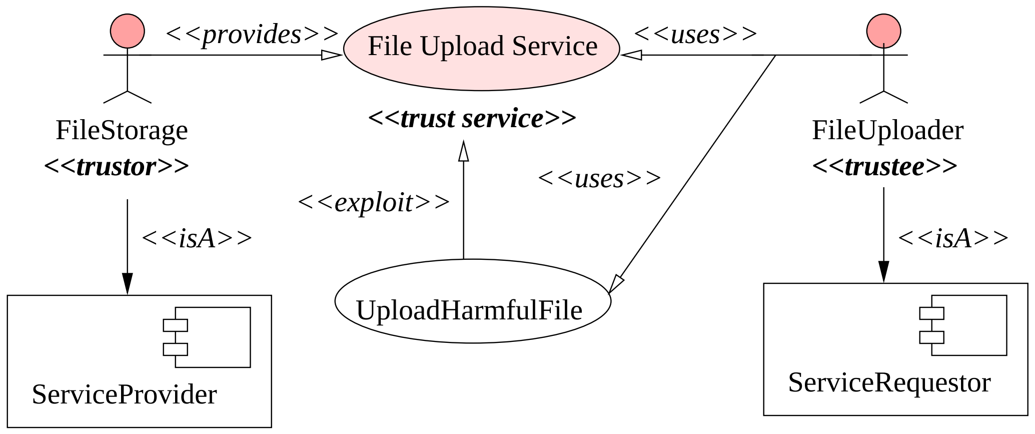


Figure 7.5: The trust-use-case diagram for the file harmful trust scenario

which when executed on another client's machine will be able to send the information stored in the cookie, gathering sensitive information and harming other users. The file server needs to stop the uploading of files containing malicious scripts, normally written in JavaScript. This scenario is particularly harmful as it can execute the popular XSS attack [78] against other users of the server. The simplest way to detect this trust violation is to check whether the file contains any possible scripts (*i.e.*, $< script >$ and $< /script >$).

Figure 7.5 presents the trust-use-case diagram representing the file harmful scenario. `FileUploader` is the service requestor, and `FileStorage` is the service provider. The server trusts a client by providing it the file upload service, and the client invokes it. However, the client can upload harmful files ($<< exploit >>$) by using the `UploadHarmfulFile` activity.

Figure 7.6 provides the trust-class diagram for the file harmful trust scenario. The $<< trustor >>$ has a `goal` of ensuring safe uploading. The `min-trust-level` is `HIGH` since the uploading of this type of file can cause serious harm to other users of the server. The $<< trust - concern >>$ is `FileHarmful` whose `level` is `s-to-u`,

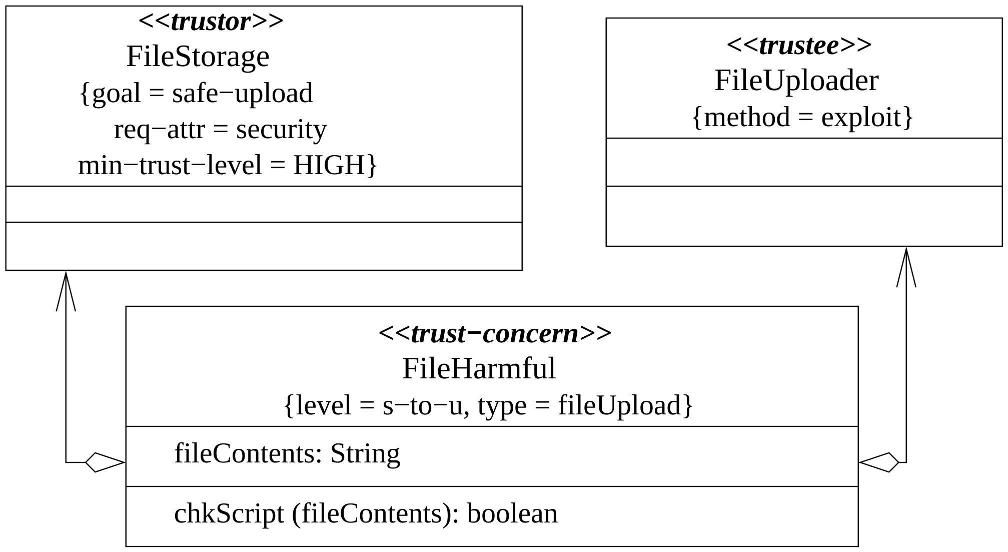


Figure 7.6: The trust-class diagram for the file harmful trust scenario

and `type` is fileUpload. The `chkScript(...)` function checks the `fileContents` for possible scripts.

Figure 7.7 presents the trust-state machine diagram representing the file harmful trust scenario. Upon granting the file upload service, the client uploads the file by sending the file (*i.e., Upload* state), and the file contents are checked by `chkScript(...)` function[1]. The interaction is considered unsatisfactory ($I(U)$) or satisfactory ($I(S)$) depending on whether the file contains a script or not.

The code of the `FileHarmful` trust rule is provided in Figure 7.8; the $<< trust-concern >>$ is the `FileHarmful` class. At first, the `level` and `min-trust-level` are set according to the specification (Line 2) and local variables are declared and initialized (Lines 3-9). The `chkScript(...)` function determines whether the file contents stored in the `fileContents` variable contains a script (Lines 10-16), `newTrustState()`

[1]For the sake of simplicity, we consider that the file contents are directly accessible through the `fileContents` variable. However, in practical applications, the implementation of file contents should differ based on the filetypes used.

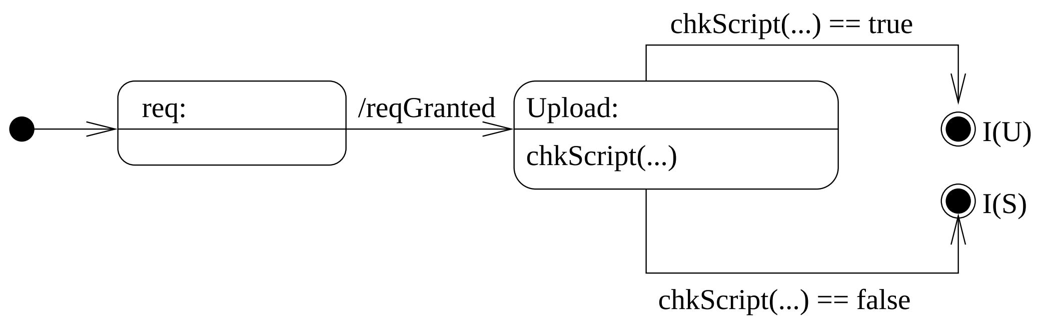
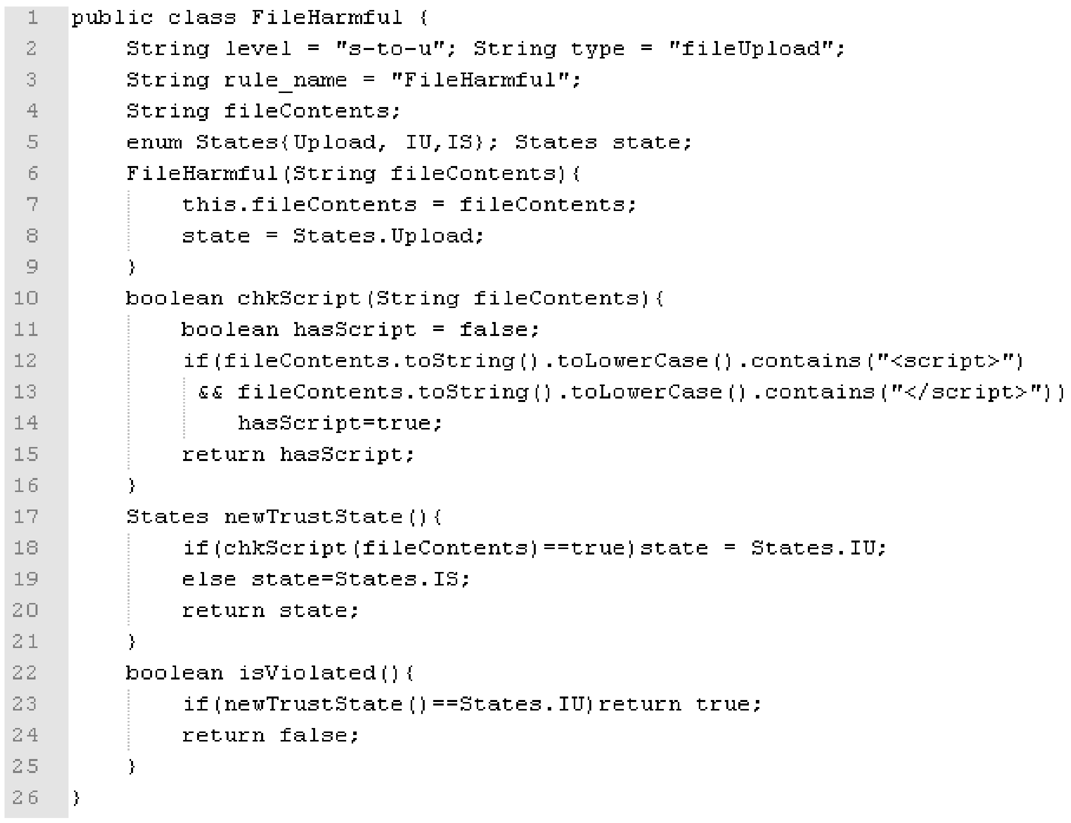


Figure 7.7: The trust-state-machine diagram for the file harmful trust scenario



```
public class FileHarmful {
    String level = "s-to-u"; String type = "fileUpload";
    String rule_name = "FileHarmful";
    String fileContents;
    enum States{Upload, IU,IS}; States state;
    FileHarmful(String fileContents){
        this.fileContents = fileContents;
        state = States.Upload;
    }
    boolean chkScript(String fileContents){
        boolean hasScript = false;
        if(fileContents.toString().toLowerCase().contains("<script>")
         && fileContents.toString().toLowerCase().contains("</script>"))
            hasScript=true;
        return hasScript;
    }
    States newTrustState(){
        if(chkScript(fileContents)==true)state = States.IU;
        else state=States.IS;
        return state;
    }
    boolean isViolated(){
        if(newTrustState()==States.IU)return true;
        return false;
    }
}
```

Figure 7.8: The `FileHarmful` trust rule

provides the final state (Lines 17-21), and `isViolated()` returns true if the final state is IU (Lines 22-26).

To test this trust scenario, we construct an event as $\{sr_1$, sp_1, $UploadDocFile$, $aFile - txt - 1 - < script > .. < /script >$, sID, $t_{session}\}$, where the `fileContents` are $< script > .. < /script >$. The trust is found to be violated as the file contains script.

7.2.2 Trust Violation in File Search Service

In this section, we specify the illegal access attempt trust scenario for the file search service and generate the trust rule from the specification.

Illegal Access Attempt Trust Scenario

An advantage of sharing resources through file storage grid is that the servers are distributed, and the failure of one server will not close down the whole file storage system. However, since the servers are publicly accessible, there is always the risk of clients maliciously trying to access the resource of other clients. A popular and well known attempt to do this task is the SQL injection attack [77, 80] against the search engine of the grid storage system. The attack is performed by maliciously crafting a search query which may grant a normal client access to other clients' resources. For example, a client, while searching for files using the file search service, can provide ';'x==x" in the file search field. If the server is vulnerable to the SQL injection attack, it will make two queries from the input: ';' and 'x==x'. The second query always returns true, and so the server might provide all the file information from its database some of which may be confidential. The other way to execute this attack is

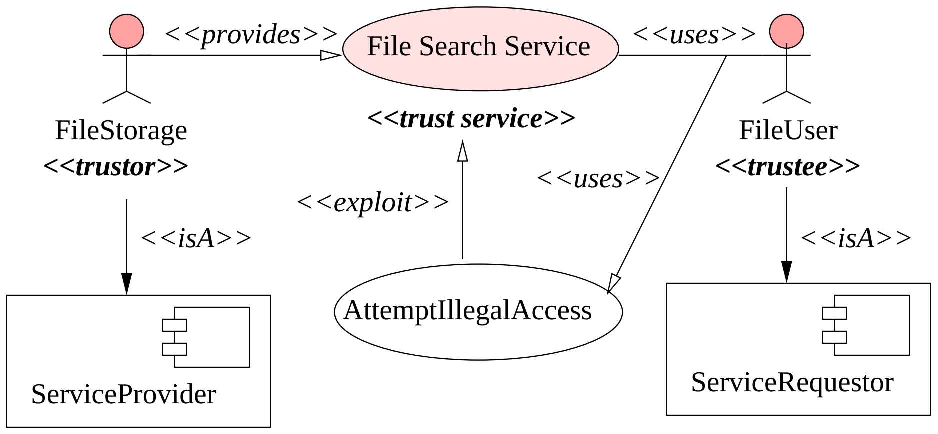


Figure 7.9: The trust-use-case diagram for the illegal access attempt trust scenario

to use 'and', 'or', or 'union' instead of '=='. A client is untrustworthy if it attempts to access such resources. We encapsulate such illegal access attempt behavior in this trust scenario. This scenario needs to be updated if any illegal access attempt other than the SQL injection attack happens against the system.

The trust-use-case for the illegal access scenario is presented in Figure 7.9, where the $<< trustor >>$ (`FileStorage`) trusts the $<< trustee >>$ (`FileUser`) in using its file search service, which the $<< trustee >>$ exploits in order to gain unauthorized access to the resources of other parties by using the `AttemptIllegalAccess` activity.

Figure 7.10 provides the trust-class diagrams specifying the illegal access attempt. The `goal` of the $<< trustor >>$ is `confidentiality` and `integrity`, as the server needs to protect the resources from unauthorized parties. The `req-attr` is `security`. The `min-trust-level` is `HIGH`. The $<< trustee >>$ attempts to exploit (`method`) the file search service. The $<< trust - concern >>$ is `IllegalAccessAttempt` with `level` set to `s-to-u` and `category` set to `fileSearch`. The submitted query for file is captured by `searchQuery`. The function `isMalformedQuery(...)` takes the `searchQuery` as parameter and checks the query for an SQL injection attack.

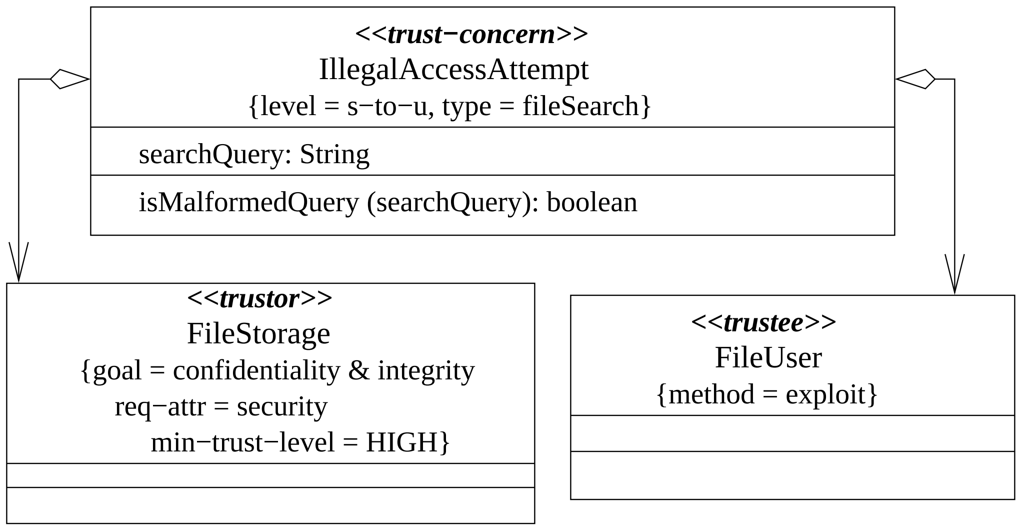


Figure 7.10: The trust-class diagram for the illegal access attempt trust scenario

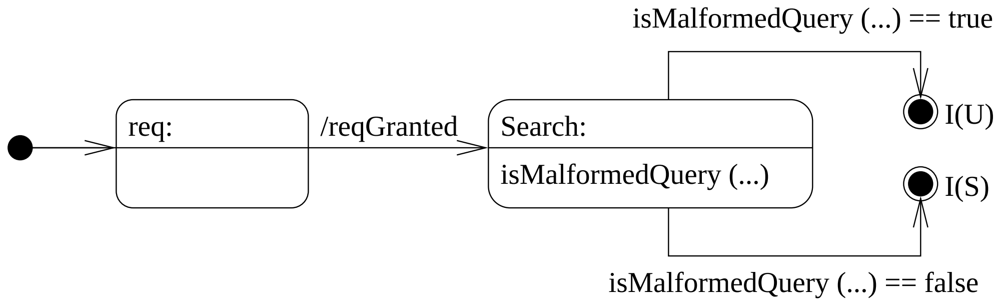
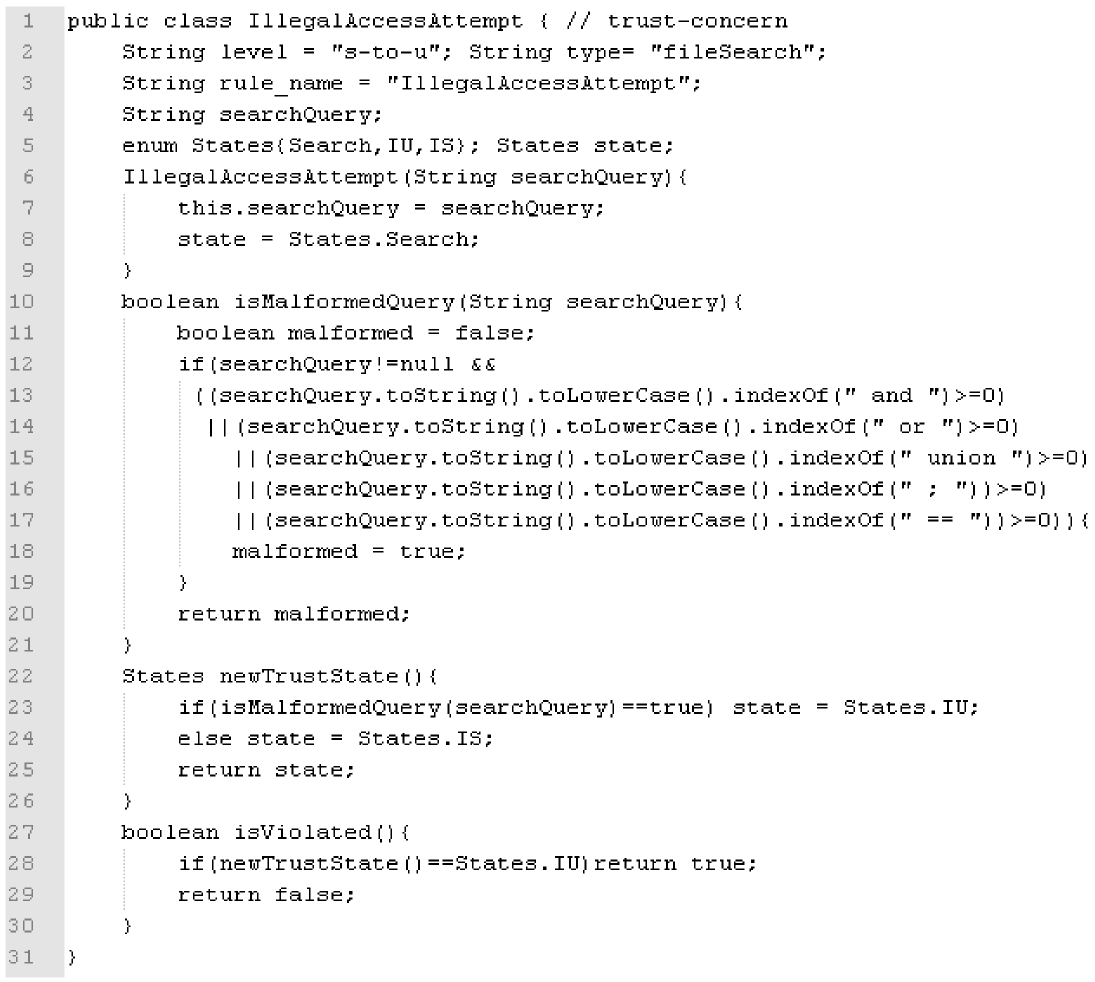

isMalformedQuery (...) == true
req:
/reqGranted
Search:
isMalformedQuery (...)
I(U)
I(S)
isMalformedQuery (...) == false

Figure 7.11: The trust-state diagram for the illegal access attempt trust scenario

The trust-state-machine diagram of the illegal access attempt is provided in Figure 7.11.When the user submits the query, the *Search* state is reached. If the user inputs a normal query, the interaction with the user is considered satisfactory ($I(S)$); however, if the user inputs a malformed search query, the interaction is treated as unsuccessful ($I(U)$).

Figure 7.12 presents the implemented code for the `IllegalAccessAttempt` trust rule. At first, the `level` and `type` are set based on the specification (Line 2), and the `rule_name`, `searchQuery` and `States` are declared and initialized (Lines 3-9).



```
public class IllegalAccessAttempt { // trust-concern
    String level = "s-to-u"; String type= "fileSearch";
    String rule_name = "IllegalAccessAttempt";
    String searchQuery;
    enum States{Search,IU,IS}; States state;
    IllegalAccessAttempt(String searchQuery){
        this.searchQuery = searchQuery;
        state = States.Search;
    }
    boolean isMalformedQuery(String searchQuery){
        boolean malformed = false;
        if(searchQuery!=null &&
         ((searchQuery.toString().toLowerCase().indexOf(" and ")>=0)
          ||(searchQuery.toString().toLowerCase().indexOf(" or ")>=0)
            ||(searchQuery.toString().toLowerCase().indexOf(" union ")>=0)
            ||(searchQuery.toString().toLowerCase().indexOf(" ; "))>=0)
            ||(searchQuery.toString().toLowerCase().indexOf(" == "))>=0)){
            malformed = true;
        }
        return malformed;
    }
    States newTrustState(){
        if(isMalformedQuery(searchQuery)==true) state = States.IU;
        else state = States.IS;
        return state;
    }
    boolean isViolated(){
        if(newTrustState()==States.IU)return true;
        return false;
    }
}
```

Figure 7.12: The `IllegalAccessAttempt` trust rule

The function `isMalformedQuery(...)` checks the `searchQuery` for possible SQL injection attempts (Lines 10-21), returning true if the query is malformed (Line 20). `newTrustState()` returns the final trust state as IU if the query is malformed, otherwise the final state is IS (Lines 22-26), and `isViolated()` checks for trust rule violations as usual (Lines 27-30).

To test this trust scenario, we send a service session event from a client sr_1 as $\{sr_1$, sp_1, $SearchDocFile$, "; *x or x*", sID, $t_{session}\}$, where the requested service name is $SearchDocFile$ and the parameters for this search service are "; *x or x*". The service parameters form the `searchQuery`. The `IllegalAccessAttempt` trust rule finds that the query is malformed and mentions that trust is violated.

7.2.3 Trust Violation in File Open Service

In this section, we specify the remote file inclusion trust scenario for the file open service and generate the trust rule from the specifications.

Remote File Inclusion Trust Scenario

A malicious user may try to open remote file using the file open service of the file server, where the remote file may contain malicious content. Once such a file is open in the server space, it can harm the file server by gaining extra privileges or by running malicious scripts. The simplest way to detect this violation is to have a list of valid files on the file server which the file server considers as safe. If the user attempts to open any file outside this white list, a remote file inclusion trust violation is detected. The scenario is also called a local file inclusion attack [78].

Figure 7.13 presents the trust-use case diagram for the remote file inclusion trust

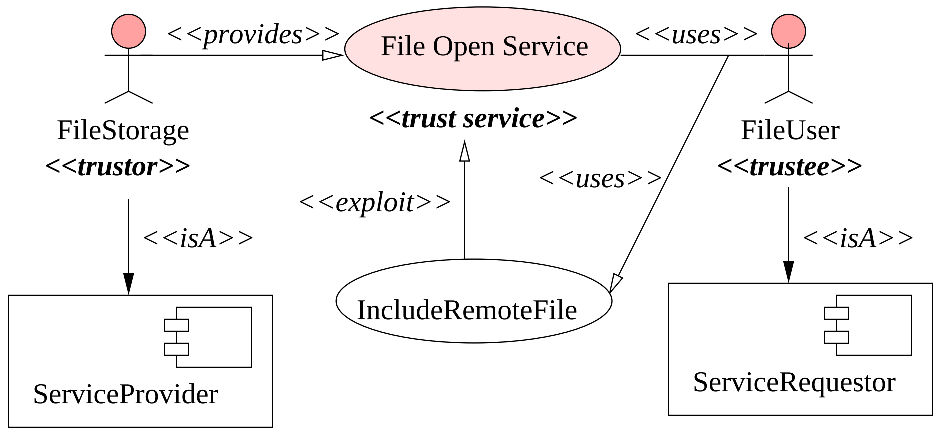


Figure 7.13: The trust-use-case diagram for the remote file inclusion trust scenario

scenario. `Fileuser` is the service requestor, and `FileStorage` is the service provider. The server trusts the requestor by providing it the file open service, and the requestor invokes ($<<$ *uses* $>>$) it. However, the requestor can open remote file ($<<$ *exploit* $>>$) by using the `IncludeRemoteFile` activity.

Figure 7.14 provides the trust-class diagram for the remote file inclusion trust scenario. The $<<$ *trustor* $>>$ has its `goal` set to the safety of the server while the $<< trust-concern >>$ is `RemoteFileInclusion` with `level` set to `s-to-u` and `type` set to `fileOpen`. The function `chkRemoteFile(...)` is used to check whether the requested file for opening is in the known list of the server. If it is not, the server considers that the user is trying to open a remote file in the server and returns true.

Figure 7.15 presents the trust-state machine diagram for the remote file inclusion trust scenario. The `chkRemoteFile(...)` function checks whether the file is in the safe list of the server. If so, the interaction with the client is successful ($I(S)$). Otherwise, the interaction is unsuccessful ($I(U)$).

The implemented code of the `RemoteFileInclusion` trust rule is provided in Figure 7.16, where the `level` and `type` are set (Line 3). and variables are declared

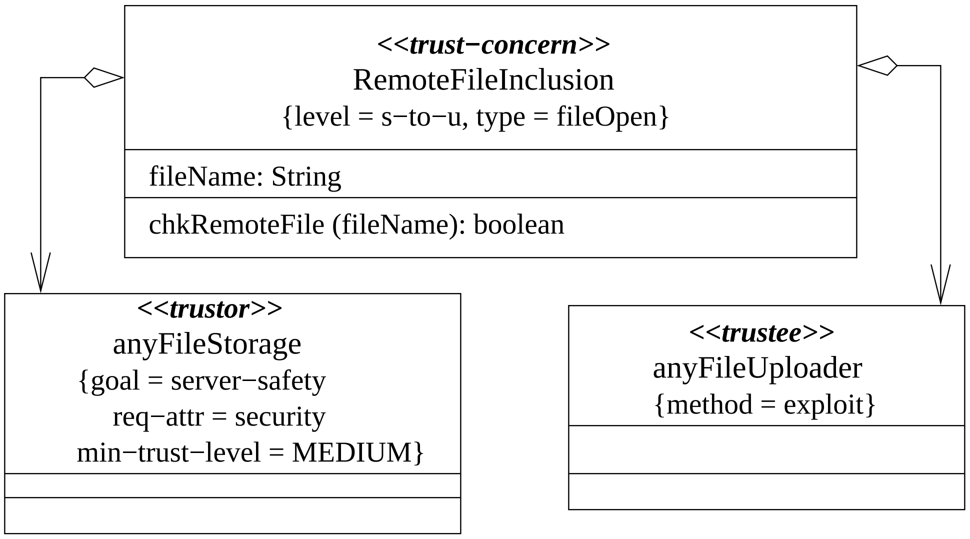


Figure 7.14: The trust-class diagram for the remote file inclusion trust scenario

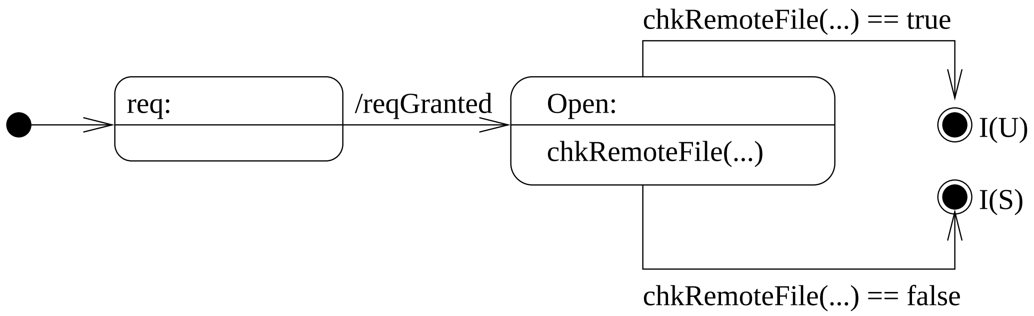
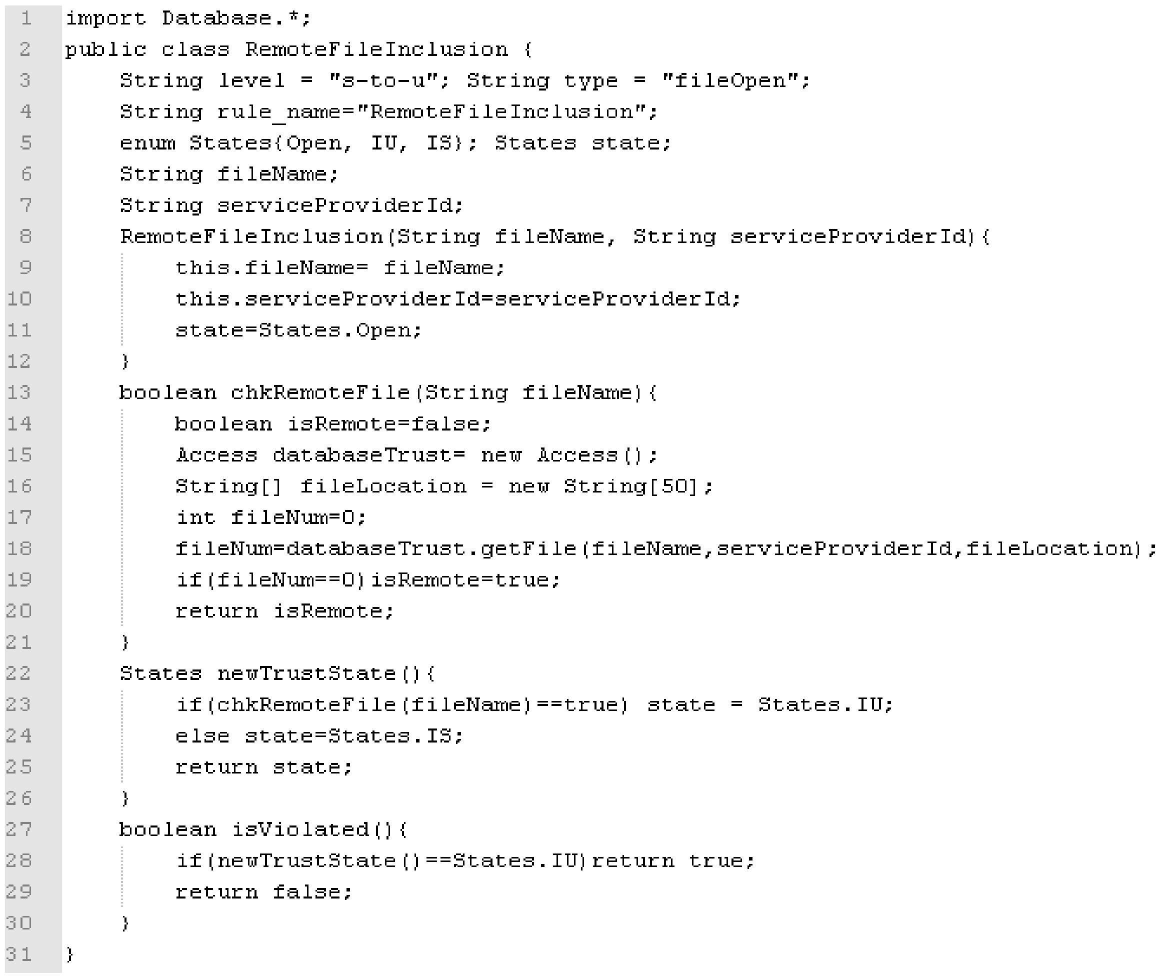


Figure 7.15: The trust-state-machine diagram for the remote file inclusion trust scenario



```
import Database.*;
public class RemoteFileInclusion {
    String level = "s-to-u"; String type = "fileOpen";
    String rule_name="RemoteFileInclusion";
    enum States{Open, IU, IS}; States state;
    String fileName;
    String serviceProviderId;
    RemoteFileInclusion(String fileName, String serviceProviderId){
        this.fileName= fileName;
        this.serviceProviderId=serviceProviderId;
        state=States.Open;
    }
    boolean chkRemoteFile(String fileName){
        boolean isRemote=false;
        Access databaseTrust= new Access();
        String[] fileLocation = new String[50];
        int fileNum=0;
        fileNum=databaseTrust.getFile(fileName,serviceProviderId,fileLocation);
        if(fileNum==0)isRemote=true;
        return isRemote;
    }
    States newTrustState(){
        if(chkRemoteFile(fileName)==true) state = States.IU;
        else state=States.IS;
        return state;
    }
    boolean isViolated(){
        if(newTrustState()==States.IU)return true;
        return false;
    }
}
```

Figure 7.16: The `RemoteFileInclusion` trust rule

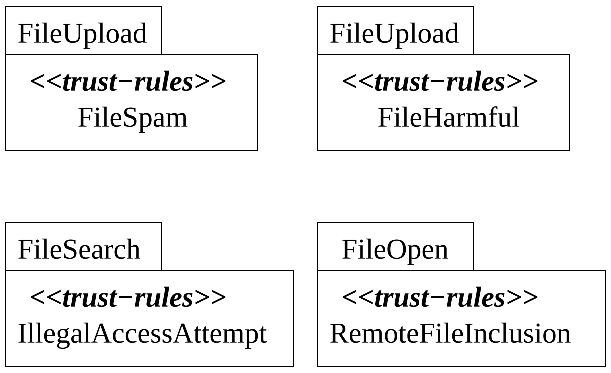


Figure 7.17: Trust-package diagrams to store trust rules for file upload, search, and open services

and initialized (Lines 4-12). `chkRemoteFile(...)` looks into the file database based on the provided file name (Lines 13-21). Every service provider has a file database containing the names of files it considers safe and valid (Lines 18). If the provided file name is not found in the file database, the file is considered a remote file (Line 19). `newTrustState()` returns the final state as IU if the file is considered remote (Lines 22-26).

To test this trust scenario, we construct an sSN event as $\{sr_1$, sp_1, $OpenDocFile$, $aFileName$, sID, $t_{session}\}$, where the requested service is $OpenDocFile$. The name of the file is sent to the `RemoteFileInclusion` trust rule to check whether it is remote to the server, querying the file database. If it finds the file name in the database, the interaction is trustworthy, otherwise a violation of trust is detected.

Figure 7.17 presents the `FileUpload` trust-package to store the `FileSpam` and `FileHarmful` trust rules, `FileSearch` trust-package for the `IllegalAccessAttempt` trust rule, and `FileOpen` trust-package for the `RemoteFileInclusion` trust rule.

7.3 Evaluation of Trust Modeling and Monitoring

In this section, we analyze the different cases of the trust-based service granting algorithm (recall the 4 cases from Section 4.3) and the trust monitoring algorithm (recall the 2 cases from Section 5.4). Our focus of this evaluation is to see whether our trust-aware service providers are able to monitor service-based interactions from trust perspectives, update trust values, make run-time decisions on interactions and whether they can decide on granting services to requestors based on the trustworthiness of the requestors. Experimental results from the prototype confirm that we can construct such a trust-aware service-based software.

In all cases, the service provider is sp_1. The recommenders are sp_2, sp_3, and sp_4. All the sRQ events are sent to the Trust Engine by the Event Dispatcher module. The trust equations used in our model have a number of constants. The default constant values used in our implemented system are as follows. The value of w_b in calculating confidence is 0.8 to emphasize that the users should be provided with more chances even if they were untrustworthy before. The value of δ in calculating direct trust is 0.8 to give more weight to previous confidence values. This means that a single recent untrustworthy interaction will not outweigh a history of trustworthy interactions and the entity may be given another chance. The value of Υ in time-based ageing parameter is 0.2 because in our system, there were only a few interactions (*e.g.,* 100 at a time). η is chosen as 1 to provide a direct recommendation with more accuracy. Λ is 10 in path-based ageing parameter, since we have created in total 10 service providers. Ψ is 0.2 in path-based ageing parameter, since all the service providers in our system are considered almost trustworthy. The value of ζ in updating recommendation accuracy is 0.8 to give more weight to previous recommendation

Table 7.2: Values of different constants used in experiments

Constant Value	**Constant Value**	**Constant Value**
$w_b = 0.8$ (Equation (4.3))	$w_d = 0.2$ (Equation (4.3))	$\delta = 0.8$ (Equation (4.4))
$\Upsilon = 0.2$ (Equation (4.9))	$\eta = 1$ (Equation (4.10))	$\Lambda = 10$ (Equation (4.12))
$\Psi = 0.2$ (Equation (4.12))	$\zeta = 0.8$ (Equation (4.13))	$\alpha = 0.8$ (Equation (4.20))

accuracy. It is chosen since a recommender can be wrong recently, however that cannot be used to decide its reliability. The value of α in the total trust calculation is 0.8 to emphasize more on self-observation and is used as 0.3 when sp_1 has had no direct interaction with sr_1. The constant values are summarized in Table 7.2, where each column provides the constant values used in experiments along with their corresponding equations. Although the constant values used in the experimental evaluation are chosen after careful considerations, they might change depending on the target service-based system.

Since we did not use any of the general trust or recommendation calculation equations in the implementation, we did not consider any priority value (*i.e.,* $\rho(c_i)$) of different services in our prototype. The constant values specified in Table 7.2 are listed in `MonitorConfigurations.xml` (recall Figure 5.5), from which different modules of the Trust Monitor read those for calculation purposes. This configuration file also provides the list of recommenders for sp_1 used in experiments. The evaluation of different cases of trust-based service granting algorithm is provided in Section 7.3.1, while Section 7.3.2 evaluates the cases for trust-based interaction monitoring.

7.3.1 Evaluation of Trust-Based Service Granting Algorithm

According to Section 4.3, the service granting algorithm has four different cases. We will go case by case in the following paragraphs.

Case (a): Known For The Same Service

In this case, the requestor sr_1 is known and has interacted previously for the requested service `UploadDocFile`. To test this case, we store a direct trust value of sr_1 in the `Direct Trust` repository for the service. When the request is received, the `Direct Trust` repository is searched for possible values. Since a direct trust value is already there, the client is considered known for this service. The `Trust Engine` sends request to Recommendation Engine to query for new recommendations. The Recommendation Engine sends rRQ to sp_2, sp_3, sp_4 and receives rRP accordingly. sp_4 does not have the requested value, and so it requests sp_5. Based on the reply from sp_5, sp_4 forwards it to sp_1. Since this recommendation from sp_5 has the visited path-length of 3, it is considered as an indirect recommendation. The Recommendation Engine calculates it, and then stores all the new recommendations by deleting the old ones in the `Recommendations` repository. The execution control then returns to the Trust Engine which retrieves the recommendations of sp_2, sp_3 and sp_5 from the `Recommendations` repository, and their corresponding previous recommendation accuracies from the `Recom-Accuracy`. If there is no previous recommendation accuracy available for a particular recommender on a requestor about a requested service, it is considered as a completely new recommender on sr_1 for $UploadDocFile$. The recommendation accuracy for this new recommender is set to 1 by default (*i.e.,* initially, the recommender is considered as the most accurate on sr_1 for $UploadDocFile$). The total trust is calculated and the compared against interaction threshold ($I.T.$) of this service. The result of the comparison is sent to the Trust Decision Notifier, which constructs a $serviceDecision$ based on the result and passes the execution flow to Trust Actions by sending it the $serviceDecision$. For example, if the calculated total trust is greater than the $I.T.$, the $serviceDecision$ has the form $\{sr_1$, $UploadDocFile$, $Accept$, $InteractionTime\}$.

Case (b): Known But Not For The Same Service

In this case, the service requestor is sr_3 and the service s is `UploadDocFile`. However, the sr_3 has previous interaction record with the sp_1 for `UploadPDFFile` service. The sp_1 has the repository `Similar`, where the similarity between the two services are listed as 0.50 (If there are a number of similar services, we consider the service with the highest similarity value). The program execution flow then follows exactly as in the first case.

Case (c): Unknown But Recommendations Available

In this case, the service requestor is sr_5 and the requested service s is `UploadDocFile`. sr_5 has had no previous interaction with sp_1 based on the `Direct Trust` repository, *i.e.*, , sp_1 has no entry in the `Direct Trust` repository for sr_5. The Recommendation Engine is called by the Trust Engine to query for recommendations, which are found. Since the requestor is unknown to sp_1, any recommendation regarding the sr_5 is considered to have accuracy 1 by default. However, the value of α is reduced to 0.3 in this case since sp_1 has no previous interaction value for sr_5.

Case (d): Unknown And Recommendation Not Available

In this case, the service requestor is sr_6 who is completely new to sp_1 and to all recommenders. Therefore, the Trust Engine provides a default direct trust value 0.5 to sr_6 for the requested service s and notifies the Trust Action Notifier accordingly.

The direct trust value is stored in the `Direct Trust` repository. After this case is executed, the requestor becomes known to the system. Therefore, next time the user requests the same service, it is treated as one of first two cases (*i.e.,* Cases (a) or (b)).

7.3.2 Evaluation of Trust Monitoring Algorithm

Upon receiving service session (sSN) events, the Event Dispatcher hands the event to the Trust State Analyzer which analyzes the event against the corresponding trust rules retrieved from the `ServiceTrustContext.xml` file. The service requestor is sr_1, the service provider is sp_1, the service s is `UploadDocFile`, and the trust rules are `FileExcess` and `UploadCompletion` for sSN. The implementation details of the `FileExcess` trust rule are provided in Section 7.2. The `UploadCompletion` trust rule is satisfied when the `FileExcess` trust rule is not violated. We use the three configuration file provided in Figures 5.3, 5.4, and 5.5 in our experiments. There are two cases to consider, depending on whether a trust rule is violated or not.

Case (a): Service Requestor Violates Trust Rule

In this case, the sSN event has the form $\{sr_1$, sp_1, $UploadDocFiles$, $docFileName - doc - 200 - fileContents$, $sr1300089544370$, $t_{session}\}$, where 200 is the size of the uploaded file in megabytes (MB). According to the `ServiceDesciptor.xml` file, the `maxPOST` is 100 MB. The `isViolated()` function in the `FileExcess` trust rule returns true, so an alert is generated in the `Alerts` repository in the form $\{sr_1$, $UploadDocFile$, $FileExcess$, $sr1300089544370$, $t_{session}\}$. Since the `FileExcess` trust rule is violated, the `UploadCompletion` trust rule is also violated. The Trust State Analyzer then calculates the confidence (μ) and notifies both the Trust Engine and the Trust Decision Notifier. The Trust Engine updates the direct trust, and recommendation accuracies of sp_2, sp_3 and sp_4, while the Trust Decision Notifier notifies the Trust Actions of the service decision.

Case (b): Service Requestor Follows Trust Rule

In this case, the event has the form $\{sr_1, sp_1, UploadDocFile, docFileName - doc - 50 - fileContents, sr1300089544371, t_{session}\}$, where the file size is smaller than the maximum acceptable limit. The `isViolated()` function returns false, and so the `UploadCompletion` trust rule is satisfied. No alert is generated, and the Trust Engine and Trust Decision Notifier are sent notifications accordingly. The trust values and recommendation accuracies are updated by the Trust Engine.

7.4 Performance Overhead

The trust-aware service-based software thus developed employs the trust monitor for trust-aware execution. The monitor provides trust-based analysis and dynamic decisions based on event analysis at run-time. However, the monitor creates some performance overhead also. The overhead is of three types[2]:

- Delay in providing a decision on an sRQ event,
- Delay in analyzing an sSN event, and
- Delay in a long recommendation chain.

The three types of performance overhead are discussed in details in the next three subsections.

[2]The actual overhead may depend on the target service-based system under monitoring.

7.4.1 Delay in Providing Decision on an sRQ Event

A provider performs two tasks before making a decision on an sRQ event: the retrieval of previous direct trust value with the requestor for the specific service from the `Direct Trust` repository, and the handling of recommendations from other service providers. The handling of recommendations includes the requests for recommendations and the receipt of the corresponding recommendation replies. In a service-based system without the trust monitor, these two tasks would not be present.

We analyze the performance of our system for sRQ. The delay is calculated by taking the difference between the sending time of an sRQ event and the receipt time of the corresponding decision in an sRP. The delay recorded was solely because of the processing of an event in an entity. In the experimental setup, we use one service provider (sp_1) to provide services, three service providers (sp_2, sp_3, and sp_4) to provide direct recommendations. We vary the number of service requests from 10 to 100 from a service requestor sr_1. We run the experiment for each setup 10 times and take the average to minimize errors.

Figure 7.18 presents the experimental result with and without the trust monitor while providing a service reply (sRP). The system without the trust monitor just receives the sRQ event and provides sRP randomly (*i.e.,* without using any trust-based analysis), while the system with the trust monitor employs trust-based analysis before providing sRP. The result shows that the trust-based processing of an sRQ introduces some delay in providing the corresponding service decision. However, the result is encouraging since the delay does not increase with the increase in service requests.

The response from a provider without the trust monitor requires almost constant

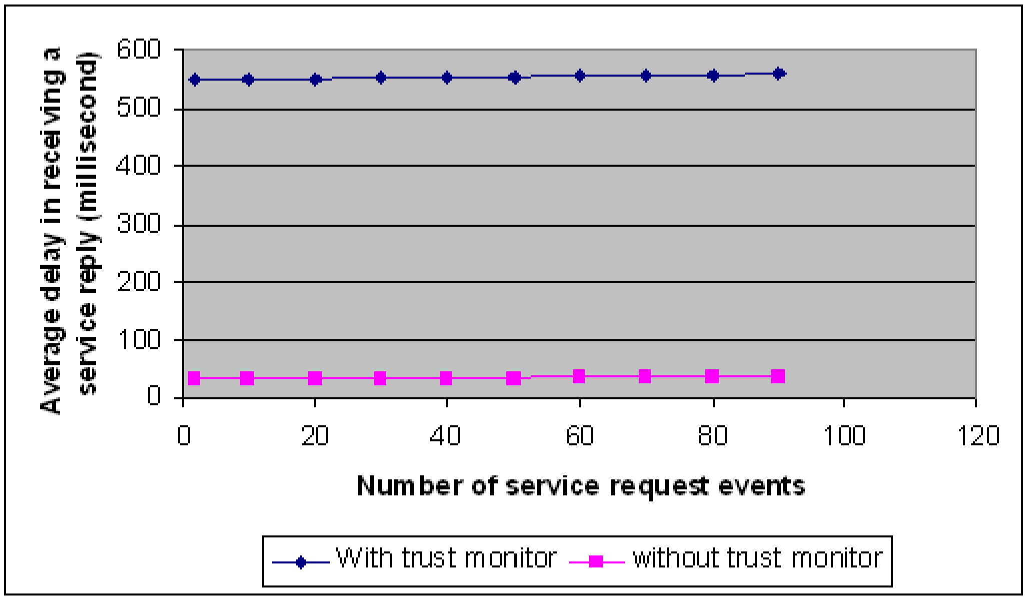


Figure 7.18: The average delay in receiving a service reply (sRP) with respect to a service request (sRQ)

time (in the range of $30 - 40$ milliseconds), while the response from a provider with the trust monitor also requires an almost static time (in the rage of $500 - 600$ milliseconds). The reason for this is as follows. All of the modules in our provider software are implemented as specific behaviours (*i.e.,* threads) in the Jade platform. With the arrival of each service request, the provider creates new instance of different modules of the monitor. For example, for 100 service requests at a time, the provider software creates 100 instances of each working module. The creation of these instances takes almost constant time, and so the total time required to create 100 instances of working modules is almost the same as the time required to create 10 instances. Since the instances start working with the arrival of service requests, the execution is performed in parallel for each service request. However, the communication between individual modules of a software takes time, which is one reason of having more time in trust-based processing. Moreover, sp_1 sends recommendation requests and receives recommendation replies from three other providers which need time. The accessing of each of the shared database tables is performed in a synchronized manner which creates a slight increase in the average delay with the increase in service requests.

7.4.2 Delay in Analyzing an sSN Event

Our trust-aware software uses the trust monitor to analyze a service session (sSN) event. The analysis is performed in two steps: the analysis of the sSN based on the trust rules, and the retrieval, calculation and update of direct trust, recommendation, and recommendation accuracies based on the analysis of the sSN. The delay is calculated by taking the difference between the sending time of an sSN event and the corresponding reply time based on the service decision on the sSN. To examine the overhead of this analysis, we send a number of sSN events to the sp_1 varying from 10 to 100, where the requested service is `UploadDocFile`. The provider uses the configuration files listed in Figure 5.3, Figure 5.4, Figure 5.5. Figure 7.19 provides the experimental results which show that the incorporation of the trust monitor to analyze an sSN introduces some delay to the processing of the event; however, the delay remains almost constant with the increase in service session events.

The delay in providing a service reply without the trust monitor remains almost constant in the range of $30 - 40$ milliseconds, while the delay with the trust monitor also remains almost constant in the range of $80 - 90$ milliseconds. The reason is the same as the processing of service request events discussed in the previous subsection. However, the monitor does not need to send or receive recommendations to analyze an sSN. Therefore, the delay occurred is only due to the analysis of trust rules and the

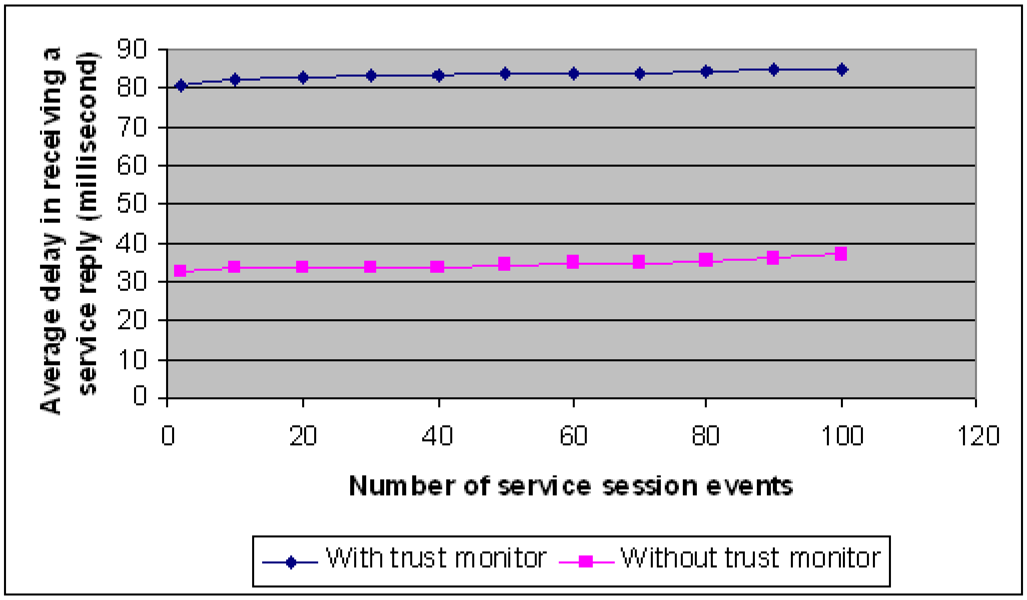


Figure 7.19: The average delay in receiving a service reply (sRP) with respect to a service session event (sSN)

accessing of the database for the retrieval and update of trust values. Nevertheless, the slight increase in average delay with the increase in service session events is due to the synchronized accessing of shared database tables by individual instances.

The combined analysis of Figures 7.18 and Figure 7.19 provides another interesting result. While the average delay is in the range of $500-600$ milliseconds for service request events, it is in the range of $80-90$ milliseconds for service session events, showing the average difference as $420-520$ milliseconds. The major difference in handling an sRQ and an sSN is the handling of recommendations. Therefore, the providers in our system should try to avoid recommendations whenever possible while handling service request events. We did not do that since we evaluate our system based on our trust-based service-granting algorithm that queries for new recommendations to make decision on each service request.

7.4.3 Delay in a Long Recommendation Chain

In our trust-aware software system, we allow both direct and indirect recommendations. Since recommendations are handled as sending of recommendation requests and receiving of the corresponding replies, there is some delay in providing run-time decisions on an sRQ. However, while analyzing the delay on an sRQ, we only considered direct recommendations. Since we allow both direct and indirect recommendations in our system, we were interested to see how much overhead would be created while a provider allows indirect recommendations (*i.e.,* when the maximum allowable path-length in an rRQ is more than 2). We were interested to see the impact of handling recommendations with long chain, *i.e.,* when the path-length for indirect recommendations varies.

We send a recommendation request (rRQ) from sp_1 to another provider which delegates the recommendation request to its neighbor if it does not have the recommendation value. This goes on until the maximum allowable path length is reached. In that case, the recommendation value is provided as -1 from the last provider, which means that no such recommendation value is found within the maximum allowable path-length limit. To do this experiment, we assumed a long chain from sp_1 to sp_{10}, where sp_1 asks sp_2 for recommendation, sp_2 to sp_3, and so on. We further assumed that a provider can ask for recommendations to only one another provider, *i.e.,* , sp_1 only to sp_2 and sp_2 only to sp_3. We varied the number of recommenders from 1 to 9 (*i.e.,* sp_2 to sp_{10}). We send recommendation requests from sp_1 and calculate the difference between a recommendation request and the corresponding recommendation reply in sp_1.

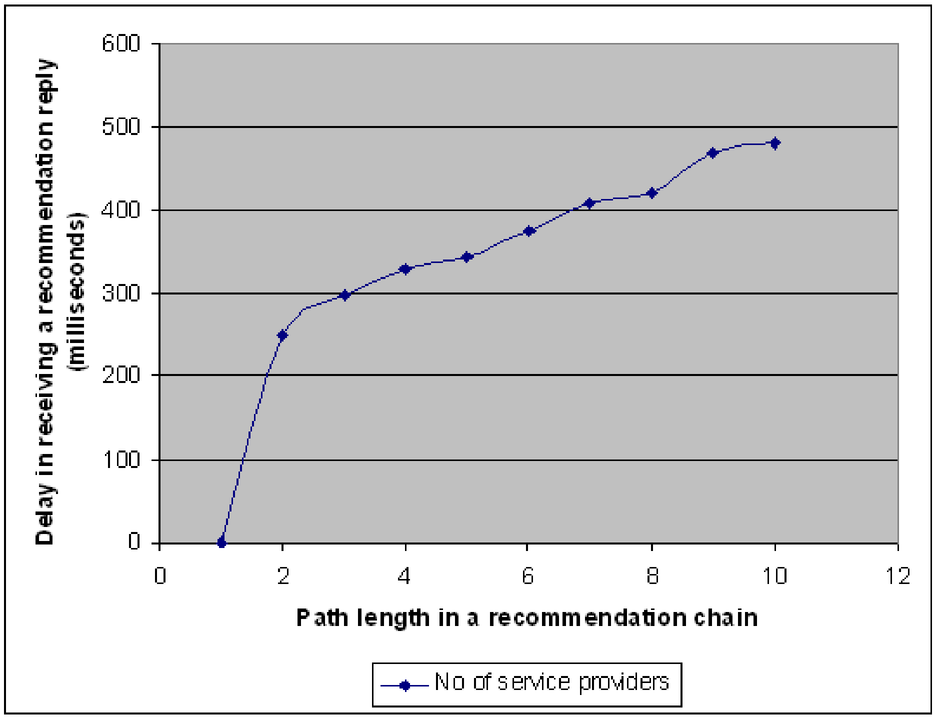


Figure 7.20: The average delay in receiving a recommendation reply (rRP) in a long recommendation chain

Figure 7.20 presents the experimental results which show that the delay in receiving a recommendation reply increases with the increase of path-length in a recommendation chain, maintaining an almost linear relationship with path-lengths. It should be noted here that a path-length of 10 should be avoided in our system since the recommendation value from a such distance is decreased considerably because of our path-based ageing parameter. For example, for a recommendation value of 0.8, with maximum allowed path length as 10 and visited path length as 10, the value of indirect recommendation using Equation (4.11) will be 0.67. The difference between the original and derived recommendation values is thus 0.13. Therefore, the providers in our system should not use long recommendation chain, if they want to use recommendations properly and to provide quick responses to service requests.

7.5 Summary

In this chapter, we discuss the experimental evaluation of our proposed service-based software development framework. We evaluate the applicability of UMLtrust by specifying a number of trust scenarios from file sharing applications. We specify trust scenarios based on three prominent file sharing services, file upload, search and open. We generate and test trust rules from the specified trust scenarios.

We evaluate different cases of the trust modeling and trust monitoring approach to see their applicability in our prototype. We found that the trust-aware service providers in our developed prototype are able to analyze service-based interactions based on trust rules, calculate and update the trustworthiness of requestors using the trust model, and make run-time decisions based on the trust-based service granting algorithm and trust monitoring algorithm.

To find the overhead of using the trust monitor in our prototype, we conduct three experiments. The first experiment concludes that there is some delay in providing trust-based service granting decision on service request events, although the average delay remains almost constant with the increase in service requests. The second experiment shows that the run-time monitoring and analysis of service-based interactions does not create that much overhead. It should be noted that the purpose of this measurement is to show that the performance overheads remain almost static, which makes the approach applicable for large-scale systems. Therefore, the focus of the first two experiments was not show the difference between the two response time delays that occur in the system with and without the monitor. The third experiment confirms that a large recommendation chain is probably not a good idea when there is a need for prompt reply to service request events.

While recommendations are necessary in trust-based decision making, their handling takes a considerable time. We conclude that the providers in our system should try to avoid sending and receiving recommendations whenever possible. Although our current trust-based service granting algorithm does not allow this, we can modify it to some extent. For example, when a service requestor is known to a service provider with a high direct trust value, the provider can make decisions only based on its self-observations. This can be applied to cases (a) and (b) of trust-based service granting (recall the four cases of Section 4.3). Moreover, the providers can exchange recommendations with each other periodically instead of querying for recommendations based on service requests. This will help in providing prompt responses to service requests in all the cases of the trust-based service granting algorithm.

Chapter 8

Conclusions

8.1 Summary

Due to the openness of service-based systems, untrustworthy service requestors can misuse and exploit service providing software. Therefore, the provider software need to consider trust concerns in their development and monitor service-based interactions. A trust-aware software analyzes service-based interactions from trust perspectives, calculate the trustworthiness of service requestors based on the analysis, and makes trust-based decisions. The major contribution of this thesis lies in handling the following issues: the specification of trust scenarios to develop a service-based software in a trust-aware manner, and the incorporation of trust calculation schemes in service-based automatic interaction analysis and decision process.

To address the above mentioned issues, we propose a trust-aware service-based software development framework which allows the specification of trust scenarios using our proposed UML profile called UMLturst [1]. A trust scenario binds interested parties together based on a context. The trust scenarios are converted to trust rules for monitoring purposes. The applicability of UMLtrust is illustrated using examples from file sharing applications.

To incorporate trust computation to the total system development, we propose a trust model called CAT (Context-Aware Trust) [2]. A service requestor is penalized for violating a trust rule, while it is rewarded for having no violations. Based on this analysis, the trustworthiness of requestors is calculated which helps in making automatic decisions on whether a service requestor should be provided with the requested service. The service granting decision is facilitated by our proposed trust-based service granting algorithm. The trust model and the service-granting algorithm are illustrated using examples from a file sharing grid.

To support run-time trust-based interaction analysis and decision making, we propose an automatic trust monitoring approach. A trust monitoring architecture is provided which is assumed to reside in each trust-aware service provider software. The trust rules and the trust model are incorporated into the trust monitor. A trust monitoring algorithm is presented to analyze service-based interactions, calculate the trustworthiness of requestors, and make trust-based dynamic decisions. The trust monitoring approach is elaborated using examples from a file sharing grid.

To examine the effectiveness of the proposed trust-aware service-based software development framework, we have developed a prototype file sharing grid. In this grid structure, each of the service providers employs our trust monitoring architecture to monitor service-based interactions at run-time based on some trust rules, calculates the trustworthiness of service requestors based on the trust model, and makes trust-based dynamic decisions based on the algorithms.

8.2 Limitations and Future Work

Our future enhancements to the system will concentrate on addressing the following limitations of our research work. While specifying a trust scenario, we assume that the identity of a trustee is properly resolved. For example, in a session hijacking attack [89], the attacker hijacks the session of a legitimate user to gain illegal access to the system. While this attack can be detected by using proper method, the identity of the attacker is hard to find. We use UMLtrust to specify trust scenarios for the facilitation of determining the trustworthiness of service requestors. Since in the above mentioned attack, the attacker spoofs the identity of a legitimate user, we cannot address this kind of scenario using UMLtrust.

In CAT, it is considered that the network running a trust model is secure from false recommenders, and therefore, the recommendation from an entity is considered to truly be from that entity. However, in real situations, this might not be the case. For example, CAT does not consider defense against the Sybil attacks [91], where a malicious entity creates a large number of artificial entities for the purpose of providing false recommendations. The focus of the malicious entity is to influence trust-based decision making. We consider that the problem of pseudonymity can be handled by an authentication system. The authentication system is responsible for providing unique identifications to the interacting entities. For example, in a resource sharing grid, the users belong to particular organizations [46]. The organizations subscribe to specific service providers for services. Therefore, the authentication of the users is basically performed by the organizations.

We use CAT for evidence-based trust decision making, where the trust rules are used to monitor the interactions between the entities. Based on the monitoring, trustworthiness of the interacted entities are measured. However, CAT measures the accuracies of the corresponding recommendations after each interaction. Since the measurement of accuracy is performed by calculating the difference between the direct interaction trust and the recommendations, unreliable recommendations are detected and discarded accordingly. In this way, we can also partially address the colluding malicious node attack [90], where malicious nodes combine to provide false recommendations about an entity. However, it is worth mentioning here that CAT is not intended to provide defense against the above mentioned attacks.

The incorporation of the trust monitor into the total system imposes some performance overhead. The analysis of performance overhead of our system shows that the reliance of recommendations should be avoided as much as possible when there is a need for prompt reply. However, our current trust-bases service granting algorithm does not support this, and so we need to modify the algorithm to address the above mentioned performance issue.

The trust monitoring algorithm only indicates whether an interaction is successful or unsuccessful. In a real world environment, this might not be sufficient. For example, the service provider may want to provide warning to the service requestors before penalizing them with distrust values. The reason for doing this is as follows. A service requestor may upload a large file beyond the acceptable limit of a service provider without any bad intention. The first few such attempts could be treated as mistakes by considering the requestor as error-prone. However, if it does this misbehavior beyond an acceptable number, it surely is untrustworthy. The current trust monitoring algorithm does not address this issue. However, the incorporation of this feature will make the monitoring algorithm more sensitive to error-prone service requestors.

In our experimental evaluation, we assign values to different equations constants. Although we have determined these constant values after careful consideration, we need to confirm the constant values against real-world service-based systems. The proposed approach needs to be adapted to real-world service-based systems to analyze the robustness of the calculation equations and determine the configuration values of different constants used in the equations. For this purpose, a standard database of real-time service-based events is needed which is currently not available.

Although our approach of developing and automatic monitoring trust-aware service-based software is a comparatively new initiative, we show the necessity and effectiveness of this approach in this thesis. However, to make it complete based on the limitations mentioned above, we will work on a number of directions. First, UML-trust will be extended to cover more trust scenarios of diverse natures. Second, CAT will be extended to address the above discussed attacks. Third, the calculation equations of CAT will be evaluated against different systems. Fourth, the trust-based service granting algorithm will be modified to decrease reliance on recommendations in making decisions. And finally, the trust monitoring algorithm will be extended to make it more sensitive to error prone service requestors.

Bibliography

[1] Uddin, M.G., Zulkernine, M. UMLtrust: Towards developing trust-aware software, to appear in *Proceedings of the 23rd Annual ACM Symposium on Applied Computing (SE Track)*, Fortaleza, Brazil, March 2008, 6pp.

[2] Uddin, M.G., Zulkernine, M., Ahamed, S.I. CAT: A context-aware trust model for open and dynamic systems, to appear in *Proceedings of the 23rd Annual ACM Symposium on Applied Computing (TRECK Track)*, Fortaleza, Brazil, March 2008, 6pp.

[3] Bichler, M., Lin, K.J. Service-oriented computing, in *IEEE Computer*, **39**(3); 2006. IEEE CS Press: 99–101.

[4] Huhns, M.N., Singh, M.P. Service-oriented computing: Key concepts and principles, in *IEEE Internet Computing*, **39**(3); 2005. IEEE CS Press: 75–81.

[5] Bennett, K., Layzell, P., Budgen, D., Brereton, P., Macaulay, L., Munro, M. Service-based software: the future for flexible software, in *Proceedings of the Seventh Asia-Pacific Software Engineering Conference*, Singapore, 2000. IEEE CS Press: 214–221.

[6] Deubler, M., Grünbauer, J., Jürjens, J., Wimmel, G. Sound development of secure service-based systems, in *Proceedings of the 2nd International Conference on Service Oriented Computing*, New York, USA, 2004. ACM Press: 115–124.

[7] Dimitrakos, T. A service-oriented trust management framework, in *Proceedings of 1st International Joint Conference on Autonomous Agents and Multi-Agent Systems (LNCS vol. 2631)*, Bologna, Italy, 2002. Springer: 53–72.

[8] Ryutov, T., Zhou, L., Neuman, C., Foukia, N., Leithead, T., Seamons, K. Adaptive trust negotiation and access control for grids, in *Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*, Washington, USA, 2005. IEEE CS Press: 55–62.

[9] Rajbhandari, S., Contes, A., Rana, O. F., Deora, V., Wootten, I. Trust assessment using provenance in service oriented applications, in *Proceedings of the 10th IEEE on International Enterprise Distributed Object Computing Conference Workshops*, Hongkong, 2006. IEEE CS Press: 65–72.

[10] Gambetta, D. Can we trust trust?, in *Trust: Making and Breaking Cooperative Relations*, Gambetta, D. (ed.), Chapter 13, 1988. University of Oxford: 213–237.

[11] Haque, M., Ahamed, S. I. An omnipresent formal trust model (FTM) for pervasive computing environment, in *Proceedings of the 31st Annual IEEE International Computer Software and Applications Conference*, Beijing, China, 2007. IEEE CS Press: 49–56.

[12] Axelsson, S. Research in intrusion detection systems: A syrvey and taxonomy, in *Technical Report*, Department of Computer Engineering, Chalmers University of Technology, Goteborg, Sweden, 2000.

[13] Allen, J., Christie, A., Fithen, W., McHugh, J., Pickel, J., Stoner, E. State of the practice of intrusion detection technologies, in *Technical Report 99-15*, Software Engineering Institue, Carnegie Melon, Pittsburgh, PA, USA, 2000.

[14] Basin, D., Doser, J., Lodderstedt, T. Model driven security for process-oriented systems, in *Proceedings of the 8th ACM symposium on Access control models and technologies*, Como, Italy, 2003. ACM Press: 100–109.

[15] Yu, B., Singh, M. P. An evidential model of distributed reputation mechanism, in *Proceedings of the 1st International Joint Conference on Autonomous Agents and multiagent systems*, Bologna, 2002. ACM Press: 294–301.

[16] Robinson, W. N. Monitoring web service requirements, in *Proceedings of the 11th IEEE International Conference on Requirements Engineering*, Kyoto, Japan, 2003. IEEE CS Press: 65–74.

[17] Yu, E., Liu, L. Modelling trust for system design using the *i** strategic actors framework, in *Proceedings of the Internatioal Workshop on Deception, Fraud, and Trust in Agent Societies (LNCS v2246)*, Barcelona, 2001. Springer: 175–194.

[18] Horkoff, J., Yu, E., Liu, L. Analyzing Trust in Technology Strategies, in *Proceedings of the International Conference on Privacy, Security, and Trust*, Ontario, Canada, 2006. McGraw-Hill: 21–32.

[19] Grandison, T. Trust management for internet applications, *PhD Thesis*, University of London, July 2002.

[20] Blaze, M., Feigenbaum, J., Lacy, J. Decentralized trust management, in *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, Oakland, USA, 1996. IEEE CS Press: 164–173.

[21] Blaze, M., Feigenbaum, J., Keromytis, A. D. KeyNote: Trust management for public-Key infrastructures, in *Proceedings of the 6th International Workshop on Security Protocols (LNCS v1550)*, UK, 1998. Springer: 625–629.

[22] Resnick, P., Miller, J. PICS: Internet access controls without censorship, in *Communications of the ACM* **39**(10); 1996. 87–93.

[23] Chu, Y., Feigenbaum, J., LaMacchia, B., Resnick, P., Strauss, M. REFEREE: Trust Management for Web Applications, in *World Wide Web Journal*, **2**(3); 1997. 127–139.

[24] Herzberg, A., Mass, Y., Michaeli, J., Naor, D., Ravid, Y. Access control meets public key infrastructure, or: assigning roles to strangers, in *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, Oakland, USA, 2000. IEEE CS Press: 2–14.

[25] OMG. OMG Unified Modeling Language Specification, in *Technical report*, Object Management group, March 2003.

[26] OMG. OMG Object Constraint Language Version 2.0, in *Technical report*, Object Management group, May 2006.

[27] English, C., Terzis, S., Nixon, P. Towards self-protecting ubiquitous systems: monitoring trust-based interactions, in *Personal and Ubiquitous Computing*, **10**(1); 2005. Springer: 50–54.

[28] Jøsang, A. The right type of trust for distributed systems, in *Proceedings of The 1996 Workshop on New Security Paradigms*, Lake Arrowhead, CA, 1996. ACM Press: 119–131.

[29] Zacharia, G., Maes, P. Trust management through reputation mechanisms, in *Applied Artificial Intelligence*, **14**(9); 2000. 881-908.

[30] Mezetti, N. Towards a model for trust relationships in virtual enterprises, in *Proceedings of the 14th Database and Expert Systems Applications Workshop*, Prague, Czech Republic, 2003. IEEE CS Press: 420–424.

[31] Huynh, T., Jennings, N., Shadbolt, N. FIRE: An integrated trust and reputation model for open multi-agent systems, in *Proceedings of the 16th European Conference on Artificial Intelligence*, Valencia, Spain, 2004. 18 – 22.

[32] Jøsang, A, Ismail, R. The beta reputation system, in *Proceedings of the 15th Bled Conference on Electronic Commerce*, Slovenia, 2002.

[33] Quercia, D., Hailes, S., Capra, L. B-trust: Bayesian trust framework for pervasive computing, in *Proceedings of The Fourth International Conference on Trust Management (LNCS v3986)*, 2006. Springer: 298–312.

[34] Sabater, J., Sierra, C. REGRET:A reputation model for gregarious societies, in *Proceedings of the 4th workshop on deception fraud and trust in agent societies*, Montreal, Canada, 2001. 61–70.

[35] Azzedin, F., Maheswaran,M. Trust modeling for peer-to-peer based computing systems, in *Proceedings of the International Syposium on Parallel and Distributed Processing*, San Francisco, USA, 2003. IEEE CS Press: 10pp.

[36] Teacy, W., Patel, J., Jennings, N., Luck, M. Coping with inaccurate reputation sources: Experimental analysis of a probabilistic trust model, in *proceedings of fourth international joint conference on autonomous agents and multiagent systems*, 2002. 997-1004.

[37] Wang, Y., Singh, M. Formal trust model for multiagent systems, in *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, India, 2007. 1551–1556.

[38] Wang, Y., Singh, M. Trust representation and aggregation in distributed agent systems, in *Proceedings of the 21st International Conference on Artificial Intelligence*, Boston, 2006.

[39] Billhardt, H., Hermoso, R., Ossowski, S., Centeno, R. Trust based service provider selection in open environments, in *Proceedings of the 22nd Annual ACM Symposium on Applied Computing*, Seoul, Korea, 2007. ACM Press: 1375–1380.

[40] Ries, S. Certain trust: a trust model for users and agents, in *Proceedings of the 22nd Annual ACM Symposium on Applied Computing*, Seoul, Korea, 2007. ACM Press: 1599–1604.

[41] Sherchan, W., Loke, S., Krishnaswamy, S. A fuzzy model for reasoning about reputation in web services, in *Proceedings of the 21st Annual ACM Symposium on Applied Computing*, Dijon, France, 2006. ACM Press: 1886–1892.

[42] Kotsovinos, E., Williams, A. BambooTrust: practical scalable trust management for global public computing, in *Proceedings of the 21st Annual ACM Symposium on Applied Computing*, Dijon, France, 2006. ACM Press: 1893–1897.

[43] Jurca, R., Faltings, B. Towards incentive-compatible reputation management, in *Trust, reputation and security: theories and practice (LNCS v2631)*, 2002. Springer: 138-147.

[44] Wang, Y., Vassileva, J. Trust and reputation model in peer-to-peer networks, in *Proceedings of the Third International Conference on Peer-to-Peer Computing*, Sweden, 2003. IEEE CS Press: 150–157.

[45] Lin, C., Varadharajan, V. Trust based risk management for distributed system security - a new approach, in *Proceedings of the First International Conference on Availability, Reliability and Security*, Vienna, Austria, 2006. IEEE CS Press: 6–13.

[46] Dimmock, N., Belokosztolszki, A., Eyers, D., Bacon, J., Ingram, D., Moody, K. Using trust and risk in role-based access control policies, in *Proceedings of the 9th ACM symposium on Access control models and technologies*, New York, USA, 2004. ACM Press: 156–162.

[47] Dimmock, N., Bacon, J., Ingram, D., Moody, K. Risk models for trust-based access control (TBAC), in *Proceedings of the Third Annual Conference on Trust Management (LNCS v3477)*, 2005. Springer: 364–371.

[48] Capra, L. Musolesi, M. Autonomic trust prediction for pervasive systems, in *Proceedings of the 20th International Conference on Advanced Information Networking and Applications*, Vienna, Austria, 2006. IEEE CS Press: 481–488.

[49] Xiong, L., Liu, L. Building trust in decentralized peer-to-peer electronic communities, in *Proceedings of the 5th International Conference on Electronic Commerce Research*, Montreal, 2002.

[50] Toivonen, S., Lenzini, G., Uusitalo, I. Context-aware trust evaluation functions for dynamic reconfigurable systems, in *Proceedings of the Models of Trust for the Web workshop*, Edinburgh, Scotland, 2006.

[51] Etalle, S., Winsborough, W. Integrity constraints in trust management, in *Proceedings of the 10th ACM symposium on Access control models and technologies*, Stockholm, Sweden, 2005. ACM Press: 1–10.

[52] Skene, J., Skene, A., Crampton, J., Emmerich, W. The monitorability of service-level agreements for application-service provision, in *Proceedings of the 6th International Workshop on Software and Performance*, Buenes Aires, Argentina, 2007. ACM Press: 3–14.

[53] Sandhu, R., Zhang, X. Peer-to-peer access control architecture using trusted computing technology, in *Proceedings of the 10th ACM symposium on Access control models and technologies*, Sweden, 2005. ACM Press: 147–158.

[54] Chakraborty, S., Ray, I. TrustBAC: integrating trust relationships into the RBAC model for access control in open systems, in *Proceedings of the 11th*

ACM symposium on Access control models and technologies, California, USA, 2006. ACM Press: 49–58.

[55] Zhang, Y., Lin, K., Hsu, J. Accountability monitoring and reasoning in service-oriented architectures, in *Journal of Service Oriented Computing and Applications*, **1**(1); 2007. Springer: 35–50.

[56] Rochford, K., Coghlan, B., Walsh, J. An agent-based approach to grid service monitoring, in *Proceedings of the 5th International Symposium on Parallel and Distributed Computing*, Timisoara, Romania, 2006. IEEE CS Press: 345–351.

[57] Spanoudakis, G., Mahbub, K. Requirements monitoring for service–based systems: towards a framework based on event calculus, in *Proceedings of the 19th International Conference on Automated Software Engineering*, Linz, Austria, 2004. IEEE CS Press: 379–384.

[58] Letia, T., Marginean, A., Groza, A. Z-based agents for service-oriented computing, in *Proceedings of the Service-Oriented Computing: Agents, Semantics, and Engineering (LNCS v4504)*, Honolulu, HI, USA, 2007. Springer: 160–174.

[59] FIPA in `http://www.fipa.org/` (November 2007)

[60] Yan, Y., Cordier, M-O., Pencole, Y., Grastien, A. Monitoring Web service networks in a model-based approach, in *Proceedings of the 3rd European Conference on Web Services*, Växjö, Sweden, 2005. IEEE CS Press: 192–203.

[61] Mao, H., Hunag, L., Li, M. Service-based grid resource monitoring with common information model, in *Proceedings of the IFIP International Conference on*

Network and Parallel Computing (LNCS v3779), Beijing, China, 2005. Springer Berlin / Heidelberg: 80–83.

[62] Jurca, R., Faltings, B., Binder, W. Reliable QoS monitoring based on client feedback, in *Proceedings of the 16th International Conference on World Wide Web*, Banff, Canada, 2007. ACM Press: 1003–1012.

[63] Baresi, L., Ghezzi, C., Guinea, S. Smart monitors for composed services, in *Proceedings of the 2nd International Conference on Service-Oriented Computing*, New York, USA, 2004. ACM Press: 193–202.

[64] Sahai, A., Machiraju, V., Wursterl, K. Monitoring and controlling internet-based e-services, in *Proceedings of the 2nd IEEE Workshop on Internet Applications*, San Jose, CA, 2001. IEEE CS Press: 41–48.

[65] Peng, L., Koh, M., Song, J., See, S. Grid service monitoring for grid market framework, in *Proceedings of the 14th IEEE International Conference on Networks*, Singapore, 2006. IEEE CS Press: 1–6.

[66] Kelly, N., Jithesh, P., Donachy, P., Harmer, J., Perrott, R., McCurley, M., Townsley, M., Johnston, J., McKee, S. GeneGrid: a commercial grid service-oriented virtual bioinformatics laboratory, in *Proceedings of the 2005 IEEE International Conference on Services Computing*, USA 2005. IEEE CS Press: 43–50.

[67] Deng, Y., Wang, F. A heterogeneous storage grid enabled by grid service, in *ACM SIGOPS Operating Systems Review*, **41**(1); 2007. ACM Press: 7–13.

[68] Foster, I., Kesselman, C., Tuecke, S. The anatomy of the grid: enabling scalable virtual organizations, in *International Journal of High Performance Computing Applications*, **15**(3); 2001. 200–222.

[69] Hwang, J., Lee, C., Kim, S. Trust embedded grid system for the harmonization of practical requirements, in *Proceedings of the 2005 IEEE International Conference on Services Computing*, Orlando, USA, 2005. IEEE CS Press: 51–60.

[70] Durad, M., Cao, Y. A vision for trust managment grid, in *Proceedings of the 6th IEEE International Symposium on Cluster Computing and the Grid*, Singapore, 2006. IEEE CS Press: 34–44.

[71] Foster, I., Jennings, N., Kesselman, C. Brain meets brawn: Why grid and agents need each other, in *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems*, New York, USA, 2004. IEEE CS Press: 8–15.

[72] Górski, J., Jarzebowicz, A., Miler, J., Olszewski, M. Trust case: justifying trust in an IT solution, in *Reliability Engineering & System Safety: Elsevier*, **89**(1); 2004. 33–47.

[73] OMG Management Group. UML^{TM} profile for modeling quality of service and fault tolerance characteristics and mechanisms, in *Technical Report*, 2004.

[74] Vraalsen, F., Lund, M., Mahler, T., Stlen, K. Specifying legal risk scenarios using the CORAS threat modelling language, in *Proceedings of the 3rd Internatioal Conference on Trust Management (LNCS v3477)*, France, 2005. Springer: 45–60.

[75] Jürgens, J. Secure system development with UML, Springer-Verlag, 2003.

[76] Selic, B., Rumbaugh, J. Using UML for modeling complex real-time systems, Rational (ObjecTime), March 1998.

[77] EUROSEC GmbH Chiffriertechnik & Sicherheit, Secure programming in PHP, in *Secologic Project*, 2005.

[78] Bezroutchko, A. Secure file upload in PHP web applications, in `http://www.scanit.be/uploads/php-file-upload.pdf`, 2007. (August 2007)

[79] File upload. in `http://ikiwiki.info/todo/fileupload/` (August 2007)

[80] Snyder, C., Southwell, M. Preventing SQL injection in *Pro PHP Security*, SpringerLink: 249–261.

[81] Hussein, M., Zulkernine, M. UMLintr: a UML profile for specifying intrusions, in *Proceedings of the 13th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems*, Potsdam, Germany, 2006. IEEE CS Press: 279–286.

[82] Grandison, T., Sloman, M. A survey of trust in internet application, in *IEEE Communications Surveys & Tutorials*, **3**(4); September 2000. 15pp.

[83] Bellifemine, F., Caire, G., Poggi, A., Rimassa, G. Jade: A white paper, in *EXP in Search of Innovation*, **3**(3); 2003. 14pp.

[84] Jade - Java Agent Development Framework in `http://jade.tilab.com/` (November 2007)

[85] Eclipse. `http://www.eclipse.org/` (November 2007).

[86] MySQL AB. `http://www.mysql.com/` (November 2007).

[87] Jade Primer and Tutorial. `http://www.iro.umontreal.ca/~vaucher/Agents/Jade/JadePrimer.html` (December 2007).

[88] W3C Recommendation. Extensible Markup Language (XML) 1.0 (second edition), in *Technical Report*, October 2000.

[89] Kolšek, M. Session fixation vulnerability in web-based applications, in *ACROS Security*, December 2002. available in `http://www.acros.si/papers/session_fixation.pdf` (December 2007)

[90] Damon, M. Doug, S., Dirk, G. A mechanism for detecting and responding to misbehaving nodes in wireless networks, in *Proceedings of the 4th IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks*, California, USA, 2007. IEEE CS Press: 678–684.

[91] Douceur, J. The sybil attack, in *Proceedings of the 1st Int Workshop on Peer-to-Peer Systems (LNCS v2429)*, Cambridge, MA, USA, 2002. Springer: 251–260.

[92] McKnight, D., Chervany, N.L. The meanings of trust, in *Technical Report MISRC Working Paper Series 96-04*, University of Minnesota, Management Information Systems Research Center, 1996.

[93] Grandison, T.W. Conceptions of trust: Definitions, constructs and models, in *Trust in E-Services: Technologies, Practices and Challenges* (Rongong Song ed.), Chapter 1, 2006. IDEA Group Inc: 1–28.

[94] Marsh, S. Formalizing trust as a computational concept, in *PhD Thesis*, Department of Computer Science and Mathematics, University of Sterling, 1994.

Appendix A

XML Files

A.1 ServiceDescriptor.xml

```
<?xml version = "1.0" encoding = "UTF-8"?>
<TargetSystem name = "ServiceProvider" id = "sp1">
  <ProvidedServices>
    <ProvidedService provides = "UploadDocFile"
      service-params = "fileName, fileSize, fileType, fileContents">
    </ProvidedService>
    ...
  </ProvidedServices>
  <ServiceConstraints>
    <ServiceConstraint>
     <service>UploadDocFile</service>
     <maxPOST_Val>100</maxPOST_Val>
     <maxPOST_Type>MB</maxPOST_Type>
    </ServiceConstraint>
    ...
  </ServiceConstraints>
</TargetSystem>
```

A.2 ServiceTrustContext.xml

```
<?xml version = "1.0" encoding = "UTF-8"?>
<ServiceTrustContexts>
    <Service>
        <Name>UploadDocFile</Name>
        <Thresholds>
            <Threshold>
                <Name>Interaction-Threshold</Name>
                <val>0.52</val>
            </Threshold>
            <Threshold>
                <Name>Rec-Threshold</Name>
                <val>0.50</val>
            </Threshold>
        </Thresholds>
        <trust_rules_UploadDocFile>
            <trust_rule_UploadDocFile>
                <Name>FileExcess</Name>
                <importance>MEDIUM</importance>
                <category>disbelief</category>
            </trust_rule_UploadDocFile>
            <trust_rule_UploadDocFile>
                <Name>UploadCompletion</Name>
                <importance>HIGH</importance>
                <category>belief</category>
            </trust_rule_UploadDocFile>
        </trust_rules_UploadDocFile>
    </Service>
    ...
</ServiceTrustContexts>
```

A.3 MonitorConfigurations.xml

```
<?xml version ="1.0" encoding = "UTF-8"?>
<TargetSystem name = "ServiceProvider" id = "sp1">
 <RecommenderList>
    <Recommender>
        <Name>ServiceProvider</Name>
        <Recommender-id>sp2</Recommender-id>
    </Recommender>
    <Recommender>
        <Name>ServiceProvider</Name>
        <Recommender-id>sp3</Recommender-id>
    </Recommender>
    <Recommender>
        <Name>ServiceProvider</Name>
        <Recommender-id>sp4</Recommender-id>
    </Recommender>
 </RecommenderList>
 <Constants>
    <Constant>
        <EquationConstant>wb</EquationConstant>
        <val>0.8</val>
    </Constant>
    <Constant>
        <EquationConstant>delta</EquationConstant>
        <val>0.8</val>
    </Constant>
    <Constant>
        <EquationConstant>Upsilon</EquationConstant>
        <val>0.2</val>
    </Constant>
    <Constant>
        <EquationConstant>eta</EquationConstant>
        <val>1</val>
    </Constant>
    <Constant>
```

```
        <EquationConstant>Lambda</EquationConstant>
        <val>10</val>
    </Constant>
    <Constant>
        <EquationConstant>Psi</EquationConstant>
        <val>0.2</val>
    </Constant>
    <Constant>
        <EquationConstant>zeta</EquationConstant>
        <val>0.8</val>
    </Constant>
    <Constant>
        <EquationConstant>alpha</EquationConstant>
        <val>0.8</val>
    </Constant>
 </Constants>
</TargetSystem>
```