Chapter 3

A Trust Monitoring Architecture for Service-Based Software

Mohammad Gias Uddin* and Mohammad Zulkernine**

* Dept. of Electrical and Computer Engineering, Queen's University,

Kingston, Canada K7L 3N6. Email: gias@cs.queensu.ca

** School of Computing, Queen's University,

Kingston, Canada K7L 3N6. Email: mzulker@cs.queensu.ca

Abstract. Service-based software can be misused by potentially untrustworthy service requestors while providing services. A service-based system is usually dynamic due to mutual collaboration among stakeholders to achieve goals, perform tasks and manage resources. However, it lacks the presence of a central authority to monitor the trustworthiness of service users. In this chapter, we propose a trust monitoring architecture, called TrAM (**Tr**ust **A**rchitecture for **M**onitoring) to monitor the trustworthiness of service users at run-time, facilitating the analysis of interactions from trust perspectives. Monitoring allows the enforcement of corrective actions that may protect the software by mitigating major unwanted incidents. The performance of the architecture has been evaluated by monitoring a prototype file-sharing grid.

1. Introduction

In service-based software systems, stakeholders are scattered across different organizational domains, and they can join and leave the systems at any time. A service-based system usually operates through spontaneous interactions with limited reliance on a specific central control authority. This inherent nature of decentralization introduces security concerns as software may be exploited by potentially untrustworthy stakeholders on whom the software has minimal or no control. Uncertainty is prevalent due to its open nature, so it may not be always sufficient to use *'hard security'* mechanisms to protect services from malicious and unwanted incidents. For example, illegal access to resources can be avoided using access control mechanisms. However, a malicious user with access to system resources from several administrative boundaries can still use different services that may provide that user with ample opportunities to break into the system. Given that, a trust monitoring architecture is necessary for the run-time analysis of the services based on the trustworthiness of the service requesters.

Trust is considered as ‘soft security’ and it 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” [1, 2]. Trust incorporates risk analysis to examine potential risks or opportunities the interactions may invite to the total system. In this chapter, we present a monitoring architecture for analyzing service interactions from trust perspectives by identifying the contexts of trust concerns in trust rules that are prevalent in such interactions. A trust rule snapshots system events encapsulating a service outcome that is crucial to the target system from trust perspectives [6]. The proposed architecture is called TrAM (**Tr**ust **A**rchitecture for **M**onitoring), and it may reside in each service providing software. The architecture allows the analysis of the trustworthiness of users based on trust rules and calculation schemes [6, 7]. A service requestor is penalized for the violation of trust rules and rewarded for no such violations, which thus facilitates the quantification of the trustworthiness of the corresponding entities. Collaborative decision making is introduced by incorporating the recommendations from similar service providers. The performance overhead of the architecture has been evaluated based on the monitoring of a prototype trust-aware file-sharing grid.

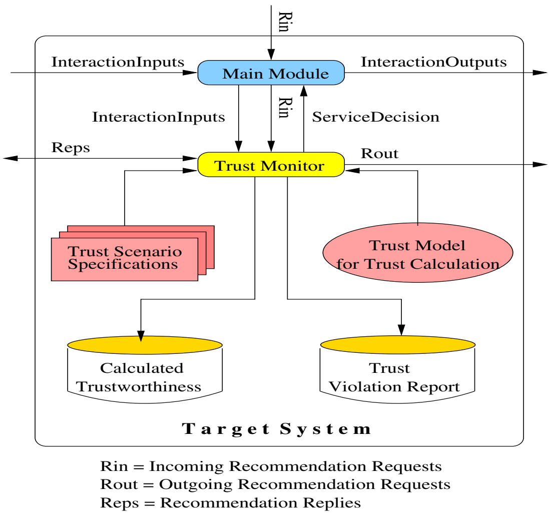

Rin
InteractionInputs
Main Module
InteractionOutputs
InteractionInputs
Rin
ServiceDecision
Reps
Trust Monitor
Rout
Trust Scenario Specifications
Trust Model for Trust Calculation
Calculated Trustworthiness
Trust Violation Report
Target System
Rin = Incoming Recommendation Requests
Rout = Outgoing Recommendation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

Fig. 1. Working environment of the monitoring architecture

Fig. 1 presents an overview of the trust monitoring architecture, where the target system is any service provider. An interaction is initiated when a service user requests a service. The events received from requestors by the Main Module of a provider are called *InteractionInputs*. The provider uses the Trust Monitor to analyze the interactions with the requestors which are forwarded to it by the Main Module. For a service request, the Trust Monitor provides a decision (*ServiceDe-*

cision) on whether to grant the service or not. Upon the granting of services, the monitor analyzes interaction events related to the corresponding session based on trust scenario specifications represented 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. The Main Module receives incoming recommendation requests (*Rin*) from other service 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 calculated trust values are stored in the repositories. Alert reports are generated and logged for any violation of a trust rule.

The rest of the chapter is organized as follows. The monitoring architecture is described in detail in Section 2. Section 3 provides the implementation and evaluation. In Section 4, the proposed architecture is compared and contrasted with the related work. Section 5 identifies the limitations and future research directions.

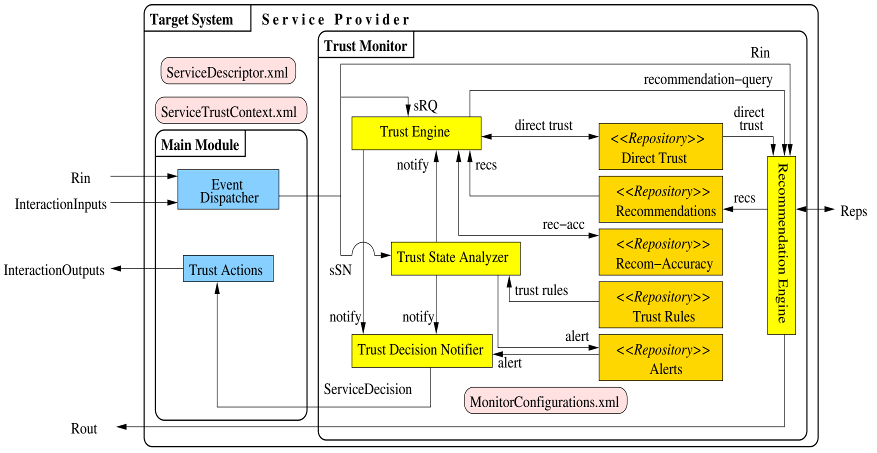
2. TrAM: The Trust Architecture for Monitoring

TrAM (Trust Architecture for Monitoring) is composed of a number of modules to analyze and calculate the trustworthiness of stakeholders and make trust-based run-time decisions. The architecture is presented in Fig. 2, and the modules and the related entities are described in detail in this section.

InteractionInputs are service request events (*sRQ*) or service session events (*sSN*). Upon the granting of a service to a user by a provider, a service session is initiated, during which the user 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 receives the *sRQ* and the *sSN* as primary inputs. A provider makes a recommendation request (*rRQ*) to other providers about the requestor and receives recommendation replies (*rRP*) from other providers. The secondary inputs to the Event Dispatcher are the recommendation requests from other service providers through *Rin*.

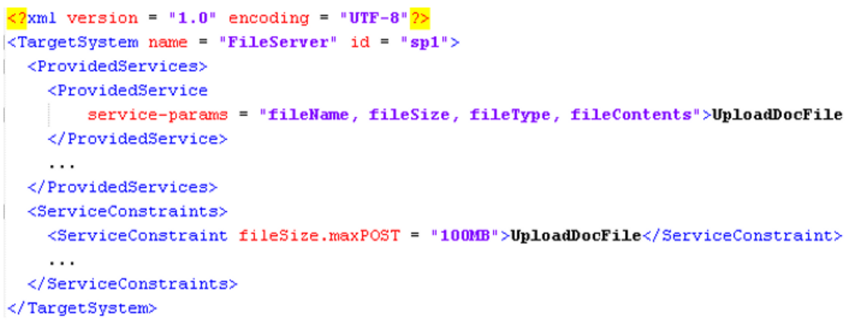
The *sRQ*s are forwarded to the Trust Engine, and *sSN*s to the Trust State Analyzer of the Trust Monitor. The *Rin*s are forwarded to the Recommendation Engine, from where *rRQ*s are sent as *Rout* to other providers. The replies to recommendat- ion requests (*rRP*) are received and sent by the Recommendation Engine through *Reps*. The Trust Engine provides decisions to grant or reject service request (*i.e.*, *sRQ*s), while the Trust State Analyzer checks *sSN*s against possible trust rules and provides decisions based on the state of the risk outcomes of the interaction. 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 service replies are provided as *sRP*s through *InteractionOutputs* to requestors. 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 Fig. 3. The target system is a provider with ID *sp1*, 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*. *sp1* employs constraints on this service which users 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 corresponding `ServiceConstraints` in the corresponding risk state space construction.



Reps = recommendation replies, recs = recommendation values
Rin = incoming recommendation requests, Rout = outgoing recommendations requests
rec – acc = recommendation – accuracy, sRQ = service request event, sSN = service session event

Fig. 2. TrAM : Trust Architecture for Monitoring



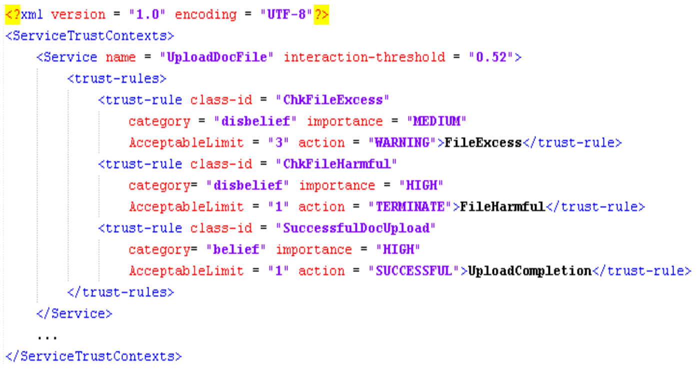
```
<?xml version = "1.0" encoding = "UTF-8"?>
<TargetSystem name = "FileServer" id = "sp1">
  <ProvidedServices>
    <ProvidedService
        service-params = "fileName, fileSize, fileType, fileContents">UploadDocFile
    </ProvidedService>
    ...
  </ProvidedServices>
  <ServiceConstraints>
    <ServiceConstraint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

Fig. 3. A snippet of ServiceDescriptor.xml

A snippet of `ServiceTrustContext.xml` is presented in Fig. 4. The `interac-tion- threshold` is used to denote the minimum trust value necessary for a user to be of fered the service; in this case, if the user has a previous total trust value greater than or equal to 0.52, the *UploadDocFile* service will be granted to the user. The trust rules to analyze the *UploadDocFile* service are "`FileExcess`", "`FileHarmful`" and "`UploadCompletion`". The `class-id`s in each of the trust rules designate the corresponding module used to deploy the corresponding trust rule. The `FileExcess` trust rule checks whether the uploaded file meets the server maximum file size constraint (*i.e.*, `fileSize.maxPOST`). A user may accidentally try to upload such large file once or twice. However, if the user executes such attempts beyond an acceptable limit, it surely is untrustworthy and should be considered carefully before granting any further uploading service. If this rule is violated, the requestor is penalized by a *disbelief* value of *medium* as denoted by `category` and `importance` respectively. The `AccpetableLimit` of such misbehavior is 3, *i.e.*, the user will be warned (`action = "WARNING"`) for such misbehavior up to three times, after which the service will not offered to the corresponding user anymore for the particular interaction[1]. The "`FileHarmful`" trust rule examines the uploaded file for any harmful contents (e.g., virus-infected file or the presence of any objectionable contents in the *fileContents*) and has `importance` value set as *high* with `action` as *terminate* and `AcceptableLimit` as `1`, interpreted as follows*:* the service offering of uploading doc file will be terminated (`action = "TERMINATE"`) to the corresponding user as soon as (`AcceptableLimit = "1"`) the uploaded file is detected as harmful, and also the user will be penalized a disbelief value (`category = "disbelief"`) of high for such misbehavior (`importance = "high"`). The "`UploadCompletion"` trust rule checks for the successful completion of the service. This trust rule is not violated if the user uploads files maintaining all the service constraints; that is, if the `FileExcess` and the `FileHarmful` trust rules are not violated. If this trust rule (*i.e.*, `UploadCompletion`) is not violated, the requestor is awarded a belief (`category = "belief"`) value of high (`importance = "high"`). Moreover, the interaction with the user will be considered as trustworthy as soon as (`AcceptableLimit = "1"`) the user uploads legitimate doc file, and will be sent a notification of successful interaction (`action = "SUCCESSFUL"`).

The Main Module has two parts: Event Dispatcher and Trust Actions. All the incoming events are received by the Event Dispatcher and forwarded to the different modules of the Trust Monitor. The incoming events to the Event Dispatcher are of three types: service requests (*sRQ*), service sessions (*sSN*), and recommendation requests (*rRQ*). Upon receipt of an event, the module delegates a *sRQ* to the Trust Engine, *sSN* to the trust state analyzer, and *rRQ* to the Recommendation Engine of the Trust Monitor. The Trust Actions module provides service replies to requestors by following the *ServiceDecisions* obtained from the Trust Decision Notifier of the Trust Monitor. For example, based on a *sRQ*, the module can offer or reject services, or based on an *sSN*, the module can terminate an unsatisfactory interaction.

[1] The values of different attributes and constants will depend on the corresponding target system.



```xml
<?xml version = "1.0" encoding = "UTF-8"?>
<ServiceTrustContexts>
    <Service name = "UploadDocFile" interaction-threshold = "0.52">
        <trust-rules>
            <trust-rule class-id = "ChkFileExcess"
                category = "disbelief" importance = "MEDIUM"
                AcceptableLimit = "3" action = "WARNING">FileExcess</trust-rule>
            <trust-rule class-id = "ChkFileHarmful"
                category= "disbelief" importance = "HIGH"
                AcceptableLimit = "1" action = "TERMINATE">FileHarmful</trust-rule>
            <trust-rule class-id = "SuccessfulDocUpload"
                category= "belief" importance = "HIGH"
                AcceptableLimit = "1" action = "SUCCESSFUL">UploadCompletion</trust-rule>
        </trust-rules>
    </Service>
    ...
</ServiceTrustContexts>
```

Fig. 4. A snippet of ServiceTrustContext.xml

The Trust Monitor analyzes interactions, calculates the trustworthiness of the interacting entities and makes decisions. It has four basic sub-modules: the Trust State Analyzer, the Trust Engine, the Recommendation Engine, and the Trust Decision Notifier. The sub-modules are discussed in the following subsections. To substantiate these discussions, we show three most prevalent scenarios using sequence diagrams: user requesting a service (Fig. 5), user violating a trust rule (Fig. 6), and user performing a trustworthy interaction (Fig. 7).

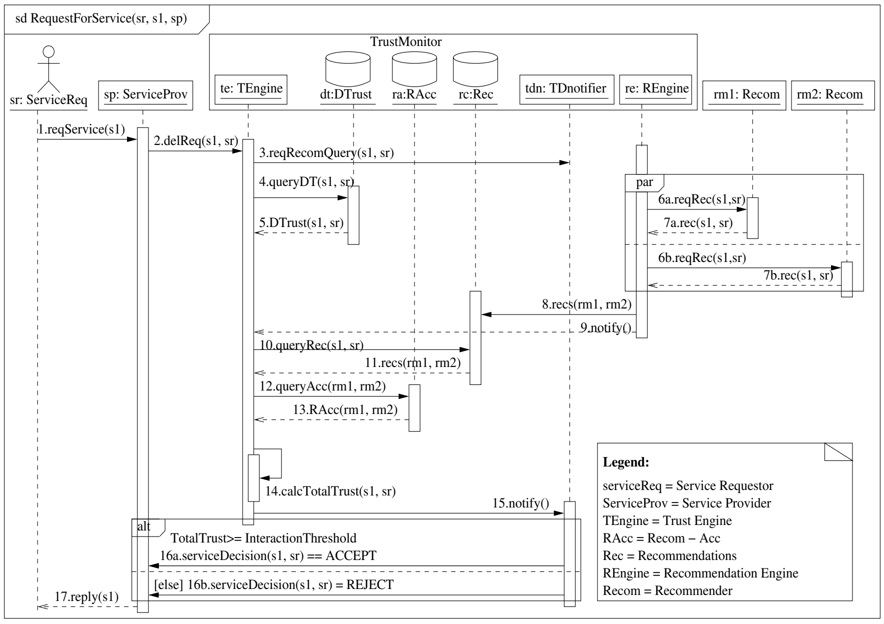


Fig. 5. Sequence diagram for a service request

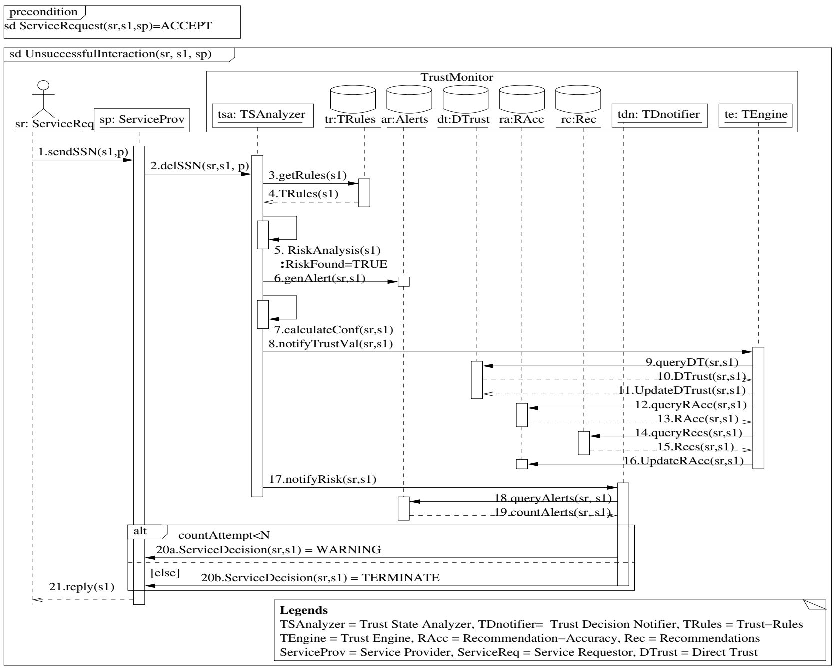


Fig. 6. Sequence diagram for a user violating a trust rule

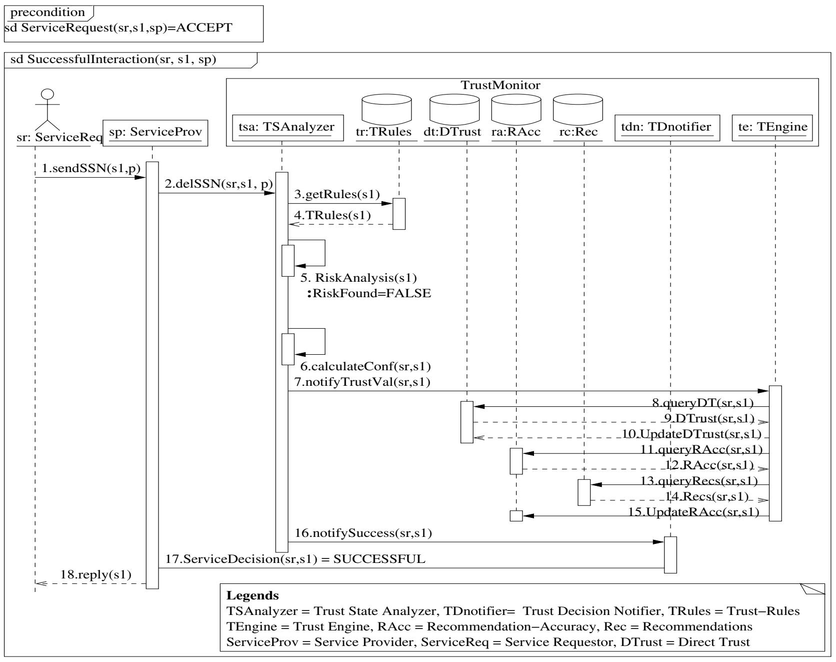


Fig. 7. Sequence diagram for a user performing trustworthy interactions

2.1 Trust State Analyzer

This module constructs trust-based risk state space to analyze service session events (*sSN*s) using trust rules from the Trust Rules repository. Upon the arrival of a service session 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. Whenever a trust rule is violated, this module generates an alert in the `Alerts` repository. An alert has the form $\{sr, s_i, r, sID, t_{alert}\}$, where *sr* is the requestor, s_i is the requested service, *r* is the trust rule that is being violated, *sID* is the ID of the session in which the violation was detected, and t_{alert} is the time when alert was generated based on the identification of potential risks in the corresponding interactions.. 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 potential risk-state info in the interaction status and makes decisions accordingly. The Trust Engine is notified of a confidence value only when a potential risk state is confirmed through the convergence to a confirmed untrustworthy state. However, the user is warned each time a potential risk is found in the corresponding interaction that is deemed as suspicious but requires further analysis. The Trust Decision Notifier is notified of any potential risk state information. In addition, the Trust Decision Notifier is notified of any successful interaction with service users. The total belief (I_b) (range [0, 1]) of provider *E1* on requestor *E2* for service s_i at time *t* from interaction *I* (i.e., *sRQ*) is calculated using Eq. 1, where *B(rn)* contains the belief value of the trust rule indexed as *rn*, and n_b is the total number of trust rule(s) related to belief outcome(s)[2]. Similarly, total disbelief I_d (range [0, 1]) is calculated using Eq. 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 outcome. The confidence (μ) (range [0, 1]) of *E1* on *E2* about service s_i is calculated using Eq. 3 (w_b (range [0, 1]) as the weight assigned to I_b.)

$$I_b(E1, E2, s_i, t) = \frac{d(E1, E2, s_i, t)}{n_d}, where\ \ d(E1, E2, s_i, t) = \sum_{rn=0}^{n_d} B(rn) \tag{1}$$

$$I_b(E1, E2, s, t) = \frac{b(E1, E2, s, t)}{n_b}, where\ \ b(E1, E2, s, t) = \sum_{rn=0}^{n_b} B(rn) \tag{2}$$

$$\mu(E1, E2, s_i, t) = w_b I_b(E1, E2, s_i, t) + (1 - w_b) I_d(E1, E2, s_i, t) \tag{3}$$

[2] For the sake of simplicity, we denote both the provider and the requestor as entities (E).

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 that are used by the Trust Decision Notifier 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. This new direct trust is the compared against the previous recommendations from the `Recommendations` repository that were used to make service granting decision to the corresponding service user. The comparison facilitates the understanding of the provider on the accuracies of the corresponding recommendations in its decision making phases. The recommendation accuracies are stored in the `Recom-Accuracy` repository. If the measured accuracy falls below a pre-determined recommendation-accuracy accuracy threshold, the corresponding recommendation is considered as unreliable for the particular type of interactions. However, it should be noted that the measure of recommendation accuracy is not used as determining the trustworthiness of the corresponding recommenders; rather its purpose is to identify the reliability of recommendations in a particular provider decision state space. In following the context-awareness nature of trust [7] that "a recommender *r1* may not be reliable to a provider *sp1* for a service s_i, but it may still be considered as reliable for another service s_j ($i \neq j$)" and "based on the deployment of a recommendation accuracy in different providers, a recommender *r1* may considered as unreliable in provider *sp1* for service s_i, but it may still be regarded as reliable in the provider *sp2* for the same service". The direct trust T_D (range [0, 1]) of *E1* on *E2* for service s_i at time *t* is calculated using Eq. (4), where δ (range [0, 1]) is a weighting factor. The value of T_D thus changes after each interaction based on the outcome of the interaction.

$$T_D(E1, E2, s_i, t) = \delta T_D(E1, E2, s_i, t-1) + (1-\delta)\mu(E1, E2, s_i, t) \tag{4}$$

The accuracy (*A*) (range [0, 1]) of a recommender *E3* in providing a recommendation to a provider *E1* about requestor *E2* regarding service s_i is calculated using Eq. 5, where $\Delta R(E3,E1,E2, s_i, t)$ calculates the difference between the provided recommendation and the calculated direct trust. The calculation of recommendation-accuracy follows [8], but tailored to service attributes in TrAM. $R(E3,E1,E2, s_i, t)$ denotes the recommendation value provided by *E3* to *E1* about *E2* regarding service s_i at time *t*. Each provider keeps an accuracy table (*AT*) in the `Recom-Accuracy` repository, where it updates the accuracy of every recommendation after the corresponding interaction. The accuracy of *E3* to *E1* about *E2* regarding service s_i at time *t* in the *AT* is denoted by $AT\ (E3, E1, E2, s_i, t)$. The update in the *AT* is performed using Eq. 6 by considering previous recommendation accuracy (*AT (E3, E1, E2,* s_i*, t−1)*) and new recommendation accuracy (*A (E3, E1, E2,* s_i*, t)*). ζ (Range [1, 0]) weights the importance of previous and current accuracies. Using Eq. 5, and 6, unreliable recommendations are detected. A recommender is considered as most reliable with accuracy 1 and most unreliable with accuracy 0.

$$\begin{aligned} &A(E3,E1,E2,s_i,t) = 1 - \Delta R(E3,E1,E2,s_i,t), \\ &where, \nabla R(E1,E2,s_i,t) = | R(E3,E1,E2,s_i,t) - T_D(E1,E2,s_i,t) | \end{aligned} \quad (5)$$

$$AT(E3,E1,E2,si,t) = \zeta AT(E3,E1,E2,s_i,t-1) + (1-\zeta)A(E3,E1,E2,s_i,t) \quad (6)$$

The calculation of total trust is a function of direct trust, recommendation and recommendation-accuracy [7,8], and is used to make the trust-based service granting decision for an *sRQ*. The Trust Decision Notifier is informed of this trust value.

2.3 Recommendation Engine

This module provides a recommendation reply (*rRP)* in response to a recommendation request (*rRQ),* receives recommendations from other providers, and stores the recommendation values in the `Recommendations` repository. A recommendation value is at most equal to the corresponding direct trust value to avoid any overstating about users in the system [8]. For example, if a provider has a direct trust value of 0.8 on a user about a particular service, it should provide a recommendation value no greater than 0.8.

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 service request is granted if the calculated total trust value from the Trust Engine is at least equal to the interaction threshold of the requested service, otherwise the request is rejected. A *ServiceDecision* is constructed as {*sr,* s_i*,* `Accept`*, t*}, if the request for service s_i from requestor *sr* is accepted at time *t*, or as {*sr, s,*`Reject`*, t*} if it is rejected. Based on the notification of any potential risk outcome in an interaction, such as the detection of file uploading beyond the server allowed maximum file size using the `FileExcess` trust rule, the Trust Decision Notifier queries the `Alerts` database to determine the total number of such misbehavior from the corresponding user for the particular service. This total number is then compared against sever allowed such maximum attempts (*i.e.*, `AcceptableLimit`) in `ServiceTrustContext.xml` (recall Fig. 4). If the total number of such misbehavior falls below the acceptable limit, the Trust Decision Notifier constructs a *ServiceDecision* as {*sr, s_i, Unsatisfactory,* `WARNING`, `FileExcess`, *sID, t*} to give warning to the corresponding requestor *sr* of the unsatisfactory interaction between them, but continues to offer the s_i service to the user (*i.e., UploadDocFile*). Here, *sr* is service- user; *sID* is the ID of the corresponding session that was initiated between the provider and requestor for the service s_i. However, if the number of such attempts reaches the acceptable limit, the interaction with the user for the particular service usage is terminated by providing a *ServiceDecision* as {*sr, s_i, Unsatisfactory,* `TERMINATE`, `FileExcess`, *sID, t*}. A *ServiceDecision* is constructed as {*sr, s, Satisfactory, sID, t*} if the interaction was successful without violating any trust rules.

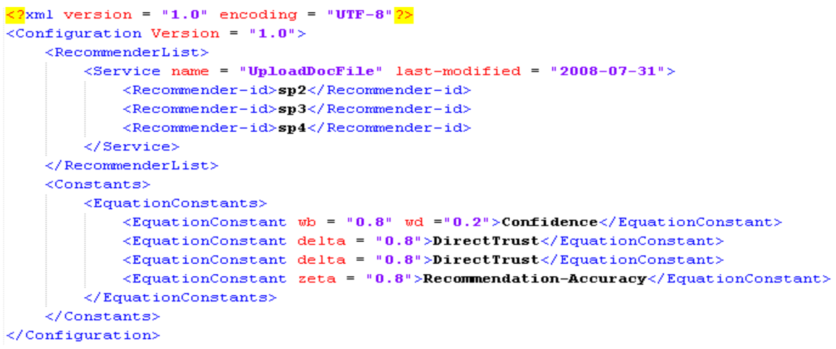

```
<?xml version = "1.0" encoding = "UTF-8"?>
<Configuration Version = "1.0">
    <RecommenderList>
        <Service name = "UploadDocFile" last-modified = "2008-07-31">
            <Recommender-id>sp2</Recommender-id>
            <Recommender-id>sp3</Recommender-id>
            <Recommender-id>sp4</Recommender-id>
        </Service>
    </RecommenderList>
    <Constants>
        <EquationConstants>
            <EquationConstant wb = "0.8" wd ="0.2">Confidence</EquationConstant>
            <EquationConstant delta = "0.8">DirectTrust</EquationConstant>
            <EquationConstant delta = "0.8">DirectTrust</EquationConstant>
            <EquationConstant zeta = "0.8">Recommendation-Accuracy</EquationConstant>
        </EquationConstants>
    </Constants>
</Configuration>
```

Fig. 8. A snippet of MonitorConfigurations.xml

The `MonitorConfigurations.xml` file denotes the list of recommenders and the constant values used in the trust equations. A snippet of the `MonitorConfigurati ions.xml` is provided in Fig. 8. The `RecommenderList` tag shows the list of recommenders to whom *sp1* asks for recommendations for a particular service, such as *UploadDocFile*. The provider continuously refreshes its database to update the list of such recommenders (as identified by `last-modified`) and identifies the recommenders by their IDs in the system, such as *sp2, sp3* and *sp4*. The `Constants` tag includes the constant values used in the trust calculations. For example, the value of w_b in calculating confidence (Eq. 3) is 0.8.

3. Implementation and Experimental Evaluation

We develop a prototype file sharing grid [9] in Jade (Java Agent Development Environment) [10], by focusing on three types of file sharing services: file upload, open and search. The trust scenarios are modeled using UMLtrust [6] and converted to trust rules. The `Trust Rules` repository is developed based on the different trust scenarios [6] (see Table 1). The other repositories (i.e., `Direct Trust`, `Recommendations`, `Recom-Accuracy`, and `Alerts`) are developed as database tables in MySQL 5.0 [11]. The providers and requestors are implemented as Jade agents. Events are generated by employing the `ACLMessage` (Agent Communication Language Message), with the different modules of the architecture as 'behaviours' of Jade. The summary of the implementation environment is provided as follows:

- System Configuration: Pentium 1.886 GHz Dell machine. 1 GB RAM.
- Development Languages: Java, XML, MySQL 5.0 [11].
- Development Platform: Jade 3.5 [10], Eclipse IDE 3.2.

The service providers employ the monitor to analyze interactions and decide accordingly. Experimental results show that the proposed architecture can analyze service-based interactions from trust perspectives, measure trustworthiness, and make automatic decisions. The performance of the service provider is measured while analyzing service session, service request, and recommendation requests events. However, the monitor creates some performance overhead also. The overheads discussed in the next subsections are of three types, delay in providing a decision on a service request event (*sRQ*), delay in analyzing a service session event (*sSN*), and delay in a long recommendation chain.

Table 1. Elicited trust scenarios for a file sharing server

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

3.1 Delay in Providing Decision on a Service Request Event (sRQ)

A provider retrieves previous direct trust value with the requestor for the service from the `Direct Trust` repository, and handles recommendations from other providers. The handling of recommendations includes the requests for recommendations and the receipt of the corresponding replies. In a service-based system without the trust monitor, these two tasks would not be present. The delay is calculated by taking the difference between the sending time of a service request event and the receipt time of the corresponding decision in a service reply. In the experimental setup, we use one service provider (*sp1*) to provide services, three providers (*sp2*, *sp3*, and *sp4*) for recommendations. We vary the number of service requests from 10 to 100 from a requestor *sr1*. We run the experiment for each setup 10 times and take the average to minimize errors. (see Fig. 9). 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 a *sRQ* introduces some delay in providing the corresponding service decision. However, the result is encouraging since the delay does not increase with the increase of the number of service requests, *i.e.*, scalability will not be an issue as the number of requesters grow. The response from a provider without the trust monitor requires almost constant 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 range of 500–600 milliseconds). The reason for this is as follows. All of the modules in our provider software are implemented as specific behaviors (*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 creates 100 instances of each working module. The creation times of these instances are almost constant, so the total time required to create 100 instances of working modules is almost the same as the time required to create 10 instances. Therefore, the execution is performed in parallel for each service request.

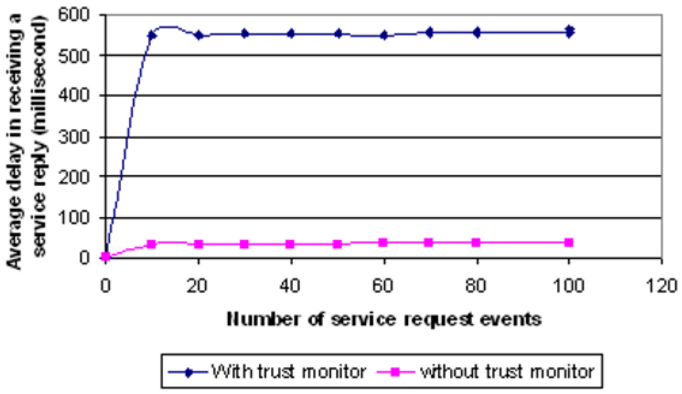


Fig. 9. The average delay in receiving a service reply for a service request

3.2 Delay in Analyzing a Service Session Event (sSN)

TrAM analyzes a service session (*sSN*) event in two steps: compare *sSN* against trust rules, and update direct trust, recommendation, and recommendation accuracies. The delay is calculated by taking the difference between the sending time of an *sSN* event and the corresponding reply time. To examine the overhead, we send a number of *sSN*s to the *sp1* varying from 10 to 100, where the requested service is *UploadDocFile*. Fig.10 provides the results which show that the analysis of an *sSN* introduces some delay; however, the delay remains almost constant with the increase in 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 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 Figs. 9 and 10 shows that the average delay is in the range of 500–600 milliseconds for *sRQ*, while it is in the range of 80–90 milliseconds for *sSN*, having the average difference as 420–520 milliseconds. The major difference in handling a *sRQ* and an *sSN* is the handling of recommendations.

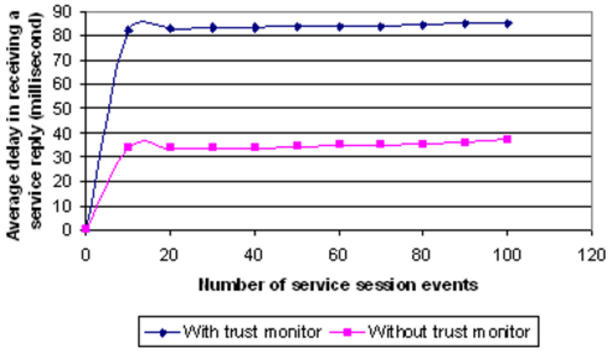


Fig. 10. The average delay in receiving a service reply for a service session

3.3 Delays in Long Recommendation Chains

Since recommendations are handled as sending of recommendation requests and receiving of the corresponding replies, there is some delay in providing runtime decisions on a *sRQ*. However, while analyzing the delay on a *sRQ*, we only considered direct recommendations (*i.e*, the recommendations from immediate neighbors [7]). Since we allow both direct and indirect recommendations in our system, we were interested to see the impact of handling recommendations with long chain, *i.e.*, when the path-length for indirect recommendations varies. To do this experiment, we assumed a long chain from *sp1* to *sp10*, where *sp1* asks *sp2* for recommendation, *sp2* to *sp3*, and so on. We further assumed that a provider can ask for recommendations to only one another provider, *i.e.*, *sp1* only to *sp2* and *sp2* only to *sp3*. We varied the number of recommenders from 1 to 9 (*i.e.*, *sp2* to *sp10*), and send recommendation requests from *sp1*to calculate the difference between a recommendation request and the corresponding reply. Fig. 11 presents the results which show that the delay in receiving a recommendation reply increases almost linearly with the increase of path-length in a chain.

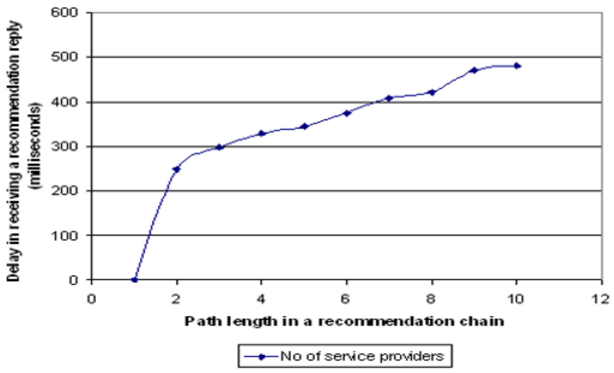


Fig. 11. The average delay in receiving a recommendation reply.

3.4 Monitoring Overhead

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 architecture applicable for large-scale systems. Therefore, the focus of the first two experiments was not to show the differences 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.

4. Related Work

Many trust-based approaches are proposed by focusing on trust-based policy management [3-5, 24-32]. The monitoring of the trustworthiness of service requestors has not been adequately addressed so far. English et al. [4] neither support any trust rules to perform automatic trust monitoring of service-based software nor they present any calculation schemes to quantify the trustworthiness of stakeholders. Trust-based spam detection [24] and reputation-based social network systems [25, 26] assume that the trust values are available, hence necessitating the incorporation of a monitor like ours in their systems. Since we establish dynamic trust relationships between service providers and requestors based on the automatic monitoring of service usage, our architecture can be applicable in any social network-based systems that require sharing of resources. Unlike our XML-based service and trust monitor configuration, facts from past interactions are used in [27, 28]. Some other trust-based access control mechanisms [3, 5, 29-32] define the syntax and semantics of the corresponding policy languages, and the deployment of the policy languages in the target system requires language-based parsers and compilers. We compare and contrast those work in Table 2.

Table 2. Trust-based monitoring approaches

Work	Domain	Trust Mechanism
TBRM [3]	Information system	Policy-based access control
SPM [4]	Ubiquitous system	Interaction-based system monitoring
TRBAC [5]	Information system	Policy-based access control
SureMsg [24]	Email services	Reputation-based email exchanging
EigenTrust [25]	Peer-to-peer system	Reputation-based trust negotiation
FuzzyTrust [26]	Semantic web	Social network-based trust formation
FuzzyWeb [27]	Web services	Rule-based service access
TAP [28]	Service-based software	Policy-based software access control
ICTM [29]	Information system	Policy-based access control
ATN [30]	Open grid system	Trust negotiation, access control
TBAC [31]	Information system	Policy-based access control
TrustBack [32]	Information system	Role-based access control
Our work	Service-based software	Interaction-based service monitoring

A number of monitoring architectures exist for service quality analysis and automatic service composition [12–23]. We quantify the trustworthiness of requestors by monitoring their service usages, while the existing architectures monitor service providers to improve service quality to end users. Table 3 summarizes those research with respect to our work.

Table 3. Selected work on service monitoring

Work	Monitored Attributes	Monitored Entity	Trust Quantification
FQoS [12]	Service quality in user feedback	Provider	N
AMR [13]	Service accountability in composition	Provider	N
SMC [14]	Errors in service execution	Provider	N
MSLA [15]	Service constraints for mutual safety	provider, user	N
RM [16]	Service quality using requirements	Provider	N
WSR [17]	Exceptions in web service for quality	Provider	N
ZAS [18]	Service timeliness, type checking	Provider	N
WSN [19]	Errors in service execution	Provider	N
AGSM [20]	Quality of service in grid	Provider	N
GSM [21]	Quality of service for end users	Provider	N
SGR [22]	Resource allocation status	Provider	N
IBS [23]	Quality in service execution	Provider	N
Our work	Service safety in trust concerns	Requestor	Y

5. Conclusions and Future Work

Due to the pervasiveness of software in our everyday activities, it is important to monitor trust relationships between the users and the system to analyze the vulnerabilities and opportunities the relationships may invite to the system. In this chapter, we present a trust monitoring architecture called TrAM, to automatically analyze service-based interactions from trust perspectives. TrAM employs trust rules to analyze such interactions and uses trust calculation schemes to quantify the trustworthiness of service users. TrAM not only makes run-time decision for service provision but also employs dynamic decision on the risk status of the service that may suggest the premature termination of an interaction to protect the corresponding stakeholders. The proposed architecture is implemented in a trust-aware file sharing grid and evaluated under different trust conditions and performance overhead related concerns. Our future enhancements to the system will concentrate on addressing the following limitations. While specifying a trust scenario, we assumed that the identity of a trustee is properly resolved. It was also assumed that the network is secure from false recommenders. However, in real situations, this might not be the case always.

Acknowledgments

This research is partially funded by the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

[1] Gambetta D (1988) Can we trust trust? In: Trust: Making and Breaking Cooperative Relations. Chapter 13. University of Oxford: 213–237.

[2] Yu B, Singh MP (2002) An evidential model of distributed reputation mechanism. In: Proc. of the 1st Intl. Joint Conf. on Autonomous Agents and multi-agent systems. Italy. ACM Press: 294–301.

[3] Lin C, Varadharajan V (2006) Trust based risk management for distributed system security - a new approach. In: Proc. of the 1st International Conference on Availability, Reliability and Security. Vienna, Austria. IEEE CS Press: 6–13.

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

[5] Dimmock N, Bacon J, Ingram D, Moody K (2005) Risk models for trust-based access control (TBAC). In: Proc. of the 3rd Annual Conference on Trust Management (LNCS v3477). France. Springer: 364–371.

[6] Uddin MG, Zulkernine M (2008) UMLtrust: Towards developing trust-aware software. In: Proc. of the 23rd ACM Symposium on Applied Computing. Brazil. ACM Press: 831–836.

[7] Uddin MG, Zulkernine M, Ahamed SI (2008) CAT: A context-aware trust model for open and dynamic systems. In: Proc. of the 23rd Annual ACM Symposium on Applied Computing. Fortaleza, Brazil. ACM Press: 2024–2029.

[8] Azzedin F, Maheswaran M (2003) Trust modeling for peer-to-peer based computing systems. In: Proc. of the International Symposium on Parallel and Distributed Processing. USA. IEEE CS Press: 10pp.

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

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

[11] MySQL 5.0 Reference Manual (2008). In: MySQL Enterprise Server.

[12] Jurca R, Faltings B, Binder W (2007) Reliable QoS monitoring based on client feedback. In: Proc. of the 16th Intl. Conference on World Wide Web. Canada. ACM Press: 1003–1012.

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

[14] Baresi L, Ghezzi C, Guinea S (2004) Smart monitors for composed services. In: Proc. of the 2nd International Conference on Service-Oriented Computing. USA. ACM Press: 193–202.

[15] Skene J, Skene A, Crampton J, Emmerich W (2007) The monitorability of service-level agreements for application-service provision. In: Proc. of the 6th International Workshop on Software and Performance. Buenos Aires, Argentina. ACM Press: 3–14.

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

[17] Robinson WN (2003) Monitoring web service requirements. In: Proc. of the 11th IEEE International Conference on Requirements Engineering. Japan. IEEE CS Press: 65–74.

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

[19] Yan Y, Cordier MO, Pencole Y, Grastien A (2005) Monitoring Web service networks in a model-based approach. In: Proc. of the 3rd European Conference on Web Services. Vaxj, Sweden. IEEE CS Press: 192–203.

[20] Rochford K, Coghlan B, Walsh J (2006) An agent-based approach to grid service monitoring. In: Proc. of the 5th Intl. Symposium on Parallel and Distributed Computing. Romania. IEEE CS Press: 345–351.

[21] Peng L, Koh M, Song J, See S (2006) Grid service monitoring for grid market framework. In: Proc. of the 14th IEEE International Conf. on Networks. Singapore. IEEE CS Press: 1–6.

[22] Mao H, Hunag L, Li M (2005) Service-based grid resource monitoring with common information model. In: Proc. of the IFIP International Conf on Network and Parallel Computing (LNCS v3779). Beijing, China. Springer: 80–83.

[23] Sahai A, Machiraju V, Wursterl K (2001) Monitoring and controlling internet-based e-services. In: Proc. of 2nd Workshop on Internet Applications. USA. IEEE CS Press: 41–48.

[24] Zhang W, Bi J, Wu J, Qin Z (2007) An approach to optimize local trust algorithm for SureMsg service. In: Proc. of the ECSIS Symposium on Bio-inspired, Learning, and Intelligent Systems for Security. Edinburgh, UK. IEEE CS Press: 51–54.

[25] Kamvar SD, Schlosser MT, Molina-Garcia H (2003) The eigentrust algorithm for reputation management in P2P networks. In: Proc. of the 12th International Conference on World Wide Web. Budapest, Hungary. ACM Press: 640–651.

[26] Lesani M, Bagheri S (2006) Applying and inferring fuzzy trust in semantic web social networks, in Proc. of the Canadian Semantic Web. Quebec City, Canada. Springer: 23–43.

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

[28] Rajbhandari S, Contes A, Rana OF, Deora V, Wootten I (2006) Trust assessment using provenance in service oriented applications. In: Proc. of the 10th IEEE on Intl. Enterprise Distributed Object Computing Conference Workshops. Hong Kong. IEEE CS Press: 65–72.

[29] Etalle S, Winsborough W (2005) Integrity constraints in trust management. In: Proc. of the 10th Symposium on Access Control Models and Technologies, Sweden. ACM Press: 1–10.

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

[31] Chakraborty S, Ray I (2006) TrustBAC: Integrating trust relationships into the RBAC model for access control in open systems. In: Proc. of the 11th ACM Symposium on Access Control Models and Technologies. California, USA. ACM Press: 49–58.

[32] Dimmock N, Belokosztolszki A, Eyers D, Bacon J, Ingram D, Moody K (2004) Using trust and risk in role-based access control policies. In: Proc. of the 9th ACM Symposium on Access Control Models and Technologies, New York, USA. ACM Press: 156–162.