

# Realizing S-Reliability for Services via Recovery-driven Intrusion Tolerance Mechanism

Quyen Nguyen<sup>1</sup> and Arun Sood<sup>1,2</sup>

<sup>1</sup>International Cyber Center and Department of Computer Science

George Mason University, Fairfax, VA 22030

<sup>2</sup>SCIT Labs, Clifton, VA 20124

{*qnguyeng@gmu.edu, asood@gmu.edu*}

## Abstract

*Service-Oriented Architecture (SOA) paradigm facilitates the design of large systems as a set of loosely coupled services interacting with each other. These services, in turn, can be combined to form a more complex service. But, for services to be useful, they must satisfy non-functional requirements, especially security-related quality of service. Unfortunately, software vulnerabilities expose these services to malicious actors, and make them susceptible to attacks. Therefore, security quality of service is critical in order to ensure confidentiality, integrity, and availability for system data and services. Due to the distributed and decentralized nature of services, publishing and guaranteeing security quality of service are crucial so that potential applications and clients can make use of the provided services. In this paper, we will first discuss how Intrusion Tolerance Quality of Service (IT-QoS) can be modeled and specified. Then, we will show how a recovery-driven intrusion tolerance architecture is able to ensure differentiated levels of S-Reliability, an important IT-QoS for a Service.*

## 1. Introduction

Many enterprises have adopted the SOA paradigm, since it facilitates the design of a large system as a set of loosely coupled services. Furthermore, these services can be combined to form more complex services. Web Service is one prominent implementation for a service that utilizes open standards, specifications and web protocols. The benefits of Web Services within an SOA system have to be balanced with the security challenges. As with any software system, a Service must satisfy security quality and other non-functional requirement to ensure availability, integrity, and confidentiality. To guarantee security quality is even more challenging for services which may be deployed in distributed network hosts within an enterprise network or over a wide-area network. Another trend is the utilization of services provided by Cloud Computing provided by entities external to an organization. Since a centralized quality

control cannot be done, we need QoS guarantees for the services participating in an application. Security attacks have become more sophisticated, so that a system cannot rely solely on intrusion prevention and detection for its security protection. We propose that intrusion tolerance systems (ITS) should be part of the solution for securing computer information systems. As distinct from the intrusion avoidance of current systems, ITS systems focus on containing the losses. Interestingly, as an emerging software architecture paradigm, SOA potentially poses challenges to the design of intrusion tolerance systems as recognized in [1].

The contribution of this paper is two-fold. First, we introduce the notion of Intrusion Tolerance QoS (IT-QoS) for which we propose a specification mechanism based on UML (Unified Modeling Language) [12]. For a service implemented as Web Service, adaptive application of the WSDL (Web Service Description Language) extension described in [2] is a candidate to describe IT-QoS. Second, we propose a new quality metric S-Reliability that measures reliability in the presence of malicious attacks. We show that SCIT (Self-Cleansing Intrusion Tolerance) architecture [4,5] can realize differentiated levels of S-Reliability, for a service or composite service. This work relies on the quantitative analysis results presented in our previous work [5].

This paper is organized as follows. In section 2, we summarize the work related to the aspects of QoS for SOA services, and intrusion tolerance systems. Then, in section 3, we briefly discuss the notion of IT-QoS, and how it can be modeled and publicized. Section 4 contains an overview of SCIT architecture. In section 5, we present the schemes for computing S-Reliability levels, which serve as a foundation for the delivery of services endowed with the assurance of these reliability levels. The paper ends with a conclusion and discussion of future work in section 6.

## 2. Related Work

The concept of Quality of Security Service (QoSS) was discussed in [3] as a new element of QoS for distributed systems. The authors illustrated the

usefulness of QoS and used different use case scenarios to show the different security levels required. Trade-off between security levels and cost due to computing and network resources is also examined.

Recently, there has been a lot of interest in specifying, managing and ensuring QoS for services in an SOA environment [2,14,15]. Most of the work in that area was mainly focused on performance, reliability, and availability of services. Some studies focused on the static modeling to provide a predetermined QoS set [14,15]. Dynamic approach using linear programming optimization to meet QoS constraints based on runtime operating environment variations was presented in [13]. Liu [6] introduced the concept of Quality of Information Assurance (QoIA) to address the non-functional requirements of intrusion tolerance for database transaction processing services. To support QoIA, the proposed architecture had two additional components: one for clients to subscribe different levels of QoIA, and the other for providers to adapt the data service in order to meet QoIA [6].

There has been extensive work done to analyze intrusion-tolerant system qualities in terms of survivability and dependability. In [7], Stroud et al. provided a qualitative analysis of the MAFTIA architecture using a case study of an Internet application to show how the MAFTIA services of the MAFTIA middleware make the system survivable and resilient in the face of malicious faults. While MAFTIA is a European project, SITAR was part of the DARPA funded program called OASIS (Organically Assured and Survivable Information Systems) [9]. The authors of [8] used transition state diagrams to study the behavior of the SITAR architecture; quantitative analysis was provided, which led to a closed form for the Mean-Time To Security Failure (MTTSF). Software rejuvenation was proposed in [10][10] to limit the impact of software faults and aging and increase cluster survivability. Sousa et al. [11] proposed a scheme that combines periodic system rejuvenation with a “reactive recovery”, which can be triggered when the perceived threat goes beyond a tolerable threshold.

### 3. Intrusion Tolerance QoS

*SOA and QoS.* Service is the building block of an SOA-based software architecture. Service Providers implement, deploy, and publish services to a Service Registry. Service Consumers discover services that they need, and invoke them based on the interfaces specified by the Service Providers. At a high level, these interfaces contain information about the operations along with their required parameters and return values. For instance, the provider of a Web Service uses the WSDL standard to publish its service

operations to a UDDI [18] registry. It has been recognized that besides the functionalities, non-functional requirements or QoS characteristics should also be published [2]. The implications are: a) service providers must have a mechanism to determine and guarantee the published QoS levels of the services; b) service consumers must have the capability to select a service based on both functionalities and QoS; c) software designers should be able to construct composite services from component services whose aggregated capabilities and QoS satisfy the functional needs and the quality constraints of the composite services.

Inspired by the work done in [2,3,6], we propose the QoS concept for Intrusion Tolerant systems - IT-QoS. At design time, the IT-QoS characteristics can be modeled using the UML profile standard specified in [12] for fault-tolerant systems, since it is very natural to transpose fault tolerance QoS to the domain of intrusion tolerance. For instance, the <<QoS Characteristic>> *reliability* becomes S-Reliability that we will study in this paper, and the <<QoS Dimension>> *time-between-failures* can be adapted to specify MTTSF.

Furthermore, the IT-QoS characteristics of an intrusion tolerant Web Service can be specified via the extension Q-WSDL proposed in [2] with the condition that the semantics of the Q-WSDL Schema elements be extrapolated to intrusion tolerance. As such, TimeBetweenFailure, ServiceReliability, etc. will be adapted respectively to TimeBetweenSecurityFailure, Service S-Reliability, etc. In a later section, we will focus on the quality S-Reliability of a service.

### 4. SCIT Architecture

The SCIT architecture is based on the assumption that a system cannot be perfectly devoid of vulnerabilities. Since malicious hackers keep changing their tactics, the architecture does not rely on prevention and detection mechanisms. Instead, the goal is to minimize the opportunity window that can be exploited by potential intruders.

*Architecture Components.* At the core of the architecture is the Central Controller which manages the lifecycle of the server nodes. A server node is an entity embodied by a virtual machine. Applications and

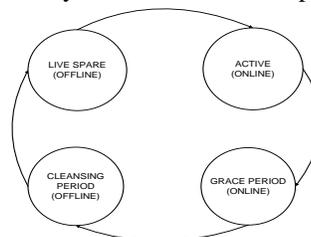


Figure 1. States of a SCIT server node.

**Table 1. List of Parameters and Metrics**

Notation/Terminology	Description
$W_o$ - Online Window	Time duration in which a node is accepting client requests.
$W_g$ - Grace Period Window	Time duration in which a node is finishing requests in its queue.
$T_{cleansing}$	Time duration in which a node goes through cleansing.
$N$	Number of nodes at all states.
$N_{max}$	Maximum number of nodes that can be supported with the given hardware configuration.
$\lambda$	Poisson parameter of malicious attacks among incoming requests.
$\mu$	Poisson parameter of residence time of a malicious attack.

services are deployed in server nodes. Each server node is rotated by the Controller to go through the states depicted in Figure 1. During the “Grace Period”, a node stops receiving new requests and attempts to complete outstanding ones still in its queue. A “Live Spare” node means that it has been cleansed and ready to become active. For security protection, the Controller resides on a separate host, and the link from the Controller to an “Online” server node is only one-way. *Implementation.* The current implementation of SCIT is based on virtualization technology such as VMWare, which allows the instantiation of multiple servers with diverse guest operating systems on a single host machine, and speeds up considerably the rotation time. Additionally, Terracota is used for sharing session data between servers [4] so that the request handoff from one active server to another can be performed seamlessly. Table 1 lists the SCIT parameters and metrics discussed in this paper.

### 5. Service S-Reliability

Reliability is one of the major QoS characteristics, along with performance. S-Reliability extends the notion of service reliability in the SOA context, where it is defined as the probability that a service operation performs correctly without failure [14]. The failures considered in the current literature are generally related to system faults. If we extend the fault concept to the intrusion tolerance context as in [7], then it would be natural to introduce the S-Reliability QoS. By Service S-Reliability, we mean the probability that a service performs reliably and correctly in an environment in spite of malicious intrusions.

In this section, we will present a scheme that allows the SCIT Controller to ensure a required S-Reliability of a Service. In order to support IT-QoS levels, the controller will need additional modules to control these levels and interface with the service registry within the SOA environment.

#### 5.1. Mean Time to Security Failure

In our previous work [5], we have derived a lower bound for the Mean Time to Security Failure (MTTSF) for a SCIT system based on Semi-Markov Chain. With the assumption that attack arrivals are independent and follow a Poisson distribution, the computation showed

that we can make MTTSF depend solely on the exposure window  $W_o$ , the time duration that a SCIT node containing applications or services is online to serve requests:

$$MTTSF \geq F(W_o), \text{ where}$$

$$F(W_o) = \frac{h_0 + h_1}{1 - e^{-2\lambda W_o}} + h_2 \quad (1).$$

In the function  $F(W_o)$ ,  $h=(h_0, h_1, h_2)$  is the vector composed of mean sojourn times in each of the states G(ood), V(ulnerable), and A(ttacked) respectively. The above relation reveals the following implications:

- As the online window decreases, the intrusion tolerance expressed by MTTSF increases.
- As the attack rate  $\lambda$  becomes smaller, the online window can be relaxed to be larger while still achieving the same level of intrusion tolerance. Thus for a more secure environment, we can increase the exposure window and reduce the frequency of cleansing imposed on SCIT nodes.

#### 5.2. Resource Usage

The possible value range of the exposure window is inter-dependent with the cleansing time, and the number of nodes. In other words, decreasing the exposure window does require additional computing resources. In this section, we want to find the minimum number of SCIT nodes that are necessary to achieve a required value for the online window  $W_o$ . Note that we have argued in [5] that:

$$W_g \leq W_o \quad (2).$$

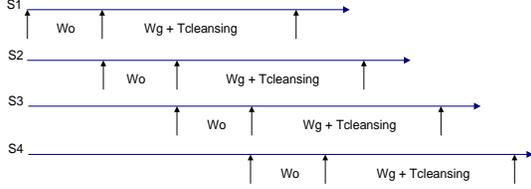
We conjecture that this minimum number of nodes  $N$  is given by the formula:

$$N = \left\lceil \frac{W_g + T_{cleansing}}{W_o} \right\rceil + 1 \quad (3).$$

If we let  $W_g = W_o$  in (3), then:

$$N = \left\lceil \frac{T_{cleansing}}{W_o} \right\rceil + 2 \quad (3a).$$

Expression (3a) implies that we need at least 3 nodes in the architecture. As an example, we consider a SCIT system where  $W_o = 2$  min. and  $(W_g + T_{cleansing}) = 5$  min. The following diagram shows that in this case, we need a total 4 servers for SCIT operations.



**Figure 2. SCIT operation timeline.**

From Figure 2, server S1 is not ready until server S4 goes to grace period, which implies that we need 4 servers at a minimum to ensure continuous operation of the SCIT system in the example.

We have seen previously that increasing MTTSF will require decreasing  $W_o$ . According to expression (3), a decrease of  $W_o$  implies more nodes to use in a SCIT system, hence, more computing resources will be utilized.

The maximum number  $N_{max}$  of SCIT nodes is determined by the capacity of a given set of hardware with static system and memory configuration. In other words,  $N$  in expression (3) and (3a) above cannot exceed the limit  $N_{max}$ .

### 5.3. Atomic Service

We begin by examining how SCIT can provide a predictable S-Reliability in the case of an Atomic Service. A service is *atomic* if it is perfectly autonomous and does not internally invoke any other services. According to [16], there is a relationship between the Reliability and the failure rate. If the failure rate  $\theta$  is constant, then the Reliability of a system or component at time  $t$  is:

$$R(t) = e^{-t\theta} \quad (4).$$

But, the failure rate is the inverse of Mean Time to Failure. Therefore, if we apply (4) to the case of S-Reliability of a Service deployed with SCIT, then we can write:

$$R(t) = e^{-\frac{t}{MTTSF}} \quad (5).$$

Transitively, S-Reliability depends on the exposure window  $W_o$  in a SCIT architecture. Indeed, we know from (1) that MTTSF can be controlled by  $W_o$ . So, combining (1) and (5) allows us to describe the impact on S-Reliability  $R(t)$ : decreasing  $W_o$  will increase MTTSF, hence increase  $R(t)$ . Thus, service providers will have a quantitative mechanism to control and guarantee the S-Reliability of SCIT-ized services.

With  $W_o$  being the determining factor for the level of S-Reliability, expression (3a) specifies the number of nodes necessary to achieve that level. From the service provider point of view, the question is what are the S-Reliability levels that can be offered with a given physical hardware configuration. For a number of available nodes  $N_i$ , a service provider can select to offer  $L_i$  levels of S-Reliability. But, the total number of nodes at all levels must satisfy the following constraint:

$$\sum_{i=1}^{i=K} (L_i \cdot N_i) \leq N_{max} \quad (6),$$

where  $K$  is the count of different values for  $N_i$ . From expression (3a), we deduce the bounds for  $W_o$  in terms of  $N$  and  $T_{cleansing}$ :

$$\begin{cases} T_{cleansing} \leq W_o & \text{if } N = 3 \\ \frac{T_{cleansing}}{N-1} < W_o \leq \frac{T_{cleansing}}{N-2} & \text{if } N > 3 \end{cases} \quad (7).$$

The following pseudo code in Table 2 sketches the steps for a Service-level SCIT Controller to provide different levels of S-Reliability, given a hardware configuration allowing  $N_{max}$  number of nodes. This scheme would allow service providers to develop a set of differentiated S-Reliability levels to offer.

**Table 2. Scheme 1: S-Reliability Control.**

Input data:  $N_{max}$ ,  $T_{cleansing}$ ,  $\lambda$ ,  $\mu$  where

- $N_{max}$  is maximum total number of nodes allowed.
- $T_{cleansing}$  depends on the service such as memory footprint.
- $\lambda$ ,  $\mu$  are estimated values, and may be based on historical data analysis.

1. Select  $K$  different values for number of nodes:  $(N_1, N_2, \dots, N_k)$ .
2. Select  $K$  values for number of levels  $L_i$  to be offered for each corresponding value of  $N_i$ :  $(L_1, L_2, \dots, L_k)$ .
3. **for** each  $i$  from 1 to  $K$
4.   **if**  $(N_i > 3)$  {
5.     Use expression (7) to compute acceptable value range  $G_i$  for  $W_o$ ;
6.      $G_i \leftarrow T_{cleansing} / [(N_i-2)(N_i-1)]$ ;
7.     Divide range  $G_i$  by  $L_i$ :  $G_i/L_i$ .
8.   }
9.   **for** each  $j$  from 1 to  $L_i$
10.     **If**  $(N_i = 3)$  then  $W_o \leftarrow j \cdot T_{cleansing}$ ;
11.     **Else**  $W_o \leftarrow T_{cleansing} / (N_i-1) + j \cdot G_i/L_i$ ;
12.     Compute MTTSF using (1) and  $W_o$ .
13.     Derive  $R_{ij}(t)$  using (5) and MTTSF.
14.   **endfor** each  $j$
15. **endfor** each  $i$ .

The first two steps in the above pseudocode (Table 2) require the service provider to select the values for the  $(N_i)_{i=1..k}$  and  $(L_i)_{i=1..k}$ . The optimal selection of these values subject to the constraint expressed in (6) is a Knapsack Problem which is known to be NP-complete [17]. Steps 4 to 11 propose one way to illustrate how to select values of  $W_o$  corresponding to S-Reliability levels. For  $N > 3$ , the values of  $W_o$  form an arithmetic progression with constant increment:

$$\frac{T_{cleansing}}{(N_i - 2)(N_i - 1)L_i} \quad (8).$$

Expression (8), which is calculated in steps 5-7, shows that the choices for  $W_o$  are influenced by the time to cleanse a service. In step 11, we select the value for

$W_o$  in the case  $N > 3$ . For  $N=3$ , the values of  $W_o$  are multiples of  $T_{cleansing}$  as shown in step 10.

Once  $W_o$  is chosen, the MTTSF and S-Reliability can be computed as in steps 12 and 13. Using the S-Reliability Control Scheme, a service provider can come up with a set of possible configurations corresponding to different levels of reliability, which comprise a set of IT-QoS levels to be published in the service registry. There are two modes of operations: static and dynamic.

*Static Mode.* Based on the assessment of the environment and service usage in terms of security and the assumption that the operating conditions are unchanged, a service can be deployed with a static set of configurations with predefined IT-QoS levels.

As an example, assume that the values for  $\lambda$  and  $\mu$  are respectively  $10^{-5}$  and  $10^{-6}$ . Assume that  $T_{cleansing} = 1500$  time units and  $(h_1, h_2, h_3) = (1/3, 1/3, 1/3)$ . We select 2 levels with 3 nodes and 1 level with 4 nodes:  $N_1 = 3$ ,  $N_2 = 4$ ,  $L_1 = 2$  and  $L_2 = 1$ . The following table shows the numerical results when we execute the scheme listed in Table 2. These will be the various S-Reliability levels of a service to be offered to clients.

**Table 3. Example of S-Reliability Levels.**

Level	Number of nodes	$W_o$	MTTSF	$R(t=1000)$
1	3	3000	1969	0.60
2	3	1500	7641	0.87
3	4	750	30040	0.96

*Dynamic Mode.* In this mode, each IT-QoS level can be adjusted dynamically depending on the execution environment. This would require a monitor component to collect data and re-evaluate whether the estimates of the malicious attack parameters  $\lambda$  and  $\mu$  are still valid. If it is determined that those parameters have changed, then the service provider has to make adjustments in order to keep the same levels of reliability previously advertised. Table 4 shows the computational steps for dynamic adaptation.

**Table 4. Scheme 2: Dynamic Adaptation.**

<p>Input data: <math>N_{max}</math>, <math>T_{cleansing}</math>, <math>\lambda</math>, <math>\mu</math>, <math>(N_i)</math>, <math>(L_i)</math>, <math>(R_{ij})</math>, <math>(W_{ij})</math> where</p> <ul style="list-style-type: none"> <li><math>N_{max}</math> is maximum total number of nodes allowed.</li> <li><math>T_{cleansing}</math> depends on the service such as memory footprint.</li> <li><math>\lambda</math>, <math>\mu</math> are estimated values, and may be based on historical data analysis.</li> <li><math>(N_i)</math> denotes K values for number of nodes.</li> <li><math>(L_i)</math> denotes K number of levels for each <math>N_i</math>.</li> <li><math>(R_{ij})</math> is the set of all desired reliabilities with <math>i=1..K</math>, <math>j=1..L_i</math>.</li> <li><math>(W_{ij})</math> is the set of exposure window values associated with <math>(R_{ij})</math>.</li> </ul> <p>1. <b>for</b> each i from 1 to K</p>
--

<p>2. <b>for</b> each j from 1 to <math>L_i</math></p> <p>3. <math>flag \leftarrow false</math>.</p> <p>4. Using (5), compute MTTSF from <math>R_{ij}</math>.</p> <p>5. Using (1), compute new <math>W_{ij}</math> from MTTSF.</p> <p>6. Verify that (7) is satisfied:</p> <p>7. <b>if</b> (true)</p> <p>8. Update with new value for <math>W_{ij}</math>.</p> <p>9. <math>flag \leftarrow true</math>.</p> <p>10. <b>else</b> {</p> <p>11. Adjust <math>N_i</math> to make (7) true.</p> <p>12. Check whether (6) still holds.</p> <p>13. <b>if</b> (true)</p> <p>14. Update system with new values for <math>W_{ij}</math> and <math>N_i</math>.</p> <p>15. <math>flag \leftarrow true</math>.</p> <p>16. <b>endif</b>.</p> <p>17. } // else</p> <p>18. <b>endif</b>.</p> <p>19. <b>if</b> (<math>flag = false</math>)</p> <p>20. Recompute <math>R_{ij}</math> using MTTSF obtained in step 4 in (1) and (5).</p> <p>21. <b>endif</b></p> <p>22. <b>endfor</b> each j</p> <p>23. <b>endfor</b> each i.</p>
---

The main idea of the adaptation scheme in Table 4 is to adjust the values of the exposure windows so that the services keep the same levels of reliability as advertised. This can be done without increasing the number of nodes for rotation as in step 8 of Table 4. In other cases, we need to adjust the number of nodes (step 11). However, this adjustment may violate constraint (6), in which case we have to re-compute and re-publish the new reliability level (step 20).

With the IT-QoS levels published in the service registry, a Service Consumer can select an appropriate service that satisfies both its functional and IT-QoS requirements.

Similarly, based on the guaranteed IT-QoS levels, a Service or Application Designer will be able to compute the overall IT-QoS while selecting which services to use in composing a new application or composite service.

#### 5.4. Composite Service

Schemes 1 and 2 described above apply to atomic services to control their S-Reliability levels. One major benefit of SOA is to allow the architect to design atomic services which usually reside in lower layers of the system architecture, and reuse those services to build Composite Services. If services are implemented by Web Services, then composing services can be achieved using BPEL (Business Process Execution Language), and their execution requires a BPEL engine. So, how do we provide various S-Reliability levels for composite services? According to [14], a composite service based on programming constructs such as BPEL will contain one or more of the following structures: serial composition, parallel

composition, loop, and conditional branch. In terms of reliability, *loop* iterations can be viewed as serial execution of the same service over and over. Branches in a conditional structure are parallel branches whose executions have some associated probabilities. Thus, we only need to consider two cases of serial and parallel composition. Furthermore, we can apply expression (9) from [14,15] to the case of S-Reliability of a service composed by atomic services  $S_i$ ,  $i=1..n$  with S-Reliabilities  $R_i$ ,  $i=1..n$ :

$$R = \begin{cases} \prod_{i=1}^{i=n} R_i & , \text{if serial composite} \\ 1 - \prod_{i=1}^{i=n} (1 - R_i) & , \text{if parallel composite} \end{cases} \quad (9).$$

Consequently, the S-Reliability of a new service can be derived from the composing services. Therefore, controlling a composite service's S-Reliability is reduced to controlling the S-Reliabilities of all the composing services, which can be achieved by applying Schemes 1 and 2 above.

## 6. Conclusion

In this paper, we have proposed the concept of IT-QoS for services in an SOA system. A recovery-driven approach has been presented as an enabling architecture for service providers to offer differentiated levels of S-Reliability for services. Note that our scheme to guarantee S-Reliability levels makes no assumption about the behavior and possible vulnerabilities of a service. Starting from the derived lower bound of MTTSF in a SCIT-ized system, we have presented two schemes to assure IT-QoS for services: a static model and a dynamic adaptation. The latter would allow the re-tuning of the system according to changes of the operating environments in terms of security in order to maintain the same levels of previously published IT-QoS.

For completeness, another aspect worthy of further investigation and alluded earlier is to select the number of S-Reliability levels in such a way to optimize the utilization of the given hardware platform. Devising heuristics algorithms for that optimization is an interesting problem in its own right.

## 7. Acknowledgement

The authors would like to thank the anonymous reviewers for their valuable comments that help revising the final version of this paper.

## 8. References

- [1] Partha Pal et al. "What Next in Intrusion Tolerance". WRAITS 2009, Lisbon, Portugal. Available: <http://wraits09.di.fc.ul.pt/wraits09paper1.pdf>.
- [2] Andrea D'Ambrogio. "A Model-driven WSDL Extension for Describing the QoS of Web Services". *IEEE International Conference on Web Services (ICWS'06)*, 2006.

- [3] Cynthia Irvine and Timothy Levin. "Quality of Security Service". *Proceedings of the 2000 workshop on New security paradigms*, pp. 91-99, Ireland, 2001.
- [4] Anantha K. Bangalore and Arun K Sood. "Securing Web Servers Using Self Cleansing Intrusion Tolerance (SCIT)". *2009 Second International Conference on Dependability*.
- [5] Quyen Nguyen and Arun Sood. "Quantitative Approach to Tuning of a Time-Based Intrusion-Tolerant System Architecture". WRAITS 2009, Lisbon, Portugal.
- [6] Peng Liu. *Architectures for Intrusion Tolerant Database Systems*. Proceedings of the Foundations of Intrusion Tolerant Systems (OASIS '03), 2003.
- [7] Robert Stroud, Ian Welch1, John Warne, and Peter Ryan. "A Qualitative Analysis of the Intrusion-Tolerance Capabilities of the MAFTIA Architecture". *Proceedings of the 2004 International Conference on Dependable Systems and Networks (DSN'04)*, 2004.
- [8] Bharat B. Madan, Katerina Goseva-Popstojanova, Kalyanaraman Vaidyanathan, and Kishor S. Trivedi. "A Method for Modeling and Quantifying the Security Attributes of Intrusion Tolerant Systems". *Dependable systems and networks-performance and dependability symposium (DSN-PDS) 2002*.
- [9] *Organically assured and survivable information systems*. <http://www.tolerantsystems.org>
- [10] Khin Mi Mi Aung, Kiejin Park and Jong Sou Park. "A Rejuvenation Methodology of Cluster Recovery". *CCGrid 2005, IEEE International Symposium* Vol. 1, pp. 90 - 95, May 2005.
- [11] Paulo Sousa et al. "Resilient Intrusion Tolerance through Proactive and Reactive Recovery". *13<sup>th</sup> IEEE International Symposium on Pacific Rim Dependable Computing*, 2007.
- [12] Object Management Group. "UML Profile for Modeling Quality of Service & Fault Tolerance Characteristics & Mechanisms, v1.1", April 2008.
- [13] Valeria Cardellini et al. "QoS-driven Runtime Adaptation of Service Oriented Architectures". *ESEC-FSE'09*, Amsterdam, The Netherlands, August 2009.
- [14] Paolo Bocciarelli and Andrea D'Ambrogio. "Model-driven Performability of Composite Services". *SIPEW 2008*, LNCS 5119, pp. 228-246, 2008.
- [15] Daniel A. Menasce et al. "Capacity Planning for Web Services: metrics, models, and methods". Prentice Hall Professional Technical Reference, 2001.
- [16] Kishor Shridharbhai Trivedi. "Probability and Statistics with Reliability, Queuing, and Computer Science Applications, 2nd Edition". Wiley-Interscience, 2001.
- [17] R. M. Karp. "Reducibility Among Combinatorial Problems". In *Complexity of Computer Computations* (eds. R. E. Miller, and J. W. Thatcher), pp. 85-104, New York: Plenum Press, 1972.
- [18] OASIS. UDDI Version 3.0.2. Available: <http://www.oasis-open.org/committees/uddi-spec/doc/spec/v3/uddi-v3.0.2-20041019.htm>.