Design of Service–Based Systems with Adaptive Tradeoff Between Security and Service Delay

Stephen S. Yau, Min Yan, and Dazhi Huang

Department of Computer Science and Engineering
School of Computing and Informatics
Arizona State University
Tempe, AZ 85287-8809, USA
{yau,min.yan,dazhi.huang}@asu.edu

Abstract. Service-based Systems (SBS) have the advantage of composing distributed systems from various services provided by multiple providers transparently. In addition to functional correctness, multiple non-functional QoS requirements should also be satisfied in such systems. Among these QoS requirements, security protection and real-time performance are the two major concerns. However, neither application users, nor service providers, have adequate control over such QoS of SBS. In this paper, an approach to the design of SBS with the capability of tradeoff between security and service delay of composite services running across various service hosts is presented in order to satisfy both security and real-time performance requirements simultaneously.

1 Introduction

Service-Oriented Architecture (SOA) enables rapid integration of loosely-coupled and reusable services for on-demand applications [1]. A service is a self-contained software entity with network-addressable interfaces that offer well-defined capabilities using standard protocols like SOAP. In SOA, services which do not invoke other services are considered as *atomic services*. *Composite services* are services composed of atomic services provided by various service providers following a specific workflow. Systems based on SOA, called *service-based systems (SBS)*, comprise of various services offered by distributed service providers over networks [2], [3], [4]. For most mission-critical applications in SBS, the services should be provided with not only functional correctness, but also with nonfunctional properties, such as timeliness, security, availability, throughput, and fault-tolerance, which are referred to as QoS attributes [5]. Development of SBS to satisfy the requirements of multiple QoS imposes great challenges because the satisfaction of one QoS requirement often requires certain sacrifices for the others. Therefore, proper tradeoffs among the requirements of QoS attributes are often required. However, current software engineering techniques cannot effectively support such tradeoffs.

Among multiple QoS attributes, we will focus on the design of SBS with tradeoff between security and real-time performance requirements, because they are two major concerns of many mission-critical applications in various areas, such as healthcare,

B. Xiao et al. (Eds.): ATC 2007, LNCS 4610, pp. 103–113, 2007.

e-commerce, military and homeland security. In this paper, we will present an approach to the design of SBS with the capability of tradeoff between security and service delay under various situations. Our approach includes providing an ontology for developers to specify functionalities and QoS properties of atomic services with service interfaces, as well as a methodology for the user to specify security and real-time performance requirements. In order to satisfy security and real-time performance requirements simultaneously, our approach employs a control-based mechanism to perform the tradeoff between security and service delay of SBS when situations change.

2 Current State of Art

For the real-time aspect, a heuristic algorithm for searching the optimal service composition for SBS was presented in [6]. A generic algorithm to find optimal or near optimal migration decisions was introduced in [7] to facilitate the execution of real-time web services. In [5], [8], performance control architectures and algorithms were presented for software systems and web servers based on feedback control theory in order to optimize throughput.

For the security aspect, an approach to provide development and runtime support for situation-aware security (SAS) in autonomic computing systems was presented in [9]**.** A UML profile describing QoS in SOA, such as security and fault-tolerance, was presented [10]. The security vector of a computing task was presented in [11] to characterize the security requirements in n-dimensional space, where each dimension specifies a Boolean security requirement, which is either satisfied by the computing task or not satisfied.

Some research has been done to consider both security and real-time performance. For web-based information systems, a method to improve the performance gain by adaptively adjusting the security level of the system was presented [12]. The concept of security vector is extended to a vector of discrete security levels [13], instead of Boolean values, to represent a security requirement of particular protection strength. By dynamically controlling the security levels of computing tasks, several dynamic security-aware scheduling algorithms were presented to improve the systems' overall performance in terms of security and throughput for distributed computing systems. Some heuristic approaches have been presented to solve the problem of QoS-aware service composition to satisfy users' QoS requirements [14]. However, these approaches have the difficulty to be directly applicable to SBS, in which the situations of users, service providers and system environments are dynamically changing.

3 Our Overall Approach

In order to design SBS which can dynamically satisfy both the security and service delay requirements at runtime, the following challenges need to be addressed:

C1) How to generate formal specification of the security and service delay?

C2) How to model user's requirements of the security and service delay of a service?

C3) How to evaluate the security protection status for atomic and composite services at runtime?

C4) How to dynamically perform the tradeoff between security and service delay for a composite service involving multiple service providers such that both requirements are met?

Our approach, which addresses these challenges, consists of the following six steps:

Given an SBS, with the information of which services are provided by which service providers, and the application requirements of the user,

S1) Service providers use the ontology in Sec. 4 to specify the service interfaces.

S2) The user specifies the expected security protection and service delays of the composite services under various situations in the form of *SPARQL query* [15] to be described in Sec. 4.

S3) Based on the user's expected QoS of composite service in ***S2)***, a QoS-aware service composition process [14] selects the atomic services with appropriate service interfaces specified in ***S1)*** to generate composite service.

S4) The developer uses *User Expectation Function (UEF)* (see Sec.5) for evaluating services' security protection status at runtime when system situation changes.

S5) The developer uses the situation awareness (SAW) agent approach in [3] to synthesizing QoS monitoring modules in order to collect context data, including system states, events generated by services and other extraneous events in the system environment. Context data is reported to the tradeoff controller as *situation* [16].

S6) Our tradeoff controller developed in Sec. 6 dynamically evaluates the security protection status using *UEF* in ***S4)***, and analyzes the tradeoff between security and service delays with feedback of system situations from QoS monitoring in ***S5)***. If both security and service delay requirements are satisfied in current situation, the controller will either adjust the service's security vector [13] to optimize security protection or achieve better real-time performance. Otherwise, the user's security and service delay requirements need to be relaxed, or the service composition needs to be reconfigured to improve its real-time performance, or both.

In our approach, the service interface in our ontology (***S1***) is associated with the security vector and service delay of the atomic service under a specific situation, which tackles Challenge ***C1)***. The SPARQL query (***S2***) tackles Challenge ***C2)*** by specifying user's QoS requirements for services' security and service delays. *UEF* (***S4***) is derived from SPARQL query to evaluate services' security protection status during runtime, which tackles Challenge ***C3)***. The tradeoff controller (***S6***) tackles Challenge ***C4)*** by adjusting security vector to meet user's security and service delay requirements under all situations.

4 Specifying Service Interface and User's Expectation

In this section, we will present the ontology for specifying service interfaces in ***S1)*** and the SPARQL query [15] for specifying user's security and real-time requirements in ***S2)***.

4.1 Specification of Service Interfaces for Security and Service Delay

To compose a service with QoS constraints [14], an atomic service should have clear service QoS capability specification and related context and situation information for the service to be discovered and used in service composition. To achieve this, we use the extended *OWL-S with situation ontology (SAW-OWL-S)* [18] to model QoS properties of a service under various situations, as shown in Figure 1. Security levels in SBS are normalized as discrete levels in the range of [0, 1] to represent various security protection capabilities using the security level models [13], [17]. Tasks with high security level usually have higher protection strength, lower throughput, consume more resources and cause longer delay than tasks with low security level. Each atomic service in an SBS follows the SBS's definition of security level. The security vector is set to certain levels according to current situation, as described in Sec. 6. A service's execution is considered as a process in *SAW-OWL-S*. A process with different security vectors generates different service delays. An atomic service's functionality, together with its specific security vector and designated service delay, is called a *service interface*. Service's delay for a specific security vector can be obtained using distributed real-time benchmark suite techniques [19]. Based on service interface specifications, appropriate atomic services can be chosen for service composition with QoS constraints [14]. Note that atomic services that have dependent relations in a workflow require their service interfaces to have appropriate security correlation. For example, if two services need to communicate with each other at some point in a workflow, their security level of encryption/decryption algorithm should be the same. Otherwise, they will not be able to understand each other's message.

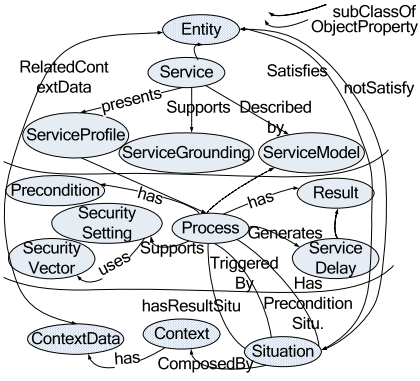

Fig. 1. SAW-OWL-S for security and service delay specification (only related and important classes and relations are shown)

4.2 Specification of User's Expectations for Composite Service

Using SPARQL Query Language for OWL-S [15], a user can specify the requirements for the security protection and service delay of a composite service under situation *situ* as follows:

PREFIX situ: < CS.owl#aS-3-3>
PREFIX pref: <P (P_1, P_2, …P_n)>
PREFIX ns: <example domain>
SELECT ?service_name
WHERE { ?x ns:S ?S. ?y ns:delay ?delay.
?z ns:tradeoff ?tradeoff. ?k ns:security_rating ?security_rating.
FILTER (?S_1 >=B_1 && ?S_2 >= B_2 &&… && ?S_n >= B_n

&&?delay < d && ?tradeoff = = flag &&?security_rating > r)
?x ns: service_name ? service_name. }

In the above requirement specification in SPARQL query, ***situ*** is a *situation* [3], [4], which is a set of contexts in SBS over a period of time that affects future system behavior. ***P*** is the vector of security preference, given by the user for relative priority of different security mechanisms in vector S_i. $P = (P_1, P_2, \ldots, P_n)$, $0 \leq P_i \leq 1$, $\Sigma P_i = 1$. For example, if $P_1 = 0.4$, $P_2 = 0.2$, this means that the user thinks confidentiality aspect (S_{i1}) of service is more important than integrity aspect (S_{i2}). Symbol "?" is used in SPARQL language for querying about the variables (such as *service_name*, security vector *S,* etc.) of the services that meet the requirements in *FILTER* statement. ***B*** is the vector of baseline security levels, given by the user in the form of $B = (B_1, B_2, \ldots, B_n)$. ***r*** is the lowest acceptable security rating of any security mechanism. In Secs. 5 and 6, we will discuss the details of security rating and tradeoff algorithm. ***B***, ***P*** and ***r*** are generated by security requirement engineering [10]. ***d*** is the longest tolerable service delay set by the user. *d* can be obtained from real-time requirement engineering process [20]. During QoS-aware service composition process [14], constraint solver in SBS can map *d* of composite service A to every atomic services using existing off-line constraint deduction process [4] with the input of timing constraints, workflow specification, system status and system control service information. ***flag*** is the tradeoff Boolean indicator. If the user wants a composite service to have shortest service delay possible and only baseline security protection, user will use 0 for *flag*. If the user wants optimal protection, and only requires service delay not bigger than *d*, the *flag* is set to 1.

5 Runtime Security Protection Evaluation Criteria

In ***S4)*** of our approach, the evaluation of service's security protection status at runtime is needed. A $service_i$*'s* real-time performance can be measured by service delay d_i. A $service_i$*'s* security protection status can be obtained in a similar way as the security measurement presented in [13], [21]:

$$G_{Si}(s_i, situ, t) = s_i \times P = s_{i1}*P_1 + s_{i2}*P_2 + \ldots + s_{in}*P_n\,, \quad (1)$$

The measurement of security protection strength $G_{Si}(s_i, situ, t)$ is based on the level of each security mechanism in use sand vector s_i of security preference *P*. However, only this measurement cannot reflect a service's security protection status in the dynamic environment. Intuitively, a service functioning in a hostile environment has more risk for security breaches than a service with the same security vector running in a safe environment. Safety of the environment can be reflected by the security-related system events monitored, such as failure login attempts, the number of encrypted packets using the same key, the number of illegitimate DHCP server packets filtered out by firewall and TLS/SSL handshaking failure rate. Hence, we introduce the concept of *security rating* SR_i of $service_i$:

$$SR_i = (sr_{i1}, sr_{i2}, \ldots, sr_{in})\,, \quad (2)$$

where $0 \le sr_{ij} \le 1$, sr_{ij} represents the monitored status of the j^{th} aspect of the service's security mechanisms. Initially, all sr_{ij} in SR_i is set to 1 because no security event has been monitored. As the service keeps running, SR_i can be updated according to organization-specific *security event processing rules* made by security domain experts. Security event processing rules can be categorized into the following two classes:

- ***Rate-based rules***, in which the security rating is a function of the amount of security events occurred. For example, a security domain expert of an organization can make a rule that sr_1 will reduce 2% whenever 10^2 packets are encrypted with the same key. This rule depicts the wear out of trust towards an encryption key being used constantly.
- ***Exception-based rules***, in which security rating is only related to whether a specific security event occurs in the system environment. For example, if the security log is modified by an unknown user, this indicates that an attacker may have an illegitimate privilege for system resource access. Therefore, the security domain expert can set the rule that the security rating of authorization mechanism will drop to 0 under this condition.

In our approach, AS^3 logic is used to specify such rules because AS^3 logic provides modalities for declarative specifications of SAW [3].

Similar to (1), we have the measurement of security rating G_{SRi} (SR_i, *situ*, *t*):

$$G_{SRi}(SR_i, situ, t) = SR_i \times P = sr_{i1}*P_1 + sr_{i2}*P_2 + ... + sr_{in}*P_n \quad (3)$$

Now we define the *User Expectation Function*, $UEF(s_i, situ, t)$, as a runtime evaluation criteria of a $service_i$*'s* security protection at time *t* under situation *situ* as follows:

$$UEF(s_i, SR_i, situ, t) = G_{Si}(s_i, situ, t) + G_{SRi}(SR_i, situ, t) \quad (4)$$

UEF can be interpreted as a service's security protection status which is affected by both the security mechanisms used by the service and the service's execution environment.

In service composition of a composite service, an atomic service is invoked based on its position in the workflow and whether its specified precondition situations are satisfied. Therefore, we define the *readiness for service* under *situ*: $r_i(situ)$ as 1 if the preconditions for the atomic service $service_i$ have been satisfied under situation *situ*, and the $service_i$ is ready for execution or is currently being executed; otherwise, $r_i(situ)$ is defined as 0.

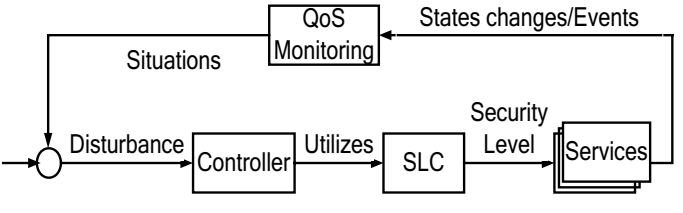

Fig. 2. Our control model for the security and service delay tradeoff in a composite service

The overall security protection *UEF (A, situ, t)* of a composite service *A* can be measured by the sum of contribution from *UEF (serv$_i$, situ, t)* of all the ready atomic services as follows:

$$UEF\ (A,\ situ,\ t) = \Sigma\ r_i(situ)^*\ UEF(s_i,\ SR_i,\ situ,\ t) \quad (5)$$

6 Control-Based Tradeoff

In this section, we will present a tradeoff controller for tradeoff between security and service delay for SBS in ***S6)***. The model of the tradeoff controller is shown in Figure 2. Generally, there are many system-specific control services provided by the underlying SBS for managing system settings. One such service, called *Security Level Controller* (*SLC*) [13], is used by tradeoff controller to adjust security levels of the atomic services of a composite service. The execution process of the composite service produces security events and results in system state change, which are captured by QoS Monitoring module and reported to tradeoff controller as feedback of situations. With the input of situation *situ*, the current security vector s_i and previous security vector s_i', the estimated *service$_i$'s* delay on host j can be generated by *SLC* similar to *earliest start time* in [13]:

$$estd_i\ (serv_i,\ s_i',\ s_i,\ situ) = r_j + e_i\ (serv_i,\ s_i,\ situ) + h(s_i',\ s_i\ ) \quad (6)$$

where r_j represents the remaining execution time of a previous service on host j, $e_i(serv_i,\ s_i,\ situ)$ represents the estimated execution time of service$_i$ obtained from the service interface, and $h(s_i',\ s_i\ )$ represents the time overhead of reconfiguring *service$_i$*'s security from s_i' to s_i. *SLC* is for security level reconfiguration and delay estimation for a single atomic service. The overall security and service delay tradeoff of a composite service is achieved by the tradeoff controller using algorithm shown in Figure 3.

The tradeoff controller analyzes if every atomic service of a composite service can satisfy the security requirement *B* while still satisfy service delay requirement in line 4. If *B* cannot be satisfied by any atomic service, the composite service will be rejected in line 23, and both the user and service providers will be notified. If user's tradeoff focus for a composite service is on minimizing service delay (*flag*=0), we will use *B* for all its atomic services in line 5; If user's tradeoff focus for a composite service is on improving security protection (*flag*=1), then the tradeoff controller uses *SLC* to adjust security levels of atomic services in lines 6-21. *SLC* will start from the security mechanism whose security rating has been decreased most, and stop until the mechanism whose security rating been increased most. For security mechanisms whose security ratings have changed the same amount of value, *SLC* will start from the mechanism with the highest security preference value P_m to the lowest value P_n. If the result of adjusting the security level of an atomic service *service$_i$* by *SLC* can increase the composite service's *UEF* and does not lead to a delay violating requirement d_i (in line 19), the current security level will be accepted for *service$_i$*. Otherwise, *service$_i$* will go back to its previous security level (in line 20).

Let us consider an example service of enhanced netmeeting *eNM,* which is composed of the atomic services VoIP *($Serv_{VoIP}$),* FTP *($Serv_{FTP}$),* and video-on-demand *($Serv_{VOD}$).* We only consider three aspects of security mechanisms: confidentiality (s_1), integrity (s_2) and authorization (s_3). Suppose for "video conference" situation, the user

sets P = (0.2, 0.3, 0.5) and 0.65 as lowest acceptable security rating r, and 1 for *flag* value. For confidentiality S_1, $Serv_{VoIP}$ supports three levels: level 0.22: DES, level 0.44: 3-DES, and level 1: AES. The levels are obtained by dividing the key lengths of DES (56bits) and 3DES (112bits) by the key length of the strongest AES (256bits) encryption. Assume at time *t-1*, the security vector of VoIP is s_{VoIP}= (0.22, 1, 1), and the security rating is sr_{VoIP}= (0.7, 1, 1). The voice compression standard for VoIP suggests the maximum acceptable delay for VoIP in wired communication to be 150 *ms* [22].

1. **for** each host server *j* **do**
2. Sort component atomic services of the composite service A on *j* according to service priority in workflow of A from high to low
3. **for** each $service_i$ in the sorted service group of the host *j* **do**
4. **if** $estd_j(serv_i, s_i', B, situ) \le d_i$, **then**
5. **if** *flag* = = 0, **then** s_i = B, except security level correlated, **continue**;
6. **else if** *flag* = = 1, and ,
7. **if** there exists an $sr_{ik} < r$, **then** $s_i = B$, except security level correlated;
8. Use (5) to calculate *UEF (A, situ, t)* and store the result in *U*;
9. **for** each sr_{ik} of $service_i$ **do**
10. $\Delta\ sr_{ik} = sr_{ik}(t) - sr_{ik}(t\text{-}1)$;
11. **end for**
12. Sort s_{ik} of $service_i$'s security level s_i according to $\Delta sr_{ik} * P_k$, so that $\Delta sr_{i1} * P_1 < \Delta sr_{i2} * P_2 < \ldots < \Delta sr_{in} * P_n$; if $\Delta sr_{im} * P_m = \Delta sr_{in} * P_n$, sort s_{im} and s_{in} according to P, so that $P_m > P_n$;
13. **for** each s_{ik} , $1 \le k \le n$, **do**
14. **if** s_{ik} has not been correlated, **then**
15. **while** $s_{ik} \le \max\{S_{ik}\}$ **do**
16. Increase security level of s_{ik} with *SLC* to next level;
17. Correlate security level with other services with dependency in workflow;
18. Use (5) to calculate *UEF (A, situ, t)* with current s_i and SR_i of $service_i$;
19. **if** $UEF(A, situ, t) \ge U$ **AND** $estd_i(serv_{i,}, s_i', s_i, situ) \le d_i$, **then** $U = UEF(A, situ, t)$;
20. **else** decrease s_{ik}, **break**;
21. **end while**
22. **end for**
23. **else** reject composite service A, notify both users and service providers of A;
24. **end for**

Fig. 3. The tradeoff algorithm for the controller of a composite service

Assume that the security domain expert gives a rate-based security event processing rule that "security rating of confidentiality will reduce 2% whenever 10^2 packets are encrypted", which can be specified in AS^3 logic as follows:

SERV1) *Encypt_pack (int(pack#), rbr, saw_rbrAgent)* → *serv(int(pack#), rbr, saw_rbrAgent)*
AS1) *serv(int(packe#), rbr, saw_rbrAgent)* ∧ *pack# >= 10^2* → *diam(k([Decrease, 0.02], monitor_until(-1, success), saw_packageAgent))*

In the above rate-based rule, *saw_rbrAgent* is a synthesized SAW agent responsible for services related to rate-based rules (rbr). *saw_packageAgent* is a synthesized SAW agent monitoring the situations related to packages on networks.

At time *t*, the QoS Monitoring detects that the number of packets encrypted is 500 packets per second. Hence, sr_1 decreases to 0.6. According to lines 9-11 of our tradeoff algorithm in Figure 3, we can calculate the change of security rating Δsr_{VoIP1} = 0.6-0.7 = -0.1. Assume that the security levels and ratings of S_{FTP} and S_{VOD} unchanged at time *t* and there is no security dependency. For S_{VoIP}, after the sorting process of line 13, we have $\Delta G_{SRVoIP} = \Delta sr_{VoIP1} * P_1$ = -0.1*0.2 = -0.02. Hence, as in lines 13–20, we increase the security level of confidentiality S_{VoIP1}. According to the experimental results of voice over IPSec [23], the *VoIP* service with different encryption algorithms for confidentiality purpose have the delays shown in Table 1.

Table 1. Tradeoff options for service S_{VoIP}

S_{VoIP1}	ΔG_{SLVoIP}	delay	$\Delta_{UEF(eNM, s, t)}$
0.44	(0.44-0.22)*0.2 = 0.044	110 ms	0.044-0.02 = 0.024
1	(1-0.2)*0.2 = 0.16	152 ms	0.16-0.02 = 0.14

Although the security level 1 of S_{VoIP1} can increase *UEF(eNM, s, t)* further, the delay at level 1 becomes 152 *ms*, which is unbearable according to line 20. Based on our tradeoff algorithm in Figure 3, S_{VoIP} will adapt its confidentiality security level to s_{VoIP1} = 0.44 (3DES), which will improve *UEF (eNM, s, t)* and guarantee that eNM finishes less than 150*ms* as well.

7 Conclusion and Future Work

In this paper, we have presented a control-based approach to the design of SBS with the dynamic balance between service's security and performance due to the change of the situations of SBS. We have extended *SAW-OWL-S* to specify service interface. *SPARQL query* is used to depict user's expectations for security, service delay and tradeoff preference. *User Expectation Function* has been derived to measure security protection at runtime. Future work includes simulation of our approach to evaluate its effectiveness, and relaxation of QoS requirements and composite service reconfiguration after service requests are rejected.

Acknowledgment

This research was supported by National Science Foundation under grant number CNS-0524736 and DoD/ONR under MURI Program, contract number N00014-04-1-0723. The authors would like to thank Zhaoji Chen, Junwei Liu, Yin Yin, and Luping Zhu for many helpful discussions.

References

1. Jones, S.: Toward an Acceptable Definition of Service. IEEE Software 22(3), 87–93 (2005)
2. Yau, S.S., et al.: Situation-Awareness for Adaptable Service Coordination in Service-based Systems. In: Proc. 29th Annual Int'l. Computer Software and Application Conf. pp. 107–112 (2005)
3. Yau, S.S., et al.: Automated Agent Synthesis for Situation Awareness in Service-based Systems. In: Proc. 30th Annual Int'l. Computer Software and App. Conf. pp. 503– 510 (2006)
4. Yau, S.S., et al.: A Software Cybernetic Approach to Deploying and Scheduling Workflow Applications in Service-based Systems. In: Proc. 11th Int'l. Workshop on Future Trends of Distributed Computing Systems, pp. 149–156 (2007)
5. Abdelzaher, T.F., et al.: Feedback Performance Control in Software Services. IEEE Control Systems Magazine 23(3), 74–90 (2003)
6. Tsai, W.T., et al.: RTSOA: Real-Time Service-Oriented Architecture. In: Proc. 2nd IEEE Int'l. Workshop on Service-Oriented System Engineering, pp. 49–56 (2006)
7. Hao, W., et al.: An Infrastructure for Web Services Migration for Real-Time Applications. In: Proc. 2nd IEEE Int'l. Workshop on Service-Oriented System Engineering, pp. 41–48 (2006)
8. Lu, C., et al.: Feedback Control Architecture and Design Methodology for Service Delay Guarantees in Web Servers. IEEE Trans. on Parallel and Distributed Systems 17(9), 1014–1027 (2006)
9. Yau, S.S., Yao, Y., Yan, M.: Development and Runtime Support for Situation-Aware Security in Autonomic Computing. In: Proc. 3rd Int'l. Conf. on Autonomic and Trusted Computing, pp. 173–182 (2006)
10. Wada, H., Suzuki, J., Oba, K.: A Service-Oriented Design Framework for Secure Network Applications. In: Proc. 30th Annual Int'l. Computer Software and App. Conf, pp. 359–368 (2006)
11. Spyropoulou, E., Levin, T., Irvine, C.: Calculating Costs for Quality of Security Service. In: Proc. 16th Annual Conf. Computer Security Applications, pp. 334–343 (2000)
12. Son, S.H., Zimmerman, R., Hansson, J.: An Adaptable Security Manager for Real-Time Transactions. In: Proc. 12th Euromicro Conf. on Real-Time Systems, pp. 63–70 (2000)
13. Xie, T., et al.: Real-Time Scheduling with Quality of Security Constraints. Int'l. Jour. High Performance Computing and Networking (2006)
14. Berbner, R., et al.: Heuristics for QoS-aware Web Service Composition. In: Proc. Int'l Conf. on Web Services, pp. 72–82 (2006)
15. SPARQL Query Language for RDF. W3C Working Draft 26 (2007) http://www.w3.org/TR/rdf-sparql-query/
16. Yau, S.S., Liu, J.: Functionality-based Service Matchmaking for Service-Oriented Architecture. In: Proc. of 8th Int'l. Symp. on Autonomous Decentralized Systems, pp. 147–152 (2007)
17. Kang, K., Son, S.: Systematic Security and Timeliness Tradeoffs in Real-Time Embedded Systems. In: Proc. 12th IEEE Int'l. Conf. on Embedded and Real-Time Computing Systems and Applications, pp. 183–189 (2006)
18. Yau, S.S., Liu, J.: Incorporating Situation Awareness in Service Specifications. In: Proc. 9th IEEE Int'l. Symp. on Object and Component-oriented Real-time Distributed Computing, pp. 287–294 (2006)

19. Cavanaugh, C.D.: Toward a Simulation Benchmark for Distributed Mission-Critical Real-time Systems. In: Proc. Networking, Sensing and Control, pp. 1037–1042 (2005)
20. Goldsack, S.J., Finkelstein, A.C.W.: Requirements Engineering for Real-time Systems. Jour. Software Engineering 6(3), 101–115 (1991)
21. Wang, C., Wulf, W.A.: A Framework for Security Measurement. In: Proc. National Information Systems Security Conf. pp. 522–533 (1997)
22. Barbieri, R., Bruschi, D., Rosti, E.: Voice over IPsec: Analysis and Solutions. In: Proc. 18th Annual Computer Security Applications Conference, pp. 261–270 (2002)
23. Nascimento, A., Passito, A., Mota, E.: Can I Add a Secure VoIP Call. In: Proc. 2006 Int'l. Symp. On a World of Wireless, Mobile and Multimedia (2006)