Title
A service accountability framework for QoS service management and engineering

Journal
Information Systems and e-Business Management, 7(4)

ISSN
1617-9854

Authors
Lin, Kwei-Jay
Chang, Soo Ho

Publication Date
2009-09-01

DOI
10.1007/s10257-009-0109-5

Peer reviewed

Inf Syst E-Bus Manage (2009) 7:429–446
DOI 10.1007/s10257-009-0109-5

ORIGINAL ARTICLE

A service accountability framework for QoS service management and engineering

Kwei-Jay Lin · Soo Ho Chang

Received: 15 April 2008 / Revised: 18 September 2008 / Accepted: 5 January 2009 /
Published online: 24 January 2009

Abstract Service science, management and engineering (SSME) research is to study the methodology and technology for service innovation, design, development and delivery. Since service industry is very quality-sensitive and trust-dependent, we propose a service accountability management framework to detect, diagnose, defuse and disclose the root cause for any problematic service process. The accountability support is important for SSME since service processes often rely on external service providers to deliver part of the service functionalities. A service system must have effective yet efficient mechanisms to ensure that every external service is delivering a consistent and acceptable level of performance to meet the end-to-end quality of service (QoS) of the whole service process. In this paper, we present the accountability framework, identify the components in an accountable service architecture, and design an accountability diagnosis methodology. We also briefly present the inteLLigent Accountability Management Architecture (LLAMA) project which implements the accountability service bus (ASB), an agent-based middleware to support the monitoring, diagnosis, and reconfiguration of e-services. LLAMA ASB interacts with accountability agents to monitor services and the Accountability Authority to automatically diagnose faulty situations. The LLAMA technology is useful to ensure the QoS in SSME-based systems.

Keywords Service-oriented architecture · Accountability · Quality of service · Diagnosis · LLAMA service bus

K.-J. Lin · S. H. Chang (✉)
Department of Electrical Engineering and Computer Science,
University of California, Irvine, CA 92697, USA
e-mail: sooho.chang@gmail.com

1 Introduction

The area of service science, management and engineering (SSME) is a multi-disciplinary research and education endeavor to study the methodology and technology for service innovation, design and delivery in order to make service industry more effective and scalable. In recent years, business globalization, server automation, and componentization of service functions have induced a tremendous growth on Web-based services. Advances on the Internet technology and the popularity of Web-dependent life style have created new service business opportunities. A service business may be used to deliver knowledge, utility, experience, information, or other intellectual content to its clients. The capabilities to (1) innovate knowledge-intensive services, (2) create business values from domain expertise, and (3) design and develop new B2C and B2B business models, allow many enterpreneurs to set up innovative and successful service businesses at an unprecedent rate.

To maintain and to grow a successful service business, however, takes more than just innovative ideas. Service industry is qualitatively different from product-oriented business. They are much more dynamic and demanding than traditional product-oriented business. Moreover, they are very quality- and trust-sensitive. Therefore, the question of how a service business can guarantee manageability and visibility throughout their service delivery process is very important and challenging in SSME.

The goal of our SSME research is to develop the IT technology for quality-sensitive services. In our research, we design new service management models. We also develop an IT system framework with a powerful middleware support for service composition and execution. By building a service infrastructure using current and emerging IT technologies (including wireless Internet, software agents, pervasive devices, etc.), we want to provide the scientific and technical support that helps companies become successful service-oriented businesses.

1.1 SOA for SSME

One way to meet the service scalability and manageability needs is for companies to divide their service processes into reusable service components that can be better managed individually. Moreover, companies may want to outsource some service components for cost efficiency and growth capability reasons. Enterprises should concentrate on their core competency and avoid wasting resources on providing mundane or non-essential functions. For example, small hospitals may rely on external specialists and laboratories to perform medical or laboratory tests; engineering consulting companies may rely on law firms to conduct patent search or filing.

To achieve an effective outsourcing, one can adopt the service-oriented architecture (SOA) paradigm. SOA has recently become a popular paradigm to integrate distributed heterogeneous IT components and applications (Bichler and Lin 2006; Huhns and Singh 2005). Using SOA, enterprise systems can execute transactions across multiple server domains at distributed locations, compose

complex business processes, and integrate services to enable collaborations among business partners. However, SOA is not restricted to composing IT software services only; the SOA paradigm can be used to integrate other types of services, including those by human, by portable devices, by physical systems, and by intelligent software agents. In general, the loose coupling and detached ownership of services give SOA the attractive properties of openness, flexibility, and agility.

1.2 Accountable SOA

When a service provider subcontracts some service components to external service providers, it is imperative for the service provider to have a mechanism to ensure that all external servers provide an acceptable and consistent level of performance, in order to meet the overall quality requirements from its customers. The behavior of individual services in a service process must be monitored in order to settle any responsibility issue. Any under-performed external service should be replaced immediately to ensure the required quality of service (QoS) level. Therefore, service systems should have simple-yet-effective mechanisms to conduct:

1. monitoring of services and identification of likely faults or problems,
2. inspection and reasoning about the correctness of individual services, and
3. dynamic reconfiguration of services and service processes.

To provide a holistic solution to the above requirement, we propose the framework of *accountability* as a means to monitor services, identify problems, and make remedies. Accountability requirements have been adopted in real life by many public institutions such as government agencies, hospitals and non-profit organizations as a comprehensive measure to provide operation transparency and to furnish a responsible attitude toward any unacceptable behavior. We believe that accountability for professional services should be carefully studied in SSME by considering the unique characteristics of service outsourcing. Specifically, we believe an accountability measure for service industry should clearly specify the expected behavior, model inter-dependencies among service components, diagnose and identify faulty service entities, and defuse and recover from problems at run time.

The contribution of this paper is that we present a comprehensive study on accountable services. We identify the key components of an accountable service system architecture, and study the flow of accountability support. We also analyze the relationships among different quality attributes and study ways to conduct diagnosis to find the actual cause of a quality deficiency. We then present a middleware project that is designed to support the accountability of e-services. Although the current implementation is for e-services, the design and the architecture can be used to support other types of services (including human, physical and professional services).

This paper is organized as follows. Section 2 provides the background on accountability. The accountability computing model and components are presented in Sect. 3. The inteLLigent Accountability Management Architecture (LLAMA)

middleware for e-service accountability is presented in Sect. 4. Related work on service QoS is presented in Sect. 5. Concluding remarks are given in Sect. 6.

2 Introduction to accountability

Accountability has become a major concern for business management in USA, especially after the ratification of the Sarbanes-Oxley Act of 2002 (Sarbanes-Oxley 2002, also known as the Public Company Accounting Reform and Investor Protection Act of 2002), which establishes new enhanced accountability standards for all public company management and public accounting firms. The Public Company Accounting Oversight Board (PCAOB) is given the responsibility of overseeing, regulating, inspecting, and disciplining accounting firms in their roles as auditors of public companies. The Act has made accountability a mandatory requirement for organizations. Partly to meet this needs in many enterprises, IBM has identified "*Managing Business Integrity*" as one of the top four technical challenges in its Global Technology Outlook (GTO) in 2007 (Mohan 2007). An effective quality management infrastructure is essential to maintain business integrity. All these provide the motivation for our research on accountability in order to regulate service quality.

2.1 General accountability

The notion of accountability is used for clarifying the responsibility for certain problems in complex procedures among different parties. It is a comprehensive quality assessment concept in the course of complex interaction patterns (such as invocation, negotiation, nested loops, fork and join, etc., Nissenbaum 1994). When problems occur in complex transactions among different parties, assessing the source of the problems and figuring out the responsible parties for the problems usually are non-trivial.

Horsch (1996) reports a project on results-based accountability for public institutions. It identifies the following design elements for accountability systems:

1. Objective: outcomes that articulate what programs are to achieve;
2. Quality: indicators to measure whether or not outcomes have been achieved;
3. Benchmark: performance standards to assess how programs are progressing;
4. Monitoring: data collection instruments to regularly obtain indicator data;
5. Feedback: periodic collection and analysis of data for decision making and reporting.

Among the five elements of the accountability design, the first three are application-specific, to be defined by application designers. On the other hand, IT may be used to manage the last two elements: monitoring and feedback. We thus focus our study on the IT mechanisms for service monitoring and accountability analysis.

Accountability in the context of computer decision systems has been studied in Johnson and Mulvey (1995). The authors have discussed four accountability issues:

1. *Fault*: we must first identify what has gone wrong in a system. This is usually identified from some expected activities but performed in substantially different ways. From an observed fault, we may look for the cause that leads to the problematic situation.
2. *Causality*: given an observed fault, there may be one or more associated causes. The causality is a direct or indirect relationship between the fault and its causes. When understanding causality, we need to consider the possibility of having a chain of causes where one action affects another. Given a chain of causes, one can trace the fault propagation path and identify the root cause.
3. *Role*: a role is a specific type of functionality or behavior played by a party. Some roles in interactions may trigger or otherwise prevent problematic situations; the parties who play such roles are to be responsible for the problems identified.
4. *Liability*: given a problematic situation, there is usually some consequence due to the service failure. The service provider should be responsible for the consequence. The liability may be as simple as suspending the service or as serious as paying for the complete damage due to the faulty service.

2.2 Accountability in SOA

Service-oriented architecture has become a popular and prominent paradigm to integrate IT components and applications (Bichler and Lin 2006). Our SSME research decides to study the accountability framework using SOA. Through the SOA framework and IT support, we hope to offload the accountability management burdens from human service operators to IT infrastructure. Accountability imposes that all services deployed (regardless whether by human, machine or software) have the obligation to accept responsibility or to account for their actions (Lin 2007). By imposing accountability on services, service clients are provided with the transparency of understanding abnormal behaviors in service collaborations. As a result, they may receive a higher standard on problem resolution when subscribing services with accountability support.

We now review the four accountability attributes from SOA's perspective.

- *Fault* in SOA is the result of a service execution which deviates from the client's expectation on the service. When a service is invoked, the service is expected to fulfill some functional and nonfunctional capabilities. If a service result does not comply with the a priori agreement with a client, it is deemed to have a fault.
- *Causality* in SOA is a relationship between a fault and one or more services that produce the results directly or indirectly. In many cases, a fault in a service execution can be the cause for another fault, resulting in a chain of faults.
- *Role* in SOA is a well-defined responsibility fulfilled by a party during service execution. Service providers and service clients are two most obvious roles in SOA. Other roles in SOA include service monitors, fault handlers, etc.
- *Liability* in SOA may include service replacement, process reconfiguration, and/or compensation transactions. In the simplest case, a faulty service will be

replaced by another capable service candidate. It may also cause the service process to be recomposed with a different workflow, or be restarted with a valid set of input values. Finally, due to the erroneous executions that have occurred, a service system may need to initiate a compensation transaction to undo previous executions. Depending on the types of the problem situations, different liability actions should be applied.

We now use an example to discuss the four attributes. Suppose a *Flight Reservation* service process includes several services such as searching flight availability, validating client identification, checking credit information, making payment, and issuing tickets. There are different *roles*: airlines, customer reservation service, payment system, etc. A problem in this service process may be related to that, for example, the service for accepting payments does not complete the execution within an acceptable duration. Hence, there is a *violation* on the QoS value of response time on payment service. This problem may propagate to other services in the business process; e.g. it may *cause* a fault on the service that issues flight ticket and thus aborting the reservation. This illustrates the relationship between the (root) cause and the resulting problem. To remedy the problem and to make the system liable, a recovery action should be applied to replace the faulty payment service with another service providing the same functionality. In some cases, even when the transaction was aborted, a ticket should be re-issued at the quoted price as long as the customer has initiated the payment process. The system is *liable* for the delay and should honor the quoted price even when it is no longer available.

From the above example, we can see that the issue of accountability is more than simply detecting a fault. It must identify the actual problem and also provide the action to resolve the liability. All these may need to be done in real time since customers may request other alternative action immediately if no satisfactory result can be delivered in real time. Therefore, an automated and capable accountability support is critical to accountable SOA.

3 Accountable service computing model

When a problem occurs in a service system, the problem should be recognized, the original causes should be identified and resolved, and the players that are responsible for the faults should take appropriate remedy actions. The key phases of accountable computing thus include *Detect*, *Diagnose*, *Defuse*, and *Disclose* as shown in Fig. 1. Each phase has its goals and artifacts. As the executions of the phases are conducted in sequence, the artifacts of the phases are continuously

Fig. 1 Phases of accountability management

elaborated. In this way, a service system may be continuously improved. In this section, we study the issues for each phase.

3.1 The detection phase

The detection phase is to be implemented by (1) dynamically acquiring status on services and the environment, (2) computing the values of relevant quality attributes, and (3) comparing the values of quality attributes to the threshold values in service level agreement (SLA).

An accountable SOA system must first determine the expected behavior in a service system. The criteria for determining acceptable behaviors are usually derived from service policies and SLA that includes quality attributes and constraints of services. A policy in SOA is a formal statement representing assertions on the requirements of services. Examples of SOA policies include authentication requirements for sensitive information processed by a service, and the predetermined length of response time for time-sensitive services.

Service providers and consumers define their policies on services. A service contract is then derived from a mutual agreement on various measurable quality aspects of the service in order to enable runtime measurement and calculation. Quality attributes are defined with an acceptable range for normal service executions (Johnson and Mulvey 1995; O'Brien et al. 2005). Example quality attributes are availability, accessibility, and performance. The values of these quality attributes can be computed from measurements on services and the environment. For example, availability is defined as the percentage of time during which a service is available and performance can be measured by dispatch time and latency time.

Faults in service processes may be present at different levels: including *platform level*, *service level*, and *process level*. The platform level includes the system infrastructure, middleware, resources, and communication network. The service level is defined by service invocations, services deployed, and QoS interfaces. The process level is defined by business process specification and service quality from the end-to-end point of view. Therefore, an accountable system must detect all kinds of abnormality at all three levels. Measurements on services and environment can be gathered by various service monitoring methods such as Baresi and Guinea (2005) and Morgan et al. (2005).

Some quality attributes are applicable to all three levels, while others are only present at a specific level. For example, performance at different levels may be measured by the throughput of a host, the response time of a service and of a business process. However, scalability, the degree of consistency on response time, may be more related to the platform rather than the service or process since processes generally depend on the platform where they are deployed.

3.2 The diagnosis phase

Given some detected fault or faults, an accountable service system must analyze the problems detected to identify the likely causes. We define the *cause* as an origin of

the abnormality or a situation (context) which has resulted in the abnormality. An example of the origin of abnormality can be a failure of a service specified in a business process and its impact on subsequent services.

We classify causes into three classes: (1) causes arisen from problematic services, (2) causes arisen from malfunctioned infrastructure, and (3) causes arisen from invalid service invocations by clients. Causes of the first type are often found on poor behavior of services such as "service not responding" and "unexpected service behavior". Causes of the second type are similar to that found in conventional application system management. For example, service systems may be overloaded with an excessive number of invocations or network may be congested. Causes of the third type are related to invalid input values or parameters submitted by clients, and incompatible input and output parameters between two connected services.

There are different ways to diagnose faults and to identify causes. We have studied a probabilistic model in Zhang et al. (2007), which adopts the Bayesian network model to assess services upon the observation of some abnormal behavior. The project utilizes monitoring agents and intelligent diagnosis methods, so that accountability is efficiently assessed. However, as in any probabilistic model, the result of assessing accountability and diagnosed results is not always correct. Additional checking is required to confirm the true cause of a fault.

Another way to conduct diagnosis is to use a statistical approach which utilizes the history of service invocations and rules for determining the cause based on the observed abnormality. In this approach, we need to derive and define diagnosis rules from the statistical analysis on earlier occurrences of (*Abnormality Type*, *Cause Type*) pairs. The more comprehensive service logs are recorded, the more reliable diagnosis may be produced. Statistical studies have been reported in Wang et al. (2005) and Ardissono et al. (2005).

Fault diagnosis may be carried out with the consideration of specific relationships among quality attributes as shown in Fig. 2. For example, an unsecured host which receives a flood of attack messages may make a Web service on the host unavailable. In turn, the unavailable Web service can cause a slow performance of a business process which invokes the Web service. A correct diagnosis should be conducted by tracing the relationship between different quality attributes, from performance to security via availability in this example.

3.3 The defusing phase

This phase is to resolve the problem determined from the diagnosis phase. The actual method of defusing problems largely depends on the type of the cause.

For causes from problematic services, a defusing method usually is to replace malfunctioning services. The replacement can be for a service, a portion of a process execution path, or the whole proess. A problematic service may be replaced by a compatible service which provides the same functionality and possibly stronger QoS. When replacing a portion of an execution path, the execution path is updated by considering the dependency between replaced services and their neighbor

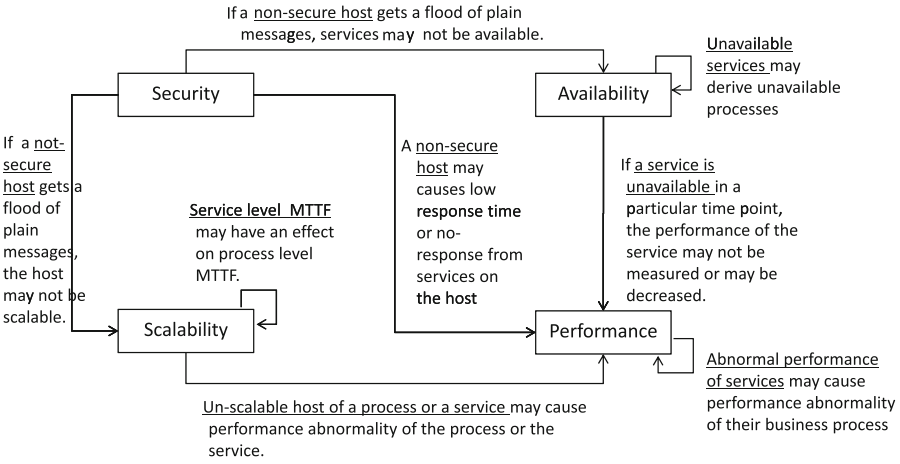

Fig. 2 Relationships among quality attributes

services. When replacing the whole process path, the path is re-defined without using the problematic service.

For causes from malfunctioning infrastructure, a defusing method must identify all services affected and exclude them from providing further services. After that, the system can reconfigure or reboot the middleware environment. For causes of the third type, e.g. invalid input parameters from clients, the defusing method is to request for a new set of input values, or select another service that can accept the invocation correctly.

3.4 The disclosure phase

In real life accountability situations, a liability is imposed on the party that has caused a service failure. In SSME, this phase is to apply a post-mortem remedy on the result of a faulty service. The remedy may be to compensate the service client due to the damage from the service delivery failure, or to penalize the server so that it is prevented from making similar faulty service in the future.

The former remedy may be carried out as compensating transactions for the client. For example, an aborted transaction may be restarted. An incorrect purchase may be returned and credited. A delayed transaction may need to be honored with the original price and terms, even with extra discounts. Such remedy actions may be executed as “exception handling” transactions by a penalty decision maker in the accountability framework.

To penalize faulty servers (such as business process designers, service providers, network administrators, SOA middleware managers), the information about their unacceptable service record can be used to change the qualification or reputation of these services. The result of disclosure can be reflected on the service repository for future references. That is, the causes for certain abnormality, the effectiveness of defusing methods for the causes, and reputation of the services can be effectively considered before making future invocations of the services.

3.5 Accountability architecture components

To provide the functionality needed for accountable service computing, a component architecture for accountable service system is shown in Fig. 3. The architecture consists of four components: *Monitor*, *Inspector*, *Handler*, and *Recorder*.

The component *Monitor* is to detect abnormality of services. It consists of the following sub-components: Service Policy, Service Agreement, Fault Detector, and Abnormality. The sub-component Fault Detector provides ways to recognize abnormality by monitoring services, comparing current states to Service Agreement, and describing abnormal situations.

The component *Inspector* is to analyze the abnormality and identify the origin of the abnormality. It consists of sub-components: Diagnosis Conductor and Root Cause. For the abnormality detected, the sub-component Diagnosis Conductor identifies original causes using its diagnosing methods such as probability model or rule-based model, and produces a description of causes in a predefined template.

The component *Handler* is to recover a problematic service system from the causes identified. It consists of sub-components: Recovery Decision Maker, Recovery Plan, and Recovery Manager. Recovery Decision Maker generates a recovery plan on which Recovery Manager defuses the root causes. The recovery plan is defined with appropriate defusing methods for the types of causes and system management guidelines.

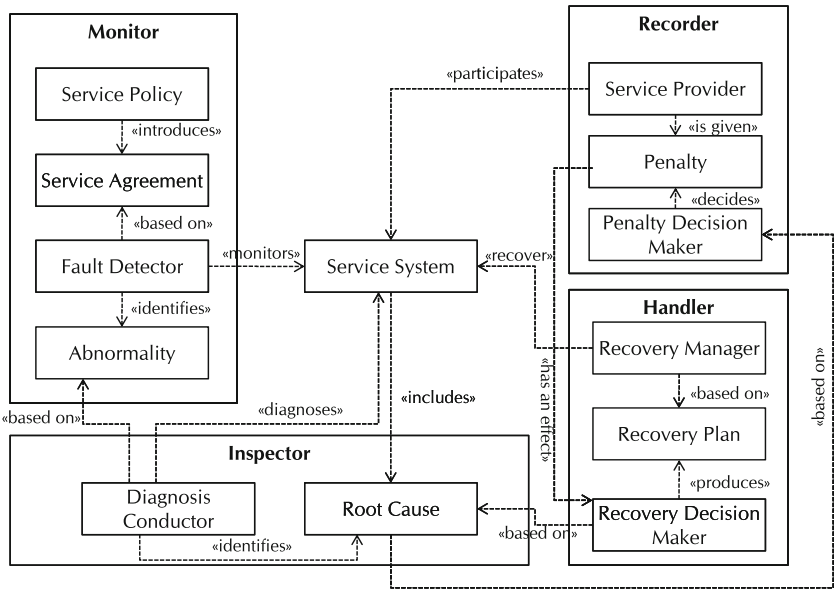

Fig. 3 Accountability framework

The component *Recorder* is to determine responsible players for the causes and relevant penalties for them. It consists of sub-components: Penalty Decision Maker, Service Provider, and Penalty. Penalty Decision Maker decides the specific penalty for responsible players.

We now elaborate on two components: Monitor and Inspector. These specialized components with multiple quality attributes and diagnosis levels from the accountability framework are shown in Fig. 4. In the figure, the main components, Detector and Diagnoser, are derived from Fault Detector in the Monitor component and Diagnosis Conductor in the Inspector of the accountability framework respectively. They are behavioral components which provide functionality, while Symptom and Causes are required information for detection and diagnosis. The Symptom and Cause are derived from Abnormality in Monitor and Root Cause in Inspector, and they are data used by the Detector and the Diagnoser.

In our design, a *Detector* is a software component to notify and to provide more detailed symptoms to diagnosers. Since published services in a service system are either atomic or composite services, detected abnormality is from either atomic or composite services (i.e. business process). Therefore, we need detectors for composite services which are deployed in a system and detectors for atomic services which may reside in another system. Moreover, we also need application level detectors which detect abnormality occurred in applications, as well as system level detectors.

A *Diagnoser* is a software component which gets detected symptoms and returns identified causes. In the diagnosis step, diagnoser requires three types of data: (1) more information related to the detected abnormality, (2) decision rules to identify causes, and (3) decision rules to determine further diagnosis.

A *Symptom* is a detected indicator in an abnormal service system. Symptoms are classified into *Exposed Symptoms* and *Hidden Symptoms*. Exposed symptoms are immediately observable and sent to diagnosers by detectors. However, hidden

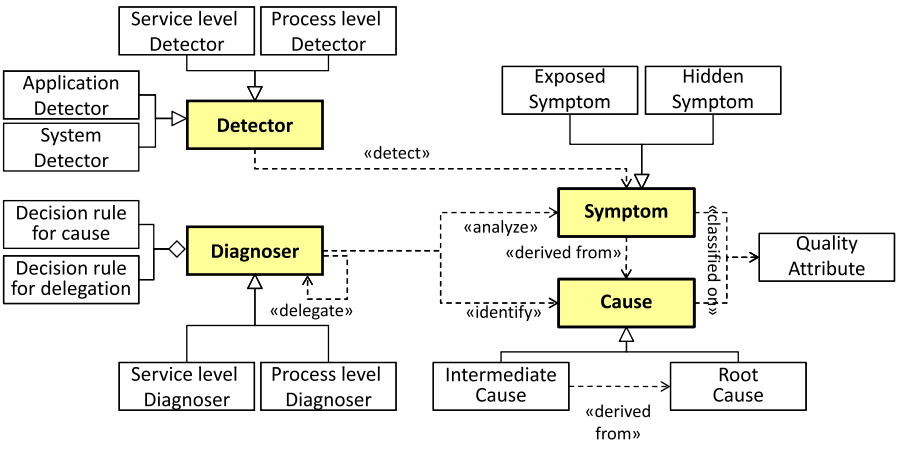

Fig. 4 Specialized concept from accountability framework

symptoms are not directly visible, but will be furnished by detectors only if diagnosers request for that information.

Causes are classified into intermediate causes and root causes. Intermediate causes may be produced by other causes so it needs more detailed diagnosis. However, root causes are the very origin of problems so that they should be reported. For example, a high CPU or memory utilization may be a possible cause of a slow service. But too many external requests may cause a high CPU or memory utilization, and an extreme high number of requests may be due to the malfunction of a firewall. In other words, a non-operational firewall may be the root cause and others are just intermediate causes.

The Diagnoser and Detector can be deployed as service containers or business process containers. Wherever they are deployed, the roles of the components are the same but the data they read are different. For example, if a Detector is deployed for a business process, it detects only exposed or hidden symptoms of the business process executions but not the executions of service implementation which are invoked from the business process. However, if the Detector is deployed on a server hosting an invoked service from the business process, the Detector only focuses on detecting abnormality in the service executions. Therefore, symptons, causes, and the diagnosis process for the corresponding diagnosis target are all circumstantial.

4 Accountability support using LLAMA

The above discussion provides a comprehensive requirement of an accountable service system. We now present the design of the LLAMA middleware which supports the dynamic and efficient e-service accountability. LLAMA includes the mechanisms to route and to monitor e-services, and dynamically adapt by rerouting service invocations away from faulty services. LLAMA is built on top of the open source Mule Enterprise Service Bus (ESB) (MuleSource 2007) that provides routing and interception mechanisms. We extend Mule to the LLAMA Accountability Service Bus (ASB) to provide the accountability support. Although the current LLAMA implementation is specific for e-services, we believe the design and the architecture can be used to support other types of accountable services (including human, machine and professional services).

4.1 LLAMA components

Figure 5 shows the elements of the LLAMA accountability middleware, as well as service client and service providers.

For the *Monitor* and *Inspector* components, LLAMA provides the Accountability Authority (AA) component and the Agent component that interact with each other in a sequence of detecting and diagnosing activities. To be more effective, AA assigns a minimum number of accountability agents to monitor and interact with a number of services.

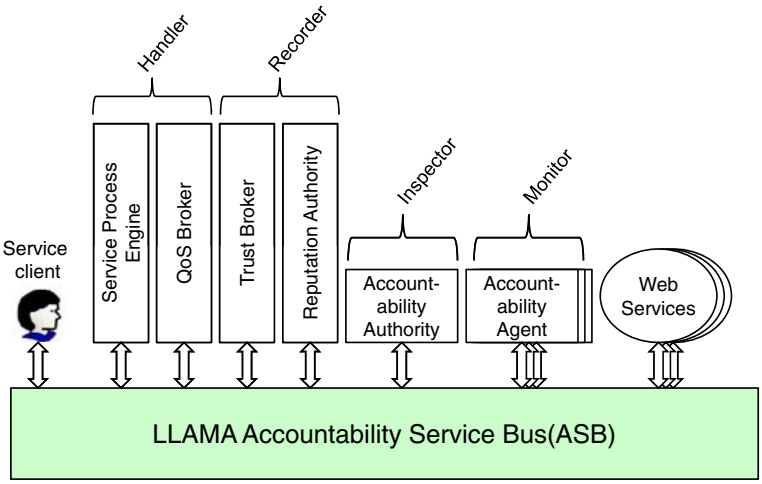

Fig. 5 LLAMA components

- *Accountability Agents* are used to monitor the behavior of Web services to which they are assigned and report monitoring information to the AA.
- *Accountability Authority* is responsible for the diagnosis of services with problematic behaviors in order to ensure that the process produces the desired outcome. Moreover, it is responsible for initializing a run-time process reconfiguration when the original process fails to produce desired outcome. The current implementation of AA adopts the Bayesian network model for determining probable causes.
- The *LLAMA ASB* provides the infrastructure for easy and scalable integration of system components and services in the process. That is, it provides fundamental supports for (1) dynamic routing, via a routing table and a performance interceptor, (2) efficient and customizable profiling and logging components to report performance data regarding services and the host, and (3) a configuration gateway (CGW) to support the configurability of external service components.

For the *Handler*, LLAMA provides *Service Process Engine* and *QoS Broker* to configure process and to recover from diagnosed problematic situations.

- *QoS broker* provides process planning and service selection, in order to assist the service requester in meeting the end-to-end QoS requirements for a service process. That means, the QoS broker provides the most appropriate path which can be replaced for given causes when problematic situation is diagnosed. The broker's jobs include:
  - Tracking, which is responsible for storing both functional and QoS data about various service candidates.
  - Planning, which composes a business process at the functional level based on various requirements.
  - Selection, which chooses specific services to fulfill each specific functional need based on the user's quality of service requirements.

- A *service process engine* for service process orchestration. Such an engine can be used by the service requester to coordinate a service process.

For the *Recorder*, LLAMA provides *Trust Broker* and *Reputation Authority* to maintain the reputation information for services.

- *Trust brokers* are responsible for evaluating, aggregating, and managing the reputation of services. Service's reputation is a QoS parameter that affects the service network composition: services with better reputation are more likely to be chosen. Once a service provider who is responsible for a problem is determined from diagnosis phase, corresponding Trust Broker records the information for future references.
- *Reputation authority* is queried by the trust broker for information regarding reputation information for various services. Trust Broker is to suggest trustable services for requests within a local region, whereas Reputation Authority provides globally trustable services by interacting with Trust Brokers.

4.2 Accountability operations using LLAMA

Figure 6 shows how components of LLAMA middleware interact with each other to provide service accountability. When a service consumer requests a service, the LLAMA ASB records the information of the transaction such as response time or

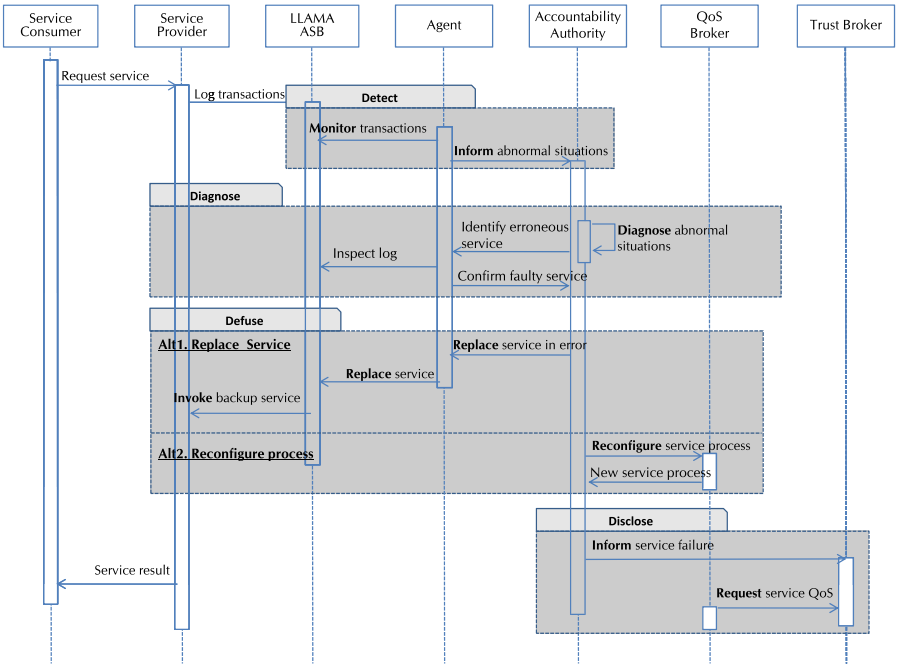

Fig. 6 Behavioral model of LLAMA components

throughput. In the detection phase, an agent is responsible for several services on the LLAMA ASB, monitors the transaction logs within a predefined time interval, and informs any detected abnormal situations to AA. In the diagnosis phase, AA identifies suspicious services that are probable root causes based on the Bayesian network model, and asks corresponding Agents for confirmation of the abnormal situation. Once the root causes are identified, AA makes a recovery plan which is to replace a service or reconfigure the whole or partial service invocation paths of the process. To replace a service, ASB reroutes all service requests for it to a substitute service by using the dynamic router, whereas QoS broker chooses a backup process to reconfigure the process. Finally, faulty services creating root causes are recorded in the Trust Broker which provides QoS information to QoS broker for future service selections.

An example of accountability operation cycle is now shown for the QoS, performance attribute. Assume that there is a business process using several web services which are deployed on LLAMA ASB. During execution, agent recognizes that a web service is not working on time, and reports it to AA. In response, AA begins identifying suspicious services which may be the origin of the fault, and asks other agents about the response time of other services. Using the reports from all agents, AA identifies potential problematic services and produces the most likely faulty service(s) based on the Bayesian network reasoning. AA then asks agents to double-check the execution status of those service(s) in doubt. If an agent finds a strong evidence to confirm faulty service(s), AA will replace those service(s) and ask LLAMA ASB to reroute future service requests to available replacements.

For other services that are not deployed on LLAMA ASB, they can still be monitored and diagnosed if they provide required information to an agent. It is necessary for agents to be allowed to monitor the status of service executions and to communicate with AA on the status so that AA can detect and diagnose the exact fault in service executions.

5 Related work

There are several service quality concepts which are related to accountability, as shown in Table 1. Their goals and constructs may appear to be similar to accountability. A close analysis, however, shows that accountability has its own goals and values.

Autonomic computing is an approach to develop systems that are capable of self-management, using self-configuring, self-healing, self-optimizing, and self-protecting (IBM 2006). Each of the self-* methods is implemented with four operations in sequence; monitoring, analyzing, planning, and executing. These operations are analogous to those defined in accountability. However, autonomic computing is motivated more from the goal of self management, whereas accountable service computing is motivated to provide reliable and transparent services. Even if a service is managed in a fully autonomic manner, it may not be considered to be accountable because accountability demands additional quality transparency such as

Table 1 Comparison of QoS issues

Quality attributes	Objective	Technical challenge	Methods
Autonomic computing	Make services self-managed	Optimization (pro-active decisions)	Self-config., self-protection, self-healing, self-optimization
Security	Threat resistance	Algorithm complexity (resource integrity)	Encryption, firewalls, securing channels
Safety	Failure avoidance	Property validation	Program analysis
Trust	Service trustworthy	Reputation collection, collusion analysis	Recommendation
SLA	QoS agreement	Contract	QoS measurement
Accountability	Problem identification	Causality and probability	Diagnosis and reasoning

determining responsible party for the given problem, which are not typically presented in autonomic computing.

Security in computing systems means protecting systems from unauthorized access, use, disclosure, disruption, modification, or destruction (Pfleeger and Pfleeger 2003). Security focuses on defending against threat, whereas accountability considers diagnosing threats for security related problems. It also considers a comprehensive set of quality attributes beyond security. Hence, a secure service may or may not be accountable.

Safety in computing systems is a condition where theories and engineering approaches are defined to prevent foreseeable accidents and to minimize the result of unforeseen ones (Leveson 1995). The primary concern of system safety is the management of hazards: their identification, evaluation, elimination, and control through analysis, design and management procedures. It can be differentiated from accountability in the similar way that security is compared to accountability. Safety can be enhanced by accountable service computing, but it is not the main objective of accountability.

Trust is the subjective probability by which an individual expects that another individual performs a given action on which its welfare depends (Josang et al. 2007). Trust is related to reputation in SOA, where consumers often rely on measures of reputation to acquire trustable services. Services with high trust would yield less problem during invocation, and accountability framework can be used to provide trustable services. By choosing services with high trust, service systems tend to be more stable and provide high QoS, but it does not necessarily mean that those systems are accountable.

Service level agreement is a specification of contracting between service providers and consumers and it is generally accompanied with contracts on QoS (Papazoglou 2007). SLA is focused on defining agreements between two parties, however accountability is to identify problems and resolve them based on the specification of SLA.

From above discussions, we can see, some notions of accountability also appear in related concepts such as autonomic computing, security, safety, trust, and SLA. However, accountability has different motivation and focus. It is to promote accountable services and focuses more on investigating problem. Accountability therefore is a more comprehensive quality aspect than security, safety, and trust.

6 Conclusion

To guarantee service quality and delivery, we propose the service accountability management framework to detect, diagnose, defuse and disclose the root cause of a problematic service process. The accountability support is very important for SSME-based systems since many service processes utilize external service providers to deliver part of the service functionalities. Therefore service-oriented systems must have mechanisms to ensure that every internal or external service is accountable of its performance, in order to provide the end-to-end QoS of the whole service process. In this paper, we have reviewed the requirements of a comprehensive accountable framework, designed the components in an accountable services architecture, and presented an accountability management methodology. We also present the LLAMA ASB project which is an agent-based middleware to support the monitoring, diagnosis, and reconfiguration of e-services. The ASB design is useful for all types of services to ensure the quality of service in SSME-based systems. By deploying services on an ASB, we can make services and service processes easier to monitor, to diagnose and to manage. In the future, we plan to study how to adopt the ASB concept in SSME for delivering accountable services in real world, not just in SOA systems.

Acnowledgments K. J. Lin was supported in part by a Visiting Fellow Grant (96-2811-E-001-003) from the National Science Council of Taiwan, ROC for his visit at the Academia Sinica. S. H. Chang was supported in part by the Korea Research Foundation Grant funded by the Korean Government (MOEHRD) (KRF-2007-357 D00218).

References

Sarbanes-Oxley Act (2002) http://www.sec.gov/about/laws/soa2002.pdf

Ardissono L, Console L, Goy A, Petrone G, Picardi C, Segnan M, Dupre DT (2005) Enhancing web services with diagnostic capabilities. In: Proceedings of the Third European Conference on Web Services (ECOWS '05), p 182

Baresi L, Guinea S (2005) Towards dynamic monitoring of WS-BPEL processes. In: Proceedings of International Conference on Service-Oriented Computing (ICSOC 2005). Springer, pp 269–282

Bichler M, Lin KJ (2006) Service-oriented computing. IEEE Comput 39(3):99–101

Horsch K (1996) Results-based accountability systems: opportunities and challenges. The Evaluation Exchange II (1) http://www.gse.harvard.edu/hfrp/eval/issue3/theory1.html

Huhns MN, Singh MP (2005) Service-oriented computing: key concepts and principles. IEEE Internet Comput

IBM (2006) An architectural blueprint for autonomic computing http://www-01.ibm.com/software/tivoli/autonomic/

Johnson DG, Mulvey JM (1995) Accountability and computer decision systems. Commun ACM 38(12):58–64

Josang A, Ismail R, Boyd C (2007) A survey of trust and reputation systems for online service provision. Decis Support Syst 43

Leveson NG (1995) Safeware: system safety and computers. Addison-Wesley, Reading

Lin KJ (2007) Accountable services. In: IEEE International Conference on e-Business Engineering (ICEBE)

Mohan C (2007) IBM Research global technology outlook (GTO) https://www.almaden.ibm.com/u/mohan/abstracts.html

Morgan G, Parkin S, Molina-Jimenez C, Skene J (2005) Monitoring middleware for service level agreements in heterogeneous environments. In: Challenges of Expanding Internet: E-Commerce, E-Business, and E-Government. Springer, Berlin, pp 79–83

MuleSource (2007) The mule project http://mule.codehaus.org/display/MULE/Home

Nissenbaum H (1994) Computing and accountability. Commun ACM 37(1):72–80

O'Brien L, Bass L, Merson P (2005) Quality attributes and service-oriented architectures. Technical Note CMU/SEI-2005-TN-014

Papazoglou M (2007) Web services: principles and technology. Prentice-Hall, Englewood Cliffs

Pfleeger CP, Pfleeger SL (2003) Security in computing. Prentice-Hall, Englewood Cliffs

Wang G, Wang C, Chen A, Wang H, Fung C, Uczekaj S, Chen YL, Guthmiller WG, Lee J (2005) Service level management using qos monitoring, diagnostics, and adaptation for networked enterprise systems. In: Proceedings of the Ninth IEEE International EDOC Enterprise Computing Conference (EDOC '05). IEEE Computer Society, Washington, DC, USA, pp 239–250

Zhang Y, Lin KJ, Hsu JY (2007) Accountability monitoring and reasoning in service-oriented architectures. Journal of Service-Oriented Computing and Applications (SOCA) 1(1)