Accountability as a Service for the Cloud
From Concept to Implementation with BPEL

Jinhui Yao
School of Electrical and Information Engineering
University of Sydney
jinhui@ee.usyd.edu.au

Shiping Chen
Networking Technologies Laboratory
CSIRO ICT Centre
Australia
shiping.chen@csiro.au

Chen Wang
Networking Technologies Laboratory
CSIRO ICT Centre
Australia
Chen.Wang@csiro.au

David Levy
School of Electrical and Information Engineering
University of Sydney
dlevy@ee.usyd.edu.au

John Zic
Networking Technologies Laboratory
CSIRO ICT Centre
Australia
john.zic@csiro.au

Abstract—The emergence of computing resource provisioning known as the Cloud has revolutionized the modern day computing. It has provided a cheap and yet reliable alternative computing platform for whoever with huge needs for computing resources. Moreover, its charm has been further reinforced by the concept of Service Oriented Architecture (SOA), which allows your business processes conducted by services to be flexibly integrated with other collaborating services to form new value-added business products. In this sense, the enormous computing capability and transmission bandwidth of the Cloud make it an ideal platform to be a serviceful computing environment. However, the overall correctness of the SOA depends on the correctness of all participants. As systems like this usually span multiple administration domains, concluding the faulty service and making the provider responsible become a challenging task. In this paper, we propose a novel design to enforce strong accountability in the SOA deployed in the Cloud. With this accountability, not only faults can always be bound to their causes, this binding is always provable and undeniable. We have implemented a demonstrative system to show its effectiveness in real practice.

Keywords—cloud computing; accountability; service oriented architecture; trustworthiness

I. INTRODUCTION

The emergence of computing resource provisioning known as the Cloud has revolutionized the modern day computing. It has provided a cheap and yet reliable outsourcing model for whoever with huge needs for computing resources. The clients can rent their desired computing and storage resources to conduct their business processes on a pay-as-you-go basis while leaving behind all the maintenance works. This outsourcing model cast significant attractions to the enterprises that wish to minimize their IT budgets.

On the other hand, the concept of Service Oriented Architecture (SOA) allows flexible and dynamic collaborations among different service providers. A service provider can either directly offer his service to end users or compose the service with other collaborating services to form new value-added business products [20].

The wide adoption of the concept of SOA further reinforces the impact of the Cloud computing for that, the enormous computing capacity and internal/external transmission bandwidth of the Cloud make it an ideal computing environment for deploying or composing services. Different service providers in the Cloud will collaborate with each other to form business processes. These participators in the process behave according to the predefined and mutually agreed upon business logic and Service Level Agreement (SLA). The overall correctness of the operation of the business process largely depends on the compliance of each of the participators. Any incompliance/violation from the participators is regarded as violations which, to various extents, can negatively affect the value of the business product associated with this business process. Therefore, a robust mechanism is needed to detect, log and resolve those incompliance. Once the cause of the violation is found, each violator should be hold accountable for his misbehaviour, i.e. being penalized according to its impact. This mechanism to assure the compliance to the respective business logic and SLA is critically essential for such outsourcing and collaboration to work.

Indeed, in order to build a trusted serviceful computing environment in the Cloud, the ability to deal with the compliance is critical and the lack of this has prevented many potential customers from using the Cloud. We here adopt the informal definition of trust (based on comments

---

1 This document serves as the technical paper for IEEE Service Cup at IEEE International Conference on Web Services 2010.
something can be trusted when (a) it can be unambiguously identified; (b) it operates unhindered and (c) the user has either first hand experience of consistent good behaviour or knows someone who can vouch for consistent good behaviour. A more formal trust definition is presented in IETF RFC4949 Internet Security [17].

However, it is difficult to validate the compliance of all the participators involved in the collaboration for the fact that, the composed business process usually spans several administrative domains, each of which will have its own interests and priorities. Knowing that an admission to a violation may cause penalties in some form, it is conceivable that a participator may intend to deceive and hide this fact. This represents a challenge and obstacle to the adoption of cloud computing for inter- and intra-enterprise collaborations.

We here regard one's capability to assure its associated compliance to regulations or agreements as its trustworthiness. Based on the notions of trust presented, a trustworthy system [24] is defined as: a system that is already trusted, and continues to warrants that trust because the system’s behaviours can be validated in some convincing way. In this paper, we propose a novel design to achieve Trustworthy Serviceful Cloud (TEC) by enforcing strong accountability. In such a system, the root of a violation can always be identified and associated with responsible (or guilty) entity(s), and this association is supported by non-disputable evidence. Then, we propose a method to enforce such trustworthiness with standard definitions of business logic and service level agreements. Finally we deploy the system into a computing cloud to evaluate its effectiveness.

### II. A MOTIVATING SCENARIO

Consider an online, one-stop loan application service: customers place their loan application and subsequently, receive the loan offer which is the cheapest among multiple banks. The process involves interactions between four parties, as shown in Fig. 1. First, a one-stop loan application service allows customers lodge the application and fill in their personal information. His personal information will first be used to obtain a credit score from the credit rating authority, and then the score is attached with other personal information to be sent to the loan bidding company. The bidding company forwards the application to multiple loan companies, and select the cheapest offer available (if any) to return to the applier.

We use the above scenario to exemplify an ordinary business process with service composition. Each service node takes job request, finishes the job and reply with the result. In such process, a failure from any of the nodes would result complete failure at the final output, and usually the source of the failure may not always be able to be logically determined.

For instance, a loan applier, Bob, later on finds out that he can be offered a cheaper loan through direct contact with one of the loan companies (claimed to be involved by the bidding company), proving that the one-stop loan application service has failed its promised service outcome. Bob is not able to know if it is the credit rating authority which gave him a bad rating, or that loan company is actually not involved during the bidding. Intuitively, Bob may hold the one-stop loan application service responsible. At this point, the application service may attempt to alter its system record to prove its own innocence and push the blame to the bidding company. In turn, the bidding company could also do the same, and shift the blame to either the credit rating authority or to the loan companies. It can be seen even in this simple example that a mechanism is required to prevent this “buck passing” or denial of failure. This mechanism or service is essential to control the correctness of an SOA-based business process. Once a node is determined to have failed to meet expected behaviours, actions must be taken so that it can be excluded or replaced.

### III. ACCOUNTABILITY

Accountability is a concept to make the system accountable and trustworthy by binding each activity to the identity of its actor [3]. Such binding should be achieved under the circumstance that all actors within the system are semi-trusted. That is, each identified actor may lie according to their own interest. Therefore the bindings must be supported by provable or non-disputable evidence.
In our approach, accountability can be incorporated into activity based process by requiring the actor (conductor) of the process to log non-disputable evidence about the activities in a separate domain from the domain of its own. Fig. 2 shows an example of such incorporation. In the example, domain A is required to perform logging operations before and after conducting the activity in its process. The evidence needs to be logged should contain enough information to describe the conducting activity. In the simple case in our example, intuitive enough, the evidence should include the states of the factors concerning the start of the activity (e.g. input variables) and the factors concerning its completion (e.g. output value).

As aforementioned, the logged evidence needs to be non-disputable so as to undeniably link the activity to its actor. That is, if by the logger or any other entity, the evidence can be proven to be associated with an activity conducted by a domain, no other entities can prove otherwise.

To achieve this, we assume the employment of PKI in all the domains in our example, so that each of them has its own associated public-private key pair issued by certificate authorities. The logging procedure is as follows: First, the logger (domain A) signs the evidence (E) by its private key (KA-) to create a digital signature of it (SA). The evidence and its signature are then logged at a separate domain (domain B). When received, domain B creates a receipt by signing domain A’s signature with domain B’s private key (KB-). At last, the receipt (SB) is sent back to the logger (domain A) in the reply. Through this procedure, since the digital signature is un-forgeable, the signature of the evidence (SA) and the receipt (SB) enable both domains to prove that domain A has logged this evidence at domain B. In another word, even though neither domain trusts the other, both of them can prove its own correctness with the signature it has kept.

IV. SYSTEM DESIGN

A. Accountability as a Service

As illustrated in previous section, two domains need to be involved for logging the evidence, and the logged evidence should be analyzed continuously so as to evaluate the logger’s correctness. In a cloud computing environment where services are dynamically introduced and dismissed, applying this concept can be hard. Some literatures address this issue by deploying logging and analyzing mechanisms in all of the services involved in the composition. In such systems, service nodes have to volunteer to sign on each others’ evidence, and send auditing/challenge requests to each other from time to time in order to verify each other’s compliance. This can result in a chaotic situation where there is no global (or overall) common knowledge of the system’s state. This in turn results in inconsistent individual logs, and potentially a large amount of message exchanges caused by entities trying to discover this global state.

![Figure 3. Overall system design](image)

In our design, we propose to use a central accountability service to enforce accountability on all the participating business services. Fig. 3 shows the system model of our Trustworthy sErviceful Cloud (TEC). The space in the Cloud has been split into two domains: the Accountability Service Domain (ASD) and the Business Service Domain (BSD). In the BSD, business services (BS) compose with each other to conduct complicated business processes, like the one-stop loan application service composition example we used. Each service in the BSD keeps a close association with the accountability services (AS) in ASD so as to ensure that the BSes are held accountable.

The AS here can either be provided by the Cloud, or any other trusted third parties as long as they obtain no benefit of assuring the correctness of the business services, and it play a neutral role in the Cloud. Therefore, this topology satisfies the concept of accountability previously discussed. The misbehaviors of a service in either domain inevitably mean that it is willing to take the risk that it will be exposed in another domain. This mutual constraint on services in the two domains is the main strength of our approach to achieving trustworthiness. Below we list the core functionalities of the AS node:

i.) Evidence logging: Non-disputable evidence associated with the activities conducted must be logged in real-time. Such history logs should be sufficient enough for any later disputes with respect to the predefined correctness of business processes.

ii.) Monitoring and Auditing: Through the analysis of the activity events, the system's state is continuously monitored to provide underlying participating services or end-users monitoring information to assist their operations.

iii.) Fault resolution: Once an exception or violation is detected or reported, the root cause should be discovered in a provable manner and actions taken e.g. forbidding the violating service from further interaction until the dispute/violation is cleared.
In the following sections, we will illustrate the detailed operations of TEC according to the core functionalities identified above.

B. Evidence Logging with BPEL transformation

Because the domains involved in the process need to log evidence for their activities, logging operations must be incorporated into the logger’s business process. Essentially, this incorporation should require minimal modifications to existing implementations and can be done through automation. In this paper, we propose an approach to allow this automation for the business processes which are orchestrated by process descriptive languages. Process descriptive languages define the business processes that involve activities associating with multiple external/internal services. This definition is usually in the form of scripts, which will be interpreted by orchestration engines [7] to conduct the process accordingly.

A good example of the process descriptive language will be Business Process Execution Language (BPEL) [9]. BPEL models the business activities into several basic activity types, and then compose those types to describe the whole process. The core activity types include: i) receive, receiving the request from a requestor, this activity type will specify the variable to assign the input data to; ii) invoke, invocation to an endpoint (service), invoke activity type will specify the variable used as the input and the variable used to store the output data for this invocation (if there is any); iii) reply, associating to a receive activity, a variable will be specified to be returned to the requestor as the result.

When executing processes defined in BPEL, it is these three activity types that describe the interactive activities with services involved in the process. Intuitively enough, in order to preserve evidence about business process, these are the activities that need to be logged.

To add logging activities into the process, we can insert invoke activity types into the BPEL script to invoke certain endpoint (logging service) with the evidence to be logged. And due to the distinct natures of receive, invoke and reply activity types, the rules used to decide the insertion locations are in fact quite straightforward. For the receive activity, an invoke should be inserted right after it, to log the variable carrying the input data the receive activity has just assigned. For the invoke activity, one invoke should be inserted before this activity and another to be inserted after. The two inserted invoke activities are used to log the input data and the output data of the invocation respectively. And finally for the reply activity, an invoke needs to be inserted right before it to log the result data which is about to be returned to the requestor. The invocation endpoint for the invoke activities inserted should either be a service in the same domain of the logger, or a trusted third party nominated by the logger, which in turn signs the evidence on the logger’s behalf and forward the signed evidence to the AS.

To further illustrate this transformation process, we have presented an example in Fig. 4 Fig. 4-a shows the graphical view of an ordinary sample BPEL created by Eclipse BPEL plugin. This simple process is started by receiving an input (ReceiveInput), then a partner link (collaborating service) is invoked in turn (InvokePartnerLink), and finally, reply the result to the client (ReplyClient). Fig. 4-b is the BPEL script after the transformation. We can see in Fig. 4-b four logging invoke activities (the InvokeLogging serie) have been inserted, one after the “ReceiveInput”, one before and one after “InvokePartnerLink” and one before “ReplyClient”. Because BPEL is entirely based on xml schema, any xml schema parser will be capable of analysing and inserting activities into it. The details of our implementation of such BPEL transformer will be shown in the evaluation section.

After the BPEL transformation, the logger needs to register with AS to start the logging. Certain documents should be submitted in order to give the AS sufficient knowledge about the logger. These documents can include: i) BPEL scripts, describing the business process so that the AS can verify if the activities the logger is conducting is legitimate; ii) Service Level Agreement (SLA) definitions, stating the logger’s guarantees on the quality of service (Qos) and the method to evaluate the compliance to them with the evidence logged (e.g. WSLA [25]); iii) identification proof, certificates issued by authorities with the logger’s private key to prove its identity (e.g. X.509). iv) the WSDL of a trusted archive service nominated by the logger to receive the log receipts issued by the AS.
C. System Monitoring and Auditing

Service providers participate in the business processes are constrained by the pre-established Service Level Agreement (SLA) and business process logic (BPEL). The behavior of the underlying services should be compliant with those definitions. Validation on this compliance is carried out through the analysis on the evidences submitted to the AS. This analysis aims to determine if the activities conducted or proposed by services i.) satisfy the SLA; and are ii.) legitimate with respect to the business logic. Note that this verification must be provable and non-disputable so that, the result of the analysis is sound and convincing.

Fig. 5 displays the inner structure of the AS node. Analysis on the evidences is done by two components. First, the Monitoring Logic, which continuously analyzes the submitted loggings to find if they can prove the logger’s correctness. The second one is the Auditing Logic, which periodically (or upon request) audit all the evidences and previous analysis results stored in the archive in order to identify suspicious activity patterns (or collect requested evidences). The main purpose of the audit is to identify or solve those violations or errors which can be uncovered only after a long time (or by reporting). For example, when the customer later on finds out that the loan offer is not the cheapest and reports to the AS. In this case the related evidence in the Archive will be audited to track the source of the fault.

The Business Logic (BPEL) and SLA registered at the AS are used to generate the Monitoring Logic, Auditing Logic as well as the Exception Logic.

As previously illustrated, BPEL defines the correct interactions between the services. In particular, it will define which service should be invoked for what purpose. This information can be used when monitoring to evaluate the correctness of the inter-activities conducted by collaborating services. For instance, in the one-stop loan application service composition example we used previously, if the Loan Bidding Company chooses not to involve Star Loan Company in the bidding, the expected loggings (according to the BPEL) from the Star Loan Company will never be received by the AS since that service is not invoked during the bidding, thus the business process error will be noticed.

SLA definitions need to contain the admitted obligations from service provider, the methods to evaluate the compliance to those obligations, and the compensation rules for violations. SLA may be defined in several standard formats. In this paper, we use WS-agreement [25] (WSLA) to illustrate our approach. A sample WSLA is shown in Table 1. The sample WSLA has two parts. The first part defines a ServiceLevelObjective which is a guarantee offered by the CreditCheckCompany. The guarantee states that the response time must be less than 1.6 seconds. The second part elaborates the method to validate the compliance to the previous guarantee, that is, by using the value of the ReceiveTime (the time when CreditCheckCompany receives the request) to minus the value of ReplyTime (the time when the result is ready to be replied to the requestor).

D. Fault Resolution

Once an exception is noticed, or reported, the root (causer) of the exception must be determined according to the knowledge the AS has about the composition. Once the exception is successfully linked to a guilty service, certain
actions need to be taken by the AS in response to stop the misbehavior or minimize its impact.

Violations to the WSLA should be handled according to the compensation rules defined in the WSLA. The most common form of compensation is through penalty, in this case, a penalty report will be constructed and sent to the compensator. The penalty report should contain the details about the violations and the evidence to support these accusations.

In case of violations to the business process, depending on the severity of the violation, different procedures can be followed. In general, the AS may first send warnings to the violator and temporarily tolerate it, until a violation limit is reached, the AS may send notices to all other service nodes to dismiss the violating service. Again, the notice should contain non-disputable evidence to convince them about the violation.

V. Demonstration System in Amazon EC2

The Amazon Elastic Compute Cloud [5] is a computing resource provisioning service that charges the user according to the CPU usage. Users can deploy their services and business processes in the computing instances they rent in EC2. Computing instances can communicate with each other with speed close to a LAN. While making use of this computing environment, users can be quite concerned about other collaborators’ compliance as well as their own. In this section, we will elaborate our demonstration system in EC2 to achieve TEC.

A. Implementation details

Our implementation used Apache Tomcat 5.5 [8] as our Servlet container, and Axis2 1.5 [6] as our web service engine. We chose BPEL for defining the business logic and WS-Agreement (WSLA) for SLA in our AS nodes. Apache Orchestration Director Engine (ODE) [7] has been used to conduct the business process defined in BPEL.

The BPEL parser and transformer are implemented in JAVA by using W3C document object model (DOM) [26]. It turns out that it is quite handy to incorporate the logging activities into the business process, because the insertion rules we defined previously are straight forward to apply. Transformed BPEL scripts can be redeployed by simply dropping into the ODE process folder, ODE will realize the modifications and use the new process to retire the out-dated one. Overall, we find incorporating the concept of accountability into a running business process is convenient and can be done with little impact or modifications on the business process implementations. The operations involved in BPEL transformation can be seen in the video provided at the end of this paper.

In order to allow the AS node to learn the context in BPEL and WSLA, the parser was also used in AS to extract the needed information from those definitions. The Archive is implemented as a PostgreSQL database [21] which is accessed through JDBC interface.

B. Demonstration System

The AS and the five underlying business service nodes have been deployed on six standard computing stances in Amazon EC2. These are virtual machines with computing power equivalent to 1GHz CPU and 1.7GB memory. We then used this system to simulate the scenario that AS agent is deployed in the cloud computing environment to observe the correctness of the operations of services running in the environment.

To simulate our one-stop loan application service example, the composition has been defined in the BPEL scripts with logging activities incorporated and deployed in each of the business service node. The message in the communications are simply random data, the receiver of a message will encrypt the data with the node’s private key, and produce the result cipher text as the return data or the input data used to invoke other services. These messages here represent the loan application request, credit score or loan offers, and the encryption operation creates some processing latency.

The five nodes thus form an ordinary business workflow with embedded logging operations to log evidence for all receive, invoke, and reply activities. The WSLA definitions used for each of the services are similar to the example in table 1, except the value of response time guarantees are different for specific business services. The certifying services as well as the receipt archiving services are deployed in the same computing instances of their associated business services.

A client (at the University of Sydney) sends request to the first service node – one-stop loan application company, and the process goes on until the loan offer is returned back to the client. Note this experiment demonstrates the nature of service composition, in that the correctness of individual nodes directly affects the overall correctness. Any error introduced by any single service node may make final outcome incorrect.

Now with the evidence logged with the AS, any source of errors can be efficiently discovered. In this experiment, the operation involves a deterministic encryption process. Given an input, the output must be constant. In real composition, operations will be more complex than our example, however, as long as it is deterministic, verification is possible with proper evidence. This is the main benefit of our non-disputable logging.

C. Monitoring Console

In order to visualize the capability of the AS in monitoring the status of underlying business process, we have implemented a monitoring console in the AS node to show the information it has collected and concluded. A screenshot of the monitoring console is show in Fig. 6.
The console consists of four panels. The **Document Panel** displays the documentations registered by the services participating in the business process. Like in the figure, each of the five services has registered its BPEL, WSDL and WSLA. The **Process Panel** displays the overview of the business process, animation is used to show the interactions between the underlying services so it can be seen that at which stage the process is being conducted. The **Service Status Panel** displays the status and statistics of an individual service. Those include general Qos such as response time and up time, transmission speed with associated PartnerLinks, and fulfillment to the SLAs. The **Overall Status Panel** displays the status and statistics of the whole business process. It shows the number of jobs have been done, the stage of current process and the healthiness of the process concluded by the AS. A demonstration video showing the use of this monitoring console is also provided at the end of this paper.

## VI. RELATED WORK

Service compliance has been decently studied in recent years. Its difference to conventional Qos study is that, apart from gathering the evidence, service compliance requires these evidences to be validated against certain agreements or regulations in automation. In works like [18][20][10], an embedded monitoring and validating logic is incorporated into the deployed service’s BPEL. The logic is generated according to SLA and implemented in the form of try-catch blocks to check required attributes. In [15], the authors introduced a standard language to describe the validation logic for the SLA. This language is used to generate checking scripts (code) to be inserted into the service engine as WS handlers. Some other works have attempted to conduct compliance validation offline. Studies like [14][17][27] require the service providers to archive the events. When a problem occurs, those event trails are analyzed to find the responsible entity. The main difference of these approaches to ours is that they assume the service entities will strictly monitor their own behaviors and confess to their own incompliance even when such admission can lead to penalties due to SLA violations. However, our approach does not make this assumption, as all service entities expected to behave in any possible manner.

Some other works in the area share the point with us that system correctness is verified through mechanisms that are provable and undeniable. [1][3] require service nodes to maintain temper-evident logs about their receiving and sending requests as well as service state digests. Correctness is verified through service nodes volunteer to send challenge and audit requests to each other from time to time to collect those logs for validation. [24] is an attempt to achieve secure accounting of utility storage, and it requires the storage service provider as well as the client to sign for every request so that the amount of usage can not be denied by either party. [2] uses a similar approach to [24], however it uses such a method to achieve certified accountable tamper-evident storage service. Instead of undeniable usage, it uses the signed actions logged to verify the correct state of stored data. Any changes cast by both client and the service is provable and undeniable. The concept of our work is generated from [11][12], in which the authors have proposed the idea that, accountability can be used for verification of
services’ correctness with regard to established service agreements. Their study has been well developed and refined in our work. We incorporate the AS into the business process through the transformation of BPEL scripts, and the compliance is validated against standard SLA definition.

VII. CONCLUSIONS

In this paper, we introduce the Accountability Service to enforce compliance on the service providers, who participates in business collaborations in the Cloud. This accountability mechanism can be conveniently incorporated into existing business processes defined with process descriptive languages (e.g. BPEL). With strong accountability enforced, we can build a Trustworthy sErviceful Cloud (TEC), where incompliance can always be concluded with provable and non-disputable evidence. We implemented TEC into Amazon EC2 with a sample business process which shows that, our design can enforce strong accountability in various aspects and is easy to deploy.

REFERENCES

[23] Trustworthy system definition, at IETF RFC 4949

Demonstration Videos:

Video 1: BPEL Transformation Example URL: http://www.youtube.com/watch?v=69pFVJk5oB4
Video 2: Monitoring by Accountability Service URL: http://www.youtube.com/watch?v=Js-5gDGJU5s