Modelling collaborative services for business and QoS compliance

Jinhui Yao†, Shiping Chen*, Chen Wang*, David Levy† and John Zic*

*Information Engineering Laboratory, CSIRO ICT Centre, Australia
{Firstname.Lastname}@csiro.au
†School of Electrical and Information Engineering, University of Sydney
{jinhui, dlevy}@ee.usyd.edu.au

Abstract—In recent years, we have witnessed a range of innovations in the ‘service’ related technologies, such as Software as a Service, Platform as a Service and Infrastructure as a Service. Along with the Service Oriented Architecture, companies can wrap their technological product as a service, to collaborate with others. Facing the ever-escalating global competition, such collaboration is crucial. The viability of this paradigm highly depends on the compliance and therefore the trustworthiness of all collaborators. However, it is challenging to achieve trustworthiness in such a dynamic cross-domain environment, as each participant may deceit for individual benefits. As a solution, we have proposed to enforce strong accountability to enhance the trustworthiness. With this accountability, incompliance can always be determined in a provable and undeniable way. In this paper, we extend our work by proposing a novel modeling of the collaborative business process. Based on this modeling, we thoroughly analyze the evidence and proving procedure needed for different types of compliance, and evaluate the extent to which those compliance can be indeed proved. We have implemented a demonstrative system to show its effectiveness in real practice.

Index Terms—accountability, compliance, trustworthiness, service oriented architecture, service collaboration

I. INTRODUCTION

In recent years, we have witnessed a range of innovations in the ‘service’ related technologies and concepts. Following the Software as a Service (SaaS), Platform as a Service (PaaS), Infrastructure as a Service (IaaS) and many more “as a Service” concepts have been proposed. Along with the widely adopted Service Oriented Architecture (SOA), companies, organizations can wrap various kinds of technological product they are offering as a service, to collaborate with services provided by others to form new value added business products. Facing the ever-escalating global competition, such collaboration is crucial to for their survival.

The correctness of the inter-organizational collaboration relies on the correctness of all participators, that is, if the collaborator is compliant to pre-defined business logic, or Service Level Agreement (SLA). It follows that, the viability of this paradigm highly depend on the trustworthiness of the behaviors of all collaborators. Here we adopt the definition of trustworthiness on IETF [15]: a trustworthy system is a system that is already trusted, and continues to warrants that trust because the system’s behaviors can be validated in some convincing way.

It is a challenging task to preserve trustworthiness in such a dynamic cross-domain environment, as each participator will have their individual priorities and interests. Given that admission to violations may lead to penalties in some form, it is conceivable that they may intend to deceive and hide this fact. Therefore, a mechanism to detect and prove incompliance is needed for this collaboration paradigm to prosper.

As a solution, we have proposed to enforce strong accountability to enhance the trustworthiness [17][19]. While this will be elaborated shortly in later sections, in short, accountability provides means to verify compliance according to evidence in a provable and undeniable way. In our past work, we have described the overall architecture of using Accountability Service (AS) to aggregate evidence and verify compliance. To utilize this design in the real practice, detailed methodology, both in form of theory and implementation need to be developed. This requires a decent study on the nature of the collaboration, or be more specifically, a generic modeling of the problem domain is needed. This model shall serve as the ground basis for conceptualizing higher level objects, and as a tool for analyzing and reasoning about the compliance issues in the collaboration.

In this paper we extend our work by developing a quantitative model to represent the horizontal and vertical structures of the collaborations involving multiple un-trusted parties. Using this model, we classify four types of compliance and determine the logging needed for their verification. Based on the compliance types, we extensively analyze and reason about the extent to which different compliance types can be verified in a provable using the evidence logged. Then we evaluate the practical effectiveness of the model and methodology proposed by implementing them in a collaborative business process.

II. SERVICE COMPLIANCE: A MOTIVATING SCENARIO

Inherited from our previous work, we use a loan application process as the running example in this paper. As shown in Fig. 1, the process requires the collaboration of five entities. First, a loan application web portal allows customers lodge the application and fill in their personal information. This 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 (Star Loan & Ocean Loan), and select the cheapest offer available to return to the applicant.
In this typical collaboration scenario, the overall correctness of the system depends on the correctness of all participants. As each of them may be interested to violate the collaboration rules for their own benefit or for avoiding possible penalties, the causer of a failure may be extremely difficult to determine. Therefore, a mechanism is required to prevent this denial of failure. This mechanism is essential to control the correctness of the business process established.

Accountability is a concept to make the system accountable and trustworthy by binding each activity to the identity of its actor [21]. Such binding should be achieved under the assumption that all actors 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 undeniable link the activity to its actor. 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. Any evidence logged must be signed by the logger and receipted by the loggee (i.e. Domain B). Through this procedure, since the digital signature is un-forgable, the signature of the evidence and the receipt enable both domains to prove the factor that domain A has logged such evidence at domain B.

With this concept, we propose to use a central accountability service (AS) to enforce accountability on all the participating business services (BS). 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. Each service in the BSD keeps a close association with the accountability services (AS) in ASD so as to ensure that the BS are held accountable. In this setting, the AS continuously receives logs and analyzes the evidence to verify the compliance of all the underlying participating services in the collaboration.

In our past work, we have illustrated the logging protocols and demonstrated how this design can be used to verify compliance effectively. So in the following several sections, we will describe the model we have developed based on this design to define the detailed methodology as what evidence need to be submitted and how compliance can be determined.

### III. Modeling the Collaboration

A collaborative business process may involve many service nodes from different administration domains. To clearly describe the settings of a collaboration, one needs to look at both its horizontal and vertical structure. With respect to a participating service, in the horizontal structure of the collaboration, this service interacts with all other participating services. Whereas in the vertical structure, this service may first, belong to a specific trust domain (e.g. a company) so as some other collaborating services, and second, have its physical service node(s) deployed in an infrastructure provided by other entities (apart from this company). This vertical structure contains much information essential for verifying one’s compliance. For instance, the service provider should not be blamed for the fault of the infrastructure provider. Our modeling intends to capture both the horizontal and vertical structure of the collaboration.

Different service providers collaborate with each other to form business processes. And small processes are integrated to form massive ones. We model the business process (P) formed through collaboration as a tuple

\[ P = (N, P, V) \]

where \( N \) is the service node involved in the process, \( P \) is the sub-process and \( V \) is the directed edge connecting them. The building block of a business process is the service nodes involved. The “node” here refers to the physical computing instance where the service application is deployed at. A service node is modeled as

\[ N = (D_{in}, D_{out}, F, L) \]
Where $D_{in}$ and $D_{out}$ are the input and output data of the service during execution (if there is any), $L$ is the physical location where the node is deployed (e.g. in the company, in the cloud). $F$ refers to the function of this service node, the internal computational logic, which is a sequence of activities ($A$) with the first action to take input ($A_{in}$) and the last action to emit output ($A_{out}$) (if there is any):

$$F = \prod_{i=0}^{n} A_i = A_{in} \ast \prod_{i=1}^{n-1} A_i \ast A_{out}$$

(3)

the product symbol ($\prod$) here refers to sequential or non-sequential (e.g. parallel) relationships between the activities. A service node may have a range of functions (computational logic) designed for different input data, here for simplicity, we model the node to be dedicated to only one function.

Every participant (e.g. company, organization) in the collaboration is regarded as service entity ($En$) which owns a group of service nodes, that is,

$$En = \{N_1, N ... N_3\}$$

and each entity may have its own trusted partners

$$T = \{En_1, En_2...En_i\}$$

In short, a business process consists of the service nodes provided by different service entities and inter-connected by directed edges while certain entities may be in the same trust domain (e.g. belong to the same financial group).

Corresponds to the ownership of service nodes and the trust structure in the collaboration, directed edges ($V$) could have many types. Broadly, they can be classified into three: a) inner edges ($V_{inner}$), edges between the nodes owned by the same entity; b) external trusted edges ($V_{trust}$), edges between the nodes owned by two mutually trusted entities and c) external un-trusted edges ($V_{out}$), edges between two nodes whose owner barely trust each other. For instance, the term means the inner edge from node $a$ ($N_a$) to node $b$ ($N_b$): $A_{N_a}^{N_b} \leftarrow inner \ A_{in}$.

We can elaborate this model with our running example. For instance, the One-stop Loan App Company is our $En_2$, which has only one service node deployed in Amazon EC2. It can be expressed as $En_1 = \{N_1\}$ and the node

$$N_1 = \{D_{application}, D_{result}, F_{loan}, EC2 - 184.72.253.241\}$$

Similarly, let us suppose the Credit Check Company is our $En_3$, which has only one service node deployed in Microsoft Azure, it can be expressed as $En_2 = \{N_2\}$ and the node

$$N_2 = \{D_{person}, D_{rating}, F_{credit}, Azure - 192.78.24.33\}$$

Suppose the two entities are in the different domain, the edge between them is then. A complete example is shown in Fig. 3. The vertical structure of a collaboration is presented in different views. Process view displays the horizontal structure of the collaboration. Service entities from different (or same) trust domains interact with each other to form the business process. The deployment structure of the services is shown in the infrastructure view. It displays the locations of the deployed service nodes in different computing provisioning clouds – Amazon Web Service$^1$ and Windows Azure$^2$ in our example. By capturing both horizontal and vertical aspects of the collaboration, many specific details can be taken in to consideration when analyzing the compliance of participants. For instance, the location of the deployment may assist the diagnosis process in which the actual computing instance can be probed to test its availability; the logging amount can be reduced for interactions between entities belong to the same trust domain.

A. Procedural compliance

Procedural Compliance aims to verify if the procedures of the process has been correctly carried out, or more generally, if a service node has been invoked (e.g. activity sequence compliance). Suppose we have inner edge $N_a \rightarrow inner \ N_b$, the sub-process consists of node $a$ and node $b$ can then be described as

$$P : \{A_{in,t0} \ast ... \ast A_{N_a}^{N_b} \ast A_{out,t3}\} \rightarrow \{A_{in,t2} \ast ... \ast A_{N_b}^{N_b} \ast A_{out,t3}\}$$

(6)

Procedural compliance for node $a$ ($Comp_{procedure}^{Na}$) entails that: in a process $P$, $\exists V_{Na,Nb} \in P$, if $\exists A_{in,t0}^{Na}$ and

$^1$Amazon Web Service http://aws.amazon.com/

The image contains a page of text from a document, which appears to be discussing computer science concepts, particularly related to logging and compliance. The text is a continuation of a discussion on the logging costs and computational compliance, involving definitions and equations related to computational compliance and procedural compliance. The page contains mathematical expressions and logical statements related to this topic. The text discusses the conditions for logging and the relationship between the logging activities and the time frame required for compliance. It also mentions the estimation of the logging costs and the implications for QoS (Quality of Service) compliance. The text is technical in nature and requires a good understanding of computer science concepts to comprehend fully.
From the table, we can see that, apart from procedural compliance, all other three types of logging will incur large overhead which grows with the size of the data transferred between the nodes. Among those three, QoS compliance requires evidence to be logged immediately after the action, this stringency certainly brings further costs in real practice.

V. Compliance analysis

As aforementioned, even though the participating service providers are instructed to submit evidence to demonstrate their compliance, it is highly likely they may i) choose not to submit certain evidence or ii) submit bogus evidence to avoid possible penalties. In our design, we make very limited assumption on the logging and the truthfulness of the evidence submitted.

Before a service node has been concluded for being compliant for a specific action, the AS node constant hypothetically presume it is incompliant. The confidence of such hypothesis – incompliance hypothesis confidence \( Conf \in [0,1] \) is a continuous value, with 1 being definitely incompliant and 0 being definitely compliant. It approaches 1 as more and more evidence suggest incompliance (in some case, the fact that no evidence is submitted itself indicates violations). The AS\( ^\prime \)s node continuously updates \( Conf \) for all observed activities that are happening in the collaboration.

Evidence logged by a node can reflect its compliance, and sometimes a service node’s compliance can be inferred indirectly by evidence logged by other nodes. This section discusses the inference and reasoning the AS\( ^\prime \)s node conducts to verify the four types of compliance we have previously defined.

A. Procedural compliance analysis

To prove if node \( a \) has invoked the node \( b \), AS node needs to receive the evidence logged by node \( b \), this evidence unambiguously prove node \( b \) logged evidence (as it has been received) which further proves it has been invoked, that is

\[
E_{procedure}^{Nb} \Rightarrow A_{log,ty}^{Nb} \Rightarrow A_{in,t2}^{Nb} \Rightarrow Comp_{procedure}^{Na}
\]

Where ‘\( \Rightarrow \)’ symbol refers to material implication. Procedural compliance can also be inferred indirectly, any service node that will be invoked by node \( b \) and its succeeding nodes can log evidence to prove the process has been through node \( b \), that is, if \( \exists V_{Nb,Nc} \in P \) then

\[
E_{Nb} \Rightarrow A_{in}^{Nc} \Rightarrow A_{out}^{Nb} \Rightarrow A_{in}^{Nb} \Rightarrow Comp_{procedure}^{Na}
\]

The confidence for procedural incompliance of node \( a \) is then

\[
Conf_{procedure}^{Na} = 0 \begin{cases} \text{if } E_{procedure}^{Nb} \\ \text{if } \exists V_{Nb,Nc} \text{ and } \exists E_{Nb}^{Nc} \end{cases}
\]

and the confidence approaches 1 as the expected evidence from node \( b \) continues to be missing, in this case, it can be adjusted as

\[
Conf_{procedure}^{Na} = f_{conf} \{ t - \varepsilon (t_y) - \varepsilon (T_{Nb,AS}) \}
\]

where \( t \) is the current time and \( \varepsilon (t_y) \) is the expected time node \( b \) should log (the threshold time) which can be estimated according to the historical logging (i.e. the time node a logs plus the average transmission latency between \( a \) and \( b \)), and \( \varepsilon (T_{Nb,AS}) \) is the expected latency for the log to be received by AS\( ^\prime \)s node. \( f_{conf} \) is a function to increase the confidence according to the unusual delay that has occurred. Note that, in case even node \( a \)’s log is also missing, time obtained from nodes prior to node \( a \) can be used for the estimation.

B. Content compliance analysis

Content compliance can be proved when both \( E_{content}^{Na_{out}} \) and \( E_{content}^{Nb_{in}} \) have been received, they reflect the input and output of both service nodes,

\[
E_{content}^{Na_{out}}, E_{content}^{Nb_{in}} \Rightarrow A_{out,t1}^{Na}, A_{in,t2}^{Nb} \Rightarrow D_{out,t1}^{Na}, D_{in,t2}^{Nb}
\]

The compliance predicates on the consistency between input and output data, assuming \( \exists E_{content}^{Na_{out}} \cap \exists E_{content}^{Nb_{in}} \), the compliant and incompliant state can be expressed as

\[
Conf_{content}^{Na,Nb} = \begin{cases} 0 \text{ if } D_{out,t1}^{Na} = D_{in,t2}^{Nb} \\ 1 \text{ if } D_{out,t1}^{Na} \neq D_{in,t2}^{Nb} \end{cases}
\]

Unlike procedural compliance, content compliance cannot be inferred indirectly, it can only be verified by using the evidence from the sender and the receiver. When the evidence is missing, the confidence can be derived through

\[
Conf_{/content}^{Na,Nb} = \begin{cases} f_{conf} \{ t - \varepsilon (t_y) - \varepsilon (T_{Na,AS}) \} \text{ if } \exists E_{content}^{Na_{out}} \text{ or } \exists E_{content}^{Nb_{in}} \\ f_{conf} \{ t - \varepsilon (t_y) - \varepsilon (T_{Nb,AS}) \} \text{ if } \exists \neg E_{content}^{Na_{out}} \end{cases}
\]

C. Computational compliance analysis

As stated previously an approximation function is needed to simulate the computation of the service node to map the input / prior states to the output / post states. Sometimes the approximation function will be difficult to obtain in the real practice or may need domain experts to conduct the analysis. However, as long as the relevant evidence is recorded, this provides a means to investigate what actually happened in a post-facto manner even though costly.

The verification of computational compliance need to be carried out after the conclusion of the content compliance of the edge connecting to the node and the edge connecting this node to the succeeding node, i.e. \( V_{Na,Nb} \) and \( V_{Nb,Nc} \) for computational compliance on node \( b \) according to (7). This is to ensure the input and the output in evidence are the actual data incurred in the collaboration. Given the content compliance, the function of the service node can be inferred as

\[
E_{compute}^{Nb_{in}}, E_{compute}^{Nb_{out}} \Rightarrow D_{in,t2}^{Nb}, S_{t2}^{Nb_{out}}, D_{out,t3}^{Nb}, S_{t3}^{Nb} \Rightarrow F_{Nb}
\]

With content compliance, by assuming an approximate function, the compliant and incompliant state are expressed as

\[
Conf_{compute}^{Nb} =
\]

303
\[
\begin{align*}
0 & \text{ if } F^{N_b} \{ D^{N_b}_{in,12}, S^{N_b}_{12} \} = \{ D^{N_b}_{out,t_3}, S^{N_b}_{t_3} \} \\
1 & \text{ if } F^{N_b} \{ D^{N_b}_{in,12}, S^{N_b}_{12} \} \neq \{ D^{N_b}_{out,t_3}, S^{N_b}_{t_3} \}
\end{align*}
\]

When any needed evidence is missing, this can be reflected in the confidence of the two content compliance (input and output edge), in this case the confidence of computational incompliance equals to the one of two content incompliance confidence whichever is bigger:

\[
Conf_{\neg \text{compute}}^{N_b} = \max \left\{ Conf_{\neg \text{content}}^{N_a,N_b}, Conf_{\neg \text{content}}^{N_b,N_c} \right\}
\]

### D. QoS compliance

QoS compliance is more challenging to be verified compared to the other three, because it is determined by recording the time elapsed when a service node is conducting certain activity. As an external party, it is difficult for the AS node to find out the actual time elapsed from the evidence submitted by the logger, since the network congestion may arbitrarily affect the transmission latency, and this fact may be exploited by dishonest participators to hide incompliance.

Therefore, AS node needs to record the time when the evidence is received and use it to determine the credibility of the claimed activity time enclosed in the evidence. For example, if \( t_1 \) is provided in the evidence as the time the node a claims when activity happened, the evidence will reach at AS node at \( t_1 + T_{AS} \). QoS compliance requires the logger to submit evidence immediately after the occurrence of the activity, so the confidence of evidence \( E_{QoS,t} \) being bogus can be computed as

\[
Conf \left( \neg E_{QoS,t_1}^{N_a} \right) = f_{\text{conf}} \left( t_1 - (t_1 + T_{AS} - \varepsilon(T_{Na,AS})) \right)
\]

Where \( \neg E_{QoS,t_1}^{N_a} \) represents the logical state of evidence \( E_{QoS,t_1}^{N_a} \) being incorrect, \( \varepsilon(T_{Na,AS}) \) is the expected (average) transmission latency between node a and AS node.

Similarly, dishonest nodes can also exploit the possible transmission congestion at its input/output edge. For response time QoS for example, a dishonest node may log evidence a moment after it has started processing a job or log evidence before the output can be sent to the next node, in order to reduce its processing time observed by AS node. So apart from the confidence of the time in the submitted evidence, the confidence of the transmission latency at service’s input/output edge also needs to be analyzed. For simplicity, let us assume the confidence estimation equations are linear, the confidence of dishonest transmission time at the input edge of node b is

\[
Conf \left( \neg T_{t_1,t_2}^{Na,Nb} \right) = f_{\text{conf}} \left( t_2 - t_1 - \varepsilon(T_{Na,Nb}) \right) + Conf \left( \neg E_{QoS,t_2}^{Nb} \right)
\]

Where \( T_{t_1,t_2}^{Na,Nb} \) is the transmission latency between node a and node b with duration \( t_2 - t_1 \), and \( \varepsilon(T_{Na,Nb}) \) is the expected transmission latency between node a and node b. In this way, the confidence of QoS incompliance of node b can be computed as

\[
Conf^{Nb}_{QoS} = Conf \left( \neg T_{t_1,t_2}^{Na,Nb} \right) + Conf \left( \neg T_{t_3,t_4}^{Nb,Ne} \right)
\]

The response time of node b can be calculate by subtracting \( t_3 \) and \( t_2 \) (7) and this confidence reflects how reliable this result is, the higher the incompliance confidence, the less the this subtracting result can be trusted.

### VI. Evaluations

In order to visualize the capability of our approach in monitoring the status and compliance of underlying business process, in the past, we have implemented a compliance monitoring console, to show the information the AS node has collected and concluded. We now have extended the console according to the model we built. A screen-shot of the current implementation of the monitoring console is show in Fig. 4.

The console consists of four panels. The Document Panel displays the documentations registered by the services participating in the business process, such as BPEL, WSDL and WSLA. The top right panel displays the overview of the business process, animation is used to show the interactions between the underlying entities or service nodes so it can be seen that at which stage the process is being conducted. Two views are provided (which can be switched between), the process view shows the horizontal structure of the collaboration and displays the interactions at the entity level.

In the infrastructure view, the business process is represented in terms of service nodes deployed in different clouds. Those nodes are identified according to their IP address, should any incident occurs, the suspected computing instance will be located by its IP address for diagnosis. Note that, each service entity may have several service nodes deployed in the clouds. The infrastructure view can help to narrow down the fault domain to the instance level, that is, find out with instance owned by which entity is being incompliant.

The bottom left panel shows the performance statistics of the services being monitored, as well as the incompliance confidence about their behaviors. Both performance and confidence are continuously updated according to the evidence received. The statistics shows the numerical performance according to the evidence while the confidence exposes the credibility of such values and the fulfillment of other non-numerical compliance types. The bottom right 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 health of the process concluded by the AS.

Apart from the console, we have implemented an experimental business process in Amazon EC2 – a computing resource provisioning service that charges the user according to the CPU usage. Our implementation used Apache Tomcat 5.5 as our Servlet container, and Axis2 1.5 as our web service engine. We use BPEL to orchestrate the service nodes to form our loan application business process. Apache Orchestration

\[3\] Web Service Level Agreement http://www.research.ibm.com/wsla/
Director Engine (ODE)\(^4\) has been used to conduct the business process defined in BPEL.

We deployed 16 service nodes on 16 computing instances with computing power equivalent to 1GHz CPU and 1.7GB memory. With one being accountability service node, each of the five service entities in our example has 3 service nodes deployed in the cloud. The 15 nodes are orchestrated by BPEL scripts to simulate the business process in our running example. The service nodes belong to the same service entity form a simple sequential composition (i.e. sequentially invoked). Logging actions are inserted into the BPEL scripts as ‘invoke’ actions to the AS node. Details of this can be found in our previous work [20].

With the 15 service nodes, we have tested the overhead introduced when services log for different types of compliance. The results are shown in Fig. 5. We have tested the business process with request message size from 1KB (equivalent to a long sentence) to 50KB (equivalent to a medium size document). Procedural compliance is the only type which incurs minor overhead regardless of the message size. Other three types all introduce substantial extra latency (up to 60%) which grows as the request message becomes larger. However, this result is quite as expected, as those three types of compliance simply requires the same message to be re-transferred (more than ones in some case). One way to reduce this overhead is to log hash of the data instead of the data, this would require the loggers to archive the data until it is no longer needed for compliance verification. In this way, the overhead will be significantly reduced to close to the extra latency of procedural compliance logging. Nevertheless, the testing results suggest substantial cost when logging entire evidence for strong accountability, in the real practice, such stringent logging shall only be applied to critical actions.

VII. RELATED WORK

Compliance is an issue which has been extensively studied in recent years. Traditionally, monitoring techniques are applied for its verification. Approaches like [7][11] are heuristics to conveniently capture processing data at the service node during the execution. Some other approaches like [10][13][16] let service nodes emit evidence to a central authority to process them aggregateply. Focus of these approaches is usually on the ease of deployment and measuring accuracy. It is often assumed in these approaches that the collected evidence is not bogus and the incompliant entity will admit the violation. Our work, on the other hand considers a more hostile environment where all service entities are expected to behave in any possible manner and deceive for their own benefit. Approaches like [22][5] share this point with us that cryptographic techniques are employed to achieve provability.

There are many existing modeling of business process, popular ones like BPEL and XPDL\(^5\). They are designed solely to describe or define the action sequence of the process. Whereas our approach aims to capture the trust relationships and deployment details for compliance analysis and reasoning. In fact, it is common that approaches for compliance analysis will develop a model based on similar concept to BPEL and XPDL. Approaches like [1][4] model the process as a sequence of event traces emitted by the service nodes. Compliance is verified through matching the patterns of the event or mine the event traces [2]. Approaches like [8][3], model such

\(^4\)Apache ODE http://ode.apache.org/

\(^5\)XML Process Definition Language (XPDL) http://www.wfmc.org/xpdl.html
execution as a sequence of service state changes, with the assumption that states are all preserved, compliance is verified by examining the causality between the states. In a similar way, Petri-net [12] has been used to model the actions and the state changes in a process [6][14]. Our modeling method differs from those approaches in the way that we model the different domains and implementation infrastructures, and explicitly define the difference between the actions happened and the actions observed by the AS node.

Inference and reasoning involved in compliance assurance mostly focus on verifying the logical consistency and causality of the events. As in [18][9], correctness of certain action is proved by looking up previous actions to check if the actor has been properly authorized. In our approach, the AS node first reason about the credibility of the evidence, then analyze the extent to which the compliance can be proved by it.

VIII. CONCLUSIONS

In this paper, we proposed a novel modeling of the collaborative business process, as an extension of our previous work to enforce strong accountability for compliance. This model captures both the horizontal structure (process level) and the vertical structure (infrastructure level) of the collaboration. With this model, we classified and defined four generic types of compliance, and we thoroughly analyze the evidence and the logging required for their verification. We have implemented a monitoring console to demonstrate the utilization of our modeling. We evaluate the costs of the logging in an experimental business process deployed in Amazon EC2, the overhead observed is substantial yet still acceptable. Different logging schemes shall be widened applied to meet the specific compliance stringency required.

IX. REFERENCES


