Appeared in the Proceedings of the Fourth International Conference on Availability, Reliability and Security (ARES '09)

Web Service Trust:

Towards A Dynamic Assessment Framework

George Spanoudakis
Department of Computing
City University London
London, UK
e-mail: G.Spanoudakis@soi.city.ac.uk

Stephane LoPresti
School of Engineering and Design
Brunnel University
London, UK
e-mail: Stephane.Lo-Presti@brunel.ac.uk

Abstract— **Trust in software services is a key prerequisite for the success and wide adoption of Services-Oriented Computing (SOC) in an open Internet world. However, trust is poorly assessed by existing methods and technologies, especially in dynamically composed and deployed SOC systems. In this paper, we discuss current methods for assessing trust in Service-Oriented Computing and identify gaps of current platforms, in particular with regards to runtime trust assessment. To address these gaps, we propose a model of runtime trust assessment of software services and introduce a framework for realizing the model. A key characteristic of our approach is the support that it offers for customizable assessment of trust based on evidence collected during the operation of software services and its ability to combine this evidence with subjective assessments coming from service clients.**

Keywords: trust, runtime assessment, web services

I. INTRODUCTION

To become pervasive and open up to a wide range of applications and the general public, Service-Oriented Computing (SOC) needs to foster dynamic and trustworthy service interactions. The current state of the art in SOC systems provides complex and flexible means for static and dynamic service discovery and composition, and tailoring services to the consumer. However, current SOC technology provides less adequate support for assessing the trustworthiness of services, especially in situations where the latter need be dynamically linked and deployed by SOC systems.

As SOC aims to create business and commercial communities through the use of distributed services, service trust comes naturally into the equation as a prerequisite of the wider acceptance of the SOC paradigm and its development to its full potential. This is because service clients need to be able to trust the services that they want to use remotely and ascertain dynamically that the services they already use remain trustworthy. Also, without building trust assessment within SOC systems, services of newcomers won't have an opportunity to get deployed in the presence of service providers with well-established brands. Furthermore, it should be appreciated that services cannot be delivered cost-effectively without trust. This because the lack of explicit social or context relationships in the establishment of service agreements and the absence of trust may necessitate complex and costly legal agreements, which would not be easy to achieve in dynamic service deployment contexts.

Despite the recognition of the importance and necessity of trust in human interactions and exchanges and, as a consequence, the recent increase of the volume of the literature on this topic (e.g. [5], [9], [11], [14], [17]), the role of trust in SOC has not been fully realised yet and trust is currently assessed poorly for the purposes of the new paradigm. This is because, most of the existing work addresses service trust through methods and techniques for securing service interactions, including standards for representing security credentials such as WS-Trust [2]. There have also been platforms for reasoning about trust and security properties in particular service deployment architectures (e.g. grid [1], [23]). However, existing work fails to address some important aspects of service trust, notably the need to assess trust in dynamically composed and deployed services, ground it not only on subjective opinions but also on dynamically acquired information about the behaviour and quality of services in diverse deployment contexts, and the ability to evaluate the accuracy and risk of trust assessments [20].

To address these shortcomings, in this paper we propose a model of dynamic trust assessment of web services and an architecture for realising this model. The proposed trust assessment model combines objective information acquired during the deployment of services in different contexts at runtime with subjective recommendations from service clients and uses this information to provide trust assessments for different service provision promises expressed within service level agreements (SLAs). The realisation of this approach is based on a platform that incorporates runtime service monitoring as a key element and offers trust assessment services to service clients and providers.

The rest of the paper is structured as follows. In Section 2, we present a scenario that forms the basis for explaining our approach. In Section 3, we present the basic conceptual model of trust assessment and the architecture of the platform that realises the model. In Section 4, we discuss the generation of trust assessment in more detail. In Section 5, we give an account of related work and, finally, in Section 6 we provide some concluding remarks and present directions for future work.

II. A SCENARIO OF TRUST ASSESSMENT IN SERVICE BASED SYSTEMS

To indicate the requirements for dynamic trust assessment and demonstrate the basic elements of our approach in the rest of the paper, we will use a scenario of a service based e-healthcare system that has been used in the EU-IST Framework 6 project SERENITY[1]. The system that underpins our scenario is shown in Figure 1. In this system, patients are equipped with a *Health Monitoring Equipment (HME)* which monitors their medical conditions and notifies a *Health Treatment Service (HTS)* of health signals that should be examined by a doctor. When HTS receives a notification from HME, it searches through a registry of collaborating doctors that it maintains, and if it finds a doctor who could deal with the reported symptoms, it contacts him/her with a request to handle the case. To receive requests from HTS, doctors are equipped with PDAs running a specific web service that enables the processing of HTS requests.

A doctor can decline or accept a request from HTS. In the latter case, (s)he subsequently examines the data and makes a medical recommendation for the patient to HTS. This recommendation may be of different types including the provision of a medical prescription, a recommendation for admission to hospital, or a recommendation for no further action. The doctors collaborating with HTS may offer their services under different *Service Level Agreements* (SLAs). A doctor may, for instance, have agreed to offer recommendation services only within specific working hours or only for patients which do not suffer from particular medical conditions.

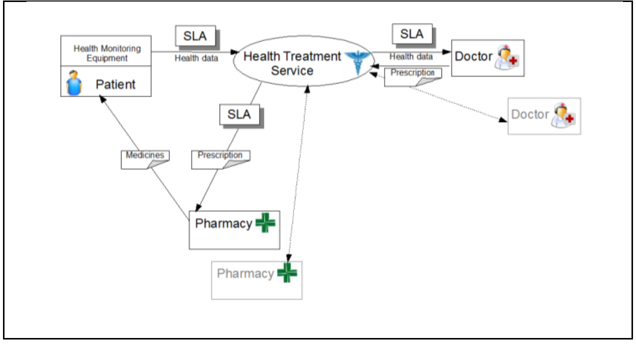


Figure 1. A Dynamic Web Services e-Healthcare Scenario

In cases where a doctor responds with a medical prescription, HTS contacts a pharmacy to order and dispatch the set of prescribed medicines to the patient. To achieve this, HTS calls pharmacies which can accept medical prescription orders through a pharmacy web service and physically deliver the prescribed medicines to the patients. HTS maintains a list of pharmacy services covering different locations. Pharmacy services are available under different SLAs. Some of them, for example, may promise to deliver ordered medications to certain postcodes within a maximum time limit or accept orders only if the cost of the service is to be covered by HTS. Other pharmacies, however, may also accept orders whose cost is to be covered by the medical insurance of the patient.

Trust is fundamental to the deployment of doctor and pharmacy services in our scenario and may be affected by different factors and assessed in different ways. In the case of doctor services, for example, trust may be affected by the real availability of doctors within the service hours that their contract/SLA determines. For any such factors which are important for the trust that HTS casts upon specific doctor services, up-to-date assessments of these services with respect to the relevant factors will be necessary for the continuation of the deployment of the services.

In the case of pharmacy services, trust may depend on service availability and/or the ability of a pharmacy service to deliver correctly ordered prescriptions within the promised time. Furthermore, it should be noted that since initially the HTS will not have a sufficient history of interactions with a particular pharmacy service to assess its delivery efficiency, it would beneficial for it to obtain an assessment of this factor based on interactions of the particular service with other systems.

Current SOC technologies enable the implementation of the above scenario, but are unable to tackle the trust assessment issues related to it. More specifically, current trust assessment models either neglect runtime evidence of service performance against designated trust criteria or, when they take such evidence into account, the evidence is typically limited to interactions between the client needing the trust assessment and the specific service rather than taking into account a broader picture of actual service interactions different clients and contexts of deployment.

III. OUR APPROACH

A. Trust assessment model

Our approach to the trust assessment of web services is based on the conceptual model shown in Figure 2.

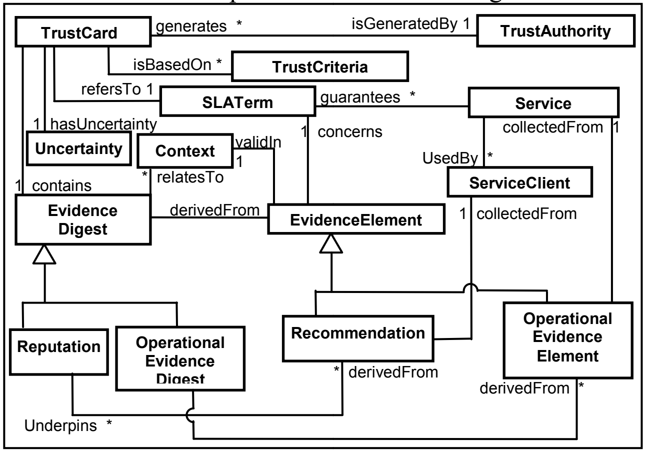


Figure 2. Conceptual model of a trust assessment

According to this model, trust assessment is performed by some *TrustAuthority*. This authority is independent from the provider of a particular service and its clients. The role of the trust authority may be undertaken by a service registry in cases where the registry has an interest in assessing the trustworthiness of the services that it lists or some other party.

The basic outcome of the trust assessment activity is a *trustcard*. A trustcard is generated by the trust authority

[1] SERENITY focuses on runtime support for security and dependability – see http://www.serenity-project.org

(as shown by the association *TrustCard–isGeneratedBy→TrustAuthority*[2] in Figure 2) to represent an assessment of trust that the authority has in a particular term of the SLA under which a service is offered at a given instance of time and in a particular context. Hence, a trustcard always refers to an SLA term (see the association *Trustcard–relatesTo→SLATerm* in Figure 2), expressing some Quality-of-Service (QoS) factor (e.g. availability of service as a whole, performance of particular service operations) or behavioural service property. The association of trustcards with particular SLA terms reflects the need for providing fine grain trust assessments with respect to specific properties. Thus, in our approach, trust cannot refer to service providers in general without reference to a particular service that they provide. This is because providers may be trusted for some of the services they provide but not other. A bank, for example, may be trusted for its basic retail banking services but not its investment services.

To substantiate the trust assessment that it represents, a trustcard includes *evidence digest (ED)* elements (see the association *TrustCard–Contains→EvidenceDigest* in Figure 2). An evidence digest element is derived from processing several different trust related evidence elements for a service, as expressed by the association *EvidenceDigest –derivedFrom→EvidenceElement* in Figure 2. An *evidence element (EE)* is a signed and verified package of data which has been collected either during the operation of the service or from a particular service client and is relevant to a trust assessment. An evidence element related to the availability of a service operation is, for example, the response time for a particular call of the service operation.

Evidence elements can be of two different types, namely *Operational Evidence Elements (OEE)* and *Recommendations*. The elements of the former type (i.e., operational evidence elements) are collected during the operation of a service at runtime and, therefore, provide objective operational evidence about the service. Recommendations, on the other hand, are provided by service clients and, thus, constitute subjective evidence.

Both recommendations and operational evidence elements are related to a particular *context* of service deployment and are valid only within this context, as expressed by the association *EvidenceElement –validIn→Context* in Figure 2. The response time recorded for a particular service invocation may have, for example, as context the service level agreement under which the service made the particular response, the client to which the response was made and its characteristics (e.g., its location, whether it's a business or private client), the time when the response was made (peek vs. off-peak service hours), and the location of the service itself (if the service is mobile).

An evidence digest element presents an aggregated account of a set individual evidence elements which underpin a trust assessment. The average prescription delivery time of a pharmacy service in the scenario of Section 2, for example, would be an evidence digest for the individual delivery times collected for the particular service. Similarly the total number of cases where a request for a service to a particular doctor has been declined would be an evidence digest for all the individual cases spotted with respect to this trust criterion. Evidence digests which are derived from recommendation elements following some reputation algorithm (e.g., simple summation, average, Bayesian [9]) constitute a *Reputation* in our trust model.

Like evidence elements, evidence digests are also relative to a context that enables their interpretation (see association *EvidenceDigest–relatesTo→Context* in Figure 2). The context of a digest element in a trustcard refers to characteristics of the deployment of the service including characteristics of the operational environment of the service (e.g. type of used infrastructure, network conditions, applicable security mechanisms and policies), the periods of service deployment, the types of service clients etc.

Trustcards are given to service clients by the trust authority upon request. When they receive a trustcard, service clients may perform their own trust judgment by considering the information in the trustcard with regards to the specific context in which it was requested and other trust requirements (e.g., their acceptable level of risk, interests, and preferences). It should also be noted that trustcards are associated with an *uncertainty* measure. This measure reflects the uncertainty which may exist in a trust assessment if there are conflicting and/or limited evidence elements for the trust assessment criteria. Finally, trustcards get the status of *trust certificates* when the trust authority which has produced them assumes also legal responsibility for them.

B. Trust assessment platform

The realisation of trust assessments and provision of trustcards at runtime is supported by a platform whose general architecture is shown in Figure 3. This platform acts as a server that can provide trustcards to service clients requesting them for the services that they use, and trust assessment services to service providers wishing to subscribe to the platform in order to get their services assessed and thereby expand their client base. Trust assessment services may also be requested by third parties for specific services if these parties wish to act as brokers for the services. This role can be typically assumed by service registries that are willing to provide information for and access to services, only if the latter agree to become the subject of a continuous trust assessment during their deployment[3]. Service clients can enquire the platform for trustcards that cover specific trust assessment criteria and service providers can subscribe to the platform to initiate assessments of their services.

As shown in Figure 3. the trust assessment platform incorporates six components, namely a *TrustAssessmentManager*, an *EventBus*, *Monitors*, *EventCaptorGenerators*, *EventCaptors,* and

[2] In the paper, we use the term *ClassA–associationEndName→ClassB* to refer to an association between *ClassA* and *ClassB* in the conceptual trust model that has an association end called *associationEndName* attached to *ClassB.*

[3] Currently, we assume that the trust assessment platform operates separately from a service registry. This, however, needs not to be the case and the trust assessment platform can be incorporated within a broader service registry infrastructure.

TrustCriteriaGenerators (the components of the platform appear in grey boxes in Figure 3).

The *TrustAssessmentManager* provides the interface for accessing the platform. This interface (i.e., the *TrustAssessmentInterface* in Figure 3) offers operations for: (i) requesting trustcards for particular properties of particular services, (ii) requesting the initialisation of trust assessments of services based on specific properties, and (iii) providing information for activating the emission of events from services to the platform. The trust assessment manager has also responsibility for maintaining evidence elements and evidence digests that underpin the dynamic creation of trustcards, and managing trust assessment subscriptions by notifying all subscribers of updates in the trustcards that they have subscribed for.

The trust assessment manager is built on the top of *monitors*. Monitors collect the evidence elements required for trust assessment, and aggregate them into evidence digests. The evidence elements collected by monitors correspond to web-service events captured at runtime (e.g. service invocations and responses) and invocations of specific operations of the platform by service clients to provide subjective information about web-services.

In the current prototype of the platform, the monitors are implemented as web-services based on the monitoring framework described in [12][13]. This framework uses monitoring rules and assumptions which determine what information should be collected from different services, what conditions this information should satisfy to be usable, and how it could be combined to generate evidence digests that can substantiate assessments with respect to specific trust criteria (see Section 4 for more details). Monitors provide a *MonitoringInterface* (see Figure 3) that is used by the *TrustAssessmentManager* to provide the rules that need to be monitored in order to obtain the evidence elements and digests required for trust assessments.

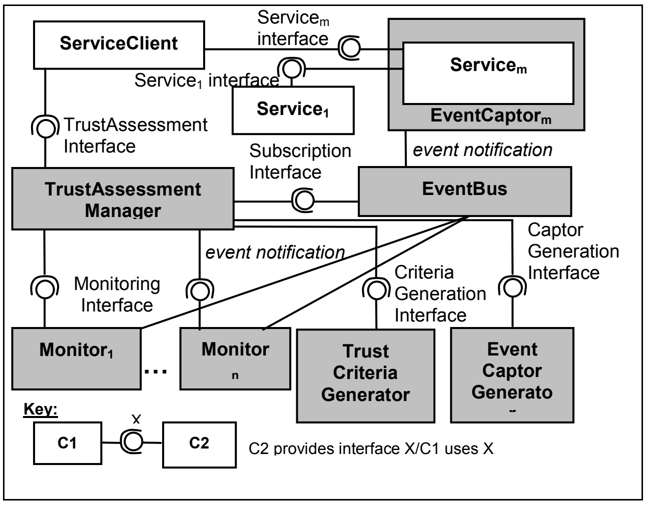


Figure 3. Architecture of the trust assessment platform

The *TrustCriteriaGenerator* is the component of the platform that generates the monitoring rules and assumptions which are used for carrying runtime trust assessments. The generation of these rules and assumptions is driven by the guaranteed terms in the SLA of a service that needs to be assessed by the trust assessment platform. These terms are provided to the *TrustCriteriaGenerator* by the *TrustAssessmentManager* through the *CriteriaGeneratorInterface* of the former component. The design of the platform allows the incorporation of different types of trust criteria generators in order to support the generation of trust assessment specifications from different languages for specifying SLAs. An example of generating monitoring rules from service guarantee terms expressed in WS-Agreement (as described in [12]) is given in Section 4.

The trust assessment platform also incorporates different types of *event captors*. Event captors operate remotely from the platform and have responsibility for intercepting messages which are exchanged between web services and their clients, generating events to represent these messages and sending the events to the platform for analysis. Event captors are service proxies which are generated automatically by the platform based on the WSDL specification of a service and the criteria set for its trust assessment.

The generation of event captors is the responsibility of the *EventCaptorGenerators* in the platform. By virtue of their automatic generation, event captors are expected to execute correctly in all circumstances and be able to collect the required evidence elements from a service at runtime, following their installation in the operational environment of the service.

The notification of events from event captors to the monitors takes place through the *EventBus*. This component operates as a publish/subscribe notifier that forwards events to the monitors according to subscriptions of the latter for specific types of events of particular services. These subscriptions are generated by the *TrustAssessmentManager* using the *SubscriptionInterface* of the *EventBus*.

Although, a detailed account of the way in which our platform is secured is beyond the scope of this paper, we should note that basic security issue is addressed by the use of a public key infrastructure (PKI). This means that the links between the platform and the event captors use secure channels (typically SSL/TLS tunnels), which provide communication confidentiality. Also all the data which are exchanged within the platform are integrity-protected, using typically a keyed-Hash Message Authentication Code (HMAC), and possibly bound to particular software configurations if Proof-Carrying Code (PCC) or Trusted Computing are used. In that case, cryptographic keys can be hardware-protected in the Trusted Platform Module and secure channels are reinforced via mutual attestation of software configurations on both sides of the communication (a detailed account of the use of Trusted Computing for this purpose is given in [10]).

In connection with basic security, we should also note that the design of the trust assessment platform assumes that services and service clients are identified by their public keys with the use of Certification Authorities (CAs) that are accepted as valid by the trust authority that operates the platform, so that information can be encrypted to them and only read by the intended recipient. Reciprocally, the various actors ensure mutual authentication by encrypting trustcard requests or evidence element exchange with the public key of the platform.

IV. Generation of Trustcards

In the following, we discuss how to specify and operationalise trust assessments using the trust assessment platform. Initially, we first provide an overview of the specification of trust assessment criteria and then we give an example of deriving these criteria from an SLA and using them to generate trustcards.

A. Specification of trust assessment criteria

The criteria for the generation of trustcards are specified in an Event Calculus based language which is supported by the monitor of the trust assessment framework, called *EC-Assertion*. A detailed account of this language is beyond the scope of this paper and may be found in [12][13]. In the following, however, we give a brief overview of it to enable the reader understand the example of trust assessment that we present later.

EC-Assertion is a first-order temporal logic language based on Event Calculus [18] which supports the specification of monitoring conditions in terms of *events* and *fluents*.

An event in EC-Assertion is an occurrence that takes place at a specific instance of time (e.g., invocation of a system operation, receipt or dispatch of a message) and may have an effect. The occurrence of an event in EC-Assertion is represented by the predicate *Happens(e,t,*$\Re$*(lb,ub))*. This predicate denotes that an instantaneous event *e* occurs at some time *t* within the time range $\Re$*(lb,ub)* (i.e., lb ≤ t ≤ ub). Events represent invocations of system operations, responses from such operations, or exchanges of messages between different system components and are expressed as terms of the following form: *e(_id, _sender, _receiver, _status, _sig, _source)*. In this term: (i) *_id* is a unique identifier of the event, (ii) *_sender* is the identifier of the agent (system component or external actor) that sends the message represented by the event, (iii) *_receiver* is the identifier of the system component that receives the message represented by the event, (iii) *_status* represents whether the event is a request (REQ) or a response to a request (RES), (iv) *_sig* is the signature of the dispatched message or the operation invocation/response represented by the event, comprising the operation name and its arguments/result, and (v) *_source* is the identifier of the component where the event was captured.

Fluents in EC-Assertion are conditions which may change over time (e.g. a fluent may indicate that a system has received a message) and are initiated and/or terminated by events. Fluents are represented by relations between objects of the form $rel(O_1, ..., O_n)$ where *rel* is the name of a relation which associates the *n* objects O_1, …, and O_n. The initiation or termination of a fluent *f* due to the occurrence of an event *e* at time *t* is denoted in EC-Assertion by the predicates *Initiates(e,f,t)* and *Terminates(e,f,t)*, respectively. An EC-Assertion formula may also use the predicates *Initially(f)* and *HoldsAt(f,t)* to denote that a fluent *f* holds at the start of the execution of a system and at time *t,* respectively. A formula in EC-Assertion is specified in terms of the above predicates and has the general form *body* $\Rightarrow$ *head*. Formulas can be of two different types: monitoring rules or assumptions. Monitoring rules are formulas which are checked against runtime events to establish if they are satisfied and their meaning is that if the *body* evaluates to True, the *head* must also evaluate to *True*. Assumptions, on the other hand, are formulas which are used to generate information. If the *body* of an assumption evaluates to *True* at runtime, its *head* is assumed to be *True* by deduction.

EC-Assertion is used to express operational specifications of trust assessment criteria in the trust assessment platform. These specifications are generated from SLAs by the trust criteria generators of the platform. An example of this generation is given in the following section but prior to this it is important to provide some justification of the choice of EC-Assertion as the language for the specification of trust assessment criteria in our approach.

This choice has been motivated by the fact that EC-Assertion advocates a generic but simple modeling ontology based on events and fluents, and supports an explicit representation of the time of occurrence of these events and fluents and the specification of explicit constraints regarding this time. Due to these characteristics, EC-Assertion enables the specification of a wide spectrum of behavioural and QoS properties for software services. These properties may be atomic or aggregate, i.e., they may refer to specific interactions between a service and its environment (e.g. the response time of a service for specific operation invocation calls) or sets of such interactions (e.g. the average response time of all the invocations of a particular service operation). Finally, due to its foundation upon Event Calculus, EC-Assertion has a formal and well understood semantics which is important for the specification of formal and precise trust assessment properties.

B. Example

Our example of trust assessment criteria is based on the e-healthcare system we introduced in Section 2 and focuses on the assessment of trust for pharmacy services. Enquiries for trustcards for such services may be raised by HTS in different circumstances, including for example cases where the pharmacy service which is currently used for a specific area is not responding and a runtime search for a replacement has identified services that HTS has not used before and, therefore, has no trust in. A trustcard may also be requested in cases where HTS wants to reassure itself about the trustworthiness of a pharmacy service that it has used before at runtime. HTS can request a trustcard when the need for it arises (pull mode) and/or subscribe to the trust platform for regular trustcard updates for a given service (push mode). Trustcards can be requested for specific service properties that have been specified in the SLA published by the service.

In our example, suppose that the SLA of the pharmacy service includes a term indicating that for patient addresses which are within a given set of postcodes the delivery time is guaranteed to be at most 60 minutes after the acceptance of an order. Suppose also that the pharmacy service offers the following operations:

(i) An operation *confirm_order(patient_address, orderRefNum)* which a client can call in the service to confirm an order following an earlier quotation for a requested prescription by the pharmacy service. This operation takes as input the reference number of the order that is confirmed (*orderRefNum*) and the delivery address of the patient.

(ii) An operation *delivered(orderRefNum)* which is called by the agents who deliver prescriptions to patients to confirm that the prescription related to *OrderRefNum* has been delivered to the patient.

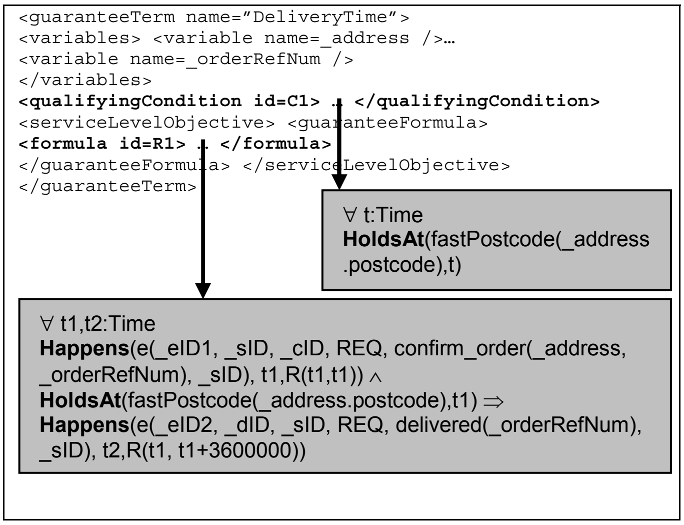
Given the above operations, the SLA guarantee term about the maximum prescription delivery time can be specified as shown in Figure 4. This specification is based on the extended schema for *WS-Agreement* that has been introduced in [12]. In WS-Agreement, the specification of an SLA guarantee term consists of a *qualifying condition* specifying a precondition that should be satisfied for the enforcement of a service guarantee term, and a *service level objective* that specifies the conditions that must be met in order to satisfy the service guarantee term. In the extended schema for *WS-Agreement* introduced in [12], both qualifying conditions and service level objectives are specified in EC-Assertion.

The EC-Assertion formula shown in Figure 4 [4] specifies a service level objective expressing the maximum prescription delivery time that is promised by the pharmacy service. The formula states that following a call to the pharmacy service *_sID* to confirm an order request by a service client *_cID* (see the event *e(_eID1, _sID, _cID, RES, confirm_order(_address, _orderRefNum), _sID)* in the formula), the deliverer of the prescription (*_dID*) will also call the pharmacy service to confirm the delivery of the order within 3600000 milliseconds (see the event *e(_eID2, _dID, _sID, REQ, delivered(_orderRefNum), _sID)* in the formula).

A request for a trustcard can then refer to exactly this item in the SLA of the pharmacy service. Upon receiving the request, the trust assessment platform checks whether it can provide a trustcard matching the request and, if it can, it generates the trustcard for the particular property of the service and sends it to the requester. The availability of appropriate trustcards depends on: (a) whether the trust assessment authority has initialised a trust assessment activity for the SLA of the particular service before it receives the request for the trustcard, and (b) if by the time that it receives the request it has a sufficient number of evidence elements to generate a trustcard.

The trust assessment platform can generate automatically the trust assessment criteria for the SLA guarantee term shown in Figure 4 in order to capture the primitive evidence elements and produce the evidence digests required for the trust assessment of the particular SLA term of the pharmacy service. For SLAs expressed in the extended form of WS-Agreement, as in Figure 4. , the generation of trust assessment criteria is based on the following transformation pattern:

```
(GTA1): <C> ∧ <B> ∧ <H> ∧
        HoldsAt(Satisfied(<GT>,<S>,_SGT), t) ⇒
        Initiates(<H>, Satisfied(<GT>,<S>,_SGT+1)
        max(tVar(<B>⇒<H>))+1)
```



```
<guaranteeTerm name="DeliveryTime">
<variables> <variable name=_address />…
<variable name=_orderRefNum />
</variables>
<qualifyingCondition id=C1> … </qualifyingCondition>
<serviceLevelObjective> <guaranteeFormula>
<formula id=R1> … </formula>
</guaranteeFormula> </serviceLevelObjective>
</guaranteeTerm>
```

∀ t:Time
HoldsAt(fastPostcode(_address.postcode),t)

∀ t1,t2:Time
Happens(e(_eID1, _sID, _cID, REQ, confirm_order(_address, _orderRefNum), _sID), t1,R(t1,t1)) ∧
HoldsAt(fastPostcode(_address.postcode),t1) ⇒
Happens(e(_eID2, _dID, _sID, REQ, delivered(_orderRefNum), _sID), t2,R(t1, t1+3600000))

Figure 4. An example of a pharmacy service SLA

```
(GTA2)  <C> ∧ <B> ∧ ¬<H> ∧
        HoldsAt(Violated(<GT>,<S>,_VGT), t) ⇒
        Initiates(<B>, Violated(<GT>,<S>,_VGT+1),
        max(tVar(<B> ⇒ <H>))+1)
```

In this pattern,

- *<S>* is a placeholder for the identifier of the relevant service and *<GT>* is a placeholder for the identifier of the service guarantee term
- *<B>* and *<H>* are placeholders for the body and the head of the EC-Assertion formula *B* ⇒ *H* that defines the service level objective of the term
- *<C>* is placeholder for the context of the service guarantee term (i.e., the conjunction of the qualifying conditions under which the service guarantee term must be satisfied in WS-Agreement).
- *max(tVar(<B> ⇒ <H>))* is the time variable in the formula *B* ⇒ *H* that can take the maximum possible value (i.e., the time variable of the event that is expected to occur after all other events in the formula)

The application of the above transformation pattern generates two assumption formulas GT_{A1} and GT_{A2} for each service guarantee term GT. The first formula (GT_{A1}) maintains a satisfaction counter of the guarantee term and increases its value in cases where when the context conditions of the term (i.e., <C>) are satisfied, the EC formula that defines the term (i.e., the EC-Assertion formula *<B>* ⇒ *<H>*) is also satisfied The second formula (GT_{A2}) maintains a counter of cases where even though the context conditions of the service guarantee term are satisfied, the EC-Assertion formula that defines the term is violated (i.e., *<C>* ∧ *<B>* ∧ ¬*<H>* is True). The two counters used in such cases are represented by the fluents *Satisfied(<GT>,<S>,_SGT)* and *Violated(<GT>,<S>,_VGT)*, respectively.

Based on the above pattern the following two assumptions will be generated for the formula *R1* in Figure 4. :

```
R1A1: ∀ t1,t2:Time
Happens(e(_eID1, _sID, _cID, REQ,
confirm_order(_address,_orderRefNum), _sID),
t1,R(t1,t1)) ∧
HoldsAt(fastPostcode(_address.postcode),t1) ∧
Happens(e(_eID2, _dID, _sID, REQ,
delivered(_orderRefNum), _sID), t2,R(t1,
t1+3600000)) ∧
```

[4] The figure shows the formula in a high level non XML syntax of EC-Assertion. Underscored names in the formula denote non time variables and all the non explicitly qualified variables are assumed to be universally qualified.

```
HoldsAt(Satisfied(R1,_sID,_SR1), _t2) ⇒
Initiates(e(_eID2, _dID, _sID, REQ,
delivered(_orderRefNum), _sID),
Satisfied(R1,_sID,oc:self:add(_SR1,1)),t2+1)

R1A2: ∀ t1,t2:Time
Happens(e(_eID1, _sID, _cID, REQ,
confirm_order(_address, _orderRefNum), _sID),
t1, R(t1,t1)) ∧
HoldsAt(fastPostcode(_address.postcode),t1)  ∧
¬ Happens(e(_eID2, _dID, _sID, REQ,
delivered(_orderRefNum), _sID), t2,
R(t1, t1+3600000)) ∧
HoldsAt(Violated(_sID,_VR1) _t2) ⇒
Initiates(e(_eID1, _sID, _cID, REQ,
confirm_order(_address,_orderRefNum),_sID),
Violated(R1, _sID, oc:self:add(_VR1,1)),t2+1)
```

The above formulas are generated by instantiating the transformation pattern using the following substitution of terms in the service guarantee formula R1 for the placeholders of the pattern:

- <C>: HoldsAt(fastPostcode(_address.postcode), t)
- <B>: Happens(e(_eID1, _sID, _cID, REQ, confirm_order(_address, _orderRefNum), _sID), t1,R(t1,t1)) ∧ HoldsAt(fastPostcode(_address.postcode), t1)
- <H>: Happens(e(_eID2, _dID, _sID, REQ, delivered(_orderRefNum), _sID), t2,R(t1, t1+3600000))
- <GT>: R1
- <S>: _sID

Following the generation of the above assumptions, the trust assessment platform assigns them to the monitor. When it receives them the monitor starts checking the *confirm_order* and *delivered* events and uses the formulas $R1_{A2}$ and $R1_{A2}$ to deduce the values of the satisfaction and violation counters for R1. The formulas $R1_{A2}$ and $R1_{A2}$ also determine the evidence elements and digests that need to be maintained for the trust assessment of the particular SLA term. In particular, at any given time point *t*,

- the evidence elements include the instances of $R1_{A2}$ and $R1_{A2}$ that have been generated by the monitor following unification with runtime events up to time *t* in order to update the values of the guarantee term satisfaction and violation counters, and
- the evidence digests are the values of the counters _SR1 and _VR1 at *t*.

This information is maintained by the platform and used to generate a trustcard when requested. The generated trustcards are associated with an uncertainty measure about the trustworthiness of a service with respect to the particular service guarantee term expressed by the card. This measure is calculated as the number of the violations of the service guarantee term over the total number of the cases of its assessment (i.e., the ratio *_VR1/(_VR1 + _SR1)* in our example).

It should be noted that the generation of a trustcard may depend on additional criteria that the trust authority may have for the particular case. In our example, these criteria may include the minimum number of evidence elements (i.e., instances of $R1_{A2}$ and $R1_{A2}$) that the authority must have before generating a trustcard. More complex criteria may also be used by the trust authority. Such a criterion could be to have evidence elements from interactions of the service with at least K different clients before issuing a trustcard.

V. Related work

Trust is a complex concept studied from multiple perspectives [22] including the prominent perspectives of system security and the socio-economic modelling of trust. In the following we provide an overview of work in these two areas, noting that the gap between them is still significant.

In the security domain, different strands of research have addressed issues of authentication, authorisation, confidentiality and availability of SOCs and have generated standards such as WS-Trust and WS-Policy for representing security credentials and policies regarding these credentials [4]. Trust reasoning and management platforms are also being developed in this context for Grid applications [1][23]. Trust management proposed more complex credential policies and standard algorithms for checking policy compliance, such as in KeyNote [3] and SULTAN [8], but these solutions suffer from the lack of efficiency and expressiveness in practice. Trusted Computing [11] is a recent security paradigm that proposes to semi-formally implement a *chain of trust* providing the ability to represent reliably and communicate securely software configuration states along execution chains (e.g., BIOS, boot loader, virtualisation kernel, OS kernel). This is based on hashes of application binaries and configuration files, called *measurements*, which are signed using hardware-protected cryptographic keys. These measurements do not provide any information about properties of the application and may lack information about its operational environment. A fundamental issue in the security domain is the fact that trust is only considered as the trust that a security user, or security service consumer, has in a remote security provider following a chain of cryptographic tools.

On the other hand, work in the socio-economic area of trust has proposed a plethora of models and systems which tackle the concept of trust from a wider perspective. Trust factors, such as credibility, ease of use or risk, have been elicited in e-commerce [5] offering design and (web) interface elements and models of trust that can be used as starting points for HCI developers. Design methodologies of pervasive systems have been augmented with trust requirements [10] such as audit trail (or accountability), harm, reliability and accuracy. Other work has focused on topics such as website credibility [7] and reputation [9], [17]. McKnight and Chervany [14] give a wide account of research on trust in the domains of management, sociology, economics, politics, science and psychology.

Most modern approaches to trust consider static information and fixed scenarios. The runtime assessment of trust includes the dynamic verification of whether trust rules are consistent with the behaviour of a system (multiagent systems [16], open and distributed systems [15]) and the monitoring of trust specifications (e.g. adaptive systems trust monitoring [19]). There is also some work on monitoring the dynamic operation of partners in virtual organizations (VOs) in order to identify their failure to meet obligations established during the formation of VOs [6].

The approach that we take in this paper differs from the above work in two respects: it takes a comprehensive view over trust assessment and uses runtime data to generate and update trustcards expressing trust in specific terms of SLAs between service providers and consumers. Security credentials and policies can be modelled in our framework [21] but the service client is not limited to these and can use other properties such as reliability and efficiency. Subjective information can also be integrated into the approach via the use of service client recommendations. Furthermore, our approach addresses dynamic service use scenarios and captures the composite nature of trust in the concept of trustcards. Finally, the process of property assessment is automatic and can derive trustcards on the fly, while service clients are updated when changes occur.

VI. Conclusion and Future Work

In this paper, we have discussed the issue of assessing trust in SOC systems based on runtime information and highlighted the limitations of existing approaches in this respect. To address these limitations, we have proposed a model for dynamic assessment of web service trust and a platform that realises the model. The assessment of trust in this model is based on trustcards. Trustcards provide assessments of the trustworthiness of web services with respect to different criteria. These assessments are based on information collected by monitoring web services in different operational contexts and subjective assessments of trust provided by different service clients and situated in specific operational contexts. Trustcards enable service clients to make better informed and dynamic decisions about the services they deploy and adapt dynamically to changes that may affect the trustworthiness of services.

The platform that we have proposed to realise the model is based on a service monitoring framework developed at City University [12][13] and extends it with capabilities that enable the dynamic generation, provision and maintenance of trustcards. More specifically, the underlying monitoring framework provides the implementation of the monitors in the platform and has been extended by components as discussed in Section III.

The proposed platform has been partially implemented and some of its components, notably the ones supporting the automatic generation of event captors, are still under development. Current work looks also at extensions of the platform to make it usable by coalitions of users wanting to form trust assessment collectively rather than centralised trust authorities as we have assumed here. We are also investigating ways of assessing the fit of existing trustcards to trust assessment requests coming from contexts that are not identical to the context of the generation of the trustcards.

Acknowledgment

The work reported in this paper has been has been funded by the European Commission Information Society Technologies Programme as part of the projects SERENITY (contract FP6-27587) and SLA@SOI (contract FP7-216556).

References

[1] M. Ahsant, et al., “Dynamic Trust Federation in Grids”, Proc. of 4th Int. Conf. on Trust Management, 2006

[2] S. Anderson, et al., “Web Service Trust Language”, specs.xmlsoap.org/ws/2005/02/trust/WS-Trust.pdf, 2005

[3] M. Blaze, et al., “KeyNote: Trust Management for Public-Key Infrastructures”, Proc. of the 1998 Security Protocols International Workshop, LNCS, Vol. 1550, 1998

[4] M. Coetzee and J. Eloff, “Autonomous trust for web services”, Internet Research: Electronic Networking Applications and Policy, Vol. 15, No. 5. 2005, pp. 498-507

[5] L. Corritore, et al., “On-line trust: concepts, evolving themes, a model”, Int. J. of Human-Computer Studies, 58(6): 737-758, 2003

[6] T. Dimitrakos, “Towards a Trust and Contract Management Framework for Dynamic Virtual Organisations”, Proc. of eChallenges 2004. 2004

[7] B.J. Fogg, et al., “What Makes Web Sites Credible: A Report on a Large Quantitative Study”, Proc. of CHI 2001, 2001

[8] T. Grandison and M. Sloman, “Trust Management Tools for Internet Applications”, Proc. of the 1st International Conference on Trust Management, LNCS 2692, 2003

[9] A. Jøsang. “Trust and Reputation Systems”, In Foundations of Security Analysis and Design IV, FOSAD 2006/2007 Tutorial Lectures. Springer LNCS 4677, 2007.

[10] S. Lo Presti, M. Butler, M. Leuschel and C. Booth, “Holistic Trust Design of E-Services”, In Trust in E-Services: Technologies, Practices and Challenges, 113-139, IDEA Group, 2007

[11] S. Lo Presti, “Trusted Computing”, In Hacking Exposed Linux, 3rd edition. McGraw-Hill Osborne, 2008

[12] K. Mahbub and G. Spanoudakis, “Monitoring WS Agreements: An Event Calculus Based Approach”, Test and Analysis of Service Oriented Systems, (eds) L. Baresi, E. diNitto, Springer, 2007, pp. 265-305

[13] K. Androutsopoulos, C. Ballas, C. Kloukinas, K. Mahbub and G. Spanoudakis, “V1 of dynamic validation prototype”. Deliverable A4.D3.1, SERENITY, 2006

[14] D.H. McKnight and N.L. Chervany, “The Meanings of Trust”, Technical Report MISRC Working Paper Series 96-04, University of Minnesota, 1996.

[15] N. Osman and D. Robertson, “Dynamic Verification of Trust in Distributed Open Systems”, Proc. of IJCAI 2007, 2007

[16] D. Osman, et al. “Run-Time Model Checking of Interaction and Deontic Models for Multi-Agent Systems”, Proc. of AAMAS 2006

[17] P. Resnick et al. “Reputation systems”, Communications of the ACM, 43(12):45-48, 2000

[18] M. P. Shanahan, “The Event Calculus Explained”, in Artificial Intelligence Today, Lecture Notes in Artificial Intelligence 1600: 409-430, Springer, 1999

[19] H. Shrobe and J. Doyle, “Active trust management for autonomous adaptive survivable systems”, In Self-Adaptive Software, LNCS 1936, 2001

[20] G. Spanoudakis, “Dynamic Trust Assessment of Software Services”, Proc. of 2nd International Workshop on Service Oriented Software Engineering (IW-SOSE '07), 2007

[21] G. Spanoudakis, C. Kloukinas, and K. Androutsopoulos, “Towards security monitoring patterns”, Proc. of the 2007 ACM Symposium on Applied Computing, 2007

[22] S. Staab, et al. “The Pudding of Trust”, *IEEE Intelligent Systems* 19, 5 (Sep. 2004), 74-88

[23] Trusted Web Services, http://www.trustedwebservices.org/