2010 IEEE International Conference on Services Computing

A Trust Ontology for Semantic Services

Wanita Sherchan, Surya Nepal, Jonathon Hunklinger, Athman Bouguettaya
CSIRO ICT Centre, Australia
Email: {Firstname.Lastname}@csiro.au

***Abstract*—We propose a novel semantic service trust organization that uses an ontological approach to model service trust. In particular, our ontology-based organization supports various trust phases including trust bootstrapping, atomic service trust, trust composition and trust propagation. We describe the implementation of the proposed trust organization.**

***Keywords*-Semantic Services; Web Services; Trust Ontology; Trust Management; Trust Service; Reputation**

I. Introduction

Emerging trends in the World Wide Web indicate a shift from the current data-oriented Web to a service-oriented Web [1]. The service-oriented Web (*Service Web*), consisting of services and the Web enriched with semantics, will create an environment where users and applications can query and compose services in an automatic and seamless manner. Enriching the Web with semantics would facilitate the organization, location and quality-based querying of these services. As a result, the Service Web will be a place where a large number of Web services will compete to offer similar functionalities. The Service Web will have unique characteristics such as (i) a service provider may also be a service consumer and vice-versa, (ii) services may outsource part of their functionality to other Web services [2], (iii) services may not have interacted before, and (iv) services may provide misleading information about their Quality of Service (QoS). For a service consumer, a major concern would be selecting which service to invoke among the competing services. Therefore, a key requirement for the Service Web is to provide a *trust* framework for quality access and retrieval of services [3].

Various approaches have been proposed for establishing trust in Web Services [4], [5], [6], [7]. Since the Service Web environment is open, distributed, and semantically enabled, it is not only necessary to have trust techniques but these techniques should also be annotated with semantics to facilitate quality access and retrieval of trust information. This is especially significant considering the fact that various trust algorithms and metrics [8] are available for trust management. In this respect, there is a need for a common trust service ontology that captures the relationships between various trust concepts relevant to service-oriented environments. Such a trust service ontology would improve knowledge re-usability and semantic interoperability of trust services regardless of trust algorithms/mechanisms used by the services.

A number of trust ontologies have been proposed for a variety of application domains in an effort to organize and formalize trust concepts and relationships [9], [10], [11], [12], [13], [14], [15]. Current approaches to trust ontology have focused on a particular aspect of trust such as transitivity [9], trust classification based on the types of trustees in the environment [11], dynamic characteristics of trust [14], analysis and modelling of trust relationships [3] and trust classification based on trust decision input factors [15]. Trust ontologies specifically developed for semantic service-oriented environments also focus on issues such as the properties/dimensions of trust [12], types of trust based on how trust is evaluated in semantic service-oriented environments [11] and how trust ontology can be used for selection based on security and trust [10]. Existing service trust ontologies have focused on concepts related to trust evaluation and making trust decisions. These ontologies do not consider the general trust concepts such as *trust bootstrapping* and trust concepts specific to services such as *trust composition* and *trust propagation* in composite services.

This paper introduces the concept of *trust service ontology* and discusses trust concepts and their relationships specific to semantic service-oriented environments. In this ontology, we consider not only services trust but also *trust as a service*. Our trust service ontology captures the whole life cycle of services trust - from initialization of trust with *trust bootstrapping* to various phases specific to services trust, such as *composition* and *propagation* of trust in composite services. This is a major departure from the current literature that focuses on trust evaluation and trust decision making. The major contributions of the paper can be summarized as follows.

- The proposed approach is "truly" service-oriented where each node of the ontology, representing a stage in trust life cycle, is mapped to the service providing this functionality. This enables easy representation of the whole life cycle of services trust.
- The proposed approach differentiates and explicitly defines the relationship between trust concepts and the trust algorithms. Existing approaches do not make such explicit distinction between them.
- Separating trust algorithms from trust concepts enables the proposed approach to model a large number of

978-0-7695-4126-6/10 $26.00 © 2010 IEEE
DOI 10.1109/SCC.2010.42

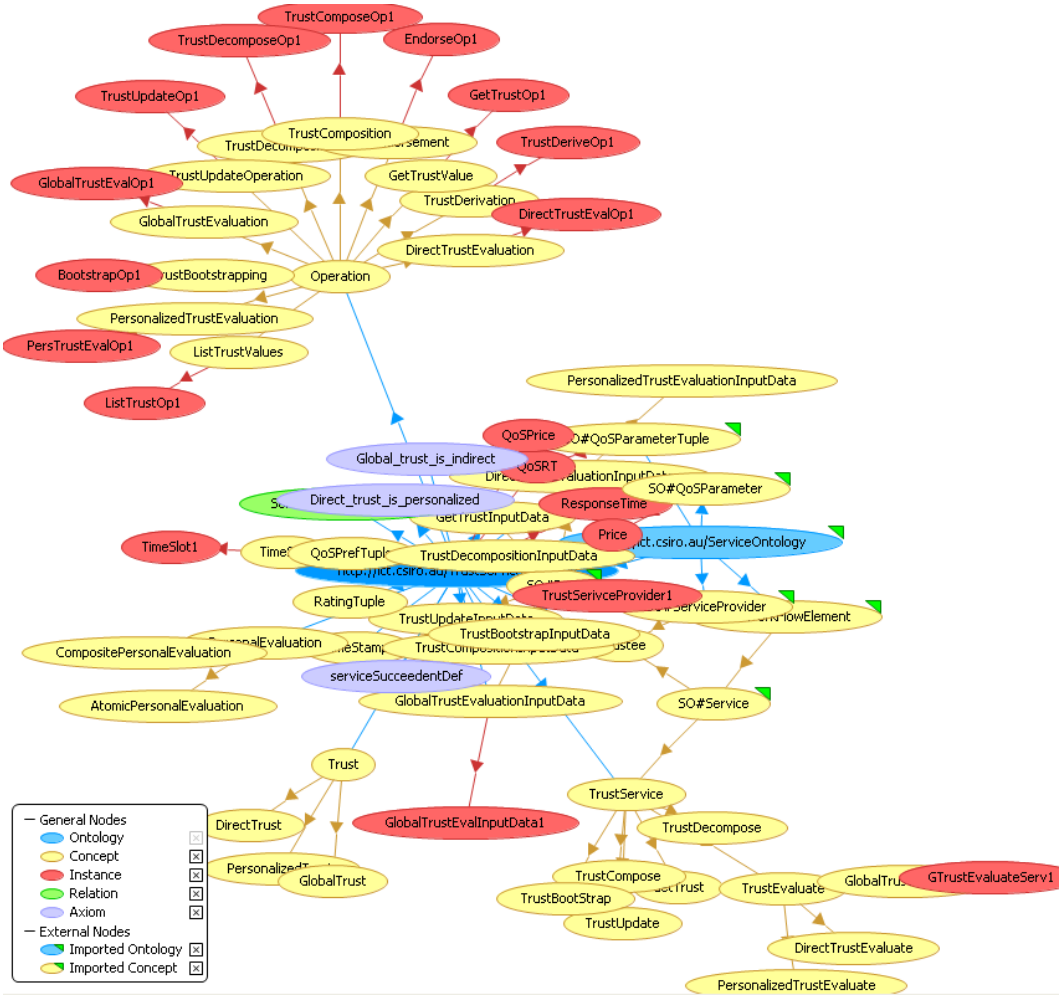

Figure 1. Trust service ontology

trust algorithms provided by different service providers. For example, a number of bootstrapping algorithms provided by different third parties can be mapped to the node representing "trust bootstrapping" service.

The remainder of the paper is structured as follows. Section II presents the major concepts and relationships in our proposed trust service ontology. Section III discusses the implementation of a trust management framework that uses the proposed trust service ontology. Using a case study of a travel scenario, we describe how the trust service ontology helps in services trust management. Finally, Section IV concludes the paper and points to future directions.

II. Trust Service Ontology

The trust service ontology we propose focuses on reputation-based trust in Service Web. We use a travel scenario throughout the paper to illustrate the concepts in the proposed trust service ontology. The ontology is based on the notion of *community*. Consider a travel scenario with three communities: *airline*, *hotel* and *taxi*. Trust for an airline `DomesticQantas` will be evaluated within the airline community. A service or a provider may be registered with more than one community. For example, the `Qantas airways` may be registered to both Domestic and International airline communities. It would maintain separate trust levels in the separate communities. Trust level attained in one community can be transferred to another community as a form of trust bootstrapping in the new community. However, trust is separately maintained in all the communities a service/provider is registered with. Thus, communities form a kind of differentiation of trust context. Services in the three communities: *airline*, *hotel* and *taxi* can be composed to form travel composite (package) services that belong to the *travel* community. Throughout this paper, we use Alice and Bob as two registered consumers of the *travel* community who travel frequently for business and leisure.

A. Basic Concepts

Figure 1 shows the visualization of the proposed Trust Service Ontology in Web Service Modelling Language (WSML) [16]. The ontology has three basic concepts - *trust*, *trust service* and *operation*. Other concepts in the ontology are based on these basic concepts.

Trust: The proposed trust service ontology considers *trust* to be reputation-based. It is evaluated based on feedback ratings provided by service consumers. The concept of *Trust* is defined as a tuple:

`Trust[Trustee,Trustor,TimeStamp,TrustValue, EvaluationCriteria,Confidence,EvaluationPeriod, NumOfInteractions]`

Trustee is a service/provider to which the *Trust* refers. *Trustor* is the entity whose level of trust on the trustee is captured by the *Trust*. *TimeStamp* is the time when the trust value was generated. *TrustValue* is the actual trust value such as "7" (numerical) or "very trustworthy" (fuzzy). *EvaluationCriteria* is the criteria based on which trust is evaluated (i.e., the QoS parameters for trust evaluation). *Confidence* is the level of confidence on the trust evaluation. *EvaluationPeriod* is the duration of history considered in the trust evaluation. *NumOfInteractions* specifies the size of history (i.e., the number of past interactions) considered in the trust evaluation.

Inclusion of information such as *TimeStamp*, *EvaluationCriteria*, *NumOfInteractions*, and *EvaluationPeriod* provides context to *Trust* such that two *Trust* instances for the same service/provider may be compared directly based on these properties. For example, two instances of *Trust* referring to the same service `DomesticQantas` have different *TimeStamp* values. This implies that one of the instances is more recent than the other and therefore more indicative of the current trustworthiness of that service. Figure 2 shows the WSML representation of an instance of *Trust* for `DomesticQantas`, an *airline* service.

```
instance TrustInstance1 memberOf Trust
    hasTrustee hasValue DomesticQantas
    hasTrustor hasValue Airline
    hasTimeStamp hasValue "2009-01-25-12:40"
    hasTrustValue hasValue "0.8"
    hasEvaluationCriteria hasValue price, security, reliability, performance
    hasConfidence hasValue "0.8"
    isBasedOnEvaluationPeriod hasValue "1-02-23"
    isBasedOnNumOfInteractions hasValue "136"
```

Figure 2. An instance of `Trust` in WSML

Trust Service: A *TrustService* is a Web service that provides various trust functionalities such as trust bootstrapping, trust evaluation, trust update, trust composition and trust propagation. *TrustService* can be of several types based on the type of functionality provided by the service. Each of these are defined as a subconcept of *TrustService*. At the abstract level, a *Trust Service* is defined by the tuple:

`TrustService[ServiceID,ServiceProvider, QoSParameterTuple,Input,Operation,Output]`

ServiceProvider is the provider of the *TrustService*. *QoSParameterTuple* defines a list of QoS parameter tuples consisting of pairs of QoS parameter and advertised QoS value for that parameter. *Input* is the input to the *TrustService*. *Operation* defines the operations/functionalities provided by the *TrustService*. *Output* defines the output of the *TrustService*.

Specific types of *TrustServices* have specific definitions for the Input, Operation and Output attributes. All other attributes remain the same. Figure 3 shows the WSML representation of the abstract concept *TrustService*.

```
concept TrustService
    nonFunctionalProperties
        description hasValue "concept of trust service"
    endNonFunctionalProperties
    hasServiceID ofType _string
    hasServiceProvider ofType ServiceProvider
    hasQoSParameterTuple ofType QoSParameterTuple
    hasInput ofType Data
    hasOperation ofType Operation
    hasOutput ofType Data
```

Figure 3. WSML definition of the concept `TrustService`

Operation: An *Operation* specifies the particular location and functionality of a service. A service may have one or more operations. An *Operation* is defined by the tuple:

`TrustService[OperationName,SoapLocation,SoapAction]`

where all attributes are self descriptive. Figure 4 shows the WSML representation of an instance of the concept *Operation*.

```
instance AirlineBootstrapOp memberOf Operation
    hasOperationName hasValue "AirlineBootstrapOperation"
    hasSoapLocation hasValue
    "http://152.83.70.208:8080/axis2/services/TravelTrustManagementService"
    hasSoapAction hasValue "urn:TrustBootstrap"
```

Figure 4. An instance of `Operation` in WSML

B. Types of Services Trust

We have identified several types of services trust based on the purpose of trust evaluation. All of these trust types are sub-concepts of *Trust*. Figure 5 shows the different types of trust.

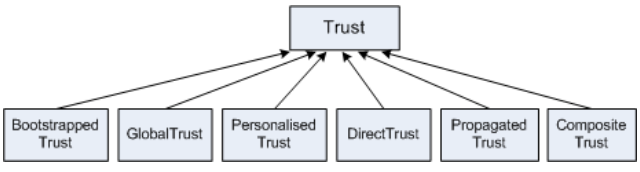

Figure 5. Types of services trust

Bootstrapped Trust: A new service/provider in a community needs to be assigned a nominal trust value to ensure that newcomers are not unfairly disadvantaged [17]. We define such initial trust as *BootstrappedTrust*. For example, a new flight service `DomesticVirginBlue` is registered in the *airline* community. Since this service is new, its trust cannot be calculated from interaction history. Therefore, this service will be assigned a nominal *BootstrappedTrust*. As `DomesticVirginBlue` gets invoked and evaluated by consumers, its *BootstrappedTrust* will be updated to reflect its trustworthiness. When `DomesticVirginBlue` gets registered in another community, its current trust in the *airline* community will be used to bootstrap its trust in the new community. *BootstrappedTrust* is defined by the tuple:

`BootstrappedTrust[Trustee,Trustor,TimeStamp, BootstrappedTrustValue]`

Trustor is the community within which the Trustee's *Trust* is initialized, i.e., in the above example, the *airline* community.

Global Trust: In a community based environment, when trust evaluation is based on the collective perception of the whole community, such trust is termed as *GlobalTrust*. Everyone in the community has the same level of trust for a particular trustee (service/provider). The concept of community based trust (i.e., *GlobalTrust*) is defined by the tuple:

`GlobalTrust[Trustee,Trustor,TimeStamp, GlobalTrustValue,EvaluationCriteria,Confidence, EvaluationPeriod,NumOfInteractions]`

Trustor is the community within which *GlobalTrust* is evaluated. In the above example, as `DomesticVirginBlue` gets invoked and evaluated by its consumers, its *BootstrappedTrust* will be updated to become its *GlobalTrust*. All feedback received by `DomesticVirginBlue` will be incorporated to obtain its *GlobalTrust*.

Personalised Trust: When trust evaluation incorporates the trustor's preferences with respect to various quality parameters, such trust is termed as *PersonalisedTrust*. When trust is personalised, the trust evaluation for the same trustee may be different depending on who the trustor is. Furthermore, trustor identification may be used to weigh the personalised trust values supplied by the trustor. *PersonalisedTrust* is defined by the following tuple:

`PersonalisedTrust[Trustee,Trustor,TimeStamp, PersonalisedTrustValue,EvaluationCriteria,Confidence, EvaluationPeriod,NumOfInteractions,QoSPreferences]`

Trustor is the consumer whose preferences have been considered in the personalised trust evaluation. *QoSPreferences* defines the preferences of the trustor with value pairs of *QoSParameter* and corresponding importance of that parameter to the user specified by *Weight*. In our example, for airline services, Alice considers on time departure of flights to be more important than price whereas Bob prefers flights to be cheap regardless of their punctuality. Given these differences, for the same flight service `DomesticVirginBlue`, Alice's *PersonalisedTrust* would be different from Bob's.

Direct Trust: The concept of trust based on direct past interactions between the trustor and trustee is defined as *DirectTrust*. Direct trust is a type of *PersonalisedTrust*. The concept of *DirectTrust* is defined by the tuple:

`PersonalisedTrust[Trustee,Trustor,TimeStamp, DirectTrustValue,EvaluationCriteria,Confidence, EvaluationPeriod,NumOfDirectInteractions]`

For *DirectTrust*, trustor preferences for various QoS parameters do not need to be specified. Trust evaluation is based on direct past evaluations, and therefore, trustor preferences are implicit in the evaluations. In our example, Alice's *DirectTrust* on `DomesticVirginBlue` is computed based only on her past evaluations of `DomesticVirginBlue`.

Composite Trust: The concept of trust for a composite service, i.e., *CompositeTrust*, is based on the trust for the component services. Various algorithms may be used to determine the composite trust value from the atomic trust values of the component services and may depend on the type of composition (i.e., vertical, horizontal or mixed) [18]. The concept of composite trust is defined by the following tuple:

`CompositeTrust[Trustee,TimeStamp,CompositeTrustValue]`

CompositeTrust is typically used to facilitate selection among different service compositions providing the same functionality. Therefore, typically, *CompositeTrust* would be single use, i.e., used for comparisons and then discarded. For composite services that are likely to be used/invoked by many consumers, *CompositeTrust* is stored and will be regarded as the *GlobalTrust* for the composite service. In this respect, *CompositeTrust* can be considered as *BootstrappedTrust* for composite services. In our scenario, consider a new composite service `WeekendGetawayPackage` consisting of existing services `DomesticQantas`, `HolidayInn` and `SilverTopTaxi` is available in the travel community. When Alice is considering weekend package options, the system will compute the *CompositeTrust* for `WeekendGetawayPackage` based on the trust for `DomesticQantas`, `HolidayInn` and `SilverTopTaxi`. Since `WeekendGetawayPackage` is available all through summer, its *CompositeTrust* will be considered to be its *BootstrappedTrust* and stored. From this point onwards, consumers may evaluate the composite service `WeekendGetawayPackage` just like an atomic service. Ratings provided for `WeekendGetawayPackage`

will be used to update its *GlobalTrust*.

Propagated Trust: The concept of trust for the component services propagated from the trust score assigned to the composite service is termed as *PropagatedTrust*. Similar to *CompositeTrust*, *PropagatedTrust* also depends on the type of service composition, namely, horizontal, vertical or hybrid. The concept of *PropagatedTrust* is defined by the tuple:

`PropagatedTrust[Trustee,TimeStamp,`
`PropagatedTrustValue,TypeOfPropagation]`

TypeOfPropagation specifies whether the propagation is vertical, horizontal or hybrid, consistent with the type of service composition. In the above example, ratings assigned to the composite service `WeekendGetawayPackage` are propagated to the component services `DomesticQantas`, `HolidayInn` and `SilverTopTaxi` to obtain their *PropagatedTrust*. The *PropagatedTrust* for each component service is then used to update the corresponding *GlobalTrust*.

C. Trust Bootstrapping

Evaluating the trustworthiness for newly deployed Web services (newcomers) is a major issue in semantic service-oriented environments because historical information is not available regarding newcomer behaviours. When a service or a provider first joins a community, it is necessary to bootstrap their trust, i.e., assign them an initial trust value. If the newcomer is assigned the lowest default trust value, they may be overlooked over other services. If the newcomer is assigned a high initial trust value, then services/providers have the motivation to discard their identities and start fresh if their trust level falls below a certain threshold. Therefore, the trust management system should not only promote newcomers in the community, but also encourage existing members to keep their reputation/trust profiles.

The concept of *trust bootstrapping* refers to initialization of trust for a newcomer in the community. Trust bootstrapping can be performed using several algorithms such as direct inference and community based bootstrapping [17]. Regardless of the algorithms used, our service trust ontology captures trust bootstrapping as a *TrustBootstrap* service, a subconcept of *TrustService*. The *TrustBootstrap* service bootstraps/initializes the trust of a new trustee in the community. Since *TrustBootstrap* is a subconcept of *TrustService*, it has all the attributes of *TrustService* and is defined by the tuple:

`TrustBootstrap[ServiceID,ServiceProvider,`
`QoSParameterTuple,BootstrapInput,`
`BootstrapOperation,BootstrapOutput]`

BootstrapInput defines the input data required for the *TrustBootstrap* service consisting of *TrusteeID* and *CommunityID* indicating the service/provider whose trust is to be initialized within a particular community. *BootstrapOperation* defines the bootstrap operations/functionalities offered by the service such as community based trust bootstrapping [17] and endorsement (a form of bootstrapping). *BootstrapOutput* defines the output of the *TrustBootstrap* service, i.e., *BootstrappedTrust*. Figure 6 shows the WSML representation of the *TrustBootstrap* service.

```
instance TrustBootstrapServ1 memberOf TrustService
    hasServiceID hasValue "WS2005"
    hasServiceProvider hasValue TravelTrustManagementServiceProvider
    hasQoSParameterTuple hasValue QoSPrice
    hasQoSParameterTuple hasValue QoSRT
    hasBootstrapInput hasValue BootstrapInput1
    hasBootstrapOperation hasValue BootstrapOp1
    hasBootstrapOutput hasValue BootstrapOutput1

concept BootstrapInput1 memberOf Data
concept BootstrapOutput1 memberOf BootstrappedTrust

instance QoSPrice memberOf QoSParameterTuple
    hasQoSParameter hasValue Price
    hasQoSValue hasValue "$6"

instance QoSRT memberOf QoSParameterTuple
    hasQoSParameter hasValue ResponseTime
    hasQoSValue hasValue "30secs"

instance Price memberOf QoSParameter
instance ResponseTime memberOf QoSParameter
```

Figure 6. An instance of `TrustBootstrap` service in WSML

D. Trust Composition and Propagation in Composite Services

Trust composition and propagation are the major trust concepts that are specific to the services domain. Complex services can be composed from atomic services to fulfil user queries. Therefore, the trust for the composite service needs to be evaluated from the trust on the component services.

Trust Composition: The *TrustCompose* service composes the trust value of a composite service from the trust values of the component services. This feature is useful for comparison of trust for candidate service compositions. *TrustCompose* is a subconcept of *TrustService* and is defined by the tuple:

`TrustCompose[ServiceID,ServiceProvider,`
`QoSParameterTuple,ComposeInput,`
`ComposeOperation,ComposeOutput]`

ComposeInput defines the input data required for the *TrustCompose* service consisting of *TrusteeID*, *NumOfComponentServices*, *ComponentServiceIDs* and *CompositionWeights*. *ComposeOperation* defines the trust composition operations/functionalities offered by the service. *ComposeOutput* defines the output of the *TrustCompose* service, i.e., *CompositeTrust*. In the context of trust composition, *CompositionWeights* is significant as it specifies the importance of each component service in the service composition. This information is obtained from the domain ontology. In our travel scenario, for travel package

services consisting of *airline*, *hotel* and *taxi* services, *airline* service may have higher importance, and hence higher weights in the composition, compared to other services. This information would be obtained from the travel domain ontology which would capture historical information on relative importance of a component service in a service composition.

Trust Propagation: Similar to trust composition, trust propagation is another significant feature of services trust. The *TrustPropagate* service, a subconcept of *TrustService*, decomposes and propagates the trust value assigned to a composite service to its component services. It is defined by the tuple:
`TrustPropagate[ServiceID,ServiceProvider, QoSParameterTuple,PropagateInput, PropagateOperation,PropagateOutput]`
PropagateInput defines the input to the service consisting of *CompositeServiceID*, *AssignedTrustValue*, *NumOfComponentServices*, *ComponentServiceIDs* and *CompositionWeights*. *AssignedTrustValue* specifies the trust value provided by the consumer to the composite service, i.e., the trust value to be propagated to the component services. For example, Alice assigns a rating *8* to the `WeekendGetawayPackage`. This rating would be propagated to the component services `DomesticQantas`, `HolidayInn` and `SilverTopTaxi` by a `TrustPropagate` service implementing an algorithm such as [19].

E. Trust Evaluation and Update

We have identified different types of trust evaluations for different types of trust. Trust evaluations are subconcepts of *TrustService*.

Direct Trust Evaluation: The *DirectTrustEvaluate* service evaluates the *DirectTrust* for a given trustee (service/provider) from a given trustor's perspective. It is defined by the following tuple:
`DirectTrustEvaluate[ServiceID,ServiceProvider, QoSParameterTuple,DirectEvaluateInput, DirectEvaluateOperation,DirectEvaluateOutput]`
DirectEvaluateInput consists of *TrusteeID*, *TrustorID* and *EvaluationPeriod*. *DirectEvaluateOutput* specifies the output of the *DirectTrustEvaluate* service, i.e., *DirectTrust*.

Personalised Trust Evaluation: The *PersonalisedTrustEvaluate* service evaluates the *PersonalisedTrust* for a given service/provider(trustee) from a given trustor's point of view. It is defined by the tuple:
`PersonalisedTrustEvaluate[ServiceID,ServiceProvider, QoSParameterTuple,PersonalisedEvaluateInput, PersonalisedEvaluateOperation, PersonalisedEvaluateOutput]`
PersonalisedEvaluateInput consists of *TrusteeID*, *TrustorID*, *EvaluationPeriod* and *QoSPreferences*. *QoSPreferences* specify the preferences of the trustor for the personalised trust evaluation.

Global Trust Evaluation: The *GlobalTrustEvaluate* service evaluates the *GlobalTrust* for a given trustee (service/provider) and is defined by the tuple:
`GlobalTrustEvaluate[ServiceID,ServiceProvider, QoSParameterTuple,GlobalEvaluateInput, GlobalEvaluateOperation,GlobalEvaluateOutput]`
GlobaltEvaluateInput consists of *TrusteeID*, *TrustorID* and *EvaluationPeriod*. The *Trustor* in this case is the community within which the *GlobalTrust* is evaluated.

Trust Update : Evaluated trust values whether they be *GlobalTrust* or *PersonalisedTrust* or *DirectTrust* need to be updated periodically to keep them current. The concept of *TrustUpdate*, a subconcept of *TrustService* captures this functionality. It is defined by the tuple:
`TrustUpdate[ServiceID,ServiceProvider, QoSParameterTuple,UpdateInput, UpdateOperation,UpdateOutput]` *UpdateInput* consists of *TrusteeID* and *UpdateTarget*. *UpdateTarget* specifies which trust instances are to be updated, i.e., whether instances of *GlobalTrust* or *PersonalisedTrust* or *DirectTrust* or all trust instances are to be updated.

III. Ontological Support for Trust Management

Based on the proposed *trust service ontology*, a trust management framework for the Service Web has been implemented in the context of Web Service Management System (WSMS) [20]. The framework is implemented and deployed as a trust management service. The trust management service can be used independently as well as integrated as a component of WSMS. In the following, we discuss the implementation architecture and explain trust management using our proposed trust service ontology in the case of a travel scenario described in Section II.

Figure 7 shows the implementation architecture of the proposed ontology-based trust management framework. The trust management framework is responsible for managing trust of atomic services as well as composite services. The current implementation has three trust services that provide various trust functionalities, three Web services to support trust management and several Web portals for invoking those services. The Web Services are developed using Java and deployed on Apache Tomcat/5.5.27. using the Apache Axis2 Web service version 1.4.1. The Web services and their access stubs were created using the Axis2 eclipse plug-ins, Axis2 Service Archiver and Axis2 Code Generator. Next, we describe the components of the framework and demonstrate how the trust service ontology is used in trust management.

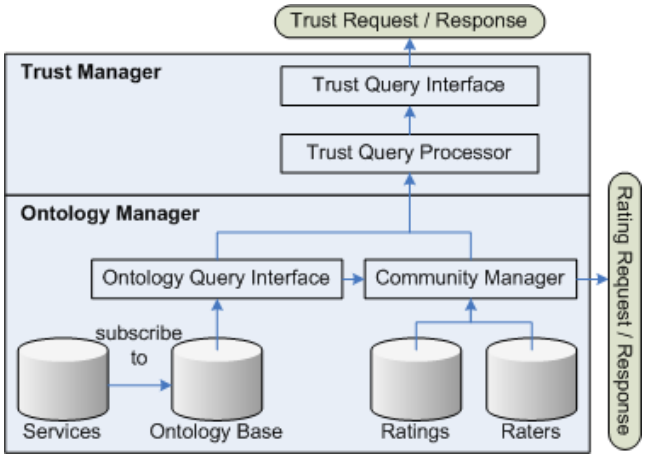

Figure 7. An implementation architecture

Ontology Manager: The ontology manager includes a community manager, an ontology base and an ontology query interface. The *community manager* uses ontology to organize services into different types of communities. Services providing a similar type of service belong to the same community. For example, all airline services belong to an airline community. They also belong to the `travel` community as the concept `airline` is a sub-concept of the concept `travel`. Similarly, all travel package services belong to the `travel` community. The community manager also maintains *raters* and *ratings*. The *raters* are the consumers who are willing to share their experiences with others, and the *ratings* are the feedback given by the consumers against a service's QoWS parameters. The community manager provides appropriate interfaces to the other components of the *trust management service*. Additionally, it provides Web portals for supporting a variety of functionalities such as rater registration, consumer feedback and service management (e.g., addition of new service or QoWS parameter for a service).

The *ontology base* consists of two ontologies - a *service ontology* and a *trust service ontology*. The *service ontology* stores the ontology definitions of services within a specific domain (in our case, the "travel" domain). The *trust service ontology base* stores the ontology definitions of services trust and trust services. The trust service ontology also models the relationships between different types of trust and the dependencies between different trust functionalities. A node in the trust service ontology defines a type of functionality offered by a service. Some examples of such nodes are `TrustBootstrapping`, `TrustComposition` and `TrustEvaluation`. Each node in the ontology is associated with a list of services that provide the defined functionality. This association enables functionality-based service discovery by first locating the corresponding node in the ontology, then locating the Web services that subscribe to the node.

The *ontology query interface* supports two types of queries in the ontology base: *functionality query* and *Web service query*. The functionality query is to locate a node in the ontology and retrieve related information such as its relationship with other nodes. The Web service query is to find a list of Web services that provide a specific functionality, identified by a certain node in the ontology. The corresponding concrete Web services can be retrieved by checking whether they subscribe to a node.

Trust Manager: The trust manager includes a trust query interface and a trust query processor. The *trust query interface* provides a query interface to the users of the *trust management service*. The current implementation supports two types of interfaces: Web service and Web portal. The Web service interface is used by other Web services to query trust of a service (or services). For example, a service orchestrator in WSMS may call the trust service to inquire the trust value of a particular service using the Web service interface. The Web portal interface is designed to be used by an end user (or customer). This enables the *trust service* to be used independently and directly by the customer.

The *trust query processor* interacts with the ontology manager to process a given trust query. For example, the trust query processor receives a trust query for an airline service `DomesticQantas`. With the help of the ontology manager, it identifies that `DomesticQantas` is a new service in the `airline` community, therefore, no trust information exists for `DomesticQantas`. Then it uses the ontology manager to identify and invoke a suitable *TrustBootstrap* service which will evaluate the *BootstrappedTrust* for the `DomesticQantas` service and store it in the *trust service ontology*. As the `DomesticQantas` service gets invoked and evaluated by consumers in the community, its *BootstrappedTrust* will be updated to become its *GlobalTrust*. The trust query processor will invoke appropriate services to manage this.

In another case, the trust query processor receives a trust query for a composite service, `WeekendGetawayPackage` consisting of the services `DomesticQantas`, `HolidayInn` and `SilverTopTaxi`. With the help of the ontology manager, the trust manager will identify that the `WeekendGetawayPackage` is a new composite service, and therefore, will find and invoke a suitable *TrustCompose* service to compute *CompositeTrust* for the `WeekendGetawayPackage` service based on the trust for the component services. Since the `WeekendGetawayPackage` service is available all through summer, its *CompositeTrust* value will be stored as its *BootstrappedTrust* and updated as it gets invoked and

evaluated by consumers.

Each time the composite service `WeekendGetawayPackage` receives a rating from the consumer, the provided rating will be propagated to the component services, namely, `DomesticQantas`, `HolidayInn` and `SilverTopTaxi` services. The trust query processor, again with the help of the ontology manager, identifies and invokes a *TrustPropagate* service for this propagation of trust. The obtained *PropagatedTrust* for each component service (e.g., `DomesticQantas`) is then used to update its corresponding *GlobalTrust*. Therefore, after each invocation of *TrustPropagate* service, a *TrustUpdate* service will need to be invoked. The trust query processor obtains this dependency information from the *trust service ontology*.

IV. Conclusion

We have presented a trust service ontology for the Service Web. The key feature of the proposed ontology is that it captures the whole life-cycle of services trust, from initialization of trust with *bootstrapping* to all phases relevant to services trust such as *composition* and *propagation* in composed services. In addition, our trust service ontology incorporates all standard concepts in trust such as trust *evaluation* and *update*. We classify trust based on the purpose of trust evaluation. Our trust service ontology not only considers services trust but also trust as a service. As a result, we identify and formalize different types of trust services and their relationships. We have implemented a trust management framework that utilises our proposed trust service ontology. We presented a case study of a travel scenario to illustrate how our trust service ontology helps in developing and deploying efficient and scalable trust management in semantic service-oriented environments.

References

[1] Q. Yu, X. Liu, A. Bouguettaya, and B. Medjahed, "Deploying and Managing Web services: Issues, Solutions, and Directions," *VLDB Journal*, vol. 17, no. 3, pp. 537–572, 2008.

[2] M. Tian, A. Gramm, H. Ritter, and J. Schiller, "Efficient selection and monitoring of qos-aware web services with the ws-qos framework," in *WI'04 proceedings*, Washington, DC, USA, 2004, pp. 152–158.

[3] W. Zhao and V. Varadharajan, "Trust Management for Web Services," in *ICWS'08 proceedings*, Washington, DC, USA,, 2008, pp. 818–821.

[4] E. M. Maximilien and M. P. Singh, " Conceptual Model of Web Services Reputation ," *ACM SIGMOD Record*, vol. 31, no. 4, pp. 36–41, 2002.

[5] S. Elnaffar, Z. Maamar, H. Yahyaoui, J. Bentahar, and P. Thiran, "Reputation of communities of web services - preliminary investigation." in *AINA'08 Workshop proceedings*, 2008, pp. 1603–1608.

[6] E. Chang, T. S. Dillon, and F. Hussain, *Trust and Reputation for Service-Oriented Environments: Technologies For Building Business Intelligence And Consumer Confidence.*, 2006.

[7] W. Conner, A. Iyengar, and T. Mikalsen, "A Trust Management Framework for Service-Oriented Environments," in *WWW'09 Proceedings*, 2009, pp. 891–900.

[8] S. Ruohomaa and L. Kutvonen, "Trust management survey," in *iTrust'05 Proceedings*, 2005, pp. 77–92.

[9] J. Huang and M. S. Fox, "An Ontology of Trust - Formal Semantics and Transitivity," in *ICEC'06 proceedings*, August 2006, pp. 259–270.

[10] S. Galizia, "WSTO: A Classification-Based Ontology for Managing Trust in Semantic Web Services," in *ESWC'06 proceedings*, 2006, pp. 697–711.

[11] E. Chang, T. S. Dillon, and F. Hussain, "Trust Ontologies for E-Service Environments," *International Journal of Intelligent Systems*, vol. 22, pp. 519–545, 2007.

[12] M. Zhu and Z. Jin, "Trust Analysis of Web Services based on a Trust Ontology," in *KSEM'07 proceedings*, November 2007, pp. 642–648.

[13] M. Taherian, R. Jalili, and M. Amini, "PTO: A Trust Ontology for Pervasive Environments," in *AINA'08 Workshop proceedings*, 2008, pp. 301–306.

[14] Q. Zhang, F. Miao, Z. Yuan, Q. Zhang, and Z. Fan, "Construction of a Dynamic Trust Ontology Model," in *CIS'08 proceedings*, 2008, pp. 394–398.

[15] L. Viljanen, *Trust, Privacy and Security in Digital Business*, 2005, vol. 3592/2005, ch. Towards an Ontology of Trust, pp. 175–184.

[16] "The Web Service Modeling Language (WSML)," 2008, http://www.wsmo.org/wsml/wsml-syntax.

[17] Z. Malik and A. Bouguettaya, "Reputation Bootstrapping for Trust Establishment among Web Services," *IEEE Internet Computing*, vol. 13, no. 1, pp. 40–47, 2009.

[18] B. Medjahed and A. Bouguettaya, "A multilevel composability model for semantic web services," *IEEE Trans. on Knowledge and Data Engineering*, vol. 17, no. 7, pp. 954–968, 2005.

[19] S. Nepal, Z. Malik, and A. Bouguettaya, "Reputation Propagation in Composite Services," in *ICWS'09 proceedings*, 2009.

[20] X. Zhou, S. Chen, A. Bouguettaya, and K. Xu, "Supporting bioinformatic experiments with a service query engine," in *SERVICES'09 Proceedings*, 2009, pp. 717–723.