Applied Mathematics & Information Sciences
An International Journal

http://dx.doi.org/10.12785/amis/072L51

A Trustworthy Service Computing Framework through a Semantic Messaging Model

Woongsup Kim

Department of Computer & Information Communication Engineering, Dongguk University, Seoul, Korea

Received: 15 Nov. 2012, Revised: 25 Jan. 2013, Accepted: 28 Jan. 2013
Published online: 1 Jun. 2013

Abstract: In this paper, we present a framework supporting reliable and trustworthy service-oriented workspace sharing environments. To this end, we propose a communication model to facilitate a complex level of interactions among workspace participants. To implement a communication model, we adopted a frame-based approach, in which each frame has slots to be filled. Slots in a frame are analyzed and filled using the semantic information provided from service participants. In addition, we provide a trust model for predicting behaviors of service providers. Our trust model ensures reliability by indicating the extent to which a service is reliable. To estimate risks and the degree of service satisfaction, we employ historical user ratings. In addition, we evaluated a case from the design and manufacturing domain. We conclude that our proposed approach will contribute to realizing trustworthy workspace sharing in a service oriented environment.

Keywords: Service Computing, Semantic Web, Web Services, Computer Supported Collaborative Work, Process Management, Workspace

1 Introduction

Service-computing is a distributed component computing model that supports interoperability among heterogeneous systems, based on platform-neutral open web standard format [1,2]. Service computing is realized through web service technology where its inputs and outputs are composed of XML-based open standard, and any external component can make successful interactions so long as the component complies to input/output specification provided from web services.

Certain challenges that can inhibit the implementation of workspace sharing based on a service-computing paradigm, one of the most noticeable challenges being the issue of ”trustworthiness.” In computing society, trustworthiness inherently addresses to secure, reliable, and available computing systems. During service-oriented workspace sharing processes, participants may face uncertainties in terms of trustworthiness such as service availability, service reliability, service safety, or service integrity. These uncertainties come from two major limitations of current web service technology. In web services, description logics are used to describe functions and capabilities of services [3]. However, they are not well-suited to representing quantitative knowledge in terms of performance and reliability. In addition, communications within web services rely on RPC-type messaging via the Simple Object Access Protocol (SOAP) [4]; however, SOAP is not sufficient to implement the complex communication required for collaboration between participants.

Uncertainties in a service computing environment fall into two categories: pre-purchase uncertainties and fulfillment uncertainties. Pre-purchase uncertainties occur when service consumers cannot decide whether or not to purchase a certain service. Even semantically well-written descriptions of a service may not be good enough for users to be convinced of the service usability. Fulfillment uncertainties involve situations in which a user invokes a service only to find that the service cannot provide the functionality claimed in the descriptions. Service consumers may have concerns about defective or low-quality services.

In response to pre-purchase uncertainty, recommender systems and third-party certification methods have been proposed [5,6]. Recommender systems are methodologies that utilize evaluations records from past transactions in the community. Third-party certification is a traditional business method. For example, some traditional business

* Corresponding author e-mail: woongsup@dongguk.edu

transactions require a Letter of Credit (L/C) which guarantees secure payment. Even though a L/C is issued by only certified banks, third-party certification can be provided by any organization within the community. Although useful, these approaches are not sufficient to provide performance information or reliability information. It is thus hard to build the reliability models or the performance models, that are required to predict and improve the trustworthiness of service partners.

To manage fulfillment uncertainty, agreement-based systems and assertion-based monitoring have been designed [7,8]. Agreement-based systems utilize a contract created by both parties. One specific example is the Service-Level Agreement (SLA) from IBM and HP [9]. To check contract fulfillment, companies maintain a separate module that can check service fulfillment as described in the contract. Another approach is, assertion-based monitoring, in which, assertions are determined based on requirement documents and then inserted into process logic [10]. When execution reaches each embedded assertion point, a service fulfillment indication is reported to clients, and analyzed through formal modeling languages such as SPIN [11].

Although these approaches are beneficial, intrinsic anonymity (i.e., the customer's lack of familiarity or relationship with services and the people involved) still makes users hesitant to join a service-oriented collaborative environment. Complex interaction is a way of testing new boundaries and seeing how far one can go when no one knows about such boundaries. Instead of judging services through categories, interactive communication allows a service consumer to choose which services to use in a service-oriented environment. In this paper, we present a framework for supporting reliable service-oriented workspace sharing environments. To this end, we provide a communication model to support a complex level of interactions among workspace participants. To implement a communication model, we adopted a frame-based approach, rather than the more conventional finite state machine (FSM) approach which has limited flexibility for generating message sequences. In a frame-based approach, messages have slots to be filled. Slots are analyzed and filled using the semantic information provided from service participants. We also exploit Process Grammar [12,13] to implement hierarchical dialogue structure and selection rules.

In addition, we provide a trust model for predicting behaviors of service providers. Our trust model presents probabilities that indicate the extent to which a service is reliable. We employ ratings by previous service users in order to estimate how much the user will be satisfied after using the service. The probabilities are used to build the risks and the degree of service satisfaction. The remainder of this paper is organized as follows. In section 2, we present relevant issues and solutions. Section 3 discusses considerations for implementing a service oriented workspace sharing environment. In section 4, we propose a communicative model to be used for workspace sharing. In section 5, we develop trustworthiness issues for the service oriented environment. In particular, we propose a methodology to be used for behaviors prediction and run-time analysis in services. The implementation is described in section 6, where we also describe experiments to evaluate the proposed models.

2 Background

Generally, Web Services are described using WSDL and cataloged using the Universal Description Discovery and Integration (UDDI) protocol. UDDI registries contain so-called white pages for each registered services [14]. Description Logics [15] are a good candidate for this purpose, and they are used to classify services and ground individual service functionality.

For the trusted service discovery a reputation models are proposed [16,17,18]. Reputation model provides for recording past transactions and evaluations for each service so that clients can estimate available services to improve future service discovery decisions. For extended trustworthy decisions, Quality of Service (QoS) factors are integrated into UDDI. In this model, service behaviors regarding QoS factors are recorded in the service registry, so that future users can refer to them during service selection [19,20]. Such QoS factors include capacity, response time, latency, throughput, and service accuracy.

For secure Web Services, several security methodologies are proposed in various computing domains such as authentication, access control, distributed security policy, and message layer security. Policy-level security is principally concerned with the existence of security guards and their role. There have been many proposals regarding policy-level security. However, they focus mainly on server-side policy. The server decides the access level to its own resources from policy documents. In Web Service environments, services are spatially dispersed such that the design of a policy-level security system on distributed objects on the network is necessary. The most common approach is to separate policy logics from business logic [21]. Clients obtain certifications from access control server for a specific service. A service provider then examines certification from clients to decide access level. Message-level security manages secure message exchanges. A common technique in a traditional lower-level approaches are the Secure Socket, secure Transport Layer, Internet Protocol Security, and implementing Virtual Private Networks. Although this approaches work in many situations, it has a serious disadvantage when applied to a service oriented environment. The traditional lower-level security mechanisms are not appropriate for secure end-to-end communication as the intermediary domains may use different security tunnels. WS-Security [22] is designed to construct secure SOAP message exchanges through end-to-end web applications. WS-Security defines how to sign or encrypt SOAP messages to assure message confidentiality.

Efforts are made to enhance reliability and reduce unreliable service behaviors, which timeouts, runtime errors, and violation of functional contracts. One way to incorporate service level reliability would be to use standardized headers containing information such as transactional bracket markers and context information. Such information is added to messages to be used for monitoring service transactions (such as beginning of transaction, commitment, roll-back and so on) with minimal impact on existing applications. Another approach is to use service choreographies. This approach mainly utilizes annotations inside process logic [23,24], and requires steps of incorporating assertions into the standards business logic. Other approaches include deploying specialized intermediary processes whose specific function is to ensure that the requirements of service usage are being met.

To support workspace sharing in a service oriented environment, many standards such as ARCOL [25], CNP [26], XLBC [27], KQML [28], and FBCL [29] have been proposed for use in software collaboration. These languages are designed for agents to talk with other agents and work together. They are designed to facilitate collaboration process by providing agent communication mechanism since communication play a fundamental role in collaboration [30]. Formal semantics for KQML have been proposed to associate a meaning to a message [31]. The semantics are captured by feasibility conditions and a rational effect. However, even though these approaches can be employed in Web Service-based applications, they have severe limitations when used in certain environments such as collaborative design and processes. The reason comes from the fact that they are not designed to represent quantitative aspects (e.g. performance) within the functional specification of such system.

3 Considerations for Workspace Sharing in Service Computing Framework

To support workspace sharing in service-oriented environment, we observed the required characteristics that should be incorporated into service oriented collaborative behavior modeling. To realize service oriented workspace sharing, we first identify four types of features needed for modeling collaborative process behavior to be shared among partners [32].

– Workspaces should be highly reconfigurable. Workspaces should be easily composed and divided into sub workspaces respecting dynamic business needs.
– End-to-end communication must support complex level of business communications. Communication protocol should not restric the expressive power of business communication.
– Workspace control history should be recorded with time relationship.
– Users or tasks should be grouped and controlled securely.

Considering the above features, we identified the requirements that must be considered in order to realize trustworthy service-oriented workspace sharing in a heterogeneous and distributed collaboration environment.

– *Service Based Deployment*: Services implemented through web services interfaces are accessible from any third party software components. Therefore, to facilitate service interoperability, services should be published with web services interfaces.
– *Proactive Process Enactment*: Business processes often require modification in their business workflows due to business environmental changes. Therefore, the service-oriented framework should be able to dynamically configure process paths at run-time.
– *Sharing Service Execution*: A service model must provide sharing functionality where participants can monitor and evaluate process execution status. In addition, side effects from process execution require run-time coordination between partners in order to accomplish an optimal solution.
– *Multi User Access Control*: For secure collaboration, it is necessary to have appropriate access control support for multiple users, where resources and workspaces are distributed and shared at some level. The system should be able to regulate users' access pattern dynamically: the level of sharing of resources must be restricted by certain policies. Based on user policy, various access levels such as writing, reading, or removing should be assigned to each individual user.
– *Support for Complex Communicational Behavior*: services should be able to perceive their environments and act with other software components. Web service applications should be able to refuse an action when necessary. Services should be able to communicate with each other to resolve complex business issues among collaboration participants.

4 A Communicative Model for Workspace Sharing in Service Oriented Environment

To support service-oriented complex interactions required for workspace sharing, we propose a communicative model to control and guide service operations through end-to-end message exchanges. Our communicative model represents communicative behavior of workspace collaboration among distributed applications. To this end, we employ speech act theory [33], which is widely adopted in common agent communication languages. To generate and exchange web service based communication messages, we take a frame-based approach.

4.1 Communicative Model

Our communicative model is structured in a triple <*Intention, Action, Target*>. Intention refers to the purpose of a dialogue. Intention provides reason of a message in a dialogue. Our model has eight types of Intention such as 'request', 'propose', 'accept', 'reject', 'query', 'answer', 'assertion', and 'declare'. The receiver can identify a message type by checking the Intention field and prepare appropriate responses. For example, if a message has 'query' as an intention, then a service provider may prepare answers. If a message has 'request' intention, then a service provider may prepare providing the corresponding services. Intention of 'declare' can be used for notification purposes. If a service requester has an authority to control an aspect of services, then they can use 'declare' to notify some properties or rules. Messages to be broadcasted can use 'declare' intention. The service provider can deliver the response to a query with 'answer' intention. 'assertion" intention is for regular status reports.

Target represents any instance which appears in a collaborative process. Input/outputs, operations, messages, and exceptions can be described as a Target. The Target field defines a namespace and ontology in the namespace is used to interpret what target means. Target is designed to remove ambiguity in communication as much as possible. In our approach, target uses OWL and RDF to represent itself. Target can be described with associated attributes. For quantitative analysis of system behavior, we add special attributes such as "linkFrom", "linkTo", "hasPreCondition", and "hasEffects", to translate the target into a Petri Net based representation. Since Petri nets are widely used in quantitative analysis of system behavior, our communicative model can, unlike other communication langauges, be used for system behavior analysis and predictions.

Action is classes that can change an object's state or property. Therefore, Action is composed of various information such as object information, object's initial state, operations to affect object states, and operations' effect. We model Action as the following form.

$$\phi \underset{\delta=<e,O>}{\Longrightarrow} \phi'$$

where O is a target object, ϕ encapsulates properties of a target object O before transition, ϕ' encapsulates properties of the object O after transition takes effect, and δ shows a labeling function (enactment property e $\times$ target object O) that enables transitions.

In our communicative model, we defined five execution states representing properties ϕ of a target object (Table 1).

Table 1: Five basic execution states in a communicative model

Execution States	Description
Initial	indicate that a service is not accessed so far.
Ready	show that input data is provided to a service, but that service is not yet activated.
Running	indicate that a service is activated and executing a task at a given time.
Finished	indicate that service execution is completed and the service generates outputs. There are two sub classes *success* and *fail* to notify whether a service execution creates appropriate outputs.
Exception	indicate that a service execution fails due to some external reasons. Various Runtime Exceptions can be defined as subclasses.

We also define six basic types of enactment properties *e* which affect the property of a target object (Table 2). Each enactment property can have multiple attributes based on target characteristics. For example, 'execution' property in 'Casting' service has three attributes 'High Pressure Casting', 'Vacuum Casting', and 'Low Pressure Casting' as execution options.

Table 2: Six types of enactment properties

Enactment Property	Description
ProvideInput	deliver needed data to a service.
RetrieveOutput	retrieve service execution traces to participants
InvokeEnactment	force to start service execution. But service execution traces needs not be reported.
Execution	monitor service execution traces.
Rollback	force rollback when a system exception happens.
EnforcedRollback	force rollback from some business needs rather than system exceptions

Table 3 shows how our communicative model is applied for each task status change. We assume a process is structured linearly in the order data d_0 as input to task t_1, which produces output d_1.

Table 3: Examples of Communicative Model in a general process

Execution Traces to Report	**Example Usage**
Providing necessary inputs to d_0 before starting a task t_1	$initial \xrightarrow[<provideInputs,d_0>]{} ready$
Forcing task t_1 to execute	$initial \xrightarrow[<invokeEnactment,t_1>]{} ready$
Confirm task t_1 in execution	$ready \xrightarrow[<execution,t_1>]{} running$
Confirm task t_1 in execution with 'Vacuum Casting' option.	$ready \xrightarrow[<execution,t_1.VacuumCasting>]{} running$
Task t_1 is completed and bind outputs from task t_1 into a data object d_1	$finished \xrightarrow[<retrieveOutput,t_1>]{} finished$ $initial \xrightarrow[<retrieveOutput,d_1>]{} ready$

4.2 Frame Based Message Exchanges

To support flexible message exchanges and hence to represent complex communicative behavior, we employs a frame-based approach in order to exchange meaningful messages. A frame is defined as a data structure that represents a typical situation. A frame forms the semantics of a concept, and consists of the properties and the contents of each property.

Formally a message can be defined via a λ-expression as follows.

$$\lambda x\ (\lambda y_0\ y_1\ \ldots\ y_{k-1}((x\ z_0)\ y_0\, y_1\ \ldots y_{k-1}\ (x\ z_{k-1}))$$
$$p_0\ p_1\ \ldots\ p_{k-1}\ s$$

where s is a service name, p_i, $0 <$ i $\leq n$, are grounded properties bounded to the service s's capabilities. y_i, $0 <$ i $\leq n$, are capabilities s has, z_i, $0 < i \leq n$, are y_i's contents bounded with types, and k is the number of properties to query.

For example, suppose that a participant wants to query the behavioral semantics of a service 'vacuumAssistedCasting'. Then the corresponding query is given by

$$\lambda x\ (\lambda\ y_0\, y_1 (y_0\ (x\, z_0)\, y_1\ (x\, z_1)))\ composedOf$$
$$hasStatus\ \ vacuumAssistedCasting$$
$$\xrightarrow[\beta-reduction]{}\ ((VacuumAssistendCasting\ z_0)$$
$$hasStatus\ (VacuumAssistedCasting\ z_1)).$$

To answer queries, partners should provide contents with values that can be called as needed. The answering message is represented as follows. Bold faced words represent the message generated from the answering participant.

$(composedOf\ \ (VacuumAssistedCasting\ \ ($**VacuumCasting Trimming QualityAssurance SelectedMaterial Dies TrimDies CastedProduct TrimmedProduct** : **FinishedProduct) :type of String list**$)\ hasStatus$ $(VacuumAssistedCasting$ **applied: type of states**$))$

In order to query non functional behavior for each of the sub functional systems, for example, the corresponding queries take the following forms.

$$\lambda x\ (\lambda y_0\ \ (y_0\ (x\ \ z_0)\ \ processingTime$$
$$vacuumAssistendCasting\ \xrightarrow[\beta-reduction]{}\ ((processingTime$$
$$hasStatus\ (VacuumAssistedCasting\ z_0)).$$

To answer the query, a service provider should provide values for z_0, which can be called by a service consumer. For example, the answer in this case will be

$(processingTime\ \ (VacuumAssistedCasting,$ **15 : type of integer**$))$.

Figure 1 shows the actual query message created from a service participant and the query message is represented with OWL.

5 Trustworthy Workspace Sharing for Service-Oriented Environment

Trustworthiness in service computing has the purpose of improving service provisioning in terms of service reliability and efficiency performance. To realize trustworthy collaborative workspace in service computing environment, we developed a trust model for predicting service behavior. The goal of our trust model is to avoid malicious or poorly working services involving in

```
<service:conversation rdf:ID="message3">
        <service:hasIntention>
                <service:intention rdf:ID="intentionID">
                        <service:parameterType rdf:about="#query"/>
                </service:Intention>
        </service:hasIntention>
        <service:Target rdf:ID="objectClass">
                <service:LogicalService rdf:about="providerNS:#Trimming"/>
        </service:Target>
        <service:Action rdf:ID="actionClass">
                <service:LogicalService rdf:about="providerNS:#Trimming"/>
                        <service:executionTime
rdf:resource="providerNS#contract"/>
                </service:LogicalService>
        </service:Action>
</service:conversation>
```

Fig. 1: A Query Message for Execution Traces)

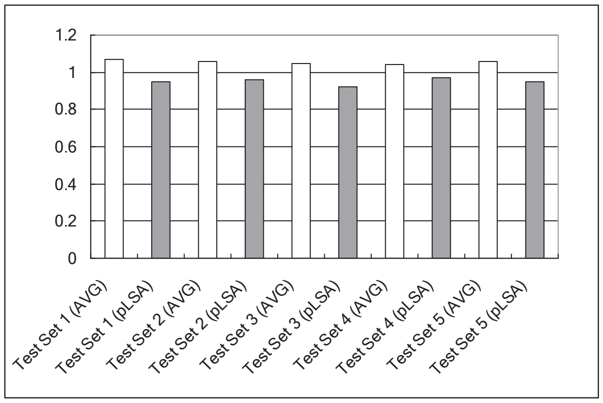

Fig. 2: Performance Comparisons on Average Prediction Errors

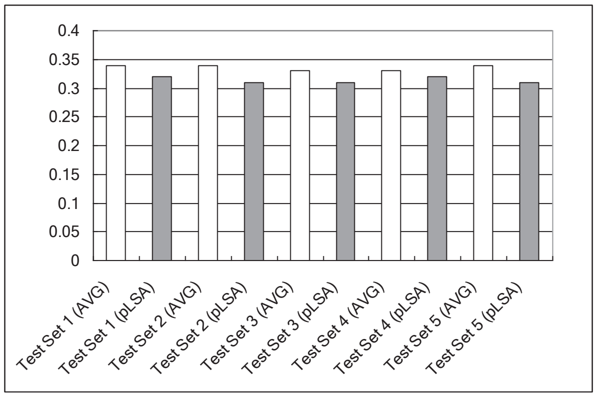

Fig. 3: Performance Comparison on Non-Acceptability Rate

collaborative workspace as participants, by pre-evaluating service reliability and efficiency performance.

To predict reliability, we first estimate the probability that users will give a particular rating to a particular service. We consider a set of users $U = u_1, u_2, \ldots, u_n$, a set of services $S = s_1, s_2, \ldots, s_m$, and a set of ratings $R \subset \aleph$, where n is the number of users and m is the number of available services. We assume that users u_i made explicit ratings $v \in V$ on a service s_j such that $U \times S \to V$.

The first step in our approach is to compute the expected numerical ratings of (u, v) before the actual ratings are known. To this end, we adopt probabilistic Latent Semantic Analysis (pLSA) [34]. pLSA enable to estimate the degree of how much a service execution satisfies all the other participants in the same workspace.

The probability that a user u_i satisfied to a degree $r \in \aleph$ after using service s_j can be defined as follows.

$$P(r_{i,j} > c|s_j, u_i) = \sum_{z \in Z} P(r_{i,j} > c|z, s_j) P(z|u_i) \quad (1)$$

where u_i is a service user, s_j is an available service, c is the minimum required level of the service satisfaction, r_i is the expected user u_i rating for a service s_j, and z is hidden space used for pLSA.

To simplify the user-service co-occurrence dependency structure, we use a conditional probability density $P(r|s,u)$ using a Gaussian model rather than $P(r|s_j, u_i)$ in Equation 1. Thus we introduce $\mu_{s,z} \in \Re$ for the mean service rating and $\sigma_{s,z} \in \Re$ for the spread of service ratings. We define $P(r|s,u)$ as

$$P(r|u,s) = \sum_{z \in Z} P(z|u) P(r; \mu_{s,z} \sigma_{s,z}) \quad (2)$$

where $p(r;\mu,\sigma) = \frac{1}{\sqrt{2\pi}\sigma} exp[-\frac{(v-mu)^2}{2\sigma^2}]$. To obtain $P(z|u)$, we employ the expectation maximization algorithm [36].

Finally, the expected probability of rating $r_i > c$ is calculated using Equation (2) as follows. Equation (3) is used to estimate the predictive behavior of services.

$$P(r_{u_i,s_j} > c) = \sum_{r > c} P(r|u,s) \quad (3)$$

In order to evaluate our model in Equation 1, we first run our model on MovieLens [37]. To simplify the experiments, we used five base-test set split data instead of a full data set. The split data consists of 80

The results are shown in figure 2. To evaluate our method, we define a prediction to be acceptable if the difference between our prediction and the actual rating is less than or equal to 1.0. That is, we assume that our prediction is acceptable if it is within a certain range, here between 0.0 and 1.0). The comparison is shown in figure 3. From the results in figure 2 and figure 3, our model shows reasonably good performance compared to the averaging method.

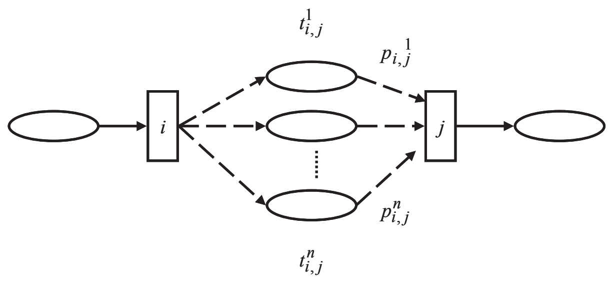

Fig. 4: A Net Based Model to analyze non-functional behavior

We also propose a methodology for analysis of non-functional behavior of services. To this end, we adopt a net-based quantitative analysis technique in a service oriented environment. Quantitative analysis enables one to trace, analyze and predict quantitative information, by examining its numerical, measurable characteristics such as performance factors.

The quantitative parameters for non-functional behavior are classified as performance parameters and reliability parameters defined as follows.

–*efficiency parameter* is the (average) time or cost to execute a given set of services
–*Reliability parameter* is the probability of entering and successfully executing a given service

The execution time or cost can be measured by monitoring services. The probability of unsuccessful services depends on the service consumer's criterion. A service consumer may define a service as a success whenever they get output regardless of the quality or whenever they experience a particular level of defects in the service. The probability can be assessed from past experience of services by testing the same services several times.

Figure 4 illustrates a net structure to be evaluated using quantitative analysis. We represent the target in our communicative model to be transformed into a Petri-Net based model, so any service in our approach can be represented in the same way as in figure 4. Oval objects connected to a rectangle object represent n service candidates available from an input specification object i. The notation $p^n_{i,j}$ over the dashed arrow indicates the probability that the service n can be executed successfully. Hence, $p^n_{i,j}$ represents service reliability. The notation $t^n_{i,j}$ represents performance parameters such as cost and execution time.

Provided there is no information about the probability distribution of any inputs, then the assumption of our model is that the probability $p^n_{i,j}$ of successful service execution is determined by the expected ratings from Equation (3). Obtaining $t^n_{i,j}$ is more complex. Assume that a service provider posts their particular service performance using a parameter T, we obtain $t^n_{i,j}$ using Equation (4)

$$t^n_{i,j} = T + delay = T + T \cdot \frac{\lambda e^{-\lambda(Er|s,u))}}{K} \qquad (4)$$

where $\lambda = \frac{1}{\sigma_{r,s}}$, K is constant, $E(r|s,u)$ is an expected service rating from a user u for a service s, and $\sigma_{r,s}$ is the standard deviation for the probability distribution of ratings r on a service s.

6 Implementation and Evaluation

6.1 Implementation

We built a framework to support trustworthy service oriented workspace sharing environment based on Java 6.1 and Apache Axis 2.2. Our framework is built as a standalone application and is composed of the following components.

–*Process Engine*'s primary purpose is to create, manage, and enact a process instance. The Process Engine instantiates the process after it loads up the definition. Once instantiated, a process is executed and monitored by the *Process Engine*.
–*Cockpit* is a communication interface connecting a user to the *Process Engine*. It couples a user and an engine by transmitting the user's decision on process creation and enactment to the *Process Engine*.
–*Message Interpreter* handles message generation based on our communicative model. *Message Interpreter* is connected *Process Engine* and deliver messages to *Service Connector*.
–*Service Connector* connects *Message Interpreter* with published Web Services. Messages generated from *Message Interpreter* is translated into SOAP format through *Service Connector*.

6.2 A Case Study

In this section, we discuss how our proposed model enables service oriented workspace sharing. To this end, we developed a case study of a die casting process for thermoelectric fan housing, from American steel founder's society [38]. The die casting process are composed of several sub tasks such as 'Design Requirements', 'Die Design', and 'Part Analysis', and 'Casting Product'. We assume each task is assigned to different workers and they share their workspace through service oriented environment. Worker A works for product design and is assigned 'Design Requirements'. B design Dies considering overflow vent and gates. C selects casting materials with numerical methods. D finally do casting works with product design and die design. Figure 5 illustrates a die casting collaborative

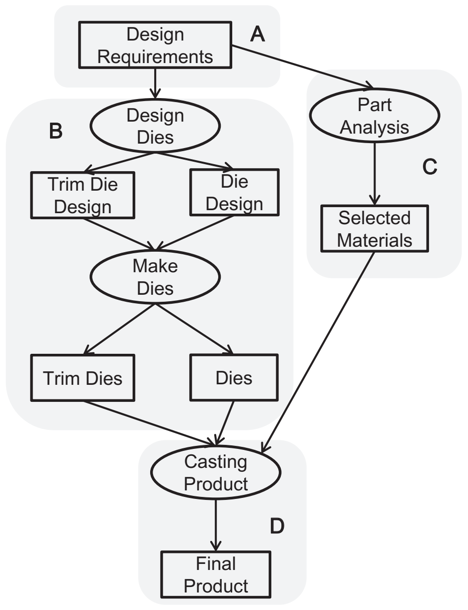

Fig. 5: A Die Casting Design Process

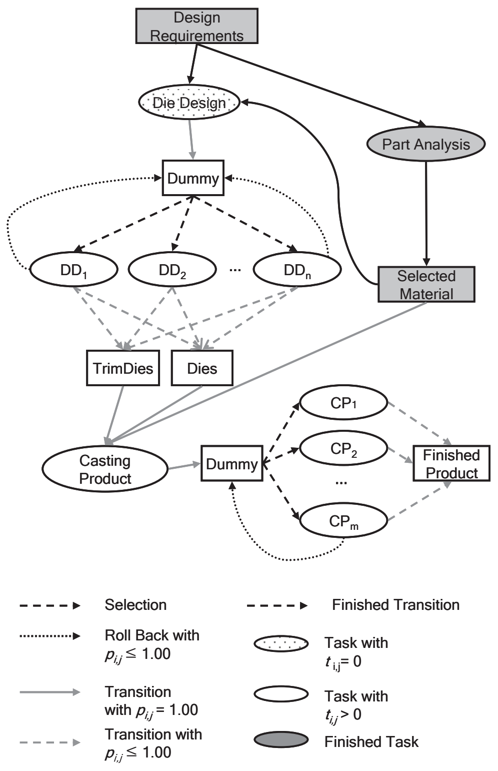

Fig. 6: A Die Casting Design Process

process. Based on figure 5, *B* and *C* can work concurrently while *D* should wait until *B* and *C* complete their tasks.

Suppose later that *D* found a problem on the final product as the final product fails in a mechanical stress testing. After a certain diagnosis, *D* concludes that the product's strength is not uniformly distributed and some parts of a product is too weak to pass the stress testing. Then *D* report the problem and diagnosis to other workers. Then they *B* founds the overflow design is wrong and some parts gets hot too early. In such case, *B* can notify the problem to *A* as the problem can be from wrong product design. So *B* should be able to notify redesign is needed to *A*, while canceling all the work done in 'Design Dies' task and 'Part Analysis' task. Our communicative model can support all the needed interactions on the service oriented environment. Below is a sequence of messages m_1, m_2, m_3, and m_4 from *B* and *A*. Through m_1 and m_2, *B* cancels 'Part Analysis' task and 'Design Dies' task. Using m_3, *B* asks redesign of the product to *A*. *A* answer yes with m_4.

$$m_1 = < declare, hasStatus(DesignDies), \\ finished \underset{<EnforcedRollback, DesignDies>}{\Longrightarrow} initial >$$

$$m_2 = < declare, hasStatus(PartAnalysis), \\ finished \underset{<EnforcedRollback, PartAnalysis>}{\Longrightarrow} ready >$$

$$m_3 =< propose, hasStatus(DesignRequirements), \\ finished \underset{<EnforcedRollback, DesignRequirements>}{\Longrightarrow} initial >$$

$$m_4 =< accept, hasStatus(DesignRequirements), \\ finished \underset{<EnforcedRollback, DesignRequirements>}{\Longrightarrow} initial >$$

6.3 Evaluation of Trustworthiness

In this section, we apply our trust model to the scenario depicted in figure 5. The prediction has three phases. First, we design service behaviors based on service capability published in the form of OWL or RDF. For the die casting process example, figure 5 present a service behavior. The next phase is to build the dynamic service behavior. In the second phase, new properties are added such as rollback and selection steps with transition probabilities and transition costs. Figure 6 illustrates the dynamic service behavior from the design of figure 5.

The available services are also associated with performance parameters based on our trust model. Since 'DieDesign' and 'CastingProduct' are about to be outsourced, their performance values are set to 0.0 and their tasks are filled with dots as a placeholder for the future relationship. The grey data and task objects indicate the completed task. In figure 6, we are in the middle of execution of the whole process, where the task ‘PartAnalysis’ is finished and the two tasks ‘DieDesign’ and ‘CastingProduct’ need to be associated with particular services. The reliabilities and performances can be obtained through our trust model presented in the section 5.

The third phase of our prediction is to estimate the trustworthiness of the process. It is also capable of offering analysis aimed at identifying the impact of various factors. To this end, we run simulations based on available choices by building the corresponding stochastic Petri net models. In order to evaluate the scenario in figure 5, we randomly chose 100 items for both DD_i and CP_i from MovieLens data to generate user satisfaction for each DD_i and CP_i. The average rating for user satisfaction is 3.41. We assume the service consumer satisfies the service outcome when the consumer’s rating is over 3.5. Failure ratings are set to 2.5. The failure rate indicates the expected probability that the service delivery brings a rating less than 2.5.

For evaluation, we identify three types of evaluation criteria: 1) the least failure rate (LF), 2) the best satisfaction (BS), and 3) the maximal quality (MQ). LF specifies the selection with the minimum failure probability, BS indicates the selection for the best rated service, and MQ gives the selection for maximum probability satisfying a certain rating. In order to simplify the evaluation, we assume that all the service providers post the same values for performance factors such as cost and delivery time. Table 4 shows the prediction results based on each criterion. Based on Table 4, selections may vary based on criteria. Table 5 illustrates the ranking differences between predictions and actual results. Ranking provide the ordered list of service recommendation. Therefore ranking correctness is also important as ranking affects selection results. From the samples tested, the overall ranking errors from our approach are measured significantly lower than general averaging methods. These results demonstrate that our trust methods are reasonably acceptable for predicting trustworthiness.

Table 4: Failure Rate and Quality Differences in various selection criteria

Criteria	**Failure Rate(%)**	**Quality Difference(%)**
BS	0.72	12
LF	0.68	15
MR	0.66	19

Table 5: Rank difference comparison on our method and averaging method

Criteria	**Rank Difference in Trust Model**	**Rank Difference in Averaging Method**
Failure Rate	2.4	4.1
Quality Rate	3.1	6.2

7 Conclusion

In this paper, we present a new framework providing workspace sharing for service-oriented computing. Our work focuses on issues of service trustworthiness through message semantics. It provides a new, solid foundation for reliable service-oriented computing for a collaborative workspace sharing environment. In particular, we developed a communicative model and a trust model. We, then, propose a methodology for predicting service behavior and hence enhancing system trustworthiness.

Based on the proposed methodology, service behavior is analyzed stochastically. Adopting our frame-based approach enables partners to be monitored and enacted so as to verify the service status as valid through purely semantic dialogue interpretation. Moreover, our proposed approach is designed in a platform neutral way, retaining the spirit of a service-oriented paradigm. We believe that our approach will satisfy the need of users to control ability and customization availability in Web Services.

Our implementation is based on a standalone server-application. Thus, collaboration based on service computing and its reliability can be realized by using our proposed framework. Moreover, our proposed approach can also be integrated into current legacy systems or existing Web Service based standards, as we have proposed platform neutral and message oriented data formats. Since our messages follow the SOAP format and since the semantics based on OWL are integrated into messages, software modules with any inference engine can be implemented to support the proposed method. In addition, we provide an evaluation based on the example of a design and manufacturing domain. Although we conducted experiments on the artificial data sets, we believe that our work will contribute to the goal of realizing true service oriented collaboration over broad ranges of collaborative process domains.

Acknowledgement

This work has been supported by Dongguk University Research Fund DRIMS 2012.
The authors are grateful to the anonymous referee for a careful checking of the details and for helpful comments that improved this paper.

References

[1] M. K. Nair, S. M. Kakaraddi, K. M. Ramanarayan, V. Gopalakrishna, Proceedings of the 2009 IEEE International Conference on Services Computing, 528-531 (2009).
[2] R. Kubert, S. Wesner, Proceedings of the 2012 IEEE Ninth International Conference on Services Computing, 578-585 (2012)
[3] F. Baader and W. Nutt, The description logic handbook: theory, implementation, and applications, Cambridge Uniersity Press, 43-95 (2003).
[4] A. Davis, D. Zhang, Proceedings in the fourth international symposium on multimedia software engineering, 48-55 (2002).
[5] J. Huang, C. Lin, X. Kong, Y. Zhu, Proceedings of the 2011 IEEE International Conference on Services Computing, 184-191 (2011).
[6] X. Ye, M. Gao, Proceedings of the 2012 IEEE Ninth International Conference on Services Computing, 242-249, (2012).
[7] A. Lazovik, M. Aiello, and M. Papazoglou, Proceedings of the 2nd international conference on service oriented computing, 94-104 (2004).
[8] W. Ma, V. Tosic, B. Esfandiari, and B. Pargurek, Proceedings in international workshop on Business services networks, 7-7 (2005)
[9] L. J. Zhang and D. Anrdagna, Proceedings in the 2nd international conference on Service oriented computing, 173-182 (2004).
[10] L. Baresi, C. Ghezzi, and S. Guinea, Proceedings in the 2nd international conference on service oriented computing, 193-202 (2004).
[11] V. K. Shanbhag Proceedings of the 8th international SPIN workshop on Model checking of software, 252-271 (2001).
[12] R. Baldwin and M. J. Chung, IEEE Computer, 54-63, 1995.
[13] M. J. Chung, P. Kwon, and B. Pentland, ASME Transaction Journal of Mechanical Design, **124**, 364-374 (2002).
[14] S. Banerjee, S. Basu, S. Garg, S. Garg, S. Lee, P. Mullan, P. Sharma, Proceedings of the 3rd international workshop on Middleware for grid computing, 1-6 (2009).
[15] A. Artale, E. Franconi, Proceedings of the sixth international workshop on Temporal Representation and Reasoning, 2-5 (1999)
[16] D. Kong, Y. Zhai, Proceedings of the 2012 International Conference on Cloud and Service Computing, 176-179 (2012).
[17] Z. Malik, A. Bouguettaya, The VLDB, **18**, 885-911 (2009).
[18] J. Yao, W. Tan, S. Nepal, S. Chen, J. Zhang, D. DeRoure, C. Goble, Proceedings of the 2012 IEEE Ninth International Conference on Services Computing, 454-461 (2012).
[19] A. Hamalainen, P. Jappinen, J. Porras, Proceedings of the Fourth International ICST Conference on Communication System Software and Middleware, **20** (2009).
[20] K. Yang, A. Galis, H. Chen, Mobile Networks and Applications, 15, 488-510 (2010).
[21] T. Hofmann, Proceedings in the 22nd annual international ACM SIGIR conference on Research and development in information retrieval, 50-57 (1999).
[22] W. Liu, T. Pang, H. Qu, Proceedings of the Management and Service Science, 1-3 (2009).
[23] A. Lazovik, M. Aiello, and M. Papazoglou, Proceedings in the 2nd international conference on service oriented computing, 94-104, (2004).
[24] L. Baresi, C. Ghezzi, and S. Guinea, Proceedings in the 2nd international conference on service oriented computing, 193-202 (2004).
[25] M. D. Sadek, Proceedings in ESCA/ETRW Workshop on the Structure of Multimedia Dialogue, 1-29 (1999).
[26] R. G. Smith, IEEE Transactions on Computers, **C-29**, 1104-1113 (1981).
[27] W. J. v. d. Heuvel and Z. Maamar, Communications of the ACM, **46** (2003).
[28] T. Finin, R. Fritzson, D. McKay, and R. McEntire, Proceedings in the third international conference on Information and knowledge management 465-463 (1994).
[29] H. Weigand and W. J. v. d. Heuval, International Journal of Electronic Commerce, **2**, 45-66, (1999).
[30] M. J. Wooldridge, IEEE Proceedings Software Engtineering, **144**, 26-37 (1997).
[31] Y. Labrou and T. Finin, LNCS, **1365**, 209-214 (1998).
[32] W. Kim and M. J. Chung, Proceedings in the 6th international conference on Web Information System Engineering, 217-230 (2005).
[33] J. Shearle, Syntax and Semantics: Speech Acts, **3**, 59-82 (1975).
[34] T. Hofmann, ACM Transactions of Information Systems, **22**, 89-115 (2004).
[35] K. Bhar, C. Fournet, and A. D. Gordon, Proceedings in the 11th ACM conference on Computer and communication security, 268-277 (2004).
[36] H.C. Kim, Z. Ghahramani, IEEE Transactions on Pattern Analysis and Machine Intelligence, **28**, 12, 1948-1959 (2003).
[37] B. N. Miller, I.vAlbert, S. K. Lam, J. Konstan, J. Riedl, Proceedings of the 8th international conference on Intelligent user interfaces, 263-266 (2003).
[38] M. J. Chung, P. Kwon, B. Pentland, and S. Kim, Proceedings in 2002 ASME International Mechanical Engineering Congress & Exposition, 33-42 (2002).

Woongsup Kim received the B.S. degree in Computer Engineering from Seoul National University in 1998, M.S.E. degree in Computer and Information Science from University of Pennsylvania in 2001 and Ph.D. degree in Computer Science from Michigan State University in 2006, respectively. Since 2007, he has been an Assistant Professor of Department of information and communication engineering at Dongguk Univeresity. Research interests include software engineering, semantic web, Service oriented architecture, and cloud computing.