Trusted Interaction Patterns in Large-scale Enterprise Service Networks

Florian Skopik, Daniel Schall, Schahram Dustdar
Distributed Systems Group
Vienna University of Technology
Argentinierstraße 8/184-1, A-1040 Vienna, Austria
{skopik|schall|dustdar}@infosys.tuwien.ac.at

***Abstract*—The evolution towards cross-organizational collaboration and interaction patterns has led to the emergence of scalable, Web services-based composition infrastructures. The success of service-oriented architecture (SOA) was mainly influenced by the standardization of composition languages such as BPEL. However, compositions require humans to be in the loop and ways to interface with people in a service-oriented manner. In this paper, we discuss Human-Provided Services (HPS) enabling the seamless integration of human capabilities in SOA. In complex and large-scale environments, processes might span interactions among partially unknown participants residing in different organizational units. To address the problem of trusted selection of participants, we introduce a mining approach for the automatic inference of trust relations. Unlike a security-based view on trust, our approach relates to the emergence of trust across humans and services from a social perspective.**

***Keywords*-interaction patterns; trust; mixed systems; human involvement in SOA; online help and support**

I. Introduction

Collaborations on the Web and in large-scale enterprises evolve in a rapid pace by allowing people to form communities and expertise clusters depending on their skills and interests. The management of interactions in these networks becomes increasingly complex as current tools only support messaging and addressing mechanisms developed for the early Web.

However, it is difficult - if not impossible - to control interactions ranging from ad-hoc to process-centric collaborations. In Web-scale networks, partially unknown participants might be part of processes that span multiple geographically distributed units of an organization. We argue that trusted selection of participants leads to more efficient cooperation and compositions of human- and software services. Trust can be discussed from a security perspective. In this work we follow another view that is related to how much humans or other systems can rely on software systems to accomplish their tasks [1]. We believe that trust and reputation mechanisms are key to the success of open dynamic service-oriented environments. However, trust between human and software services is emerging based on interactions. Interactions, for example, may be categorized in terms of success (e.g., failed or finished). Therefore, an important aspect of our approach is the monitoring and analysis of interactions to automatically determine trust in service-based systems. Our concepts, including trust inference and reputation management, are described in the context of an enterprise use case. Trust is built upon previous interactions and evolves over time; thus providing a reliable way to automatically compute trust. We discuss the implementation of our framework and its application in enterprise environments.

Our key contributions include: (i) We introduce trust in the context of a *mixed system* comprising services provided by human actors – Human-Provided Services (HPS, see [2]) – and 'traditional' software services. We extend our previously introduced trust model [3] and utilize it for trusted expert selection. On the one hand we demonstrate how trust influences interactions in mixed service networks; on the other hand we show the impact of interactions on trust. (ii) Coordination and compositions are strongly influenced by interaction patterns. We propose models for the propagation of trust based on delegations resulting in different types of patterns. We consider (a) Trust *referral* in triad interaction patterns, and (b) Trust *attenuation* in proxy patterns.

The main focus of this work is to present above mentioned interactions patterns, an approach for trust propagation within such patterns, and the discussion of the *TrueExpert* framework and its implementation. A detailed mathematical model on the calculation of trust propagation techniques is not within the scope of this paper.

The remainder of the paper is organized as follows. We motivate our work and introduce fundamental requirements on a system managing trust in an enterprise environment in Section II. Section III deals with the building blocks of our approach. The fundamental interaction model, as well as interaction patterns are described in Section IV. We depict an overview of the the whole framework and some implementation details in Section V. Section VI lists related work. We conclude our work with future perspectives in Section VII.

II. Coordination and Composition

A motivating use case for our work is depicted in Figure 1. A process composed of single tasks assigned to humans or covered by software services, describes the steps to produce a Computer Aided Design (CAD) drawing, and handle its final delivery to a customer. A conceptual draft, e.g., of a mechanical part, is designed by an engineer from the development department. In parallel a CAD assistant, belonging to another organizational unit, defines common symbols for the final drawing which conform to customer's requirements and international standards. After these tasks completed, a skilled drawer produces the drawing. In the

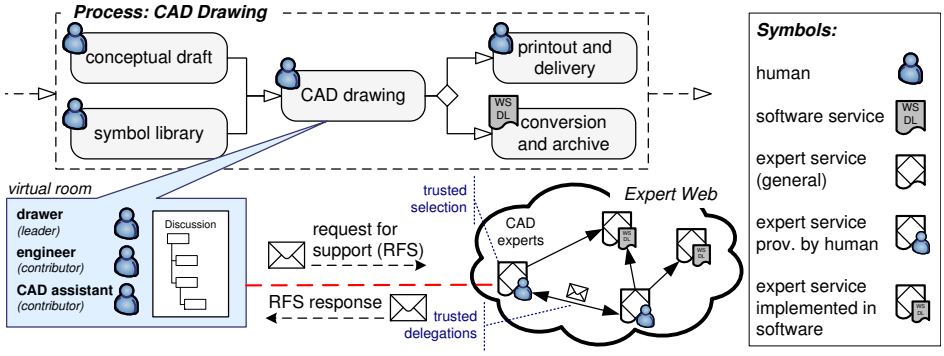

Figure 1. Involving experts from the expert web.

last step an assistant cares for printout and delivery to the customer, and a software service converts the drawing and stores it in an archive.

We assume, that the single task owners in this process exchange only electronic files, and interact using communication technologies. While various languages and techniques for modeling such processes already exist, including the Business Process Execution Language (BPEL [4]), we focus on another aspect in this scenario: *trusted online help and support*. Usually, in BPEL input and output data are rigidly specified, however, even for carefully planned processes with human participation, ad-hoc adaptation and intervention is required due to the complexity of human tasks, people's individual understanding, and unpredictable events. For instance, according to Figure 1 the drawer receives a drawing draft and a symbol library. If people have not yet worked jointly on similar tasks, it is likely, that they need to set up a meeting for discussing produced artifacts. Especially, if people belong to different, possibly geographically distributed organizational units, a personal meeting can be time- and cost intensive. Therefore, various Web 2.0 technologies, including forums, wiki pages and text chats, provide well-proven support for tele-work in collaborative environments (represented by the *virtual room* in Figure 1).

However, several challenges remain unsolved. If people, participating in the whole process, are not able to solve problems by discussion, who should be asked for support? How can third parties be contacted and informed about the current situation? How can they easily be involved in ongoing collaborations? Moreover, what are influencing factors for favoring one party over others? How is information exchanged, and how can this situation be supported by service-oriented systems?

The traditional way of discovering support is simply to ask third persons in someone's working environment, the discussion participants are convinced they are able to help, namely *trusted experts*. In an environment with a limited number of people, persons usually tend to know who can be trusted and what data has to be shared in order to proceed with solving problems of particular nature. Furthermore, they easily find ways to contact trusted experts, e.g., phone numbers or e-mail addresses. In case requesters do not know skilled persons, they may ask friends or colleagues, who faced similar problems before, to recommend experts. The drawbacks of this traditional way are that people need extensive knowledge about the skills of colleagues and internal structures of the organization (e.g., the expertises of people in other departments). The traditional way of discovering support is inefficient in large-scale enterprises with thousands of employees and probably not satisfying if an inquiry for an expert becomes a major undertaking. Even the use of today's computer-supported communication technologies cannot fully address the mentioned challenges.

The expert web. We propose the expert web, consisting of connected experts that provide help and support in a service-oriented manner. The members of this expert web are either *humans*, such as company employees offering help as online support services, or *software services*, encapsulating howtos[1], knowledge bases, and oracles[2] with intelligent reasoning capabilities. Such an enterprise service network, spanning various organizational units, can be consulted for efficient discovery of available support. Users, such as the engineer or drawer in our use case, send **r**equests **f**or **s**upport (RFSs). The users establish trust in experts' capabilities based on their response behavior (e.g., availability, response time, quality of support). This trust, reflecting personal positive or negative experiences, fundamentally influences future selections of experts.

As in the traditional sense, experts may also delegate RFSs to other experts in the network, for example when they are overloaded or not able to provide satisfying responses. Following this way, not only users of the enterprise service network establish trust in experts, but also trust relations between experts emerge.

III. Supporting Concepts

A. Human-Provided Services

Currently, there is very limited support to enable human interactions and participation in service-oriented systems. While specifications such as WS-HumanTask [5] and BPEL4People [6] define the concepts needed to model human interactions in BPEL processes, they do not address how users can create services, register their capabilities in SOA, and interact in a seamless manner using Web services technologies. The very idea of HPS is to support

[1]http://www.ehow.com
[2]http://www.wolframalpha.com

humans in offering their skills and capabilities as services (e.g., a "reviewing service" provided by one or more human actors). For example, human activities – independent of any particular process model – can be defined by the end-user and are mapped onto Web services.

HPSs act as interaction interfaces toward humans, letting users define various HPSs for different collaborative activities indicating their ability (and willingness) to participate in ad-hoc as well as process-centric collaborations. The users can manage their interactions, which might span various platforms and services.

In our previous work [2], we introduced a framework with the following features:

- A *Service registry* that does not only hold information regarding software services but also human-based information such as skill- and user profiles, preferences, and shared user context (e.g., availability and location information).
- *Interaction support* based on the notion of activities – to structure and manage interactions. Various artifacts, users, and resources can be managed by using an hierarchical activity model. Also, activities determine in which context interactions take place.
- Tools for creating a *user-defined activity model* serving as input for the definition of HPS related artifacts (e.g., WSDL and XML types).
- *Dynamic expert profiles* are created through mining of logged interactions.

B. Trust Inference Model

Managing trust in the enterprise service network, means determining relations between its human- and software service participants. Therefore, we define a directed graph $G_T = (V, E)$, where the vertices V represent expert services as well as the users of the services, and multiple edges E between them reflect their relations for different situations (i.e., in distinct scopes s described by context elements, such as the current project and problem domain). Each edge $e = (u, v, c, \rho, s)$, $e \in E$ describes the level of confidence $c = [0, 1]$ u has in v's support, and the reliability $\rho = [0, 1]$ of c ranging from totally uncertain to fully confirmed (similar to [7]). The scope s describes situations for applying c and ρ. Confidence and reliability values are recovered from interactions by monitoring, analyzing, and interpreting service interaction behavior (e.g., with respect to pre-negotiated SLAs) (see [8]).

Personal Trust. In our model, confidence c^s depends on the success of analyzed interactions and rewarding of services. If an expert service v responds reliably and dependably to u's RFSs, and u is satisfied with the answer, the relationship from u to v gets rewarded, and $c^s(u, v)$ increases in the given problem scope s. However, if RFSs are not sufficiently answered, responses arrive late or not at all, $c^s(u, v)$ decreases. An algorithm dealing with the realization of this model, also considering the evolvement over time, is presented in [3]. The reliability ρ of c^s depends fundamentally on the number of recent interactions, i.e., requests and responses. Usually, a predefined number of requests in a recent time window has to be analyzed to identify a trend in support quality. Finally, we calculate trust τ^s as shown in Equation 1.

$$\tau^s(u, v) = c^s(u, v) \cdot \rho(c^s(u, v)) \tag{1}$$

Recommendation τ^s_{rec} is built by the combined personal trust relations of neighbors in a third party service in the trust network (see Figure 2). This concept is realized by the means of trust propagation [9], where personal trust relationships are inherited from one or several intermediaries [10]. In the example in Figure 2, $\tau^{s_1}_{rec}(u, v)$ is determined by the relations of u's direct neighbors $\{w, y\}$ to v in scope s_1. Therefore, recommendation $\tau^{s_1}_{rec}(u, v)$ is calculated by averaging $\tau^{s_1}(w, v)$ and $\tau^{s_1}(y, v)$. Normally, the impact of single trust relations is weighted considering u's relations to recommenders ($\tau^{s_1}(u, w)$, $\tau^{s_1}(u, y)$).

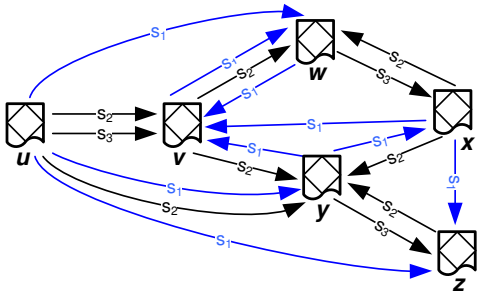

Figure 2. A network comprising trust relations in distinct scopes.

Reputation τ^s_{rep} reflects the trust in an expert from a global view. For instance, in the network in Figure 2, $\tau^{s_1}_{rep}(v)$ is built by calculating a weighted average of the trust values of v's incoming edges in scope s_1, therefore, utilizing relations from neighbor vertices $\{w, x, y\}$.

IV. Interaction Patterns and Trust

According to the motivating example in Figure 1, leaders of tasks in a process can invite participants of preceding tasks to a discussion in a virtual room, using Web 2.0 technologies such as discussion forums. If people are not able to find solutions for occurring problems, the discussion leader may consult expert services. These services are selected by considering the discussion leader's trust requirements (e.g., a minimum personal trust or reputation) in expert services.

A. Service Selection Procedure

We distinguish between two roles: *requesters* and *expert services*. Requesters, i.e., service users, are humans requesting support; expert services are provided by either humans (HPS) or are implemented in software. A human can be both a service requester and provider at the same time.

Algorithm 1 implements the procedure for selecting a service from the expert web. At first, it is determined if a scope s that sufficiently describes the current situation already exists (i.e., context similarity is above a predefined threshold ϑ_{Ctx}). Otherwise, a new scope is created

Algorithm 1 Create RFS of user u to request support from service v

Require: network G_T, user u, $problem$ descr., trust $policy$

```
/* determine s, create support activity */
Scope[] ← getAvailableContextualScopes(G_T)
EnvDescr ← collectContextElements(u);
s ← maxSimilarity(Scope[], EnvDescr)
if (sim(EnvDescr, s) < ϑ_Ctx) ∨ (s = ∅) then
    s ← createProblemScope(EnvDescr)
    addScopeToTrustNetwork(G_T, s)
end if
p ← createProblemDefinition(u, problem);
a ← createSupportActivity(p, s, policy)

/* discover list of potential services */
if penetration(G_T, s) ≥ ϑ_p then
    /* discovery based on trust requirements */
    v[] ← evalTrustRequirements(u, V, s)
else
    /* competency coverage fallback */
    v[] ← evalCompetencyCoverage(V, a)
end if
v ← selectService(v[], u.rules)

RFS ← createRequest(u, v, a)
return RFS
```

using available data from the environment. This step is performed by the support requester who specifies context data that describes the scope, such as the type of the current problem. Afterward, a support activity is created. Then the algorithm determines if there is a sufficient number of trust relations applying to s in G_T, to reliably calculate recommendations and reputation of services. If this *penetration* of s in G_T is greater than a threshold ϑ_p, then the expert services $v[\,]$ are discovered by evaluating predefined trust requirements specified by the requester. Otherwise, if $penetration(s) < \vartheta_p$, a fallback strategy is applied and the service providers' skill profiles are compared with the problem description and requirements in s (traditional skill matching). From the pool of services that fulfill the requester's trust requirements, the system picks one based on pre-defined selection rules. Finally, the request for support (RFS) is compiled. The definitions of trust requirements and service selection rules are presented in Section V.

B. Fundamental Interactions

The fundamental interactions when requesting support are depicted in Figure 3. First, the requester sends an RFS, specifying the problem and policies for treating the request. Second, the expert service sends an *intermediate answer*, informing the requester that the RFS has been accepted, rejected or delegated to other members in the expert web. Third, after fully processing the RFS, the *final answer*, including a solution for the requester's problem, is delivered. All performed interactions are monitored and evaluated by the system. The supporting expert service gets rewarded depending on the success and outcome of interactions, and a trust relation from the requester to the service is established.

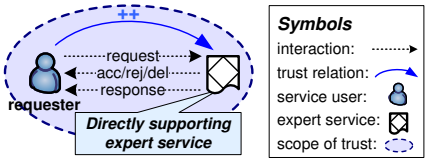

Figure 3. From interactions to trust.

C. Rewarding and Punishment

We utilize occurring interactions to infer personal trust. Therefore, we introduce the following methods for rewarding an expert's impact on ongoing discussions: (i) *automatic:* Availability of services and response behavior (e.g., rejecting RFSs) are determined through the means of interaction mining. (ii) *semi-automatic:* In the RFS the requester can specify the importance and a hard deadline for a response. Depending on whether an answer arrives in time, trust either increases or decreases. (iii) *manual:* The discussion leader may grade a service provider's support manually (e.g., 4 of 5 stars).

New confidence values are calculated in fixed time intervals i, e.g., on a weekly basis. Based on earned rewards, the current confidence value at time i is calculated by updating the recent value at $i-1$. For all kinds of rewarding we apply the concept of exponential moving average (EMA) to smoothen the sequence of rewards as shown in Equation 2. The variable rew represents the reward, given automatically by the system, or manually by the user, for support in the last time interval; rew_{max} is the maximum possible amount (if all RFSs are served reliably). EMA weights the importance of recent rewards while not discarding older ones (smoothing factor $\alpha \in [0,1]$). Because we maintain relative levels of confidence and trust ($\in [0,1]$), services can be punished for bad support by giving comparatively low rewards, i.e., $rew << rew_{max}$.

$$c_i^s = \alpha \cdot \frac{rew}{rew_{max}} + (1-\alpha) \cdot c_{i-1}^s \qquad (2)$$

D. RFS Delegation Patterns

Expert services need not process all RFSs directly, but may delegate them to other expert services due to various reasons. For instance, an expert may be overloaded and therefore, not be able to process an RFS in time. Moreover, expert services may shield other services from RFSs, e.g., only a team leader receives RFSs directly that s/he delegates then to team members. We describe the influence on trust emergence for two different types of delegation patterns:

- *Triad interaction pattern* leading to *trust referral*.
- *Proxy pattern* leading to *trust attenuation*.

Triad Interaction Pattern. We introduce a triad pattern (Figure 4) realizing delegations of RFSs within the same contextual or organizational scope, e.g., between experts in the same knowledge domain (of course, one expert may be 'located' in several scopes). A *triad proxy* receives an RFS and forwards it to one of its well-trusted services. In case of complex problems, an RFS can be split into

sub-requests to reduce response time. Furthermore, the complete RFS can be delegated to more than one expert service to increase reliability, i.e., the chance to get a suitable response. Final responses are not handled by the triad proxy. As all participating entities in this pattern belong to the same scope, e.g., knowledge domain, the expert service(s) may respond directly to the requester. A typical use case is load balancing in teams of people with same roles.

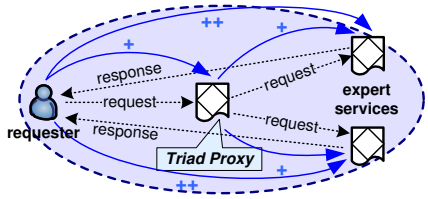

Figure 4. Triad interaction pattern.

From the requester's point of view, the triad proxy receives reduced rewards (Symbol +) for delegating but not processing the RFS. The actually supporting expert services receive rewards from the triad proxy, because of accepting the delegated RFS. This reward is also reduced, because the originator of the RFS is not the triad proxy, and the triad proxy has reduced interest in successfully processing the request (compared to one of his own RFSs). However, the requester honors the support provided by the actually supporting expert service(s) equally compared to directly supporting services (compare Fig 3, symbol ++). Therefore, we understand highly weighted trust relations from the requester to the initial expert service, acting as a triad proxy, to be *referred* to the actual expert services.

Proxy- and Master-Slave Pattern. We adopt the well-known proxy- and master-slave patterns from the domain of software engineering [11], as applied in the domain of business interactions by [12]. In contrast to the triad pattern, the initial requester does not know the expert services an RFS is delegated to. Furthermore, the proxy may perform certain pre-processing of the RFS. The proxy pattern (forward RFS to only one expert service) and master-slave pattern (split RFS and forward to several 'slaves') are used for delegations across contextual or organizational scopes. For instance, the requester in Figure 5 sends an RFS to the proxy residing in the same scope. This proxy can rephrase an RFS ('translate') to be understood by an expert in another scope. The response of the expert service is processed again, before forwarding it to the initial requester. A typical example may be a head of department who acts as contact person for external people, while the actual members are not visible to the outside.

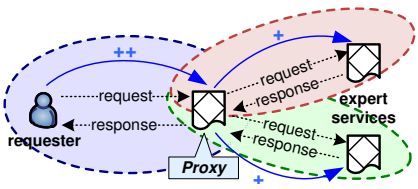

Figure 5. Proxy and master-slave patterns.

In contrast to the triad pattern, no trust relations from the requester to the actually supporting expert services across scopes are established. Therefore, the requester highly rewards the proxy, similar as directly supporting services. However, because the originator of the RFS is not the proxy (therefore, being less dependent on the response of the expert services), rewards given from the proxy to the expert services are reduced (equally to the behavior of the triad proxy). This leads to a trust *attenuation* from the expert service's point of view. Furthermore, the number of people or services building trust in the expert service(s) is smaller in this pattern. This reduced visibility of services' contributions has negative impact on their reputation.

V. TrueExpert Architecture

We depict the overview of the centralized TrueExpert architecture in Figure 6. The block on the left side contains common services from activity-centric collaboration systems. On the right side, the TrueExpert services are shown. The lower layer comprises of services supporting fundamental concepts, including data access, message routing and interaction logging; on higher level services for trust management, RFS creation, and expert ranking are located. These services are utilized via SOAP- and REST Web service interfaces from a user portal, implemented as Java Portlets[3] on top of the Liferay[4] enterprise portal.

We outline exemplary some implementation details, focusing the realization of the introduced trust concepts, including the RFS model, trust requirements for service discovery and selection, and rules for RFS flow control.

A. Trust Requirements Rules

The trust network is implemented as directed graph using the JUNG[5] framework. In this graph model, the edges are annotated with trust metrics, i.e., personal trust, recommendation, reputation and their reliability, and references to collections of contextual information (scopes) describe situations for applying them.

```
global TrustGraphDAO tgDAO;
rule "calculate score of services (without reliability)"
 salience 100
 no-loop true
  when
   service:Service()  // all potential services
   user:User()        // myself
   scope:Scope()      // my current problem scope
  then
   URI scopeId =  scope.getURI();
   URI userId = user.getUserURI();
   URI serviceId = service.getServiceURI();
   int trust = tgDAO.getPersonalTrust(userId, serviceId, scopeId);
   int rel = tgDAO.getTrustReliability(userId, serviceId, scopeId);
   int rec = tgDAO.getRecommendation(userId, serviceId, scopeId);
   int rep = tgDAO.getReputation(serviceId, scopeId);
   int score = 0.5*trust+0.3*rec+0.2*rep; // personal weighting
   service.getPersonalization().setMetric("trust",trust);
   service.getPersonalization().setMetric("rel",rel);
   service.getPersonalization().setMetric("rep",rep);
   service.getPersonalization().setMetric("myScore",score);
end
```

Listing 1. Personalized trust aggregation.

[3]http://jcp.org/aboutJava/communityprocess/final/jsr168/
[4]http://www.liferay.com
[5]http://jung.sourceforge.net

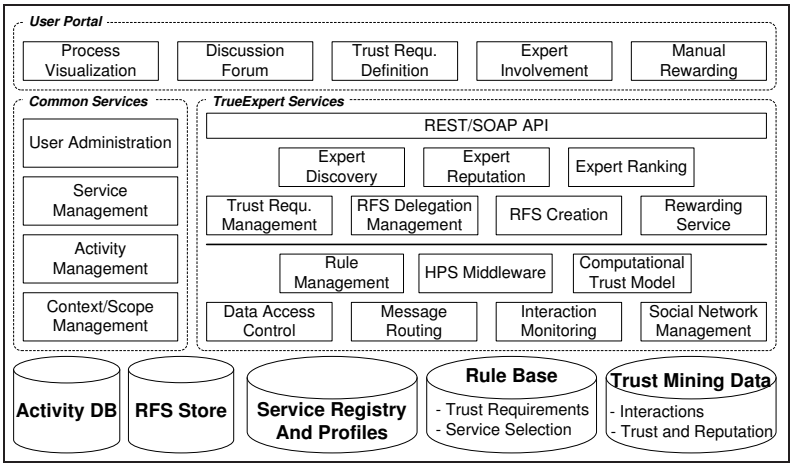

Figure 6. System architecture enabling trusted help and support in the expert web.

We use the popular *Drools*[6] engine, to let users define their own trust requirements on potentially supporting expert services. Listing 1 shows an example of aggregating trust data about services, i.e., combine personal trust, recommendation and reputation values to `myScore`, which can be used for subsequent service discovery operations. Before the depicted rules are applied, a user looking for support has to provide his/her own `user` profile and his/her current problem domain (`scope`), e.g., in form of activities [3]. Furthermore, copies of all available `service` profiles are loaded in the working memory of the rule engine. After evaluation, each service profile is temporarily personalized for the user.

Listing 2 shows some example rules, evaluating a user's trust requirements based on the personalized service profiles. Each service fulfilling at least one rule becomes a potentially supporting expert service. The `selected`-flag indicates that a service is applicable in the given scope (and for the given problem) from the user's point of view.

```
rule "Select service by average score"
 salience 50
  when
   service:Service(personalization.getMetric("myScore" >= 0.85)
  then
   service.getPersonalization().setSelected(true);
end
rule "Select personally trusted services"
 salience 50
  when
   service:Service(personalization.getMetric("trust") > 0.7 &&
             personalization.getMetric("rel" > 0.5)
  then
   service.getPersonalization().setSelected(true);
end
```

Listing 2. Discover services upon requirements.

B. RFS Routing

Different system strategies for selecting one expert service from the pool of discovered services that cover a user's trust requirements, can be realized with selection rules (Listing 3). For instance, in case of urgent requests, the system picks services with low workload (enabling load balancing); however, for supporting risky tasks, a service with high reputation is selected.

[6]http://www.jboss.org/drools/

```
rule "Urgent RFS"
  salience 50
  when
   rfs:RFS(eval(policy.flag.urgent))
   serviceList:List()  // all services fulfilling trust requirements
  then
   Service service = getServiceLWL(serviceList, 3);
   rfs.assignService(service);
end
rule "Risky RFS"
  salience 50
  when
   rfs:RFS(eval(policy.flag.risky))
   serviceList:List()  // all services fulfilling trust requirements
  then
   Service service = getServiceHR(serviceList);
   rfs.assignService(service);
end

// get services with low work load (below numRFS in queue)
function Service getServiceLWL(List serviceList, numRFS) {...}
// get service with highest reputation
function Service getServiceHR(List serviceList) {...}
```

Listing 3. Rules for service selection.

An excerpt of the RFS schema definitions is shown in Listings 4, defining complex data structures, and Listing 5, defining the binding of the HPS WSDL to the (HPS) infrastructure services (e.g., document review services).

- The `GenericResource` defines common attributes and metadata associated with resources such as documents or policies. A `GenericResource` can encapsulate remote resources that are hosted by a collaboration infrastructure (e.g., document management).
- The RFS `Policy` plays an important role for controlling interaction flows, e.g., time constraints, delegation behavior including decisions whether to respond to the requester directly or to a delegating proxy, and so on.
- `Request` defines the structure of an RFS (here we show a simplified example). From the user's point of view XML Forms (XForms[7]) are used to render graphical user interfaces.
- A `Reply` is the corresponding RFS response (we omitted the actual XML defintion).

[7]http://www.w3.org/MarkUp/Forms/

```
<xsd:schema tns="http://myhps.org/rfs">
 <xsd:complexType name="GenericResource">
  <xsd:sequence>
   <xsd:element name="Location" type="xsd:anyURI"/>
   <xsd:element name="Expires" type="xsd:dateTime"/>
  <xsd:sequence>
 </xsd:complexType>
 <xsd:complexType name="Request">
  <xsd:sequence>
    <xsd:element name="Policy" type="GenericResource"/>
    <xsd:element name="ReviewDoc" type="GenericResource"/>
    <xsd:element name="Comments" type="xsd:string"/>
  </xsd:sequence>
 </xsd:complexType>
 <xsd:element name="ReviewRequest" type="Request"/>
 <xsd:element name="AckReviewRequest" type="xsd:string"/>
 <xsd:element name="GetReviewReply" type="xsd:string"/>
 <xsd:element name="ReviewReply" type="Reply"/>
</xsd:schema>
```

Listing 4. RFS schema definition.

```
<wsdl:portType name="HPSReviewPortType">
 <wsdl:operation name="GetReview">
  <wsdl:input xmlns="http://www.w3.org/2006/05/addressing/wsdl"
   message="GetReview" wsaw:Action="urn:GetReview">
  </wsdl:input>
  <wsdl:output message="AckReviewRequest" />
 </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="HALSOAPBinding" type="HPSRFSPortType">
 <soap:binding style="document"
 transport="http://xmlsoap.org/soap/http"/>
</wsdl:binding>
```

Listing 5. WSDL RFS binding.

The protocol (at the technical middleware level) is asynchronous allowing RFSs to be stored, retrieved, and processed. For that purpose we implemented a middleware service (HPS Access Layer - HAL) which dispatches and routes RFSs. In Listing 5, `GetReview` depicts a WSDL message corresponding to the RFS `ReviewRequest`. Upon receiving such a request, HAL generates a session identifier contained in the output message `AckReviewRequest`. A notification is sent to the requester (assuming a callback destination or notification endpoint has been provided) to deliver RFS status updates for example; processed RFSs can be retrieved via `GetReviewReply`. Note, the detailed notification mechanism is not described in this paper that focuses on the realization of trustworthy interaction patterns. More about HPS in detail can be found in [2].

C. RFS Delegation

```
rule "Delegate RFS when overloaded"
  when
   rfs:RFS()
   service:MyService(rfsQueue.length > 10)
  then
   rfs.setResponse(RFSResponse.DELEGATED,
     new Message("I'm really busy, so I delegated your RFS."));
end
rule "Reject urgent RFS on Fridays"
  when
   rfs:RFS(eval(flag.urgent))
   service:MyService(workingproperties.lastDayOfWeek==Calendar.FRIDAY)
  then
   if(GregorianCalendar.getInstance().get(Calendar.DAY_OF_WEEK) == Calendar.FRIDAY)
    rfs.setResponse(RFSResponse.REJECTED,
              new Message("It's Friday, don't stress me out!"));
end
```

Listing 6. RFS acceptance and delegation rules.

We realize delegations by the means of ECA[8] rules. While events for applying rules are hard-coded, e.g. on receiving RFSs, conditions to be met and consequences can be flexibly configured by service providers (`MyService`). Listing 6 shows two example rules for automatically responding to RFSs. Requests are delegated (if possible) when the work load is high, i.e., there are more than 10 requests waiting to be processed. Furthermore, no urgent requests are accepted on Fridays. The definition of rules can be restricted for users based on contractual terms.

D. Rewarding Mechanisms

Based on interaction success and reliability of services, configured rules calculate rewards. These rules operate on top of interaction logs and look similar to the presented rules before. Our current implementation accounts for fundamental metrics as depicted in Table I. After evaluation of rewarding rules, the system updates confidence values and their reliability in the corresponding trust scopes. Besides these interaction metrics, we incorporate a manual reward provided by users. Both, automatic and manual rewards, are equally weighted and merged (averaged), to extend the dimensions of trust, and strengthen its information value for the users of the expert web.

Table I
METRICS UTILIZED FOR TRUST DETERMINATION.

metric name	range	unit	description
availability	[0,100]	%	ratio of replied to unreplied RFSs
responsiveness	[0,96]	hrs	average response time in hours
experience	[0,∞[	1	number of RFSs served
reciprocity	[-1,1]	1	ratio of provided to consumed help
man. reward	[0,5]	1	manually assigned scores

VI. RELATED WORK

The support of ad-hoc collaboration is a major challenge in cooperative information systems. The system in [13] is capable of supporting ad-hoc interactions in virtual teams. Based on log analysis, human interaction patterns can be extracted [12]. In [13], activity-centered computing and its patterns were introduced.

Major software vendors have been working on standards addressing the lack of human interaction support in service-oriented systems. WS-HumanTask [5] and Bpel4People [6] were released to address the emergent need for human interactions in business processes. These standards specify languages to model human interactions, the lifecycle of human tasks, and generic role models. Role-based access models (see [5] and [14]) are used to model responsibilities and potential task assignees in processes. In HPS, we also follow a Web services-based approach to support human interactions in a service-oriented manner. However, HPS and Bpel4People are complementary, and not competing approaches. HPSs are services that can be created by the end-user, whereas Bpel4People defines concepts (e.g., role model and logical people groups) to model interactions in BPEL-based application scenarios.

[8]Event-Condition-Action

Marsh [15] introduced trust as a computational concept, including a fundamental definition, a model and several related concepts impacting trust. Based on his work, various extended definitions and models have been developed. Some surveys of trust related to computer science have been performed [10], [16], [17], which outline common concepts of trust, clarify the terminology and describe the most popular models. From the many existing definitions of trust, those from [17], [18] describe that trust relies on previous interactions and collaboration encounters, which fits best to our environment. SOA-based infrastructures are typically distributed comprising a large number of available services and huge amounts of interaction logs. Therefore, trust in SOA has to be managed in an automatic manner. A trust management framework for service-oriented environments has been presented in [19], however, without considering particular application scenarios with human actors in SOA. Context dependent trust was investigated by [10], [15], [16], [17]. Context-aware computing focusing modeling and sensing of context can be found in [20], [21], [22].

VII. Conclusion and Outlook

In this paper we introduced concepts, centered around trust and Human-Provided Services in mixed service systems, and discussed their application in context of the *expert web* enterprise use case. We introduced models for rewarding interactions and establishing trust on top of delegation patterns that typically occur in real-world scenarios. Besides the detailed use case and application of trust in enterprise networks, we outlined the realization of our approach, focusing implementation details that apply ECA rules.

Currently, we are going to apply our software solution in the COIN[9] project. This project deals with research in supporting collaboration of people within virtual organizations built of SMEs. Our TrueExpert framework will be utilized by real end-users to allow more efficient adoption of available knowledge, spanning humans and services from various organizations and domains. Experiences and results collected during this empirical evaluation will be used to improve our underlying trust model.

Acknowledgment

This work is supported by the European Union through the IP project COIN (FP7-216256).

References

[1] M. Salehie and L. Tahvildari, "Self-Adaptive Software: Landscape and Research Challenges," *ACM Trans. on Autonomous and Adaptive Systems*, vol. 4, no. 2, May 2009.

[2] D. Schall, "Human Interactions in Mixed Systems - Architecture, Protocols, and Algorithms," Ph.D. dissertation, Vienna University of Technology, 2009.

[3] F. Skopik, D. Schall, and S. Dustdar, "The Cycle of Trust in Mixed Service-oriented Systems," in *Euromicro SEAA*, 2009, pp. 72–79.

[9] http://www.coin-ip.eu/

[4] OASIS, "Business Process Execution Language for Web Services, Version 2.0." 2007.

[5] M. Amend et al., "Web Services Human Task (WS-HumanTask), Version 1.0." 2007.

[6] A. Agrawal et al., "WS-BPEL Extension for People (BPEL4People), Version 1.0." 2007.

[7] T. D. Huynh, N. R. Jennings, and N. R. Shadbolt, "An Integrated Trust and Reputation Model for Open Multi-agent Systems," *Autonomous Agents and Multi-Agent Systems*, vol. 13, no. 2, pp. 119–154, 2006.

[8] F. Skopik, D. Schall, and S. Dustdar, "Trustworthy Interaction Balancing in Mixed Service-oriented Systems," in *ACM Symposium on Applied Computing*, 2010.

[9] C.-N. Ziegler and G. Lausen, "Propagation Models for Trust and Distrust in Social Networks," *Information Systems Frontiers*, vol. 7, no. 4-5, pp. 337–358, 2005.

[10] D. Artz and Y. Gil, "A Survey of Trust in Computer Science and the Semantic Web," *Journal of Web Semantics*, vol. 5, no. 2, pp. 58–71, 2007.

[11] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-oriented Software*. Boston, MA, USA: Addison-Wesley, 1995.

[12] S. Dustdar and T. Hoffmann, "Interaction Pattern Detection in Process-oriented Information Systems," *Data Knowl. Eng.*, vol. 62, no. 1, pp. 138–155, July 2007.

[13] P. Moody et al., "Business Activity Patterns: A New Model for Collaborative Business Applications," *IBM Systems Journal*, vol. 45, no. 4, pp. 683–694, 2006.

[14] J. Mendling, K. Ploesser, and M. Strembeck, "Specifying Separation of Duty Constraints in BPEL4People Processes," in *Business Inf. Systems*, 2008, pp. 273–284.

[15] S. P. Marsh, "Formalising Trust as a Computational Concept," Ph.D. dissertation, University of Stirling, April 1994.

[16] A. Jøsang, R. Ismail, and C. Boyd, "A Survey of Trust and Reputation Systems for Online Service Provision," *Decision Support Systems*, vol. 43, pp. 618–644, 2007.

[17] T. Grandison and M. Sloman, "A Survey of Trust in Internet Applications," *IEEE Communications Surveys and Tutorials*, vol. 3, no. 4, 2000.

[18] L. Mui, "Computational Models of Trust and Reputation: Agents, Evolutionary Games, and Social Networks," Ph.D. dissertation, MIT, December 2002.

[19] D. Kovac and D. Trcek, "Qualitative trust modeling in SOA," *Journal of Systems Architecture*, vol. 55, no. 4, pp. 255–263, 2009.

[20] G. D. Abowd, A. K. Dey et al., "Towards a Better Understanding of Context and Context-Awareness," in *International Symposium on Handheld and Ubiquitous Computing*, 1999, pp. 304–307.

[21] N. A. Bradley and M. D. Dunlop, "Toward a Multidisciplinary Model of Context to Support Context-aware Computing," *Human-Computer Interaction*, vol. 20, pp. 403–446, 2005.

[22] S. W. Loke, "Context-aware Artifacts: Two Development Approaches," *IEEE Pervasive Computing*, vol. 5, no. 2, pp. 48–53, 2006.