Formation and interaction patterns in social crowdsourcing environments

Daniel Schall

Distributed Systems Group,
Vienna University of Technology,
Argentinierstrasse 8/184-1,
A-1040 Vienna, Austria
E-mail: schall@infosys.tuwien.ac.at

Abstract: Over the past years, the web has transformed from a pool of statically linked information to a people-centric web. Various web-based tools and social services have become available enabling people to communicate, coordinate, and collaborate in a distributed manner. In this work, we consider social crowdsourcing environments that are based on the capabilities of human-provided services (HPS) and software-based services (SBS). Unlike traditional crowdsourcing platforms, we consider collaborative environments where people and software services jointly perform tasks. We propose formation and interaction patterns that are based on social principles. The idea of our social crowdsourcing approach is that interactions emerge dynamically at runtime based on social preferences among actors. The evolution of interactions is guided by a monitoring and control feedback cycle to recommend competitive compositions and to adjust interaction behaviour. Here we present fundamental formation patterns including link mesh and broker-based formations. We discuss the prototype implementation of our service-oriented crowdsourcing framework.

Keywords: crowdsourcing; social networks; formation patterns.

Reference to this paper should be made as follows: Schall, D. (2013) 'Formation and interaction patterns in social crowdsourcing environments', *Int. J. Communication Networks and Distributed Systems*, Vol. 11, No. 1, pp.42–58.

Biographical notes: Daniel Schall is a Postdoctoral Research Scientist at the Information Systems Institute, Vienna University of Technology. Prior to joining the Distributed Systems Group, he worked as a Technical Associate at Siemens Corporate Research (SCR) in Princeton, New Jersey, USA. In his research, he focuses on techniques to programme and control compositions in service-oriented systems that are strongly influenced by dynamic interactions between human-provided and software-based services (i.e., mixed service-oriented systems).

1 Introduction

Today's web applications facilitate interactive knowledge sharing, information exchange, user-centred content creation, and collaboration on the WWW. Even in business environments, Web 2.0 tools increasingly provide users the free choice to interact or collaborate with each other in virtual communities. The web becomes thereby a medium of interwoven human and service interactions.

These principles have also changed models for computing on the web by utilising human manpower through crowdsourcing platforms (Davis, 2011; Howe, 2008). Applications range from enterprise environments (Vukovic, 2009) to open internet-based platforms such as Amazon Mechanical Turk (Amazon, 2011) (MTurk). These online platforms distribute problem-solving tasks among a group of humans. Crowdsourcing follows the 'open world' assumption allowing humans to provide their capabilities to the platform by registering themselves as services. Some of the major challenges are monitoring of crowd capabilities, detection of missing capabilities, strategies to gather those capabilities, and tasks' status tracking (Brabham, 2008). There are two obstacles hampering the establishment of seamless communications and collaborations in service-oriented crowdsourcing environments:

1 the dynamic discovery and composition of resources and services

2 flexible and context-aware interactions between people residing in different (sub-)communities.

Here we address challenges related to human interactions in dynamic service-oriented systems. Service-oriented architecture (SOA) enables the design of applications that are composed from the capabilities of distributed software services. In SOA, compositions are based on web services following the loose coupling and dynamic discovery paradigm. Unlike traditional SOA-based approaches, we consider mixed service-oriented systems that are established upon the capabilities of human-provided and software-based services (SBS) (Schall, 2011). The integration of human capabilities in a service-oriented manner is motivated by the difficulties to adopt human expertise into software implementations. Instead of dispensing with human capabilities, people handle tasks behind traditional service interfaces. In contrast to process-centric flows (top-down compositions), we advocate flexible compositions wherein services can be added at any time exhibiting new behaviour properties.

In our previous work we introduced the building blocks for mixed service-oriented systems (also known as socially-enhanced services computing). The building blocks include: human-provided services (HPS) (Schall et al., 2008) and the HPS framework (Schall, 2011), metrics and a socially-aware reputation algorithm (Schall, 2012), trust inference techniques (Skopik et al., 2010) and behaviour-based network adaptation (Psaier et al., 2010). The idea of our previously performed work was to realise a *feedback cycle*; in a manner similar to the monitoring, analysing, planing and execution (*MAPE*) loop as known in control theory; to monitor, analyse and adjust interaction styles between actors at runtime. The natural evolution of HPS and mixed service-oriented systems are the formation of groups of humans and software-services to work jointly on complex tasks that require different capabilities and different areas of expertise. In this work we introduce formation patterns for social crowdsourcing environments. By looking at the MAPE cycle, these patterns play a significant role

within the planning and execution phase, which have not been sufficiently covered in our previous work. Our contributions are as follows:

1 We discuss the various building blocks needed to blend human capabilities into novel service-oriented crowdsourcing systems. Such systems play a significant role in future crowdsourcing systems and global service-oriented markets. We introduce the layers of our architecture enabling the seamless integration of social principles in technical (service-oriented) systems.

2 We introduce novel socially-based formation patterns in service-oriented crowdsourcing environments. These patterns include

   - mesh-based link establishment
   - broker-based formations.

3 We present an evaluation approach of these patterns using a web services-based testbed generator framework and web-based tools for the visualisation of simulation results.

This work is structured as follows: Section 2 introduces our socially-enhanced services computing approach including the architectural layers. The following Section 3 presents socially-based formation patterns. Our implementation is discussed in Section 4 followed by a discussion on related work in Section 5. We conclude the paper in Section 6.

2 Socially-enhanced crowdsourcing

A socially-enhanced approach for combining human and software services brings and number of advantages that have not been exploited in existing service-oriented crowdsourcing platforms.

- Top-down composition and interaction models are typically designed for long-term use. Dynamic environments that are short- to medium-lived such as open crowdsourcing systems require dynamic interaction models. Flexible interactions with the purpose of communicating, coordinating, and collaborating need to be supported in a service-oriented manner.
- Theories found in social network analysis are promising candidate techniques to support flexible interactions. Since interactions take place dynamically, capturing the purpose and context of interactions to infer meaningful social relations remains challenging.
- Social network principles such as formation algorithms help to overcome limited information exchange in separated collaborative networks through propagation of profile data. From the technical point of view, adaptive information flows need to be supported using services technology. Information needs to be discovered and exchanged based on the underlying social network.

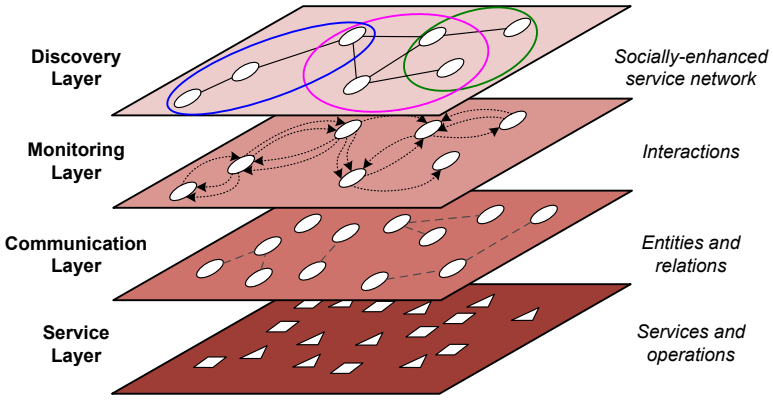
2.1 Architectural layers

In our approach, we position crowdsourcing in a business oriented setting, applying concepts found in service-oriented systems. In SOA, resources and services are discovered at run-time (late binding) based on a search query. Interactions between services take place based on standardised (XML-based) messages and documents. The benefit of this approach is that related standards and tools have been well adopted in the business process management (BPM) community. Thus, it is possible to integrate our techniques and prototypes in existing enterprise computing infrastructures.

Our approach is structured in four layers (see Figure 1):

- *service layer* consisting of services and available operations that can be discovered and invoked at runtime
- *communication layer* denoting the communication pathways within socially-augmented service networks
- *monitoring layer* capturing interactions between entities including human and software services
- *discovery layer* extracting relevant social networks to assist in the formation process.

Figure 1 Architectural layers (see online version for colours)



2.2 Service and communication layer

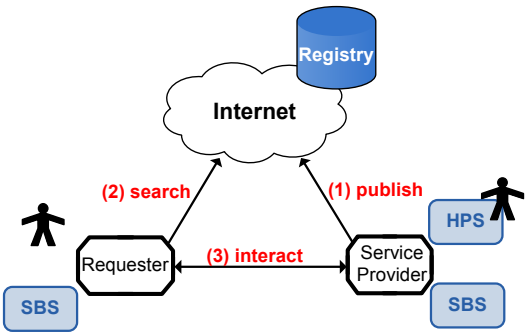
Web services play a fundamental role in supporting flexible collaboration scenarios. Also human interactions can be supported in a service-oriented manner using technologies such as SOAP [see HPS (Schall, 2011)]. HPS enhances the traditional 'SOA-triangle' approach by enabling people to provide services using the very same technology as implementations of SBS use. By following the SOA paradigm, three essential steps are performed (see Figure 2).

1 *Publish.* Users have the ability to create HPSs and publish the services in a distributed web-based registry. Publishing a service is as simple as posting a blog entry on the web. It is the association of the user's profile with an activity described as a service (WSDL). Interfaces provide the needed metadata support for the discovery of suitable HPSs. User profiles and services are managed as friend-of-a-friend (FOAF) (Brickley and Miller, 2010) networks. FOAF-based social network profiles support the discovery of people and resources (e.g., services).

2 *Search.* The service requester performs a keyword-based search (reflecting expertise areas) to find HPSs and/or SBSs. Ranking is performed to find the most relevant HPS based on the expertise of the user providing the service. Expertise is determined automatically by the HPS framework through context-sensitive interaction mining techniques (Schall, 2011).

3 *Interact.* The framework supports automatic user interface generation using XML-Forms technology (W3C, 2011). Thus, personalised interaction interfaces can be generated and rendered. The HPS framework can be used for interactions between humans and also for interactions between SBS and HPSs. For example, a task that is defined in the context of a business process can be outsourced to the crowd by requesting a HPS to perform the task.

Figure 2 Enhancing SOA with human capabilities (see online version for colours)



2.3 Monitoring and discovery layer

In our proposed mixed service-oriented system, people interact to perform their tasks. Work is modelled as activities, that describe the type and goal of work, temporal constraints, and used resources. As interactions take place in the context of activities, they can be categorised and weighted. SOAP is the standard message format to support interactions between distributed software services. This technology including extensions such addressing and correlation mechanisms is state-of-the-art in service-oriented environments and well supported by a wide variety of software frameworks. This fact enables the adoption of various monitoring and logging tools to observe interactions in service-oriented systems.

Interaction logs are used to infer metrics that describe the relation of single actors. Various metrics can be calculated by analysing interaction logs such as behaviour in

terms of availability and reciprocity. A simple example of a metric is the success rate of delegated tasks between two members (successfully processed tasks divided by the total number of delegated tasks). Relation metrics describe the links between actors by accounting for

1 recent interaction behaviour between actors

2 profile similarities (e.g., interest or skill similarities)

3 social and/or hierarchical structures (e.g., role models based on interactions).

However, we argue that social trust relations largely depend on personal interactions. We model a community of actors with their social relations as a directed graph, where the nodes denote network members, and edges reflect (social) relations between them. Since interaction behaviour is usually not symmetric, actor relations are represented by *directed links*.

By issuing keyword-based queries, each actor's connectivity to other community members in a particular query context Q is determined (Schall, 2011). The query context is described by a pool of keywords (e.g., describing certain expertise areas) picked from global taxonomies. Using logged interaction data and manual ratings, the link strength from one actor u to another actor v in context of Q is calculated. For that purpose, various metrics, such as availability of actors, average ratings, responsiveness and interest similarities can be calculated dynamically.

3 Social formation patterns for crowdsourcing

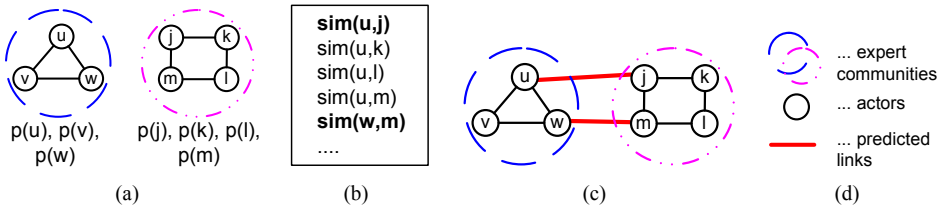
As the next step we introduce formation patterns to create compositions of HPS and SBS. We consider two models including *mesh* and *broker*-based formations to study patterns in open service-oriented crowdsourcing environments.

3.1 Link mesh

We call the first approach *link mesh*-based formation. We consider a dynamic social network model in which individual actors begin to interact based on predicted links. There are different choices how to model interactions when there are limited past observations [e.g., modelling interactions as games (Skyrms and Pemantle, 2000)]. Our approach is inspired by link prediction techniques in social networks [e.g., see work by Liben-Nowell and Kleinberg (2003)].

Consider two independent communities as depicted by Figure 3(a). Each actor has an interest profile associated with it. The first community consists of the actors u, v, w with profiles $p(u)$, $p(v)$, and $p(w)$; and the other community of the actors j, k, l, m whose profiles are depicted by $p(j)$, $p(k)$, $p(l)$ and $p(m)$. The goal is to connect these communities to perform a joint (crowdsourced) task. In our scenario, each community has expertise in a distinct field. Thus, both communities could jointly process more complex tasks depending on their expertise.

Figure 3 Link mesh approach, (a) communities (b) similarity (c) link establishment (d) symbols (see online version for colours)



In our approach, we bootstrap the communication between actors by predicting edges between them based on their similarity of profiles [see Figures 3(b) and 3(c)]. Higher similarity denotes higher common interests. The cosine similarity, a popular measure to determine the similarity of two vectors in a vector space model, is applied to determine the similarity of two actor profiles $p(u)$ and $p(j)$ [see equation (1)]. The result is a real value $sim \in [0, 1]$, whereas 0 denotes no overlap between used user profile keywords and 1 reflects identically keywords in the users' profiles.

$$sim(u, j) = \cos(p(u), p(j)) = \frac{\mathbf{p_u} \cdot \mathbf{p_j}}{||\mathbf{p_u}|| \; ||\mathbf{p_j}||} \tag{1}$$

Based on predicted edges, actors start to interact. From this point on, our system starts to monitor the interactions and to calculate a set of metrics such as link strength and reciprocity. The mutual strength of edges between actors determines which interactions are reinforced, and the network structure emerges as a consequence of the dynamics of the actors' behaviour. For example, if an actor u proofs its trustworthy behaviour over time, u is considered to be a valuable collaboration partner for its neighbours (Skopik et al., 2010).

3.2 Social brokers

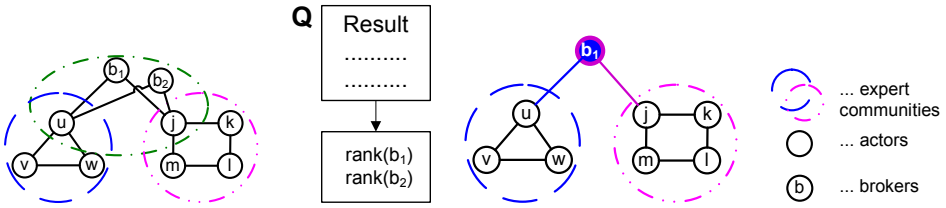
In the second approach, the goal is to connect certain regions of networks (for example, expertise clusters or communities) that were not previously connected. Brokers differ from other actors by their mediation capabilities (see Burt, 2004). Thus, such intermediaries can potentially broker information and aggregate ideas arising in different parts of a network. Instead of establishing a direct path between two independent communities, brokers mediate information flows in social crowdsourcing environments [see actors b_1 and b_2 in Figure 4(a)].

In dynamic crowdsourcing environments, it is essential that brokers monitor frequently demanded contacts, and that they update and maintain their relations to increase and strengthen their popularity, and consequently, social trust within and across communities. If demand in the network decreases, the broker must find and establish new relations.

To discover brokers [see Figure 4(b) and 4(c)], we have developed a query language called *Broker Query and Discovery Language* (BQDL, see also Schall et al., 2011) that allows for the specification of complex queries in crowdsourcing systems (e.g., finding exclusive brokers that do not cause conflicts between competing actors or communities). The language is inspired by an SQL-like syntax. However, BQDL operates on a graph

defined as $G = (N, E)$ composed of a set of nodes N and edges E. A `Select` statement retrieves nodes and edges in G as well as aggregates of graph properties (for example, properties of a set of nodes). While traditional relational databases operate on tables, BQDL uses the `From` clause to perform queries on a graph G. A `Where` clause specifies filters and policies upon nodes, edges, and paths.

Figure 4 Broker discovery and formation approach, (a) communities (b) query (c) bridging communities (d) symbols (see online version for colours)



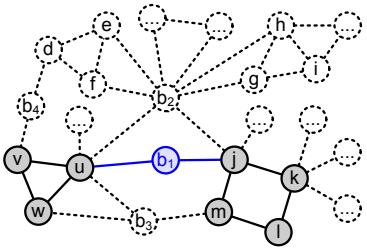
As a simple example in Figure 5, consider two initially disconnected communities (sets of nodes) depicted as variables `var source` $= \{n_1, n_2, \ldots, n_i\}$ and `var target` $= \{n_j, n_{j+1}, \ldots, n_{j+m}\}$ residing in the graph G.

Figure 5 BQDL query: find broker to connect two predefined communities (see online version for colours)

```
Input: Graph G, var source = {n1, n2, ..., ni},
       var target = {nj, nj+1, ..., nj+m}
Output: List of brokers

Select node From (
 ( Select distinct(node) From G
   Where
   /* At least one in source `knows' node */
   ( [1..*] n in source ) satisfy
      Path (n to node) as P1 With P1.length = 1 )
      as G1,
   ( target ) as G2
 )
 Where
 /* Retain all nodes that satisfy path filter */
 ( <all> n in G1.nodes ) satisfy
   /* Path to any in G2.nodes */
   Path (n to [1..*] G2.nodes) as P2
    With P2.length = 1
   and
   /* Retain all edges that satisfy edge filter */
      ( <all> e in G1.edges ) satisfy
    (e.relation = EPredicates.BIDIRECTIONAL) and
    (e.trust >= MTrust.MEDIUM)

Order by node
```



The goal is to find a broker connecting disjoint sets of nodes (i.e., not having any direct links between each other). The approach: two subgraphs `G1` and `G2` are created to determine brokers which connect the source community $\{u, v, w\}$ with the target community $\{j, k, l, m\}$ (i.e., see `From` construct). The output of the query is (the example shown in Figure 5) a list of brokers connecting $\{u, v, w\}$ and $\{j, k, l, m\}$. The lines 1 to 3 specify the input/output parameters of the query.

As a first step, a (sub)select is performed using the statement as shown by the lines 6 to 11. The statement `distinct(node)` means that a set of unique brokers shall be selected based on the condition denoted as the `Where` clause with a filter (lines 9 to 10). The term '`[1..*] n in source`', where `source` is the set of nodes passed to the query as input argument, means that at least one node $n \in G$ must satisfy the subsequent condition. Here the condition is that the node n has a link (i.e., through `knows` relations) to the source set of nodes. This is accomplished by using the `Path` function that checks whether a link between two nodes exists (the argument '`(n to node)`'). The path alias is used to specify additional constraints such as the maximum path length between nodes (here '`P1 With P1.length = 1`'). The second step is to create an alias `G2` for the target community $\{j, k, l, m\}$. By using the aliases `G1` (line 11) and `G2` (line 12) further filtering can be performed using the `Where` clause in line 14. The same syntax is used as previously in the sub-select statement (lines 9 to 10). The construct `<all>` retains nodes '`n in G1.nodes`' (`G1` holding the set of candidate brokers) that are connected to at least one node in the target community `G2` with direct links ('`P2 with P2.length = 1`'). Further filtering is performed by defining lines 22 to 24.

Here, brokers in `G1` and both the source $\{u, v, w\}$ the target community $\{j, k, l, m\}$ must have edges between each other that are bidirectional. In our graph representation, this means that each relation has to be interpreted as, for example, b_1 `knows` j and j `knows` b_1. A set of different metrics is established in our system. A specific type of metric (e.g., trust) is denoted by the namespace `MTrust`. In the specified query, each actor in the result set must share a minimum level of trust depicted as '`e.trust >= MTrust.MEDIUM`'. Trust metrics are associated to edges between actors. The term `MTrust.MEDIUM` is established based on mining data to obtain linguistic representations by mapping discrete values (metrics) into meaningful intervals of trust levels.

The last statement '`Order by node`' in Figure 5 implies a ranking procedure of brokers. This can be accomplished by using eigenvector methods in social networks such as the PageRank algorithm to establish authority scores (the importance or social standing of a node in the network) or advanced game-theoretic techniques based on the concept of structural holes (see for example, Kleinberg et al., 2008). The detailed mechanisms of this procedure are not the focus of this work.

4 Crowdsourcing evaluation environment

Our current system implementation covers various aspects including the design and runtime invocation of HPSs and SBSs as well as mining and ranking of services. These aspects have been briefly introduced in our previous discussions. An important aspect are performance issues due to the size of the social graph in large-scale crowdsourcing environments. These issues are not covered in this work. A detailed discussion on performance aspects and mechanisms for mining and ranking can be

found in Schall (2011, 2012). Here we focus on the evaluation approach and a set of selected tools helping engineers to design and evaluate formation patterns in social crowdsourcing environments.

The first step in the evaluation approach is to design and create distributed service-oriented testbeds that are populated by HPSs and SBSs. For this purpose, we utilise a testbed generator framework called Genesis2 (Juszczyk and Dustdar, 2010) enhanced with additional adaptation capabilities (Psaier et al., 2010). The next step in the evaluation process is to define the behaviour of actors in the testbed. Based on the specified behaviour, interactions emerge based on behaviour patterns. Each interaction is captured by a logging component to create a 'social interaction graph' that is later used for analysis. In the following, we outline the basic features of the Genesis2 testbed generator and our formation evaluation tools built on-top of the service-oriented crowdsourcing system's testbed.

4.1 SOA testbeds

Genesis2 is a testbed generator for runtime evaluation of SOA-based concepts/implementations (e.g., service-oriented crowdsourcing environments). It enables engineers to design, implement, deploy, and steer SOA-based environments consisting of various components. In particular it also allows to simulate the behaviour of the components including HPSs. Genesis2's most distinct feature is its ability to generate and deploy testbed instances and to perform realistic tests at runtime. Notice, the testbed is created fully distributed consisting of various backend hosts that can be deployed in large-scale cloud computing environments. A frontend lets engineers specify and update testbeds.

Here we show a simple script that can be specified to create services in the distributed evaluation environment. We make intensive use of Genesis2's abilities to programme the testbed's behaviour via Groovy (see http://groovy.codehaus.org/) scripts and to adapt these at runtime (for example, adapting strategies how tasks are processed by HPSs to analyse different network structures and the effect of formation patterns).

The demo script in Listing 1 specifies a service structure that applies an adaptable assignment strategy for forwarding requests to HPSs. Moreover, all communication between the services is captured to monitor the testbed's activity. First, a data type definition is imported from an XML schema file, which is applied as a message type for the subsequently defined web service `TranslationService` (e.g., HPS to translate documents from one language to another).

The service defines an operation that processes requests `assignProcRequest`. The logic in the closure `dStrat` maintains the current individual acceptance behaviour of the service. Note, a Groovy closure (see http://groovy.codehaus.org/Closures) is a reusable 'code block'. The content can be changed at any time during runtime for a more precise simulation (c.f., line 33). On a reject, an invalid assignment id `assId = 0` is returned. Operations `getStatus` and `getProcResult` inform about the current progress and return the processing result location, respectively. Lines 26 to 29 list the definition of a message interceptor. The interceptor allows to log the environment interactions. Logs provide observations and allow conclusions on the current system status. The *hooks* observe the input and output streams and `code` contains the processing strategy for raw logs. The lines 31 to 32 define a hosting environment and deploy service on distributed hosts.

Listing 1 Example of a Genesis2 testbed specification

```
1  def procType=datatype.create("file.xsd","desc") // xsd import
2
3  def srv=webservice.build {
4    // create human provided service
5    TranslationService(binding="doc,lit", namespace="http://...") {
6      def reportQueue = [] //queue of assignments
7      assignProcRequest(input:procType, response:int) {//assignment
8       def assId = genAssignmentId()
9       if (dStrat(input))
10       reportQueue+=input
11      else assId = 0
12      return assId
13     }
14     // result acceptance strategy as closure variable
15     dStrat={ input −> ...}
16     getStatus(assId:int, response:String) {//status check
17      return getStatusOn(assId)
18     }
19     getProcResult(assId:int, response:String) {//result of assignment
20      if (completed(assId))
21       return getResultURI(assId)
22     }
23   }
24 }[0]
25
26 def li=callinterceptor.create() // logging interceptor
27 li.hooks=[in:"RECEIVE", out :"PRE_STREAM"] // bind to phases
28 li.code={ctx −>... } // process msg
29 srv.interceptors+=li // attach monitoring interceptor
30
31 def h=host.create("somehost:8181") // import back−end host
32 srv.deployAt(h) // deploy service at remote back−end host
33 dStrat={ input −> ...} // change strategy at runtime
```

Listing 2 shows the definition of a WSDL interface that is used to interact with people in a service-oriented manner. Using WSDL as the interface description language brings the advantage that the very same standard is being used to describe SBS and HPS, thereby creating a unified service-oriented computing environment for people and software services.

The WSDL in Listing 2 is automatically generated by the testbed environment based on the previously introduced script (Listing 1). It shows also the structure of the complex data type that is passed to the HPS. The elements are used for the following purpose:

- `docTitle`: defines the name of the document to be translated by a human.
- `docUri`: contains the location (e.g., link to a document repository) where the document can be downloaded from.
- `length`: the length (number of words) of the document.
- `language`: specifies the language in which the document is written.

- `translation`: the target language to be translated to (e.g., provide translation from German to English).
- `translationUri`: the location (repository) where the translated document can be uploaded to. The person translating the document may provide an alternative location that can be requested via the web service operation `getProcResult`.
- `mimeType`: states the acceptable format of the translated document (e.g., PDF, Word document, or plain text).

Listing 2 HPS WSDL definition

```
<wsdl:definitions name="TranslationService" ...>
 <wsdl:types>
  <xs:schema elementFormDefault="unqualified" tns="http://...">
   <xs:element name="assignProcRequest"
    type="tns:assignProcRequest" />
   <xs:element name="getStatus" type="tns:getStatus" />
   <xs:element name="getProcResult" type="tns:getProcResult" />
   <!-- responses omitted-->
   <xs:complexType name="desc">
    <xs:element name="docTitle" type="xs:string" />
    <xs:element name="docUri" type="xs:string" />
    <xs:element name="length" type="xs:string" />
    <xs:element name="language" type="xs:string" />
    <xs:element name="translation" type="xs:string" />
    <xs:element name="translationUri" type="xs:string" />
    <xs:element name="mimeType" type="xs:string" />
    <!-- further details omitted-->
   </xs:complexType>
   <!-- other types... -->
  </xs:schema>
 </wsdl:types>
 <wsdl:message name="assignProcRequest">
  <wsdl:part element="tns:assignProcRequest" name="params" />
 </wsdl:message>
 <!-- messages... -->
 <wsdl:portType name="TS">
  <wsdl:operation name="assignProcRequest">
   <!-- in-/output... -->
  </wsdl:operation>
 </wsdl:portType>
 <wsdl:binding name="TSSoapBinding" type="tns:TSService">
   <soap:binding style="document" transport="http://schemas..."/>
    <!-- operations... -->
 </wsdl:binding>
 <wsdl:service name="TSService">
  <wsdl:port binding="tns:TSSoapBinding" name="TSPort">
   <soap:address location="http://somehost:8080/..."/>
  </wsdl:port>
 </wsdl:service>
</wsdl:definitions>
```

4.2 Interaction logging

The previously presented results are based on Genesis2's testbed generation capabilities and a framework for monitoring and logging interactions between services. Interactions are captured through (SOAP) message interceptors deployed within the service runtime environment. Logged messages are persistently saved in a database for analysis. An example interaction log is shown by Listing 3, which includes various SOAP header extensions for message correlation and context-aware interaction analysis.

Listing 3 HPS SOAP example

```
<soap:Envelope
 xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:hps="http://www.myhps.org/hps/">
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:wsa="http://schemas.xmlsoap.org/ws/.../addressing"
 <soap:Header>
  <hps:timestamp value="2011-04-29T17:24:18"/>
  <hps:activity url="http://.../Activity#42"/>
  <wsa:MessageID>uuid</wsa:MessageID>
  <wsa:From>http://.../Actor#Actor1</wsa:From>
  <wsa:To>http://.../Actor#Actor2</wsa:To>
  <wsa:ReplyTo>http://.../Actor#Actor3</wsa:ReplyTo>
  <wsa:Action>http://.../Type/Translate</wsa:Action>
 </soap:Header>
 <soap:Body>
  <hps:Request>
   <!-- request omitted-->
   <hps:keywords>document, translation</hps:keywords>
  </hps:Request>
 </soap:Body>
</soap:Envelope>
```

The purpose of the most important extensions is outlined in the following [see Schall (2011) for details on the implementation]:

- `Timestamp` captures the actual creation of the message and is used to calculate temporal interaction metrics, such as average response times
- `Activity uri` describes the context of interactions based on the activity performed by the user (see also Schall, 2011)
- `MessageID` enables message correlation, i.e., to properly match requests and responses
- `WS-Addressing` extensions (Box et al., 2004), besides `MessageID`, are used to route requests through the network.

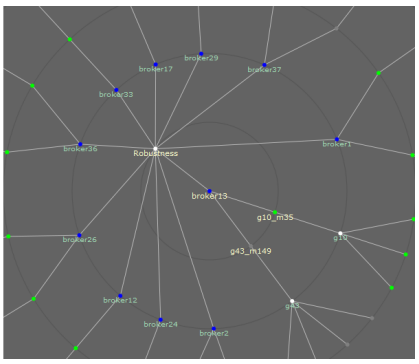
The SOAP body transports the actually exchanged message. In this example a delegation of a translation request is performed. The example shows how one actor requests some help from another one. Note, interactions are only captured to collect keywords and support the creation of user profiles. Logged data can be purged immediately after

keyword extraction. Thus, our approach of interaction observation is less intrusive compared to others (e.g., semantic analysis of captured messages). We understand today's privacy concerns as a big issue of most systems that log user data for adaptation purposes (such as establishing network links).

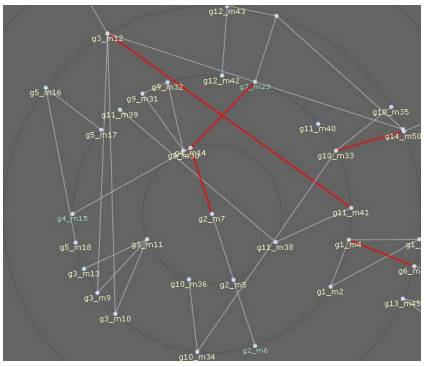
4.3 Discovery and formation tool

The implemented prototype includes a web-based formation tool assisting engineers in analysing various formation patterns. In Figure 6, we show two views, one to support the discovery of brokers and another view to analyse the effects of link mesh-based formations. The network view is obtained by mapping raw SOAP-interactions into a graph representation composed of nodes (services) and edges (interaction links). Each link holds additional data such as the number of exchanged messages, reciprocity, etc., between actors (HPS and SBS).

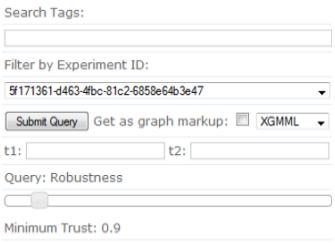
Figure 6 Web-based network visualisation and formation tool (best viewed online), (a) broker discovery view (b) visualisation controls (c) link mesh view (d) example of FOAF profile (see online version for colours)



(a)



(c)



(b)

(d)

As a first step, the engineer accesses information captured from the service-oriented collaboration environment. In our implementation, this is performed by selecting a particular set of logs which are associated with an *Experiment ID*. After issuing the corresponding query, a graph is visualised. Depending on the purpose of the evaluation (analysing broker discovery patterns or mesh-based formations), different views are shown.

The broker view (network) is centred around a query's keywords (e.g., which brokers are connected to specific expertise clusters). Our evaluation approach helps to analyse the received benefit and reputation of brokers under different conditions such as network structure and connectedness of individual communities. The user is able to select thresholds (e.g., similarity or trust-based thresholds for link strength) by moving a slider bar. For example in the link mesh scenario, a reduced (demanded) similarity threshold results in predicated trust-based edges being added to the visualisation (color online: depicted as red colored edges between nodes). Also, interactions can be retrieved in various network formats including graph markup-based representations (e.g., GML) or FOAF-based profiles that include `<foaf:interest>` tags. This feature can be used to aggregate captured networks and user profiles from distributed environments and to perform reasoning.

5 Related work

We consider service-oriented crowdsourcing environments wherein services can be added at any point in time. Following the open world assumption, the availability of services is not only shaped in a top-down manner but also influenced by users that actively contribute new services to the web.

In service-oriented business environments, standards have been established to model human-based process activities [see BPEL4People (Agrawal et al., 2007)] and human tasks [WS-HumanTask (Amend et al., 2007)]. However, these standards demand for the precise definition of interaction models between humans and services. In our approach, we combine SOA concepts and social principles. We adopt the concept of HPS (Schall, 2011) to support flexible service-oriented collaborations across multiple organisations and to provision human expertise in service-oriented crowdsourcing environments. Similarly, emergent collectives as defined by Petrie (2010) are networks of interlinked valued nodes (services). Open service-oriented systems are specifically relevant for innovative crowdsourcing applications. Crowdsourcing has received considerable attention in the last couple of years (Doan et al., 2011, Gentry et al., 2005, Yang et al., 2008). While existing platforms [e.g., MTurk (Amazon, 2011)] only support simple interaction models (tasks are assigned to individuals), social network principles support more advanced techniques such as formation and adaptive coordination.

In our approach, we focus on the strategic formation in social networks and communities. Work by Tsai (2000) investigated the evolutionary dynamics of network formation by analysing how organisational units create new linkages for resource exchange. The theory of structural holes was developed by Burt (2004) and is based on the hypothesis that individuals can benefit from serving as intermediaries between others who are not directly connected. A formal approach to strategic formation based on advanced game-theoretic broker incentive techniques was presented in Kleinberg et al. (2008).

Our approach is based on interaction *mining and metrics* to dynamically discover alliances in service-oriented collaborations that could be established based on brokers. The availability of rich and plentiful data on human interactions in social networks has closed an important loop (Kleinberg, 2008), allowing one to model social phenomena and to use these models in the design of new computing applications such as

crowdsourcing techniques (Brabham, 2008). (Artz and Gil, 2007; Mui et al., 2002). We focus on social trust (Golbeck, 2009; Ziegler and Golbeck, 2007) that relies on user interests and collaboration behaviour.

6 Conclusions

In this work, we highlighted the role of humans in SOA as *first class citizens*. Using a conceptual layered framework, we discussed various building blocks to blend human capabilities into novel service-oriented collaboration systems. The benefit of our approach is that related XML standards and tools have been well adopted in the BPM community. Thus, it is possible to integrate our techniques and prototypes in existing enterprise computing infrastructures to support also the incorporation of crowdsourcing services.

We presented novel socially-based composition patterns in SOA including *brokers* and *link mesh*-based formations. These patterns can be evaluated using a web services-based testbed generator framework which helps engineers to understand the impact of different metrics and thresholds on resulting network structures. Our future work will specifically deal with the question of stakeholder support and BPM integration. In particular, the question we attempt to answer is which stakeholders need to be involved when designing novel crowdsourcing applications and which part of the evaluation framework are made visible to them. For example, engineers may be interested in dynamic interaction and discovery policies whereas business analysts may want to design different incentive schemes for crowdsourcing services. These questions have not been addressed in our current research. Also, we plan to extend our effort in designing novel link-based reputation mechanisms for the discovery of brokers.

Acknowledgements

This work is supported by the EU through the project COIN (216256).

References

Agrawal, A. et al. (2007) ‘Ws-bpel extension for people (bpel4people)’, Version 1.0.

Amazon (2011) Available at http://www.mturk.com (accessed on 19 April 2012).

Amend, M. et al. (2007) ‘Web services human task (ws-humantask)’, Version 1.0.

Artz, D. and Gil, Y. (2007) ‘A survey of trust in computer science and the semantic web’, *J. Web Sem.*, Vol. 5, No. 2, pp.58–71.

Box, D. et al. (2004) Available at http://www.w3.org/Submission/ws-addressing/ (accessed on 19 April 2012).

Brabham, D.C. (2008) ‘Crowdsourcing as a model for problem solving: an introduction and cases’, *Convergence*, Vol. 14, No. 1, p.75.

Brickley, D. and Miller, L. (2010) Available at http://xmlns.com/foaf/spec/ (accessed on 19 April 2012).

Burt, R.S. (2004) ‘Structural holes and good ideas’, *American Journal of Sociology*, September, Vol. 110, No. 2, pp.349–399.

Davis, J.G. (2011) 'From crowdsourcing to crowdservicing', *Internet Computing, IEEE*, May–June, Vol. 15, No. 3, pp.92–94, ISSN 1089-7801, doi: 10.1109/MIC.2011.61.

Doan, A., Ramakrishnan, R. and Halevy, A.Y. (2011) 'Crowdsourcing systems on the world-wide web', *Commun. ACM*, April, Vol. 54, No. 4, pp.86–96, ISSN 0001-0782.

Gentry, C., Ramzan, Z. and Stubblebine, S. (2005) 'Secure distributed human computation', in *EC '05*, pp.155–164, ACM.

Golbeck, J. (2009) Trust and nuanced profile similarity in online social networks', *ACM Trans. Web*, September, Vol. 3, No. 4, pp.1–33.

Howe, J. (2008) *Crowdsourcing: Why the Power of the Crowd is Driving the Future of Business*, Crown Publishing Group, New York, NY, USA.

Juszczyk, L. and Dustdar, S. (2010) 'Script-based generation of dynamic testbeds for soa', in *ICWS*, pp.195–202.

Kleinberg, J. (2008) 'The convergence of social and technological networks', *Commun. ACM*, Vol. 51, No. 11, pp.66–72, ISSN 0001-0782.

Kleinberg, J., Suri, S., Tardos, E. and Wexler, T. (2008) 'Strategic network formation with structural holes', *SIGecom Exchanges*, Vol. 7, No. 3, pp.11:1–11:4.

Liben-Nowell, D. and Kleinberg, J. (2003) 'The link prediction problem for social networks', in *CIKM '03*, pp.556–559, ACM.

Mui, L., Mohtashemi, M. and Halberstadt, A. (2002) 'A computational model of trust and reputation for e-businesses', in *HICSS*, p.188.

Petrie, C. (2010) 'Plenty of room outside the firm', *IEEE Internet Computing*, Vol. 14, No. 1, pp.92–96, ISSN 1089-7801.

Psaier, H., Juszczyk, L., Skopik, F., Schall, D. and Dustdar, S. (2010) 'Runtime behavior monitoring and self-adaptation in service-oriented systems', in *SASO*, pp.164–174, IEEE.

Schall, D. (2011) 'A human-centric runtime framework for mixed service-oriented systems', *Distributed and Parallel Databases*, Vol. 29, Nos. 5–6, pp.333–360, ISSN 0926-8782, available at http://dx.doi.org/10.1007/s10619-011-7081-z (accessed on 19 April 2012).

Schall, D. (2012) 'Expertise ranking using activity and contextual link measures', *Data & Knowledge Engineering*, Vol. 71, No. 1, pp.92–113, ISSN 0169-023X, doi: 10.1016/j.datak.2011.08.001, available at http://www.sciencedirect.com/science/article/pii/S0169023X11001133 (accessed on 19 April 2012).

Schall, D., Truong, H-L. and Dustdar, S. (2008) 'Unifying human and software services in web-scale collaborations', *IEEE Internet Computing*, Vol. 12, No. 3, pp.62–68.

Schall, D., Skopik, F., Psaier, H. and Dustdar, S. (2011) 'Bridging socially-enhanced virtual communities', in *SAC '11*, ACM.

Skopik, F., Schall, D. and Dustdar, S. (2010) 'Modeling and mining of dynamic trust in complex service-oriented systems', *Information Systems*, Vol. 35, No. 7, pp.735–757.

Skyrms, B. and Pemantle, R. (2000) 'A dynamic model of social network formation', *PNAS*, August, Vol. 97, No. 16, pp.9340–9346.

Tsai, W. (2000) 'Social capital, strategic relatedness, and the formation of intra-organizational strategic linkages', *Strategic Management Journal*, Vol. 21, No. 9, pp.925–939.

Vukovic, M. (2009) 'Crowdsourcing for enterprises', in *Congress on Services*, pp.686–692.

W3C (2011) Available at http://www.w3.org/MarkUp/Forms/ (accessed on 19 April 2012).

Yang, J., Adamic, L.A. and Ackerman, M.S. (2008) 'Crowdsourcing and knowledge sharing: strategic user behavior on taskcn', in *EC '08*, pp.246–255, ACM.

Ziegler, C-N. and Golbeck, J. (2007) 'Investigating interactions of trust and interest similarity', *Decision Support Systems*, Vol. 43, No. 2, pp.460–475.