Engineering Service-Oriented Crowdsourcing for Enterprise Environments

Daniel Schall[†], Harald Psaier[†], Martin Treiber[†‡], Florian Skopik[†]
[†]Distributed Systems Group, Vienna University of Technology
lastname@infosys.tuwien.ac.at
[‡]ikangai solutions
office@ikangai.com

Date: August 2010

ABSTRACT

Crowdsourcing has emerged as an important paradigm in human problem solving techniques on the Web. More often than noticed, programs outsource tasks to humans which are difficult to implement in software. In this work we demonstrate the benefit of service-oriented architectures (SOA) applied for *enterprise crowdsourcing*. Interactions in such environments span human and software services. Crowdsourcing applications typically utilize the capabilities of people in open and dynamic Web-based systems. Dynamically changing environments, however, demand for flexible interaction models due to the changing availability of people and services. Our main contributions center around the convergence of process flows and dynamic flows in crowdsourcing environments. Here we present a real world example of a human provided service for crowdsourcing in enterprise environments. We demonstrate the application of a *human assisted image processing* service in a process-centric flow. We discuss the foundational building blocks for realizing the design, execution, and adaptation of service-oriented crowdsourcing applications.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques; H.3.5 [**Online Information Services**]: Web-based Services

General Terms

Service-oriented systems, process adaptation, crowdsourcing

Keywords

Human computation, human assisted image processing

1. INTRODUCTION

The collaboration landscape has changed dramatically over the last years by enabling users to shape the Web and availability of information. While in the past collaborations were bounded to intra-organizational collaborations using a companies specific platform, and also limited to messaging tools such as email, it is nowadays possible to utilize the knowledge of an immense number of people participating in collaborations on the Web. The shift toward the Web 2.0 allows people to write blogs about their activities, share knowledge in forums, write Wiki pages, and utilize social platforms to stay in touch with other people. Task-based platforms for human computation and crowdsourcing, including Amazon Mechanical Turk[1], CrowdFlower[2], Google's Smartsheet[3], or oDesk[4] enable access to the manpower of thousands of people on demand by creating human-tasks that are processed by the crowd. Human-tasks include activities such as designing, creating, and testing products, voting for best results, or organizing information.

Crowdsourcing. The notion of crowdsourcing describes an online, distributed problem solving and production model with increasingly interested business parties in the last couple of years [4, 13]. One of the main motivations to outsource activities to a crowd is the potentially considerable spectrum of returned solutions. Furthermore, competition within the crowd ensures a certain level of quality. According to [18], there are two dimensions in existing crowdsourcing platforms. The first categorizes the function of the platform. Currently these can be divided in communities (i) specialized on novel designs and innovative ideas, (ii) dealing with code development and testing, (iii) supporting marketing and sales strategies, and (iv) providing knowledge support. Another dimension describes the crowdsourcing mode. Community brokers assemble a crowd according to the offered knowledge and abilities that bid for activities. Purely competition based crowdsource platforms operate without brokers in between. Depending on the platform incentives for participation in the crowd are either monetary or simple credit oriented. Even if crowdsourcing seems convenient and attracts enterprises with scalable workforce and multilateral expertise the challenges of crowdsourcing are a direct implication of human's ad-hoc, unpredictable behavior and variety of interaction patterns.

[1]`http://www.mturk.com/`
[2]`http://crowdflower.com/`
[3]`http://www.smartsheet.com/`
[4]`https://www.odesk.com/`

The main questions in crowdsourcing applications include: (i) How can one model *interactions* ranging from process flows (*PFLs*), which are predefined at design time, and crowd flows (*CFLs*) *emerging* at run-time in crowdsourcing applications? (ii) How can crowdsourcing applications be controlled and adapted regarding social phenomena, such as evolving interests and expertise? (iii) How can crowds be controlled and influenced in order to deliver acceptable Quality-of-Service (QoS) levels? Prerequisite is the possibility to track the tasks' progress and recognize misbehavior, lacks of capabilities and knowledge, overload of individuals, and identification of successful but also unsuccessful collaboration formations. Finally, a major challenge remains to deploy feasible adaptation strategies to rearrange the crowd.

Our Contributions. Our proposed mixed service-oriented systems aims at addressing the following challenges:

- *Crowd Scenario.* We provide examples of crowd scenarios and highlight the advantages of an integration into an SOA environment.
- *Crowd Integration.* We describe models, patterns, and methodologies that apply SOA in order to implement crowd platforms.
- *Crowd Architecture.* We outline a novel SOA infrastructure that provides crowd platforms with the necessary adaptability in dynamic environments.

2. SOA FOR CROWDSOURCING

Service-oriented architecture (SOA) is an emerging paradigm to realize extensible large-scale systems. As interactions and compositions spanning multiple enterprises become increasingly commonplace, organizational boundaries appear to be diminishing in future service-oriented systems. In such open and flexible enterprise environments, people contribute their capabilities in a service-oriented manner. We consider mixed service-oriented systems based on two elementary building blocks: (i) Software-Based Services (SBS), which are fully automated services and (ii) Human-Provided Services (HPS) [14] for interfacing with people in a flexible service-oriented manner.

Here we discuss service-oriented environments wherein services can be added at any point in time. Following the open world assumption, humans actively shape the availability of HPSs by creating services. Interactions between HPSs are performed by using Web service-based technology (XML-based SOAP messages). Without any coordination, such systems may exhibit undesirable properties due to unexpected behavior of people. Thus, social implications caused by human participation pose additional challenges to designing large-scale mixed service-oriented systems. However, with size also the effort for managing dynamically growing and loosely coupled systems is increasing. Periodic adaptations are essential to keep a system within well-defined states, including stable load conditions or desired behavior. Due to the scale and inherent dynamics of open large-scale systems, new approaches for designing, developing and testing are required.

2.1 On-demand Workforce

A motivating scenario for discovering members of the crowd in process-centric flows is depicted in Figure 1. The **process flow** (PFL) may be composed of single tasks that are either

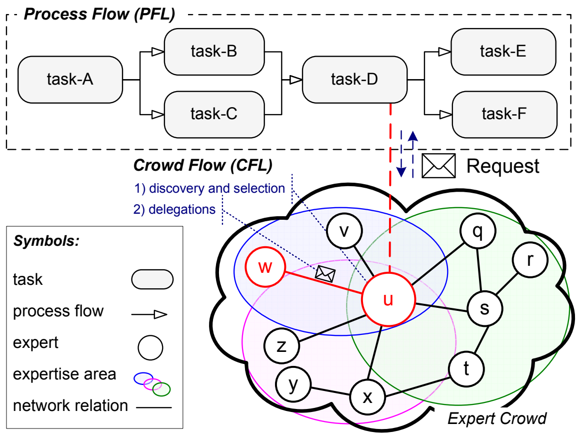


Figure 1: Utilizing crowdsourcing in process flows.

processed by corresponding Web services or are assigned to responsible persons. In this scenario, a task (`task-D`) may be outsourced to the crowd. This is done by preparing a request containing various artifacts to be processed by the crowd and additional metadata such as time constraints and complexity of the task. The first step in a mixed service-oriented systems is to discover and select a suitable HPS. Discovery and selection is based on both, matching of functional capabilities (the service interface) and non-functional characteristics such as the degree of human expertise. In the depicted case, the actor u has been selected as the responsible service for processing the given request. The selection is based on u's expertise (visualized by the size of the node in the network), which is influenced by u's gradually evolving expertise and dynamically changing interests. The novelty of our approach is that members of the crowd may also interact with each other by, for example, simply delegating requests to other members (e.g., member u delegates the request to the peer w) or by splitting the request into sub-tasks that are assigned to multiple neighboring peers in the network. In our approach, the discovery of neighbors is based on the social structure of networks (e.g., friend or buddy lists). How decisions within the crowd are made (delegation or split of tasks) emerges over time due to changing interaction preferences and evolving capabilities of people (depicted as *expertise areas*). As introduced before, these dynamic interactions are defined as **Crowd Flows** (CFL).

Flexible interaction models allow for the natural evolution of communities based on skills and interest. However, misbehavior patterns such as an increasing amount of delegations may degrade the overall processing performance of task requests within the crowd. For example, since the selection of crowd members is based on evolving skills and expertise, member u may adopt a selfish strategy by accepting and delegating a large amount of requests in order to increase its reputation within the network. Our approach and techniques help to alleviate related problems.

2.2 Real World Example

The presented example scenario illustrates a model of a PFL that comprises a task which is outsourced to the crowd. In our example, the company ikangai solutions[5] provides a service which transforms vector graphics or bitmap im-

[5]`http://www.ikangai.com`

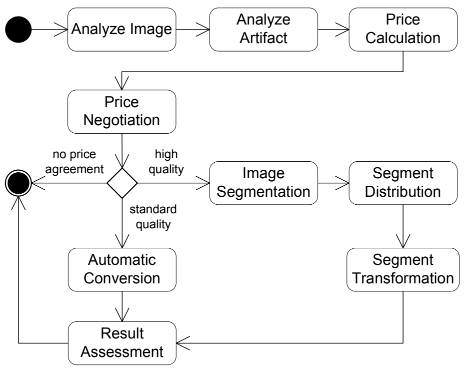


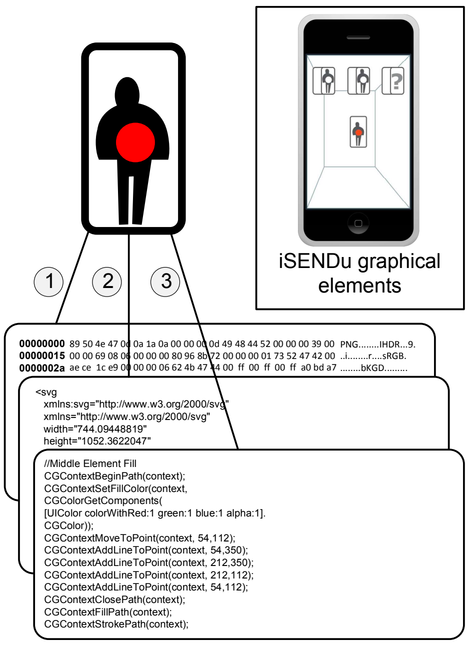
Figure 2: Human assisted image processing flow.

ages to Quartz 2D Objective-C code for use on the Apple iPhone[6]. The different available devices with varying form factors (screen size and resolution) require *manual* adaptation and tailoring of graphics to the properties of a particular device. We show a simplified version of the actual process[7] in Figure 2. The first step is *Analyze Image*. Here a software tool checks whether the image (for example, a received bitmap) needs to be converted to a vector image. After that a vector representation [6] is used to determine the complexity of the image considering the number of vertices, edges and fillings (*Analyze Artifact*). The costs (*Price Calculation*) are estimated based on the desired quality. The *Price Negotiation* steps has various alternatives. If an agreement with the customer about the price can be reached, the image is either transformed **automatically** by a software tool (standard quality depicted by the step *Automatic Conversion*) or **processed by the crowd** by performing the steps *Image Segmentation*, *Segment Distribution* and *Segment Transformation*. In either case, the result is checked by an ikangai employee (*Result Assessment*). Alternatively, the process is being aborted.

The steps *Image Segmentation*, *Segment Distribution* and *Segment Transformation* (see Figure 3) are activities performed in the crowd during the execution of the transformation process. The actual allocation of the people/services for the transformation activity depends on their expertise: experienced people receive more complex segments to transform into Objective-C code, while less experienced crowd members receive considerable less complex segments to transform into Objective-C code. The allocation is done by a supervisor (also member of the crowd) who distributes the segments accordingly. The actual degree of human involvement depends on the desired quality by the customers and the available resources, i.e., humans in the crowd. If high quality work is required, each transformation step is assisted by a human who ensures the required quality. Finally, once the result of the image conversion is received from the crowd, a company employee assesses the final result *Result Assessment*). We can summarize the necessary steps as shown in Figure 3. With the different artifacts it becomes apparent that the single tasks in each step are not trivially computable by software but require human assistance.

[6]`http://developer.apple.com/iphone/`

[7]For simplicity, we only illustrate the essential steps of the process flow without discussing detailed price negotiation procedures.



```
00000000 89 50 4e 47 0d 0a 1a 0a 00 00 00 0d 49 48 44 52 00 00 00 39 00  PNG........IHDR...9.
00000015 00 00 69 08 06 00 00 00 80 96 8b 72 00 00 00 01 73 52 47 42 00  ..i........r....sRGB.
0000002a ae ce  1c e9 00 00 00 06 62 4b 47 44 00  ff  00  ff  00  ff  a0 bd a7  ........bKGD.........
```

```
<svg
  xmlns:svg="http://www.w3.org/2000/svg"
  xmlns="http://www.w3.org/2000/svg"
  width="744.09448819"
  height="1052.3622047"
```

```
//Middle Element Fill
CGContextBeginPath(context);
CGContextSetFillColor(context,
CGColorGetComponents(
[UIColor colorWithRed:1 green:1 blue:1 alpha:1].
CGColor));
CGContextMoveToPoint(context, 54,112);
CGContextAddLineToPoint(context, 54,350);
CGContextAddLineToPoint(context, 212,350);
CGContextAddLineToPoint(context, 212,112);
CGContextAddLineToPoint(context, 54,112);
CGContextClosePath(context);
CGContextFillPath(context);
CGContextStrokePath(context);
```

Figure 3: Processing step artifacts.

We can summarize the overall process as follows:

- Step 1: The image is analyzed and transformed into a vector based representation if necessary. Depending on the desired quality of the result, this step includes the use of a set of software tools and additional involvement of humans. A company-internal expert from ikangai solutions checks the vector representation of the image and applies corrections or optimizations.
- Step 2: The vectorized image is exported and the data representation is parsed and prepared for the code generation step. Again, human assistance is necessary if the customer opts for a high quality transformation.
- Step 3: In the final step, the data is transformed into Objective-C code with a custom tool of ikangai solutions which creates Objective-C code.

After each segment has been transformed, the results are collected and merged into the final Objective-C code which is sent to the customer. An example for the application of the transformation is shown in Figure 3 along with the processing steps. A graphical user interface (see iSENDu graphical elements) was segmented into several elements and fully converted into Quartz 2D Objective-C code.

3. BUILDING BLOCKS

This section provides an overview of our SOA-based concepts that allow to design, outsource, and manage PFLs partially processed by a crowd (CFLs). As a first step, we discuss the flexible involvement of humans in service-oriented systems. Second, the discovery of human capabilities needs to be supported using widely accepted formats such as enhanced friend-of-a-friend (FOAF) profiles. Notice, the focus of our approach is not to introduce or propose entirely new message formats or standards.

Instead, we focus on the application and extension of well-established standards (e.g., WSDL for describing service interfaces and SOAP-based messages for service-oriented interactions) in crowdsourcing scenarios. Finally, distributed collaboration environments may yield undesirable behavior patterns. Our approach is based on monitoring of interactions to prevent inefficiency.

3.1 SOA-based Human Computation

Many service-oriented architectures comprise software services only. However, more and more collaboration and composition scenarios require interactions between human actors as well as software services. Current tools and platforms offer limited support for human interactions in SOA. We therefore introduced the HPS framework. The aim of the HPS framework is to (i) offer a service registry maintaining information related to human and software services (ii) enhance service-related information by describing human characteristics and capabilities (iii) define interaction patterns using Web services technology so that human actors can efficiently deal with interactions.

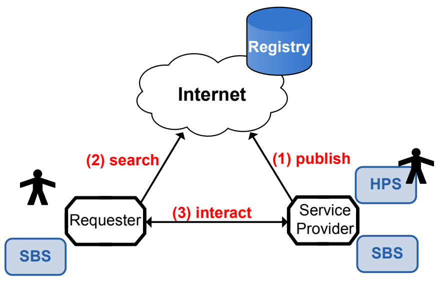


Figure 4: Enhancing SOA with human capabilities.

Human actors in crowdsourcing applications can therefore provide their capabilities and skills in a service-oriented manner. Following the SOA paradigm, three essential steps are performed:

1. **Publish.** Users have the ability to create HPSs and publish the services on the Web using a 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.

2. **Search.** The service requester performs a keyword-based search (reflecting expertise areas) to find Human-Provided or Software-Based Services. Notice, also an HPS can act as a requester as services can be composed recursively (e.g., nested interactions through delegation mechanisms). Ranking is performed to find the most relevant HPS based on, for example, the expertise of the user providing the service. Expertise is determined automatically by the HPS framework through context-sensitive interaction mining techniques.

3. **Interact.** The framework supports automatic user interface generation using XML-Forms technology[8]. Thus, personalized interaction interfaces can be generated and rendered for different devices. The HPS framework can be used for interactions between humans and also for interactions between software services (PFL) and HPSs.

[8]`http://www.w3.org/MarkUp/Forms/`

3.2 Discovery using Social Network Profiles

In crowdsourcing settings, we require knowledge about services which are available in the crowd and how these services can be discovered. Moreover, we need support for the application of interaction patterns between members of the crowd. For example, the delegation of a task like Objective-C code optimization requires knowledge about other persons (services) in the crowd which are able to offer this kind of expertise. We propose to use the SOAF framework [17] as the technical foundation to create links between persons and services in crowds in a machine readable form. Based on the FOAF framework [5], SOAF extends FOAF with regard to SOA principles. SOAF provides for basic registration and discovery mechanisms of services, persons respectively, and supports a rich set of meta information (e.g., type of service provisioning, past service use). Consequently, we emphasize the linking between entities in a SOAF network, a crowd respectively. In contrast to linking on the Internet, where a link is unidirectional and contains very limited information, SOAF models connections as bidirectional links, providing for mutual knowledge between SOAF entities. This reflects (emerging) organizational structures in crowds, where persons know each other and the services that are provided and used.

3.3 Collaboration Patterns and Behavior

Crowdsourcing and similar fragmented environments support novel opportunities of **distributed collaboration**. In such environments, one does not only encounter simple and straight forward chains of task processing within *flat structures*. The compound of participants with specific knowledge in crowds renders a broader spread of tasks possible. However, a large distribution renders it also difficult to assure the quality of the results. Therefore, in crowds members usually assume different roles in a *hierarchical structure* in which a smaller number organizes the assignment of tasks and collects and evaluates the results.

Hierarchical structures permit various types of collaboration patterns. A dominant pattern is the *parallel processing pattern*. Two or more crowd members process a portion of a task concurrently. This pattern is only feasible if a task can be partitioned in subtasks. To ensure a satisfactory final result, in parallel task processing there is usually a further crowd member involved who manages the appropriate partitioning into subtasks and the distribution of the subtasks. We identified this pattern of collaboration as the *delegation pattern* [11, 15] to compensate load distribution problems. Furthermore, after such a split of work it is usually necessary to monitor the progress of the individual subtasks and to join the results into one final result. This collaboration pattern is the *join pattern*. Delegation and join patterns are prominent in current crowdsourcing platforms. Both competition related and marketplace oriented crowd structures depend on a broad distribution of tasks and a final integration and evaluation of the results.

To manage the crowd it is important to track the collaboration paths and identify the capabilities of crowd members. Crowds issue a dynamic behavior and this information is subject to changes over time. Only online monitoring techniques allow to maintain an accurate image of the crowd, find the required resources, and distribute the tasks according to users' expertise profiles.

4. DESIGN AND IMPLEMENTATION

This section starts with a layered overview of the *Crowd Management Framework* comprising all features to model and host a mixed service-oriented crowd environment. The additional sections give detailed information on the layers' layout and provide samples of implementation.

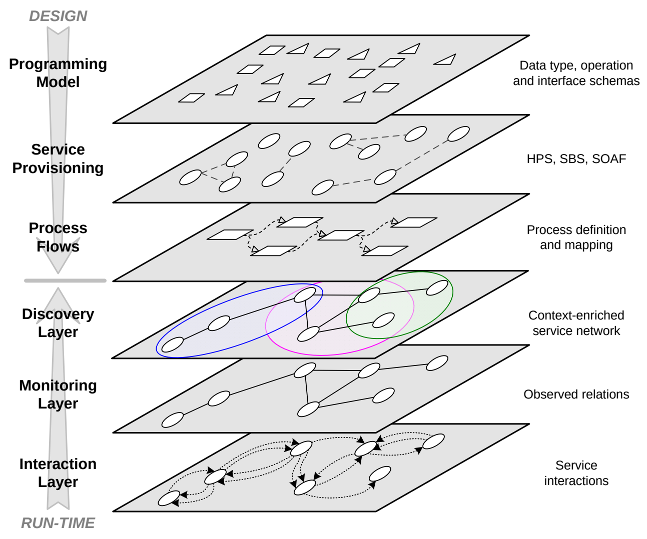


Figure 5: Layered crowd management framework.

The top three layers realize the design phase of the environment with models, services and deployable processes. The *Programming Model* supplies components and relations to model the services for the *Service Provisioning* layer. This layer provides information on the available services which can then be mapped to the process steps modeled in the *Process Flows* layer. At runtime, the services interact on the *Interaction Layer* to complete the assigned process. Interactions must be monitored (*Monitoring Layer*) for crowd management. Management includes to dissolve relations and competence fields that can be discovered and used to adapt flows. Discovery is provided by the *Discovery Layer*.

4.1 Structuring the Crowd with SOAF

By using SOAF as datamodel to structure the crowd, we provide support for publishing services. Furthermore, SOAF profiles reflect social network profiles supporting the search for services and the application of behavior patterns [11] between members of the crowd. Using SOAF meta information, we can for example derive recommendations for services or persons that act as providers for HPSs by analyzing the available information that is based on interaction data. In addition, we can create trusted networks through interaction mining [15] by integrating information concerning mutual trust between members of the crowd. Listing 1 gives an example of ikangai solutions, where a person (Christian Scherling) knows another person (Paolo Vuong) which offers an Objective-C Code Optimization service.

```
<foaf:Person rdf:about="http://.../actors.rdf#ScherlingC">
  <foaf:name>Christian Scherling</foaf:name>
  <foaf:firstName>Christian</foaf:firstName>
  <foaf:surname>Scherling</foaf:surname>
  <foaf:interest dc:title="image QA"
      rdf:resource="http://.../interests.rdf"/>
  <foaf:knows rdf:parseType="Collection">
    <foaf:Person rdf:about="http://.../actors.rdf#VoungP">
      <foaf:name>Paolo Vuong</foaf:name>
      <foaf:firstName>Paolo</foaf:firstName>
      <foaf:surname>Vuong</foaf:surname>
      <foaf:interest dc:title="code QA"
          rdf:resource="http://.../interests.rdf"/>
      <foaf:knows>
        <soaf:Connection>
          <soaf:established>August 23rd 2010</soaf:established>
          <soaf:active>true</soaf:active>
          <soaf:connectiontype>Continuous</soaf:connectiontype>
          <soaf:provides>
            <soaf:Service>
              <foaf:name>CodeOptimizer</foaf:name>
              <soaf:endpoint>...</ soaf:endpoint>
              <soaf:description>Optimizes objective C code
                </soaf:description>
              <soaf:interface rdf:resource="..."/>
              <soaf:active>true</soaf:active>
              <soaf:version>1.0</ soaf:version>
              <soaf:skill>high</ soaf:skill>
            </soaf:Service>
          </soaf:provides>
        <soaf:Connection>
      </foaf:knows>
    </foaf:Person>
  </foaf:knows>
</foaf:Person>
```

Listing 1: Sample SOAF specification.

Notice, these profiles are based on a person's view on the capabilities (services) of another person. This is based on the FOAF `<foaf:knows>` relation. For example, how one person perceives the offered capabilities and available expertise of another person (`interest` and `skill` related tags).

4.2 Hosting Environment and Deployment

The hosting environment is provided by the Genesis2 [7] framework (in short, G2). Originally designed to model Web-service testbeds, it offers its own programming model and language, an extension of the Groovy[9] script language with additional keywords and structures. These extensions represent conveniently adjustable basic elements for a service environment including models for hosts, services and their operations, registries, logging facilities, and so forth. In our overview figure G2 covers the parts of service modeling on the *Programming Model* layer, the *Service Provisioning* layer and provides the means for service deployment, interactions, runtime logging and adaptation.

4.3 WS Environment Specification

Listing 2 demonstrates an example G2 deployment script. At the beginning a model of a service is defined which is deployed and instantiated on a host at the end.

As the service represents a communication interface for a crowd member, first of all, datatypes for the proxy are imported by the `datatype` command. With reference to our scenario in Section 2 these are the image describing prop-

[9]`http://groovy.codehaus.org/`

```
 1  def imgType=datatype.create("file.xsd","ImgDscr") // xsd import
 2  def cmpType=datatype.create("file.xsd","CmpDscr") // xsd import
 3
 4  def srv=webservice.build {
 5    // create crowd web service
 6    HAIP(binding:"doc,lit", namespace="http://...") {
 7      actQueue = [] //queue of activities
 8      friends = [] //list of members
 9      id //unique id of service owner
10
11      addActivity(img:imgType, cmp:cmpType, dl:duration,
12        response:int) {
13        def actId = genActivityId()
14        if (examine(img,cmp,dl) && delegate(img,cmp,dl)) {
15          actQueue+=input
16          return actId
17        }
18        else −1
19      }
20      //delegation strategy
21      dStrat={ img,cmp,dl −> friends[0].addActivity(...)}
22      examine={ img −> ...} //capability match
23      getStatus(actId:int, response:String) {
24        return getStatusOn(actId)
25      }
26    }
27  }[0]
28
29  def li=callinterceptor.create() // logging interceptor
30  li.hooks=[in:"RECEIVE", out :"PRE_STREAM"] // bind to phases
31  li.code={ctx −>... } // process msg
32
33  def srv.interceptors+=li // attach monitoring interceptor
34
35  def h=host.create("somehost",8181) // create host for service
36  srv.deployAt(h) // deploy service at remote host
```

Listing 2: WS environment specification.

erties (`imgType`) and the estimated complexity parameter (`cmpType`). The deadline is mapped to the G2 `duration` type, thus, not imported. The service itself includes two operations. The operation `addActivity()` has as parameters the input provided by a delegation and returns an activity identifier (`actId`) if the requested task is accepted. Therefore, in Line 12 the crowd member decides, first, if she/he considers the offer (`examine`) given the requirements, and second, if a delegation (`delegation`) is possible. The second operation `getStatus()` allows to query the progress on the image transformation task. The final statements deploy the service instances to an environment's host. One particular aspect of G2 is that it allows to intercept message flows. A sample interceptor is defined from Line 29 to 33. The intercepted messages can be analyzed with respect to the environment's requirements.

4.4 Behavior Monitoring and Adaptation

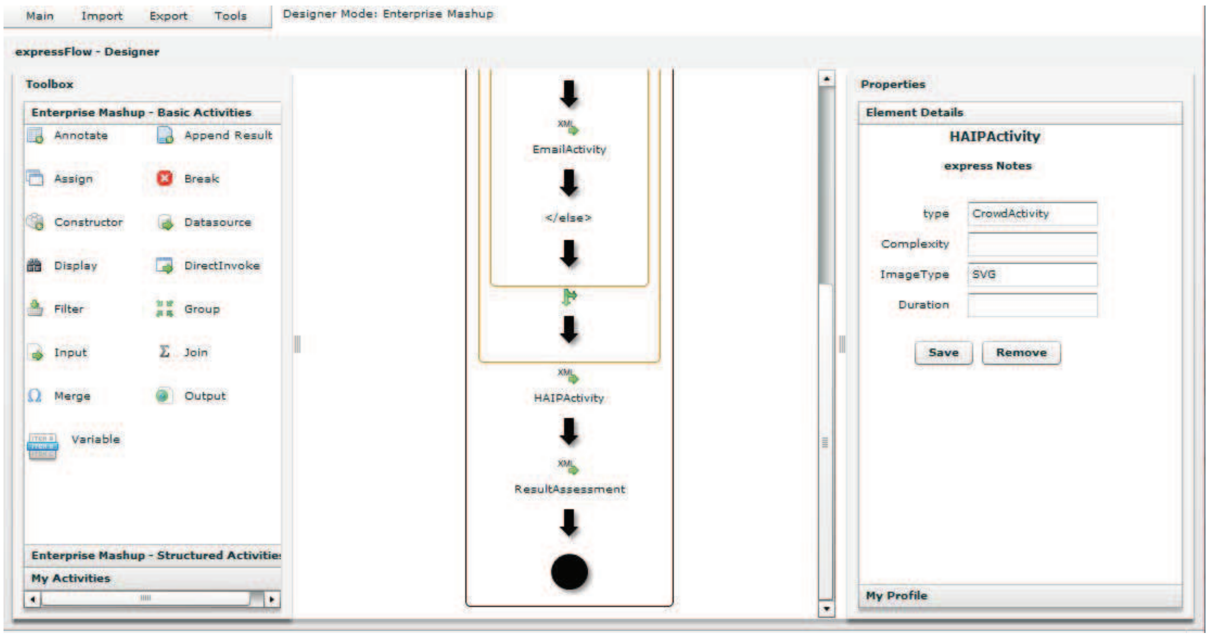
Behavior monitoring is a more specific type of interaction monitoring. It uses information from intercepted messages to infer a more abstract view on actors and their behavior. Extracted behavior patterns allow to better understand the crowd's collaboration paths and structures.

As discussed in the previous sections about collaboration patterns and SOAF, network structures are established dynamically by the developing relations between the crowd's members. Parts of major challenges in crowdsourcing are closely related to challenges in the domain of collaboration management. These composition of individuals with different capabilities, interests, and working styles exposes unpredictable behavior that interferes with the system performance, i.e., causes delays and partly incomplete tasks especially after delegations. A lack of or preference for a particular knowledge source leads to an overload of certain members of the crowd and an imbalanced utilization of the available worker resources. The questions that need to be answered are: How many tasks should be delegated to the same member in a certain time frame? How many task can a community member accept without neglecting other work?

In order to establish a balanced network, it is necessary to provide a crowd's hosting environment with logging and adaptation facilities as described in [10]. Misbehavior must be recognized early enough to deploy counter actions. In previous work [11] we defined two particular related **misbehavior patterns**, *delegation factory* and *delegation sink*. A crowd member exposes a *delegation factory* behavior if she/he accepts large numbers of tasks, however, delegates most and favors, thus overloads, one distinct associate. A crowd member becomes a *delegation sink* when she/he tries to acquire more tasks than processable with the exception that she/he has no possibility to re-delegate the tasks. Sink behavior is typical for new crowd members when they try to quickly increase their reputation by promising to work on

```
<wsdl:definitions name="HAIPService" ...>
<wsdl:types>
<xs:schema elementFormDefault="unqualified"
      targetNamespace="http://...">
<xs:element name="addActivity" type="tns:addActivity" />
<xs:element name="getStatus" type="tns:getStatus" />
<!−− responses omitted−−>
<xs:complexType name="ImgDscr">
<!−− ... −−>
<xs:element name="mimeType" type="xs:string" />
<xs:element name="size" type="xs:string" />
<xs:element name="uri" type="xs:string" />
<!−− other types... −−>
</xs:schema>
  </wsdl:types>
  <wsdl:message name="addActivity">
    <wsdl:part element="tns:addActivity" name="parameters">
    </wsdl:part>
  </wsdl:message>
  <!−− messages... −−>
  <wsdl:portType name="HAIP">
    <wsdl:operation name="getStatus">
     <!−− in−/output... −−>
    </wsdl:operation>
    <wsdl:operation name="addActivity">
     <!−− in−/output... −−>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="HAIPServiceSoapBinding"
      type="tns:HAIPService">
    <soap:binding style="document" transport="http://schemas..."/>
     <!−− operations... −−>
  </wsdl:binding>
  <wsdl:service name="HAIPService">
    <wsdl:port binding="tns:HAIPServiceSoapBinding"
       name="HAIPServicePort">
      <soap:address location="http://somehost:8080/..."/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

Listing 3: WSDL definition.



type: CrowdActivity
Complexity:
ImageType: SVG
Duration:

Figure 6: ExpressFlow online tool for PFL design.

many tasks. Our solutions in [11] suggest to select a non-intrusive approach to the necessary **behavior adaptations**. This means that environments involving humans as services need different adaptations compared to systems consisting of SBS only. Humans themselves can not be 'adapted' but instead we limit the effects of misbehavior by regulating the interaction options, delegation possibilities respectively, between the members.

The script in Listing 4 demonstrates how our G2 hosting environment updates a relation between a member and a friend by removing the friend's id from the service's friend-collection (c.f. Listing 2). The simple script defines an update array (`updateArr`) with `memberId` and the `friendId`. The G2 `webservice` element not only allows to create services but also to alter the services with closures[10]. The first (Line 3) finds the update candidate. The second (Line 3 to 5)removes the relation between the service and the friend-service.

```
1  // remove a link
2  def updateArr = [memberId, friendId]
3  webservice(it:updateArr) {s −> it[0] in s.id} { s−>
4          s.friend =− it[1]
5  }
```

Listing 4: Groovy script for link-based social network adaptation.

4.5 Designing Process Flows

PFLs can be designed using the Web-based ExpressFlow[11] editor. The editor, as shown by Figure 6 has three panes: the left pane shows a toolbox with a set of basic activities. The middle pane shows the designed PFL consisting of different kinds of activities and structures to control the flow (branches, conditions, etc.). The right pane details, for example, specific types of activities.

In this example, the `HAIPActivity` that is part of the PFL should be processed by the crowd. The three parameters of the activity (as discussed previously) are *Complexity*, *ImageType*, and *Duration*. These parameters are initialized during runtime through the execution context of the process.

The editor supports the generation of enterprise mashup code[12] (EMML) that can be deployed in an execution platform. The `HAIPActivity` is a container for *actions*. The action is, for example, a `<directinvoke>` as specified by EMML to invoke services (see Listing 5).

```
<directinvoke endpoint="$serviceURL" outputvariable="$actId"
  method="post" requestbody="$HAIPRequest" />
```

Listing 5: Service invoke example.

Before invoking the service, the corresponding SOAP envelop must be prepared by inserting the invocation parameters (based on the PFL's execution context) as shown in Listing 6.

5. RELATED WORK

Human provided services [14] close the gap between SBS and humans desiring to provide their skills and expertise in a service-oriented manner. In business environments (typically *closed* systems), human-based process activities (see BPEL4People [2]) and human tasks [3] can be modeled in a standardized manner. These standards, however, demand for a precise definition of roles and interaction models between humans and services. The application of such models is therefore limited in crowdsourcing scenarios since

[10]A groovy closure is a reusable 'code block'.
[11]`http://expressflow.com/`
[12]`http://www.openmashup.org/omadocs/v1.0/emml/`

```
<constructor outputvariable="HAIPRequest">
<soap:Envelope xmlns:soap="http://.../soap/envelope/">
  <soap:Body>
    <ns2:addActivity xmlns:ns2="http://.../G2/generated/HAIP">
      <img>
        <mimeType>image/png</mimeType>
        <size>640x480</size>
        <uri>http://somerepository/path/to/file</uri>
      </img>
      <cmp>
        <type>code optimization</type>
        <degree>high</degree>
      </cmp>
      <dl>P1DT0H0M0S</dl>
    </ns2:addActivity>
  </soap:Body>
</soap:Envelope>
</constructor>
```

Listing 6: Sample SOAP envelope.

interaction flows in open and dynamic environments typically emerge at runtime. Adaptations are required due to the complexity of human tasks, people's individual understanding, and unpredictable events. In Web-based systems, users share their expertise [19] or offer their expertise by helping other users in forums or answer communities [1]. Our approach is based on well-established standards such as WSDL for defining interfaces for HPS and FOAF-based social network profiles. The availability of rich and plentiful data on human interactions in **social networks** has closed an important loop [8], allowing one to model social phenomena and to use these models in the design of new computing applications such as crowdsourcing techniques [4]. Semantic Web service communities as introduced by [9] foster the creation of structured communities with predefined community interfaces and functionality. However, ontology structures are not well suited for crowds, because crowd structures emerge bottom up and are difficult to capture with regard to functionality and interactions between crowd members. The required flexibility induces even more unpredictable system properties responsible for various faults. Two main research directions on **self-adaptive properties** emerged in the past years. One initiated by IBM and presented by the research of autonomic computing [16] and the other manifested by the research on self-adaptive systems [12]. While autonomic computing includes research on all possible system layers and an alignment of self-* properties to all available system parts, self-adaptive system research pursuits a more global and general approach. In this work, we considered self-adaptation with the *human in the loop*.

6. CONCLUSION AND FUTURE WORK

In this paper, we proposed a novel approach for integrating human capabilities in enterprise process flows. We focused on crowdsourcing applications by discussing a real world example. Our approach is based on both top-down process modeling (PFL) and emerging interactions in the crowd (CFL). We discussed fundamental problems and building blocks supported by service-oriented architecture. While traditional SOA focuses on the discovery of software-based services, we extend this notion with Human-Provided Services. In flexible interaction environments, one must consider social preferences and behavior patterns. Thus, monitoring and adaptation is essential to keep the system within acceptable states.

Our future work will consider more elaborated process flows detailing various steps. Also, we are currently working on the integration of SLA (service level agreement) frameworks to model human quality attributes in mixed systems. Detailed results of case studies in real crowdsourcing environments will be published in future work.

7. REFERENCES

[1] E. Agichtein, C. Castillo, D. Donato, A. Gionis, and G. Mishne. Finding high-quality content in social media. In *WSDM*, pages 183–194. ACM, 2008.

[2] A. Agrawal et al. WS-BPEL extension for people (BPEL4People), version 1.0., 2007.

[3] M. Amend et al. Web services human task (WS-HumanTask), version 1.0., 2007.

[4] D. Brabham. Crowdsourcing as a model for problem solving: an introduction and cases. *Convergence*, 14(1):75, 2008.

[5] D. Brickley and L. Miller. FOAF vocabulary specification 0.98, Aug. 2010.

[6] L. Eikvil, K. Aas, and H. Koren. Tools for interactive map conversion and vectorization. In *ICDAR*, pages 927–930, 1995.

[7] L. Juszczyk and S. Dustdar. Script-based generation of dynamic testbeds for soa. In *ICWS*, 2010.

[8] J. Kleinberg. The convergence of social and technological networks. *Commun. ACM*, 51(11):66–72, 2008.

[9] B. Medjahed and A. Bouguettaya. A dynamic foundational architecture for semantic web services. *Distributed and Parallel Databases*, 17(179–206), 2005.

[10] H. Psaier, L. Juszczyk, F. Skopik, D. Schall, and S. Dustdar. Runtime Behavior Monitoring and Self-Adaptation in Service-Oriented Systems. In *SASO*. IEEE, 2010.

[11] H. Psaier, F. Skopik, D. Schall, and S. Dustdar. Behavior monitoring in self-healing service-oriented systems. In *COMPSAC*. IEEE, 2010.

[12] M. Salehie and L. Tahvildari. Self-adaptive software: landscape and research challenges. *ACM Trans. Auton. Adapt. Syst.*, 4(2):1–42, 2009.

[13] D. Schall, S. Dustdar, and M. B. Blake. Programming human and software-based web services. *Computer*, 43:82–85, 2010.

[14] D. Schall, H.-L. Truong, and S. Dustdar. Human and software services in web-scale collaborations. *IEEE Internet Computing*, 12(3):62–68, 2008.

[15] F. Skopik, D. Schall, and S. Dustdar. Modeling and mining of dynamic trust in complex service-oriented systems. *Information Systems*, 35:735–757, Nov. 2010.

[16] R. Sterritt. Autonomic computing. *ISSE*, 1(1):79–88, 2005.

[17] M. Treiber, H.-L. Truong, and S. Dustdar. SOAF – design and implementation of a service-enriched social network. In *ICWE*, pages 379–393, 2009.

[18] M. Vukovic. Crowdsourcing for enterprises. In *IEEE Congress on Services*, pages 686–692. IEEE, 2009.

[19] J. Yang, L. Adamic, and M. Ackerman. Competing to share expertise: the taskcn knowledge sharing community. In *ICWSM*, 2008.