Adaptive Provisioning of Human Expertise in Service-oriented Systems

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

ABSTRACT

Web-based collaborations have become essential in today's business environments. Due to the availability of various SOA frameworks, Web services emerged as the de facto technology to realize flexible compositions of services. While most existing work focuses on the discovery and composition of software based services, we highlight concepts for a people-centric Web. Knowledge-intensive environments clearly demand for provisioning of human expertise along with sharing of computing resources or business data through software-based services. To address these challenges, we introduce an adaptive approach allowing humans to provide their expertise through services using SOA standards, such as WSDL and SOAP. The seamless integration of humans in the SOA loop triggers numerous social implications, such as evolving expertise and drifting interests of human service providers. Here we propose a framework that is based on interaction monitoring techniques enabling adaptations in SOA-based socio-technical systems.

Categories and Subject Descriptors

C.2.4 [**Distributed Systems**]: Distributed Applications; H.3.5 [**Online Information Services**]: Web-based Services; H.4 [**Information Systems Applications**]: Miscellaneous

General Terms

Design, Human Factors, Management, Performance

Keywords

Human Expertise Provisioning, Service Adaptation

1. INTRODUCTION

The demand for models to support larger-scale flexible collaborations has led to an increasing research interest in adaptation techniques to enable and optimize interactions between collaboration partners. Such ecosystems comprising people and services that interact in different organizational units are difficult to model in a top-down manner. Challenges include, for example, changing interests and expertise of people, evolving interaction patterns due to dynamically changing roles of collaboration partners, or evolving community structures.

Web services enable loosely-coupled cross-organizational collaborations. In particular, they provide the means to specify well-defined interfaces and let customers and collaboration partners use an organization's resources through dedicated operations. However, offered resources are not restricted to information and software-based services. Also *human expertise* can be provided in a service-oriented manner. For that purpose, the Human-Provided Services (HPS) Framework [14] enables human participation in a SOA environment. A typical example is a document translation service [15] that could be implemented in software too, but mostly only with insufficient quality. HPS allows humans to provide translation services in the same manner by letting them receive and process requests through Web service interfaces. With the human in the loop, traditional service-oriented architectures (SOA) transform from pure technical systems into socio-technical systems [5]. These systems are characterized by both technical and human/social aspects that are tightly bound and interconnected. The technical aspects are very similar to traditional SOAs, including facilities to deploy, register and discover services, as well as to support flexible interactions. Additionally, the social system includes people and their habitual attitudes, values, behavioral styles and relationships. In particular, considering drifting interests of people, evolving skills, and varying collaboration incentives requires enhanced technical infrastructures in terms of flexibility and adaptability. Due to the support of loose coupling, sophisticated discovery mechanisms, and dynamic binding, Web services and SOA deem to be the ideal technical framework to realize large-scale socio-technical systems on the Web. We call the mix of software services and humans interacting on the Web a *Mixed Service-oriented System*.

The foundational pillars of such mixed systems utilized in this paper are as follows: (i) *Human-Provided Services.* We discuss the HPS concept letting people participate in pure service-oriented environments. People reflect their ability and willingness to contribute by defining and offering their own services using state-of-the-art SOA techniques. (ii) *Flexible Interaction Models.* Interaction monitoring and mining is applied to determine the behavior of services and

SAC'11 March 21-25, 2011, TaiChung, Taiwan.

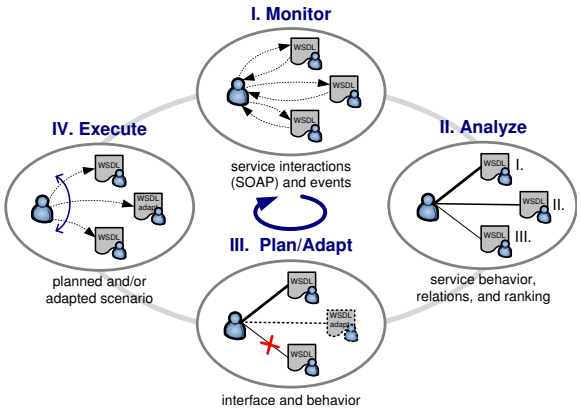


Figure 1: Adaptive run-time provisioning.

user relations. Detecting behavior and relation changes is the basis for effective service adaptations. (iii) *Adaptive Service Infrastructure.* We discuss the need for run-time adaptation. In particular, we do not only adapt service behavior, but also provided service features at run-time.

Approach Outline. Figure 1 depicts the overall approach to *flexible run-time provisioning of human expertise.* In the monitoring phase, service interactions, i.e, SOAP messages, and major system events, such as service updates, are captured. All interactions are annotated with tags and keywords to categorize requests. Then, this data is analyzed to learn about the service behavior in terms of reliability and dependability that is described by various interaction metrics [16]. Relations between clients and services are established based on these calculated ranking metrics. The actual analysis is context-aware, e.g., considers message tags to determine service behavior with respect to expertise areas. In the planning and adaptation phase respectively, services are rewarded and punished for their behavior. Capabilities influence the future provisioning of particular service operations. In the execution phase future service discovery and usage is influenced by adaptations to achieve optimal expertise provisioning in SOA.

Contributions. This paper aims at addressing the following technical challenges found in mixed systems by applying Web services technologies and social network concepts:

- *Service Avatar.* This concept is used to represent human capabilities as services on the Web. A combination of WSDL and FOAF elements describe functional and non-functional properties.
- *Personal Provisioning.* Social aspects require personalized service provisioning by establishing peer-to-peer relations between clients and service providers on demand.
- *Feedback-based Adaptation.* Observing and analyzing annotated SOAP interactions enable context-aware customization of personal services.

The remainder of this paper is organized as follows. Section 2 introduces the concept of *avatars* and its application to provide human expertise in SOA. Section 3 covers implementation details of our adaptation infrastructure and techniques. We run a small-scale simulation and discuss evaluation results in Section 4. Section 5 deals with related work and Section 6 concludes the paper.

2. HUMAN INVOLVEMENT IN SOA

We start with discussing the concept of avatars on the Web and the realization using Web service standards.

2.1 Avatars on the Web

Avatars are a computer user's representation of himself/herself, e.g., in form of a nickname or icon, in Internet communities. More advanced models further include interests and capabilities, such as in online gaming platforms. This makes an avatar the ideal metaphor to represent humans and their capabilities in service-oriented systems. Furthermore, an avatar does not only represent a human's services in an SOA environment, but can also actively act on behalf of the human it represents. Based on contextual constraints, such as the current load and assigned expertise areas, that software component can automatically categorize or reject requests. This process is configured through policies and rules in advance to shape the behavior of services and unburden the human from frequent but simple decisions. Figure 2 depicts the conceptual overview and explains our notion of avatar.

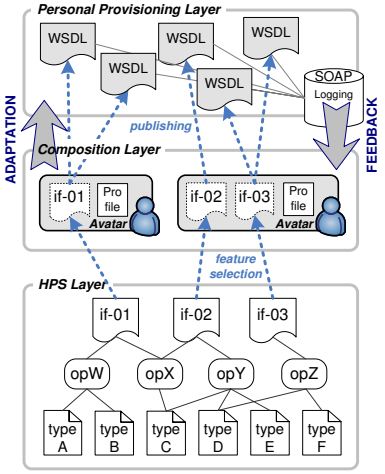


Figure 2: Adaptive human expertise provisioning.

HPS Layer. HPS [14] enhances the traditional SOA-based systems by enabling people to provide services with the very same technology as used by implementations of software-based services (SBS). Various operations for different collaborative activities indicate a provider's ability (and willingness) to participate in ad-hoc as well as process-centric collaborations. The HPS Framework provides predefined data types (XML schemas), operations, and compiled interfaces to provide particular services. The design of services is supported via a Web-based 'toolbox' (graphical user interface) enabling users to create services in a simplified manner. The creation of services does not require any knowledge related to SOA, Web services standards, or SOA runtime aspects. Based on the designed HPS, a script is parameterized to create and deploy corresponding avatars in the Genesis Hosting Environment (detailed in the following section).

Composition Layer. People who provide their expertise as services on the Web, select the required features, e.g., Web service interfaces to interact with clients, and compose them, thus, predefine the capabilities of the instances managed by their avatars. While these initial decisions represent the rather static properties of an avatar, personal profiles

(modeled as FOAF[1]) are periodically updated by our system to reflect social aspects, such as interests, interaction behavior and provided service quality. Together these static and dynamic properties characterize the avatar. The link between situation dependent profiles and composition decisions of the owner define the avatar's current providable instances. A high current load of the owner, for example, must not only update the current profile but also influence the avatars deployment strategy. Furthermore, to propagate the current situation to its instances the avatar provides instances with a connection to the current profile's state.

Personal Provisioning Layer. Clients discover avatars by accounting for (i) *functional properties (FPs)*, i.e., the type of supported interfaces, and (ii) *non-functional properties (NFPs)*, i.e., social aspects. Here, **for each single client an own service instance is deployed** (peer-to-peer style). Clearly, humans providing services cannot serve thousands of concurrent requests as software services do. However, publishing dedicated instances enables our system to personalize them gradually for each individual client that has a long-term contract with the corresponding avatar.

2.2 On-demand Creation and Deployment

The concept of personalized provisioning is enabled by creating dedicated service instances for each single customer of service providers. A standard service is instantiated (derived from the avatar) and gradually customized according to a client's requirements and a provider's behavior.

```
1  def profile = profilePlgIn.connect(); //current profile
2  def Language=datatype.create("tconf.xsd","langTypeA") //imports
3  def Status=datatype.create("tconf.xsd","statType")
4  def i=callinterceptor.create() //interaction logging
5  i.hooks=[in:"RECEIVE", out :"PRE_STREAM"] //hooks on streams
6  i.code={ m -> ...} //logged message
7
8  def arrSrv=webservice.build {//interface definition
9    // create web service
10   TranslationService(binding:"doc,lit", namespace="http://...") {
11     interceptors+=i //attach interceptor
12     docQueue = [:] //current document queue
13     repEP = "" //reporting endpoint
14     // create translateDoc operation, return doc refId
15     translateDoc(docref:String, fromLang:Language,
16                  toLang:Language, response:int) {
17       def refId = genId(docQueue) //new id for doc
18       //active pre-processing with checks
19       if (profile.checkTotalLoad() < LOAD_THR)
20         docQueue.put(refId,docref)
21       return refId
22     }
23     getJobStatus(refId:int, response:Status){
24       return report(refId)
25     }
26     cancelJob(refId:int){
27       docQueue.remove(refId)
28     }
29     setAsyncReportEP(wsdl:String) {
30       repEP = wsdl
31     }
32   }
33 }
34 def srv=arrSrv[0] // only one service declared, take it
35 def h=host.create("somehost:8181") // import back-end host
36 srv.deployAt(h) // deploy service at remote back-end host
```

Listing 1: Service deployment script.

[1]http://xmlns.com/foaf/spec/

Listing 1 displays a Groovy[2] script for the Genesis environment (G2) [9] that allows to create a *Document Translation Service* with the approach shown in Figure 2. The first line connects the script content to the human's current profile via a G2-Plugin (`profile`). In the following two lines the script imports type definitions from the *HPS Layer* (`Language` and `Status` enumeration types). The service definition follows. An array (`arrSrv`) collects the services. In our case only the `TranslationService` is defined as follows. The queue `jobQueue` collects the current jobs of a service instance which resides in the *Personal Provisioning Layer*. Operation `translateDoc` checks the assignment in an exchangeable behavior closure[3], e.g., according to the current overall load of the human determined by the profile. Then the document is moved into the input queue if all checks pass. The operation returns a unique `refId` which enables clients to manage their requests (e.g., request job status). The last three operations implement remote management. The first, `getJobStatus`, provides auto-generated job status reports. The second, `cancelJob`, allows the client to cancel an ongoing operation. The last one enables the client to set a callback endpoint for asynchronous human responses and notifications. The last three statements deploy the service in the G2 environment.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <definitions ...>
3  <types>
4   <schema elementFormDefault="qualified"
5    targetNamespace="http://socsoa.infosys.tuwien.ac.at/">
6    xmlns="http://www.w3.org/2001/XMLSchema">
7   <element name="translateDocRequ">
8    <complexType>
9     <sequence>
10     <element name="document" type="xsd:anyURI" />
11     <element name="fromLang" type="Language" />
12     <element name="toLang" type="Language" />
13    </sequence>
14   </complexType>
15  </element>
16  <simpleType name="Language">
17   <restriction base="xsd:string">
18    <enumeration value="German" />
19    <enumeration value="English" />
20   </restriction>
21  </simpleType>
22  ...
23  </schema>
24 </types>
25 <message name="translateDocRequest">
26  <part name="parameters" element="xsd1:translateDocRequ"/>
27 <message>
28 ...
29 <portType name="TSPortType">
30  <operation name="translateDoc">
31   <input message="tns:translateDocRequest" Action=.../>
32   <output message="tns:translateDocResponse" Action=.../>
33  </operation>
34  <operation name="getJobStatus"> ... </operation>
35  <operation name="cancelJob"> ... </operation>
36  ...
37 </portType>
38 <binding type="tns:TSPortType" name="..."> ... </binding>
39 <service name="TranslationService"> ... </service>
40 </definitions>
```

Listing 2: Document translation WSDL excerpt.

Listing 2 shows an excerpt of a document translation service WSDL, created with the script in Listing 1, that is provided by a human. Besides the `translateDoc` operation

[2]http://groovy.codehaus.org/

[3]A groovy closure is a reusable 'code block'.

for submitting documents (and the omitted but mandatory `setAsynchReportEP` to define the reporting endpoint for notifying about finished jobs), there are further management operations, including `getJobStatus` and `cancelJob`. Complex data types, as shown for `Language`, are used to increase the semantics of the service description, e.g., by providing enumerations of available options.

Traditional service development procedures are clearly insufficient in highly dynamic environments. As human capabilities evolve over time and interests or incentives for offering expertise change, provided operations of an HPS (and their signature) need to be adapted accordingly. For instance, a human providing a document translation service may learn a new language or discontinues the support of rarely requested options. Furthermore, some kind of support might be of low quality and/or not frequently used within a community. Another reason for changing a service's operations is the permanent delegation of responsibilities and balancing of features among a set of services.

2.3 Service Description and Discovery

An adaptive environment requires flexible service description and discovery mechanisms. Thus, before each request the client gathers dynamically compiled metadata on the current functional (FP) and non-functional properties (NFP) to update its view. This is realized by wrapping the HPS's WSDL file and an extended FOAF description into a *WS-Metadata-Exchange*[4] (MEX) document.

```
<mex:Metadata>
<mex:MetadataSection Dialect="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:definitions>
   <!-- Omitted -->
  </wsdl:definitions>
 </mex:MetadataSection>
 <mex:MetadataSection Dialect="http://xmlns.com/foaf/0.1/">
  <rdf:RDF xmlns:foaf = "http://..."
   xmlns:capability = "http://.../capability.owl#">
   <foaf:Person rdf:about="http://www.infosys.../staff/">
    <foaf:name>Harald Psaier</foaf:name>
    <foaf:interest rdf:resource="http://.../hpsaier/interests.rdf"/>
    <!-- Omitted -->
    <capability:op>
     <capability:port id="TSportType">
      <capability:op id="translateDoc">
      <capability:opwsdlxpath>
           wsdl:operation/[@name="TSportType"]
      </capability:opwsdlxpath>
      <capability:opmetricgrounding
           rdf:resource="http://.../grounding-translateDoc.xml"/>
      <capability:opmetric>
       <capability:opmetricid>cost</capability:opmetricid>
       <capability:opmetricvalue>100.0</capability:opmetricvalue>
      </capability:opmetric>
      <capability:opmetric>
       <capability:opmetricid>reliability</capability:opmetricid>
       <capability:opmetricvalue>0.8</capability:opmetricvalue>
      </capability:opmetric>
      ...
      </capability:op>
        </capability:port>
   </foaf:Person>
  </rdf:RDF>
 </mex:MetadataSection>
</mex:Metadata>
```

Listing 3: Dynamically created avatar description.

Basically there are two different reasons for initiating a discovery. First, for discovering an avatar. In that case FPs are of primary interest, combined with *capability metrics* that reflect the overall satisfaction of an avatar's clients. Second, before a request, the potentially adapted profile (including personalized metrics that reflect the avatar's behavior in the past) is retrieved. In both cases, our framework uses SPARQL[5] to define search queries on FOAF structures.

Listing 3 shows the sample response message to a MEX `GET` request. The main response body comprises the currently offered operations in a WSDL (omitted, see Listing 2) and the related NFPs in the second `MetadataSection` in FOAF format. The elements with the `capability` prefix provide the current NFP values for a related operation defined in the WSDL section. In our current implementation, such NFPs are costs and primarily quality metrics, such as an avatar's reliability and responsiveness. The *XPath statement* identifies an operation uniquely. The following metric grounding resource `opmetricgrounding` links a document with metric definitions (meaning, measurement, unit, range of values, etc.) to the listed metric ids. The description for those mined metrics is similar to [11] for modeled QoS.

3. SYSTEM AND SERVICE ADAPTATION

We discuss our approach to adaptive service provisioning by highlighting the fundamental building blocks, and in particular the adaptation of service instances itself.

3.1 Architectural Overview

Figure 3 shows an architectural overview of the whole framework that enables provisioning of human expertise.

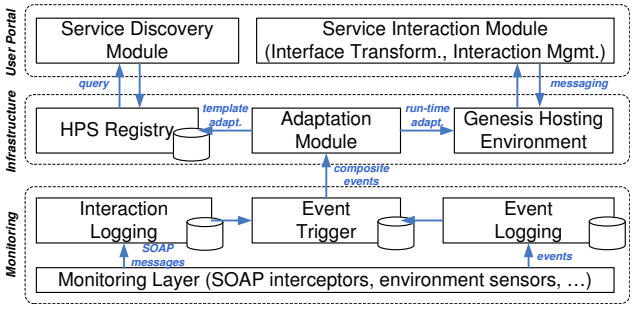


Figure 3: Architectural overview.

The major components are organized in three layers:

- *Monitoring Layer.* SOAP *interactions* and environment *events* are logged and processed. Identified composite events are *triggered* and forwarded to the *adaptation module*.
- *Infrastructure Layer.* The adaptation module checks pre-defined rules to take appropriate steps, i.e., adapting the HPS templates in the HPS registry if a service does not provide sufficient QoS or adapting the deployed services in the *G2 hosting environment* [9], e.g., removing unused or expired operations from a service instance and its WSDL interface (as shown later).
- *User Portal.* Users can discover potential services using the *discovery module*; and interact with particular instances through the *interaction module*. These interactions are logged to trigger future adaptations.

[4] `http://www.w3.org/Submission/WS-MetadataExchange/`

[5] `http://www.w3.org/TR/rdf-sparql-query/`

Parts of this system are described by references, for instance the G2 hosting environment [9], and event triggering based on SOAP monitoring [13]. Therefore, in this paper, we revisit the *interaction monitoring* concept in SOA environments from a technical point of view; deal with *service descriptions* in terms of functional and non-functional properties to support the discovery process; and demonstrate how to enable *run-time adaptations* in the Genesis hosting environment (see also [12]).

3.2 Interaction Monitoring and HPS Profiling

Interaction Model. Avatars are not statically bound to clients but are discovered at run-time. Thus, interactions are ad-hoc and dynamically performed with often not previously known partners. In SOA, interactions are typically modeled as SOAP messages. Besides standard SOAP structures we use various header extensions, such as WS-Addressing[6], temporal properties (timestamps, deadlines), and contextual annotations. The latter are realized through tags/keywords that are assigned to messages to annotate interactions. An excerpt of a typical SOAP request is depicted in Listing 4.

```
<soap:Envelope
 xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
 xmlns:types="http://socsoa.infosys.tuwien.ac.at/Type"
 xmlns:ts="http://socsoa.infosys.tuwien.ac.at/TS">
 <soap:Header>
  <types:timesent value="2010-09-01T15:13:21"/>
  <types:deadline value="2010-09-06T12:00:00"/>
  <types:context tags="Web,Services,SOA,research,paper"/>
  <wsa:MessageID>uuid:722B1240-...</wsa:MessageID>
  <wsa:ReplyTo>http://socsoa..../Actor#Florian</wsa:ReplyTo>
  <wsa:From>http://socsoa..../Actor#Florian</wsa:From>
  <wsa:To>http://socsoa..../Actor#Daniel</wsa:To>
  <wsa:Action>http://socsoa....ac.at/Type/RFS</wsa:Action>
 </soap:Header>
 <soap:Body>
   <!-- applied document translation request -->
   <!-- schema details omitted -->
 </soap:Body>
</soap:Envelope>
```

Listing 4: Simplified SOAP interaction example.

Our system utilizes temporal properties of SOAP calls to infer behavior metrics, such as the average time required to process a request, availability or responsiveness metrics (see [13, 16] for details). As demonstrated in our previous papers, metrics are calculated using the most recent history, and updated with a sliding window approach. Thus, old data ages out automatically. For the sake of simplicity, we only consider simple request-response patterns. A request can be accepted by a service and further processed by the corresponding avatar; or rejected immediately (e.g., due to the lack of free capacities). More complex, long-running interactions consisting of numerous intermediate responses are not in the scope of this paper.

Dynamic Behavior Profiles. Since interests and skills of people regarding their capabilities to process requests from different domains usually widely vary, behavior metrics are context sensitive, i.e., bound to particular expertise areas. Collections of these behavior metrics are used to calculate NFPs and finally, to calculate service capabilities. For instance, someone may be highly rewarded for providing a document translation service while his/her document review service for scientific papers is not highly ranked. Moreover, the document translation service might be successfully used for research papers in computer science, while it is not frequently used to translate business documents. Human skills and expertise evolve over time. Furthermore, interests alter and drift. Thus, our monitoring and mining approach is the key to timely compensation of behavior changes.

[6] http://www.w3.org/Submission/ws-addressing/

3.3 Adaptation Strategies

Various reasons require timely adaptations of services that may affect the whole mixed service-oriented system. In particular, we study:

- *Client-driven interventions* are the means to protect customers from unreliable services. For example, services that miss deadlines or do not respond at all for a longer time are replaced by other more reliable services in future discovery operations.

- *Provider-driven interventions* are desired and initiated by the service owners to shield themselves from malicious clients. For instance, requests of clients performing a denial of service attack by sending multiple requests in relatively short intervals are blocked (instead of processed) by the service.

In general, adaptations can be less or more intrusive. We basically focus on two distinct mechanisms: (i) interface adaptations, and (ii) behavior adaptations. Interface adaptation means that single operations of a service are temporarily or permanently modified or removed. These changes can be triggered by the system due to request overloads or falling capabilities of services (that receive low ratings from clients). Behavior adaptations are more intrusive and alter the behavior of avatars respectively of their deployed service instances (though we cannot alter the behavior of humans it represents). The Genesis framework [9] provides the ideal technical grounding to perform seamless run-time modifications, i.e., without being forced to take a service offline and redeploy it later again.

Client-driven interface adaptation example. Listing 5 demonstrates a typical adaptation desired by clients. In that case we assume that an avatar has missed deadlines several times. The system tries to protect the affected clients by automatically undeploying the `translateDoc` operation of the corresponding service instances (considered as 'lazy service'). Thus, clients can still retrieve the job status of ongoing translation requests, but are not able to send new ones. So, they are urged to discover alternative avatars or at least to negotiate a new contract with the same avatar (not shown here) who would deploy a new dedicated service instance.

```
def lazySrvArr=analysis.getLazyServices()
lazySrvArr.each { lazySrv ->
  webservice(name:lazySrv) { s-> name in s.name} { s->
    def o = s.getOperation("translateDoc") //get operation
    s.deleteOperation(o) //delete operation
    s.redeploy() //redeploy service and wsdl
  }
}
```

Listing 5: Adapt interface and undeploy operation.

Provider-driven behavior adaptation example. Listing 6 demonstrates a typical adaptation desired by providers. In that case we assume an avatar has highly varying working speeds. Thus, in case of request bursts (exceeding predefined thresholds `THR`), the system adapts the acceptance behavior regarding incoming requests before missing any deadlines. For instance, subsequent requests are rejected (or delegated to similar services [16]) instead of being queued for a longer time. The acceptance behavior is modified considering lower and upper queue size limits.

```
1  def busySrvArr=analysis.getBusyServices()
2  busySrvArr.each { busySrv ->
3    webservice(name:busySrv) { s->
4    if ("translateDoc" in s.operations.name && name in s.name)
5      def op=s.operations.grep{o -> o.name == "translateDoc"}[0]
6      op.behavior = {
7          if ( (profile.checkTotalLoad() < LOAD_THR) &&
8            (profile.checkRquFrq() < FRQ_THR)  &&
9            (jobQueue.size() < QUEUE_THR) )
10             docQueue.put(refId,docref)
11         ...
12       }
13     s.redeploy()
14 }{}
```

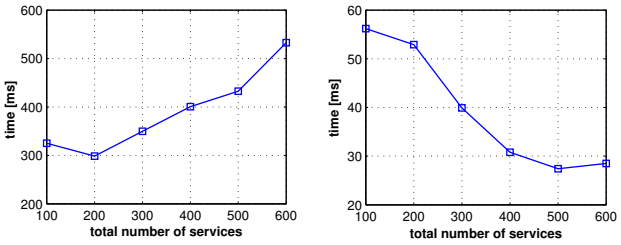
Listing 6: Adapt job acceptance behavior.

4. EVALUATION AND DISCUSSION

We perform a system evaluation in terms of performance and scalability, and functional properties, i.e., the power of the presented adaptation approach.

4.1 Performance and Scalability

We evaluated the performance of our approach regarding the whole adaptation cycle. This loop includes logging of interactions, analyzing and inferring NFPs, evaluating preconfigured triggers (e.g., a service's reliability falls below a lower bound) and performing the actual adaptation. Costs for monitoring, analysis and triggering have been measured and discussed in detail in [16]. Thus, we focus on the actual adaptation in our flexible hosting environment.



(a) service interface update (b) service closure update

Figure 4: Adaptation performance in G2.

We investigate both kinds of adaptation mechanisms, (i) interface adaptation, and (ii) behavior adaptation. Our testbed consists of variable amounts of deployed services (from 100 to 600), and we assume that an adaptation for 20% of the total number of instances is triggered. Figure 4(a) depicts the *average* time in milliseconds that is required to perform both, undeploying an operation of exactly *one* service and redeploying the modified WSDL interface. The scaleable adaptation approach of G2 aggregates change requests and performs modifications in bulks. Thus, for only 100 services in total (i.e., 20 modified instances) the average time is higher then for 200 services, but then rises nearly linearly. Figure 4(b) shows the required time for service behavior adaptations in the same environment. Here, the actual implementation of a single operation is exchanged. Note that again due to bulk modifications the *average* time for adapting one service instance decreases for higher amounts of services (approximately until 500 services). Further note that closure exchanges are approximately 10 times faster than interface adaptations which require a redeployment of the interface (but not of the decoupled underlying service instance).

4.2 Scenario Simulation

We run a small-scale simulation using the *Repast Simphony*[7] simulation toolkit to (i) test the implementation of our framework, and (ii) show the effects of different adaptation strategies. For that purpose we simulate human behavior in terms of reliability, expertise evolvement, unsteady working styles and interest drifts; and show the application of our adaptation approach.

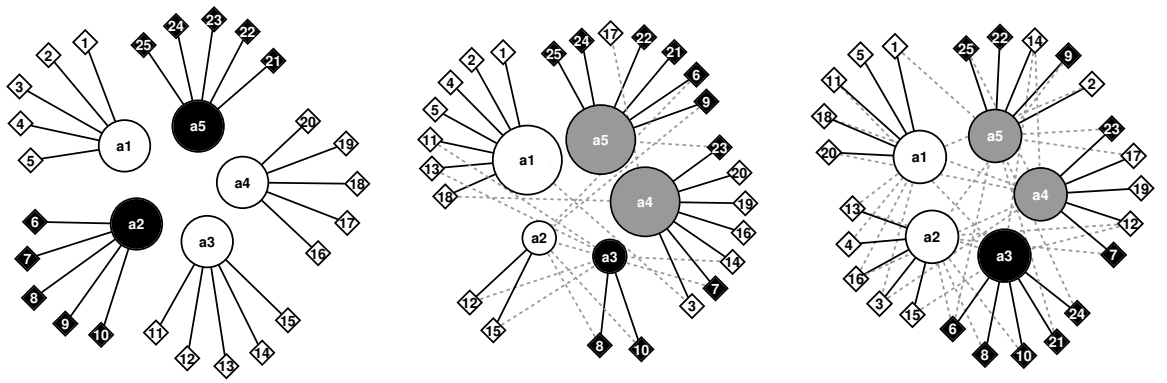
Simulation Setup. The round-based simulation environment consists of 5 avatars (a_1 to a_5) and 25 clients that have different interests and behavior. Clients already have relations to avatars, i.e., there are dedicated service instances deployed for each client. Clients send one request every 10 rounds. An avatar needs between 1 and 2 rounds (random) to process that request. Thus, an avatar can serve an average of 7 concurrent clients. Client and service/avatar interactions are produced by simulated agents while for hosting the services, capturing and analyzing interactions, and performing adaptations our actual framework is utilized.

Experiment Setup. We distinguish between two distinct expertise areas, where each area is described by 10 different tags. Avatars have interest profiles consisting of 5 tags that reflect the types of requests that they are willing to process. Clients send requests that are annotated with up to 3 different tags. Initially each avatar has a clear profile, either in area A (white nodes), or area B (black nodes) – see Figure 5(a). Clients always send requests that match exactly one expertise area (white or black) and avatars accept these requests if they match more than 50% with their profile. In the initial state we have optimal conditions. No avatar is overloaded and they match exactly their clients requirements in terms of expertise.

Experiment Run. Major problems of human roles in technical systems are caused by people's drifting skills, evolving expertise and varying incentives and perception of risks. In short, people do not follow strict specifications such as software components do. In contrast, humans that provide services may change their focus of work. Our service provisioning system is able to tackle this problem by performing appropriate run-time adaptations.

We assume that avatars shift their interests and therefore, their expertise areas. However, the deployed services for long-term clients normally remain the same (e.g., consider a provider having multiple clients, but who specializes on translating documents in new domains). While a_1's interest profile remains unchanged, the other simulated avatars

[7] `http://repast.sourceforge.net`



(a) initial state (b) final state (conservative adapt.) (c) final state (aggressive adapt.)

Figure 5: Evolving service community structures.

(agents) gradually change their profiles to reflect such natural interest shifts. Figure 5 visualizes the setup. The centered circles represent the 5 avatars and the diamond shaped symbols reflect a certain client's *static* requirements, manifested as a service instance that enables interactions between that client and the serving avatar. Comparing Figure 5(a) and 5(b) reveals that a_2 changes from the black to the white area, a_3 does exactly the opposite, and a_4 and a_5 extend their expertise areas to include both black *and* white (represented by gray nodes) expertise. As a consequence avatars refuse requests that no longer match their expertise areas. Furthermore, they register their new capabilities in the centralized service registry. Profile changes are linearly performed in the first 100 simulation rounds. Our system has additional 150 rounds to apply adaptations and to reorganize the network.

In particular, after interest shifts some avatars will not match their clients' requirements, and thus, begin to reject their requests. Our system will undeploy operations of corresponding services that are used to submit new requests and subsequently the whole service instance that connects a client and an avatar exclusively. This forces clients to query for new avatars that deploy new dedicated service instances to interact with their clients. Queries for new avatars account for matching profiles *and* previously reliable behavior (i.e., request success rate). Furthermore, based on changed interest profiles, the acceptance behavior of services is modified so that the serving avatar gets requests that match his/her new work area(s). Let us define the notion of *success rate sr* as the amount of successfully served requests in percent of one client-avatar relation, and *global success rate gsr* as the average of all single success rates. We use these metrics to measure the efficiency of applied adaptations. Note, success rates decrease if avatars attract too many clients and as a consequence become overloaded.

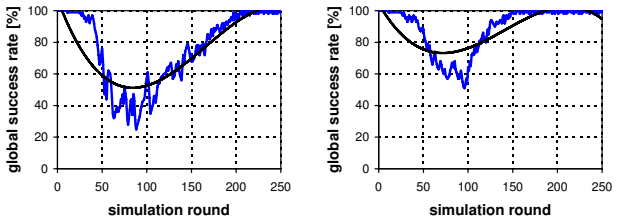
We demonstrate the impact of two fundamentally different adaptation strategies, (i) a conservative and even more tolerant strategy, and (ii) an aggressive strategy:

Conservative Strategy: The system collects multiple consecutive failures and violations of the clients interaction policies. In our simulation, clients are forced to stop interacting with an avatar if more than 5 requests are unreplied in less than 25 simulation rounds (interface adaptation due to dropping success rate). The client's memory has only a depth of 25 rounds. Thus, with that strategy, clients are considered more tolerant, they forgive short-time unreliability, e.g., caused by temporal work overloads of avatars and stay as long as possible with the same service provider.

Aggressive Adaptation Strategy: The system urges the clients to change their service providers (i.e., avatars) after the first triggered misbehavior of avatars. This strategy is more dynamic than the conservative one.

Experiment Results. Figure 5(b) visualizes the resulting network for the conservative strategy, and Figure 5(c) for the aggressive one. Solid lines represent active relations to avatars, while dashed lines visualize earlier relations. For instance, in both cases client 15 changed from using services from avatar a_3 to avatar a_2 according to their interest shifts. Obviously – and as expected – the aggressive approach triggers significantly larger numbers of adaptations compared to the conservative one.

Regarding the conservative strategy, clients change avatars rarely and only if the success rate of a serving avatar drops significantly and does not recover within 25 rounds. Thus, load on providers is unequally distributed (see Figure 5(b)). As shown in Figure 6(a), interest drifts cause longer adaptation cycles until the whole system returns to a steady state. The *gsr* sharply drops and begins to recover at round 100 (when all profile changes are finished). Note, the depth of the decrease highly depends on the tolerance of the system and clients towards unreliable avatars; i.e., if less failures and misbehavior are tolerated adaptations are triggered earlier which guarantees a higher global success rate.



(a) *gsr* (conservative adapt.) (b) *gsr* (aggressive adapt.)

Figure 6: Global success rate (and approximation).

The aggressive approach requires much more system interventions, e.g., service adaptations and (re-)deployments. The costs (compare performance evaluation before) have to be considered with respect to the overall size of the environment. However, the *sr* can be kept much higher during

the adaptation phase. Furthermore, the adaptation phase is shorter compared to a more relaxed adaptation approach, and a nearly equal distribution of load is reached (see Figure 5(c) where each avatar serves approximately the same amount of clients).

5. RELATED WORK

Major software vendors have been working on standards addressing the lack of human interaction support in service-oriented systems. WS-HumanTask [2] and Bpel4People [1] were released to address the emergent need for human interactions in business processes. While Bpel4People based applications focus on top-down modeling of business processes, *mixed service-oriented systems* target flexible interactions and compositions of Human-Provided and Software-Based Services [14]. This approach is aligned with the vision of the Web 2.0, where people can actively contribute services. In such networks, humans may participate and provide services in a uniform way by using the HPS framework [14]. The basic vision of a hybrid human-computer document translation system has been discussed by [15], however, not focusing on the realization as a service-based system.

Enhanced flexibility of complex systems is introduced by establishing a cycle that feeds back environmental conditions to allow the system to adapt its behavior. The MAPE cycle [7, 8] is considered as one of the core mechanisms to achieve adaptability through self-* properties. Based on the observed context of the environment, different adaptation strategies can be applied [6] to guide interactions between actors, the parameters of those strategies, and actions to prevent inefficient use of resources and disruptions. While autonomic computing allows for autonomous elements and applies these principles to distributed systems, current research efforts leave the human element outside the loop. The availability of rich and plentiful data on human interactions in social networks has closed an important loop [10], that allows to model social phenomena and to use these models in the design of new computing applications such as crowdsourcing techniques [3]. In the context of multi agent systems (MAS), self-configuring social techniques were introduced in [4]. A major challenge in adaptation and self-configuration is to dynamically find the most relevant adaptation parameter. Research relevant to this issue can be found in [17].

6. CONCLUSION AND FUTURE WORK

In this paper we motivated the trend towards socio-technical systems in SOA. In such environments social implications must be handled properly. With the human user in the loop numerous concepts, including personalization, expertise involvement, drifting interests, and social dynamics become of paramount importance. Therefore, we discussed related Web standards and showed ways to extend them to fit the requirements of a people-centric Web. In particular, we outlined concepts that let people offer their expertise in a service-oriented manner and covered the deployment, discovery and selection of Human-Provided Services. In the future, we aim at providing more fine-grained monitoring and adaptation strategies. An example is the translation service presented in this paper, where some language options are typically used more often, or even more successfully than others. In that case, data types could be modified to reduce the number of available language options in the WSDL interface description and to restrict input parameters. Harnessing delegation patterns that involve various participants, a complex social network perspective is established in which connections are not only maintained between one client and an avatar, but also among avatars.

Acknowledgments

This work is supported by the EU through the FP7 projects COMPAS (No. ICT-2008-215175) and S-Cube (No. ICT-2008-215483). The authors thank Lukasz Juszczyk for providing the Genesis2 framework and supporting this work.

7. REFERENCES

[1] A. Agrawal et al. Ws-bpel extension for people (bpel4people), version 1.0., 2007.

[2] M. Amend et al. Web services human task (ws-humantask), version 1.0., 2007.

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

[4] V. Bryl and P. Giorgini. Self-configuring socio-technical systems: Redesign at runtime. *Int'l Trans. on Syst. Science and App.*, 2(1):31–40, 2006.

[5] A. Cherns. The principles of sociotechnical design. *Human Relations*, 29(8):783–792, August 1976.

[6] E. Di Nitto, C. Ghezzi, A. Metzger, M. Papazoglou, and K. Pohl. A journey to highly dynamic, self-adaptive service-based applications. *Autom. Softw. Eng.*, 2008.

[7] S. Dobson et al. A survey of autonomic communications. *ACM Trans. Auton. Adapt. Syst.*, 1(2):223–259, 2006.

[8] IBM. An architectural blueprint for autonomic computing. *Whitepaper*, 2005.

[9] L. Juszczyk and S. Dustdar. Script-based generation of dynamic testbeds for soa. In *ICWS*, pages 195–202. IEEE, 2010.

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

[11] Object Management Group. Uml profile for modeling quality of service and fault tolerance characteristics and mechanisms specification, version 1.1., 2008.

[12] H. Psaier, L. Juszczyk, F. Skopik, D. Schall, and S. Dustdar. Runtime behavior monitoring and self-adaptation in service-oriented systems. In *SASO*, pages 164 – 173. IEEE, 2010.

[13] H. Psaier, F. Skopik, D. Schall, and S. Dustdar. Behavior monitoring in self-healing service-oriented systems. In *COMPSAC*, pages 357 – 366. IEEE, 2010.

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

[15] D. Shahaf and E. Horvitz. Generalized task markets for human and machine computation. In *AAAI*, 2010.

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

[17] J. Zhang and R. J. Figueiredo. Autonomic feature selection for application classification. In *ICAC*, pages 43–52. IEEE, 2006.