Programming Hybrid Services in the Cloud

Hong-Linh Truong[1], Schahram Dustdar[1], and Kamal Bhattacharya[2]

[1] Distributed Systems Group, Vienna University of Technology
{truong,dustdar}@infosys.tuwien.ac.at
[2] IBM Research - India
kambhatt@in.ibm.com

Abstract. For solving complex problems, we advocate constructing "social computers" which combine software and human services. However, to date, human capabilities cannot be easily programmed into applications in a similar way like software capabilities. Existing approaches exploiting human capabilities via crowds do not support well on-demand, proactive, team-based human computation. In this paper, we explore a new way to virtualize, provision and program human capabilities using cloud computing concepts and service delivery models. We propose novel methods for modeling clouds of human-based services and combine human-based services with software-based services to establish clouds of hybrid services. In our model, we present common APIs, similar to APIs for software services, to access individual and team-based compute units in clouds of human-based services. Based on that, we propose frameworks and programming primitives for hybrid services. We illustrate our concepts via some examples of using our cloud APIs and existing cloud APIs for software.

1 Introduction

Recently the concept of building social computers has emerged, in which the main principle is to combine human capabilities and software capabilities into composite applications solving complex problems [1, 2]. Furthermore, concrete technologies have been employed to provide human capabilities via standard, easy-to-use interface, such as Web services and Web platforms [3–5] and some efforts have been devoted for modeling and coordinating flows of human works in the process level [6, 7]. In all these works, a fundamental issue is how to program human capabilities. We observed two main approaches in utilizing human capabilities: (i) passively proposing tasks and waiting for human input, such as in crowd platforms [5], and (ii) actively finding and binding human capabilities into applications. While the first one is quite popular and has many successful applications [8–10, 5, 11], it mainly exploits individual capabilities and is platform-specific. In the second approach, it is difficult to proactively invoke human capabilities in Internet-scale due to the lack of techniques and systems supporting proactive utilization of human capabilities [2].

In this paper, we conceptualize human capabilities under the service model and combine them with software establishing clouds of hybrid services. In our approach, we explore novel ways to actively program and utilize human capabilities in a similar way to software services. Our research question is how to provision and program human capabilities using cloud service and deployment models for high level frameworks and programming languages to build "social computers".

C. Liu et al. (Eds.): ICSOC 2012, LNCS 7636, pp. 96–110, 2012.

1.1 Motivation

Hybrid services, in our notion, include software-based services (SBS) and human-based services (HBS). We argue that we could provide a cloud of HBS working in a similar manner to contemporary clouds of SBS (such as Amazon services and Microsoft Azure services) so that HBS can be invoked and utilized in a proactive manner, rather than in a passive way like in crowdsourcing platforms. Furthermore, HBS can be programmed together with SBS in a composite application, instead of being used separately from SBS as in contemporary crowdsourcing platforms.

Our goal is to program HBS and SBS together in an easier way because several complex applications need to utilize SBS and HBS in a similar way. For example, several Information Technology (IT) problems, such as in incident management for IT systems, software component development, and collaborative data analytics, can be described as a dependency graph of tasks in which a task represents a unit of work that should be solved by a human or a software. Solving a task may need to concurrently consider other relevant tasks in the same graph as well as introduce new tasks (this in turns expands the task graph). Utilizing team and hybrid services is important here as tasks are interdependent, but unlike crowdsourcing scenarios in which different humans solving different tasks without the context of teamwork and without the connectedness to SBS. Teamwork is crucial as it allows team members to delegate tasks when they cannot deal with the task as well as newly tasks can be identified and created that need to be solved. SBS for teamwork is crucial for team working platforms in terms of communication, coordination, and analytics. Therefore, it is crucial to have solutions to provision individual- and team-based human capabilities under clouds of human capabilities, in parallel with the provisioning of SBS.

These clouds require novel service models and infrastructures to provide and support on-demand and elastic HBS provisioning. We need solutions allowing us to buy and provision human capabilities via simple interfaces in a similar way to buying and provisioning virtual machines in contemporary clouds of Infrastructure-as-a-Service (IaaS) and Software-as-a-Service (SaaS). However, so far, to our best knowledge, there is no proposed solution towards a cloud model for human capabilities that enables to acquire, program, and utilize HBS in a similar way to that of IaaS, Platform-as-a-Service (PaaS) and SaaS.

1.2 Contributions and Paper Structure

We concentrate on conceptualizing the cloud of HBS and how clouds of HBS and SBS can be programmed for solving complex problems. Our main contributions are:

- a novel model for clouds of HBS and hybrid services provisioning
- a framework for solving complex problems using clouds of hybrid services
- programming primitives for hybrid services

The rest of this paper is organized as follows. Section 2 discusses our model of clouds of hybrid services. Section 3 describes a generic framework for using hybrid services. Section 4 describes programming primitives and examples utilizing clouds of hybrid services. We discuss related work in Section 5. Section 6 concludes the paper and outlines our future work.

2 Models for Clouds of Hybrid Services

In our work, we consider two types of computing elements: software-based computing elements and human-based computing elements. In software-based computing elements, different types of services can be provided to exploit machine capabilities and we consider these types of services under Software-based Service (SBS) category. Similarly, human-based computing elements can also offer different types of services under the HBS category. We consider a cloud of hybrid services as follows:

Definition 1 (Cloud of hybrid services). *A cloud of hybrid services includes SBS and HBS that can be provisioned, deployed and utilized on-demand based on different pricing models.*

In principle, a cloud of hybrid services can also be built atop clouds of SBS and clouds of HBS. As SBS and clouds of SBS are well-researched, in the following we will discuss models for clouds of HBS and of hybrid services.

2.1 Models for HBS

In principle, human capabilities can be provisioned under the service model, e.g., our previous work introduced a technology to offer individual human capabilities under Web services [3]. However, at the moment, there exists no cloud system that the consumer can program HBS in a similar way like IaaS (e.g., Amazon EC) or data (e.g., Microsoft Azure Data Marketplace). Before discussing how clouds of hybrid services can be used, we propose a conceptual model for clouds of HBS.

HBS Communication Interface. Humans have different ways to interact with other humans and ICT systems. Conceptually, we can assume that HBS (and corresponding HBS clouds) abstracting human capabilities can provide different communication interfaces to handle tasks based on a request and response model. *Requests* can be used to describe tasks/messages that an HBS should perform or receive. In SBS, specific request representations (e.g., based on XML) are designed for specific software layers (e.g., application layer, middleware layer, or hardware layer). In HBS we can assume that a single representation can be used, as HBS does not have similar layer structures seen in SBS. Requests in HBS can, therefore, be composed and decomposed into different (sub)requests. The use of the request/response model will facilitate the integration between SBS and HBS as via similar service APIs.

Unlike SBS in which communication can be synchronous or asynchronous, in HBS all communication is asynchronous. In general, the upper bound of the communication delay in and the internal request processing mechanism in HBS are unknown. However, HBS intercommunication can be modeled using:

- message-passing in which two HBS can directly exchange requests: $hbs_i \underset{request}{\rightarrow} hbs_j$. One example is that hbs_i sends a request via SMS to hbs_j. Similarly, an SBS can also send a request directly to an HBS.
- shared-memory in which two HBS can exchange requests via a SBS. For example, hbs_i stores a request into a Dropbox[1] directory and hbs_j obtains the request

[1] `www.dropbox.com`

from the Dropbox directory. Similarly, an SBS and HBS can also exchange requests/responses via an SBS or an HBS (e.g., a software can be built atop Dropbox to trigger actions when a file is stored into a Dropbox directory (see `http://www.wappwolf.com`)).

Similarly to machine instances which offer facilities for remote job deployment and execution, an HBS communication interface can be used to run requests/jobs on HBS.

Human Power Unit (HPU). The first issue is to define a basic model for describing the notion of "computing power" of HBS. Usually, the computing capability of a human-based computing element is described via human skills and skill levels. Although there is no standard way to compare skills and skill levels described and/or verified by different people and organizations, we think that it is feasible to establish a common, comparative skills *for a particular cloud* of HBS.

- the cloud can enforce different evaluation techniques to ensure that any HBS in its system will declare skills and skill levels in a cloud-wide consistency. This is, for example, similar to some crowdsourcing systems which have rigorous tests to verify claimed skills.
- the cloud can use different benchmarks to test humans to validate skills and skill levels. Each benchmark can be used to test a skill and skill level. This is, for example, similar to Amazon which uses benchmarks to define its elastic compute unit.
- the cloud can map different skills from different sources into a common view which is consistent in the whole cloud.

We define HPU for an HBS as follows:

Definition 2 (Human Power Unit). *HPU is a value describing the computing power of an HBS measured in an abstract unit. A cloud of HBS has a pre-defined basic power unit, hpu_θ, corresponding to the baseline skill bs_θ of the cloud.*

Without the loss of generality, we assume $hpu_\theta = f(bs_\theta)$. A cloud C provisioning HBS can support a set of n skills $SK = \{sk_1, \cdots, sk_n\}$ and a set of m cloud skill levels $SL = \{1, \cdots, m\}$. C can define the human power unit wrt sk_i for sl_j as follows:

$$hpu(sk_i, sl_j) = hpu_\theta \times \frac{f(sk_i)}{f(bs_\theta)} \times sl_j \tag{1}$$

For the cloud C, $\frac{f(sk_i)}{f(bs_\theta)}$ is known (based on the definition of SK). Given the capability of an $hbs - CS(hbs) = \{(sk_1, sl_1), \cdots, (sk_u, sl_u)\}$ – the corresponding hpu can be calculated as follows:

$$hpu(CS(hbs)) = \sum_{i=1}^{u} hpu(sk_i, sl_i) \tag{2}$$

Note that two HBS can have the same hpu value, even their skills are different. To distinguish them, we propose to use a set of "architecture" types (e.g., similar to different types of instruction set architectures such as x86, SPARC, and ARM), and the cloud

provider can map an HBS into an architecture type by using its skills and skill levels. Given a human offering her capabilities to C, she can be used exclusively or shared among different consumers. In case an hbs is provisioned exclusively for a particular consumer, the hbs can be associated with a theoretical utilization u – describing the utilization of a human – and $CS(hbs)$; its theoretical HPU would be $u \times hpu(CS(hbs))$. In case a hbs is provisioned for multiple consumers, the hbs can be described as a set of multiple instances, each has a theoretical power as $u_i \times hpu(CS_i(hbs))$ where $u = \sum(u_i) \leq 1$ and $CS(hbs) = CS_1(hbs) \cup CS_2(hbs) \cup \cdots \cup CS_q(hbs)$.

Using this model, we can determine theoretical power for individual HBS as well as for a set of individual HBS. Note that the power of a set of HBS may be more than the sum of power units of its individual HBS, due to teamwork. However, we can assume that, similar to individual and cluster of machines, theoretical power units are different from the real one and are mainly useful for selecting HBS and defining prices.

2.2 HBS Instances Provisioning

Types of HBS Instances For HBS we will consider two types of instances:

Definition 3 (Individual Compute Unit instances (iICU)). *iICU describe instances of HBS built atop capabilities of individuals. An individual can provide different iICU. Analogous to SBS, an iICU is similar to an instance of a virtual machine or a software.*

Definition 4 (Social Compute Unit instances (iSCU)). *iSCU describe instances of HBS built atop capabilities of multiple individuals and SBS. Analogous to SBS, an iSCU is similar to a virtual cluster of machines or a complex set of software services.*

In our approach, iICU is built based on the concept that an individual can offer her capabilities via services [3] and iSCU is built based on the concept of Social Compute Units [12]) which represents a team of individuals.

HBS Instance Description. Let C be a cloud of hybrid services. All services in C can be described as follows: $C = HBS \cup SBS$ where HBS is the set of HBS instances and SBS is the set of SBS instances. The model for SBS is well-known in contemporary clouds and can be characterized as $SBS(capability, price)$. The provisioning description models for HBS instances are proposed as follows:

- For an $iICU$ its provisioning description includes (CS, HPU, $price$, $utilization$, $location$, $APIs$).
- For an $iSCU$ its provisioning description includes (CS, HPU, $price$, $utilization$, $connectedness$, $location$, $APIs$).

From the consumer perspective, $iSCU$ can be offered by the cloud provider or the consumer can build its own $iSCU$. In principle, in order to build an SCU, the provider or the consumer can follow the following steps: first, selecting suitable $iICU$ for an $iSCU$ and, second, combining and configuring SBS to have a working platform for $iSCU$. The $connectedness$ reflects the intercommunication topology connecting members of $iSCU$, such as ring, star, and master-slave, typically configured via SBS. $APIs$ describe how to communicate to and execute requests on HBS. Moreover, similar to SBS, HBS can also be linked to user rating information, often managed by third-parties.

Pricing Factors. Similar to existing SBS clouds, we propose clouds of HBS to define different pricing models for different types of HBS instances. The baseline for the prices can be based on hpu_θ. We propose to consider the following specific pricing factors:

- utilization: unlike individual machines whose theoretical utilization when selling is 100%, ICU has much lower theoretical utilization, e.g., normal full time people have a utilization of 33.33% (8 hours per day). However, an SCU can theoretically have 100% utilization. The real utilization of an HBS is controlled by the HBS rather than by the consumer as in machine/software instances.
- offering communication APIs: it is important that different communication capabilities will foster the utilization of HBS. Therefore, the provider can also bill consumers based on communication APIs (e.g., charge more when SMS is enabled).
- connectedness: similar to capabilities of (virtual) networks between machines in a (virtual) cluster, the connectedness of an $iSCU$ will have a strong impact on the performance of $iSCU$. Similar to pricing models in existing collaboration services[2], the pricing factor for connectedness can be built based on which SBS and collaboration features are used for iSCU.

Furthermore, other conventional factors used in SBS such as usage duration and location are considered.

2.3 Cloud APIs for Provisioning Hybrid Services

Services in a cloud of hybrid services can be requested and provisioned on-demand. As APIs for provisioning SBS are well developed, we will focus on APIs for provisioning HBS. Table 1 describes some APIs that we develop for hybrid services in our VieCOM (Vienna Elastic Computing Model). These APIs are designed in a similar manner to common APIs for SBS.

Figure 1 shows main Java-based classes representing HPU, HBS and its subclasses (ICU and SCU), requests and messages for HBS (`HBSRequest` and `HBSMessage`), and skills (`CloudSkill`, `Skill`, and `SkillLevel`). Currently, we simulate our cloud of HBS. For SBS, we use existing APIs provided by cloud providers and common client APIs libraries, such as JClouds (`www.jclouds.org`) and boto (`http://docs.pythonboto.org/en/latest/index.html`).

3 Framework for Utilizing Hybrid Services

By utilizing hybrid services in clouds, we could potentially solve several complex problems that need both SBS and HBS. In our work, we consider complex problems that can be described under dependency graphs. Let DG be dependency graph of tasks to be solved; DG can be provided or extracted automatically. In order to solve a task $t \in DG$, we need to determine whether t will be solved by SBS, HBS or their combination. For example, let t be a virtual machine failure and the virtual machine is provisioned by

[2] Such as in Google Apps for Business (`http://www.google.com/enterprise/apps/business/pricing.html`)

Table 1. Main APIs for provisioning HBS

APIs	Description
APIs for service information and management	
listSkills ();listSkillLevels()	list all pre-defined skills and skill levels of clouds
listICU();listSCU()	list all iICU and iSCU instances that can be used. Different filters can be applied to the listing
negotiateHBS()	negotiate service contract with an $iICU$ or an $iSCU$. In many cases, the cloud can just give the service contract and the consumer has to accept it (e.g., similar to SBS clouds)
startHBS()	start an $iICU$ or an $iSCU$. By starting, the HBS is being used. Depending on the provisioning contract, the usage can be time-based (subscription model) or task-based (pay-per-use model)
suspendHBS ()	suspend the operation of an $iICU$ or $iSCU$. Note that in suspending mode, the HBS is not released yet for other consumers yet.
resumeHBS ()	resume the work of an $iICU$ or $iSCU$
stopHBS()	stop the operation of an $iICU$ or $iSCU$. By stopping the HBS is no longer available for the consumer
reduceHBS()	reduce the capabilities of $iICU$ or $iSCU$
expandHBS()	expand the capabilities of $iICU$ or $iSCU$
APIs for service execution and communication	
runRequestOnHBS()	execute a request on an $iICU$ or $iSCU$. By execution, the HBS will receive the request and perform it.
receiveResultFromHBS()	receive the result from an $iICU$ or $iSCU$
sendMessageToHBS()	send (support) messages to HBS
receiveMessageFromHBS()	receive messages from HBS

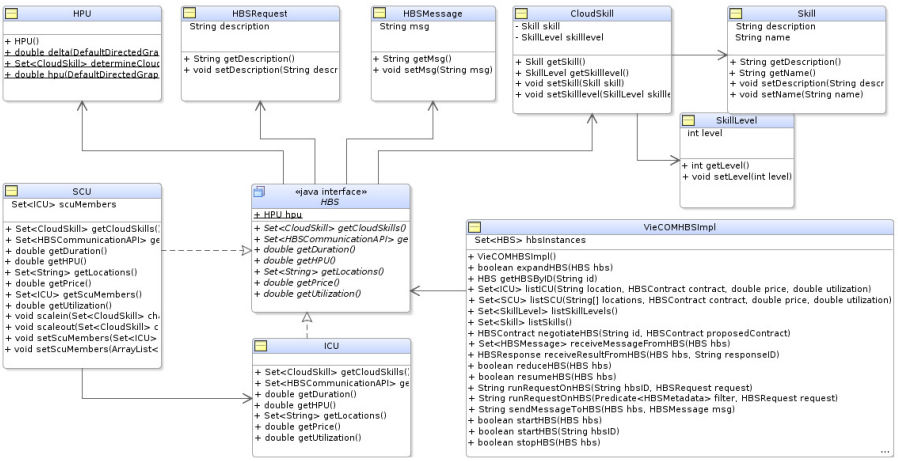

Fig. 1. Example of some Java-based APIs for clouds of HBS

Amazon EC2. Two possibilities can be performed: (i) request a new virtual machine from Amazon EC and configure the new virtual machine suitable for the work or (ii) request an HBS to fix the virtual machine. In case (i) SBS can be invoked, while for case (ii) we need to invoke an HBS which might need to be provisioned with extra SBS for supporting the failure analysis.

Our approach for utilizing hybrid services includes the following points:

- link tasks with their required human power units via skills and skill levels, before programming how to utilize HBS and SBS.
- form or select suitable $iSCU$ or $iICU$ for solving tasks. Different strategies will be developed for forming or selecting suitable $iSCU$ or $iICU$, such as utilizing different ways to traverse the dependency graph and to optimize the formation objective.
- program different strategies of utilizing $iSCU$ and $iICU$, such as considering the elasticity of HBS due to changes of tasks and HBS. This is achieved by using programming primitives and constructs atop APIs for hybrid services.

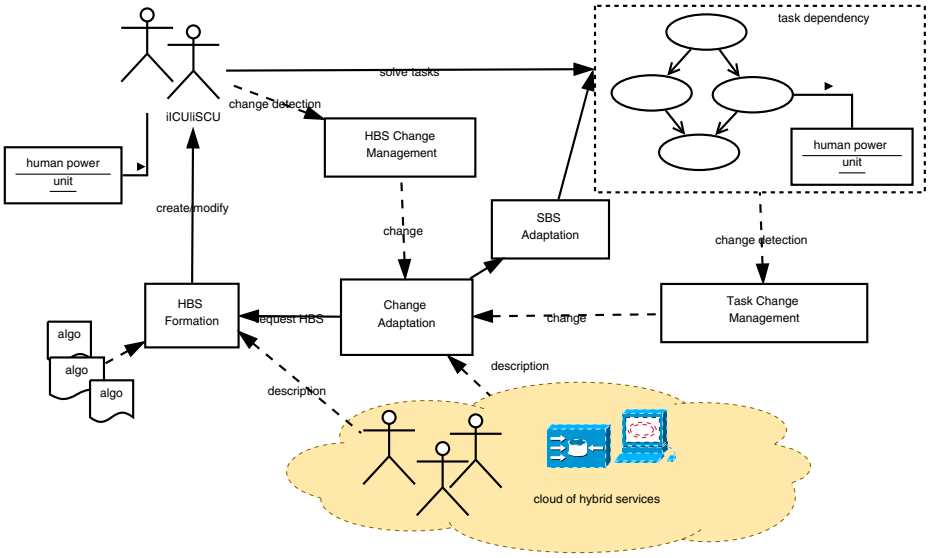

Fig. 2. Conceptual architecture

Figure 2 describes the conceptual architecture of our framework for solving complex problems. Given a task dependency graph, we can detect changes in required human computing power by using *Task Change Management*. Detected required power changes will be sent to *Change Adaptation*, which in turns triggers different operations on HBS usage, such as creating new HBS or adapting an existing HBS. The operations on HBS are provided via different algorithms, each suitable for specific situations.

When an HBS deals with a task graph, the HBS can change the task graph and its required human power units (this will trigger HBS operations again). During the solving process, HBS can change and this can be detected by *HBS Change Management*. The HBS change will be sent to *Change Adaptation*.

4 Programming Hybrid Services

In this section, we discuss some programming primitives for hybrid services that can be applied to complex application framework that we mentioned before. Such a primitives can be used in different components, such as *HBSFormation* and *ChangeAdaptation*, in our framework described in Figure 2. In illustrating programming examples, we consider a virtualized cloud of hybrid services that are built atop our cloud of HBS and real-world clouds of SBS. Consequently, we will combine our APIs, described in Section 2.3, with existing client cloud API libraries.

4.1 Modeling HPU-Aware Task Dependency Graphs

Our main idea in modeling HPU-aware task dependencies is to link tasks to required *management skills and compliance constraints*:

- human resource skills: represent skill sets that are required for dealing with problems/management activities.
- constraints: represent constraints, such as resource locations, governance compliance, time, cost, etc., that are associated with management activities and humans dealing with these activities.

Given a dependency graph of tasks, these types of information can be provided manually or automatically (e.g., using knowledge extraction). Generally, we model dependencies among tasks and required skills and compliance constraints as a directed graph $G(N, E)$ where N is a set of nodes and E is a set of edges. A node $n \in N$ represents a task or required skills/compliance constraints, whereas an edge $e(n_i, n_j) \in E$ means that n_j is dependent on n_i (n_i can cause some effect on n_j or n_i can manage n_j). Edges may be associated with weighted factors to indicate the importance of edges. The required skills, compliance constraints and weighted factors will be used to determine the required human power unit (HPU) for a task, to select $iICU$ and members for $iSCU$, and to build the connectedness for SCUs.

Examples and Implementation. Figure 3 presents an example of a dependency graph of an IT system linked to management skills. In our implementation of dependency graph, we use JGraphT (`http://jgrapht.org/`). We define two main types of Node – *ITProblem* and *Management*. All relationships are dependency. It is also possible to use TOSCA [13] to link people skills and map TOSCA-based description to JGraphT.

4.2 Combining HBS and SBS

Combining HBS and SBS is a common need in solving complex problems (e.g., in evaluating quality of data in simulation workflows). In our framework, this feature can

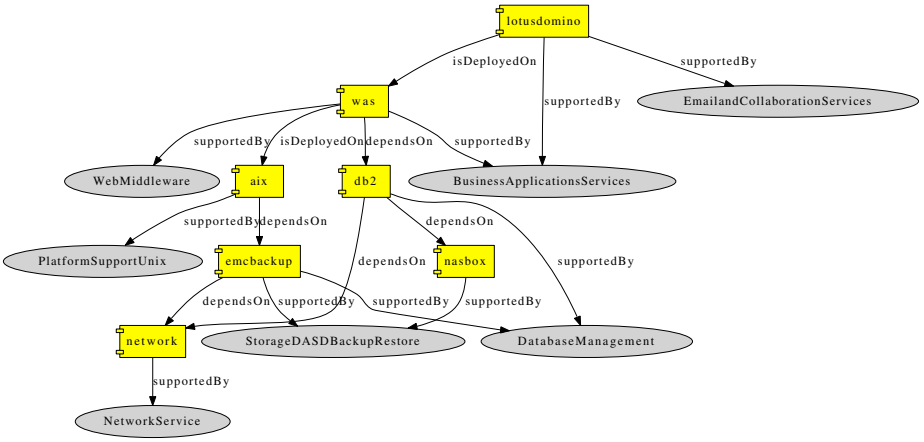

Fig. 3. An example of HPU-aware dependency graph. A component box describes a software and its problems (*ITProblem* node). An eclipse describes management skills (*Management* node).

be used for preparing inputs managed by SBS for an HBS work or managing outputs from HBS work. Furthermore, it can be used to provision SBS as utilities for HBS work (e.g., requiring HBS to utilize specific SBS in order to produce the result where SBS is provisioned by the consumer).

Examples. Listing 1.1 shows an example of programming a combination of HBS and SBS for a task using our cloud APIs and JClouds. In this example, we want to invoke Amazon S3 to store a log file of a Web application sever and invoke an HBS to find problems. Using this way, we can also combine HBS with HBS and of course SBS with SBS from different clouds.

```
//using JClouds APIs to store log file of web application server
BlobStoreContext context =
    new BlobStoreContextFactory().createContext("aws-s3","REMOVED
        ","REMOVED");
BlobStore blobStore = context.getBlobStore();
//.... and add file into Amazon S3
Blob blob = blobStore.blobBuilder("hbstest").build();
blob.setPayload(new File("was.log"));
blobStore.putBlob("hbstest", blob);
String uri = blob.getMetadata().getPublicUri().toString();
VieCOMHBS vieCOMHBS = new VieCOMHBSImpl();
//assume that WM6 is the HBS that can analyze the Web Middleware
     problem
vieCOMHBS.startHBS("WM6");
HBSRequest request = new HBSRequest();
request.setDescription("Find possible problems from " + uri);
vieCOMHBS.runRequestOnHBS("WM6", request);
```

Listing 1.1. Example of HBS combined with SBS

4.3 Forming and Configuring iSCUs

A cloud provider can form an $iSCU$ and provide it to the consumer as well as a consumer can select $iICU$ and SBS to form an $iSCU$. An $iSCU$ not only includes HBS ($iICU$ or other sub $iSCU$) but also consists of possible SBS for ensuring the connectedness within $iSCU$ and for supporting the work. There are different ways to form SCUs. In the following, we will describe some approaches for forming SCUs to solve a dependency graph of tasks.

Selecting Resources for iSCU. Given a task $t \in DG$, our approach in dealing with t is that we do not just simply take required management resources suitable for t but we need to consider possible impacts of other tasks when solving t and the chain of dependencies. To this end, we utilize DG to determine a set of suitable human resources to deal with t and t's possible impact. Such human resources establish HBS capabilities in an $iSCU$. Overall, the following steps are carried out to determine required SCU:

- Step 1: determine $DG_{BAU} \subseteq DG$ where DG_{BAU} includes all $t_j \exists$ a walk (t_j, t), t_j is the task that must be dealt together with t in typical Business-As-Usual cases.
- Step 2: determine $DG_{CA} \subseteq DG$ that includes tasks that should be taken into account under corrective action (CA) cases. $DG_{CA} = \{t_r\} \exists$ a walk(t_r, t_j) with $t_j \in DG_{BAU}$.
- Step 3: merge $DG_{SCU} = DG_{BAU} \cup DG_{CA}$ by (re)assigning weighted factors to links between $(t_k, t_l) \in DG_{SCU}$ based on whether (i) t_k and t_l belong to DG_{BAU} or DG_{CA}, (ii) reaction chain from t to t_k or to t_l, and (iii) the original weighted factor of links consisting of t_k or t_l.
- Step 4: traverse DG_{SCU}, $\forall t_i \in DG_{SCU}$, consider all (t_i, r_i) where r_i is management resource node linking to t_i in order to determine human resources.

Based on the above-mentioned description different SCU formation strategies can be developed. Note that our principles mentioned above aim at forming $iSCU$ enough

Table 2. Examples of SCU formation strategies

Algorithms	Description
SkillWithNPath	Select $iICU$ for $iSCU$ based on only skills with a pre-defined network path length starting from the task to be solved.
SkillMinCostWithNPath	Select $iICU$ for $iSCU$ based on only skills with minimum cost, considering a pre-defined network path length starting from the task to be solved.
SkillMinCostMaxLevelWithNPath	Select $iICU$ for $iSCU$ based on skills with minimum cost and maximum skill levels, considering a pre-defined network path length starting from the task to be solved.
SkillWithNPathUnDirected	Similar to $SkillWithNPath$ but considering undirected dependency
MinCostWithNPathUnDirected	Similar to $MinCostWithNPath$ but considering undirected dependency
MinCostWithAvailNPathUnDirected	Select Select $iICU$ for $iSCU$ based on skills with minimum cost, considering availability and a pre-defined network path length starting from the task to be solved. Undirected dependencies are considered.

```
DefaultDirectedGraph<Node, Relationship> dg; //graph of problems
//...
double hpu = HPU.hpu(dg); //determine
SCUFormation app = new SCUFormation( dg);
ManagementRequest request = new ManagementRequest();
//define request specifying only main problems to be solved
//....
//call algorithms to find suitable HBS. Path length =2 and
    availability from 4am to 19pm in GMT zone
ResourcePool scu = app.
    MinCostWithAvailabilityNPathUnDirectedFormation(request, 2,
    4, 19);
if (scu == null) { return ; }
ArrayList<HumanResource> scuMembers = scu.getResources();
SCU iSCU = new SCU();
iSCU.setScuMembers(scuMembers);
//setting up SBS for scuMember ...
```

Listing 1.2. Example of forming iSCU by minimizing cost and considering no direction

for solving main tasks and let $iSCU$ evolve during its runtime. There could be several possible ways to obtain DG_{BAU} and DG_{CA}, dependent on specific configurations and graphs for specific problems. Therefore, potentially the cloud of HBS can provide several algorithms for selecting HBS to form SCUs. As we aim at presenting a generic framework, we do not describe here specific algorithms, however, Table 2 describes some selection strategies that we implement in our framework. Listing 1.2 describes an example of forming an SCU.

Setting up iSCU connectedness. After selecting members of $iSCU$, we can also program SBS and HBS for the $iSCU$ to have a complete working environment. $iSCU$ can have different connectedness configurations, such as

- ring-based $iSCU$: the topology of $iSCU$ is based on a ring. In this case for each $(hbs_i, hbs_j) \in iSCU$ then we program $hbs_i \underset{request}{\rightarrow} hbs_j$ based on message-passing or shared memory models. For example a common Dropbox directory can be created for hbs_i and hbs_j to exchange requests/responses.
- star-based $iSCU$: a common SBS can be programmed as a shared memory for $iSCU$. Let sbs be SBS for $iSCU$ then $\forall hbs_i \in iSCU$ give hbs_i access to sbs. For example, a common Dropbox directory can be created and shared for all $hbs_i \in iSCU$.
- master-slave $iSCU$: an $hbs \in iSCU$ can play the role of a shared memory and scheduler for all other $hbs_i \in iSCU$.

Listing 1.3 presents an example of establishing the connectedness for an $iSCU$ using Dropbox. Note that finding suitable configurations by using HBS information and compliance constraints is a complex problem that is out of the scope of this paper.

```
SCU iSCU ;
//... find members for SCU
DropboxAPI<WebAuthSession> scuDropbox; //using dropbox apis
//...
AppKeyPair appKeys = new AppKeyPair(APP_KEY, APP_SECRET);
WebAuthSession session =
        new WebAuthSession(appKeys, WebAuthSession.AccessType.
            DROPBOX);
//...
session.setAccessTokenPair(accessToken);
scuDropbox = new DropboxAPI<WebAuthSession>(session);
//sharing the dropbox directory to all scu members
//first create a share
DropboxAPI.DropboxLink link = scuDropbox.share("/hbscloud");
//then send the link to all members
VieCOMHBS vieCOMHBS = new VieCOMHBSImpl();
for (HBS hbs : iSCU.getScuMembers()) {
    vieCOMHBS.startHBS(icu);
    HBSMessage msg = new HBSMessage();
    msg.setMsg("pls. use shared Dropbox for communication " +
        link.url);
    vieCOMHBS.sendMessageToHBS(hbs, msg);
//...
}
```

Listing 1.3. Example of star-based iSCU using Dropbox as a communication hub

```
SCU iSCU ;
//...
iSCU.setScuMembers(scuMembers);
//setting up SBS for scuMember
//...
double hpu = HPU.hpu(dg); //determine current hpu
//SCU solves/adds tasks in DG
// ....
//graph change - elasticity based on human power unit
double dHPU = HPU.delta(dg,hpu);
DefaultDirectedGraph<Node, Relationship> changegraph;
//obtain changes
Set<CloudSkill> changeCS = HPU.determineCloudSkill(changegraph);
if (dHPU > SCALEOUT_LIMIT) {
   iSCU.scaleout(changeCS); //expand iSCU
}
 else if (dHPU < SCALEIN_LIMIT) {
   iSCU.scalein(changeCS); //reduce iSCU
//...
}
```

Listing 1.4. Example of elasticity for SCU based on task graph change

4.4 Change Model for Task Graph's Human Power Unit

When a member in an $iSCU$ receives a task, she might revise the task into a set of sub-tasks. Then she might specify human compute units required for sub tasks and revise the task graph by adding these sub-tasks. As the task graph will change, its required human power unit is changed. By capturing the change of the task graph, we can decide to scale in/out the $iSCU$. Listing 1.4 describes some primitives for scaling in/out $iSCU$ based on the change of HPU.

5 Related Work

Most clouds of SBS offering different possibilities to acquire SBS on-demand. However, similar efforts for HBS are missing today. Although both, humans and software, can perform similar work and several complex problems need both of them in the same system, currently there is a lack of programming models and languages for hybrid services of SBS and HBS. Most clouds of SBS offering different possibilities to acquire SBS on-demand, however, similar efforts for HBS are missing today.

Existing systems for utilizing crowds for solving complex problems [14, 5] do not consider how to integrate and virtualize software in a similar manner to that for humans. As we have analyzed, current support can be divided in three approaches [2]: (i) using plug-ins to interface to human, such as BPEL4People[4] or tasks integrated into SQL processing systems[11], (ii) using separate crowdsourcing platforms, such as MTurk[15], and (iii) using workflows, such as Turkomatic [8]. A drawback is that all of them consider humans individually and human capabilities have not been provisioned in a similar manner like software capabilities. As a result, an application must split tasks into sub-tasks that are suitable for individual humans, which do not collaborate to each other, before the application can invoke humans to solve these sub-tasks. Furthermore, the application must join the results from several sub-tasks and it is difficult to integrate work performed by software with work performed by humans. This is not trivial for the application when dealing with complex problems required human capabilities. In terms of communication models and coordination models, existing models such as in MTurk and HPS are based on push/pull/mediator but they are platforms/middleware built-in rather than reusable programming primitives of programming models.

In our work, we develop models for clouds of HBS. Our techniques for virtualizing HBS and programming HBS in a similar way to SBS are different from related work. Such techniques can be used by high-level programming primitives and languages for social computers.

6 Conclusions and Future Work

In this paper, we have proposed novel methods for modeling clouds of HBS and describe how we can combine them with clouds of SBS to create hybrid services. We believe that clouds of hybrid services are crucial for complex applications which need to proactively invoke SBS and HBS in similar ways. We describe general frameworks and programming APIs where and how hybrid services can be programmed.

In this paper, we focus on designing models, frameworks and APIs and illustrating programming examples. Further real-world experiments should be conducted in the future. Furthermore, we are also working on the integration with programming languages for social collaboration processes [7] using hybrid services. Other related aspects, such as pricing models and contract negotiation protocols, will be also investigated.

References

1. The Social Computer - Internet-Scale Human Problem Solving (socialcomputer.eu) (last access: May 3, 2012)
2. Dustdar, S., Truong, H.L.: Virtualizing software and humans for elastic processes in multiple clouds – a service management perspective. International Journal of Next-Generation Computing 3(2) (2012)
3. Schall, D., Truong, H.L., Dustdar, S.: Unifying human and software services in web-scale collaborations. IEEE Internet Computing 12(3), 62–68 (2008)
4. WS-BPEL Extension for People (BPEL4People) Specification Version 1.1 (2009), http://docs.oasis-open.org/bpel4people/bpel4people-1.1-spec-cd-06.pdf
5. Doan, A., Ramakrishnan, R., Halevy, A.Y.: Crowdsourcing systems on the world-wide web. Commun. ACM 54(4), 86–96 (2011)
6. Oppenheim, D.V., Varshney, L.R., Chee, Y.-M.: Work as a Service. In: Kappel, G., Maamar, Z., Motahari-Nezhad, H.R. (eds.) ICSOC 2011. LNCS, vol. 7084, pp. 669–678. Springer, Heidelberg (2011)
7. Liptchinsky, V., Khazankin, R., Truong, H.-L., Dustdar, S.: Statelets: Coordination of Social Collaboration Processes. In: Sirjani, M. (ed.) COORDINATION 2012. LNCS, vol. 7274, pp. 1–16. Springer, Heidelberg (2012)
8. Kulkarni, A.P., Can, M., Hartmann, B.: Turkomatic: automatic recursive task and workflow design for mechanical turk. In: Proceedings of the 2011 Annual Conference Extended Abstracts on Human Factors in Computing Systems, CHI EA 2011, pp. 2053–2058. ACM, New York (2011)
9. Barowy, D.W., Berger, E.D., McGregor, A.: Automan: A platform for integrating human-based and digital computation. Technical Report UMass CS TR 2011-44, University of Massachusetts, Amherst (2011), http://www.cs.umass.edu/~emery/pubs/AutoMan-UMass-CS-TR2011-44.pdf
10. Baird, H.S., Popat, K.: Human Interactive Proofs and Document Image Analysis. In: Lopresti, D.P., Hu, J., Kashi, R.S. (eds.) DAS 2002. LNCS, vol. 2423, pp. 507–518. Springer, Heidelberg (2002)
11. Marcus, A., Wu, E., Karger, D., Madden, S., Miller, R.: Human-powered sorts and joins. Proc. VLDB Endow. 5, 13–24 (2011)
12. Dustdar, S., Bhattacharya, K.: The social compute unit. IEEE Internet Computing 15(3), 64–69 (2011)
13. Binz, T., Breiter, G., Leymann, F., Spatzier, T.: Portable cloud services using tosca. IEEE Internet Computing 16(3), 80–85 (2012)
14. Brew, A., Greene, D., Cunningham, P.: Using crowdsourcing and active learning to track sentiment in online media. In: Proceeding of the 2010 Conference on ECAI 2010: 19th European Conference on Artificial Intelligence, pp. 145–150. IOS Press, Amsterdam (2010)
15. Amazon mechanical turk (2011) (last access: November 27, 2011)