International Journal of Cooperative Information Systems
Vol. 22, No. 4 (2013) 1341003 (28 pages)
© World Scientific Publishing Company
DOI: 10.1142/S0218843013410037

CONCEPTUALIZING AND PROGRAMMING HYBRID SERVICES IN THE CLOUD

HONG-LINH TRUONG* and SCHAHRAM DUSTDAR†

Distributed Systems Group, Vienna University of Technology
Argentinierstrasse 8/184-1, A-1040 Vienna, Austria
**truong@dsg.tuwien.ac.at*
†dustdar@dsg.tuwien.ac.at

KAMAL BHATTACHARYA

IBM Research, Nairobi, Kenya, Africa
kamal@ke.ibm.com

Received 10 March 2013
Accepted 28 October 2013
Published 24 December 2013

For solving complex problems, in many cases, software alone might not be sufficient and we need hybrid systems of software and humans in which humans not only direct the software performance but also perform computing and vice versa. Therefore, we advocate constructing "social computers" which combine software and human services. However, to date, human capabilities cannot be easily programmed into complex applications in a similar way like software capabilities. There is a lack of techniques to conceptualize and program human and software capabilities in a unified way. 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 conceptualizing and modeling clouds of human-based services (HBS) and combine HBS with software-based services (SBS) to establish clouds of hybrid services. In our model, we present common APIs, similar to well-developed APIs for software services, to access individual and team-based compute units in clouds of HBS. Based on that, we propose a framework for utilizing SBS and HBS to solve complex problems. We present several programming primitives for hybrid services, also covering forming hybrid solutions consisting of software and humans. We illustrate our concepts via some examples of using our cloud APIs and existing cloud APIs for software.

Keywords: Hybrid services; cloud computing; human-based computation; service computing.

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,a] In such types of computers, both software

*Corresponding author.
[a]The Social Computer — Internet-Scale Human Problem Solving. socialcomputer.eu. Last access: 3 May 2012.

and humans play equally roles: they direct/coordinate as well as perform tasks, dependent on their capabilities and specific context. For example, (i) a software can analyze an image and direct scientists to carry out a quality assessment of the analysis result before sending the result to another software or (ii) a scientist can direct a software to analyze high quality of images while asking another scientist to judge if it makes sense to continue the analysis of images of low quality. While such a combination in complex systems is not new, to build and program such systems capable of supporting pro-active, highly-interactive, team-based human computation under different elastic, pay-per-use models with on-demand (cloud) software services in a unified way remains challenging.

To date, concrete technologies have been employed to provide human capabilities via standard, easy-to-use interface, such as Web services and Web platforms[2,3,b] and some efforts have been devoted for modeling and coordinating flows of human works in the process/workflow level.[4,5] In all these works, a fundamental issue is how to utilize 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,[3] and (ii) actively finding and binding human capabilities into applications. While the first one is quite popular and has many successful applications,[3,6–9] 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.[1] From a programming perspective, currently, most techniques concentrate on workflow-based solutions in which the workflow engine can find suitable humans and assign the tasks to them. However, many complex problems requiring both humans and software cannot be solved by using current workflow-based solutions, as these problems demand flexible interactions among humans and software services.

In this paper, we conceptualize human capabilities under the service model and combine them with software to establish clouds of hybrid human- and software-based services (HBS and SBS). This enables the provisioning of a large-scale number of HBS together with SBS. Our approach aims at exploring novel ways to actively program and utilize human capabilities in a similar way to software services and to provision human capabilities using cloud service and deployment models for high level frameworks and programming languages to build "social computers".

1.1. *Motivation*

Several works have shown that we need to integrate further humans and software under the service model.[1,10] Hybrid services, in our notion, include SBS and HBS. We argue that we could provide a cloud of HBS working in a similar manner to

[b] *WS-BPEL Extension for People* (*BPEL4People*) *Specification Version* 1.1, November 2009. http://docs.oasis-open.org/bpel4people/bpel4people-1.1-spec-cd-06.pdf.

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 existing crowdsourcing platforms. Furthermore, HBS can be programmed together with SBS in a composite application, enabling complex, dynamic interactions between SBS and HBS, instead of being used separately from SBS as in contemporary crowdsourcing platforms and workflows without/with little interactions.

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). By doing so, we could incorporate human capabilities in programming paradigms for "social" computers. However, so far, to our best knowledge, there is no proposed solution towards a cloud model for human capabilities that enables us to acquire, program, and utilize HBS in a similar way to that of IaaS, Platform-as-a-Service (PaaS) and SaaS. Existing technologies are not adequate, for example, workflow and language extensions and social computing platforms are focused too much on crowdsourcing models. The way to program human capabilities in contemporary workflows and crowdsourcing platforms is that either tasks are published for humans to bid them (e.g. in most crowdsourcing platforms) or tasks are directly mapped to humans by the workflow engines (e.g. in human-centric workflows). While the first mechanism allows people to select the task, it do not encourage people to interact together to solve the task. In the second mechanism, it is possible that the workflow engine actively matches suitable people to tasks, (although currently such workflow engines are not popular and they tend to search people only within local organizations). However, still human capabilities are utilized in a passive way, e.g. humans are assigned to tasks or perform simple control activities (e.g. approving a task). Furthermore, with current workflows, it is difficult to utilize large-scale human capabilities in a dynamic, selective way due to the lack of APIs to invoke human services. Furthermore, workflow-based solutions exploiting human capabilities focus on the way to define how the tasks should be done but not how can we provision humans and software in a unified manner so that both humans and software can act as computing units of a single "social computer". Overall, programming flexible and dynamic relationships among software and humans are not well supported, thus hindering the incorporation of human services into programming paradigms.

1.2. *Approach*

To incorporate humans into programming paradigms, our approach aims at tackling the following issues from different aspects:

- *Programming languages*: First of all, we need to abstract human capabilities under *compute units* and provision them under *service units*[10] so that they can be easily incorporated into high-level program elements and constructs. Second, we can extend existing programming languages to support human compute units. Finally, we need to enable different data and control flows among software and human service units via extensible APIs and develop new APIs to support other types of flows among software and human service units.
- *Multiple programming models*: By utilizing human service units as programming elements/constructs, we could support different programming models, such as shared memory, message passing, and artifact-centric models via APIs working atop the service unit abstraction. Furthermore, contemporary workflow languages can be extended to exploit large-scale HBS.
- *Execution environments*: We will need to develop several components for managing provisioning of HBS and the interaction of humans with HBS abstracting them. First computing capability/profile management will allow us to conceptualize and define computing power, pricing and incentive models. We need to monitor and enforce incentives/rewards, quality of results, availability, to name just a few. Several way of communication between different structures of HBS and SBS must be supported.

In this paper, we will focus on abstracting and conceptualizing HBS and clouds of HBS, providing APIs and presenting basic programming techniques for hybrid services.

1.3. *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.

To illustrate our work, we present several examples of how to program software-based services and HBS in a unified way. This paper substantially extends the work described in Ref. 11. We have added our approach (Sec. 1.2), substantially detailed the concepts of hybrid services and extended them to cover also other aspects like archetypes and incentives (in Sec. 2). We have added a conceptual architecture of IaaS of hybrid services (in Sec. 2.5). We also extend programming primitives by detailing our techniques, discuss how our framework can be used to implement HBS provisioning platforms, and elaborate the related work.

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 utilizing hybrid services. Section 4 presents programming primitives utilizing clouds of hybrid services. Section 5 discusses related work. Section 6 concludes the paper and outlines our future work.

2. Conceptualizing Clouds of Hybrid Services

2.1. *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 the SBS category. Similarly, human-based computing elements can also offer different types of services under the HBS category.

Definition 2.1 (Cloud of HBS). A cloud of HBS includes HBS that can be provisioned, deployed and utilized on-demand based on different pricing and incentive models.

Models for SBS and their clouds are well defined.[12] By combining SBS with our model for HBS, we consider a cloud of hybrid services as follows:

Definition 2.2 (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 and incentive models.

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

2.2. *Models for HBS*

Human capabilities can be provisioned under the service model, e.g. our previous work introduced techniques for offering individual human capabilities under Web services.[2] 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 designed and used, we propose a conceptual model for clouds of HBS that cover the following aspects: (i) communication interfaces, (ii) human power unit (HPU), (iii) HBS archetypes, and (iv) pricing and incentive models.

2.2.1. *HBS communication interfaces*

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 (at the end only humans will understand and process the messages), while the message content can be defined based on application needs. 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. Clearly, the reason is that the semantics of human-level communication messages, the way of how the human takes the messages, and the high latency of communication between a requester (whether it is a HBS or SBS) to a HBS prevent synchronous communication in which an HBS is expected to process the messages and send responses at the same time. In general, the upper bound of the communication delay in and the internal request processing mechanism in HBS are unknown (and these issues are not in the focus of this paper). However, HBS intercommunication can be modeled using the well-known message passing and shared memory models:

- *Message passing*: 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*: Shared-memory in which two HBS can exchange requests via a SBS. For example, hbs_i stores a request into a Dropbox[c] directory and hbs_j

[c] www.dropbox.com.

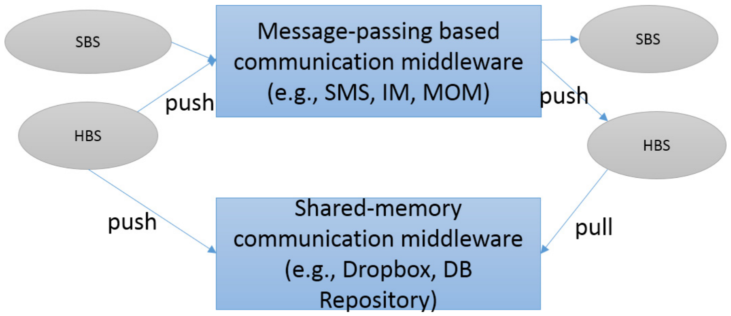
SBS
HBS
push
Message-passing based communication middleware (e.g., SMS, IM, MOM)
SBS
push
HBS
push
Shared-memory communication middleware (e.g., Dropbox, DB Repository)
pull

Fig. 1. Message passing and shared-memory communication models for services in clouds of HBS and SBS.

obtains the request 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)).

Figure 1 describes possible message passing and shared-memory communication models for services in cloud of HBS and SBS. Message-passing middleware can be further divided into different channels implemented by different services, e.g. Short Message Service (SMS) and Instant Messaging (IM) are dedicated for humans, while Message-oriented Middleware (MOM) can be used for both SBS and HBS. In particular, the implementation of communication channels for HBS — among humans and cloud middleware — can benefit from well-researched collaboration services.[13] Similarly, the shared-memory middleware can be built based on different services, offering different types of "shared-memory", such as file-based shared memory, like Dropbox and Amazon S3,[d] and database-based shared memory, like MongoDB[e] and AmazonDynamoDB.[f] Both message-passing and shared-memory communication middleware can be used internally for services within a cloud or externally for the communication between service consumers and services within clouds. Conceptually, we could have all of these middleware under the same set of APIs. In both communication models, the structures of messages sent to HBS are designed for human comprehension.

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.

2.2.2. *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

[d]http://aws.amazon.com/s3/.
[e]http://www.mongodb.org/.
[f]http://aws.amazon.com/dynamodb/.

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, e.g. 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 2.3 (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 $SK = \{sk_1, \ldots, sk_n\}$ and a set of m cloud skill levels $SL = \{1, \ldots, m\}$. C can define the HPU wrt sk_i for sl_j as follows:

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

Here $f(\frac{sk_i}{bs_\theta})$ indicates a way to determine a weighted factor when comparing the skill sk_i against the baseline bs_θ. For the cloud C, $f(\frac{sk_i}{bs_\theta})$ is known and pre-defined (based on the definition of SK). For example, let bs_θ the basic $Testing$ skill, sk_i be the basic $UnitTesting$ skill and sk_j be the basic $IntegrationTesting$ skill, $f(\frac{UnitTesting}{Testing}) = 2$ and $f(\frac{IntegrationTesting}{Testing}) = 8$ could be very simple examples.

Given the capability of an hbs – $CS(hbs) = \{(sk_1, sl_1), \ldots, (sk_u, sl_u)\}$ – 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 if their skills are different. To distinguish them, we propose to use a set of "architecture" types (see Sec. 2.2.3). 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. Given a human offering her capabilities to C, she can be used exclusively or shared among different consumers.

2.2.3. *HBS archetype*

As an HBS can potentially offer different capabilities, similar to SBS, an HBS can be considered to offer a set of types of solutions for a set of domains. For example, an HBS can offer a set of solutions as

$$\begin{aligned} SO = \{(&\{WebDataAnalytics, TwitterAnalytics\}, DataAnalytics), \\ &(\{DataCleansing, DataEnrichment\}, DataQualityImprovement)\}, \end{aligned}$$

where

$$\{WebDataAnalytics, TwitterAnalytics, DataCleansing, DataEnrichment\}$$

are types of solutions and $\{DataAnalytics, DataQualityImprovement\}$ are domains. Therefore, an HPU of an HBS can be associated with types of solutions and domains to indicate the processing capability of the HBS for a specific solution in a specific domain. To allow this association, we propose to use a set of common "architecture" types, called *Archetype*, to indicate the type of solutions in a particular domain that the HPU is determined. This is similar to, e.g. different types of instruction set architectures (such as $\times 86$, SPARC, and ARM).

2.2.4. *Pricing and incentive models*

As we have observed, SBS often comes with pricing models but many of SBS are also given free, in particular, in volunteering and peer-to-peer computing systems,[14] due to different incentives. Similarly, in crowdsourcing, free human efforts are quite popular.[15–18] In the case of HBS, it is obvious that pricing models will need to be identified for HBS, e.g. by the HBS cloud provider in agreement with the HBS provider or other methods.[19] However, an HBS can also declare itself as a free service while it may require rewards when using it via some incentive models.[g]

[g]In literature, many incentive models give rewardss of monetary values. Furthermore, in volunteering computing, there exits the concept of "pay for participation". For the sake of simplicity, we consider all monetary values under pricing models because, although in principle unlike services with commercial intention, these services impose some monetary pricing models that one has to pay when using the services. In other words, incentive models in our cloud HBS rewards non-monetary values, such as reputation.

Therefore, in our model, each HBS will be associated with a set of pricing models and incentive models. This is different from SBS which is associated with pricing models, but not with incentive models. Note that for pricing and incentive models, there must be techniques to support billing and incentive enforcement. However, similar to software systems, such billing and incentive enforcement can be decoupled. For example, when an HBS is utilized and we need to reward it, the consumer or the provider can send its results to an incentive system, which calculates and performs payments.

2.3. *HBS instances provisioning*

2.3.1. *Types of HBS instances*

For HBS we will consider two types of instances:

Definition 2.4 (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 2.5 (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[2] and iSCU is built based on the concept of Social Compute Units[20] which represents a team of individuals.

2.3.2. *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, incentive, utilization, location, APIs*).
- For an *iSCU* its provisioning description includes (*CS, HPU, price, incentive, 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.

2.3.3. *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 h 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,[h] 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.

2.3.4. *Incentive factors*

Incentive factors for ICU are determined by the ICU and/or the HBS cloud provider. This can be done when the ICU is registered and provisioned under the cloud. When enforcing the incentive models of ICU, obviously all incentives must be attributed to the ICU. For SCU it is dependent on how the SCU is structured and the incentive strategies for the SCU are implemented. Thus, when building an SCU, its incentive strategies can also be programmed, e.g. using incentive programming frameworks,[21] to allow the rewards for the whole SCU to be distributed to members of the SCU in the right way. Overall, the enforcement of incentive models will be carried out by the provider of HBS clouds.

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

2.4. *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 abstract APIs that we develop for HBS in our Vienna Elastic Computing Model.[i] These abstract APIs are designed in a similar manner to common APIs for SBS.

Figure 2 shows main Java-based classes for APIs. HPU, HBS, ICU and SCU are described by `HPU, HBS, ICU` and `SCU` classes, respectively. Requests and messages for HBS are described by (`HBSRequest` and `HBSMessage`), while skills and skill levels are described `Skill` and `SkillLevel`. The cloud skills, described in `CloudSkill`, are built from `Skill` and `SkillLevel`. `HBS` and `SBS` are subclasses of `Unit` which represents generic service units. `Unit` is associated with `Cost`, describing cost models, and `Benefit`, describing incentive models and other

Table 1. Main (abstract) 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, e.g. based on pricing/incentive, location, and skills, can be applied to the listing.
negotiateHBS( )	Allow a consumer to send and 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) if the consumer wants to use the HBS.
startHBS( )	Allow a consumer to start an *iICU* or an *iSCU*. Via this API, the consumer sends message to the HBS cloud which, among other activities, passes a notification to the HBS that the HBS is being used from the consumer perspective. Depending on the provisioning contract, the usage can be time-based (subscription model) or task-based (pay-per-use model).
suspendHBS ( )	Allow a consumer to suspend the operation of an *iICU* or *iSCU*. Note that in suspending mode, the HBS is not released for other consumers yet.
resumeHBS ( )	Allow a consumer to resume the work of an *iICU* or *iSCU*.
stopHBS( )	Allow a consumer to 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*, for example, reduce the power unit and some specific communication APIs.
expandhbs( )	Expand the capabilities of *iICU* or *iSCU*, for example, reduce the power unit and some specific communication APIs.
	APIs for service execution and communication
runRequestOnHBS( )	Execute a request on an *iICU* or *iSCU*. By execution, the HBS will receive requests from the consumers and perform them.
receiveResultFromHBS( )	Receive the result from an *iICU* or *iSCU*.
sendMessageToHBS( )	send (support) messages to HBS.
receiveMessageFromHBS( )	receive messages from HBS.

[i] dsg.tuwien.ac.at/research/viecom.

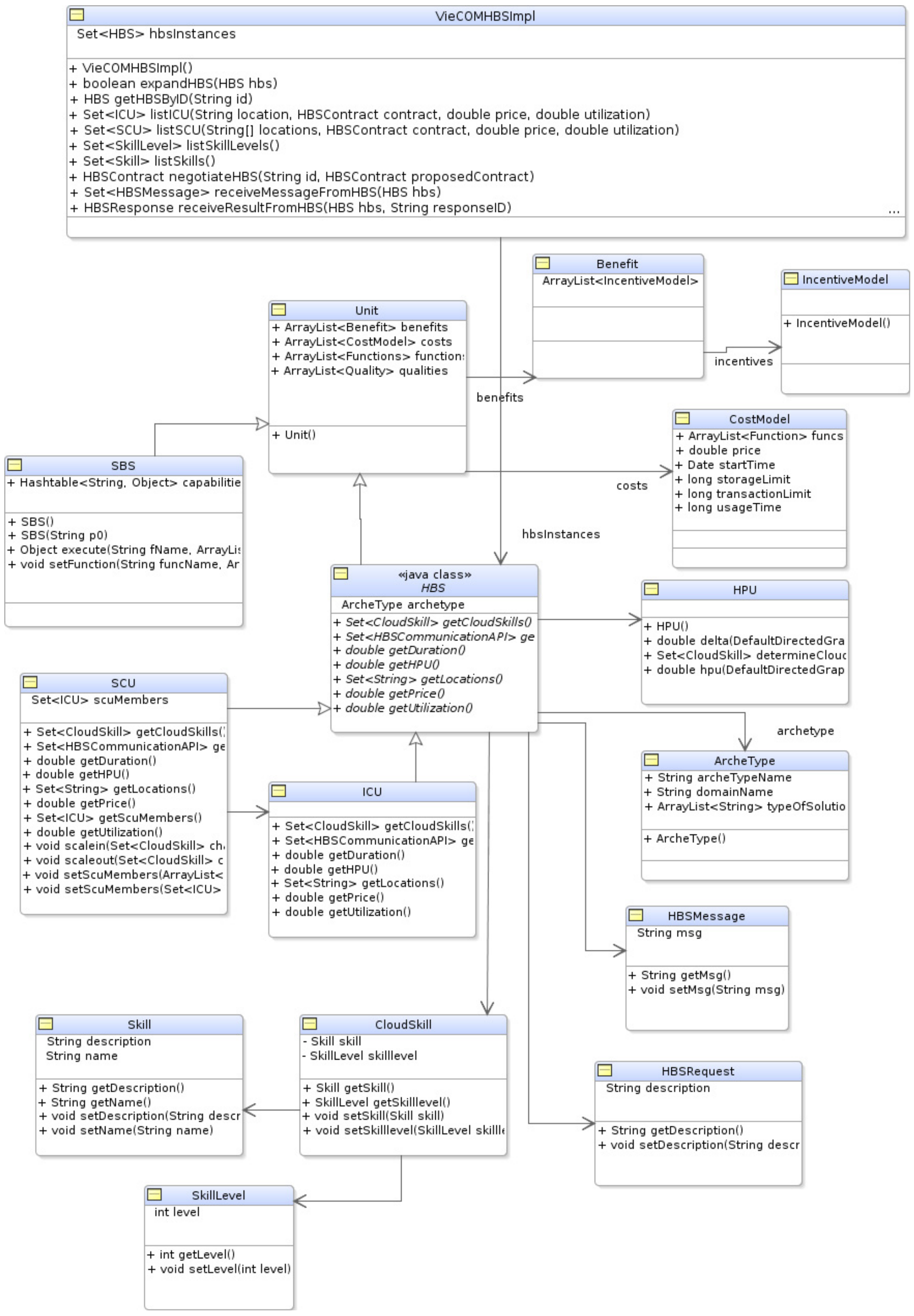
VieCOMHBSImpl
Set<HBS> hbsInstances
+ VieCOMHBSImpl()
+ boolean expandHBS(HBS hbs)
+ HBS getHBSByID(String id)
+ Set<ICU> listICU(String location, HBSContract contract, double price, double utilization)
+ Set<SCU> listSCU(String[] locations, HBSContract contract, double price, double utilization)
+ Set<SkillLevel> listSkillLevels()
+ Set<Skill> listSkills()
+ HBSContract negotiateHBS(String id, HBSContract proposedContract)
+ Set<HBSMessage> receiveMessageFromHBS(HBS hbs)
+ HBSResponse receiveResultFromHBS(HBS hbs, String responseID)
...
Benefit
ArrayList<IncentiveModel>
incentives
IncentiveModel
+ IncentiveModel()
Unit
+ ArrayList<Benefit> benefits
+ ArrayList<CostModel> costs
+ ArrayList<Functions> function
+ ArrayList<Quality> qualities
+ Unit()
benefits
CostModel
+ ArrayList<Function> funcs
+ double price
+ Date startTime
+ long storageLimit
+ long transactionLimit
+ long usageTime
costs
SBS
+ Hashtable<String, Object> capabilitie
+ SBS()
+ SBS(String p0)
+ Object execute(String fName, ArrayLis
+ void setFunction(String funcName, Ar
hbsInstances
«java class»
HBS
ArcheType archetype
+ Set<CloudSkill> getCloudSkills()
+ Set<HBSCommunicationAPI> ge
+ double getDuration()
+ double getHPU()
+ Set<String> getLocations()
+ double getPrice()
+ double getUtilization()
HPU
+ HPU()
+ double delta(DefaultDirectedGra
+ Set<CloudSkill> determineClouc
+ double hpu(DefaultDirectedGrap
SCU
Set<ICU> scuMembers
+ Set<CloudSkill> getCloudSkills(
+ Set<HBSCommunicationAPI> ge
+ double getDuration()
+ double getHPU()
+ Set<String> getLocations()
+ double getPrice()
+ Set<ICU> getScuMembers()
+ double getUtilization()
+ void scalein(Set<CloudSkill> ch
+ void scaleout(Set<CloudSkill> c
+ void setScuMembers(ArrayList<
+ void setScuMembers(Set<ICU>
archetype
ArcheType
+ String archeTypeName
+ String domainName
+ ArrayList<String> typeOfSolutio
+ ArcheType()
ICU
+ Set<CloudSkill> getCloudSkills(
+ Set<HBSCommunicationAPI> ge
+ double getDuration()
+ double getHPU()
+ Set<String> getLocations()
+ double getPrice()
+ double getUtilization()
HBSMessage
String msg
+ String getMsg()
+ void setMsg(String msg)
Skill
String description
String name
+ String getDescription()
+ String getName()
+ void setDescription(String descr
+ void setName(String name)
CloudSkill
- Skill skill
- SkillLevel skilllevel
+ Skill getSkill()
+ SkillLevel getSkilllevel()
+ void setSkill(Skill skill)
+ void setSkilllevel(SkillLevel skillle
HBSRequest
String description
+ String getDescription()
+ void setDescription(String descr
SkillLevel
int level
+ int getLevel()
+ void setLevel(int level)

Fig. 2. Example of some Java-based APIs for clouds of HBS.

types of benefits. The `VieCOMHBSImpl` class describes the collection of APIs that can be used to discover and invoke HBS, as described in Table 1. Currently, we simulate our cloud of HBS. The HBS can be accessed via APIs described in `VieCOMHBSImpl`. 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).

These APIs provide different ways to acquire and interact with HBS. How the HBS's performance management is supported despite the fact these APIs do not tell if the HBS really continues to work even though being suspended via APIs. From the consumer perspective, the HBS will receive corresponding requests (based on APIs) and s/he will understand what the messages mean. In principle the HBS should follow the messages requested by the consumer but whether the HBS really follows the request or not is a different aspect, as humans may not necessary strictly follow requests, even they must be. What happens inside an HBS work cannot be controlled by the cloud. However, two possibilities could be supported. First, the cloud of HBS can control the quality of HBS and decide how to utilize the HBS based on quality of results (e.g. time, cost and quality of data) delivered by the HBS. Similarly, the consumer can also control the quality of the HBS the consumer pays for. These control mechanisms are complex enough for being out of this paper scope and we have developed some solutions in Ref. 22. Still, due to the nature of human works, not all human activities within the HBS can be measured and controlled.

2.5. *Conceptual architecture for hybrid unit as a service*

Based on the conceptual models of clouds of hybrid services, we describe a conceptual architecture for establishing a cloud of hybrid services. Figure 3 outlines our conceptual architecture for provisioning and programming hybrid service units. At the lowest level, software, people and things can be provisioned by interfacing and integrating them to the *Service-based Middleware*. Using this middleware, we enable different types of integration for software, people and things due to their different interaction models. The *Service-based Middleware* basically provisions HBS and SBS to the consumer via programmable, extensible APIs, e.g. based on the list of APIs that we present in Sec. 2.4. The *Service-based Middleware* utilizes our concepts by abstracting software, things and people using SBS and HBS unit model. This allows consumers to access software, people and things via a uniform way in which SBS and HBS will be mapped to underlying software, things, and people. To ensure the proper operations of this cloud, we need to implement *Runtime Monitoring and Enforcement* (e.g. for monitoring and enforcing costs, incentives and quality), *Communication* (e.g. for supporting the communication among HBS and SBS), *Service Life-cycle Management* (e.g. for supporting HBS selection and formation), and *Capability/Profile Management* (e.g. for managing service capabilities and HPU).

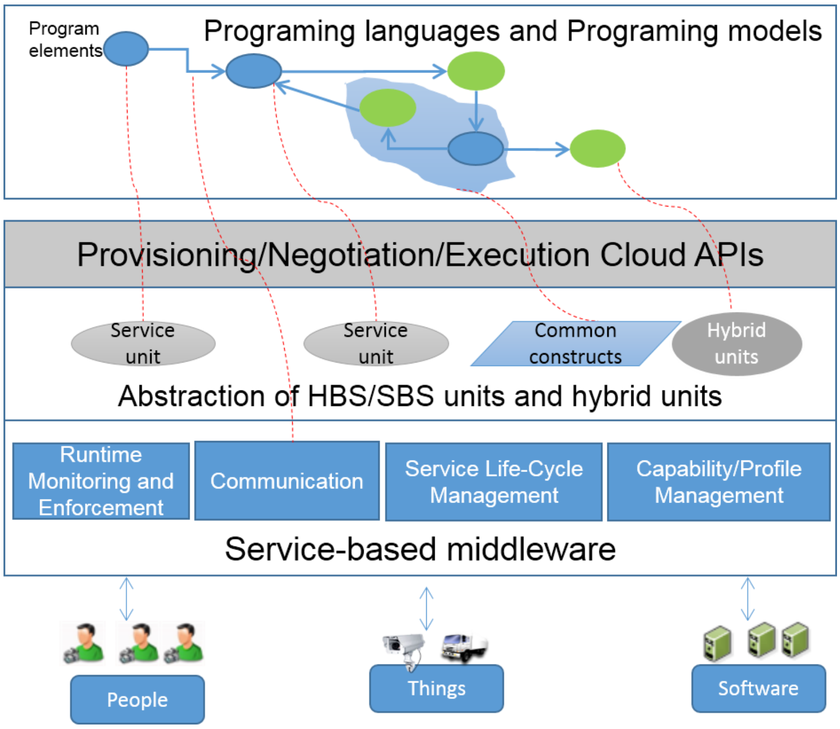
Program elements
Programing languages and Programing models
Provisioning/Negotiation/Execution Cloud APIs
Service unit
Service unit
Common constructs
Hybrid units
Abstraction of HBS/SBS units and hybrid units
Runtime Monitoring and Enforcement
Communication
Service Life-Cycle Management
Capability/Profile Management
Service-based middleware
People
Things
Software

Fig. 3. Conceptual architecture for provisioning hybrid SBS/HBS services.

The *Service-based Middleware* will be the core of a hybrid service provisioning platform. Atop this, one can program HBS and SBS by using *Provisioning/Negotiation/Communication Cloud APIs*. In the next sections, we describe some utilization possibilities and how to program hybrid services.

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. It 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 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 HPUs 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.

Figure 4 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. These operations are carried out by the *HBS Formation* service which implements and integrates different algorithms for handling requests of HBS, each suitable for specific situations. *Change Adaptation* also decides whether a change should be applied to SBS by sending change request to the *SBS Adaptation* service which will perform the change and modify the task graph accordingly. When an HBS deals with a task graph, the HBS can change the task graph and its required HPUs (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*.

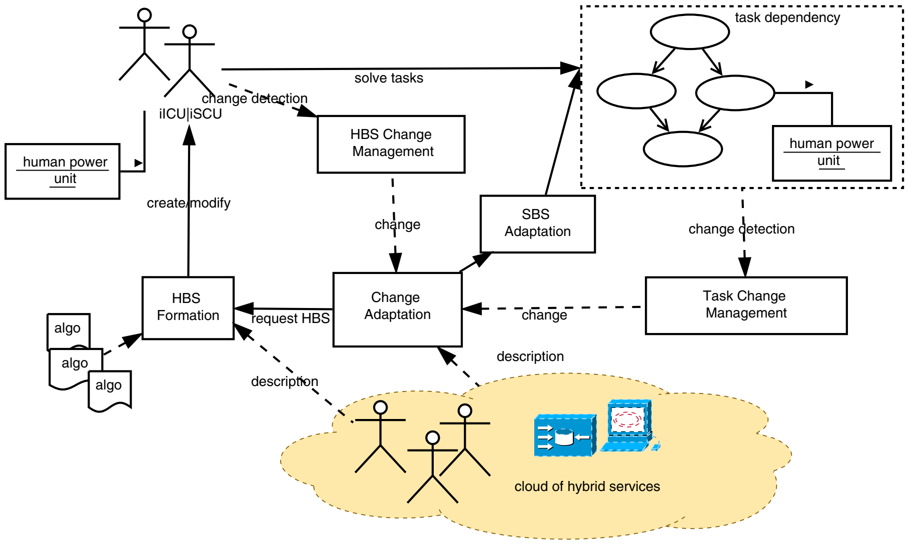

Fig. 4. Conceptual architecture.

At the time of writing, we have developed an SCU provisioning platform for forming, managing and controlling quality of SCUs for independent tasks.[22] This platform utilizes HBS from simulated ICU clouds based our concepts and APIs to form quality-aware SCUs and we also developed elasticity rules for adapting ICU and SBS.[23] The SCU expansion and reduction for dependent and evolving tasks are currently being prototyped. In the next section, we explain some concepts of programming hybrid services that are the key elements of the architecture depicted in Fig. 4.

4. Programming Hybrid Services

In this section, we discuss some programming primitives for hybrid services that can be applied to the complex framework that we mentioned before. Such primitives can be used in different components, such as *HBSFormation* and *ChangeAdaptation*, in our framework described in Fig. 4. In illustrating programming examples, we consider a virtualized cloud of hybrid services that are built on top of our cloud of HBS and real-world clouds of SBS. Consequently, we will combine our APIs, described in Sec. 2.4, with existing client cloud API libraries. Our goal in this section is not to present specific algorithms, e.g. for *HBSFormation*, adaptation strategies, e.g. for *ChangeAdaptation*, or specific applications to solve specific tasks. Instead, we present how different algorithms, strategies or applications could be developed and integrated into our framework.

4.1. *Modeling HPU-aware task dependency graphs*

4.1.1. *Task dependency graphs*

Our main idea in modeling HPU-aware task dependencies is to link tasks that are required for *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

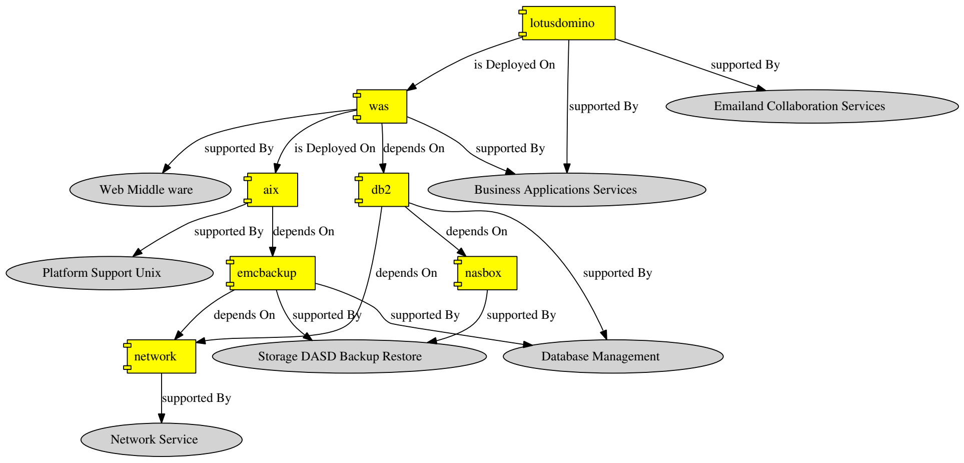

Fig. 5. An example of HPU-aware dependency graph. A component box describes a software and its problems (*ITProblem* node). An eclipse describes management skills (*Management* node).

factors will be used to determine the required HPU for a task, to select *iICU* and members for *iSCU*, and to build the connectedness for SCUs.

4.1.2. *Examples and implementation*

Figure 5 presents an example of a dependency graph of an IT system linked to management skills. In this system, we have a LotusDomino system (described by `lotusdomino`) deployed in a Web Application Server (described by `was`). The Web Application Server is deployed on an AIX server (described by `aix`) and depends on a DB2 server (described by `db2`). The DB2 server depends on a NAS box (described by `nasbox`) and a network (described by `network`). The AIX server is dependent on an EMC backup system (described by `emcbackup`) which depends on `network`. Each software node in the IT system has different requirements for HBS in order to solve IT problems arisen.

In our implementation of dependency graph, we use JGraphT.[j] We define two main types of Node — *ITProblem* and *Management*. All relationships are dependency. It is also possible to use TOSCA[24] 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 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

[j]http://jgrapht.org/.

for HBS work (e.g. requiring HBS to utilize specific SBS in order to produce the result where SBS is provisioned by the consumer).

Example: Listing 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. 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 and interaction within the *iSCU*. 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.

4.3.1. *Selecting resources for iSCU*

Figure 6 describes a general concept of how iSCU forming algorithms work. To form an iSCU, we need to consider both Business-as-Usual (BAU) and corrective

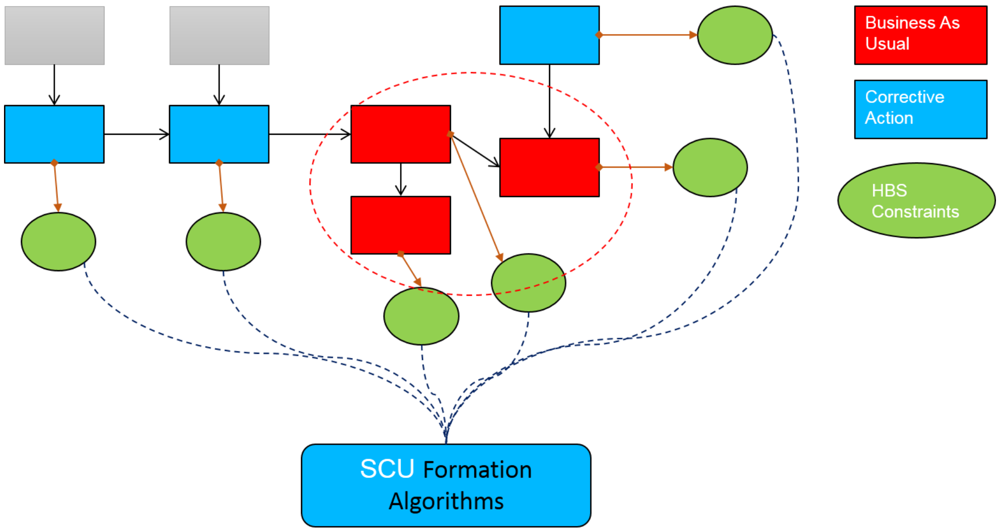

Fig. 6. General model for forming iSCU.

action (CA) cases. 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 BAU cases.
- Step 2: Determine $DG_{CA} \subseteq DG$ that includes tasks that should be taken into account under 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 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

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.
MinCostWithAvail NPathUnDirected	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.

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 2 describes an example of forming an SCU.

```
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 2. Example of forming iSCU by minimizing cost and considering no direction.

4.3.2. *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 $hbs_i \in iSCU$.

```
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 3. Example of star-based iSCU using Dropbox as a communication hub.

- *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 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.

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 HPU is changed. By capturing the change of the task graph, we can decide to scale in/out the *iSCU*. Listing 4 describes some primitives for scaling in/out *iSCU* based on the change of HPU.

```
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 4. Example of elasticity for SCU based on task graph change.

5. Related Work

Although both humans and softwares can perform similar work and several complex problems, both of them are esential 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 offer different possibilities to acquire SBS on-demand, however, similar efforts for HBS are missing today.

Cloud models and APIs for HBS: Tai *et al.*[25] outlined several research questions in cloud service engineering to support "everything is a service" in which services can be provided/integrated from different providers and charged based on different costs and values. However, contemporary systems focus only on SBS. Several frameworks for engineering cloud applications based on different IaaS, PaaS and SaaS, such as Aneka.[26] BOOM[27] have been introduced. Generally, they utilize software-based cloud resources via different sets of APIs, such as JClouds, Boto,[k] and OpenStack,[l] to develop applications under different programming models, such as MapReduce and dataflows. These frameworks do not consider hybrid services consisting of SBS and HBS, while our work supports conceptualizing and providing programming techniques for both SBS and HBS. To our best knowledge, there is no other work that proposes HBS cloud models.

Programming HBS and SBS in a unified way: Most clouds of SBS offering different possibilities to acquire SBS on-demand. However, researchers have not devoted similar efforts for HBS. A common way to utilize human capabilities is to exploit human computation programming frameworks, e.g. Crowdforge[28] and TurKit[29] and Jabberwocky framework,[30] for utilizing crowds for solving complex problems.[3,31] However, these works 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:[1] (i) using plugins to interface to human, such as BPEL4People[b] or tasks integrated into SQL processing systems,[9] (ii) using separate crowdsourcing platforms, such as MTurk,[m] and (iii) using workflows, such as Turkomatic.[6] 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 that requires human capabilities. In terms of communication models and coordination models, existing models also support messages push/pull/mediator, but they are platforms/middleware built-in rather than reusable programming primitives of programming models. Our work in this paper does not focus on managing and coordinating tasks but by proposing high-level APIs for HBS in a similar manner to that for SBS, our work could

[k]http://boto.s3.amazonaws.com/index.html.
[l]http://www.openstack.org/.
[m]Amazon mechanical turk, 2011. Last access: 27 Nov 2011.

foster the utilization of several HBS and SBS from different clouds based on cloud business models for different task management and coordination strategies.

Software tools for HBS: Some recent efforts have been devoted for software engineering tools of human-services, such as Ref. 32, and general-purpose programming languages for human computation, such as CrowdLang.[33] While they call for a better software engineering and programming languages support for human-centric systems, they do not address issues related to human services provisioning, e.g. using cloud and service models. Although we do not develop new general-purpose programming languages, we believe that if these works need to utilize human capabilities and software services in a large-scale, on-demand, pay-per-use fashion, then our models and techniques can be integrated into these software tools and languages.

Overall, compared with related 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 have described main concepts for establishing clouds of hybrid services, covering several aspects, like conceptual models and provisioning architectures for communication, pricing and incentive models, and programming APIs. Based on that, we present general frameworks and programming APIs to describe where and how hybrid services can be programmed.

In this paper, we focus on designing models, frameworks and APIs, and illustrating programming examples. We have presented a broad view on conceptualizing and programming hybrid services but have not addressed detailed activities in provisioning and managing the operation and interaction within HBS clouds as well as ICU/SCU. They will be subjects of several future research activities. Further real-world experiments should be conducted in the future to demonstrate the benefits of programming HBS and SBS in the same system. With respect to the software development for our concepts, we are currently working on programming elements/constructs/patterns for hybrid services that consider different relationships and cost/quality as first class entities in our programming models. Another direction is to work on hybrid service life-cycle management. This is also strongly related to how to monitor and enforce pricing and incentive strategies within a cloud infrastructure of hybrid services. Furthermore, we are also working on the integration with programming languages for social collaboration processes[5] using hybrid services. Other related aspects, such as pricing models and contract negotiation protocols, will be also investigated.

Acknowledgments

This paper is an extended version of the paper published in Ref. 11 The work mentioned in this paper is partially supported by the EU FP7 SmartSociety.[n]

References

1. S. Dustdar and H. L. Truong, Virtualizing software and humans for elastic processes in multiple clouds — a service management perspective, *IJNGC* **3**(2) 2012.
2. D. Schall, H. L. Truong and S. Dustdar, Unifying human and software services in web-scale collaborations, *IEEE Internet Comput.* **12**(3) (2008) 62–68.
3. A. Doan, R. Ramakrishnan and A. Y. Halevy, Crowdsourcing systems on the world-wide web, *Commun. ACM*, **54**(4) (2011) 86–96.
4. D. Oppenheim, L. R. Varshney and Y.-M. Chee, Work as a service, in *ICSOC*, eds. G. Kappel, Z. Maamar and H. R. Motahari Nezhad, Lecture Notes in Computer Science, Vol. 7084 (Springer, 2011), pp. 669–678.
5. V. Liptchinsky, R. Khazankin, H.-L. Truong and S. Dustdar, Statelets: Coordination of social collaboration processes, in *14th Int. Conf. Coordination Models and Languages* (*Coordination 2012*), Stockholm, Sweden, June 2012.
6. A. P. Kulkarni, M. Can and B. Hartmann, Turkomatic: Automatic recursive task and workflow design for mechanical turk, in *Proc. 2011 Annual Conference Extended Abstracts on Human factors in Computing Systems*, CHI EA '11 (ACM, New York, NY, USA, 2011), pp. 2053–2058.
7. D. W. Barowy, E. D. Berger and A. McGregor, 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.
8. H. S. Baird and K. Popat, Human interactive proofs and document image analysis, in *Proc. 5th Int. Workshop on Document Analysis Systems V*, DAS '02 (Springer-Verlag, London, UK, 2002), pp. 507–518.
9. A. Marcus, E. Wu, D. Karger, S. Madden and R. Miller, Human-powered sorts and joins, *Proc. VLDB Endow.* **5** (2011) 13–24.
10. S. Tai, P. Leitner and S. Dustdar, Design by units: Abstractions for human and compute resources for elastic systems, *IEEE Internet Comput.* **16**(4) (2012) 84–88.
11. H. L. Truong, S. Dustdar and K. Bhattacharya, Programming hybrid services in the cloud, in *ICSOC*, eds. Chengfei Liu, Heiko Ludwig, Farouk Toumani and Qi Yu, Lecture Notes in Computer Science, Vol. 7636 (Springer, 2012), pp. 96–110.
12. P. Mell and T. Grance, The NIST definition of cloud computing, NIST Special Publication 800-145 (September 2011).
13. H. L. Truong, S. Dustdar, D. Baggio, S. Corlosquet, C. Dorn, G. Giuliani, R. Gombotz, Y. Hong, P. Kendal, C. Melchiorre, S. Moretzky, S. Peray, A. Polleres, S. Reiff-Marganiec, D. Schall, S. Stringa, M. Tilly and H. Q. Yu, Incontext: A pervasive and collaborative working environment for emerging team forms, in *SAINT* (IEEE Computer Society, 2008), pp. 118–125.
14. O. Nov, D. Anderson and O. Arazy, Volunteer computing: A model of the factors determining contribution to community-based scientific research, in *Proc. 19th Int. Conf. World wide web*, WWW'10 (ACM, New York, NY, USA, 2010) pp. 741–750.

[n]http://www.smart-society-project.eu/.

15. A. J. Quinn and B. B. Bederson, Human computation: A survey and taxonomy of a growing field, in *CHI*, eds. D. S. Tan, S. Amershi, B. Begole, W. A. Kellogg and M. Tungare (ACM, 2011), pp. 1403–1412.
16. W. Mason and D. J. Watts, Financial incentives and the "performance of crowds", in *Proc. ACM SIGKDD Workshop on Human Computation*, HCOMP'09 (ACM, New York, NY, USA, 2009), pp. 77–85.
17. O. Tokarchuk, R. Cuel and M. Zamarian, Analyzing crowd labor and designing incentives for humans in the loop, *IEEE Internet Comput.* **16**(5) (2012) 45–51.
18. O. Scekic, H.-L. Truong and S. Dustdar, Incentives and rewarding in social computing, *Commun. ACM* **56**(6) (2013) 72–82.
19. J. J. Horton and L. B. Chilton, The labor economics of paid crowdsourcing, in *Proc. 11th ACM conf. Electronic commerce*, EC'10 (ACM, New York, NY, USA, 2010), pp. 209–218.
20. S. Dustdar and K. Bhattacharya, The social compute unit, *IEEE Internet Comput.* **15**(3) (2011) 64–69.
21. O. Scekic, H.-L. Truong and S. Dustdar, Programming incentives in information systems, in *25th Int. Conf. Advanced Information Systems Engineering* (*CAISE 2013*), Valencia, Spain, 17–21 June 2013.
22. S. Dustdar, M. Z. C. Candra and H.-L. Truong, Provisioning quality-aware social compute units in the cloud, in *Service-Oriented Computing — Proc. 9th Int. Conf. ICSOC 2013* (Springer, Berlin, Germany, 2–5 December 2013).
23. M. Z. C. Candra, H. L. Truong and S. Dustdar, Modeling elasticity trade-offs in adaptive mixed systems, in *WETICE*, eds. S. Reddy and M. Jmaiel (IEEE, Hammamet, Tunisia, 17–20 June 2013), pp. 21–26.
24. T. Binz, G. Breiter, F. Leymann and T. Spatzier, Portable cloud services using tosca, *IEEE Internet Comput.* **16**(3) (2012) 80–85.
25. S. Tai, J. Nimis, A. Lenk and M. Klems, Cloud service engineering, in *Proc. 32nd ACM/IEEE Int. Conf. Software Engineering — Volume 2*, ICSE'10 (ACM, New York, NY, USA, 2010), pp. 475–476.
26. R. N. Calheiros, C. Vecchiola, D. Karunamoorthy and R. Buyya, The aneka platform and qos-driven resource provisioning for elastic applications on hybrid clouds, *Future Generation Comp. Syst.* **28**(6) (2012) 861–870.
27. P. Alvaro, W. R. Marczak, N. Conway, J. M. Hellerstein, D. Maier and R. Sears, Dedalus: Datalog in time and space, in *Datalog*, eds. O. de Moor, G. Gottlob, T. Furche, and A. J. Sellers, Lecture Notes in Computer Science, Vol. 6702 (Springer, Oxford, UK, 16–19 March 2010), pp. 262–281.
28. A. Kittur, B. Smus, S. Khamkar and R. E. Kraut, Crowdforge: Crowdsourcing complex work, in *Proc. 24th Annual ACM Symp. User Interface Software and Technology*, UIST'11 (ACM, New York, NY, USA, 2011), pp. 43–52.
29. G. Little, L. B. Chilton, M. Goldman and R. C. Miller, Turkit: Tools for iterative tasks on mechanical turk, in *Proc. ACM SIGKDD Workshop on Human Computation*, HCOMP'09 (ACM, New York, NY, USA, 2009), pp. 29–30.
30. S. Ahmad, A. Battle, Z. Malkani and S. Kamvar, The jabberwocky programming environment for structured social computing, in *Proc. 24th Annual ACM Symp. User Interface Software and Technology*, UIST '11 (ACM, New York, NY, USA, 2011), pp. 53–64.
31. A. Brew, D. Greene and P. Cunningham, Using crowdsourcing and active learning to track sentiment in online media, in *Proc. 2010 Conf. ECAI 2010: 19th European Conference on Artificial Intelligence*, (IOS Press, Amsterdam, The Netherlands, 2010), pp. 145–150.

32. C. Dorn and R. N. Taylor, Co-adapting human collaborations and software architectures, in *ICSE*, eds. M. Glinz, G. C. Murphy and M. Pezzè (IEEE, 2012), pp. 1277–1280.
33. P. Minder and A. Bernstein, Crowdlang: A programming language for the systematic exploration of human computation systems, in *SocInfo*, eds. K. Aberer, A. Flache, W. Jager, L. Liu, J. Tang and C. Guéret, Lecture Notes in Computer Science, Vol. 7710 (Springer, Lausanne, Switzerland, 5–7 December 2012), pp. 124–137.