Virtualizing Software and Humans for Elastic Processes in Multiple Clouds– a Service Management Perspective

Schahram Dustdar and Hong-Linh Truong
Distributed Systems Group, Vienna University of Technology, Austria

There is a growing trend of combining human-based computation with machine-based computation to solve complex problems which cannot be answered with machine-based computation alone. From the computing perspective, integrating machine-based computing elements with human-based computing elements and provisioning them under the same model will facilitate the resource elasticity required by complex applications. Although certain works investigate techniques for integrating human-based computing elements with machine-based computing elements, existing computing models for such integrated computing systems are very limited. In fact, the architectures, interconnections, non-functional properties of human-based computing elements are very different from that of contemporary machine-based counterparts. Human-based computing elements are built based on social and bio concepts, thus their architectures, interconnects and non-functional properties are extremely complex and dynamic, compared with that of machine-based computing elements. In this paper, we examine fundamental issues in virtualizing human-based computing elements and machine-based computing elements using service-oriented computing concepts in order to create highly scalable computing systems of hybrid services to support the elasticity of software and people in complex applications. We will outline our Vienna Elastic Computing Model which aims at introducing techniques and frameworks to support multi-dimensional elastic processes atop multiple cloud systems of software-based and human-based services. This paper will analyze several service management issues to support the virtualization of machine-based and human-based computing elements to support such elastic processes.

Keywords: human-based computing, elasticity, cloud computing, social computing

1. INTRODUCTION

Over the last many years, several architectures of computing systems have been developed. The Symmetric Multiprocessing (SMP) architecture introduces multiple processors sharing the same memory. The parallel cluster architecture includes multiple machines, each may consists of several computing CPU/cores. The Grid computing architecture goes further by connecting multiple clusters and SMP machines from different computing sites. And, recently, the cloud computing architecture consolidates and aggregates several machines and software to virtualize and provision them by utilizing virtualization techniques. These computing architectures have basically several hardware/software components, interconnected by different interconnect networks and they represent so-called machine-based computation via a combination of hardware and software. In a machine-based computing system, basically, at the lowest level, we have machine-based computing elements connected by networks. A machine-based computing element (MCE) typically has, for example, a CPU, memory and hard disk. Depending specific machine-based computing systems, e.g., clusters or SMP, multiple programming languages have been developed to exploit MCEs and different coordination models have been developed to support multiple, concurrent tasks executed in machine-based computing systems. Until now, machine-based computing systems scale well and could collectively aggregate the capability of millions of MCEs, such as in the case of SETI@HOME[set 2011], solving complex problems.

On the one hand, recently, the Internet and underlying powerful, pervasive machine-based computing systems have enabled us to harness vast human capabilities around the world. Hu-

Author's address: Argentinierstrasse 8/184-1, 1040 Vienna, Austria.
e-mail: {dustdar, truong}@infosys.tuwien.ac.at

man capabilities can be exploited in a similar way to MCEs: humans perform certain types of computing activities (with/without assisted MCEs). In this way, humans establish another type of computing elements namely human-based computing elements. Human-based computing elements (HCEs) are also called living computing elements because the core of an HCE is a human, as opposed to MCEs which are non-living. Literature has shown that humans have been performing computing since the early history. However, in this paper, we want to emphasize how humans and software are combined in order to support large-scale computation. In our view, "software services" empowering human capabilities have a crucial role, allowing us to provide, for example, interfaces, programming models, and coordination models for human capabilities, in a similar way to that for machine-based computing systems, thus enabling us to harness massive, collective capabilities of humans on an integrated and automatic manner.

When HCEs interact with machine-based computing systems, MCEs and HCEs are being integrated. Although certain works investigate HCEs and how to integrate them with MCEs, existing computing models for such integrated computing systems are very limited. In fact, the architectures, interconnects, and non-functional properties of humans are very different from that of contemporary MCEs. Therefore, building and utilizing human/software-based computing systems are extremely challenging. To date, the developer of MCEs knows, more or less, a lot of knowledge about computing capabilities and topologies of interconnected networks, but the developer of human/software-based computing systems has very little knowledge about capabilities as well as interconnects of HCEs. For example, given a cluster of MCEs, we can easily obtain the network of the cluster but given a set of HCEs (e.g., a team of people), we might not know the topology of their interconnectedness. Furthermore, given an HCE, its computing capability might not be powerful, e.g., the manager of a software team might not know how to program Web services, but it can potentially invoke many other capable HCEs, e.g., members of the team or external collaborators, that we might not know in advance when deciding to employ the HCE. This poses several challenges in building and utilizing HCE systems efficiently, let alone the combination of MCEs and HCEs.

In this paper, we examine the architectures, interconnections, and non-functional properties of HCEs, and how HCEs can be combined with MCEs in order to facilitate the gathering of collective computing capabilities of both MCEs and HCEs in a large scale. We examine fundamental issues of HCEs and how they can integrate with MCEs to create complex, highly scalable computing systems. We will outline our Vienna Elastic Computing Model which aims at introducing techniques and frameworks to support multi-dimensional elastic processes atop multiple cloud systems of software-based and human-based services. We will analyze several service management issues in order to support the virtualization of MCEs and HCEs under the same model to support such elastic processes.

The rest of this paper is organized as follows: Section 2 discusses our motivation, approaches and related work. Section 3 outlines the Vienna Elastic Computing Model. Section 4 analyzes service management issues and presents our approaches to these issues in order to virtualize human-based computing elements under the service model. We conclude the paper and outline our future plan in Section 5.

2. MOTIVATION, APPROACH, AND RELATED WORK

2.1 Motivation

Let us elaborate some typical scenarios in which human-based computation is exploited in order to characterize HCEs and hybrid systems of HCEs and MCEs and to reflect the challenges when integrating them.

2.1.1 *Discovering patterns in images.* Figure 1 shows a scenario of using MCEs and HCEs in finding patterns in satellite images. The scenario is based on the search of Jim Gray missing [Hafner 2007] as well as on situational analysis in disaster scenarios [Voigt et al. 2007]. In this scenario, several images taken from satellites are processed by various software (machine-based computing elements) in order to detect patterns required for specific purposes (such as missing people or victims in a disaster). The problem is that, in many cases, software cannot detect certain patterns. Therefore, humans must perform the pattern analysis. In this case, humans act as a computing resource (human-based computing element). To scale with complex problems (e.g., the number of images and the time required for analyzing images) HCEs are based on either crowds of novice users/volunteers or professional teams, such as disaster response experts and geo-scientists. While both crowds and professional teams need seamless interactions with MCEs, they pose different requirements. For crowds, every HCE can work independently and there might not be any constraints on the performance and quality of the work. However, in the case of professional teams, performance and quality of results constraints are important. Furthermore, both cases require different programming models in order to utilize their capabilities. However, both MCEs and HCEs should be integrated seamlessly into a single application. This is different from existing systems in which HCEs are utilized separately from MCE-based systems.

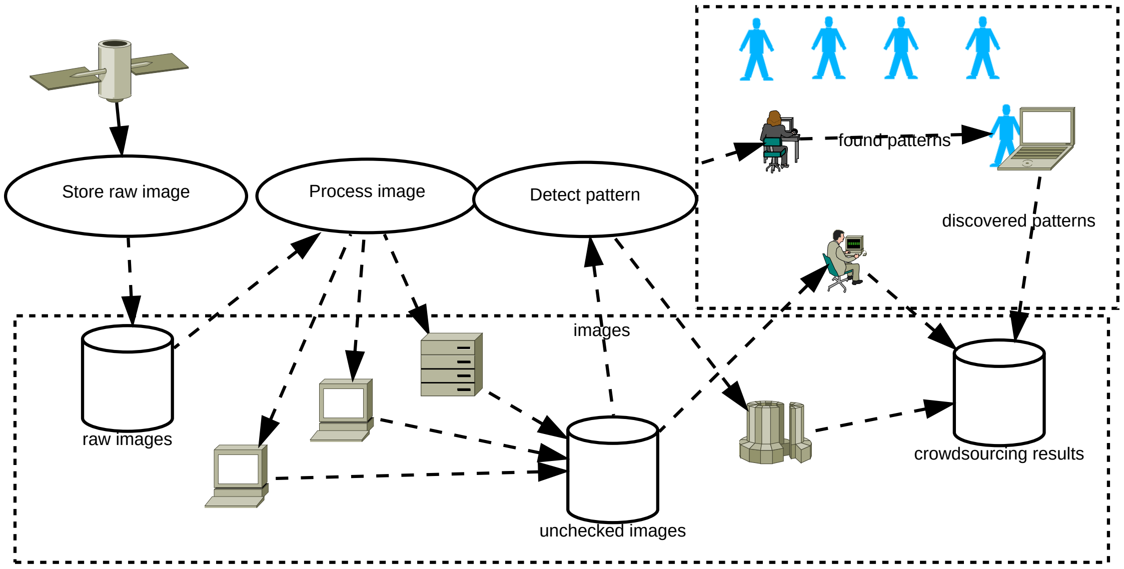

Figure. 1: Pattern discovery using MCEs and HCEs

2.1.2 *Quality evaluation in multi-scale simulations.* Figure 2 shows another example of utilizing HCEs and MCEs. Several long-running scientific simulations can be performed by MCEs, but scientists are required to analyze quality of intermediate simulation results during the simulations. The need to have scientists/HCEs in the loop is due to the complexity of simulations, such as in multi-scale simulations, and due to the optimization of MCEs used, e.g., simulations should be stopped if bad results are generated. In addition to the issues mentioned in the previous scenario, there exist work delegations that occur in cliques (rather than crowds) of scientists. For example, the quality of a simulation result can be first evaluated by PhD students, then verified by post-docs before checked by professors. Furthermore, HCEs must fulfill several constraints on computing capabilities (e.g., able to handle macro bone simulation using finite element methods) and non-functional properties (e.g., the availability).

2.1.3 *Virtualizing HCEs and MCEs for elastic processes.* The above-mentioned scenarios illustrate the need for scaling in/out both machine-based and human-based computing elements. For example, in cases of disaster scenarios or emergency situations, several types of data (e.g., images and events) need to be analyzed quickly by a large number of humans (provided by crowds) by specialists (provided by professional teams). Furthermore, human-based computing

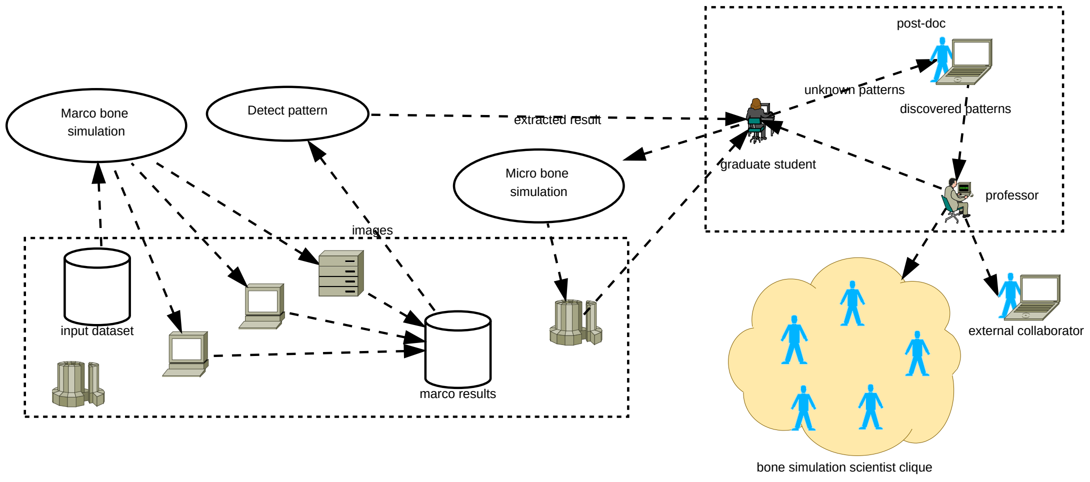

Figure. 2: Using human-based computing elements to evaluate quality of data in multi-scale simulations

elements need to *collaborate* to solve the problem, not just being invoked separately. Essentially, we need to build the elasticity of hybrid computing elements within these complex applications. Moreover, these scenarios illustrate the need to be *proactive* in scaling in/out of MCEs and HCEs to solve problems under *several constraints*, such as time, quality of results, and compliance laws. These applications should be able to decide when and how to take into account HCEs instead of MCEs actively, rather than just posting the tasks to crowds and waiting for some HCEs to take the tasks, as shown in current crowdsourcing systems. As a result of these requirements, we need techniques to support the integration of HCEs and MCEs into a single system that goes beyond contemporary crowdsourcing models. We must virtualize software and human in such a way that facilitates the management of software and human and enables the programming of them in a similar way.

2.2 Concepts of machine and human computation

Table I describes an analogy between MCE- and HCE-based computing systems. As shown in Table I, there are substantial differences between MCEs and HCEs in terms of architecture, communication and programming models. However, it is of paramount importance to virtualize them under the same model so that we can utilize them in a similar way. In our work, we aim at conceptualizing them under the service model. To date, the service-oriented computing (SOC) model has successfully virtualized hardware and software capabilities of MCEs and provided capabilities of MCEs through well-defined service interface, service information and service interaction. For example, one can use a well-defined service interface to launch a (virtual) machine in a cloud-based Infrastructure-as-a-Service (e.g., in Amazon EC2[1]), to translate a document using a cloud-based Software-as-a-Service (e.g., using the Google Translation service[2]), or to run a data query program in a cloud-based Platform-as-a-Service (e.g., using Microsoft Azure[3]). Similar to how SOC techniques have been applied to the virtualization of MCEs and to the provisioning of MCEs under cloud computing models, we could provide different types of services for HCEs, such as human-based services for individuals or teams and hybrid services including both MCEs and HCEs. From these types of services, we can build elastic processes/workflows (e.g., like in the above-mentioned scenarios) which eventually can be provided under the service model.

[1]`http://aws.amazon.com/ec2/`
[2]`http://translate.google.com/`
[3]`http://www.microsoft.com/windowsazure/`

Table I: Analogy between machine and human computation

Categories	**Properties**	**Machine-based computing systems**	**Human-based computing systems**
Computing element	basic processor	core/CPU and peripheral devices	human brain with computing assisted devices used by humans
Computing architecture	system architecture	SMP, cluster, Grid, Cloud	individual, team, clique, crowd
	interconnect architecture	bus, star, ring, tree	free-scale (social) network
	communication protocol	TCP/IP, Myrinet, Infiniband	email, social network, blog, wiki
Programming model	basic task	computing thread/process	human activity
	task distribution	send tasks to schedulers in basic computing elements that execute the tasks automatically and return the results	send tasks to MCE-based assisted software and wait for humans to retrieve the tasks from the software; humans process the tasks and upload the results into the assisted software from which the results can be pulled or pushed.
	task intercommunication	exchanging messages among thread/processes using communication APIs that are based on different programming models	different types of interactions among humans

2.3 Related work

Although both, MCEs and HCEs, can perform similar work, literature has intensive sources indicating which tasks MCEs can perform better than HCE and vice versa [Wu et al. 2004; Brill and Ngai 1999; Baird and Popat 2002]. Literature also indicates that MCEs and HCEs can coexist as well as that several complex problems need both of them in the same system. In our paper, we will not discuss the pros and cons of HCEs versus MCEs.

Several applications have shown how to utilize crowds for solving complex problems [Brew et al. 2010; Doan et al. 2011]. Many of these applications just discuss about how to utilize crowds inputs to determine solutions for specific problems; they do not discuss about how to integrate and virtualize human-based computing elements. From the programming language perspective, most efforts have been spent on extending existing languages (e.g., BPEL4People [bpe 2009]) or integrating with crowd sourcing platforms in specific query systems [Marcus et al. 2011]. In these efforts for human-based computation, the interaction between humans and MCE-based systems is mainly based on crowdsourcing platforms (e.g., Amazon Mechanical Turks [mtu 2011]). However, no existing programming language has been developed to consider humans as "processors", like in conventional programming languages.

Doan and his colleagues present a detailed discussion of how crowds are utilized in the Web [Doan et al. 2011]. However, they have not discussed how humans and software can be integrated into a single computing system.

While empowering collective intelligence by harnessing people capabilities has attracted a lot of research, our work differs from them as we examine how human capabilities can be harnessed via the service model so that they can seamlessly be integrated into large-scale, complex applications. Furthermore, in this paper we also present a detailed study of fundamental properties of HCEs and the hybrid computation model of MCEs and HCEs.

3. THE VIENNA ELASTIC COMPUTING MODEL

As we discussed in the motivation, several complex applications require techniques to take into account MCEs and HCEs in multiple clouds in an elastic manner. In other words, these applications need to scale in/out both MCEs and HCEs. Although current understanding of elasticity and elastic techniques in computing is mainly about machine-based resource elasticity [Chiu 2010], as we have discussed in [Dustdar et al. 2011], elasticity can be conducted in multiple dimensions, including resource, quality, and pricing elasticity. To support the development of applications with built-in multi-dimensional elasticity capabilities, we introduce the Vienna Elastic Computing Model (VieCOM) which is built based on the following key points:

— supporting multi-dimensional elasticity: our computing model supports the principles of elastic processes [Dustdar et al. 2011]. By considering these principles, we take into account different dimensions of elasticity. We consider scaling out/in MCEs and HCEs in our processes. Thus, we consider hybrid systems of MCEs and HCEs in our applications. Furthermore, we support price/reward/incentive models and quality models in the context of multiple clouds.

— following the SOC model: our computing model utilizes existing SOC techniques and concepts to virtualize MCEs and HCEs under the service model and to provision them under different forms of cloud systems.

— supporting an end-to-end approach, from modeling to execution: our computing model aims at providing techniques to cover different layers ranging from modeling to runtime management. This aims at closing the gap between the modeling techniques and runtime management techniques for elastic computing.

Figure 3 outlines layers in the Vienna Elastic Computing Model. We briefly explain them in the following:

— *Computing Element*: describes fundamental computing elements in VieCOM which can be machines or humans.

— *Virtualization*: describes virtualization techniques and frameworks that provision HCEs and MCEs under IT services whose properties can be elastic.
— *IT Elastic Service*: abstracts HCEs and MCEs under the service model. *IT Elastic Services* can be selected based on their properties and be invoked based on their well-defined service interfaces, such as based on SOAP or REST. *IT Elastic Services* can be elastic, e.g., their costs and quality can be dynamically changed, depending on specific contexts.
— *IT Elastic Process*: specifies IT processes built atop *IT Elastic Services*. An *IT Elastic Process* can be described in the form of workflows or distributed components.
— *Elastic Service*: specifies business service which consists of non IT services and of IT processes.
— *Elastic Runtime Management*: includes techniques and platforms to manage and execute *IT Elastic Services*, *IT Elastic Processes* and *Elastic Services* provisioned from different clouds.
— *Elastic Alignment*: specifies techniques and frameworks to support the alignment between *Elastic Service* and its corresponding IT process (specified by *IT Elastic Process*), and between *IT Elastic Process* and its *IT Elastic Service*.
— *Elasticity Modeling*: specifies techniques and frameworks for modeling elastic properties and trade-offs for services and processes.
— *Elasticity Monitoring*: specifies techniques and frameworks for monitoring the elasticity of services, processes and computing elements.
— *Application*: specifies different types of applications that are built by utilizing *Elastic Services*.

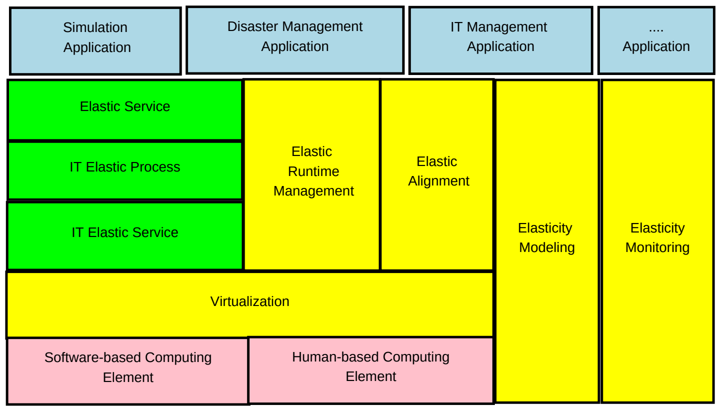

Figure. 3: Conceptual view of the Vienna Elastic Computing Model

In this paper, we will focus on the *Virtualization* and *IT Elastic Service* layers and discuss issues and state-of-the-art techniques that can support us to virtualize MCEs and HCEs into *IT Elastic Services*. We will focus on the following open questions:

— **Question 1**: how to utilize SOC techniques for virtualizing humans and how do we provision human capabilities in clouds?
— **Question 2**: how to model and manage capabilities and interactions MCEs and HCEs. This is essential for us to understand how to virtualize and manage them in a single system?
— **Question 3**: how to model interfaces for tasks and communications among HCEs and between HCEs and MCEs?
— **Question 4**: what are service relationships among HCEs?
— **Question 5**: which non-functional properties are important and how to specify and interpret them?

In the following section, we will discuss about current state of the art and some of our answers to these questions.

4. SERVICE MANAGEMENT APPROACHES AND ISSUES

4.1 Service models for virtualizing software- and human-based computing elements

In traditional service-oriented systems we have:

— software-based service (SBS): hardware and software are provisioned under the service model. The capabilities of MCEs can be accessed via well-defined service interfaces. This model is well developed and supported. Note that a service can be atomic or composite (internally presented by a composition that specifies how different services to be composed into a single service)

— service-based process/workflow: a set of connected activities, e.g., based on specific data and control flows, in which activities are mapped to SBS. Note that a process/workflow could be also provisioned as an SBS.

To virtualize HCEs under the service model and provision them in a similar manner to that for MCEs, we need to develop novel frameworks and techniques to support the development of these services, and subsequently, to develop complex applications (e.g., described by process and workflows) by utilizing these services. To this end, we have introduced two novel concepts to virtualize HCEs namely Human-provided Service (HPS) and Social Compute Unit (SCU). They are fundamental building blocks for the provisioning of clouds of human capabilities and hybrid systems of software and humans.

4.1.1 *Virtualizing individual capabilities.* the Human-provided Service (HPS) model [Schall et al. 2008] introduces techniques and frameworks for humans to specify their capabilities using Web services interfaces. The key concept is that human computing capabilities can be represented by so-called HPS which can be invoked by any software via Web service interfaces. In order to achieve this a middleware is introduced to act as an intermediate between humans and their corresponding HPS. By using HPS, we can assume that an individual human can be virtualized via the service model.

4.1.2 *Virtualizing team capabilities.* The Social Computing unit (SCU) [Dustdar and Bhattacharya 2011] introduces a concept of a team of individuals that can be selected and composed to perform complex tasks on-demand. An SCU is similar to a set of MCEs with communication and storage capabilities to perform the work. An SCU is also similar to a composite SBS but composes different humans and provides well-defined interfaces for any software to invoke it. Furthermore, an SCU has a life-cycle which can be created, virtualized, deployed and dissolved on-demand, similar to SBS.

The SCU service goes beyond the service model for individuals (e.g., HPS). A set of HCEs can be grouped into an SCU, which is established based on certain constraints, such as all HCEs have similar computing capabilities, similar social behaviors, organizational structures, domains, or locations.

4.1.3 *Hybrid systems of SBS and HBS.* By virtualizing HCEs we can provision human capabilities under human-based services (HBS). Thus there exist two new forms of HBS namely HPS (for individuals) and SCU (for composition of HCEs). By introducing HBS in a similar form for SBS, we can establish hybrid systems consisting of SBS and HBS. This will bring a new form of services that will not be found in conventional SBS-based systems or in uniform HBS-based systems: hybrid composite service of SBS and HBS whose aggregated services are software-based or human-based.

Under these types of services, we can develop hybrid processes/workflows which a process/workflow including atomic and composite SBS and HBS. However, techniques for modeling and managing such hybrid processes need to be developed. Currently, existing workflow techniques can only consider HCE in a simple way, e.g., via the utilization of WS-HumanTask [wsh 2009] and crowdsourcing systems [Barowy et al. 2011; mtu 2011].

4.1.4 *Clouds of hybrid services.* By utilizing the service model, different types of cloud computing systems, including uniform SBS, uniform HBS, and hybrid SBS/HBS, can be provisioned. Potentially, an application can be programmed to take into account SBS and/or HBS from these clouds in a similar way. First, we could develop programs that scale to a very large number of SBS and HBS using SOC techniques for discovering, comparing and composing SBS and HBS based on their capabilities, interfaces and non-functional properties. Second, since SBS and HBS are all service-based, we can provide an integration model that enables seamless execution of tasks across SBS and HBS and simplifies how HBS could be used in complex applications.

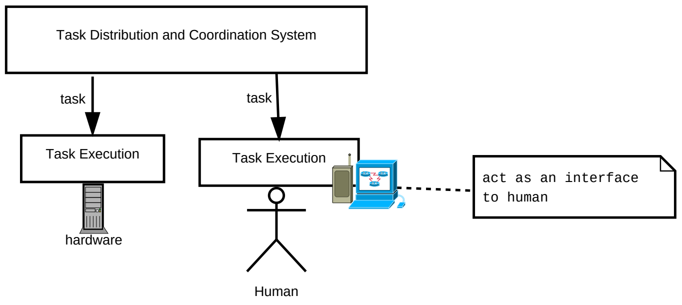
Task Distribution and Coordination System
task
task
Task Execution
Task Execution
hardware
Human
act as an interface to human

Figure. 4: Basic integration model

The basic integration model for SBS and HBS is shown in Figure 4 . Note that when receiving a task to be executed, an SBS or HBS can use different facilities to spawn new sub-tasks and ask for other SBS/HBS to process the newly-created sub-tasks (e.g., via delegation and escalation mechanisms). Despite several platforms have been developed for utilizing human computation capabilities, surprisingly, unlike for SBS-based computing systems, the coordination and task distribution for harnessing HCEs is quite simple. As shown in Table II, current support can be divided in three approaches: (i) using plug-ins to interface to human, such as BPEL4People or tasks integrated into SQL processing systems [Marcus et al. 2011], (ii) using separate crowdsourcing platforms, such as MTurk [mtu 2011], and (iii) using workflows, such as Turkomatic [Kulkarni et al. 2011]. A drawback is that all of them consider HCEs individually and HCEs in these systems have not been provisioned as HBS. As a result, an application must split tasks into sub-tasks that are suitable for individual HCEs, which do not collaborate each other, before the application can invoke HCEs to solve these sub-tasks. Furthermore, the application must join the results from several sub-tasks. This is not trivial for the application when dealing with complex problems required human capabilities. Therefore, two issues must be addressed: existing techniques must be extended to interface to HBS and SCU-based HBS should be considered. By taking into account SCU-based HBS, we expect that complex tasks can be tackled by SCUs whose members can decide how to split tasks and join the results in a way that is suitable for humans, e.g. based on context dependency rules for context-aware and social collaboration processes [Liptchinsky et al. 2012].

From the basic integration model discussed above, Figure 5 presents our vision on how we can utilize computing capabilities from clouds of SBS and HBS. Consider a developer has to write a program that scales in/out a large number of SBS and HBS. In her view, there exist multiple cloud computing systems of SBS and HBS. Thus, she needs to be able to design, submit, and monitor her programs executed in such systems. Regardless of whether SBS or HBS are needed for tasks, tasks in her programs are submitted to these systems in a similar way, based on common interfaces, capability description and discovery, and non-functional properties monitoring. To support this vision, essentially, we need to develop system software that can manage, discover, and monitor both SBS and HBS. Currently, such system software are missing.

Table II: Example of task distribution and coordination models

Category	Systems	Task Description	Task Distribution	Task Coordination	Notes
Language extension	[Marcus et al. 2011]	Specific task description integrated into SQL	Push by SQL processing systems	Split-and-join, centralized coordination	Use a procedure which waits for human input
	BPEL4People [bpe 2009]	WS-HumanTask [wsh 2009]	Push by BPEL engine	One-to-one , Centralized coordination	Plug-in to BPEL
	AutoMan [Barowy et al. 2011]	domain-specific language object	tasks are pushed by the runtime environment	centralized coordination	Tasks are designed in Automan, a domain-specific language based on Scala.
Platform	MTurk [mtu 2011]	Specific human-readable only description	humans put and get tasks into/from the platform	One-to-one, centralized coordination	Task is defined by a human and stored into MTurk. Then the task is retrieved by another human who performs the task and put the result into MTurk.
Workflow	Turkomatic [Kulkarni et al. 2011]	Human-readable task only	Tasks are arranged in workflows and pushed to humans	Complex tasks are automatically and iteratively divided into workflows by software and humans	Utilize MTurk

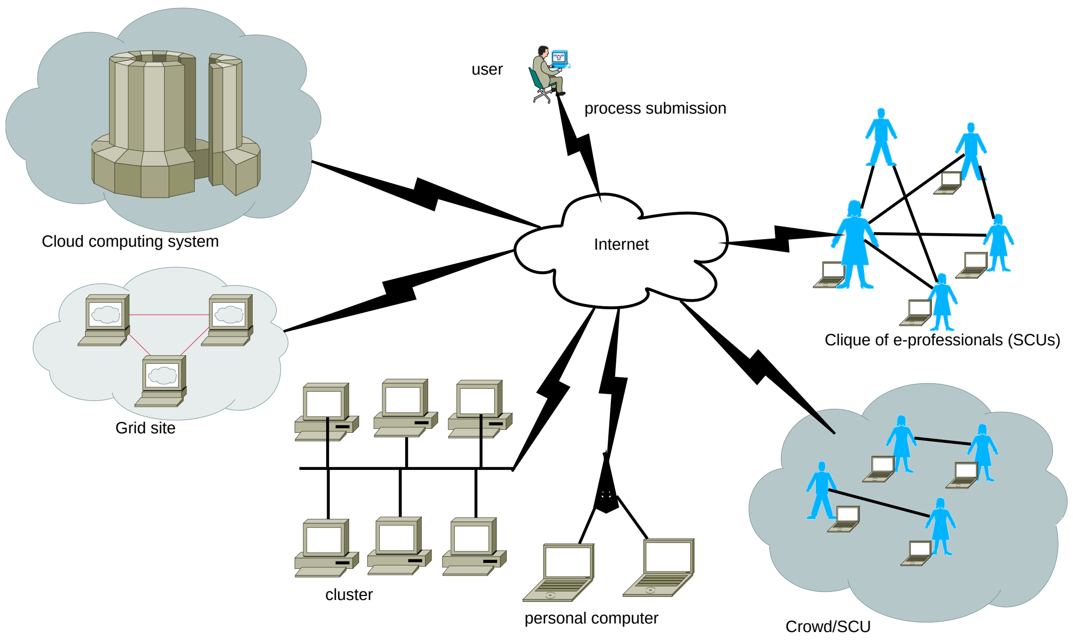

Figure. 5: Clouds of SBS and HBS computing systems

4.2 Describing and managing service information

First, let us consider the description for computing capabilities. With MCEs, we have description information about CPU/cores, memory, storage capabilities, software features, to name just a few. However, for HCEs, there exists no such a similar capability description. HCEs are currently characterized by vague description, such as skill and expertise information. Such information actually cannot describe which types of and how many activities that an HBS can handle. When a developer considers a possibility that a task in her programs should be mapped to an MCE or an HCE, e.g., between a software or a lawyer as indicated in [Markoff 2011], capability description between MCEs and HCEs must be compared. However, many existing systems, while are able to harness human capabilities, do not consider the role of the description of capabilities for HCEs. Table III shows examples of HCE capability description. There is no such a system that compares MCE capability versus HCE capability. Currently existing systems let people select the tasks themselves and, in these systems, types of tasks are statically designed either for MCE or HCE. This way may work in crowdsourcing platforms with volunteering HCEs or in uniform HCE-based systems for non-critical businesses. However, it will not support the utilization of MCE and HCEs capabilities in business critical setting in which the application would like to determine the capabilities of HCEs before handling the tasks to HCEs and/or to proactively, intelligently and dynamically decide whether an MCE or an HCE should be utilized based on specific context. By provisioning MCEs and HCEs under SBS and HBS, their capabilities can be modeled and described using similar specifications. Obviously, certain properties in HBS capability descriptions are different from that for SBS but this difference can be represented in different tuple values of the description. Since existing service capability description specifications have not been designed for accommodating human-specific capabilities, new extensions must be developed to support service information management in hybrid systems of SBS and HBS. However, we should note that being able to capture and manage SBS and HBS service information together only facilitates the capability comparison between SBS and HBS: this does not solve questions about whether an HBS can replace or be equivalent to an SBS. Specific algorithms must be developed for such comparison.

Table III: Example of current support in describing human computation capabilities

Systems	**Modeled capabilities**	**Description**
LinkedIn[LinkedIn]	individual profile	personal information, location, professional position
Live Person[Cohen 2010]	individual profile	personal information, cost, availability

Second, let us consider service information describing quality and compliance guaranteed by HBS. Such service information is crucial as potentially HBS can be taken from anywhere in the world, and unlike SBS, the quality of the task proceeded by HBS is hard to know and to predict. SBS has a well-researched descriptions for quality and compliance that cover several aspects, such as regulation and legal [Gangadharan and D'Andrea 2011]. However, we observe that it is not clear how to describe quality and compliance information for HBS. In particular, existing human descriptions have not specified quality of results, compliance, in crowdsourcing platforms. However, quality of results and regulatory compliance are crucial as applications built atop HBS can scale in/out HBS in very-large settings. In terms of service management, we need to provide techniques to allow HBS to specify possible compliance and quality of result assurance.

4.3 Service task and communication interfaces

In our view two types of interface exist among HBS namely HBS communication interface and HBS task interface:

— HBS task interface specifies which types of tasks an HCE can perform, e.g., translating documents or evaluating products.

— HBS communication interface characterizes communication protocols of an HCE, such as communication means (e.g., via email or via MTurk), bandwidth (e.g., how many tasks per day), and latency (e.g., read task only 2 hours).

Figure 6 depicts an overview of service tasks and communication interfaces between an HCE and its consumers. While HBS task interface can be straightforwardly modeled using WSDL or REST (e.g., in the HPS framework), it is more complex to model and manage HBS communication interfaces. In uniform SBS-based systems, these interfaces are typically well-defined and associated with non-functional properties. However, they are not well-described and unified for HCEs and that hinders a seamless integration among HCE and HCE with MCEs. Table IV shows examples of existing works that support the modeling and management of communication interfaces.

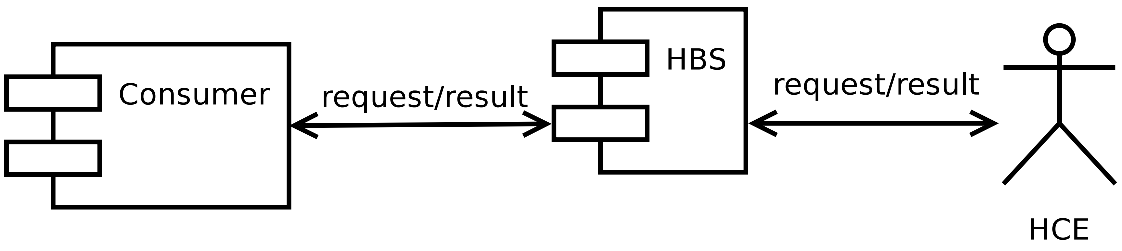

Figure. 6: General view of service tasks and communication interfaces

In our previous work in HPS, SOAP is used as a communication protocol between any consumer and HBS, and the communication from HBS to human is via specific middleware. However, in general, these communication links could be implemented via different protocols, depending specific contexts. From a service management perspective, we need to manage different communication interfaces to support the utilization of different HBS in global marketplaces.

Table IV: Examples of current communication interfaces

System	Communication Interface	Description
MTurk [mtu 2011]	Push and pull/mediator	MTurk plays an HBS for multiple HCEs. An HCE pulls requests from MTurk and pushes results to MTurk. The communication is based on Web.
HPS [Schall et al. 2008]	Push and pull/mediator	HBS is modeled as an HPS managed by a specific middleware. Specific means for HCE to obtain requests from and returns results to HPS via specific middleware. With this middleware, potentially communications can be implemented using SMS, Web, or Email

4.4 Managing service relationships

With respect to relationship management, both SBS and HBS have common types of relationships but they also have their own distinguished types of relationships (shown in Table V). Unlike SBS, whose relationships are typically based on software and hardware dependencies, HBS have certain social relationships, such as *knows* or *friends*. While several works have been investigated for relationships in software and SBS [Treiber et al. 2008a; 2008b; Papazoglou et al. 2011], currently there is a lack of techniques to understand and manage relationships between HBS. While MCEs can be linked via social networks, we foresee a difference between relationships in social networks and in service ecosystems. The latter type of relationships reflect how HBS can be composed and interacted in applications. For relationship management, we need to develop new novel frameworks that consider all types of relationships mentioned in Table V.

Table V: Common and different relationships

	Relationships	Description
Common	dependency and similarity	these relationships are based on system/software dependencies and functionality/non-functional properties similarity
SBS-to-SBS	versioning and variant	these relations are based on software service evolution [Treiber et al. 2008b; Papazoglou et al. 2011]
HBS-to-HBS	knows, friends, trusts	these relations are based on social concepts
HBS-to-SBS	knows, trusts	these relationships are based on the concept of service-enriched social network [Treiber et al. 2009]

4.5 Managing elastic, dynamic non-functional properties

4.5.1 *Classification of non-functional properties.* The first issue in managing dynamic non-functional properties (NFPs) is how to characterize NFPs for both SBS and HBS under the same classification to support the selection and composition of SBS and HBS in a single application. Certain NFPs, such as cost, availability, reliability, energy consumption, etc., are crucial information for utilizing HBS. For example, in some cases, it does not make sense to put a task, such as translation of a new sustainability governance law that is needed urgently, to the crowd without strict constraints because the result could be available only after a long time or the quality of the translation is too bad. Overall, in many cases, a developer of HBS-based programs needs to proactively decide to which HBS the tasks should be assigned. In particular, it is important when we need teamwork and when we are dealing with business critical problems. Although several efforts have been devoted for studying dynamic properties of humans and teams (see Table VI), to date, we lack unified metrics for characterizing HBS and their interactions that are comparable, searchable and programmable. The question here is that not only these metrics should be well-defined so that selection can be made, but they should also be associated with other HBS service information, such as capabilities and interfaces to support a programmable HBS system.

Table VI: Examples of metrics

Metrics	**Work**	**Description**
Trust	[Walter et al. 2009], [Skopik et al. 2009]	Monitor and determining trust of individual
human-to-service, human-to-human, and service-to-service interaction metrics	[Truong and Dustdar 2009b]	monitor and determining metrics based on human and service interactions
QoS	[Ran 2003]	describing QoS metrics for Web services
Licensing	[Gangadharan and D'Andrea 2011]	specify licensing for service
Quality of data	[Truong and Dustdar 2009a]	Determining and specifying quality of data for services

While several NFPs are common and considered in different works, putting them into a single classification will enable the comparison among HBS and between HBS and SBS. To enable the integration of HBS and SBS, we need to use the same definition for each NFP, no matter whether an NFP is associated with a SBS or an HBS. Unified metrics are important because this enables us to see human computation capabilities in a similar way with machine computation capabilities, simplifying the integration among human and machine to foster collective intelligence. Therefore, our approach is that our services will be associated with a unified classification of dynamic non-functional properties to characterize them. Figure 7 presents main metrics in our classification. Note that unlike most existing works which consider only a few metrics, like performance and pricing, we also consider other metrics, such as rights, law and jurisdiction, and context, as they are also critical in the way we utilize HBS and HBS's outputs. Furthermore, we also consider human-specific properties, such as Return on Opportunity (ROO), which tends to associate with HBS, to reflect opportunities that an HBS could bring to its customers. Similar to ROO, we need to determine HBS-specific metrics that cannot be found in SBS.

4.5.2 *Interpretation of non-functional properties.* Although similar NFPs for SBS and HBS can be put into the same classification, their interpretations for SBS and HBS may be different in many cases. When utilizing an NFP, a developer or a system must know whether the metric is associated with an SBS or HBS in order to make the decision. For example, in a global economic, the availability of a SBS and a HBS can be defined as the time it is available for use. However, it is expected that an SBS should have almost 100% availability, while the availability of HBS is much lower than 100%. Thus, just using the value of an NFP is not enough. Another important point is that not all NFPs will be associated with all SBS or HBS, similarly, the importance of an NFP is also different for SBS and HBS. For example, quality of data, including the quality of the result, maybe a very important for selecting HBS instead of SBS in certain cases. So to enable the search and discovery of SBS and HBS in a similar manner, we associate NFPs with the types of the service as well.

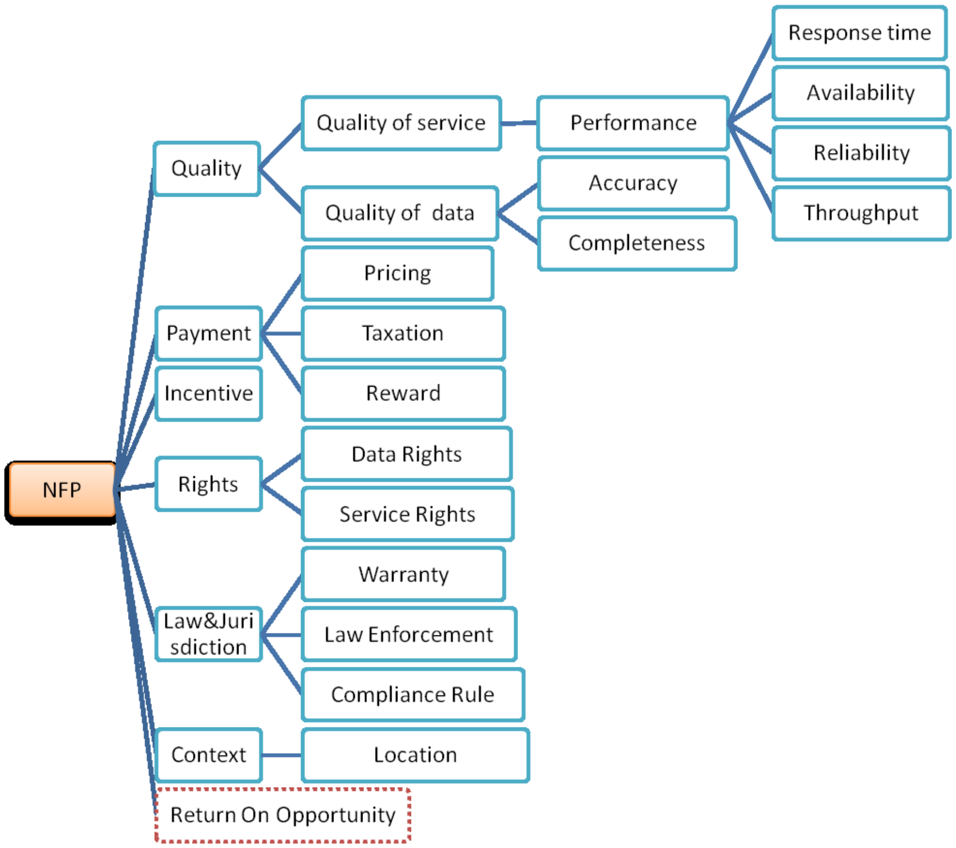

Figure. 7: Examples of non-functional properties for SBS and HBS

When we are able to access a large number of HBS, then monitoring and analysis of NFPs become extremely challenging. While we propose metrics, the monitoring and analysis techniques for such integrated systems do not exist today. This calls for further investigation in this direction.

5. CONCLUSIONS AND FUTURE WORK

Elastic computing is needed for scaling software services and teamwork efforts. Virtualizing human-based computing and machine-based computing elements under the same service model will require the development of novel middleware and management mechanisms. In this paper, we consider human-based computing elements (HCEs) and discuss fundamental characteristics of HCEs and how we could integrate HCEs and machine-based computing elements (MCEs) in order to collectively gather computation capabilities of both human and machine in a large-scale setting. We discuss techniques to virtualize HCEs based on the service model to exploit HCE capabilities under different forms of human-based services (HBS), such as HPS and SCU. By integrating HBS and SBS under the same SOC concepts, we could model a new computing system that includes a new class of computing elements with very dynamic and complex properties.

We argue that the service computing model can be used to harness human capability and to integrate their collective intelligence into a complex application. However, we show that several issues need to be solved in order to achieve this. In this paper we have identified several research topics. Currently, we are working on a novel programming model for HBS and SBS and techniques for monitoring and scheduling HBS. Furthermore, we are working on novel applications that utilize HBS in evaluating quality of simulations and situational information.

Acknowledgments

We would like to thank several our colleagues and students, Kamal Bhattacharya, Muhammad Zuhri Catur Candra, Hoa Khanh Dam, Aditya Ghose, Yike Guo, Lam-Son Le, Frank Leymann, Vitaliy Liptchinsky, Mirela Riveni, Benjamin Satzger, Ognjen Scekic, and Rostyslav Zaboloznyi for their fruitful discussions in the topics of elastic computing in hybrid systems.

REFERENCES

2009. *Web Services Human Task (WS-HumanTask) Specification Version 1.1.* http://docs.oasis-open.org/bpel4people/ws-humantask-1.1-spec-cd-06.pdf.

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

2011. Amazon mechanical turk. Last access: 27 Nov 2011.

2011. Seti@home. `http://setiathome.berkeley.edu/`. Last access: 27 Nov 2011.

Baird, H. S. and Popat, K. 2002. Human interactive proofs and document image analysis. In *Proceedings of the 5th International Workshop on Document Analysis Systems V*. DAS '02. Springer-Verlag, London, UK, 507–518.

Barowy, D. W., Berger, E. D., and McGregor, A. 2011. Automan: A platform for integrating human-based and digital computation. Technical Report UMass CS TR 2011-44, University of Massachusetts, Amherst. http://www.cs.umass.edu/ emery/pubs/AutoMan-UMass-CS-TR2011-44.pdf.

Brew, A., Greene, D., and Cunningham, P. 2010. 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*. IOS Press, Amsterdam, The Netherlands, The Netherlands, 145–150.

Brill, E. and Ngai, G. 1999. Man vs. machine: a case study in base noun phrase learning. In *Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics*. ACL '99. Association for Computational Linguistics, Stroudsburg, PA, USA, 65–72.

Chiu, D. 2010. Elasticity in the cloud. *Crossroads 16*, 3–4.

Cohen, A. 2010. Liveperson api reference: Expert profiles. http://community.liveperson.com/docs/DOC-1009. Last access: 27 Nov 2011.

Doan, A., Ramakrishnan, R., and Halevy, A. Y. 2011. Crowdsourcing systems on the world-wide web. *Commun. ACM 54,* 4, 86–96.

Dustdar, S. and Bhattacharya, K. 2011. The social compute unit. *IEEE Internet Computing 15,* 3, 64–69.

Dustdar, S., Guo, Y., Satzger, B., and Truong, H. L. 2011. Principles of elastic processes. *IEEE Internet Computing 15,* 5, 66–71.

Gaedke, M., Grossniklaus, M., and Díaz, O., Eds. 2009. *Web Engineering, 9th International Conference, ICWE 2009, San Sebastián, Spain, June 24-26, 2009, Proceedings.* Lecture Notes in Computer Science, vol. 5648. Springer.

Gangadharan, G. R. and D'Andrea, V. 2011. Service licensing: conceptualization, formalization, and expression. *Service Oriented Computing and Applications 5,* 1, 37–59.

Hafner, K. 2007. Silicon valley's high-tech hunt for colleague. New York Times. `http://www.nytimes.com/2007/02/03/technology/03search.html?ex=1328158800&en=e58764b50c8a4508&ei=5090&partner=rssuserland&emc=rss`.

Kulkarni, A. P., Can, M., and Hartmann, B. 2011. 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 '11. ACM, New York, NY, USA, 2053–2058.

LinkedIn. Profile api. `https://developer.linkedin.com/documents/profile-api`.

Liptchinsky, V., Khazankin, R., Truong, H.-L., and Dustdar, S. 2012. A novel approach to modeling context-aware and social collaboration processes. In *24th International Conference on Advanced Information Systems Engineering (CAiSE'12)*. Gdansk, Poland.

Marcus, A., Wu, E., Karger, D., Madden, S., and Miller, R. 2011. Human-powered sorts and joins. *Proc. VLDB Endow. 5*, 13–24.

Markoff, J. 2011. Armies of expensive lawyers, replaced by cheaper software. `http://www.nytimes.com/2011/03/05/science/05legal.html?_r=1&adxnnl=1&ref=science&src=me&adxnnlx=1322516490-gIU31+1mQi19qoeNE3s/Uw`.

Papazoglou, M. P., Andrikopoulos, V., and Benberno, S. 2011. Managing evolving services. *IEEE Software 28,* 3, 49–55.

Ran, S. 2003. A model for web services discovery with qos. *ACM SIGecom Exchanges 4,* 1.

Schall, D., Truong, H. L., and Dustdar, S. 2008. Unifying human and software services in web-scale collaborations. *IEEE Internet Computing 12,* 3, 62–68.

Skopik, F., Truong, H. L., and Dustdar, S. 2009. Trust and reputation mining in professional virtual communities. See Gaedke et al. [2009], 76–90.

Treiber, M., Truong, H. L., and Dustdar, S. 2008a. On analyzing evolutionary changes of web services. In *ICSOC Workshops*, G. Feuerlicht and W. Lamersdorf, Eds. Lecture Notes in Computer Science, vol. 5472. Springer, 284–297.

Treiber, M., Truong, H. L., and Dustdar, S. 2008b. Semf - service evolution management framework. In *EUROMICRO-SEAA*. IEEE, 329–336.

Treiber, M., Truong, H. L., and Dustdar, S. 2009. Soaf - design and implementation of a service-enriched social network. See Gaedke et al. [2009], 379–393.

Truong, H. L. and Dustdar, S. 2009a. On analyzing and specifying concerns for data as a service. In *APSCC*, M. Kirchberg, P. C. K. Hung, B. Carminati, C.-H. Chi, R. Kanagasabai, E. D. Valle, K.-C. Lan, and L.-J. Chen, Eds. IEEE, 87–94.

Truong, H. L. and Dustdar, S. 2009b. Online interaction analysis framework for ad-hoc collaborative processes in soa-based environments. *T. Petri Nets and Other Models of Concurrency 2*, 260–277.

Voigt, S., Kemper, T., Riedlinger, T., Kiefl, R., Scholte, K., and Mehl, H. 2007. Satellite image analysis for disaster and crisis-management support. *IEEE T. Geoscience and Remote Sensing 45,* 6-1, 1520–1528.

Walter, F. E., Battiston, S., and Schweitzer, F. 2009. Personalised and dynamic trust in social networks. In *Proceedings of the third ACM conference on Recommender systems*. RecSys '09. ACM, New York, NY, USA, 197–204.

Wu, M., Muresan, G., McLean, A., Tang, M.-C. M., Wilkinson, R., Li, Y., Lee, H.-J., and Belkin, N. J. 2004. Human versus machine in the topic distillation task. In *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*. SIGIR '04. ACM, New York, NY, USA, 385–392.

Schahram Dustdar is a full professor of computer science and head of the Distributed Systems Group, Institute of Information Systems, at the Vienna University of Technology. Dustdar is an ACM Distinguished Scientist (2009) and IBM Faculty Award recipient (2012). More info at http://www.infosys.tuwien.ac.at/staff/sd.

Hong-Linh Truong is a senior research scientist at the Distributed Systems Group, Institute of Information Systems, Vienna University of Technology. More info at http://www.infosys.tuwien.ac.at/staff/truong.