Chapter 18
Virtualizing Software and Human for Elastic Hybrid Services

Muhammad Z. C. Candra, Rostyslav Zabolotnyi, Hong-Linh Truong and Schahram Dustdar

Abstract Human capabilities have been incorporated into IT systems for solving complex problems since several years. Still, it is very challenging to program human capabilities due to the lack of techniques and tools. In this paper, we will discuss techniques and frameworks for conceptualizing and virtualizing human capabilities under programmable units and for provisioning them using cloud service models. We will discuss how elastic composite applications can be built by combining programmable units of software-based and human-based services in the Vienna Elastic Computing Model.

18.1 Introduction

Utilization of human computation capabilities allows us to solve complex computational problems. This approach has been practiced at least since the middle of 80s, when Richard Dawkins presented an interactive evolution application in which preferences of user were used to lead evolution process [1]. To improve the quality and throughput of such human-enriched systems, in later approaches [2] this concept was extended by joining efforts from many people. However, the term "Human computation" in the modern meaning is believed to be coined out in 2005 [3].

M. Z. C. Candra (✉) · R. Zabolotnyi · H.-L. Truong · S. Dustdar
Distributed Systems Group, Vienna University of Technology,
Argentinierstrasse 8/184-1, 1040 Vienna, Austria
e-mail: m.candra@dsg.tuwien.ac.at

R. Zabolotnyi
e-mail: rstzab@dsg.tuwien.ac.at

H.-L. Truong
e-mail: truong@dsg.tuwien.ac.at

S. Dustdar
e-mail: dustdar@dsg.tuwien.ac.at

A. Bouguettaya et al. (eds.), *Advanced Web Services*,
DOI: 10.1007/978-1-4614-7535-4_18,

Recently, with the broad availability of Internet and the emergence of Internet-based technologies, techniques for human-based computation have been investigated intensively and developed rapidly. At the time of writing, a large number of people who are interested in contributing to complex problem solving can be found almost effortlessly [4]. This leads to the ever growing existence of the so-called collective intelligence which allows massive online human-based problem solving, such as wiki websites [5] and reCAPTCHA [6, 7]. This online human-based problem solving approach is usually associated with the term "crowdsourcing" [8]. On the other hand, professionals are also employed, as part of e-science and business workflows, for solving human-related tasks. They are utilized together with software in several complex workflows [9], using different technologies, such as BPEL4People [10] and WS-HumanTask [11].

While both crowdsourcing and workflows enable us to utilize human computing capabilities, they do not view human capabilities as a programmable unit that can be acquired, utilized and released in an elastic manner. Unlike software-based compute units (e.g., virtual machines and software services) that can be scaled in/out easily with today's cloud computing technologies, human efforts cannot be easily programmed in the way that they can be added, removed and interacted dynamically in parallel with quality, cost, and benefit control. In most cases, either workers are statically assigned to tasks based on their roles [10] or workers bid for suitable tasks that they can work on [12]. When workers bid on suitable tasks, elasticity of human computation capabilities is hindered as there is an uncertainty of whether someone will select a task or not. If the task has demanding requirements (e.g., workers with more than 10 years of image recognition experience), appropriate worker may not be available even in a large crowd of people [13]. Services where workers bid on suitable tasks make integration between humans and software in some composite applications more complicated because sometimes it is preferable to actively select a worker or to identify that such type of worker is not available, rather than to wait for the worker's initiative. To allow seamless integration of human into computation systems, it should be possible to use humans as programmable compute units, which are similar to other types of compute units [14], that can be scaled in/out based on quality, cost, and other benefits constraints.

Our aim in this chapter is to examine current techniques in virtualizing and programming human efforts in crowdsourcing and people-centric business processes in order to develop a novel way to program human capabilities for solving complex problems. In our view, human capabilities can be abstracted into programmable units and then can be provisioned under the service model, which can be easily specified and invoked in programs. To this end, we discuss challenges in supporting programming human capabilities and virtualizing human capabilities under human-based services. We will also present our approach in designing, deploying and executing human-based services.

The rest of this paper is organized as follows. Section 18.2 gives an overview of human computation approaches. Section 18.3 describes challenges and concepts for virtualizing human capabilities under programmable units. Section 18.4 studies existing techniques for realizing human capabilities as programmable units for elastic composite applications. Section 18.5 describes our solutions developed in the Vienna Elastic Computing Model. We conclude the paper and outline our future work in Sect. 18.6.

18.2 Overview of Human Computation Approaches

18.2.1 Crowdsourcing Platforms and Techniques

Several efforts have been done for mapping and building taxonomies from existing public crowdsourcing market [3, 15, 16]. According to [15], existing crowdsourcing scenarios can be categorized into three types:

- The first type is *"contest crowdsourcing"* where a contest is performed to obtain the best available solution for a certain problem, such as in *99designs* [17] and *Threadless* [18].
- The second type is *"task marketplace crowdsourcing"* in which typically simple and unrelated tasks are posted by clients, while registered workers will choose and solve the tasks. *Amazon Mechanical Turk* [19] and *CloudCrowd* [20] are some examples of this type.
- Finally, the third type is *"bid crowdsourcing"* where complex problems submitted by clients and the best bid from professionals will be chosen to solve the problems. Platforms such as *InnoCentive* [12] and *TopCoder* [21] support this model.

Several works focus on the enterprise crowdsourcing. Some elaborated lists of research agendas for enterprise crowdsourcing are presented in [22] and [23]. The distinction between public and enterprise crowdsourcing is discussed in [24], especially what factors affect the sustainability of the project's community. A sample crowdsourcing scenario in software development domain is discussed in [16]. An enterprise crowdsourcing solution is also provided by *CrowdEngineering* [25]. Using a proprietary crowdsourcing tools and infrastructure, it provides out-of-the-box vertical applications in the domain of customer care, sales, and survey.

Another interesting crowdsourcing approach that is actively developing nowadays are the human-based computation games [26] that present computation challenges to humans in an entertaining way. This approach presents great answers to human-based computation problems as game participants are motivated and interested in the task solving process because of game's entertainment. Also they try to get the highest score, which commonly represents the best solution of the stated problem. Foldit [27], a set of online challenges GWAP [28], and Phylo [29] belong to this category.

18.2.2 People-Centric Business Processes

With the growing popularity of Service-Oriented Computing (SOC), building of distributed systems by the means of service composition becomes more and more popular. We have been seeing many efforts done to integrate humans into business processes built atop Web services. In workflow-based systems, the Workflow Management Systems (WfMSs) manage the assignments and executions of tasks, which can be either software-based or human-based tasks. In the case of human-based tasks, each instance of the task is placed in the work-list of all eligible workers. The assignment of the task can be enforced by the WfMS, or the workers may be allowed to voluntarily select the task from the work-list [30]. In particular, BPEL4People [10] can be used as an extension for Web Services Business Process Execution Language (WS-BPEL) [31] to enable human interaction in business process.

However, these human-based task modeling approaches have several limitations. For seamless integration of human-based services into a Service-Oriented Architecture, we need a way to define, discover, and invoke human-based services in similar manner as we define, discover, and invoke Web services. Therefore, human tasks execution is no longer limited to a single organizational boundary.

18.2.3 Humans as Programmable Units

Conceptually, in crowdsourcing and people-centric business processes, human efforts can be considered as program elements, e.g., objects and statements in programs executing some instructions. However, the current way of programming human-related tasks is very different from that for software-related tasks. Very often, we have different design phases and techniques for specifying human-related tasks, using different tools [10, 32].

Consider, for example, a Web-service-based people-centric business process. Typically software-related tasks are programmed using a Web service composition technique [33]. It allows service providers to define interfaces to their services which the composed business process connects to [33]. Even though human-related tasks are also programmed and composed using service interfaces, the current techniques do not allow humans as service providers to define their own services. Also, the lack of capabilities for human-based service publication and discovery hinders some advance techniques such as automatic and adaptive service composition. Furthermore, in the approaches described above, humans as compute units have to adapt to the system and actively search for the tasks to solve [19]. Little effort has been spent for techniques to program applications to actively consider possibilities of human capabilities to decide how to use human computation.

AutoMan [34] is an example of the computation platform that allows integration of humans and software. AutoMan allows to specify a set of tasks to the workers in the form of function calls in a platform-independent manner. Additionally, AutoMan provides ability to specify required quality, time and price. However, AutoMan has some limitations that can be critical for some applications or might be limiting for others. For example, it defines only a limited list of task types and constrains specification allows to specify only upper limit. Also it forces application developers to specify human tasks in common crowdsourcing models.

Another platform worth mentioning is Jabberwocky [35]. Jabberwocky declares that humans and software have the same rights and programming possibilities. Jabberwocky provides a high level domain-specific language for task declaring, which is translated to the map-reduce pattern [36], what may be limiting or redundant for some applications.

Both AutoMan and Jabberwocky focus on the customer side, e.g., defining tasks utilizing human capabilities via crowd platforms, but they do not concentrate on developing techniques at the service provider side, e.g., developing human-based service provisioning models. Recently, techniques from SOC and cloud computing have been investigated for abstracting and provisioning human capabilities. One of the first approaches is to allow human capabilities to be described and published via Web services [37]. Furthermore, teams of people could be also established and provisioned under the service model, called Social Compute Unit (SCU) [38]. Overall, in this approach human capabilities can be categorized into Individual Compute Unit (ICU) and Social Compute Unit (SCU) and realized by service technologies. They can therefore be considered as programmable compute units [14] and belong to the so-called Human-based Service (HBS) built atop human-based computing elements, an analogy to software-based services (SBS), which is built atop machine-based computing elements [39]. This enables, for example, the possibility to unify HBS and SBS with the introduction of the virtualization layer [39] allows to simplify software development with HBS and SBS integration into scalable cloud-based service-oriented computing systems [3].

18.3 Incorporating Humans into Program Paradigms

18.3.1 Challenges

Thanks to the spread of the Internet, it becomes much easier and faster to find appropriate humans to perform the requested task. However, due to complexity and dynamicity of human possibilities and relations, it is still a huge challenge to proactively utilize human computation capabilities. In contemporary crowdsourcing platforms, it is common to put the tasks in a form of open call [8], but this approach assumes that appropriate workers will find the task and solve it within time constraints, what might be a challenge for a new and not popular type of tasks. Even more, people participating in a specific project are often homogeneous and, despite the size, the required person for a rare and unusual task might be missing. This problem can be solved either by popularization of the project or by active searching of an expert for a specific task, which goes beyond existing crowdsourcing models and requires additional efforts from the project's developers or supporters.

An active expert search approach, similar to the SBS invocation behavior, is that the worker plays only a passive role by presenting her possibilities and capabilities and waiting for incoming tasks. Active service search techniques are widely used for SOA-based systems [10, 40], but for HBS selection they have some major drawbacks that will be discussed in the following.

- First of all, this approach usually assumes that characteristics of the provided service are either static or changing only occasionally. It contradicts with the fact that human abilities can be very dynamic and even change during the day.
- Also, even when human worker is rated with respect to quality of the results, usual active service selection ignores the fact that selected human workers may consult with other experts for challenging tasks or even use solution of others. Currently it is also complicated (if it is possible at all) for a selected worker to redirect the task to another expert or worker who might be more experienced or has better chances to solve the specified task.
- Furthermore, for conceptual business tasks, problem description can be very complicated and challenging. Worker may have difficulties understanding the task, require some additional clarification, or perform the task incorrectly.

Another issue in programming human capabilities is that the task might be given not to a single person, but to a closely-connected group or a team of people. Such a group or team can be modeled as SCU and it cannot be referenced in the same way for separate workers, as abilities and characteristics of such a group/team are completely different from that of the separate worker. Nowadays the target worker type is selected at the stage of task generation, but there might be situations when it is impossible to do so. Required worker type may depend on the content of the task, quality, or cost constraints, which are known only in runtime. In such cases we must be able to develop abstract compute units and select appropriate humans for tasks right before the task assignment.

Summarizing what have been said above, integration of human worker into SOA-based system faces challenges such as the following:

1. the dynamic nature of non-functional properties of HBS
2. the need to consult with others or to redirect tasks to expert in the field
3. the need to support clarifying the task or receiving additional information interactively at runtime
4. the need to support different task structure depending on whether tasks will be processed by a single person or teams

We will discuss these challenges and present our approach to handle these challenges. We will focus on the first and the last one. Additionally, we will provide appropriate infrastructure that will allow solving other challenges on the level of the communication protocol.

18.3.2 *Virtualizing Humans as Programmable Compute Units*

Nowadays, the SOC model has been flourishing and widely used to model the hardware and software functionalities of machine-based computing elements (MCEs). Through standardized service interfaces, these functionalities can be accessed and composed for solving particular problems. However, for complex computational problems, we need to include human-based computing elements (HCEs) into the ecosystem for solving particular steps of the complex problem. Therefore, it is of paramount importance to have conceptual frameworks and tools for integrating HCE into service-based systems. If the HCE will be accessible in the same way as MCE, it will allow selecting the actual processing unit dynamically, depending on the current preferences in processing duration, cost, or results quality. To allow this, actual workers should be hidden behind another abstract layer, which would allow unification of task assignment information provision about the processing unit.

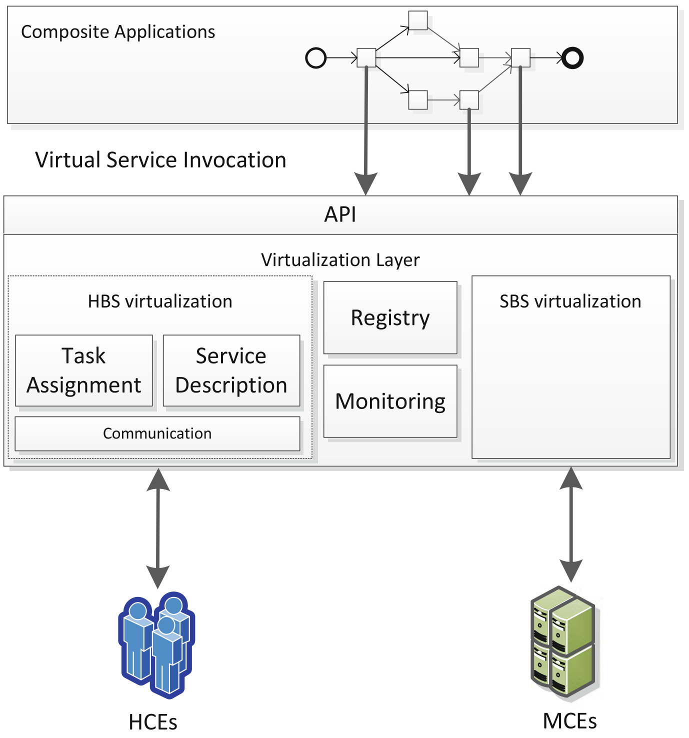

Fig. 18.1 Virtualizing and provisioning humans using SOC

One way to do this is to virtualize and unify HCE and MCE functionality to access them through a well-defined service interface just like it is traditionally done in SOC. Under the service model, everything is a service. Therefore, a distributed application may invoke available distributed services regardless of the underlying service type (MCE or HCE). Virtualizing HCEs under the same service model as MCEs also allows service providers (e.g., human workers) to offer their services through a standardized/common service description. This way, HBS discovery and negotiation become easier. This virtualization layer can also solve some of the aforementioned problems: it will provide unified interface that allows processing units to provide feedback to the system, calculate worker's qualities and preferences in run-time, or provide additional task context on request.

Furthermore, as with MCEs, application developers should be able to compose services involving HCEs. Through this virtualized services, application developers can compose mixed SBS and HBS either statistically during design-time or dynamically during run-time. Figure 18.1 depicts this concept of mixed service compositions using virtualized HBS and SBS. Since the virtualization and provisioning of SBS are known, we discuss possible approaches for virtualizing HBS:

- *Communication*: well-known techniques for communicating humans input/output have been developed. Such techniques will allow highly flexible and unrestricted types of communication between humans and HBS virtualization layer. All implementation details of communication will be hidden from applications, communication can be based on any technology, as long as it can be represented in the form of function invocation. This allows us to use well-known SOAP-based web-services along with RESTful services, FTP file transfer or e-mail/IM for task assignment to human worker. Note that human-related challenges mentioned above (e.g., task redirection, clarification request) can be solved in the protocol-specific way or even with ability to employ human consultant in exceptional situations. Of course, reliability and speed of such communication techniques are hardly comparable, therefore this also has to be taken into account during statistics calculation and SLA enforcement algorithms. Additionally, in some cases, the communication layer may require asynchronous service invocation, which also should be stated in service description and considered by the consumer.
- *Task Assignment*: as the main role of the HBS virtualization layer is to forward invocation requests and to provide responses, *Task Assignment* will handle all HBS service invocations. The main role of Task Assignment is to present virtualized HBS as a part of the system and allow seamless invocations and response retrieving. When Task Assignment receives a request, it converts this request into the representation that can be handled by the virtualized HBS. For example, it can prepare task in a human-understandable form (e.g., e-mail or IM message). When a response arrives, a timeout occurs or a call is canceled, Task Management converts available response into system model entities and returns back to the component that requested HBS. One important feature of Task Assignment is that it should allow composite applications to acquire, invoke and release HBS in an elastic manner, based on their specific constraints.
- *Service Description*: we need to develop models for describing HBS to allow HBS consumers to select appropriate HBS in runtime. *Service Description* provides existing services descriptions and functionality in a unified format. This component

allows getting all static service information, which includes also invocation cost, SLA agreement and allowed input data. All this information can be used to select the set of services that can handle requests.

- *Monitoring*: as discussed, human capabilities are very dynamic and cannot be described statically. For these needs, *Monitoring* is responsible for gathering and providing such dynamic information as average invocation duration, invocation jitter, communication problems and results quality/completeness. These properties can be used to validate SLA restrictions, rank available alternative services or balance request load, if one of the services is overloaded or has too long response time. Additionally, *Monitoring* manages list of assigned tasks and allows calculating current service load or billing information.
- *Registry*: we also need the *Registry* for storing, searching, filtering and providing the set of available HBS that can be searched based on the specified restrictions. The *Registry* would support *Service Description* models for HBS and SBS.

Finally, all features of the virtualization layer can be exposed via a set of APIs, designed in a similar fashion to APIs for contemporary cloud systems, to allow different applications to select and invoke HBS on-demand based on elasticity constraints.

18.4 State of the Art

In this section we discuss the state of the art of the technologies, which can be used to implement virtualization of SBS and HBS. To make discussion clear, we center the discussion around an example scenario to show how a composite application utilizes HBS and SBS. The scenario shown in Fig. 18.2 represents an application system used for mitigating and handling natural disasters. This application system mainly consists of 3 components: the data analysis workflow, the decision support system, and the disaster response workflow.

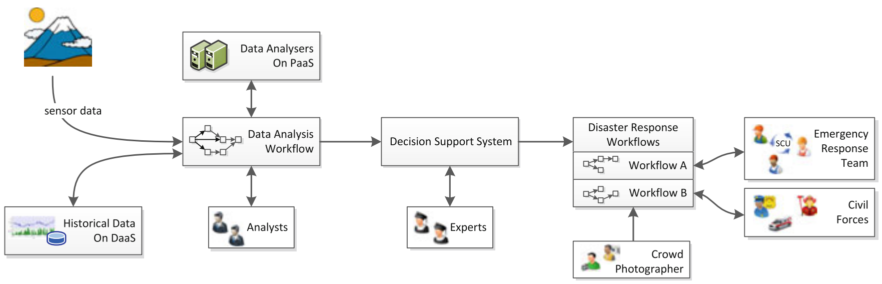

Fig. 18.2 Natural disaster management application

The data analysis workflow received data from sensors which capture nature activities such as earth vibration, rain and snow precipitation, wind speed, and so on.

Upon analyzing the data the workflow will generate signals to indicate whether certain activities may lead to a disaster and require further investigation. This workflow utilizes a Data-as-a-Service (DaaS) for storing and retrieving historical data through a defined SBS. A data analysis algorithm software running on a PaaS also provides services for analysis tasks. Furthermore, depending on the nature of the sensor data, the workflow may also invoke an HBS for manual data analysis provided by professional analysts.

Analysis results sent to the decision support system (DSS) will be used by the decision maker to decide whether a disaster warning should be declared. In situation where further consultation is required, the DSS may invoke an HBS to start expert's consultation service. When a disaster warning is declared, the disaster response workflow is initiated.

The disaster response workflow provides control over the disaster response and recovery activities. An SCU consisting of emergency response teams automatically assembled when necessary. The workflow may also invoke external workflows which control external team such as civil forces. Furthermore, the workflow may also initiate tasks to crowdsourcing platforms for obtaining pictures of the disaster location.

18.4.1 Composition Techniques

18.4.1.1 Syntax and Semantic for HBS

In SOA, applications are built by the means of composition of distributed services. Each application component is a service providing a particular set of functionalities. For example, on the aforementioned Natural Disaster Management application, the Data Analyzers component can be realized as external service which provides capability to analyze streams of sensor data for monitoring nature activities. Furthermore, we can also wrap the functionalities of human analysts and experts as services. Once we compose this various services properly, we obtain a composite application for Natural Disaster Management.

Service composition relies on the service description with respect to its functional and non-functional properties. Functional properties of a service describe its inputs, behavior, and outputs. These properties may be the data manipulation processing, the calculations, or other particular functionality which defines how the service is supposed to behave. On the other hand, non-functional properties (NFPs) describe the quality dimensions on which the user of the service could rely. The de-facto standards for describing the functional capabilities of a service is Web Service Description Language (WSDL) [41]. A WSDL description of a service provides a machine-readable definition so that users know how the service should be called. By evaluating a WSDL description of a service, users can decide whether the service matches with the functional requirements of the application. The quality descriptions of the services, also known as Quality of Services (QoS), are normally defined in Service Level Agreement (SLA) document. SLA provides formal definition of quality level in the form of a contract on which the user and provider of a service agree. Several standards for defining SLA are widely used. Some of the standards are *Web Services Agreement (WS-Agreement)* [42], *Web Service Level Agreement (WSLA)* [40], and *Web Services Policy (WS-Policy)* [43].

Syntax used in the aforementioned specification languages for defining functional and non-functional properties of services may be applicable for both SBS and HBS. A work was done to allow the usage of WSDL as HBS description language [44]. This allows us to describe the interface to services provided by human. For example, on our example scenario, the Data Analyzer service (an SBS) and Analyst service (an HBS) may offer similar service, i.e., analyzing sensor data. However, different interfaces can be defined for both type of services; the Analyst HBS may have a human collaborative platform such as Dropbox as interface.

While defining syntax for describing HBS may be straightforward, defining semantic of HBS description can be much more challenging compared to SBS description semantic. Human services functionality contains intangible aspects which are hard to define formally. HBS and SBS have different NFPs and the semantics of their similar NFPs can be different (Fig. 18.3 lists some examples of NFPs for HBS and SBS). For example, the SBS Data Analyzer service may be described to have 99 % availability. The interpretation of this value is widely understood. However, how would we define an HBS Analysis service that has 99 % availability? What does 100 % availability of human services entitle? This aspect HBS properties interpretation currently remains as an interesting research challenge in the service engineering area.

Metric Dimension	*MCEs Metrics*	*HCEs Metrics*
Resources	Number of resources, utilization, storage capacity, bandwidth capacity	Number of resources, utilization
Quality	Response time, throughput, availability	Response time, rating, availability, throughput, task acceptance rate
Cost and Benefit	Cost / API calls, virtual instance / hours	Task price, hourly price, reputation point

Fig. 18.3 Example of metrics for HBS and SBS

The SLA standards used above, for example WSLA, are designed to deal with virtually any types of QoS metrics. Therefore, theoretically it should be possible to use such standards to define SLA of HBS. However, there are two most important challenges that we should deal with: the *definition* and the *measurement* of the HBS metrics. For example, how can we model the expertise metric and how do we measure it. The quality of SBS, such as computing power, response time, and so on, can be defined and measured easier. But that is not the case for HBS. In most cases, the definition and measurement of HBS metrics is domain specific. Therefore, once we could address these two important challenges, at least for a particular domain we are interested in, we could use similar methodology for defining SLA mentioned above.

18.4.1.2 Design-Time and Run-Time Composition

Once we have a formal description of services, the composition of those services becomes possible. There are various service composition tools available [33]. In the business domain, some of the prominent examples are Business Process Execution Language for Web Services (BPEL) [31] and Business Process Modeling Notation (BPMN) [52]. Petri-Net is also a common tool used for composing services [53]. These composition tools are used during design-time by developers to compose workflow-based applications containing various invocations of services.

Many attempts have been undertaken to address run-time flexible composition issues in workflow systems and Process-Aware Information System (PAIS) in general. Organizations may need to refine their processes to adapt to changing environments due to new requirements, competitions, and laws. Papers, such as [54] and [55], propose methodologies to deal with flexibility issues in workflows, especially to manage running instances while evolving the workflow to a new schema. Those techniques discussed above traditionally deal only with SBS. There are some efforts to allow integration of human in service composition. BPEL4People [10] and WS-HumanTask [11] are some prominent examples. However, these approaches do not see human task in term of human as a *service* provider. Hence, it cannot utilize human capabilities when they are described as *services* such as discovering services just like we normally do in SBS.

The aforementioned service composition techniques deal with the functional requirement of the application. Other techniques are introduced to obtain a QoS-aware service composition. Consider we have a workflow as described in our Natural Disaster Management application. Each component, either human-based or software-based, is described as a service. Functional properties of those services are defined in a Web service description document, such as using WSDL, and the NFPs are defined in SLA specification, such as in WSLA. The service functionalities are orchestrated using BPMN tool. Furthermore, there are some service providers offering same service for each functionalities with different QoS. The SBS Data Analyzers service is provided by some SaaS providers. The HBS Analysis service is provided by a pool of human analysts, and so on. The next question is, how would we select which particular service providers to use in the application? This QoS-aware composition problem is an optimization problem; i.e., which service providers should be invoked so that we get an optimized (or satisfied) solution without violating the constraints.

Finding an optimized QoS-aware composition of services is known as NP-hard problem [56]. Some approaches based on integer programming [57], heuristics [58], and genetic algorithm [59, 60] have been proposed. These approaches can be applied during design-time, to help the developer choosing appropriate services for the application. They can also be used during run-time to allow late-binding of services. Optimizing service composition during run-time is more challenging. It requires an acceptable performance so that the optimization can be done in real-time. It should also consider interdependencies between services and how changes on one service may affect others or even stop the entire process instance. These approaches are currently available only for SBS. Addressing this composition service issues for HBS presents interesting open challenges for the service computing community.

Furthermore, some works have been done for more advance composition tools. Approaches to compose services in non-procedural ways are introduced in SELF-SERV [45] and SWORD [46]. Several tools such as CPM [47], Mentor [48], SELF-SERV [45], and OSIRIS [49] provide distributed workflow engine which allow web services to be composed and executed in distributed or peer-to-peer environment. To obtain an autonomic service composition, JOpera [50] provides an advance tool for composing services and a run-time environment with self-configuring, self-healing, and self-tuning capabilities. MarcoFlow [51] goes beyond the orchestration of human actors into a service composition by allowing distributed orchestration of user interfaces the users need to participate in the process. However, mostly, these tools focus on software-based services; and further works are required to integrate human-based services to the systems.

18.4.1.3 Services Matching and Discovery

On the famed *SOA triangle*, a service-based system not only consists of service providers and service clients, but also service discovery agents. Theoretically, the discovery agent functions as a bridge so that providers may publish their offered services and clients may find suitable services. Service discovery is done through a matching algorithm to find services with appropriate functional and non-functional properties.

The simplest service matching algorithm is keyword based searching. Other advance matching approaches were also proposed. Semantic, ontology, and similarity based matching have been employed to enhance the service matching [61–63]. Those matching algorithms focus on service functionality matching. To take NFPs into account, many works have been done for obtaining QoS-aware service discovery [64–66]. The aforementioned techniques for service discovery are designed for SBS. Service discovery for HBS is a new and challenging area for research. Human factors, such as skills, expertise, and reputations should be taken into account for effective discovery of HBSs. Several works, such as [67] and [68], have been done to address those issues. Trust network such as friend-of-a-friend (FOAF) network also provides important information about the HBS providers. In [69], a Broker Query and Discovery Language (BQDL) is proposed to discover suitable brokers who connect independent subgroups in professional virtual communities, such as normally found in social networks.

Although some standards for service registry exists, many providers prefer to use ad-hoc mechanisms for informing clients about their services. The situation is similar in the case of HBS. Currently there are no formal registries used for HBS discovery. We can consider task-based crowdsourcing marketplaces such as Amazon Mechanical Turk [19] as ad-hoc HBS registries. These crowdsourcing marketplaces have been flourishing dramatically in the recent years. However, the lack of formal service publications in these registries has been hindering automatic services matching and discovery for HBS.

18.4.2 Virtualization Techniques

18.4.2.1 Communication Interface to HCEs

The communication layer is responsible for delivering tasks and retrieving results from external service and handling other types of communication in a transparent way for the rest of the system. This part is already well-known for SBS, but for HBS it is only developing. For example, Amazon Mechanical Turk [19] provides a web-site with available jobs for a registered workers where they can select jobs they like from the set of available tasks (named HIT, Human Intelligence Task). But the set of operations available to the workers is limited: they are only allowed to select HITs and submit results, which often satisfies neither workers nor the creators of the task. To solve these problems, different companies present their own solutions that extend Amazon Mechanical Turk functionality and provide additional features required by participants. For example, Scalable Workforce [32] allows workers to subscribe on some subset of the HITs, extend worker's profile and allows workers to deliver feedback or clarification requests [70]. But the web-site is not the best way to communicate with the human workers. For example, Aardvark [67] tries to use existing human communication channels like Instant Messaging (IM), e-mail, SMS, Twitter, or others. Furthermore, this allows the worker to ask additional questions or to forward request to another person in case the worker cannot solve the task.

18.4.2.2 Task Assignment

Several systems provide SOAP or RESTful APIs for task assignment. For example, Amazon Mechanical Turk provides a SOAP or RESTful web-service, what makes it easy to integrate in the system that needs some work to be performed by the human. Furthermore, to simplify understanding and interaction with corresponding web-service, Amazon also provides set of API libraries for popular programming languages. Similar APIs are provided by other platforms, such as CrowdFlower [71] or CloudCrowd. However, Web service interface is not the only interface for creating and assigning a task. Some systems have provided few different interfaces to interact with different customers. For example, search engine and question answering service ChaCha [72] additionally provides ability to state tasks for a people through web-site, SMS, or phone applications. In most systems, it is the worker who selects tasks: if the required parameters are met, the worker is allowed to take any task, assuming that the worker takes only interesting and feasible tasks. However, this approach oversimplifies task assignment. It introduces situations when some tasks are not handled by anyone or handled with a huge delay. Instead, to guarantee fast and still correct response, some systems (e.g., Aardvark) tries to assign tasks themselves. With this way, systems have to know workers' profiles, current load, and availability.

Besides the capabilities of APIs, by relying on specific APIs of particular crowdsourcing platforms, such as Amazon Mechanical Turk or CrowdFlower, for utilizing human capabilities, we cannot easily program and scale in/out human capabilities from different platforms, as the API provided by different platforms is usually completely incompatible and often crowd workers do not know anything about the task source company. Therefore, the standardization and unification of the APIs for acquiring and invoking human capabilities is important, which would allow customers to select crowdsourcing platforms without carrying about future changes or even to use more than one platform to diverse risks and improve results speed and quality.

18.4.2.3 Service Description

Service description models, which allows collecting, generation and representation of available information about the underlying service, are not well studied for HBS. Amazon Mechanical Turk stores information about worker's qualifications and result acceptance rate. Scalable Workforce proposes to create full worker profile with photo, areas of expertise, interests and last activity and efficiency [70]. Such description system is usually good enough, but it hardly allows comparing different human-based services to detect who could do the specific task better. To allow this, service description models should analyze which similar tasks were already assigned to workers and how they managed to solve these tasks. Also it might be good to know the current load, non-functional properties of the workers, and current interests in this type of tasks, as these factors can influence results quality and service selection strategy.

18.4.2.4 Registry

Registry systems for SBS have been well developed. For example, Amazon AWS Marketplace[1] allows to find different virtual machines and software, while Microsoft Azure Marketplace[2] enables the search for data assets. For HBS, *Registry* is usually implemented by the database of registered users and their last activity information. In the systems where tasks are selected by workers, the role of the Registry is not large: usually it is just statistical information. Aardvark used to store and regularly verify a lot of additional information about users (e.g., last activities, last response time, and current task load). To allow fast service searching and query processing, access to such registry has to be optimized and important fields have to be indexed. Additionally, such systems require more information from users during the registration

[1] https://aws.amazon.com/marketplace/

[2] https://datamarket.azure.com/

and often might have difficulties in assigning tasks to the new human-based services, as information about them is not known yet and they are least preferable than older ones. This issue can be partially solved with the help of qualification tests or assigning previously solved tasks, but still this is an open challenge.

18.4.2.5 Monitoring

Monitoring service is responsible for gathering statistical information and verification of the task solution. As the tasks for human-based services are challenging, often it is hard to validate results' quality and speed. For example, Amazon Mechanical Turk leaves this to the requesting companies, which usually try to either estimate efforts or compare results to another worker. To introduce more intellectuality to this process, some companies invented algorithms that could be used to validate how fast and carefully workers were performing the task. For instance, CrowdControl proposed few interesting techniques that dramatically raise the quality of results [73]: it proposed more than 15000 rules to determine whether the solution is correct and worker performed job carefully and whether it is better to check solution again. Based on task validation results, CrowdControl changes the rating of the workers, what also influences on how much they will be paid now and how often validated in the future. In Yahoo! Answers tasks and solutions are usually unstructured, but readers rate the answers and select the best result. Another approach to solve the tasks with the appropriate quality of results is that tasks are usually split on the small slices that are sent to the few people to compare their results to each other [73]. But this approach also does not work well due to the fact that there are quite a few tasks that can be divided and results merged automatically. Correct results for several types of tasks, such as translation, pattern recognition or content generation, often are impossible without knowledge of the whole goal.

18.5 Programming Elastic Composite Applications in the Vienna Elastic Computing Model

The complexity of executing and managing elastic applications becomes even higher when we have to deal with clouds containing SBS and HBS. In this section, we outline steps in designing, deploying and executing composite applications consisting of HBS and SBS in our Vienna Elastic Computing Model (VieCOM), which offers techniques and frameworks to support multi-dimensional elastic processes of hybrid services represented under programmable units. Our approach addresses issues related during *design*, *deployment*, and *runtime* stage of composite applications. Figure 18.4 depicts the overall flow of our steps.

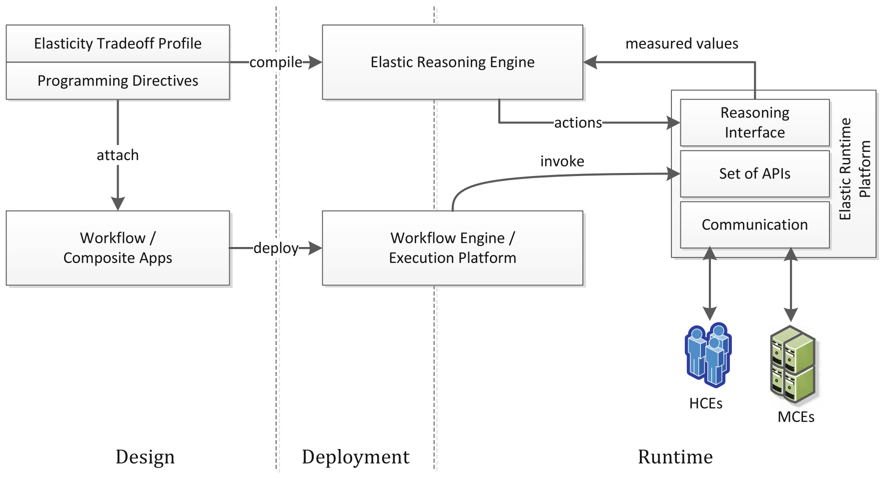

Fig. 18.4 Steps in programming and executing hybrid services in VieCOM

18.5.1 Multi-Dimensional Elastic Application

An elastic application should be able to address issues from two standpoints: it should consider resource provisioning constraints from its resource providers, and it must satisfy its own customers' demand at the same time. Therefore, it is important for an application designer to consider not only the resources but also the trade-off between cost and quality. Consider, for example, a Software as a Service (SaaS) which consists of many application components; each component with its own quality metrics such as performance, availability, throughput, and so on. These quality metrics may be dynamically specified by the customers and affect the SaaS provider's decision to scale-up or down resources. These changes will eventually affect cost needed for resource provisioning and cost charged to customers.

Traditionally, we have seen this elastic computing model being applied to cloud of SBS. However, the concept of elasticity can also be applied to hybrid cloud consisting of SBS and HBS. The principles of elastic processes [74] define various facets of elasticity that capture process dynamics. The elastic properties of applications are multi-dimensional. Figure 18.5 depicts our concept of multi-dimensional elasticity, classified into resource, quality, and cost and benefits. In these classes, several subclasses exist. During run-time, these elastic metrics are measured. The measured metrics can then be used to reason about adaptive actions needed to achieve a certain degree of required elasticity. A typical example for scaling Infrastructure-as-a-Service (IaaS) can be used to explain this reasoning process: when average utilization of running machines exceeds certain threshold, then start another machine.

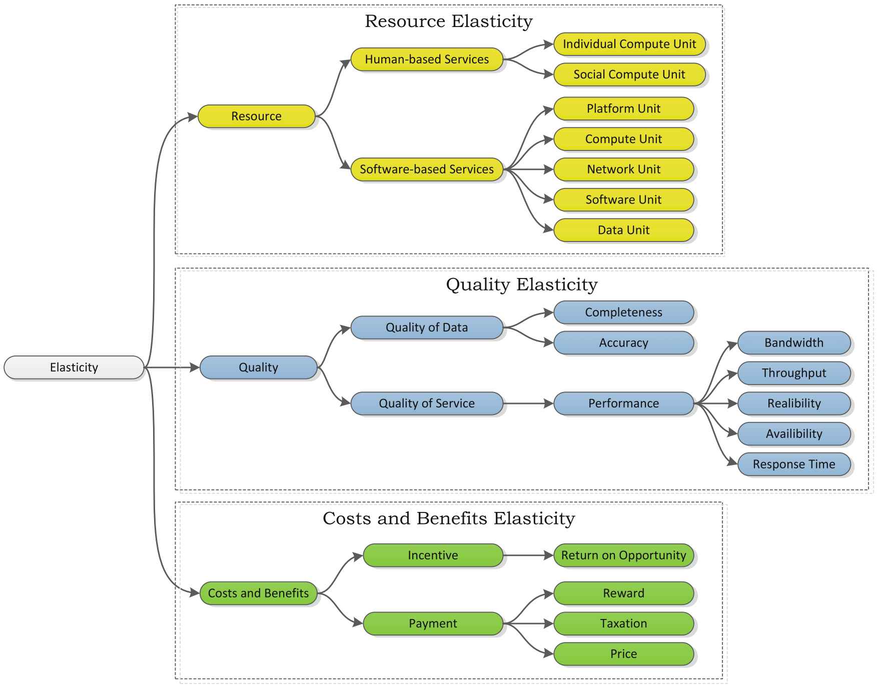

Fig. 18.5 Multi-dimensional elasticity

18.5.2 Modeling Process Elasticity

In our framework, elasticity is modeled by the notion of *Elasticity Profiles* which can be attached to workflows or distributed applications. An elasticity profile contain constructs to define elastic *objects*, *metrics*, and *rules*. The objective of modeling elastic processes is essentially to define the behavior of the process in response to the changing properties of the process' objects.

In an elastic process, we deal with objects and manipulation of the objects. These *elastic objects* are tasks (such as in workflows/processes), or software components (such as in distributed applications) that can be elastic by utilizing software-based or human-based cloud resources. Elastic objects of processes can be either individual tasks or process fragments. In order to make process' objects become elastic objects, two steps are needed: first, elastic properties must be associated with the objects during the modeling phase; and second, at runtime, the elastic reasoning engine decides elastic strategies for these objects based on their properties and runtime information.

Our framework uses a collection of rules describing the elastic aspects of the system. A process designer specifies these rules to model the dynamic changes of resources, quality, and cost of the system. Below are some examples of rules for expressing dynamic behavior of resources:

- When the average utilization of the human workers on the active pool is above 8 hours per day then add additional workers to the pool.
- A human-task requester wants to pay a cheaper price if the worker takes more than 1 hour to finish the task.

An elasticity profile will be deployed to our Elastic Reasoning Engine (ERE) and the application deployed to an execution engine. The elasticity profiles deployed to the ERE contain all definitions required to achieve the desired elasticity. The ERE is a *production rule system* which consists primarily of a set of rules about elastic behavior. The core element of this engine is a *forward-chaining inference engine* used to reason about adaptability actions required to achieve the desired elasticity.

The Elastic Runtime Platform (ERP) manages resources required for process executions. This underlying runtime layer provides the execution platform and resource management for elastic processes. This platform can be in the form of a cloud infrastructure, a scientific or business workflow engine, or it can also be a crowdsourcing platform as a human task execution environment. The monitor component of the ERP is responsible for capturing events of elastic objects and monitors their data. When a task is created, its corresponding elastic object is asserted to the ERE. Using the deployed set of rules, ERE decides which actions are necessary to obtain the desired behavior.

18.5.3 Executing Hybrid Services on the Cloud

Existing approaches exploiting human capabilities via crowds do not support well on-demand, proactive, team-based human computation. In VieCOM, we have proposed a novel method for invoking HBS in a similar manner as invoking SBS [75]. In our model, we present common APIs, similar to APIs for software services, to access individual and team-based compute units in clouds of human-based services. For example, Table 18.1 presents some APIs for provisioning HBS. Such APIs are provided at the cloud service level by HBS cloud providers. Therefore, they can be utilized by workflow engines and any application. The key idea is that based on elastic profiles, the ERE can utilize the APIs to find suitable HBS and depending on the elasticity constraints/rules, the ERE can invoke suitable HBS using these APIs. Furthermore, the ERE can use similar APIs for SBS, e.g., based on JCloud,[3] to invoke corresponding SBS.

[3] http://www.jclouds.org/

18.6 Conclusions and Future Work

In this paper, we discussed the challenges of programming human capabilities as programmable compute units. We have studied techniques for virtualizing human capabilities and how to incorporate humans into program paradigms. As we show in the paper, several techniques developed for crowdsourcing platforms and people workflows are not flexible enough to support the concept of program "humans" in complex, elastic applications. We have discussed our approach in virtualizing human capabilities as programmable compute units, realized and provisioned under the service model, to allow seamless integration between humans and software.

We have presented steps in designing, deploying and executing elastic composite applications in our Vienna Elastic Computing Model. We are currently prototyping an integrated development environment to support these steps, thus we will concentrate on integration aspects of HBS modeling, reasoning and execution by exploiting proposed APIs for clouds of HBS. Furthermore, our future work will focus on intelligent task assignment based on elasticity trade-offs in hybrid systems of software and humans.

Table 18.1 Main APIs for provisioning HBS [75]

APIs	Description
listSkills ();listSkillLevels()	list all pre-defined skills and skill levels of clouds
listICU();listSCU()	list all ICU and SCU instances that can be used.
negotiateHBS()	negotiate service contract with an HBS
startHBS()	start an HBS
suspendHBS ()	suspend the operation of an HBS
resumeHBS ()	resume the work of an HBS
stopHBS()	stop the operation of an HBS
reduceHBS()	reduce the capabilities of HBS
expandHBS()	expand the capabilities of HBS
runRequestOnHBS()	execute a request on an HBS
receiveResultFromHBS()	receive the result from an HBS
sendMessageToHBS()	send (support) messages to HBS
receiveMessageFromHBS()	receive messages from HBS

References

1. The blind watchmaker. Website http://en.wikipedia.org/wiki/The_Blind_Watchmaker.
2. Johnston, V., Caldwell, C.: Tracking a criminal suspect through face space with a genetic algorithm. Handbook of, Evolutionary Computation (1997) G8
3. Quinn, A., Bederson, B.: Human computation: a survey and taxonomy of a growing field. In: Proceedings of the 2011 annual conference on Human factors in computing systems, ACM (2011) 1403–1412
4. Howe, J.: The rise of crowdsourcing. Wired magazine **14**(6) (2006) 1–4

5. Leuf, B., Cunningham, W.: The wiki way: quick collaboration on the web. (2001)
6. recaptcha: Stop spam, read books. Website (2012) http://recaptcha.net/.
7. Von Ahn, L., Maurer, B., McMillen, C., Abraham, D., Blum, M.: recaptcha: Human-based character recognition via web security measures. Science **321**(5895) (2008) 1465–1468
8. Howe, J.: The rise of crowdsourcing. Website http://crowdsourcing.typepad.com/cs/2006/06/crowdsourcing_a.html.
9. Reiter, M., Breitenbücher, U., Dustdar, S., Karastoyanova, D., Leymann, F., Truong, H.L.: A novel framework for monitoring and analyzing quality of data in simulation workflows. In: eScience, IEEE Computer Society (2011) 105–112
10. Kloppmann, M., et al.: WS-BPEL extension for people-bpel4people. Joint white paper, IBM and SAP (2005)
11. Agrawal, A., et al.: Web Services Human Task (WS-HumanTask), version 1.0. (2007)
12. Home — innocentive. Website (2012) http://www.innocentive.com/.
13. Amatriain, X., Lathia, N., Pujol, J., Kwak, H., Oliver, N.: The wisdom of the few: a collaborative filtering approach based on expert opinions from the web. In: Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval, ACM (2009) 532–539
14. Tai, S., Leitner, P., Dustdar, S.: Design by units - abstractions for human and compute resources for elastic systems. IEEE Internet Computing (2012)
15. La Vecchia, G., Cisternino, A.: Collaborative workforce, business process crowdsourcing as an alternative of bpo. Current Trends in Web, Engineering (2010) 425–430
16. Vukovic, M.: Crowdsourcing for enterprises. In: Services-I, 2009 World Conference on, Ieee (2009) 686–692
17. Logo design, web design and more. design done differently — 99designs. Website (2012) http://www.99designs.com/.
18. Threadless graphic t-shirt designs: cool funny t-shirts weekly! tees designed by the community. Website (2012) http://www.threadless.com/.
19. Amazon mechanical turk. Website (2012) http://www.mturk.com/.
20. Work from home — cloudcrowd - we're working on it. lots of us. Website (2012) http://www.cloudcrowd.com/.
21. Topcoder, inc. — home of the world's largest development community. Website (2012) http://www.topcoder.com.
22. Brabham, D.: Crowdsourcing as a model for problem solving. Convergence: The International Journal of Research into New Media Technologies **14**(1) (2008) 75
23. Vukovic, M., Bartolini, C.: Towards a research agenda for enterprise crowdsourcing. Leveraging Applications of Formal Methods, Verification, and Validation (2010) 425–434
24. Stewart, O., Huerta, J., Sader, M.: Designing crowdsourcing community for the enterprise. In: Proceedings of the ACM SIGKDD Workshop on Human Computation, ACM (2009) 50–53
25. Crowdengineering - crowdsourcing customer service. Website (2012) http://www.crowdengineering.com/.
26. von Ahn, L.: Games with a purpose. Computer **39**(6) (june 2006) 92–94
27. Solve puzzles for science — foldit. Website (2012) http://fold.it/.
28. gwap.com - home. Website (2012) http://www.gwap.com.
29. Phylo. Website (2012) http://phylo.cs.mcgill.ca.
30. Salimifard, K., Wright, M.: Petri net-based modelling of workflow systems: An overview. European journal of operational research **134**(3) (2001) 664–676
31. Jordan, D., et al.: Web Services business Process Execution Language (WS-BPEL) 2.0. OASIS Standard **11** (2007)
32. Scalable workforce - mechanical turk software. Website (2012) http://www.scalableworkforce.com/.
33. Milanovic, N., Malek, M.: Current solutions for web service composition. Internet Computing, IEEE **8**(6) (2004) 51–59
34. Barowy, D., Berger, E., McGregor, A.: Automan: A platform for integrating human-based and digital computation. Technical report, Technical report, University of Massachusetts, Amherst (2012)

35. Ahmad, S., Battle, A., Malkani, Z., Kamvar, S.: The jabberwocky programming environment for structured social computing. In: Proceedings of the 24th annual ACM symposium on User interface software and technology, ACM (2011) 53–64
36. Dean, J., Ghemawat, S.: Mapreduce: Simplified data processing on large clusters. Communications of the ACM **51**(1) (2008) 107–113
37. Schall, D., Truong, H.L., Dustdar, S.: Unifying human and software services in web-scale collaborations. IEEE Internet Computing **12**(3) (2008) 62–68
38. Dustdar, S., Bhattacharya, K.: The social compute unit. Internet Computing, IEEE 15(3) (2011) 64–69
39. Dustdar, S., Truong, H.L.: Virtualizing software and humans for elastic processes in multiple clouds-a service management perspective. International Journal of Next-Generation Computing (IJNGC) (2012)
40. Keller, A., Ludwig, H.: The WSLA framework: Specifying and monitoring service level agreements for web services. Journal of Network and Systems Management **11**(1) (2003) 57–81
41. Christensen, E., Curbera, F., Meredith, G., Weerawarana, S., et al.: Web Services Description Language (wsdl) 1.1 (2001)
42. Andrieux, A., et al.: Web Services Agreement specification (WS-Agreement). In: Global Grid Forum. Number GFD. 107 (2004) 1–47
43. Vedamuthu, A.S., Orchard, D., Hirsch, F., Hondo, M., Yendluri, P., Boubez, T., Yalçınalp, U.: Web Services Policy framework 1.5. W3C Recommendation (September 2007)
44. Schall, D., Truong, H., Dustdar, S.: The human-provided services framework. In: 10th IEEE Conference on E-Commerce Technology, IEEE (2008) 149–156
45. Benatallah, B., Sheng, Q., Dumas, M.: The self-serv environment for web services composition. Internet Computing, IEEE 7(1) (2003) 40–48
46. Ponnekanti, S., Fox, A.: Sword: A developer toolkit for web service composition. In: Proc. of the Eleventh International World Wide Web Conference, Honolulu, HI. (2002)
47. Chen, Q., Hsu, M.: Inter-enterprise collaborative business process management. In: Data Engineering, 2001. Proceedings. 17th International Conference on, IEEE (2001) 253–260
48. Muth, P., Wodtke, D., Weissenfels, J., Dittrich, A., Weikum, G.: From centralized workflow specification to distributed workflow execution. Journal of Intelligent Information Systems **10**(2) (1998) 159–184
49. Schuler, C., Weber, R., Schuldt, H., Schek, H.: Peer-to-peer process execution with osiris. Service-Oriented Computing-ICSOC 2003 (2003) 483–498
50. Heinis, T., Pautasso, C., Alonso, G.: Design and evaluation of an autonomic workflow engine. In: Autonomic Computing, 2005. ICAC 2005. Proceedings. Second International Conference on, IEEE (2005) 27–38
51. Daniel, F., Soi, S., Tranquillini, S., Casati, F., Heng, C., Yan, L.: From people to services to ui: distributed orchestration of user interfaces. Business Process Management (2010) 310–326
52. White, S.: Introduction to BPMN. (2004)
53. Hamadi, R., Benatallah, B.: A petri net-based model for web service composition. In: Proceedings of the 14th Australasian database conference-Volume 17, Australian Computer Society, Inc. (2003) 191–200
54. Casati, F., Ceri, S., Pernici, B., Pozzi, G.: Workflow evolution. Data & Knowledge Engineering **24**(3) (1998) 211–238
55. Reichert, M., Rinderle-Ma, S., Dadam, P.: Flexibility in process-aware information systems. Transactions on Petri Nets and Other Models of Concurrency II (2009) 115–135
56. Canfora, G., Di Penta, M., Esposito, R., Villani, M.: An approach for qos-aware service composition based on genetic algorithms. In: Proceedings of the 2005 conference on Genetic and evolutionary computation, ACM (2005) 1069–1075
57. Zeng, L., Benatallah, B., Ngu, A., Dumas, M., Kalagnanam, J., Chang, H.: Qos-aware middleware for web services composition. Software Engineering, IEEE Transactions on **30**(5) (2004) 311–327
58. Berbner, R., Spahn, M., Repp, N., Heckmann, O., Steinmetz, R.: Heuristics for qos-aware web service composition. In: Web Services, 2006. ICWS'06. International Conference on, IEEE (2006) 72–82

59. Wada, H., Champrasert, P., Suzuki, J., Oba, K.: Multiobjective optimization of sla-aware service composition. In: Services-Part I, 2008. IEEE Congress on, Ieee (2008) 368–375
60. Canfora, G., Di Penta, M., Esposito, R., Villani, M.: A lightweight approach for qos-aware service composition. In: Proceedings of 2nd international conference on service oriented, computing (ICSOC04). (2004)
61. Benatallah, B., Hacid, M., Leger, A., Rey, C., Toumani, F.: On automating web services discovery. The VLDB Journal **14**(1) (2005) 84–96
62. Wu, J., Wu, Z., Li, Y., Deng, S.: Web service discovery based on ontology and similarity of words. Jisuanji Xuebao(Chin. J. Comput.) **28**(4) (2005) 595–602
63. Pathak, J., Koul, N., Caragea, D., Honavar, V.: A framework for semantic web services discovery. In: Proceedings of the 7th annual ACM international workshop on Web information and data management, ACM (2005) 45–50
64. Ran, S.: A model for web services discovery with qos. ACM Sigecom exchanges **4**(1) (2003) 1–10
65. Xu, Z., Martin, P., Powley, W., Zulkernine, F.: Reputation-enhanced qos-based web services discovery. In: Web Services, 2007. ICWS 2007. IEEE International Conference on, Ieee (2007) 249–256
66. Ali, R., Rana, O., Walker, D., Jha, S., Sohail, S.: G-qosm: Grid service discovery using qos properties. Computing and Informatics **21**(4) (2012) 363–382
67. Horowitz, D., Kamvar, S.: Searching the village: models and methods for social search. Communications of the ACM **55**(4) (2012) 111–118
68. Schall, D., Skopik, F., Dustdar, S.: Expert discovery and interactions in mixed service-oriented systems. Services Computing, IEEE Transactions on (99) (2011) 1–1
69. Schall, D., Skopik, F., Psaier, H., Dustdar, S.: Bridging socially-enhanced virtual communities. In: Proceedings of the 2011 ACM Symposium on Applied Computing, ACM (2011) 792–799
70. Turker communication. Website (2012) http://www.scalableworkforce.com/software-features-and-benefits/turker-communication/.
71. Crowdsourcing, labor on demand - crowdflower. Website (2012) http://crowdflower.com/.
72. Questions and answers chacha. Website (2012) http://www.chacha.com/.
73. Harris, D.: Exclusive: Crowdcontrol launches, brings ai to crowdsourcing. Website (2011) http://gigaom.com/cloud/exclusive-crowdcontrol-launches-brings-ai-to-crowdsourcing/.
74. Dustdar, S., Guo, Y., Satzger, B., Truong, H.: Principles of elastic processes. Internet Computing, IEEE **15**(5) (2011) 66–71
75. Truong, H., Dustdar, S., Bhattacharya, K.: Programming hybrid services in the cloud. In: 10th International Conference on Service-oriented Computing (ICSOC 2012), Shanghai, China (Nov 12–16 2012)