IEEE TRANSACTIONS ON
EMERGING TOPICS
IN COMPUTING

Received 1 April 2013; revised 19 June 2013; accepted 29 June 2013. Date of publication 15 July 2013;
date of current version 20 September 2013.

Digital Object Identifier 10.1109/TETC.2013.2273359

Vita: A Crowdsensing-Oriented Mobile Cyber-Physical System

XIPING HU[1], TERRY H. S. CHU[2], HENRY C. B. CHAN[2] (Member, IEEE), AND VICTOR C. M. LEUNG[1] (Fellow, IEEE)

[1]Department of Electrical and Computer Engineering, The University of British Columbia, Vancouver, BC V6T 1Z4, Canada
[2]Department of Computing, The Hong Kong Polytechnic University, Hong Kong

CORRESPONDING AUTHOR: X. HU (xipingh@ece.ubc.ca)

This work was supported in part by the Canadian Natural Sciences and Engineering Research Council through the DIVA Strategic Network, TELUS and other industry partners, and The Hong Kong Polytechnic University.

ABSTRACT As a prominent subcategory of cyber-physical systems, mobile cyber-physical systems could take advantage of widely used mobile devices, such as smartphones, as a convenient and economical platform that facilitates sophisticated and ubiquitous mobile sensing applications between humans and the surrounding physical world. This paper presents Vita, a novel mobile cyber-physical system for crowdsensing applications, which enables mobile users to perform mobile crowdsensing tasks in an efficient manner through mobile devices. Vita provides a flexible and universal architecture across mobile devices and cloud computing platforms by integrating the service-oriented architecture with resource optimization mechanism for crowdsensing, with extensive supports to application developers and end users. The customized platform of Vita enables intelligent deployments of tasks between humans in the physical world, and dynamic collaborations of services between mobile devices and cloud computing platform during run-time of mobile devices with service failure handling support. Our practical experiments show that Vita performs its tasks efficiently with a low computation and communication overhead on mobile devices, and eases the development of multiple mobile crowdsensing applications and services. In addition, we present a mobile crowdsensing application, *Smart City*, developed on Vita to demonstrate the functionalities and practical usage of Vita.

INDEX TERMS Mobile cyber-physical system, crowdsensing, system architecture, social network, cloud.

I. INTRODUCTION

In recent years, the capabilities of contemporary mobile devices such as smartphones have been improving a lot. These capabilities, such as significant computational resources (processing capability, local storage), multiple communication radios (second/third/fourth generation cellular, WiFi, Bluetooth, WiFi Direct, etc.), various sensing modules (cameras, accelerometer, gravity sensors, etc.), and high level programming languages (Java in Android, Object C in iOS), enable mobile devices to form mobile cyber-physical systems (CPS) in our daily lives [1].

Similar to traditional CPS [2], which are integrated computing and communication systems that process and react to sensing data from the external physical environment, and transform the way humans interact with the physical world, mobile CPS could be considered as a prominent subcategory of CPS with inherent mobility features [3]. Also, different from conventional embedded sensor systems, in which nodes are usually stationary due to the high energy-cost of movements [4], mobile CPS could be built on mobile devices that travel with their owners, and can take measurements in different phenomena (e.g., transportation status) and react in the physical word at multiple places throughout the day [5]. Thus, mobile CPS could provide a convenient and economical platform that facilitates sophisticated and ubiquitous mobile sensing applications between humans and the surrounding physical world.

Mobile crowdsensing refers to a broad range of social and community-based sensing paradigms employing mobile devices and wireless networks [6]. Different from conventional sensing solutions using specialized networks of sensors, mobile crowdsensing aims to leverage human intelligence to collect, process, and aggregate sensing data using individuals' mobile devices (e.g., using a camera to capture

a specific target), so as to realize a higher quality and more efficient sensing solution. Therefore, a connection exists naturally between mobile crowdsensing and mobile CPS, where mobile crowdsensing could be an emerging direction of mobile CPS, and mobile CPS are potentially promising implementation approaches for mobile crowdsensing.

Currently, most of the existing solutions of mobile CPS for crowdsensing are application specific and/or run on different custom-made software and hardware platforms with different capacities [7]–[9], and they usually follow different service interaction standards. For instance, consider two different applications of mobile CPS both developed for the same transportation scenario and run on different mobile devices, but one is designed to detect traffic accidents, and the other is designed to find out the most appropriate route. These two applications may not be able to share the same type of sensing data and related services directly. This may significantly constrain the sensing fusion and wide interaction between the digital world and physical world, as the horizon of individual people and sensing scope and capacity of various mobile devices are limited. Moreover, use of complex purpose-built mobile CPS applications may result in complicated operations that seriously constrain the user experience of a mobile CPS, and may not work well in some mobile devices with low capacities [3]. The work in [10], which utilizes Twitter to construct a crowdsensing system capable of supporting multiple crowdsensing applications is a step in the right direction. However, this system lacks the flexibility to meet the diverse and ubiquitous service requirements of different participants engaged in multiple crowdsensing tasks of mobile CPS.

Further, different from conventional mobile embedded systems, which are focused on the system's computations, mobile CPS usually are human-centric and need humans to participate and operate some specific control tasks (e.g., processing sensing data), so as to achieve seamlessly integrated sensing, computing, and control functions [11], [12]. However, as the application purposes and specialties of individuals are various and the capacities of their mobile devices are heterogeneous, while the current research work about mobile CPS is mostly focused on the optimization of computing tasks, such as avoiding over provisioning of the servers to mobile devices [3], an optimization mechanism that could simultaneously support efficiently and effectively allocating the computing tasks and human-based tasks among individuals in a mobile CPS is still lacking. Consequently, there is a need for a crowdsensing-oriented mobile CPS, which could support the efficient development, deployment and management of different customized mobile crowdsensing applications with standard and universal service interactions. Also, this system should support effectively allocating and optimizing the diverse computation tasks and human based tasks of mobile CPS simultaneously during run-time. To the best of our knowledge, such a system does not currently exist. The main objective of this work is to develop and validate an open and effective mobile CPS for ubiquitous crowdsensing applications.

In addition, as the availability of an individual's mobile device may be unreliable in mobile CPS, such as due to the crash of mobile operating systems, battery exhaustion, and intermittent networking disconnection, this may result in service failures and impede the pervasive use of crowdsensing applications [13]. However, most of the existing solutions [14]–[16] for service failure handling rely on specific protocols, and could not support different crowdsensing applications of mobile CPS widely. Therefore, in developing the crowdsensing-oriented mobile CPS, we also design and incorporate a novel mechanism that handles possible service failures, in addition to ensuring a high level of reliability of mobile crowdsensing applications when allocating and processing the computational tasks in the mobile CPS.

To address the needs identified above that are not efficiently met by existing solutions of mobile CPS, in this paper, we propose a novel mobile distributed system – a crowdsensing-oriented mobile CPS called Vita for mobile crowdsensing applications. Leveraging the popular social networking services by integrating our previously proposed MS2A [17], Vita adopts and orchestrates a series of open source techniques to provide a systematic approach to develop a mobile CPS for multiple and diverse mobile crowdsensing application scenarios. Our experiments show that Vita has affordable computation and networking overheads, and is efficient and effective for practical applications. Our major contributions are as follows:

- We propose Vita, a mobile CPS for mobile crowdsensing applications across mobile devices and cloud computing platforms, and present its design and implementation. Vita provides a seamless, comprehensive and universal solution that supports efficient development, deployment and management of multiple mobile crowdsensing applications/tasks for both application developers and end users.
- Vita is the first mobile CPS that simultaneously supports effectively allocating human resources and computing resources among individuals, with a novel service failure handling mechanism that does not depend on specific protocol on mobile platform.
- We develop novel applications based on Vita for deployment in Android devices to demonstrate the practical applications of Vita.

The rest of the paper is organized as follows. Section II gives some background on mobile CPS for crowdsensing applications, and discusses the requirements of developing mobile CPS. Section III presents the overall architecture and key components of Vita, and discusses how it meets the requirements of mobile CPS for crowdsensing. Section IV presents strategies for the implementation of Vita. Section V demonstrates a concrete application example developed on Vita to illustrate the functionality of Vita. Section VI shows practical experiments to evaluate Vita. Section VII reviews other mobile CPS for crowdsensing and compares them with Vita. Section VIII concludes this paper.

II. BACKGROUND

There are many examples of mobile CPS for crowdsensing applications. In this section we first briefly present the existing and potential mobile CPS based crowdsensing applications, and then based on these, we discuss various technical requirements of mobile CPS, and propose the techniques and methodologies we incorporate in Vita to address these requirements.

1) VEHICULAR SOCIAL NETWORKING

A large number of people in urban areas spend hours on their daily commute to and from work, traveling along the same routes at about the same time. Their travel patterns are highly predicable and regular. Consequently, with the help of mobile CPS, there is an opportunity to form recurring virtual mobile communication networks and communities between these travelers or their vehicles, i.e., vehicular social networks (VSNs). Through mobile crowdsensing, a VSN could aggregate travel information to measure the potential traffic congestion [18], find the most appropriate (e.g., lowest fuel consumption, shortest time) route in real-time [19], and detect traffic accidents and providing situational awareness services to first responders [20].

2) ENVIRONMENTAL MONITORING

The wide deployment of sensing technologies in our daily living environments and pervasive usage of mobile devices bring great opportunities for the deployment of mobile CPS to facilitate crowdsensing of human activities and interactions with the physical world. For example, through mobile crowdsensing, people can monitor their surrounding working environments and share the emerging and appropriate information with their colleagues, so that unsafe conditions can be discovered and made known to everyone that may be potentially affected in a timely manner. This can also potentially enhance the social relationship between colleagues and promote collaborations among them [21]. Also, by equipping widely deployed mobile devices with appropriate sensing modules and crowdsensing applications, mobile users could collectively measure the air quality, detect various pollutions (e.g., NO_X), and share the information with the communities in an efficient manner [22].

3) DISEASE REPORT AND CRISIS MANAGEMENT

As many mobile devices are equipped with an array of sensors, through the mobile CPS for crowdsensing, diverse sensing data and citizen reports from mobile devices can be triaged and acted on in real-time by individuals/communities, which will facilitate disease reporting and crisis management [23], and potentially bring economic benefits of up to $1.4 trillion [24]. For instance, the Ministry of Health in Cambodia uses GeoChat [25], a crowdsourced sensing interactive mapping application, for disease reporting and staff alerts which enables rapidly escalated responses to potential outbreaks. Also, Ushahidi [26] has been used to crowdsource and map crisis information from multiple sensing data streams in real-time through mobile devices, so as to coordinate field teams' activities and provide remote support from outside an earthquake zone.

The above examples show that mobile CPS based crowdsensing applications are of much practical use in daily lives. However, if we target to develop a mobile CPS to support these examples, there are several key technical requirements that need to be addressed. For example, in environmental monitoring, as the types of mobile sensing applications are quite heterogeneous, it needs a universal and standard architecture to support sensing fusion among multiple crowdsensing applications; in the applications scenarios of disease reporting and crisis (e.g., natural disasters) management, a mechanism that could not only efficiently allocate the computing tasks, but also the human-based tasks among individuals is crucial, as human factor usually play important roles in such time sensitive scenarios. In the VSNs, due to the highly dynamic network environment, a reliability mechanism is necessary to ensure the stability and correctness of crowdsensing applications.

As summarized in Table 1, in order to address these requirements, in developing the overall architecture of Vita, we adopt the design principles of REpresentational State Transfer (REST)-ful Web Services [27], which is already widely used for mobile applications and provides an ideal service-oriented architecture (SOA) to support the standard service interactions both for the development stage and at run-time. We adopt a number of open source techniques to develop a customized cloud computing platform, which could

TABLE 1. Requirements of mobile CPS.

Requirements	Descriptions	Solutions
Universal and standard architecture	To standardize and manage the service interactions across mobile devices and augment servers that provide scalable computing service to mobile devices	RESTful Web Services architecture design principle, mobile SOA framework [17], customized cloud platform implemented by using a number of open source techniques
Resource optimization mechanism	Supports efficient allocation of human based tasks among individuals and computing tasks between mobile devices and cloud platform	Leverage advantages of social networking services to design a novel application-oriented service collaboration model, and collaborate with the customized cloud platform
Reliability enhancement mechanism	To detect and recover possible unavailability of individual mobile devices when they are performing collaborative tasks with the cloud computing platform	Using mathematics modeling tool-Pertri Net and based on BPEL to design a high level service state synchronization mechanism in mobile platform

provide augmented services and collaborate with the mobile SOA framework [17] we developed before to support efficient development, deployment and management of different mobile crowdsensing applications. Also, we incorporate the social networking service MS2A implemented in our former work [17], and design a novel application-oriented service collaboration model (ASCM), which can quantify and analyze information on social contexts and sensing data (i.e., the social relationships between human and/or the physical elements surround them), so as to optimize the resource allocation in mobile CPS. In addition, based on Business Process Execution Language (BPEL) [28], we adopt mathematic modeling tool - Petri Net [29] to design a high level service state synchronization mechanism (S3M), which could be independent of specific protocols, to address the possible unavailable situations of mobile devices in mobile CPS.

III. SYSTEM DESIGN

As shown in Figure 1, the overall architecture of the Vita system consists mainly of two parts: the mobile platform and cloud platform. The mobile platform provides the initial environment and ubiquitous services to enable users to participate in and operate crowdsensing tasks through their mobile devices. The cloud platform provides a central coordinating platform to store and integrate the diverse data and tasks from crowdsensing and social networking service providers, as well as the development environment to support the development of mobile crowdsensing applications. In this section, we present the key components of Vita, and discuss how they meet the requirements of mobile crowdsensing as stated above.

A. MOBILE SOA FRAMEWORK

The mobile SOA framework (along with the mobile SOA server shown in Figure 1) has been proposed and implemented in our former work [17]. It is an extensible and configurable framework that is based on the specifications and methodologies of RESTful Web Services. It integrates popular social networking services (i.e., Facebook and Google+) and adopts an SOA to support the development of multiple mobile web service-based applications and services in an efficient and flexible way, with standard service interaction specifications that enable dynamic service composition during mobile devices' run-time. Further, in Vita, this mobile SOA framework also works as a bridge between the mobile platform and cloud platform of Vita over the Internet.

B. VITA CLOUD PLATFORM

For the design of the cloud platform of Vita, we adopt the RESTful Web Service-based architecture design methodologies and specifications, so as to provide an open, extensible and seamless architecture across the mobile and cloud platform of Vita. The cloud platform of Vita mainly consists of four components: management interface, storage service, deployment environment, and process runtime environment.

1) MANAGEMENT INTERFACE

It provides the development environment and application programming interfaces (APIs) to support application developers and enable third party service providers to participate in the development of different applications and services for mobile crowdsensing. Also, it makes use of open APIs provided by commercial social network websites like Facebook

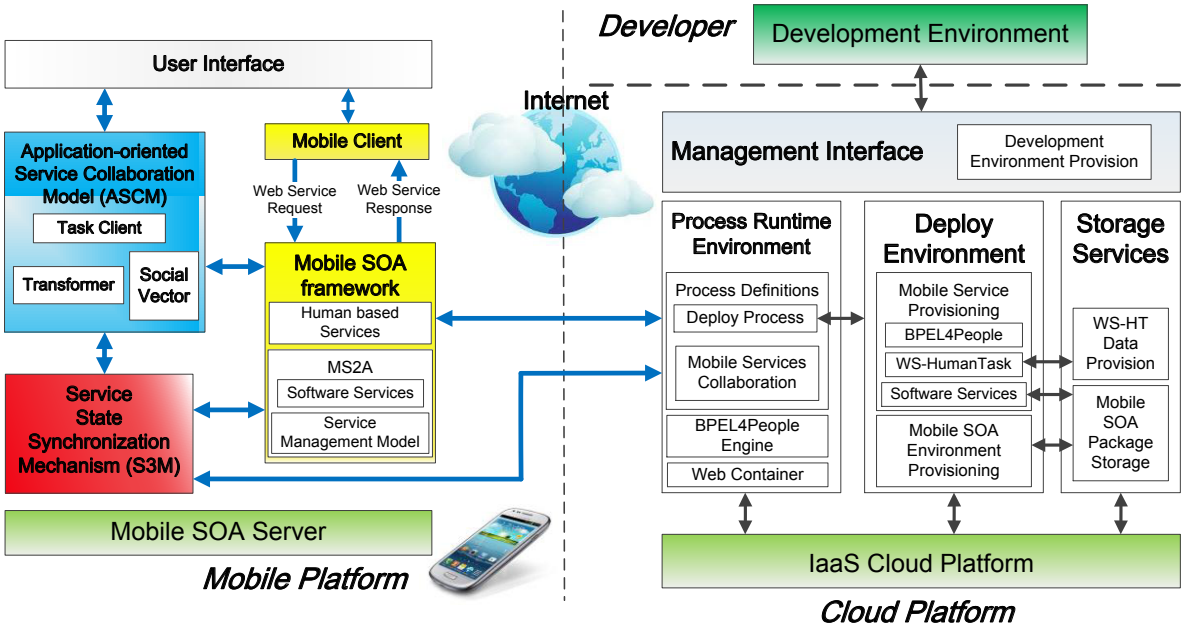

FIGURE 1. Overall architecture of the Vita system.

to access social networking services and disseminate crowd based social information to some popular social networks.

2) STORAGE SERVICE

It supports automatic backup of Vita system data, such as data related to software services, installation files of Vita in the mobile devices, task lists and results of mobile crowdsensing, and sensing data uploaded by the mobile devices through Vita.

3) DEPLOYMENT ENVIRONMENT

It enables dynamic deployments of the mobile platform and various web services of Vita to different mobile devices according to their capacities and practical application requirements, so as to support their users' participation in different mobile crowdsensing applications. Also, it could automatically deploy new applications and services uploaded by developers to the process runtime environment for executing diverse mobile crowdsensing applications.

4) PROCESS RUNTIME ENVIRONMENT

It is based on open source techniques (more details will be provided in Section 4), and provides the crowdsensing platform, which could coordinate, process, and combine multiple crowdsensing results from different mobile devices in real-time.

During a mobile device's run-time, once Vita receives a web service request from the *Mobile Client*, it can automatically analyze the requested Uniform Resource Locator (URL) and the related parameters encapsulated by Hypertext Transfer Protocol (HTTP), so as to determine the specific Java class to invoke the corresponding web services based on the configuration files. After the operation of the related web services, Vita will return the results to the *Mobile Client* in the form of REST-style data through HTTP. Thus, compared to traditional SOA-based solutions for developing and deploying mobile applications, one advantage of this architecture design is that the application developers do not need to be concerned with mapping relations about specific service requests to corresponding service resources, but can focus on the development of the application itself. For example, the

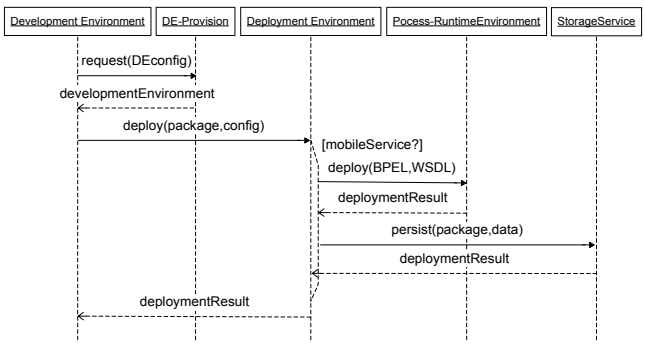

FIGURE 2. Process flow of application development in Vita.

work flow of application development in Vita consists of five steps, as shown in Figure 2 as elaborated as follows.

(i) Once the Deployment Environment Provision model on the cloud platform receives the development environment request with configuration, it can automatically deploy all of the components of the software needed to construct the development environment according to the assessment of the developer's software and hardware configurations, so as to set up the development environment for the application developer who intends to develop new services based on the Vita system.

(ii) Based on the service-oriented programming model of the mobile SOA framework, the developers can easily and efficiently implement diverse mobile crowdsensing applications. After a developer has finished the development, s/he can submit the deployment request with the related software package and configuration file. Further, the Deployment Environment analyzes whether the software package can be deployed to the cloud platform and/or to a mobile device.

(iii) If the software package needs to be deployed on the cloud platform, the Deployment Environment will deploy it to the Process Runtime Environment according to the specification of BPEL and the service description of the Web Service Definition Language (WSDL). After the Process-Runtime Environment receives and finishes the related deployment request, it will return the deployment result to the Deployment Environment.

(iv) If the software package needs to be deployed on mobile devices later, the Deployment Environment will deploy the related software components and data by submitting the request message consisting of (package, data) to the Storage Service module. Then the Storage Service module will store them in the cloud infrastructure and return the related deployment result.

(v) Finally, the Deployment Environment will return the deployment result to Development Environment, and present the results to the developer.

C. APPLICATION-ORIENTED SERVICE COLLABORATION MODEL

In this section, we introduce a novel resource optimization mechanism called application-oriented service collaboration model (ASCM) for allocating human-based tasks among individuals, and computing tasks between mobile devices and cloud computing platform efficiently and effectively.

As shown in Figure 1, the ASCM mainly consists of three components: Task Client, Transformer, and social vector. The *Transformer* can transfer the diverse application requests and tasks (which are *task instance data* based on the specifications of WS-HumanTask [30]) from users to a standard data format as a web service [31], [32] during a mobile device's runtime; implementation details about it will be introduced in Section IV-B. The *Task Client* works as a bridge between the ASCM and the mobile SOA framework; it could receive the application requests of users and invoke the existing services in the mobile SOA framework to accomplish the tasks. Also, the *Task Client* can invoke the social networking

services incorporated in the mobile SOA framework to obtain social information. The social vector can obtain application tasks information from the *Transformer*, and service and social information from the mobile SOA framework through the *Task Client*. Based on these, the social vector quantifies the distance and relationship between two physical elements (i.e., people, mobile devices, server of the cloud platform) and virtual elements (i.e., software service, computing/human based task) to facilitate the deployment of computing tasks and human based tasks among different devices and users of mobile CPS according to different specific application scenarios of crowdsensing.

1) SOCIAL VECTOR

Different from the former work [3], [33]–[35] about mobile CPS which mainly focus on the optimization of the system itself, in the design of the ASCM, based on the human-centric principle, we adopt a top-down design methodology. We suppose that in a mobile CPS, since every element has some inherent relations with a user, such as a task is finished by some people using their mobile devices, and a mobile device (contains the services available on it) belongs to a user, thus the allocation of the tasks (both computing and human-based) could be based on the relation between them and the users. The social vector is a tool in ASCM which is applied to optimize the allocation of a task to a group of physical elements (e.g., people, cloud servers). It is comprised of various attributes about the information of a user and the computing resource available to her (e.g., her mobile device, and cloud server connected to her mobile device). Social attributes can be continuous, discrete or Boolean. Continuous social attributes include computation power (e.g., the number of floating-point operations that the device can perform per second), communication capacity (e.g., bandwidth available for communicating with other devices), and remaining battery time, which are obtained via a general API in the device. Discrete social attributes include the number of similar tasks executed, the number of remaining tasks, and the number of reused resources for a particular task, which are updated and recorded in each mobile device individually whenever a task is completed. Boolean social attributes include whether a mobile user has related knowledge or prefers to do a particular task, which are input by users after the deployment of a Vita crowdsensing application on mobile devices. In essence, an m-dimensional social vector $\overrightarrow{V_{T,U}} = [a_{T,U_1} a_{T,U_2} \ldots, a_{T,U_m}]$ which has m attributes could be used to quantify the relationships between a humanbased or computing task T and a mobile user U, which assists Vita in assigning multiple and heterogeneous tasks to mobile users in an effective and efficient manner.

Here we use an example to illustrate how a social vector works in ASCM of Vita to optimize the allocation of human based tasks. Similar methodologies could also be applied for the optimization of computing tasks. Suppose that there are M mobile users, which are $U_1, U_2, \ldots, U_M$, and N multiple and heterogeneous tasks, which are $T_1, T_2, \ldots, T_N$. A straightforward but ineffective way is to assign the tasks to the mobile users randomly. A better approach is to assign each task to a group of mobile users with the most appropriate resources and knowledge to carry out the tasks (i.e., based on their attributes as defined in the social vectors). Assume that a mobile user S has the most appropriate resources and knowledge to execute the task T_i. We can define an ideal social vector

$$\overrightarrow{V_{T_i,S}} = [\overrightarrow{V_{T_i,S}}(1), \overrightarrow{V_{T_i,S}}(2), \ldots, \overrightarrow{V_{T_i,s}}(m)]$$

Besides, assume that there is a group G formed by a group of user U_j, where $j = 1, 2, \ldots, L_i$ and $\sum_{i=0}^{N} L_i = M$. The group social vector between the group G and the task T_i is defined by

$$\overrightarrow{V_{T_i,G}} = \frac{1}{L_i}\left[\sum_{j=1}^{L_i} \overrightarrow{V_{T_i,U_j}}(1), \sum_{j=1}^{L_i} \overrightarrow{V_{T_i,U_j}}(2), \ldots, \sum_{j=1}^{L_i} \overrightarrow{V_{T_i,U_j}}(m)\right]$$

where $\sum_{j=1}^{L_i} \overrightarrow{V_{T_i,U_j}}(x) = a_x$. Note that for each group, the group social vector gives the centroid of the group. A user of Vita seeks to find a group of L_i mobile users $U_1, U_2, \ldots, U_{L_i}$ such that the $\overrightarrow{V_{T_i,G}}$ is the closest to the ideal social vector $V_{T_i,S}$. Referring to Figure 3, for purposes of illustration, we define a simple social vector with two attributes a_1 and a_2 . Given ten social vectors (i.e., ten mobile users), we select a group with the group social vector $\overrightarrow{V_{T_A,G}}$, which is closest to the ideal social vector $\overrightarrow{V_{T_A,S}}$. Based on these, we adopt two intelligent computing techniques: Genetic Algorithm (GA) [36] and K-means [37] clustering as optimization methods in social vector. Note that GA provides more optimized solutions but requires more processing, while K-means clustering provides less optimized solutions using less processing.

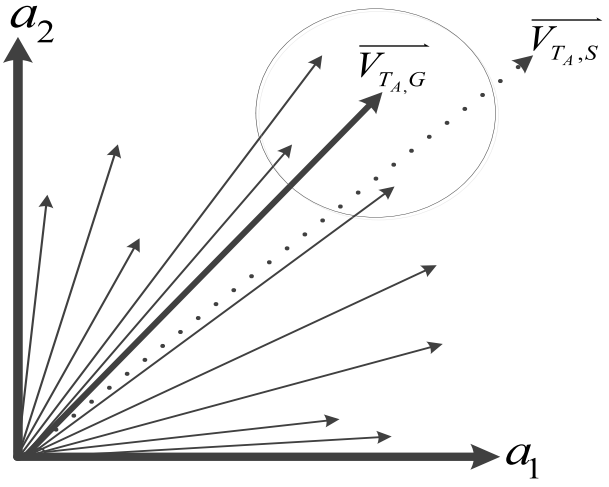

FIGURE 3. Social vectors for task T_A.

For purpose of illustration, here we present a GA-based clustering algorithm for assigning people to different

tasks. Inspired by the evolution of living organisms, GAs are intelligent computing techniques for finding optimized solutions. Basically, practical solutions are expressed as ''chromosomes'', which can be mixed to generate new chromosomes through a crossover process. Sometimes, a mutation process can also be employed to introduce small changes in chromosomes after a crossover operation. After many generations of crossover and mutation operations, a close-to-optimal solution is obtained. As an example, assume there are ten people. Each of them has an identity number $\{0, 1, \ldots, 9\}$. Initially, random solutions are generated to assign the people into groups. Groups can be combined to form a chromosome. Chromosomes are selected based on their fitness values. The fitness value is defined as the distance between the group social vector and the ideal social vector. Then, α chromosomes are selected through a random process based on their fitness value. For assigning people to different groups effectively, chromosomes with a higher fitness have a higher probability of being selected (higher fitness value implies smaller distance between the group social vector and the ideal social vector). Note that a chromosome may be selected more than once. After selecting $\alpha/2$ pairs of chromosomes, each of them is mixed in the crossover process. Assume that two parent chromosomes, A and B have been selected. After that, $\beta\%$ of A is mixed with (1- $\beta\%$) of B to produce C and (1- $\beta\%$) of B is mixed with $\beta\%$ of B to produce D. Some identity numbers may need to be added or removed to ensure the identity numbers of all people are in the chromosomes and without duplication. The best two chromosomes among A, B, C and D survive. Finally, any identity number may be replaced by another with a pre-defined mutation probability. The above steps are repeated 1000 times (i.e., 1000 generations).

Besides, considering the computationally extensive nature of GA-based algorithm, we also adopt a K-means-clustering-based algorithm in social vector. It randomly assigns people into groups so that each group has roughly the same number of people. For each person, it can calculate the distance between his/her social vector and the centroid of the social vector of his/her group. For maintaining the desired number of people in each group, a person who is closest in his/her group is assigned to the second closest group if the number of people in the original group exceeds the predefined limit. The above process is repeated until no people need to be re-assigned from one group to another. In other words, the process ends when all groups become stable. Since the choice of initial people in group can greatly affect the final result (i.e., the smallest sum of distance between group social vectors and ideal social vectors), the best result of multiple trials of different initial people in groups will be adopted.

2) WORKING THEORY OF ASCM IN VITA FOR MOBILE CROWDSENSING APPLICATIONS

Here we use two examples to illustrate the overall working theory of ASCM with other components of Vita for mobile crowdsensing applications. Assume that there are two roles: **A** – common mobile users; **B** – mobile users who participate in crowdsensing and provide human-based service. The same user may have multiple roles, such as **A** and **B** simultaneously, but since the definitions of different roles are distinct and different processes are provided in order to support these roles, thus we present them separately.

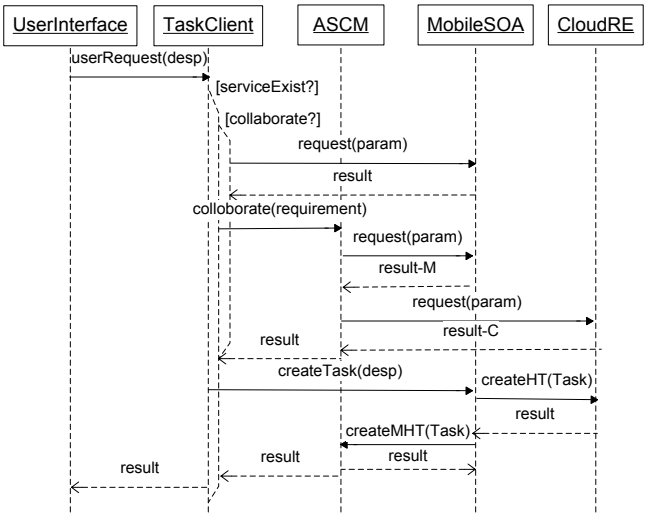

FIGURE 4. Process flow of common users of Vita.

a) Common mobile users: As shown in Figure 4, a common mobile user of the Vita system can choose functions through the user interface of his mobile device, according to his application purpose. After the *Task Client* in ASCM of Vita receives the related application request and description, it first assesses whether this application request can be satisfied through the existing services and/or a collaboration of them on the local mobile device and the cloud platform of Vita. If the assessment is positive, then the *Task Client* will assess whether this request needs service collaboration, and if it finds that the existing services in the local mobile device can directly meet this request, it will then send the request to the Mobile SOA framework and get the related result. If the existing services cannot satisfy the request, the *Task Client* will send the collaboration request to the social vector in ASCM. Then the social vector analyzes and decides the method of service collaboration based on the attributes of this request (note that all required information of using social vectors can be obtained via a general API in mobile devices, recorded in each mobile device individually whenever a task is completed or inputted by users after deploying a crowdsensing application of Vita on mobile devices), and then sends the service invocation request to the cloud platform of Vita through the Mobile SOA framework, so as to get the service collaboration results from the other mobile devices *(result-M)* via the cloud platform and/or the cloud platform itself *(result-C)*. Furthermore, the Mobile SOA framework will combine both of the results (*result-M* and *result-C*) and return the result to the *Task Client*.

If the *Task Client* finds that the original application request could not be dealt with at the beginning, it will encapsulate

the request as a new human based service by the *Transformer* of ASCM, create the related task request, and try to get the result through real-time crowdsensing, more details about this process will be introduced in **B** below. Finally, the *Task Client* will combine all of the results from the various process branches and return the final results to the user through the user interface of her mobile device.

b) Mobile users participating in crowdsensing and providing human-based services: In Role A, it is possible that during the mobile users' runtime, the current available services and collaboration results both in their mobile devices and on the cloud platform are unable to meet the purposes of their applications. In this case, Vita can automatically transfer the related service requirements of the users to standard human-based web services, and then deploy the services to the cloud platform through the mobile SOA framework according to the service requirements. After that, other mobile users can participate in crowdsensing and help to finish such a task.

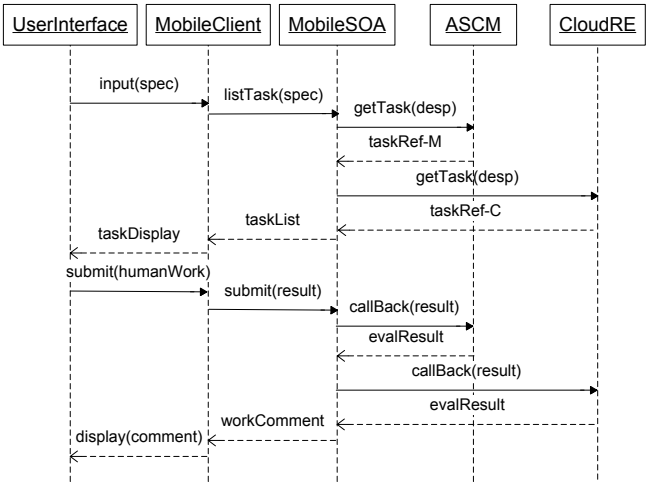

FIGURE 5. Process flow of human provided services.

As shown in Figure 5, mobile users who plan to participate in and help to finish some application tasks of crowdsensing can first submit their specialties and purposes to the *Mobile Client* through the user interface of Vita. Then, through the Mobile SOA framework, the expected description of the tasks will be sent to the ASCM and the cloud platform of Vita, and the cloud platform of Vita will push the information to other mobile users of Vita. As discussed above, with the help of social vector in ASCM, the Mobile SOA framework of Vita in the devices of other mobile users can automatically match the existing tasks and the expected tasks (according to their requirements and descriptions), and then return the results to the cloud platform. After that, the cloud platform of Vita will combine all the suitable tasks as a task list, and then return to the mobile devices of users who plan to participate in crowdsensing. Based on the task list and the related description of the tasks in the user interface of their mobile devices the users can select the crowdsensing tasks they prefer to participate in. After finishing the tasks, they can input their work through the user interface. The *Mobile Client* in their mobile devices will then submit the work to the *Transformer* of ASCM, transfer the work to standard web services, and finish the integration of the mobile crowdsensing tasks in the cloud platform of Vita. Finally, the *Mobile Client* will return the results of crowdsensing tasks with assessment comments from users (who posted the crowdsensing task before) via the cloud platform to the participants.

D. SERVICE STATE SYNCHRONIZATION MECHANISM

As mentioned in Section II, based on BPEL, we adopt the Pertri Net to design S3M, so as to handle the possible service failures of Vita, ensure the consistency of its services, and the correctness of the crowdsensing results. S3M is an optional component of Vita. Application developers could choose whether or not to integrate it in Vita according to their practical requirements (i.e., the overhead and reliability concern of the applications they are developing). S3M consists of two parts: to analyze the possible process failures of applications developed on Vita, and to automatically recover the corresponding failures to normal statuses. The design of S3M is based on the assumption that the Infrastructure as a Service (IaaS) provided by the cloud platform of Vita is always available.

1) ANALYZING THE FAILURES OF SERVICE STATE SYNCHRONIZATION ON THE VITA SYSTEM

Normally, in mobile environments, there are two reasons for the occurrence of service state synchronization failures: (i) Networking disconnection, such as the brief interruption caused by the automatic switching of networking access from cellular to WiFi, and/or a poor wireless signal, resulting in unstable networking connection; (ii). System crash in mobile devices, such as one that can occur when the batteries run out when the device is in use, or a collapse in the mobile operating system caused by instability of the mobile platform. Based on these conditions, in S3M, we design a model to analyze the possible failures in the service state synchronization of Vita.

In the model shown in Figure 6, hollow rectangles (the transition) indicate the regular state of service synchronization. A successful service state synchronization cycle of Vita around its mobile and cloud platform consists of the following steps:

(i) T0: the state synchronization initiator sends the related request to the communication channel.
(ii) T2: the request has been sent to the channel of the responder through a wireless network.
(iii) T3: the responder has received the request from its channel.
(iv) T13: the responder processes this request.
(v) T4: the responder sends the related request to the communication channel.

(vi) T5: the results are returned to the initiator through a wireless network.
(vii) T1: the initiator has received the results.

Further, the solid rectangles (the transitions) in Figure 6 indicate all of the possible failures in service synchronization. *Transitions NetInter1* and *NetInter2* indicate that due to the switch in networking access, an interruption of the network occurs when the message requesting state synchronization is being sent; and *Transition ServerCrash1* indicates that the server has crashed in the responder before it receives the synchronization request from other mobile devices (it should be noted that, here the server refers to the mobile SOA server of Vita in the mobile devices, not the servers in the Vita cloud platform). We define the three types of state synchronization failures mentioned above as *Service Unavailable Failures*, which means that the requestor will consider that the service of state synchronization is unavailable. In addition, in Figure 6, a token action is put in *SU_NI* or *SU_SC*, meaning that the *Service Unavailable Failure* has occurred.

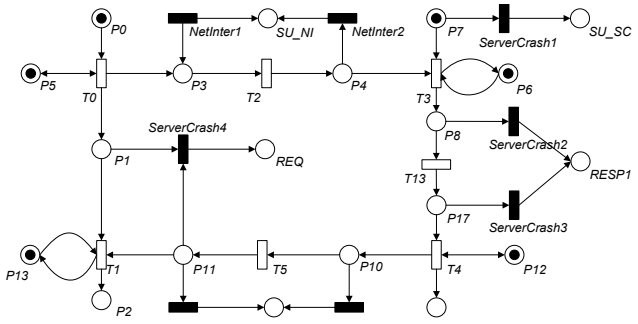

FIGURE 6. Analysis of service state failures on mobile platforms.

In addition, if *Transition ServerCrash2* or *ServerCrash3* occur, this indicates Vita's mobile SOA server has crashed while the response side (i.e., the cloud platform, and/or other mobile devices) is processing the message. This may result in the disconnection of the transportation layer and the loss of the response message. This type of failures is considered as *Pending Response Failures*. Similarly, a *Pending Response Failure* may also result when *Transition NetInter3* or *NetInter4* occurs, which refers to a network disconnection that happens when the request is being sent. Correspondingly, when a *Transition ServerCrash4* occurs we consider this to be a *Pending Request Failure*, which means that the requestor of the service synchronization crashed after it sent the request message.

2) PROCESS RECONFIGURATION-BASED SOLUTIONS FOR SERVICE STATE SYNCHRONIZATION

As the probabilities of *Pending Request Failures* and *Pending Response Failures* occurring are relatively low, and there are already well-established studies addressing these issues such as [38], thus in this paper we focus on the reconfiguration of *Service Unavailable Failures*. Considering the hardware and software constraints of mobile devices, we adopt a lightweight processing strategy. Based on the BPEL, we place *invoke* activity under *scope* activity (both are activities of BPEL), and insert the fault handler inside the *invoke* activity as a wait activity (which adds a delay) and an invoke activity to achieve the retry semantic. One constraint of this solution is that it can only send the request twice when the parter service is not available, while we can insert the scope activity with fault handler to the former fault handler multiple times so as to improve the allowances of the request times. The model of the process of reconfiguration is shown in Figure 7. If the *Transition ReSend* executes the resending of the request, and the server has been restarted (the Transition Restart happens), then this request will be accepted and the reconfiguration of the service state synchronization can be achieved.

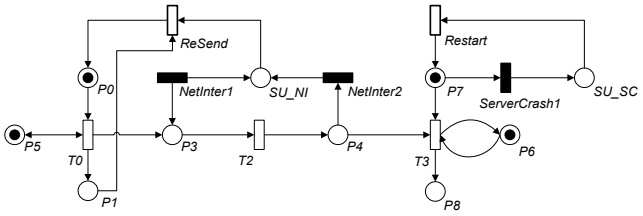

FIGURE 7. Solution for a service unavailable failure.

Moreover, just like Web Services BPEL (WS-BPEL) offers correlation sets to support transportation layer independent stateful protocols, as S3M works in a transport layer-independent way by transforming WS-BPEL specifications, thus different from the former solutions for service state synchronization [14]–[16], S3M is the first solution on mobile platform that could work independently of specific transport protocols.

IV. IMPLEMENTATION STRATEGIES

In this section, we discuss and present how the main components of the Vita system can be implemented in practice. More technical details about the key components of Vita, as well as a prototype and source codes can be found in the website of our project [39].

A. IMPLEMENTATION OF THE CLOUD PLATFORM OF VITA

In the current version of Vita, we adopt the Amazon Web Service (AWS) infrastructure services (i.e., EC2 and S3) and a series of open source techniques, such as JBoss jBPM [40], Apache ODE [41], Apache Tomcat, BPEL4People and WS-HumanTask [30] for the implementation of Vita's cloud platform. Other IaaS cloud computing platforms that are not AWS based could also be used to implement the Vita cloud platform. It consists of four parts: management interface, process runtime environment, storage service, and deployment environment.

1) MANAGEMENT INTERFACE

The management interface is implemented by integrating the Apache ODE management interface, the JBoss jBPM

management interface, and the development environment provision interface. For instance, as shown in Figure 8, the implementation of the development environment provision provides the ability to download sources and/or compiled releases of the software packages that are required for setting up the development environment, and related documentation and examples.

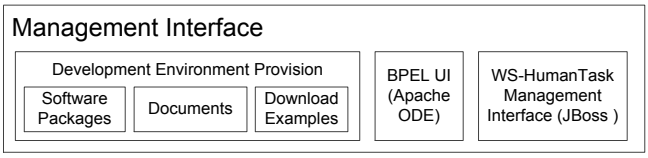

FIGURE 8. Implementation of the management interface.

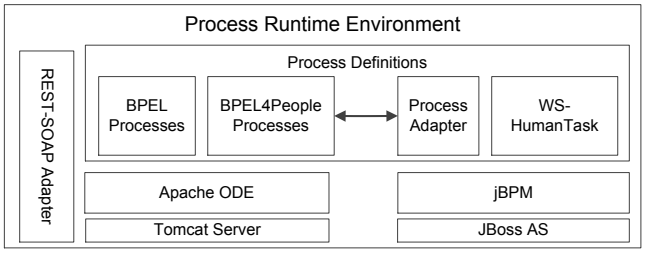

FIGURE 9. Implementation of the process runtime environment.

2) PROCESS RUNTIME ENVIRONMENT

As shown in Figure 9, the process run-time environment is implemented using two application servers: (a) the Apache Tomcat server for the setting up of the BPEL running environment - Apache ODE; and (b) JBoss AS (Application Server), on which the jBPM process manager is deployed. Based on these two open source business process runtime environments, BPEL processes and their extension BPEL4People processes can be deployed. Also, we use the process adapter to transform the corresponding BPEL4People part into the BPMN implementation of business processes, which can be integrated with WS-HumanTask. Moreover, as in the mobile environment of Vita, its services are REST-based, while BPEL only supports Simple Object Access Protocol (SOAP) based web services. Thus, the BPEL4People processes are not supported by Apache ODE. To address this issue, we use a REST-SOAP Adapter. This adapter can receive the SOAP service invocation request, and transform this request into the REST service invocation request.

3) STORAGE SERVICE

Based on the AWS S3 infrastructure, the storage service wraps the APIs for all of the data storage requirements from other modules: the data for the WS-HumanTask, the related software packages, examples of documents for the development provisioning, and the mobile SOA environment provisioning.

4) DEPLOYMENT ENVIRONMENT

The deployment environment is composed of three modules. We integrate the Apache ODE deployment environment and the JBoss jBPM deployment environment to form a base for the Management Interface to support BPEL and the BPEL4People development environment. Based on the storage service module, we implemented the mobile SOA environment provision module.

B. IMPLEMENTATION OF THE ASCM

The current implementation of ASCM in Vita is based on the Android operating system and open source techniques. As indicated in Section III-C, beyond the social vector and *Task Client*, which could be developed on Android directly, the major implementation task of ASCM is the *Transformer*. In order to implement the *Transformer*, we adopt the open source techniques BPEL4People and WS-HumanTask [30]. Based on WS-HumanTask, we develop a sub-model inside the *Transformer*, which could describe and transfer the heterogeneous humantask data to standard data format. Then through the *HumanTask Activity* that is developed based on the BPEL4People, the processed data can be encapsulated to standard web service and deployed to the mobile SOA framework as a human-based service. In addition, if a similar humantask has been deployed and finished in the cloud platform of Vita, the result can be accessed by the *Human-based Services* through the mobile SOA framework.

C. IMPLEMENTATION OF THE S3M

As mentioned in Section III-D, S3M is implemented using BPEL, and thus in order to make S3M workable on mobile devices like Android, a BPEL engine needs to be developed for mobile platforms.Considering the capacity constraints of Android devices, we developed a lightweight mobile BPEL engine on the mobile part of Vita, which architecture is shown in Figure 10.

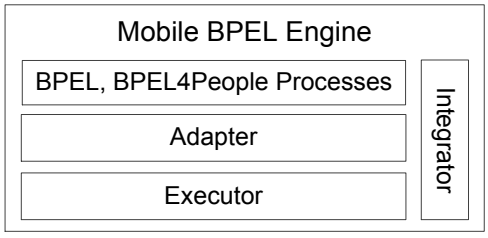

FIGURE 10. Implementation of mobile BPEL.

In the current version of Vita, the following activities are supported by the Mobile BPEL engine:

Basic activities

<receive>, receives an incoming message

<reply>, replies with the response message

<invoke>, invokes the REST-based web services

<assign>, assignment to update the values of process variables.

Structured activities

<if>, conditional branches of processes

<pick>, multiple execution branches based on incoming messages.

<sequence>, the sequential execution of sub-activities

<scope>, defines a nested activity with its own associated elements.

The communications between the cloud platform of Vita and its mobile platform employ the standard web service format based on the HTTP protocol and Extensible Markup Language (XML) data format. In addition, although BPEL interactions on the Vita cloud side are SOAP based, and its services in the mobile platform are RESTful Web Services based, the SOAP-REST transformation can be achieved using additional adapters in between, similar to the method described above forthe cloud platform.

V. APPLICATION EXAMPLE

In this section, we present a concrete application call *Smart City* developed on Vita, so as to demonstrate the functionalities of Vita and the applications of mobile CPS for crowdsensing in our daily lives. *Smart City* consists of two generic functions: services, crowdsensing; and two application specific functions: eating and shopping tour; the screen shots of some of these functions are shown in Figure 11.

1) GENERIC FUNCTIONS

Services: This function is based on the mobile SOA framework of Vita and takes advantages of the RESTful Web Service architecture. Application developers could flexibly extend new functions here according to their practical requirements for different mobile CPS based crowdsensing applications. Furthermore, based on ASCM, developers could design and develop application specific service sharing strategies here, which enable the users to easily share functions in mobile devices' run-time during some specific crowdsensing scenarios via the cloud platform of Vita.

Crowdsensing: This function is mainly based on the ASCM, social network services of the mobile SOA framework, and the cloud platform of Vita. As demonstrated in the center screen shot in Figure 11, through this function, mobile users could post crowdsensing requests through social networks and find out the potential people who could help to accomplish the tasks (with the help of social vector in ASCM), and/or accept new crowdsensing tasks by choosing the preferable task on the list.

2) APPLICATION SPECIFIC FUNCTIONS

Eating: Beyond the key components of Vita mentioned above, this function is also based on the location and map services provided by Google Map. As the screen shot in the left corner of Figure 11 shows, the eating function consists of four sub-functions: i-Ask, Search, Comment, and Photo, it is designed to enable people who travel in a new city to conveniently find out and/or share food information that they are interested in real-time.

Here, we set up an experimental application scenario, so as to demonstrate this function. In this case, it consists of five persons each carrying an Android phone and has installed the *Smart City*. Assume that one of the persons is a visitor in Hong Kong called Blair who has traveled from Vancouver,

FIGURE 11. Smart city - mobile CPS based crowdsensing application.

and she wants to taste some local food in Hong Kong which Vancouver does not have; and three of the other persons are in restaurants. As shown in Figure 11, she posts the related crowdsensing request through the eating function of the *Smart City* application on her phone, as follows: *What food can be found here that is not available in Vancouver?* After that, based on the location service, and with the help of ASCM and cloud platform of Vita, it could automatically push the request to the three persons who are dining in restaurants and have experience about this (i.e., two of them had lived in Vancouver before). Then, through the user interface of the *Smart City*, they take a photograph of the food and attach simple comments. The photograph and comments are automatically combined with the location service and map service, and then the answers are uploaded to the cloud platform of Vita. After the cloud platform of Vita gets all of the answers, it can automatically match the other answers stored in it before through the specific information. The overall answers are then returned to Blair's phone (shown in the left corner of Figure 11 which consists of three appropriate answers). We found that the time taken by Vita to return the answers after she made the request was about 2-3 minutes.

Finally, with the help of mobile crowdsensing through *Smart City*, Blair enjoys local flavors in Hong Kong that are new to her. She wants to share her experience with others, to help the next new visitors. To do this, using her phone she inputs the information similar to those obtained through mobile crowdsensing before. Such information (comments, photograph with address, etc.) can then be directly shared with other visitors with the same request, or stored on the cloud platform of Vita as a new service. Moreover, the cloud platform of Vita could record and count the statistics that which people have been made contributions to crowdsensing, and grants their prior services when they post requests in the future. This gives incentives to entice more people to join crowdsensing via the *Smart City*.

Shopping tour: Similar to the eating function, this function is designed to assist users in a city to easily find out and/or share the shopping information they are interested in real-time. As shown in the screen shot in the right corner of Figure 11, different from the above eating example that mainly demonstrates the collection of human intelligence in the mobile CPS – Vita to accomplish application purposes for people, this example demonstrates that Vita can leverage the advantages of Internet services (i.e., today's shopping information) and cloud computing platform to enable people to aggregate and realize the shopping information that they are interested in through mobile devices in a convenient and efficient manner.

VI. EVALUATIONS

In this section, we evaluate Vita in three aspects: development support, overall system and tasks performance, and the performance of ASCM. In addition, we present practical cases to demonstrate the reliability of the pro-posed S3M.

TABLE 2. Developed crowdsensing applications.

	Campus tours	**Real-time campus events**
Example	Find the WinMos lab at UBC	Find today's presentations on mobile apps and networks at PolyU
Consists of web services (for no. refer to lines of code)	- Photo Tag. Ser. (50) - Location Ser.(30) - Map Ser.(36)	- Location Ser.(30) - Map Ser.(36) -Classify & List Ser.(93)
Lines of code	275	326
	Total: 535	

A. DEVELOPMENT SUPPORT

In this part we present the design and development of two prototype applications in the campus context, so as to demonstrate the advantages and usefulness of Vita for developing mobile CPS based crowdsensing applications. The applications are developed using standard Java for Android according to the specifications of the REST Web Service and the BPEL XML schema of Vita. Both of these applications are implemented through the composition of several REST Web Services and related clients, part of the source code of which could be found in our website [39].

As shown in Table 2, the standalone lines of code (LOC) for the implementation of the applications are 275 and 326. Aside from the codes to implement the web services, other codes are mainly XML-based (for real-time service composition in mobile devices). Even though the LOC looks longer than that of the web services taken together, this is not indicative of the development effort, which is much smaller than the latter. Also, because the applications are developed in Vita, which is based on the standard specifications of RESTful web service, the services can readily be reused and composited; e.g., the location and map services are reused in both application examples. Thus, the total LOCs needed to implement these two applications together are less than the sum of LOCs to implement them individually. Furthermore, this means that if more applications are developed in Vita, more savings in development efforts will be realized through the benefits of service reuse and composition.

1) CAMPUS TOUR

This application is used to help new students find a place on campus through mobile crowdsensing. Suppose a new student wants to find the *WinMos* lab at UBC. After she posted the crowdsensing request through his Android phone, someone went past the ICICS building and knew that the lab is in this building. That person took a photograph of this building and put information on its location (*It is in the back of the ICICS building, Rm. X327*) through the photo tagging service. This application then automatically transferred the information as a *New Service* and combined it with its existing Map Service and Location Service, and sent the result, as shown in Figure 12, to the requestor's phone.

FIGURE 12. **Campus tour example and result.**

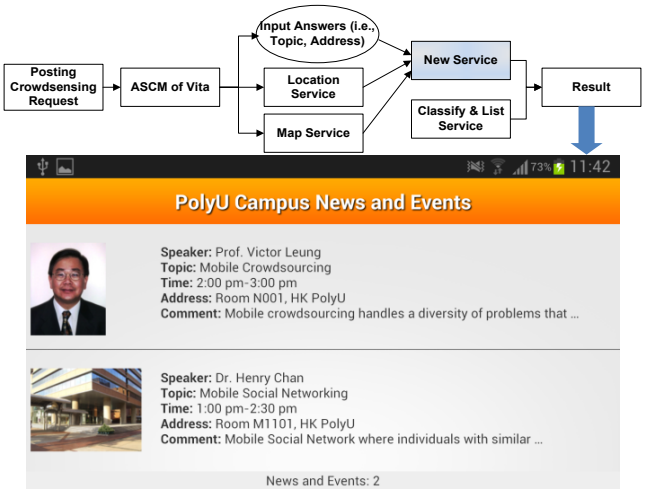

FIGURE 13. **Example and result of a real-time campus event.**

2) REAL-TIME CAMPUS EVENTS

This application can help people on campus find out about real-time events of interest. In this example, a person on campus wants to know whether or not there are some academic presentations in PolyU about mobile applications and networks, then with the similar working process as mentioned above, he can obtain answers as shown in Figure 13.

B. SYSTEM AND TASKS PERFORMANCE

We evaluate the system and task performances of Vita in terms of three parameters: time efficiency, energy consumption, and networking overhead in mobile devices, when they finish crowdsensing and concurrent computation tasks, as these parameters have great impacts on the experience of mobile users when they are participating in mobile crowdsensing. The experimental environment is: ***Hardware:*** Amazon EC2 M1 Medium Instance; 3.75 GiB memory; 2 EC2 Compute Unit (1 virtual core with 2 EC2 Compute Unit); 410 GB instance storage; 32-bit or 64-bit platform; I/O Performance: Moderate; EBS-Optimized Available: No.

Software: operating system: Ubuntu 12.04.1; Servers: ApacheTomcat 7.0.33 and JBoss AS 7.1.1; BPEL engine BPEL4People environment: ODE1.3.5 and jBPM 5.4.

Mobile Devices: A variety of mobile devices are employed, which are described below for each set of experiments.

1) TIME DELAY, BASIC BATTERY CONSUMPTION, AND NETWORKING OVERHEAD OF THE SYSTEM

We tested the basic performance of the Vita system in two places simultaneously: Vancouver and Hong Kong. The experimental mobile devices are Google Nexus 10 (Android 4.2.1 version, battery capacity 9000mAh). We ran 10 tests over 3 days, then calculated the average value of the test results. Each experiment lasted 45 minutes. Each Nexus 10 sent crowdsensing requests to the servers on the cloud platform of Vita according to a Poisson distribution with an arrival rate of $E = 5$ requests/minute, and the screens of the devices were shut off during the runtime. The time delay refers to the periods between the time that the Nexus 10 initiates a crowdsensing (with no data process) request to the cloud platform and the time that it receives the responses from the servers on the cloud platform of Vita. In addition, we classify the results of the experiment into two sets of data: common (without the S3M) and with S3M, so as to determine the additional overhead when loading the S3M, as this model is optional in stable network situations.

TABLE 3. **Overall system performance of vita.**

Parameters	Data set-Vancouver		Data set-Hong Kong		Medusa [42]
Time delay (msec)	**Common**	**With S3M**	**Common**	**With S3M**	N/A
	Max.: 22636, Min.: 2399	Max.: 21319, Min.: 10388	Max.: 15682, Min.: 2399	Max.: 20326, Min.: 10415	Max.: 138758.3, Min.:40617.16
	Ave.: 10577	Ave.: 12804	Ave.: 11507	Ave.: 12913	Ave.: 63988.86
Battery consumption	80mAh/45mins	140mAh/45mins	80mAh/45mins	140mAh/45mins	N/A
Network overhead	0.85MB/220 requests	1.27MB/231 requests	0.83MB/216 requests	1.23MB/223 requests	N/A

The experimental results are summarized in Table 3. We find that the results in Vancouver and Hong Kong are very similar, with all averaging about 11s in the common configuration and 12.8s with S3M loaded. Under the same conditions, the additional time delay with S3M is low, while the increase in battery consumption and network overhead are relatively higher, at about 75% and 43% respectively. Based on these results, developers can choose whether or not to integrate the S3M model according to their specific purposes when developing mobile crowdsensing applications on Vita. Moreover, we make a simple comparison with the related work Medusa [42], where the corresponding time delay is about 64s, although their runtime environments are different. The main reasons for this are that Medusa adopts the commercial Amazon AMT as the crowdsourcing platform and Short Message Service (SMS) to deliver the message, which involves delays of about 31s and 27s, respectively. In contrast, Vita uses open sourced services to develop the customized crowdsensing platform and exchanges service requests using standard web service messages, which is more efficient than the approach of Medusa.

2) TIME EFFICIENCY AND COMPUTATIONAL OVERHEAD OF CONCURRENT TASKS

As Vita adopts RESTful Web Services, mobile applications developed in Vita are based on web services. Also, each mobile device may need to execute multiple tasks simultaneously in the practical application scenarios of crowdsensing. Thus, we developed a web service based application on Vita, which calculates the representative benchmark *N-Queens Puzzle* [43] both in mobile devices and the cloud platform of Vita, so as to test the efficiency and computational overhead (battery consumption) of Vita's tasks. In addition, as the communication overhead of the system itself has been evaluated in the last part, and the additional communication overhead mostly depend on the types of applications (i.e., multimedia related or textual content), thus in this part we skip this aspect. In the experiment, we use Google Nexus S (Android 4.1.2 version, 1500 mAh battery capacity), and consider N from 12 to 16. The number of concurrent tasks follows a Poisson distribution with arrival rates of $E = 1$ or 3 tasks per minute, respectively. The time efficiency and battery consumption of each task were record over 1 hour periods, then the average values were computed. We considered both tasks completed only in the Nexus S or by the server in the cloud platform of Vita when being uploaded to it. The screen was shut off during the runtime of tasks.

The results are shown in Figures 14 and 15. From the results, we find that Nexus S cannot finish the tasks when $N = 16$ even when $E = 1$. When $N < 15$, both execution time and battery consumption performances of the tasks completed in Nexus S are better than those of tasks uploaded and completed in the cloud. However, when $N = 15$, both the time and battery consumptions become much higher when E is changed from 1 to 3. Considering the computationally intensive nature of the *N-Queens Puzzle* (when $N = 15$) and the hardware capacity of

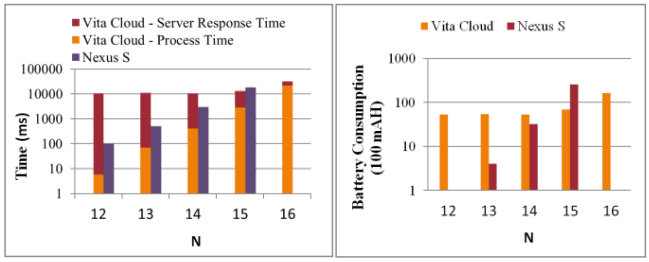

FIGURE 14. Execution time and battery consumption of the N-queens puzzle (E = 1).

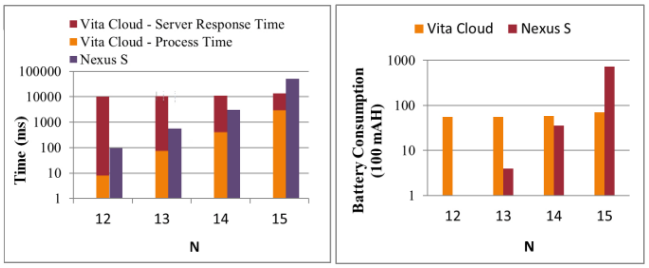

FIGURE 15. Execution time and batter consumption of the N-queens puzzle (E = 3).

Nexus S, this means that the web service-based mobile applications developed on Vita for mobile crowdsensing applications can work with high efficiency and low overheads in many of the popular Android devices when performing most of the common computation tasks. Furthermore, Vita supports applications with more complicated tasks by leveraging the advantages of cloud computing to process multiple demanding tasks (with a high concurrency value E).

C. PERFORMANCE OF ASCM

As mentioned in Section III-C.1, in the social vector of ASCM, we propose to incorporate the two approaches—Genetic Algorithm (GA) and K-means clustering as solutions for finding optimized crowd groups among different individuals for crowdsensing application tasks. Due to the experimental constraints, e.g., it is not easy to find enough volunteers to do the evaluation multiple times in real world, thus, here we set up a Java simulation program to evaluate the performance of these algorithms. The experimental mobile devices are also Google Nexus 10. Each experiment involves 15 people, each of whom has i social attributes indicating whether the person has knowledge K_i. Each social attribute has a value between 0.0 and 1.0, generated by a random number generator. We assign these people to K groups (each group corresponds to a task). A social attribute a_i indicates whether a person P has knowledge K_i. In other words, if P has K_i, $a_i = 1$; otherwise $a_i = 0$. The social vector between a person P and a set of knowledge K is defined by

$$\overrightarrow{V_{T_i,S}} = [a_1, a_2 \ldots, a_m]$$

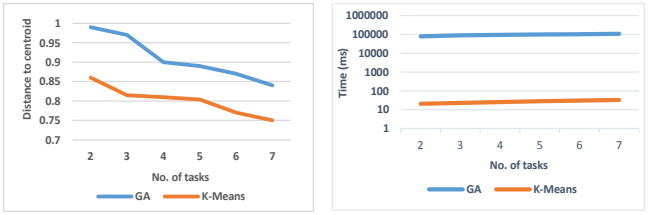

FIGURE 16. Evaluation results of GA and K-means clustering (different no. of tasks).

where m is an integer. Assume that there is a task T_i for crowdsensing which is formed by a group of people P_j, where $j = 1, 2, \ldots, L_i$, and $\sum_{i=0}^{N} L_i = M$ (M means the number of mobile users, and N means the number of tasks). The social vector between a team for crowdsensing and a set of knowledge K is defined by

$$\overrightarrow{V_{T_i,K}} = \frac{1}{L}\left[\sum_{j=1}^{L}\overrightarrow{V_{P_j,K}}(1), \sum_{j=1}^{L}\overrightarrow{V_{P_j,K}}(2), \ldots, \sum_{j=1}^{L}\overrightarrow{V_{P_j,K}}(L)\right]$$

Good teams for crowdsensing are found if $|\overrightarrow{V_{T_i,K}}|$ is as large as possible for all i. For GA, the simulation process ends after 1000 generations. As for K-means clustering, the simulation process is repeated until no people need to be re-assigned from one group to another. We evaluate the effectiveness of GA and K-means clustering under different required numbers of group. Figure 16 shows the results and comparison of GA and K-means clustering. Longer distance between the people and the centroids of groups implies that every person in the group has knowledge in a wider range. It can be seen that the average distance between people and the centroids of the groups decreases when the number of tasks increases. It is because when the number of tasks increases, the probability that people can be arranged into a farthest group is decreased. Besides, GA achieves better performance than K-means clustering because it can arrange people to groups more intelligently. With GA, the average distance between people and the centroids of groups is about 0.9 when the required number of groups is 5. With K-means clustering, the average distance between people and the centroids of groups is shorter than 0.8 when the required number of groups is 5.

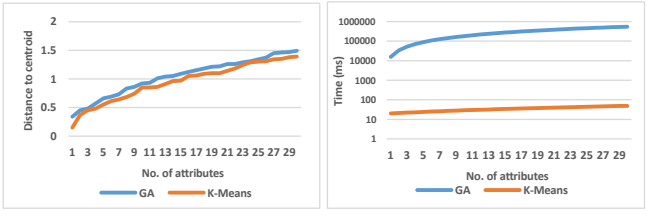

FIGURE 17. Evaluation results of GA and K-means clustering (different no. of attributes).

We further evaluate the effectiveness of GA and K-means clustering under different number of attributes. Figure 17 shows the results and comparison of GA and K-means clustering. Longer distance between the people and the centroids of their group implies that every person in the group has different knowledge. It can be seen that the average distance between people and the centroids of the group increases when the number of attributes increases. Besides, GA achieves better performance because it can assign people to different groups more intelligently. With GA, the average distance between people and community centroids is longer than 1.2 when the number of social attributes is 20. K-means clustering performs worse than GA. With K-means clustering, the average distance between people and the centroids of group is about 1.1 when the number of attributes is 20.

Note that GA can achieve better performance (i.e., longer distance between the people and the centroids of their groups) than K-means clustering but the computation overhead and response time of GA is much higher than K-means clustering. Thus GA should be employed only if the accuracy of assigning people to different groups outweighs the response time. In most cases, K-means clustering should be employed because of the limited processing power of the system.

D. RELIABILITY OF S3M

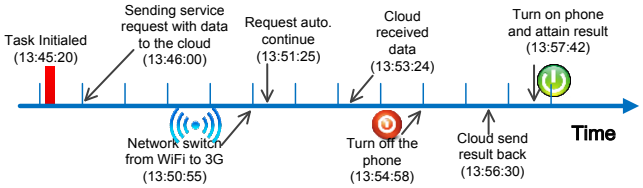

FIGURE 18. Failure detection and recovery of S3M on mobile device.

As proposed in Section III-D, we designed S3M to detect and recover the possible service failures of mobile devices when they are running tasks and collaborating with the cloud platform of Vita. Here we use a practical case to demonstrate and evaluate its feasibility. Figure 18 shows the sequence of events. After the task on a mobile device had been initialized and the related service request had been sent with data to the cloud platform of Vita for processing, to demonstrate the failures, we switched the networking access of the mobile device from WiFi to 3G, and turned off the mobile device for about 3 minutes during the experiment. It was found that the task had been processed on the cloud platform of Vita successfully, and the result of the task was eventually returned and accomplished in the mobile device. Similar service failures could also be detected and recovered with the help of S3M, as S3M includes the function to store the stage execution state on both the mobile device and the cloud platform of Vita.

VII. RELATED WORK

There have been several research works about mobile CPS based solutions for crowdsensing. The different works are differentiated by: (i) Targeting some specific application sce-

narios, e.g., using smart phone to provide feedbacks to users about their transportation behaviors [7], monitoring personal heart information [8], and predicting bus arrival times [44]; (ii) Using specific techniques to construct stand alone crowdsensing system to support multiple mobile crowdsensing applications, e.g., using Twitter [10], and using an in-node hardware abstraction layer and overlay management protocol to allow multiple applications sharing sensing data across different mobile nodes [45]; and (iii) Supporting efficient development of different mobile crowdsensing applications; e.g., the work in [46] aims to enable developers to write server-side programs in lieu of distributed programs, so as to ease the development of crowdsensing applications on smartphones.

Partly inspired by but different from these works, Vita aims to provide a general approach at the system and architecture design level by leveraging the advantages of a number of techniques, which salient parts are integrated and orchestrated into a flexible, efficient and economic platform for development and execution of crowdsensing applications. It not only supports the development of multiple mobile crowdsensing applications with standard service interactions, but also enables the deployment of a mobile distributed CPS with augmented cloud computing platform,which empowers people's mobile devices with powerful capabilities to allow them to easily and efficiently participate in and perform multiple and diverse crowdsensing tasks. In the following we contrast Vita with two existing works that are particularly close to our work.

Medusa [42] is a programming framework that provides a programming language and a runtime system for efficiently developing mobile crowd-sensing applications, which enables reduction of the tasks to smaller pieces compared to standalone systems for crowdsensing applications. It employs a distributed system architecture to coordinate the tasks between cloud servers and smart phones. Compared to Medusa, Vita has several advantages: (i) Vita provides a more flexible distributed system architecture between the mobile platform and the cloud platform. It simultaneously supports dynamic balancing of the allocation of computation tasks between mobile devices and the cloud platform according to specific application requirements and scenarios, and efficient allocation of human-based tasks among individuals. (ii) The Amazon AMT adopted by Medusa in the execution of crowdsensing tasks can only be used inside the USA. Moreover, AMT is for commercial use and is not open source. It does not support the development of customized crowdsensing mechanisms and platforms by third parties, while Vita adopts open source techniques and industrial standards such as BPEL4People, and leverages the advantages of social networking services to set up the cloud platform, which can therefore be more universally used, and supports flexible extension and customized redevelopment.

On the other hand, the current weaknesses of Vita comparing to Medusa are mainly in two aspects: (i) Vita lacks a mature incentive mechanism to encourage people to participate in crowdsensing; (ii) The security of diverse crowdsensing applications and data on Vita is still a concern, as the current implementation of Vita does not include any security mechanism. However, as Vita fully adopts an open architecture, thus the application developers of Vita can flexibly design different customized mechanisms to address these issues according to their specific requirements. Further, we shall extend Vita to include incentive mechanisms and improve security and privacy measures in the future.

ThinkAir [47] is a software framework that supports the migration of computation tasks of smartphone applications to the cloud through mobile code offloading, so as to achieve dynamic computational resource allocation and parallel execution between the smartphone and the cloud computing platform. Different from ThinkAir, Vita adopts the RESTful Web Service architecture both in its mobile platform and cloud platform. In Vita, all of the data and codes of computation tasks offloaded to the cloud from mobile devices are in the format of standard web services. This enables the data transmissions to be more application-related, making the instance object fields a better match with the corresponding methods, and avoiding unnecessary network overhead. Also, Vita supports the reuse of web services. As web services are also written as software codes, web service reuse implies code reuse. However, more than code reuse that usually takes place at development or compilation time, web service reuse further enables sharing of available services among multiple mobile applications during their runtime. This means that rather than balancing the computation tasks between mobile devices and the cloud platform at the coding stage, Vita also enables this balancing to occur dynamically at run time, and mobile users can further act as web service providers by sharing the services available in their devices with other users directly or via the cloud platform.

VIII. CONCLUSION

In this paper, we have presented Vita, a novel mobile CPS for crowdsensing, which leverages the advantages of social computing, service computing, cloud computing, and a number of open source techniques across mobile devices and cloud platform, to provide a systematic approach that supports both application developers and users for mobile crowdsensing applications. Our practical experiments have demonstrated that Vita simplifies the development of multiple mobile crowdsensing applications, and performs its tasks with a considerable time efficiency, low battery consumption and low communication overhead on mobile devices. We have also developed sample applications based on Vita to demonstrate its functionalities. To the best of our knowledge, Vita is the first comprehensive open source mobile CPS that supports the development, deployment and management of multiple mobile crowdsensing applications or tasks in an efficient and flexible manner. Also, Vita is the first mobile distributed CPS that simultaneously supports the automatic allocation of human-based tasks among individuals and computing tasks between mobile devices and cloud platform in an efficient

manner, with reliability enhancement mechanism on mobile devices, which enables mobile users to participate in and perform crowdsensing applications and tasks in a convenient and efficient way.

REFERENCES

[1] M. Conti, S. K. Das, C. Bisdikian, M. Kumar, L. M. Ni, A. Passarella, G. Roussos, G. Tröster, G. Tsudik, and F. Zambonelli, "Looking ahead in pervasive computing: Challenges and opportunities in the era of cyber–physical convergence," *J. Pervas. Mobile Comput.*, vol. 8, no. 1, pp. 2–21, Feb. 2012.

[2] J. Sztipanovits, "Composition of cyber–physical systems," in *Proc. 14th IEEE ECBS*, Mar. 2007, pp. 3–6.

[3] J. White, S. Clarke, B. Dougherty, C. Thompson, and D. C. Schmidt, "R&D challenges and solu-tions for mobile cyber–physical applications and supporting internet services," *J. Internet Services Appl.*, vol. 1, no. 1, pp. 45–56, May 2010.

[4] M. D. Ilić, L. Xie, U. A. Khan, and J. M. F. Moura, "Modeling of future Cyber–Physical Energy Systems for distributed sensing and control," *IEEE Trans. Syst., Man, Cybern., Part A, Syst. Humans*, vol. 40, no. 4, pp. 825–838, Jul. 2010.

[5] X. Li, C. Qiao, X. Yu, A. Wagh, R. Sudhaakar, and S. Addepalli, "Toward effective service scheduling for human drivers in vehicular cyber–physical systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 23, no. 9, pp. 1775–1789, Sep. 2012.

[6] R. K. Ganti, F. Ye, and H. Lei, "Mobile crowdsensing: Current state and future challenges," *IEEE Commun. Mag.*, vol. 49, no. 11, pp. 32–39, Nov. 2011.

[7] J. Froehlich, T. Dillahunt, P. Klasnja, J. Mankoff, S. Consolvo, B. Harri-son, and J. Landay, "UbiGreen: Investigating a mobile tool for tracking and supporting green transportation habits," in *Proc. ACM Human Factors Comput. Syst.*, 2009, pp. 1043–1052.

[8] P. Leijdekkers and V. Gay, "Personal heart monitoring and rehabilitation system using smart phones," in *Proc. ICMB*, Jun. 2006, p. 29.

[9] P. Mohan, V. Padmanabhan, and R. Ramjee. Nericell, "Rich monitor-ing of road and traffic conditions using mobile smartphones," in *Proc. ACM Embedded Netw. Sensor Syst.*, 2008, pp. 323–336.

[10] M. Demirbas, M. Bayir, C. Akcora, and Y. Yilmaz, "Crowd-sourced sensing and collaboration using twitter," in *Proc. IEEE Int. Symp. WoWMoM*, Jun. 2010, pp. 1–9.

[11] K. Baheti and H. Gill, "Cyber-physical systems," in *Proc. ICCPS*, Apr. 2011.

[12] R. Rajkumar, I. Lee, L. Sha, and J. Stankovic, "Cyber-physical systems: The next computing revolution," in *Proc. Design Autom. Conf.*, 2010, pp. 731–736.

[13] E. A. Lee, "Cyber physical systems: Design challenges," in *Proc. 11th IEEE ISORC*, May 2008, pp. 363–369.

[14] A. Charfi, T. Dinkelaker, and M. Mezini, "A plug-in architecture for self-adaptive web service compositions," in *Proc. IEEE ICWS*, Jul. 2009, pp. 35–42.

[15] O. Moser, F. Rosenberg, and S. Dustdar, "Non-intrusive monitoring and service adaptation for WS-BPEL," in *Proc. 17th Int. Conf. WWW*, 2008, pp. 815–824.

[16] (2009). *Web Services Atomic Transaction Version* [Online]. Available: http://docs.oasis-open.org/ws-tx/wsat/2006/06

[17] X. Hu, V. C. M. Leung, W. Du, B. C. Seet, and P. Nasiopoulos, "A service-oriented mobile social networking platform for disaster situa-tions," in *Proc. 46th HICSS*, Jan. 2013, pp. 136–145.

[18] X. Hu, V. C. M. Leung, and W. Wang, "VSSA: A service-oriented Vehicular social-networking platform for transportation efficiency," in *Proc. ACM DIVA Net.*, Oct. 2012, pp. 31–38.

[19] P. Mohan, V. Padmanabhan, and R. Ramjee, "Nericell: Rich monitoring of road and traffic conditions using mobile smartphones," in *Proc. 6th ACM Conf. Embedded Netw. Sensor Syst.*, 2008, pp. 323–336.

[20] C. Thompson, J. White, B. Dougherty, and D. Schmidt, "Optimizing mobile application performance with model—Driven engineering," in *Proc. 7th Int. Workshop Softw. Technol. Future Embedded Ubiquitous Syst.*, 2009, pp. 36–46.

[21] C. Efstratiou, I. Leontiadis, M. Picone, K. K. Rachuri, C. Mascolo, and J. Crowcroft, "Sense and sensibility in a pervasive world," in *Proc. 10th Int. Conf. Pervas.*, Jun. 2012, pp. 406–424.

[22] I. Li, A. K. Dey, and J. Forlizzi, "Understanding my data, myself: Supporting self-reflection with Ubicomp Technologies," in *Proc. 13th Int. Conf. Ubiquitous Comput.*, 2011, pp. 405–414.

[23] M. Boulos, B. Resch, D. N. Crowley, J. G. Breslin, G. Sohn, R. Burtner, W. A. Pike, E. Jezierski, and K.-Y. S. Chuang, "Crowdsourcing, citizen sensing and sensor web technologies for public and environmental health surveillance and cri-sis management: Trends, OGC standards and application examples," *J. Health Geograph.*, vol. 10, no. 67, pp. 1–29, Dec. 2011.

[24] *Chronic Disease Prevention and Overview* [Online]. Available: http://www.cdc.gov/nccdphp/overview.htm#2, Accessed: 2013.

[25] InSTEDD. (2006). *GeoChat*, Sunnyvale, CA, USA [Online]. Available: http://instedd.org/technologies/geochat/

[26] *The Ushahidi Platform* [Online]. Available: http://ushahidi.com/products/ushahidi-platform, Accessed: 2013.

[27] C. Pautasso, O. Zimmermann, and F. Leymann, "RESTful web services vs. 'big' web services: Making the right architectural decision," in *Proc. 17th Int. Conf. WWW*, 2008, pp. 805–814.

[28] (2007). *Web Services Business Process Execution Language Version 2.0* [Online]. Available: http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf

[29] T. Murata, "Petri nets: Properties, analysis and applications," *Proc. IEEE*, vol. 77, no. 4, pp. 541–580, Apr. 1989.

[30] (2013). *WS-HumanTask* [Online]. Available: http://docs.oasis-open.org/ns/bpel4people/ws-humantask/200803

[31] X. Li, S. Madnick, H. Zhu, and Y. Fan, "Reconciling semantic hetero-geneity in web services composition," in *Proc. ICIS*, 2009.

[32] X. Li, S. Madnick, and H. Zhu, "A context-based approach to reconciling data interpretation conflicts in web services composition," *ACM Trans. Internet Technol.*, vol. 8, no. 1, pp. 1–4, 2013.

[33] J. Wan, H. Yan, H. Suo, and F. Li, "Advances in cyber-physical systems research," *KSII Trans. Internet Inf. Syst.*, vol. 5, no. 11, pp. 1891–1908, Jan. 2011.

[34] J. Shi, J. Wan, H. Yan, and H. Suo, "A survey of cyber–physical systems," in *Proc. WCSP*, 2011, pp. 1–6.

[35] T. Hnat, T. Sookoor, P. Hooimeijer, W. Weimer, and K. Whitehouse, "MacroLab: A vector-based macro programming framework for Cy-ber-physical systems," in *Proc. 6th ACM Conf. Embedded Netw. Sensor Syst.*, 2008, pp. 225–238.

[36] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA, USA: MIT Press, 1992.

[37] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu, "An efficient K-means clustering algorithm: Analysis and implementation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, no. 7, pp. 881–892, Jul. 2002.

[38] L. Wang, A. Wombacher, L. Ferreira Pires, M. Sinderen, and C. H. Chi, "A state synchronization mechanism for orchestrated processes," in *Proc. IEEE 16th EDOC*, Sep. 2012, pp. 51–60.

[39] (2013). *Vita: A Crowdsensing-oriented Mobile Cyber Physical System* [Online]. Available: http://mobilesoa.appspot.com/

[40] (2012). *Jboss* [Online]. Available: http://docs.jboss.org/jbpm/v5.0/userguide/ch.Human_Tasks.html

[41] (2012). *Apache ODE* [Online]. Available: http://ode.apache.org/

[42] M. Ra, B. Liu, T. Porta, and R. Govindan, "Medusa: A programming framework for crowd-sensing applications," in *Proc. 10th Int. Conf. Mobile Syst., Appl. Services (MobiSys)*, 2012.

[43] R. Sosich and J. Gu, "Fast search algorithms for the N-queens problem," *IEEE Trans. Syst., Man, Cybern.*, vol. 21, no. 6, pp. 1572–1576, Nov./Dec. 1991.

[44] P. Zhou, Y. Zheng, and M. Li, "How long to wait?: Predicting bus arrival time with mobile phone based participatory sensing," in *Proc. 10th Int. Conf. Mobile Syst., Appl., Services*, 2012, pp. 379–392.

[45] I. Leontiadis, C. Efstratiou, C. Mascolo, and J. Crowcroft, "SenShare: Transforming sensor networks into multi-application sensing infra-structures," in *Proc. EWSN*, 2012, pp. 65–81.

[46] L. Ravindranath, A. Thiagarajan, H. Balakrishnan, and S. Madden, "Code in the air: Simplifying sensing on smartphones," in *Proc. 8th ACM Conf. Embedded Netw. Sensor Syst.*, 2012.

[47] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "ThinkAir: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *Proc. IEEE INFOCOM*, Mar. 2012, pp. 945–953.

XIPING HU joined the University of British Columbia (UBC), Vancouver, BC, Canada, in 2011, and is currently pursuing the Ph.D. degree with the Electrical and Computer Engineering Department, UBC. He was a Research Assistant with the National Research Council of Canada - Institute for Information Technology (NRC-IIT) and the University of New Brunswick (UNB), Fredericton, NB, Canada, and received the master's degree in computer science from UNB in 2011. He is the winner of silver prizes in national Olympic competitions in mathematics and physics in China, and a Microsoft certified specialist in web applications and SQL server. He has participated as a key member in several research projects, like web service security identification with Tsinghua University, Beijing, China, SAVOIR project at NRC-IIT, and NSERC DIVA research strategy network at UBC. He has published in several international conferences and journals, such as Procedia CS, IEEE HICSS, and ACM DIVANet. His current research interests include mobile social networks, software architecture, crowd sourcing, and human computer interaction.

TERRY H. S. CHU received the B.Sc. (Hons.) degree in computing from The Hong Kong Polytechnic University, Hong Kong, in 2010. He has received six scholarships, such as Simatelex Charitable Foundation Scholarship in 2010 and 2008, Bank of East Asia Golden Jubilee Scholarship in 2010, John von Neumann Scholarship in 2009, Chiang Chen Overseas Exchange Scholarship in 2007. He has received seven awards, such as Hong Kong ICT Awards in 2011 (Best Innovation and Research Bronze Award) and PolyU Micro Fund in 2012 Awards (Entrepreneurship Stream). From 2008 to 2009, he was with IBM Hong Kong on various commercial software projects. He is currently pursuing the M.Phil. degree with The Hong Kong Polytechnic University. His current research interests include mobile social networking and image forensics.

HENRY C. B. CHAN received the B.A. and M.A. degrees from the University of Cambridge, Cambridge, U.K., and the Ph.D. degree from the University of British Columbia, Vancouver, BC, Canada. In August 1998, he joined The Hong Kong Polytechnic University, Hong Kong, where he is currently an Associate Professor with the Department of Computing. His current research interests include networking/communications, Internet technologies, and electronic commerce. He was the Chair of the IEEE Hong Kong Section in 2012, and the Chair of the IEEE Hong Kong Section Computer Society Chapter from 2008 to 2009.

VICTOR C.M.LEUNG(S'75-M'89-SM'97-F'03) is a Professor and holder of the TELUS Mobility Research Chair in Advanced Telecommunications Engineering in the Department of Electrical and Computer Engineering, the University of British Columbia, Vancouver, BC, Canada, where he completed the B.A.Sc. (Hons.) and Ph.D. degrees in 1977 and 1981, respectively. He has been involved in telecommunications research with a focus on wireless networks and mobile systems for more than 30 years, which has resulted in more than 650 journal and conference papers co-authored with his students and collaborators, including several papers that won best paper awards.

He is a Registered Professional Engineer in the Province of British Columbia, Canada. He is a fellow of EIC and CAE. He was a Distinguished Lecturer of the IEEE Communications Society. He has contributed to the editorial boards of many journals, including the IEEE TRANSACTIONS ON COMPUTERS, the IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS, the IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, the IEEE WIRELESS COMMUNICATIONS LETTERS, and the IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS. He has contributed to the organization and technical program committees of numerous conferences. He was the winner of an APEBC Gold Medal in 1977 as the head of the graduating class in the Faculty of Applied Science, the NSERC Postgraduate Scholarship from 1977 to 1981, the IEEE Vancouver Section Centennial Award in 2011, and the UBC Killam Research Prize in 2012.