SoIoT: Toward A User-Centric IoT-Based Service Framework

IN-YOUNG KO and HAN-GYU KO, Korea Advanced Institute of Science and Technology
ANGEL JIMENEZ MOLINA, University of Chile
JUNG-HYUN KWON, Korea Advanced Institute of Science and Technology

An emerging issue in urban computing environments is the seamless selection, composition, and delivery of user-centric services that run over what is known as the Internet of Things (IoT). This challenge is about enabling services actuated by IoT devices to be delivered spontaneously from the perspective of users. To accomplish this goal, we propose the Service-Oriented Internet of Things (SoIoT), a user-centric IoT-based service framework, which integrates services that utilize IoT resources in an urban computing environment. This framework provides a task-oriented computing approach that enables the composition of IoT-based services in a spontaneous manner to accomplish a user task. Tasks can also be recommended to users based on the available IoT resources in an environment and on the contextual knowledge that is represented and managed in social, spatial, and temporal aspects. These tasks are then bound to a set of service instances and performed in a distributed manner. This final composition ensures the Quality of Service (QoS) requirements of the tasks and is assigned to multiple client devices for the efficient utilization of IoT resources. We prove the practicality of our approach by showing a real-case service scenario implemented in our IoT-based test-bed as well as experimental results.

Categories and Subject Descriptors: C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed applications, client/server*

General Terms: Design, Algorithms, Measurement, Performance

Additional Key Words and Phrases: Internet of Things, urban computing, user centricity, task-oriented computing, service composition

ACM Reference Format:
In-Young Ko, Han-Gyu Ko, Angel Jimenez Molina, and Jung-Hyun Kwon. 2016. SoIoT: Toward a user-centric IoT-based service framework. ACM Trans. Internet Technol. 16, 2, Article 8 (April 2016), 21 pages.
DOI: http://dx.doi.org/10.1145/2835492

1. INTRODUCTION

Urban computing is an emergent paradigm that has evolved from ubiquitous computing. Urban-computing environments differ from those considered in ubiquitous computing mainly in terms of scalability. They are denser in terms of the number of users, larger regarding their physical settings, and more diverse in relation to the types of users. The main goal of urban computing is to enable users to access networked services embedded in the computing infrastructure or on the Web anytime and anywhere

This work was supported by the ICT R&D program of the Institute for Information & Communications Technology Promotion (IITP) of the Ministry of Science, ICT and Future Planning (MSIP) of Korea [B0101-15-0334, Development of IoT-based Trustworthy and Smart Home Community Framework].
Authors' addresses: I.-Y. Ko, H.-G. Ko, and J.-H. Kwon, School of Computing, KAIST - Korea Advanced Institute of Science and Technology, 291 Daehak-ro, Yuseong-gu, Daejeon, 34141, Republic of Korea; emails: {iko, kohangyu, arunson}@kaist.ac.kr; A. Jimenez Molina, Department of Industrial Engineering, University of Chile, Republica 701, Of. 31, Santiago, Chile; email: ajimenez@dii.uchile.cl.

DOI: http://dx.doi.org/10.1145/2835492

Fig. 1. Concept of IoT-based service composition and provision.

[Shklovski and Chang 2006]. Recently, we have witnessed an increase in the number of *smart objects* in these environments, such as smartphones, smart TVs, robots, intelligent appliances, public displays, smart vehicles, and traffic sensors. Smart objects can be connected via standard protocols such that they are always accessible through the Internet or an ad-hoc network. These capabilities of smart objects have led to the emergence of a new paradigm known as the *Internet of Things (IoT)*, where diverse configurations of smart objects can be made to undertake user tasks in a collaborative manner [Atzori et al. 2010; Guinard et al. 2010].

For instance, the scenario shown in Figure 1 assumes that there is a user who drives his car downtown to buy a digital camera as his daughter's birthday present. He must then return home by 7 o'clock. Because the streets are crowded, his car's navigation system shows the fastest path by interacting with public cameras and traffic lights. Finding a parking spot on a downtown street is always challenging for him. Fortunately, his smartphone shows a suitable area to park by interacting with public cameras and parking meters that, by working together, locate parking spots. After he parks his car, his smartphone proposes the best stores to find digital cameras. This information is obtained from social network services that contain reviews of camera shops satisfying his preferences as reported by colocated users and/or by users who have visited the venues.

In order to realize this envisioned scenario, it is necessary to address the complexity that emerges from the increasing number of smart objects and from the different levels of abstraction in users' goals and smart-object functionalities. In fact, the mere abundance of smart objects in urban computing environments does not guarantee the effective, proactive support of users. Moreover, this is a challenge due to the overwhelming number of options and limited bandwidth, as well as the many users in a local environment competing with one another for the same set of smart objects. On the other hand, there is a gap between low-level functions of smart objects and high-level user goals, which are usually represented as a series of activities. Therefore, there is a need to recognize the IoT as a configuration of diverse smart objects necessary

to support user activities. The diversity of users' goals, a large amount of smart objects, and the scale of urban-computing environments make the realization of the IoT a challenging and complex issue.

To accomplish this goal, we propose what is termed the Service-Oriented Internet of Things (SoIoT), a user-centric IoT-based service framework that integrates smart objects available on the IoT. To bridge the gap between user goals and IoT functionalities and to deal with the complexity problem, we propose an ontology model with which user goals and environmental contexts can be formally represented and analyzed. We have also developed an approach to identify services that can be provided by utilizing available smart objects and having them composed together to meet user goals. We use a *task-oriented service framework* [Jimenez Molina and Ko 2011a; Huerta-Canepa et al. 2008] to provide user-centric services using IoT-based services in an urban-computing environment. In this framework, user goals are represented as an explicit *task* definition that is a coordination of activities. *Activities* consist of configurations of *abstract services* that can be instantiated by orchestrating available *service instances*, including services that can be actuated through the IoT, composed of smart objects.

These goals require a proper semantic description model of activities and urban-computing environments based on different contextual factors. In order to support user tasks, our framework uses this model to discover feasible activities based on the information of smart objects in the proximity of users as well as the contextual knowledge modeled and represented socially, spatially, and temporally. This is done with a reasoning mechanism that identifies a set of tasks that makes use of the activities, which are then proposed to the users. This final composition ensures the Quality of Service (QoS) requirements of the tasks; it is also assigned to multiple client devices for the efficient utilization of smart objects on the IoT. We demonstrate the effectiveness and practicality of our framework by showing a real-case service scenario implemented in our IoT-based test-bed and through experimental results.

The remainder of this article is organized as follows. In Section 2, we introduce SoIoT, its overall architecture, and its essential technical elements. Section 3 describes a real-case scenario in our test-bed. In Section 4, we explain the experimental results to demonstrate the effectiveness and efficiency of our system. In Section 5, we discuss related works. We present our conclusions and suggestions for future work in Section 6.

2. THE SOIOT FRAMEWORK

2.1. Framework Architecture

In order to realize our research goal of supporting user tasks with an appropriately configured IoT of smart objects available in an urban-computing environment, we identify the following technical requirements: (1) to enable users to select their tasks based on their own perspective; (2) to transform users' task needs into a form that is machine-processable; and (3) to support users' tasks and enhance them via smart objects in the IoT by semantically connecting the gaps among users' needs, tasks, and smart objects. We reflect these requirements in the system architecture of the task-oriented service framework, SoIoT, as shown in Figure 2.

The *mobile client* looks for available smart objects and notifies the *activity selector* via their inputs, outputs, preconditions, and effects. These functions are realized by means of the *smart-object discoverer* and the *IoT description manager*, respectively, which ensures the reliable and flexible detection and broadcasting of smart objects. The *task manager* selects and filters those activities that can utilize the available smart objects considering the contextual information that characterizes the urban-computing environment and the users. The contextual information is provided by the

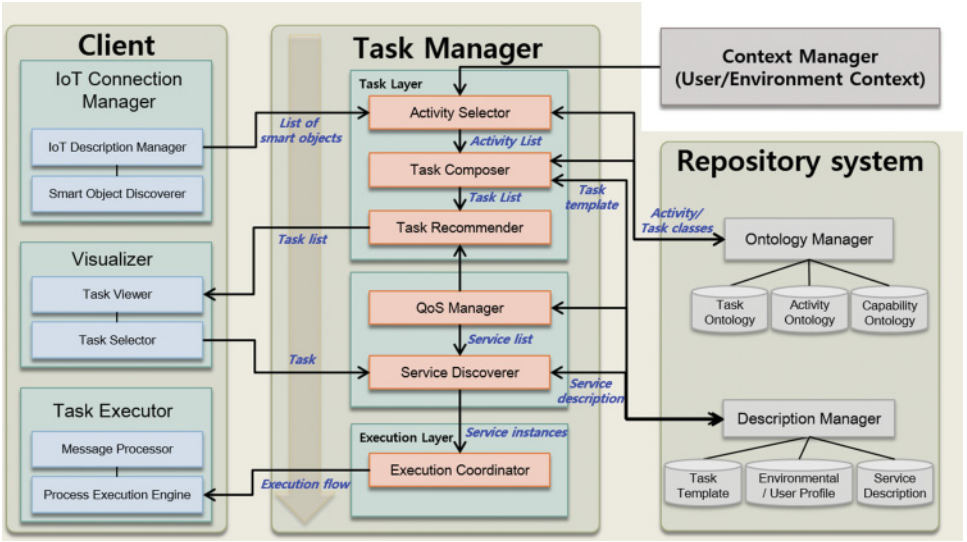


Fig. 2. Overall architecture of the SoIoT framework.

context manager. The *activity selector* carries out the selection and filtering functions using activity ontologies, which will be explained in Section 2.3. Additionally, the task manager identifies task templates from a repository that contains the coordination of filtered activities and ranks, and recommends a task list to users through a mobile client. The former is realized by the *task composer*, the latter by the *task recommender*.

Recommended tasks are presented to the users by the *visualizer*. If the user chooses one of the tasks, the client requests its execution flow from the task manager and actuates the necessary smart objects through the *task executor*. The execution flow is obtained by dynamically binding the abstract services of the activities to concrete, available service instances. The modules in charge of this are the *task broker*, the *service discoverer*, and the *execution coordinator*. We explain the details of these mechanisms in later sections.

The task manager can be run either on a designated server in a local IoT environment or in a cloud-computing environment. The repositories reside in a cloud and manage the ontologies, task/activity templates, and service descriptions. The mobile client runs on users' mobile devices, such as their smartphones. The architecture is extendable in that there can be any number of mobile clients interacting with a task manager, and new task/activity templates and services can be registered in the repositories in the cloud as they are developed to utilize new types of smart objects on the IoT.

2.2. IoT Discovery

The *IoT connection manager* looks for available smart objects around users in a local area. From the task manager, it also receives descriptions of the services that can be provided by utilizing smart objects. The service descriptions are represented in Web Services Description Language (WSDL)[1] and include information about the operations and interface details of the services. We adopted a multicast Domain Name Server (DNS) detection and broadcasting[2] method to discover and register available smart objects on the IoT. When the smart-object discoverer sends a query to the smart objects

[1]http://www.w3.org/TR/wsdl20/.
[2]http://jmdns.sourceforge.net/.

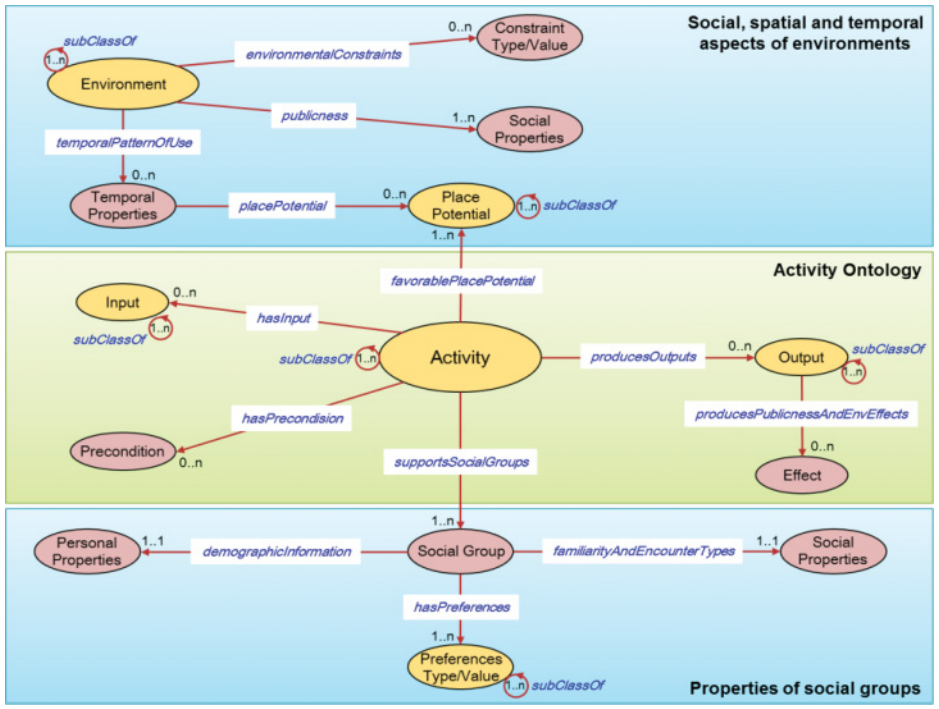


Fig. 3. Ontology model for representing activities, urban-computing environments, and social groups.

in the surrounding area, the smart objects respond with specific information, such as their names and interface descriptions.

Here, we extended the task-selection mechanism of our previous work [Jimenez Molina and Ko 2011a]. This mechanism systematically matches the capabilities of smart objects against the types of services that are appropriate for the activities of user tasks in a local environment, as described in next two sections.

2.3. A Semantic Activity Description Model

Activities are described in terms of the types of factors and properties derived from essential aspects that define the context governing the interactions among users and smart objects in urban-computing environments. The rationale for describing the contextual information, which characterizes both the urban-computing environments and the situations in which users are involved, is based on the work of Kostakos et al. [2009]. They define three aspects of an urban-scale ubiquitous computing infrastructure—the social structure of users' routines, the spatial structure of the infrastructure, and the temporal rhythms of users' behaviors in urban-computing environments.

As shown in the ontology model (Figure 3), the social aspect of an urban-computing environment is described in terms of its level of *publicness* attached to the space. This refers to the publicness spectrum of the public realm (absolutely public, quasi-public, or private places) [Banerjee 2001]. The spatial aspect is composed of *environmental constraints*, as represented by ranges of temperature, noise, humidity, and brightness. The temporal aspect is described by the *temporal pattern* of utilizing a place, which comprises the multiple *place potentials* that the place may have at different times [Kostakos et al. 2009]. The temporal pattern is described by a tuple of properties composed of the season, day of the week, phase of the day, and place potential—

the most likely user behaviors as determined by urban designers. The place-potential ontology represents the situations that can occur in a place, such as presenting, eating, and shopping [McCullough 2001]. The place potential of a place can be regarded as the locational context that needs to be considered to select location-based services.

On the other hand, user types are described in terms of the level of *familiarity* (denoting the strength of the social ties between users, such as familiars, strangers, or familiar strangers [Paulos and Goodman 2004]), and the *social encounter types* engaged in by users—shared interactions users potentially would carry out based on their personal and social characteristics. These can include physical, anonymous virtual, or identity-aware virtual encounters [Jimenez Molina and Ko 2011a; Paulos and Goodman 2004]. A social-group type is described in terms of users' *preferences*, that is, their *demographic information*, such as their age, gender, and occupation, and by other social properties of the users.

To build the activity ontology, we utilized the dataset from the American Time-Use Study[3] (2007 version), which recorded 65,635 user activities. An activity is described in terms of its *inputs*, *outputs*, *preconditions*, and *effects*. Inputs and outputs are described as independent ontologies, arranged by subsumption relationships. Outputs produce *publicness-level effects* and *environmental effects*. Thus, cases in which the activities violate the publicness level of the environment should be avoided. Finally, the environmental effects generated by activities in an environment need to be consistent with the environmental constraints attached to the place. The *favorable place potential* relationship represents the appropriate places in which to perform the activity. In addition, the types of *social groups* that can be supported by an activity can be represented in the activity ontology. Ontologies for the activities, their inputs and outputs, urban-computing environments, place potentials, and social preferences are compiled to define common vocabularies to represent their semantics. The aspects and properties described in this model are utilized by the task-selection mechanism described in the next section.

The ontology data is represented in Web Ontology Language (OWL)[4], which is a standard ontology language for the Web. The semantic reasoning with which to select and compose user tasks is implemented using Jess[5], a popular rule engine for the Java platform.

2.4. Task Selection and Composition

Figure 4 shows the overall process of selecting, instantiating, and executing a task based on the smart objects available in an IoT environment. To support a user's goal, SoIoT selects and recommends a set of feasible unit-tasks. A *unit-task* is a combination of abstract services that can be instantiated to support a user activity. An abstract service defines a set of service capabilities that can be provided by utilizing a set of smart objects. This task-selection mechanism takes place on the server side of the framework and follows a bottom-up view of the task-composition process by initially considering smart objects that are available in the surrounding environment to identify only locally supported tasks. Given these smart objects, locally supported activities are identified and later tailored into tasks based on the task templates in the task repository. A *task template* defines a flow of activities (represented as unit-tasks) that need to be performed to accomplish a user goal. A task template can be reused to accomplish similar goals in different IoT environments. During the composition process, the social, spatial, and temporal aspects in the environment are considered as restrictions when binding

[3]http://www.bls.gov/tus/.
[4]http://www.w3.org/TR/owl-features/.
[5]http://www.jessrules.com/.

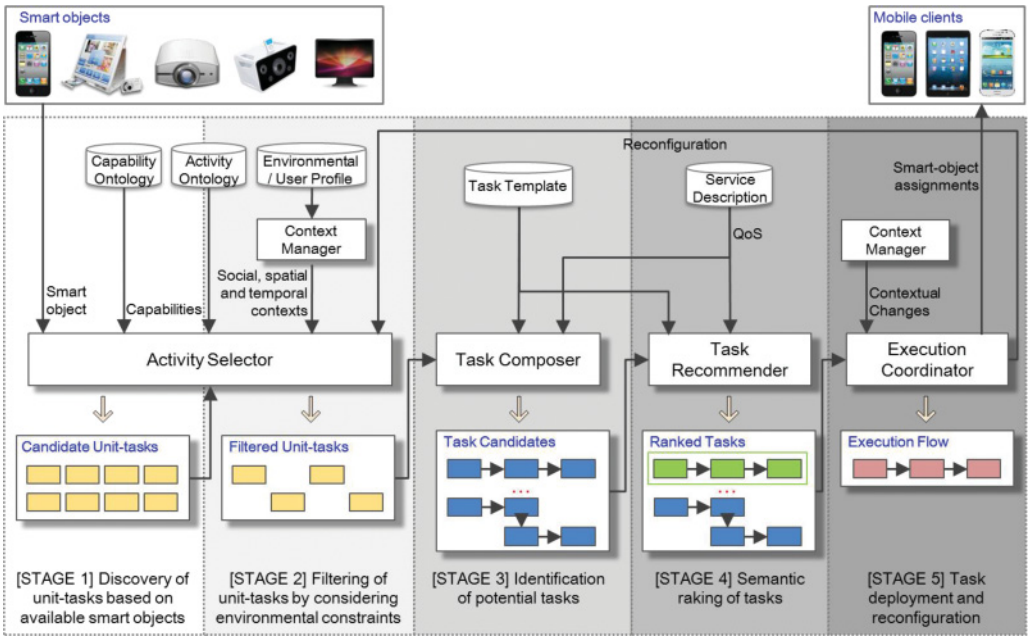


Fig. 4. Smart-object-based task selection and composition process.

activities into unit-tasks. In this way, the resulting tasks can reflect the user's context and requirements.

2.4.1. IoT-based Selection of Feasible Tasks. Activities are the building blocks of user tasks. These activities are identified by detecting smart objects surrounding a user or a group of users and by semantically matching the smart-object functions against the capabilities required by the activities (Stage 1 in Figure 4).

Then, the social, spatial, and temporal contexts are considered to filter the set of feasible activities. These resultant activities reflect the environment and users' contexts, and the social requirements (Stage 2 in Figure 4) [Jimenez Molina and Ko 2011a]. That is, (1) their effects do not violate the environmental constraints or the level of publicness of the urban-computing environment; (2) they are appropriate for the current temporal pattern of using the environment; (3) they are appropriate to support the social encounters defined by the familiarity types of users; and (4) they satisfy the users' demographic information and preferences.

The detailed method used to identify interoperable unit-tasks considering the social, spatial, and temporal contexts of an IoT environment is as follows. Let U_A and U_B be two unit-tasks, with (I_A, O_A) and (I_B, O_B) denoting their inputs and outputs, respectively.

Even when the outputs of U_B are semantically similar to the inputs or outputs of U_A, the social effect of U_B may violate the "publicness authorization level" associated with the inputs and outputs of U_A. Therefore, to verify the interoperability of U_A and U_B in terms of social effects, it is necessary to identify and analyze the publicness associated with the inputs and outputs of the unit-tasks. Let $I_A = \{i_{A_1}, \ldots, i_{A_n}\}$, $O_A = \{o_{A_1}, \ldots, o_{A_m}\}$, and $O_B = \{o_{B_1}, \ldots, o_{B_p}\}$. In addition, let $sm(I_A, O_B)$ be a function that checks the semantic similarity between the inputs of U_A and the outputs of U_B. This similarity is evaluated using the ontological semantic distance measure developed in our previous work [Jimenez Molina et al. 2009]. The semantically similar inputs of

U_A and outputs of U_B are represented as I_A^* and O_B^*, respectively, where $I_A \supseteq I_A^* = \{i_{A_1}^*, \ldots, i_{A_{n1}}^*\}$, and $O_B \supseteq O_B^* = \{o_{B_1}^*, \ldots, o_{B_{n1}}^*\}, (n_1 \leq n) \wedge (n_1 \leq p)$, with $i_{A_j}^* \coloneqq o_{B_j}^* (\forall j = 1, \ldots, n_1)$ denoting the semantic similarity between the input and output parameters. Inputs and outputs of a unit-task that have a "private" publicness authorization level can be exposed in an urban-computing environment only when there is a unit task on which the outputs have a "private" publicness level effect. Otherwise, the social effects would violate the authorization level of the first unit-task. Another example is that a "quasi-public" publicness authorization level cannot be exposed in a public environment.

To consider the spatial context, the interoperability measurement has been extended to consider the coexistence of tasks in the same place. If there are coexisting tasks, their effects are aggregated and then checked as to whether or not the aggregated effects violate the environmental constraints. In the temporal aspect, the consistency of "temporal availabilities" between the outputs of U_A and the inputs of U_B is checked. For example, a unit-task that requires an input with a "continuous-processing frequency" can interoperate with another unit-task for which outputs are generated continuously. However, the first unit-task cannot interoperate with a task that produces outputs with a "single-processing frequency."

2.4.2. Task Composition. In this phase, a set of feasible unit-tasks are combined that can be performed in the current user environment to support user activities for a task. This is done by semantically matching the filtered activities against the activities coordinated in the task templates that are available in the repository [Jimenez Molina and Ko 2011a] (Stage 3 in Figure 4). This stage produces a set of tasks that are feasible in the given environment with regard to the utilization of the existing smart objects while taking into account the contextual information.

In order to compose and provide services in a spontaneous manner, it is essential to find the set of services that meet users' requirements in terms of QoS in a reasonable response time. In a task template, the QoS requirements of the task are represented as the weights of the quality attributes of the services for each activity. When a user chooses a task template to instantiate in an environment, the user needs to check the QoS requirements of the task as a service-level agreement. For example, a user may want to watch a movie in full-HD quality in a dark (less than 70lux) and quiet (less than 100db) environment for the "Watching a Movie" task; the user needs to check whether a task template meets such requirements by accessing the description of the task template.

However, if there are many candidate services that are available in an environment, selecting appropriate services that meet users' preferences (QoS requirements) for service compositions (user tasks) can be complex [Benouaret et al 2013]. Specifically, during the task-instantiation process, finding a set of services that meet the multi-aspect QoS requirements of a task is a well-known NP-hard problem [Li et al. 2010]. Specifically, in an urban-computing environment, there may be a number of services that provide a similar level of functionality at various quality levels in different aspects. Therefore, there must be an efficient means of creating a service composition for a task while maximizing the integrated quality of the selected services. For this purpose, we developed an improved QoS constraint decomposition approach that adaptively samples services and finds the best set of services according to given QoS constraints, as described in Cho et al. [2012].

The adaptive QoS constraint decomposition approach estimates the most probable constraint values for the quality attributes of each abstract service in a task. The approach then identifies probable groups of candidate services in which services can be sampled for each abstract service. *Candidate services* are the service instances that

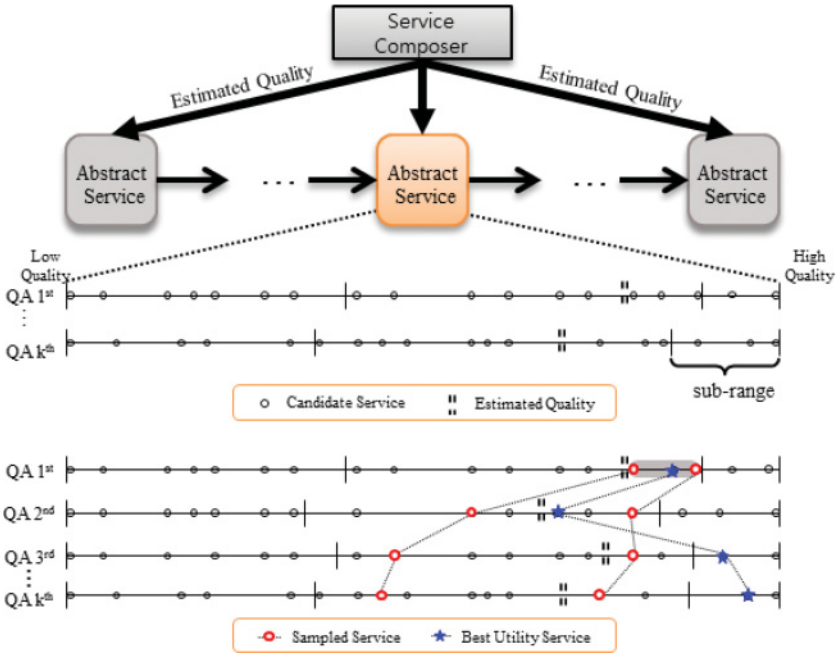


Fig. 5. Adaptive QoS constraint decomposition.

can be used to instantiate an abstract service of a user task. The quality ranges of candidate services are adaptively divided into subranges using the "estimated quality." In Figure 5, a value range between two vertical bars defines a subrange. As shown in the figure, the average values (represented as small white circles) of the quality attributes for all candidate services are calculated for each abstract service. The constraints of abstract services for each quality attribute are then calculated, and the global constraints are distributed accordingly. A distributed constraint value is referred to as an *estimated quality*, which becomes the pivot point at which to divide the entire quality range into an equal number of subranges. In each subrange, a representative service is randomly sampled.

The estimated quality is calculated as follows: Equation (1) calculates the average QoS value (AQV) of a quality attribute of an abstract service. Equation (2) then estimates the local constraints of the quality attributes (Estimated Quality, EQ) for each abstract service.

$$AQV_k^h = \frac{\sum_{i=1}^{l} q_k^i}{l} \tag{1}$$

$$EQ_k^h = \frac{AQV_k^h}{\sum_{i=1}^{n} AQV_k^i} \times GC_k \tag{2}$$

Here, q_k^i is the value of the k^{th} quality attribute of the i^{th} service, l is the number of candidate services, h is the number of QoS constraints ($1 \leq h \leq m$), m is the total number of QoS constraints, k is the number of abstract services ($1 \leq k \leq n$), n is the total number of abstract services and AQV_k^j is the average value of the k^{th} quality attribute in the j^{th} abstract service of the task. GC_k is the global constraint of the k^{th} quality attribute.

The quality ranges are divided based on the estimated quality. Given that services for which the quality is better than the estimated quality are more likely to lead to successful compositions, the subranges must be narrower for a quality attribute that has harder constraints. The service that generates the highest utility value is then selected for each abstract service, while the QoS value of a subrange represents the local constraints. The *utility of a service* is the integrated quality of the service for a task, which indicates the degree to which it meets the quality requirements of the task. The utility of a service is computed as follows:

$$Util_{service}(s_i) = \sum_{k=1}^{r} \frac{Q_{max}(k) - q_k^i}{Q_{max}(k) - Q_{min}(k)} \times w_k \tag{3}$$

In this equation, $Q_{max}(k)$ is the maximum QoS value of the candidate service instances for the k^{th} quality attribute, $Q_{min}(k)$ is the minimum QoS value of the candidate services for the k^{th} quality attribute, q_k^i is the value of the k^{th} quality attribute of the service s_i, w_k is the weight value of the k^{th} quality attribute, and r is the number of quality attributes.

The shaded bar in Figure 5 represents the appropriate QoS value range in the first quality aspect for the abstract service. In other words, the services in the shaded bar become the candidates for local optimization. Finally, the best set of services (represented by the stars in Figure 5) is found by inspecting the utility value of all combinations of service candidates. The set of services that generates the highest utility value is determined by means of integer programming.

2.4.3. Task Ranking and Recommendation. The composed tasks are ranked based on the values obtained by a utility function (Stage 4 in Figure 4). The *utility of a service composite* (a task candidate shown in Figure 4) is calculated by aggregating the utility values of the unit-tasks included in the composite. The utility of a unit-task can then be calculated by aggregating the QoS utility values of the service instances that are selected for the unit-task during the previous stage. The following equations describe the process by which the utility levels of a unit-task and a service composite are measured, respectively.

$$Util_{unit-task}(ut_k) = \sum_{i=1}^{n} \sum_{j=1}^{l} Util_{service}(s_{i,j})^* x_{i,j} \tag{4}$$

$$Util_{task} = \sum_{k=1}^{m} Util_{unit-task}(ut_k) \tag{5}$$

In these equations, n denotes the number of abstract services and l is the number of sampled service instances for the abstract services. ut_k represents the k^{th} unit-task for a given task. $s_{i,j}$ represents the j^{th} sampled service instance for the i^{th} abstract service of a unit-task. $x_{i,j}$ indicates whether or not the candidate service instance is selected to instantiate the abstract service.

The degree of semantic interoperability between unit-tasks (see Section 2.4.1) is also considered while calculating the utility value of a service composite. Finally, this ranked list of tasks is presented to the user through a mobile client such that the user can decide which one most suitably matches one's goals.

2.5. Task Execution

The final step takes place after the user has chosen a task to perform (Stage 5 in Figure 4). While the task is inferred in a bottom-up manner, its deployment is

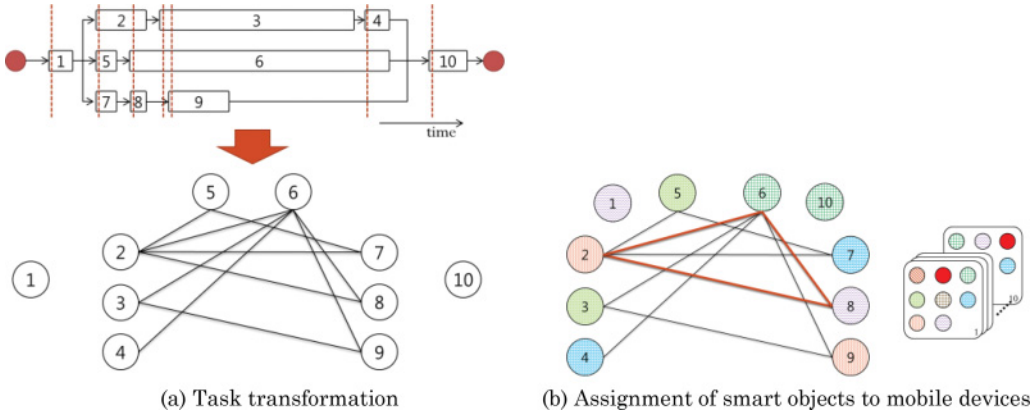


Fig. 6. Task transformation and smart-object assignment to mobile devices.

determined using a top-down approach. In other words, the service discovery engine identifies the service instances that can utilize the smart object operations to realize the abstract services of the activities in the task. These service instances that are bound to the smart objects are orchestrated into an execution flow by the execution coordinator in accordance with the coordination logic of the activities in the task. We use Business Process Execution Language (BPEL)[6] to describe such an orchestration.

The BPEL engine in the task executor of the mobile client receives the execution flow and coordinates the execution of the services by means of the message engine. This module in the mobile client manages the invocation of smart objects. This is done by sending Simple Object Access Protocol (SOAP)[7] messages via the HTTP protocol to the smart objects.

The services of a user task need to interact with each other to perform the task. The smart objects that are utilized by the services need to exchange data accordingly. Therefore, in an urban-scale IoT environment, it is inefficient to coordinate the services of a user task in a centralized manner. SoIoT provides a means of utilizing users' mobile devices to make direct connections to the necessary smart objects and to coordinate the services in a distributed manner for a user task. However, assigning services to multiple mobile devices that are associated with multiple smart objects in an IoT environment is a multifactor optimization problem, which is known as an NP-hard problem in which the optimum cannot be found in polynomial time. Therefore, we developed a heuristic approach that makes the most efficient service assignment to mobile devices using a graph-coloring algorithm [Laurent and Hao 2009]. This approach allocates the services required for a given task to the most appropriate mobile devices that are associated with the necessary smart objects. It then detects contextual changes and triggers a reconfiguration of the task, as described in Choi et al. [2013].

With a graph-coloring algorithm, the approach distributes a task to mobile devices in an efficient manner, as follows:

1) *Task transformation (Figure 6(a))*: The first step is to transform a user task into an undirected graph $G = (V, E)$, where:
V is the vertex set, which is composed of the services, S, of the user task, T; and E is the edge set, which is composed of the pairs of services that need to be performed

[6]https://www.oasis-open.org/committees/wsbpel/.
[7]http://www.w3.org/TR/soap12-part1/.

Table I. Smart Objects and Their Operations in the Test-bed Environment

Smart Object	Operations
Projector	Power On/Off, Input URL, Adjust Lamp Up/Down, Adjust Lamp Left/Right, Adjust Lamp Brightness, Adjust Lamp Contrast
Air conditioner	Power On/Off, Set Temperature, Set Wind Power
Light	Power On/Off, Brightness Setting
Smart TV	Power On/Off, Set Volume, Set Display Mode, Set Channel, Input URL
Smart board	Up, Down, Stop
Smart object	Operations
Print	Power On/Off, Input URL, Cancel

in parallel; that is, the end time of a service, S_i, should be later than the starting time of another service, S_{i+1}.

2) *Computing a maximum clique*: From the transformed graph, the maximum-size clique is found using a randomized heuristic algorithm.
3) *Matching the best mobile devices*: Find the best set of client mobile devices for the maximum clique using integer programming. (The different colors (shadings) shown in Figure 6(b) represent different mobile devices).
4) *Sorting the unassigned services*: Sort the remaining services in descending order based on the number of adjacent services.
5) *Assigning smart objects (Figure 6(b))*: Assign valid and high-utility smart objects to services in the order used in Step 4.
6) *Finding better smart objects*: Improve the assignments by applying a greedy algorithm iteratively.

If the service assignment to the client mobile devices fails, SoIoT backs off to the second stage of the composition process (Figure 4) and reconfigures the service composite to utilize a different set of smart objects in the IoT environment.

3. A USER-CENTRIC SERVICE COMPOSITION AND DELIVERY SCENARIO

In this section, we demonstrate how the SoIoT framework satisfies user-centricity requirements using a scenario based on our test-bed. When a user enters the test-bed, the user's mobile device (which has the SoIoT client) dynamically finds the smart objects available in the room. The task manager on the server side then extracts feasible activities based on the available local smart objects, after which it recommends feasible tasks from the extracted activities. The recommended tasks that reflect the high-level goal of the user are shown on the user's mobile device. When the user chooses a particular task, abstract services for the activities of the task are bound to service instances. During the process of activity instantiation, the client mobile device detects and controls smart objects, enabling service instances to utilize the smart objects properly.

3.1. Environmental Settings

Our IoT test-bed contains a number of smart objects, such as those shown in Table I. We prepared our test-bed by implementing an agent for each smart object. Agents are developed in Java and deployed on a Beagle Board[8]. An agent can receive SOAP messages from clients and execute functions on physical devices. However, each smart object provides a different type of control interface. Therefore, an adaptor is required to connect the interfaces of an agent and a smart object (see the architecture of the smart object shown in Figure 7).

[8]http://www.beagleboard.org/.

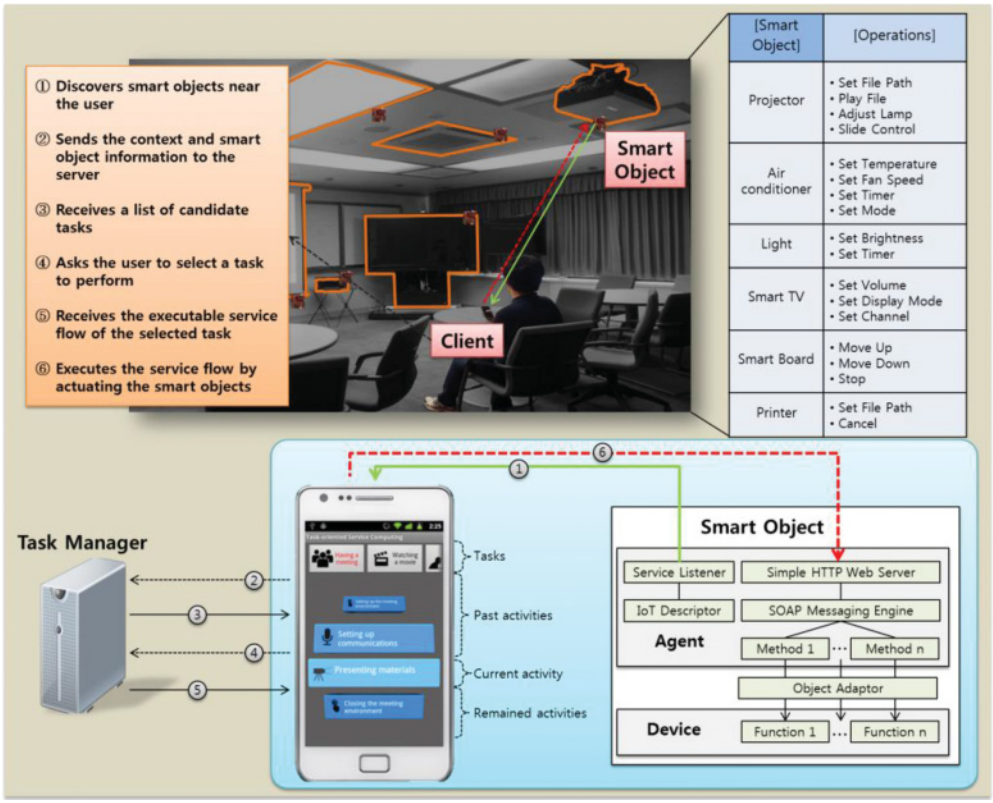


Fig. 7. Running the service scenario "have a meeting" in the test-bed environment.

For instance, in order to set the temperature of an air conditioner, a client sends a SOAP message to an agent via the HTTP protocol. The message contains an operation for setting the preferred temperature of the air conditioner. The agent extracts the operation name and the preferred temperature through the SOAP message engine. Next, a signal is generated using a method that corresponds to the operation. Finally, an object adaptor operates the function of the smart object based on the signal.

3.2. "Have A Meeting" Scenario

1) Mary is a visiting scholar in our department this fall. This afternoon, she needs to give her introductory research presentation to the members of the department. Previously, she installed the client of the task-oriented framework on her smartphone. As she enters the seminar room, her smartphone multicasts a greeting message to all of the smart objects in the room. As a reply, each smart object returns its IP address to the network and descriptions of its operations. Figure 7 shows some of the detected smart objects (projector, air conditioner, light control, smart TV, smart board, printer, smart curtains) and their operations.
2) Her smartphone sends the functionality and interface information of each of the smart objects' operations to the task manager in a room server. For instance, set temperature, set brightness, and set volume operations define the input values for the operations in charge of executing their respective smart-object functionality. In turn, the executing operations produce cool air, luminosity, and an audio stream as the respective outputs of the corresponding smart objects. In addition, the effects

produced by these outputs in the seminar room can be defined as a decreased temperature, greater brightness (in candelas), and by the noise level (in decibels) of the room.

3) With the functionality and interface information of smart objects, the activity selector identifies activities such as (1) "setting up the meeting environment," in charge of turning on and executing the lights, the projector, and the air conditioner, outputs of which suitably match the smart objects; (2) "selecting and playing a movie," for which the audio stream matches the output of a smart TV; (3) "opening curtains" to make the room brighter, for which the luminosity output is consistent with the "set brightness" operation; (4) "presenting a PowerPoint file," for which the presentation output matches that of the "play file" operation of the projector, and (5) "printing a presentation file," which is consistent with the output of the printer operations. These are only examples of the types of feasible activities that can be selected based on the smart objects installed in our test-bed.

   However, the activity selector filters out activities such as selecting and playing a movie, and opening the curtains to make the room brighter from the feasible set. The former is filtered out for the following reasons: (1) the temporal pattern of using the seminar room states that during the afternoon hours on weekdays during the spring or fall semester, the place potential is set to academic tasks (during summer vacations, during the nighttime, or during weekends, the place potential may be set to leisure tasks, allowing professors and students to use the room for such purposes); (2) the publicness-level effect ("private") of this activity is not consistent with the publicness level ("quasi-public") of the seminar room. The latter is filtered out because the environmental constraints of our test-bed state that the luminosity of the room needs to be less than 120 candelas, which is violated if daylight is added to the brightness of the internal lights.

   Given the set of resultant activities, the activity selector identifies tasks such as having a meeting or watching an online lecture as most appropriate for the situation in this scenario. The task template of the former is composed of the following activities: setting up the meeting environment, setting up the communication, presenting the material, and closing the meeting environment. Nevertheless, these two tasks are simply illustrative cases of the multiple tasks that our task manager is able to match in the repository.

4) The smartphone in Figure 7 shows the user interface when Mary selects the "having a meeting" task. The list at the top of the interface is the recommended task list sent by the task manager. It contains the selected tasks described earlier. During the execution process, the current activity is located in the center and is highlighted (see the "presenting material" activity shown on the smartphone in Figure 7). The user interface shows each activity as a visual icon. Activities already executed fade out and slide up for the subsequent activities (such as "closing the meeting environment," as shown on the smartphone), which gradually appear at the bottom.

5) The task manager retrieves the task template described in BPEL of the "having a meeting" task, after which it binds the smart objects shown in Figure 7 (except for the printer).

6) The smartphone invokes smart objects using the SOAP messages generated by the BPEL engine. In this case, the air conditioner and lights are turned on as the initial activities, and the setting up of the meeting environment commences. When the environmental setup is finished, the client waits for user input pertaining to the selection of presentation materials. As Mary chooses the file for the presentation, the file is transferred to the computer and displayed on the screen by the projector. While presenting the slides, Mary can remotely control the presentation on her

smartphone. When the meeting is over, Mary notifies her smartphone so that the client can command the projector and computer to be powered off.

This scenario is computationally complex in that the five candidate activities need to be matched against a large number of user activities (according to the American Time-Use Study dataset) based on the six smart objects that are available in the room. In addition, filtering the candidate activities by considering various factors in the social, spatial, and temporal aspects of the environment is not a trivial task, as explained in the third step of the scenario.

4. EVALUATIONS OF THE QOS-AWARE SERVICE COMPOSITION AND DISTRIBUTED COORDINATION

We evaluate the practicality of our approach in terms of its efficiency in bridging the gap between user needs and available resources. As stated in the introduction section, this refers to the capability of SoIoT to guarantee the QoS of service composites and to utilize smart objects on the IoT efficiently. For a quantitative evaluation, we conducted two different experiments. The first was intended to test the assurance of QoS as provided by our adaptive QoS-aware service selection approach. The second experiment aimed to test the efficiency of service allocations by means of our distributed coordination approach. Both experiments were conducted separately in order to isolate any cross-effect between them.

4.1. Evaluation Settings

We conducted a number of simulations to test our approach of dynamically binding abstract services to a set of generated Web services. The Web service dataset was generated based on the quality attributes in the Quality of Web Service (QWS) dataset, as measured by commercial benchmark tools [Al-Masri and Mahmoud 2008]. The QWS dataset includes QoS values of 5,000 Web services, with a number of abstract services that have more than 50 different service instances. The global QoS constraints (task-level QoS) were considered as given by the users. We tested our approach while varying the number of available Web services for the composition, the hardness of the global constraints, and the time performance. Mixed-integer programming (lp_solve, Java version 5.5.2) is used for the evaluation [Nemhauser and Wolsey 1999]. The metrics for the evaluation were (i) the ratio of successful compositions, (ii) the computation time performance, and (iii) the overall QoS of the composite and its closeness to the optimal solution.

The second part of the evaluation was done to confirm the efficiency and effectiveness of the distributed service coordination approach. These are measured as the optimality of the coordinated set of services in terms of the overall utility score and as the successfulness of avoiding service assignments that conflict with clients' mobile devices. Experiments were done by incrementally increasing the number of collaborating mobile devices and smart objects. Access permissions to smart objects were granted to clients at a 30% probability. In addition, services were generated in seven different categories in a probability distribution that was obtained from the Intel dataset, which contains information on 263,612 user sessions for 136 smartphone users [Paulos and Goodman 2004]. We performed the experiments 1000 times by generating different datasets of a user task (service composite), smart objects, and client devices.

The experiments are conducted using Eclipse Java EE Indigo SR2 and JDK 1.7.0_03. The experimental platform runs Windows 7 with an Intel Core i7-2600 CPU operating at 3.40GHz with 8.00GB of RAM.

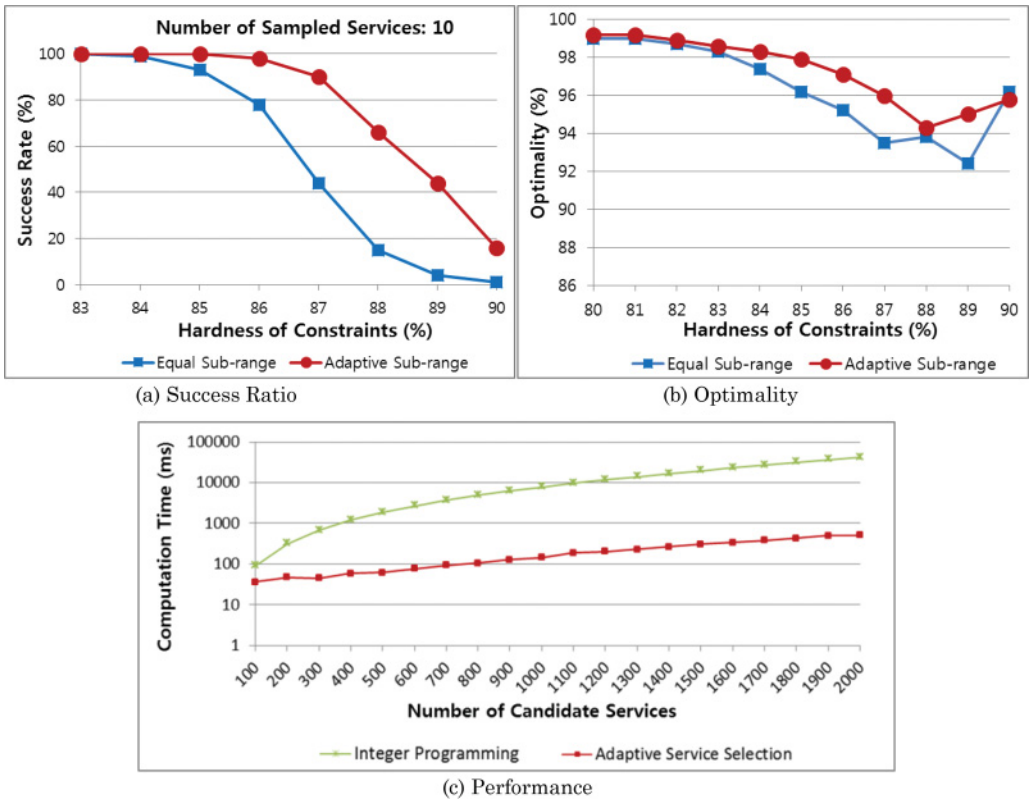


Fig. 8. Evaluation results of the adaptive QoS constraint decomposition approach.

4.2. Effectiveness of the Adaptive QoS Constraint Decomposition Approach

Success Ratio: Figure 8(a) shows the number of successful compositions as the hardness of the constraints changes. We compare our approach of considering constraints by dividing the quality range into subranges while adapting to the hardness of the constraints against the traditional approach of considering the constraints as a whole. The number of sampled services is fixed at 10 and the global constraints are initially determined by the maximum quality values for each abstract service. In the simulation, we observe whether the composite services meet the global constraints as the hardness of the constraints decreases. In addition, we run the simulation 1000 times for each hardness level of constraints. The result shows that our technique of considering constraints adaptively achieves a greater number of successful compositions as compared to sampling services from equally divided QoS subranges. In addition, we found that the number of successful compositions drops as the hardness of the constraints increases because it becomes more difficult to find a set of services that meet the global constraints.

Optimality: Figure 8(b) represents the optimality of the service compositions between the adaptive service selection and the nonadaptive service selection approaches. The optimality is determined by the ratio of utility values generated by using each of the approaches to those generated with the optimal method, which uses integer programming without sampling. For example, regarding the execution time of a given task, if the elapsed time of a service composition that is found by using integer programming is 100ms, and the elapsed time of the service composition that is made by SoIoT is

90ms, then we can say that the optimality of the service composition is 100%. However, if SoIoT can only find candidate services by which the task can be performed in 120ms, the optimality of the service composition is 83.3%. The experimental result shows that both approaches maintain optimality levels that exceed 90% regardless of the hardness of the QoS constraints because both approaches select services that have the highest utility in each subrange. However, the results of our approach are closer to the optimal values. In addition, when there are hard constraints, the standard error of the adaptive approach is much smaller than in the other case.

Performance: Figure 8(c) shows the changes in computation time when finding appropriate services as the number of candidate services increases. We compare our approach, adaptive service selection, against a method that uses integer programming without sampling. In the experiments, the number of candidate services ranges from 100 to 2000 while the number of abstract services in the task is fixed. The result shows that, when the integer programming algorithm is applied, the service composition time increases exponentially. Our approach, however, requires less computation time because the adaptive service selection process uses integer programming only with smaller samples and not for all candidate services. Therefore, even if the number of candidate services grows, it does not affect the computation time of our approach directly because its time complexity is related to the number of samples and the number of abstract services. This makes our framework scalable to a large number of services in urban-computing environments.

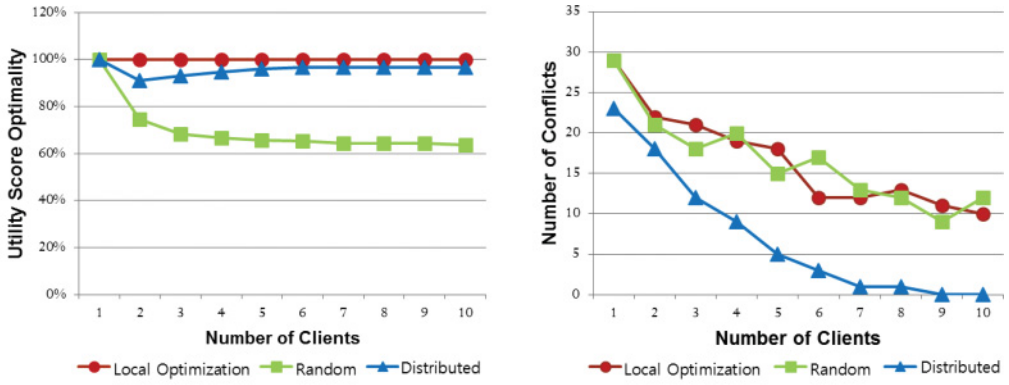
4.3. Evaluation of the Distributed Coordination of IoT-based Services

We aimed to test our service coordination approach by simulating a service composition distribution among simulated devices. The metrics considered here were the overall utility and the number of conflicts in the service distribution process. The utility score represents the efficiency of performing a user task across multiple mobile devices; it is calculated based on the communication efficiency between smart objects that are connected to the mobile devices and the time performance when executing the services. In addition, a conflict occurs when the number of concurrent services assigned to the same client exceeds one.

Test cases are created by increasing the number of client mobile devices involved in the distributed service coordination from one to ten. For each test case, 1000 experiments were performed. We used a random-coloring algorithm, which assigns services to mobile devices in a randomized manner, as the baseline approach for comparison purposes in this evaluation.

Overall Utility: The efficiency of the execution of a task is measured as the total utility score, as calculated based on the performance of the client mobile devices when running the services of a user task. As shown in Figure 9(a), the proposed approach outperforms the random-coloring algorithm by a significant margin when there are two or more clients. With one client, the total utility scores are nearly identical, as the same client is assigned to all services. In Figure 9(a), which describes the total utility scores of the random-coloring and distributed-coordination approaches relative to the local-optimization approach, we find that the distributed-coordination approach shows a result nearly as good as that of the local-optimization approach.

Conflict Avoidance: As shown in Figure 9(b), the distributed-service-coordination approach produced the lowest number of conflicts among the three approaches. When the number of collaborating clients exceeds five, local optimization and the random algorithm produce more than ten conflicts. The distributed-service-coordination approach, on the other hand, results in nearly zero conflicts when the number of collaborating clients exceeds seven. This result implies that, as the number of clients that participate in a task execution increases, the performance will not be degraded unless



(a) Utility of service distribution over mobile devices

(b) Conflict avoidance

Fig. 9. Evaluation results of the distributed-service coordination approach.

the allocation algorithm considers the flow structure of the task. As the number of collaborating clients is increased, all three approaches generate fewer conflicts (although there are some fluctuations when using the random approach). As shown in the figure, the distributed-service-coordination approach is much more effective when used to reduce the number of conflicts with fewer client mobile devices. In this simulation, when the number of client mobile devices reaches nine, the distributed service coordination approach starts to generate no conflicts, whereas other approaches continue to generate a significant number of conflicts.

5. RELATED WORKS

5.1. Task-Oriented Frameworks

Many researchers have attempted to identify user activities in an accurate and efficient manner by taking into account the contextual information of ubiquitous computing environments. Gaia [Roman et al. 2002] is a user-centric, location-aware and event-driven framework for building ubiquitous computing applications. Gaia provides a mechanism for building new applications and for making use of a predefined application model. This model consists of an ontology of generic applications that is used to generate customized applications in an active space. It allows applications to change their structure dynamically by means of interspace user migration. One of the limitations of this framework involves the static binding of user information within predefined and available applications in a ubiquitous computing environment. In addition, Gaia uses an approach in which services are defined based on a system perspective rather than on a user perspective.

Aura [Sousa and Garlan 2002] adopts a task-oriented computing approach for user-centric service provision in ubiquitous computing environments. In Aura, user-centricity is realized by describing user goals in coarse-grained tasks. As stated by Wang and Garlan [2000], "users can interact with the system in terms of high-level tasks instead of individual services or applications." Tasks are dynamically mapped to virtual services, which are, in turn, associated with actual devices or applications while maximizing the degree of user utility. Although Aura is a typical task-oriented framework, it assumes certain knowledge pertaining to the relationships between virtual services and concrete service providers. This limitation implies static binding between different levels of abstraction and granularity.

The IST Amigo framework [Mokhtar et al. 2005] is an example of a task-oriented service provision in the ambient systems domain. Ambient systems are of interest, as

they pertain to the present study with regard to how they share a number of identical features with ubiquitous computing environments. Both attempt to support user applications transparently with user goals in an environment and to meet the requirement of user-centricity. A drawback of the IST Amigo framework is that workflows are directly mapped to available services without any mediation process and without abstract layers. This leads to a lack of flexibility and prevents the reusability of compound services. That is, applications cannot be developed during runtime operations by reusing the predefined compositions of abstract services that are appropriate for specific workflows. Additionally, applications cannot easily be reconfigured by varying the composition of abstract services such that they better represent user goals.

5.2. IoT-Based Service Provision

There are also studies that attempt to connect services and real-world objects. To connect services and smart objects, for example, SECE [Boyaci et al. 2010] uses gateways to connect devices to services, while SOCRADES [Guinard et al. 2010] uses WS-* standards (BPEL, WSDL, and SOAP) to integrate services and smart objects.

Sense Everything, Control Everything (SECE) is an event-driven system that uses an English-like language. It allows a user to create new services that connect/combine communication and social devices in the real world. SECE retrieves all information from various sources to personalize the new services. In the communication between smart objects and SECE, SECE uses gateways and servers to control and connect sensors and actuators. However, this approach has several drawbacks. Using a nonstandard system can cause scalability or interoperability problems between services and the system. Our system uses Web Services standards (BPEL, WSDL, and SOAP), which are scalable and provide an easy way to connect a new smart object. Using WS-* standards offers several advantages in terms of flexibility and interoperability.

The Service-Oriented Cross-layer Infrastructure for Distributed Smart Embedded devices (SOCRADES) is a service integration system that uses Web Service standards to combine services and smart objects. In addition to the use of Web Service standards (the Device Profile Web Service), it uses a Web-oriented pattern (RESTful API) to connect a service to a smart object. The key contribution of this work is the Real-World Service Discovery and Provisioning Process (RSDPP). RSDPP helps developers discover services for which the output can be utilized in a composite service, which is built. When a new device joins a network, it multicasts a message to other devices in the network. Our approach, which identifies services based on available smart objects, is an extension of this approach for mobile and urban-computing environments. In our approach, mobile clients can discover services and orchestrate service compositions. The SOCRADES approach works by composing individual services rather than by providing high-level abstractions of user tasks.

6. CONCLUSIONS

In order to provide user-centric services utilizing IoT resources in urban-computing environments, we introduced a task-oriented service framework called SoIoT. We hold that smart objects, which are the building blocks of the IoT, can create synergy between computationally augmented objects and services in general, such as Web services. In this article, we presented how the proposed framework can utilize smart objects based on a semantic representation model of tasks and a smart object description model. In addition, a runtime infrastructure provides a spontaneous composition of services that utilize smart objects in an environment according to a task template.

The major contribution of this work is in its design of a task-oriented service framework in which smart objects on the IoT can be utilized from the perspective of users. Through experiments with our test-bed, we learned about essential issues and how to

overcome the technical difficulties associated with incorporating smart objects on the IoT for the provisioning of user-centric services. The first lesson is that the mere incorporation of the mediating process between user tasks and smart objects is not capable of demonstrating practicality to support users. It is necessary to consider the semantics of user activities, urban-computing environments, and user types. This required us to undertake exhaustive work to populate various ontologies based on our description model. As a result, we realized the effectiveness of our framework to bridge the gap between users' needs, tasks, and smart objects. In fact, the selection of meaningful tasks for a user in accordance with the situation in which the user is involved confirms the appropriateness of combining a bottom-up approach with a top-down approach. In addition, the social, spatial, and temporal contexts are shown to be effective to filter out those activities that do not contribute to supporting users' needs.

In future research, we will investigate methods that can be used to reduce the human effort required to populate the ontologies based on the proposed model (described in Section 2.3) by adopting state-of-the-art technologies for the automatic generation of ontologies. Specifically, we will focus on overcoming the technical challenges pertaining to ontology creation, such as the bottleneck problem of manual knowledge crafting, and the issues of noise, authority, and validity in Web data for ontology learning [Wong et al. 2012]. In addition, we will extend the QoS-based service composition approach to maintain reasonable optimality and successfully provide composite services. Because there can be numerous smart objects that provide the same or similar functionalities, it is necessary to resolve the problem of how the service provision framework finds the most appropriate functionality within the given QoS constraints. Last, we will include the cognitive resource-aware approach described in our earlier research [Jimenez Molina and Ko 2011b] to reconfigure the service coordination step in an efficient and user-friendly manner.

REFERENCES

Eyhab Al-Masri and Qusay H. Mahmoud. 2008. Investigating web services on the World Wide Web. In *Proceedings of the 17th International Conference on World Wide Web*. 795–804.

Luigi Atzori, Antonio Iera, and Giacomo Morabito. 2010. The Internet of Things: A survey. *Computer Networks* 54, 15, 2787–2805. DOI:10.1016/j.comnet.2010.05.010

Tridib Banerjee. 2001. The future of public space: Beyond invented streets and reinvented places. *APA Journal* 67, 1, 9–24. DOI:10.1080/01944360108976352

Karim Benouaret, Djamal Benslimane, Allel Hadjali, Mahmoud Barhamgi, Zakaria Maamar, and Quan Z. Sheng. 2013. Web service compositions with fuzzy preferences: A graded dominance relationship-based approach. *ACM Transactions on Internet Technology* 13, 4, 12. DOI:10.1145/2576231

Omer Boyaci, Victoria B. Martinez, and Henning Schulzrinne. 2010. Bridging communications and the physical world: Sense everything, control everything. *IEEE Internet Computing* 2, 16, 35–43.

Jae-Hyun Cho, Jang-Ho Choi, Han-Gyu Ko, and In-Young Ko. 2012. An adaptive quality level selection method for efficient qos-aware service composition. In *Proceedings of 2012 IEEE 36th International Conference on Computer Software and Applications Workshops*, *IEEE*. Izmir, Turkey, 20–25. DOI:10.1109/COMPSACW.2012.14

Jang-Ho Choi, Jae-Hyun Cho, Han-Gyu Ko, and In-Young Ko. 2013. Distributed coordination of IoT-based services by using a graph coloring algorithm. In *Proceedings of 2013 IEEE 37th Annual Computer Software and Applications Conference*, *IEEE*. Kyoto, Japan, 399–404. DOI:10.1109/COMPSAC.2013.67

Dominique Guinard, Vlad Trifa, Stamatis Karnouskos, Patrik Spiess, and Domnic Savio. 2010. Interacting with the SOA-based Internet of Things: Discovery, query, selection, and on-demand provisioning of web services. *IEEE Transactions on Services Computing* 3, 3, 223–235. DOI:10.1109/TSC.2010.3

Gonzalo F. Huerta-Canepa, Angel A. Jimenez Molina, In-Young Ko, and Dongman Lee. 2008. Adaptive activity based middleware. *IEEE Pervasive Computing* 7, 2, 58–61.

Angel A. Jimenez Molina, Jun-Sung Kim, Hyung-Min Koo, Byung-Seok Kang, and In-Young Ko. 2009. A semantically-based task model and selection mechanism in ubiquitous computing environments. In *Proceedings of the 13th International Conference on Knowledge-Based Intelligent Information and Engineering Systems*, *Lecture Notes in Computer Science*. Vol. 5712, Springer, Berlin, 829–837.

Angel A. Jimenez Molina and In-Young Ko. 2011a. Spontaneous task composition in urban computing environments based on social, spatial, and temporal aspects. *Engineering Applications of Artificial Intelligence Journal* 24, 8, 1446–1460. DOI:10.1016/j.engappai.2011.05.006

Angel A. Jimenez Molina and In-Young Ko. 2011b. Cognitive resources aware service provisioning. In *Proceedings of 2011 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT'11)*. 438–444. DOI:10.1109/WI-IAT.2011.251

Jing Li, Yongwang Zhao, Min Liu, Hailong Sun, and Dianfu Ma. 2010. An adaptive heuristic approach for distributed qos-based service composition. In *Proceedings of the 15th IEEE Symposium on Computers and Communications*. 687–694.

Vassilis Kostakos, Tom Nicolai, Eiko Yoneki, Eamonn O'Neill, Holger Kenn, and Jon Crowcroft. 2009. Understanding and measuring the urban pervasive infrastructure. *Personal and Ubiquitous Computing* 13, 355–364. DOI:10.1007/s00779-008-0196-1

Benoît Laurent and Jin-Kao Hao. 2009. List-graph colouring for multiple depot vehicle scheduling. *Mathematics in Operational Research* 1, 1–2, 228–245.

Malcolm McCullough. 2001. On typologies of situated interaction. *Human–Computer Interaction* 16, 2, 337–349.

Sonia B. Mokhtar, Jinshan Liu, Nikolaos Georgantas, and Valerie Issarny. 2005. QoS-aware dynamic service composition in ambient intelligence environments. In *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering*. ACM, 317–320. DOI:10.1.1.406.2303

George L. Nemhauser and Laurence A. Wolsey. 1999. *Integer and Combinatorial Optimization*. Wiley-Interscience, New York, NY.

Eric Paulos and Elizabeth Goodman. 2004. The familiar stranger: Anxiety, comfort, and play in public places. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'04)*. ACM, New York, NY, 223–230. DOI:10.1145/985692.985721

Manuel Roman, Christopher Hess, Renato Cerqueira, Anand Ranganat, Roy H. Campbell, and Klara Nahrstedt. 2002. Gaia: A middleware infrastructure for active spaces. *IEEE Pervasive Computing* 4, 1, 74–83. DOI:10.1109/MPRV.2002.1158281

Irina Shklovski and Michele F. Chang. 2006. Urban computing: Navigating space and context. *IEEE Computer* 39, 9, 36–37.

Joao P. Sousa and David Garlan. 2002. Aura: An architectural framework for user mobility in ubiquitous computing environments. In *Proceedings of 3rd Working IEEE/IFIP Conference on Software Architecture*, *IEEE/IFIP*, *Montreal*. Canada, 29–43. DOI:10.1.1.94.3088

Zhenyu Wang and David Garlan. 2000. *Task-Driven Computing*. Technical Report CMU-CS-00-154. School of Computer Science, Carnegie-Mellon University.

Wilson Wong, Wei Liu, and Mohammed Bennamoun. 2012. Ontology learning from text: A look back and into the future. *ACM Computing Surveys*, 44, 4, 20. DOI:10.1145/2333112.2333115

Received October 2014; revised October 2015; accepted October 2015