



Works in Progress

Editor: Anthony D. Joseph ■ UC Berkeley ■ adj@eecs.berkeley.edu

Activity-Based Computing

EDITOR'S INTRO

This month's Works in Progress department features six projects that relate to activity-based computing. The first project is on a hovering information service for mobile devices and is exploring the limits and constraints of their hovering approach. The second one analyzes spatio-temporal data to learn activity patterns—in particular, for taxi drivers. The third project proposes a conflict-resolution method for context-aware applications and is evaluating a prototype TV manager application, and the fourth project extends pervasive middleware to a mobile environment and attempts to manage and avoid conflicts. The fifth project is developing a novel sensor network for controlling power consumption, and the final project is developing an assistant tool to help end users design their own context-aware applications. —Anthony D. Joseph

geographical area unit, given system constraints. Such constraints might be the number of mobile devices, the amount of memory available on each device, and the number of neighboring devices. We're also undertaking the development of a hovering information service for a set of self-organizing robots.

For more information, contact Giovanna Di Marzo Serugendo at dimarzo@dcs.bbk.ac.uk.

HOVERING INFORMATION

*Giovanna Di Marzo Serugendo, University of London
Alfredo A. Villalba Castro and Dimitri Konstantas, University of Geneva*

Hovering information is a decentralized, self-organizing, infrastructure-free, and location-aware information-dissemination service built over a highly dynamic set of mobile devices.¹ It spreads and disseminates locally produced information among mobile users or applications. It stores individual pieces of hovering information on mobile devices located in the area in which the information was initially produced, even if these devices are unreliable. These pieces act as active entities, detaching themselves from physical media constraints and associating themselves with space and time. They use locally sensed data—such as the direction, position, power, and storage capabilities of nearby mobile devices—to select the next appropriate location.

When moving between mobile devices, the pieces of information use mechanisms such as active hopping, replication, and dissemination. When a mobile device doesn't have sufficient space, or for security and dependability purposes, the pieces combine these mechanisms with swarm-based algorithms to help them self-split into single pieces of information, scatter fragments of information, and recombine information.

The pieces are naturally self-adaptive and context aware in their quest for optimum storage capabilities. The hovering-information service lets applications share (sense and modify) context information related to a precise area and time so that they can adapt their behavior accordingly.

Initial proofs of concepts show that our service offers high levels of information availability even for low levels of redundancy. Current research includes studying the hovering information's absorption limits—that is, how many pieces can be stored by a

REFERENCE

1. A. Villalba Castro, G. Di Marzo Serugendo, and D. Konstantas, *Hovering Information—Self-Organising Information That Finds Its Own Storage*, tech. report BBKCS-07-07, School of Computer Science and Information Systems, Univ. of London, 2007, www.dcs.bbk.ac.uk/research/techreps/2007/bbkcs-07-07.pdf.

ACTIVITY-BASED MOBILITY INTELLIGENCE

Liu Liang and Duan Zheng-yu, Tongji University

Activity-based computing reveals how people behave in their daily lives. We're developing an approach for determining activity patterns that uses continuous tracking.

Mobile positioning systems' large-scale deployment in recent years has led to a voluminous increase in the

records of where and when people have traveled. By analyzing spatio-temporal data (latitude, longitude, time stamp, and activity), we can learn an individual's activity pattern to improve his or her ability to solve problems while traveling—problems such as how to earn money or how to identify the best route to a destination. We refer to this as mobility intelligence (performing intelligent activities while traveling). Meanwhile, by aggregating numerous individual activity patterns, we're trying to derive an urban activity pattern.

Investigating spatiotemporal activity patterns can render a quantitative understanding of crowd behavior. We intend to enhance taxi drivers' ability to make money through continuous activity-based computing aimed at identifying when the taxi is carrying a passenger.

For this purpose, we calculated the spatiotemporal distribution of taxis to infer the crowd's demand distribution. We also used the data to describe the activity life cycle (noting its start and end) of certain areas, such as a central business district, railway station, or nightclub. We also classified the taxi drivers by their income—*experts* made more money and *greeners* made less. We then further analyzed the data to determine why experts earn more. We're also using an ethnological approach to study the different ways taxi drivers operate in the city.

Moreover, we're analyzing the role of context in taxi-driver activities to reveal when and where a driver operates and how this relates to urban events (such as a large-scale public event) and the surrounding environment. We're collecting data from 3,000 taxi drivers' GPS data (latitude, longitude, time stamp, and activity), augmented by interviews with the taxi-company managers. The study concentrates on taxi drivers in Shenzhen, China.

For more information, contact Liang Liu at liuliangchina@gmail.com.

CONFLICT RESOLUTION IN MULTIUSER CONTEXT- AWARE ENVIRONMENTS

Thyagaraju G.S. and Umakanth P. Kulkarni, Sri Dharmasthala Manjunatheshwara College of Engineering and Technology
Anil R. Yardi, Walchand Institute of Technology

We propose a context-aware ubiquitous Conflict Manager to resolve conflicts for context-aware applications. Conflicts arise when multiple users access an application. To resolve such conflicts, the Conflict Manager considers user preferences and service properties to prioritize users. The assigned priority changes dynamically, depending on factors such as the user's schedule.

**To resolve conflicts,
the Conflict Manager
considers user preferences
and service properties
to prioritize users.**

The Conflict Manager comprises two parts. The *Conflict Resolver* prioritizes a user on the basis of his or her characteristics. The *Scenario Manager* computes the priority dynamically for exceptional situations that occur in the service area, considering parameters such as a conflicting user context or environmental conditions. The Conflict Manager resolves conflicts by choosing the user with the highest priority.

We've applied our conflict-resolution method to a context-aware TV application that will offer ubiquitous characteristics. For example, it will be aware of its power consumption and current number of users, capable of considering potential privacy concerns, and capable of interacting with nearby devices (such as a laptop) or appliances (such as a refrigerator). Our goal is to enable a context-aware application

that can offer personalized services to multiple users by resolving service conflicts among the users.

For more information, contact Thyagaraju G.S. at thyagaraju_gs@yahoo.co.in or Umakanth P. Kulkarni at upkulkarni@yahoo.com.

ADAPTIVE ACTIVITY-BASED MIDDLEWARE

*Gonzalo Huerta Cánepa,
Angel Jiménez Molina, In-Young Ko,
and Dongman Lee, Information
and Communications University*

Recognizing users' activities and executing tasks on their behalf without intervention are two themes of activity-based computing. While the former lets us determine context, the latter is the core of this computing paradigm. Both tasks are difficult to accomplish. In addition, even more issues arise if we consider user mobility and the need to support users' activities regardless of their location. Furthermore, users have opposing goals, so we need a way to resolve possible conflicts between them.

To support developers in this regard, we're extending our pervasive middleware, Active Surroundings, to a mobile environment (see figure 1). Our system represents the user's goals as a collection of tasks and actions, which it further describes as abstract resources and services. It maps these abstract components to concrete instances in every space the user visits, providing an interspace service provision (the adaptation and continuation of tasks between these spaces) based on the activities.

The context manager on users' mobile devices identifies user activities based on the current context, an ontology, and a classifier that uniquely characterizes the activities. In every location, information about the space is retrieved through direct interac-

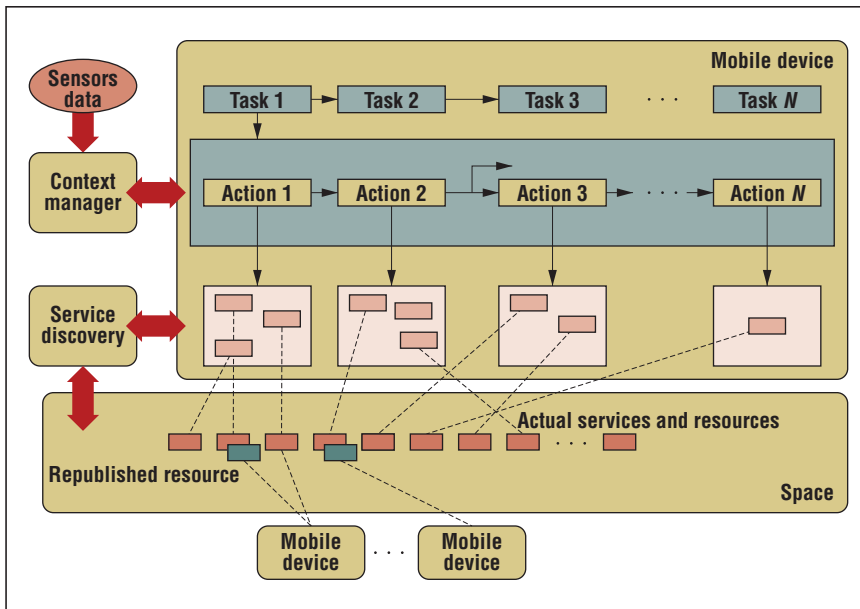


Figure 1. Active Surroundings. The extended architecture for our activity-based computing middleware supports continuity of tasks and conflict resolution and avoidance.

tion between the device and context providers. Activities that match that context are triggered by the device on the user's behalf. A selection process enables tasks based on those activities, and the abstract services that composed them are mapped to concrete instances using a service discovery component. The concrete services are run if all of their resource requirements are met.

To avoid interference between users, our middleware manages conflicts at the service level and avoids them at the resource level. At the service level, if another user's activity will be affected by the execution of some of our services, the service discovery process finds alternative services based on the needed features and the effect on the space and other users. At the resource level, we avoid conflicts by assigning a resource to the first user who requests that resource. In this case, the new owner can share the resource by modifying its attributes so that the owner's tasks aren't affected if another user requests the same resource.

For more information, contact Angel Jiménez Molina at angeljim@icu.ac.kr.

A HOME SENSOR NETWORK THAT CONTROLS POWER CONSUMPTION

*Jaewook Jung, Youngjae Kim, and Minsoo Hahn, Information and Communications University
Tatsuya Yamazaki, Tetsuo Toyomura, and Takashi Matsuyama, National Institute of Information and Communications Technology*

The power-consumption pattern of appliances is tightly coupled with user activity. So, gathering power-consumption data as well as time and location information could help home energy-saving services focused on decreasing global warming.

We're developing a novel sensor network, consisting of one manager module and multiple client modules, that monitors a user activity to help save energy. The manager analyzes the measured power consumption data and selects which service to provide on the basis of the user activity it detects. A client connected to an appliance measures the appliance's power consumption and sends the measured

data to the manager. The client then receives a control command from the manager through ZigBee (www.zigbee.org) and controls the supplying power rate accordingly.

We've applied our sensor network to the National Institute of Information and Communications Technology's Ubiquitous Home, a smart-home research project. The network provides three energy-saving services—one for when there's just one person in the home and a second and third for when there's two or more people during the day and for nighttime use, respectively. These services are based on power consumption, time of day, location, and number of people in the home, and they can reduce energy consumption by 4.54 to 45.45 percent.

For more information, contact Jaewook Jung at jwjung@icu.ac.kr.

A TOOL FOR DEVELOPING ACTIVITY-BASED APPLICATIONS

*Thyagaraju G.S. and Umakanth P. Kulkarni, SDM College of Engineering and Technology, Dharawad
D.B. Kulkarni and Anil R. Yardi, Walchand College of Engineering and Technology, Sangli*

Although researchers have developed numerous context-aware applications for storing and retrieving contextual information, developing context-aware activity-based ubiquitous applications is still difficult. We're developing a system that will let end users visually design a variety of these applications, including those based on policies.

Our system's basic architecture comprises sensors (the physical layer), an aggregator, software widgets, a software interpreter, a context server, and context applications (see figure 2).

Our system will also include a toolkit that incorporates all the basic architecture's principles and will help end

users design their own context-aware applications. For example, consider our Meeting Assistant, a prototype mobile, context-aware, activity-based application for inviting meeting attendees and for taking and retrieving notes. The Meeting Assistant has the following architectural components for design purposes:

- The Member (widget) acquires contact info and details of each member.
- The Scheduler (server) schedules the meeting place and calls all the members.
- The Agenda (widget) acquires the complete meeting agenda.
- The Notepad (widget) acquires the user's notes and relevant information.
- The User (server) aggregates all information about the user.
- The Question (widget) acquires members' questions and relevant info.
- The Location (widget) acquires arrivals or departures of members.
- The Recording (interpreter) detects whether audio or video is recorded.
- The Meeting (server) contains all the information about the meeting.

Our system provides all the necessary components for the user to develop activity-based applications.

To develop an application, we've proposed a set of application-based policies for each action. Generally, the programming structure for all applications can be divided into three main components. The values of these components determine the policy values.

The *context variables* are the set of variables that must store the context data. For example, for the Meeting Assistant, some of the context variables with the following assumed values:

```
count: INT: 0
Discussion: Y/N
A/C: ON/OFF
```

The *context actions* component is

the set of all possible actions required for the applications to run. For example, the Meeting Assistant's set might include

```
UPDATE(Varno,value);
StopLight();
StartLight( );
VaryLight(intensity)
```

The *context policies* are the set of all possible policies. This component uses if-else policy conditional statements. The following is for a very simple policy:

```
if ( count > 0)
    Startlight( )
else
    Stoplight( )
```

We plan to continue working on the Meeting Assistant and similar applications to help us develop our generic

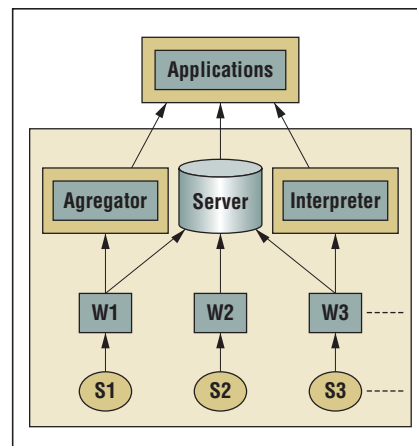



Figure 2. The architecture of our system for developing activity-based applications. Here we show the relationship between the different components. ("S" stand for sensors and "W" for widgets.)

tool. For more information, contact Thyagaraju G.S. at thyagaraju_gs@yahoo.co.in. 

Call

for Articles

IEEE Software seeks practical, readable articles that will appeal to experts and nonexperts alike. The magazine aims to deliver reliable information to software developers and managers to help them stay on top of rapid technology change. Submissions must be original and no more than 5,400 words, including 200 words for each table and figure.

Author guidelines: www.computer.org/software/author.htm
Further details: software@computer.org
www.computer.org/software

IEEE Software