

IWSAWC 2005

International Workshop on Smart Appliances and Wearable Computing

Adjunct Proceedings

June 10th, 2005 in Columbus, Ohio USA

P03: *SBAM: A Socket-level Bandwidth Aggregation Mechanism*

Hiroshi Sakakibara, Masato Saito and Hideyuki Tokuda (Keio University)

P04: *Prototyping a Real-World-Oriented Monster-Collection Game*

Nao Kawanishi, Yoshihiro Kawahara, Hiroyuki Morikawa and Tomonori Aoyama (The University of Tokyo)

P05: *A File System for Resource Abstraction in Ubiquitous Computing*

Till Riedel and Christian Decker (Tec0, University of Karlsruhe)

P06: *Activity Model using Location and Places' Attributes for Navigation Services*

Yuki Matsukura, Naohiko Kohtake, Kazunori Takashio and Hideyuki Tokuda (Keio University)

SBAM: A Socket-level Bandwidth Aggregation Mechanism

Hiroshi Sakakibara Masato Saito Hideyuki Tokuda
Graduate School of Media and Governance, Keio University
5332, Endo, Fujisawa, Kanagawa, 252-8520, Japan
E-Mail: {skk, masato, hxt}@ht.sfc.keio.ac.jp

Abstract

Due to the recent explosive growth of network technology, the number of wearable computers and PDAs with wireless network technologies, such as 802.11b, 802.11a, Infrared, Bluetooth and Ultra WideBand (UWB), has increased. Furthermore the volume of web contents and network services which require wide bandwidth, such as VoIP, have increased. It is difficult to receive such kinds of contents by exploiting single N/I because of the lack of bandwidth. However, these network interfaces are used only one at a time. An intuitive solution against the starvation of bandwidth is to exploit several interfaces at the same time. In this paper we propose A Socket-level Bandwidth Aggregation Mechanism (SBAM), which aggregates the bandwidth of several interfaces connected to the computer, even if the media are heterogeneous. Since SBAM is implemented in socket layer of the operating system, it can avoid the drawbacks caused by the implementation in network stack or application layer and achieves bandwidth aggregation efficiently almost with minimal changes to existing software.

1. Where to Implement?

Since we aim to apply of SBAM to existing applications and network protocols, we have made assumptions similar to real life. They are gradual deployment and heterogeneous wireless media. The following are the requirements for SBAM:

- R1: dealing with different kind and number of wireless media
- R2: ease of deployment
- R3: use of existing applications without modification
- R4: fitting into the existing network protocols

Although Bandwidth Aggregation Mechanism (BAM) can be implemented in network stack, each of it has drawbacks. Since layer 2 and 3 are processed in hop-by-hop manner, we cannot acquire end-to-end network information. In addition, we need to deploy the new implementation on many switches and routers. Therefore R1 and R2 are not satisfied. 802.3ad [3] is a related work that is a layer 2 technology that aggregates bandwidth between hosts with multiple N/Is and switches. Implementation in layer 4 is the best choice for performance and most of the related works do so [1, 2]. However deploying a new layer 4 protocol is hard because of broad installation of TCP and UDP. Furthermore, we must modify existing software for the protocol. There-

fore R2, R3 and R4 are not satisfied. Implementing BAM as application does not satisfy R3, since existing software must be modified to use its API. Furthermore the author of [1] reported that BAM in application does not scale well with increase of N/Is, so R1 is not satisfied.

Socket is a feature of an operating system, which means that it is unnecessary to modify the existing software and implementation of SBAM scales well with increase of N/Is. In addition, various transport layer protocols can be exploited for BAM because this is placed upon layer 4, and end-to-end network information can be collected. These are the reasons we chose the socket layer for implementing BAM.

2. SBAM

In this section, we show the detail of SBAM. First we explain each function of SBAM, then show the operation when sending and receiving data.

2.1. System Architecture

SBAM consists of 5 functions shown in figure 1. The delay and the available bandwidth of each link are measured in Network Monitoring Function (NMF); the former using ICMP echo reply, and the latter using packet pair measurement method. These information are exploited in Send-data Schedule Function (SSF), which schedules the amount of data that should be sent from each N/Is. The scheduling algorithm of SSF can be divided into two parts; transmission of subdivision of bandwidth-delay product part (A) and transmission in proportion to bandwidth part (B). Part A is executed at the start of sending data, then transits to part B. In part A, bandwidth-delay product (BDP) of each link is calculated, and the subdivision of BDP is sent on a bigger BDP link. After that data which is proportional to bandwidth is sent on each link as part B. This algorithm aims to fill the links between source and destination host. Send-data Divisioning Function (SDF) divides data to send into a number of N/Is based on the decision of SSF and attaches SBAM headers to each. The divided data are aggregated into one data flow in Receive-data Aggregation Function (RAF) at a destination host. SBAM header includes a packet sequence number and data length of payload. N/I state Notification Function (NNF) notifies a number of N/Is from destination to source host. This enables source host to send to multiple N/Is on destination host.

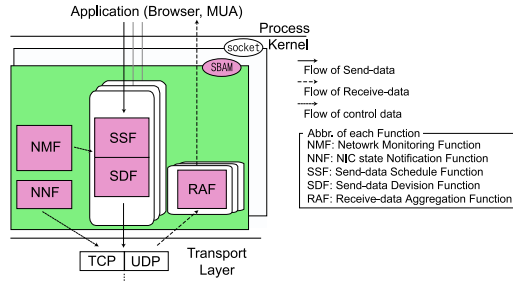


Figure 1. System Architecture of SBAM

2.2. Send- and Receive-Operation

SSF starts to measuring delay and available bandwidth of each link before sending data. In case of the existence of multiple N/Is on the destination host, information is sent through NNF, and informed to NMF. Then the amount of data that will be transmitted from each N/Is is decided by SSF based on the link state, and the data division and SBAM header attachment are done in SDF. Finally the data is passed to lower protocol stack, such as TCP or UDP.

When receiving data through SBAM, NNF notifies a number of available N/Is to the source host. Received data is ordered into a sequence and SBAM header is removed in RAF, then data is passed to application.

3. Implementation and Evaluation

Current prototype implementation of SBAM has SSF and RAF on FreeBSD 5.1R and supports two 802.11b N/Is. Although the wireless media is the same, we have exploited various media types, such as 1Mbps, 2Mbps, 5.5Mbps and 11Mbps. We installed a dummynet host to control delay and bandwidth between two hosts to approximate the experiment environment. We also implemented application program, called ABAM, which has NMF, SSF, SDF, NNF and RAF to compare the throughput difference between SBAM's socket level implementation, and application level implementation. ABAM is also used to evaluate the algorithm of SSF.

At first we verified the throughput increase using SBAM and ABAM on IBM Thinkpad X30 with two 802.11b N/Is. The results show that the throughput increased by a factor of 1.6 compared to single N/I on both SBAM and ABAM. However the increased throughput of SBAM was 97.4% compared with that of ABAM. This is caused by many calls to *rtalloc()* and linear search algorithm in the current SBAM implementation.

Next, we verified the efficiency of the two parts in SSF by measuring the queue length on the destination host and the packet loss rate. For the evaluation of part A, we sent data with and without part A in SSF to measure the queue length. Figure 2 shows queue length on destination host which was required to order the packets through different links. Vertical axis is the queue length and horizontal axis is time since first packet sent. The result shows queue length increased slightly with part A. This implies we need to estimate packet sequence of arrival in our algorithm.

For evaluating part B, we sent data with and without part B and measured packet loss rate. With part A, no packet loss was de-

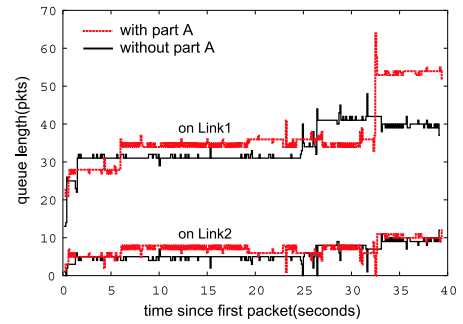


Figure 2. Effect of BDP part (part A)

tected, but without part B, loss was 41.9%. We found that part B utilized available bandwidth effectively.

4. Summary and Future Work

In this paper we proposed SBAM, a Socket-level Bandwidth Aggregation mechanism. We implemented SBAM in socket layer and also implemented an application program, ABAM, with several SBAM functions. SBAM and ABAM with two 802.11b N/Is achieved by a factor of 1.6 throughput increase compared to that with single N/I. Although the increased throughput of SBAM was 97.4% compared with that of ABAM. Frequent calls to *rtalloc()* and the linear search algorithm caused this. We found that part A in SSF needs to take care of arrival sequence number on destination host, and part B utilizes available bandwidth effectively.

Issues for future work are:

- to decrease queue length on destination host,
- to increase throughput,
- to implement rest of the functions,
- to evaluate in various environments, for example using TCP, using more than 3 N/Is and so on,
- dynamic attaching and detaching of N/Is and
- seamless loaming in various wireless environment.

5. Acknowledgement

This work has been conducted in Ubila Project by Ministry of Internal Affairs and Communications (MIC).

References

- [1] H.-Y. Hsieh, K.-H. Kim, Y. Zhu, and R. Sivakumar. A receiver-centric transport protocol for mobile hosts with heterogeneous wireless interfaces. In *Proceedings of MobiCom '03*, pages 1–15, New York, NY, USA, September 2003. ACM Press.
- [2] H.-Y. Hsieh and R. Sivakumar. A transport layer approach for achieving aggregate bandwidths on multi-homed mobile hosts. In *Proceedings of ACM MobiCom '02*, pages 83–94, New York, NY, USA, September 2002. ACM Press.
- [3] IEEE 802.3ad Standard (IEEE Computer Society LAN MAN Standards Committee). *Aggregation of Multiple Link Segments*, 2000.

Prototyping a Real-World-Oriented Monster-Collection Game

Nao Kawanishi[†] Yoshihiro Kawahara[‡] Hiroyuki Morikawa[†] Tomonori Aoyama[‡]

[†]Graduate School of Frontier Sciences,
The University of Tokyo
5-1-5-612 Kashiwanoha, Kashiwa-shi,
Chiba, 277-8561, JAPAN

{river24,mori}@mlab.t.u-tokyo.ac.jp

[‡]Graduate School of Information Science and Technology,
The University of Tokyo
7-3-1 Hongo, Bunkyo-ku,
Tokyo, 113-8656, JAPAN

{kawahara,aoyama}@mlab.t.u-tokyo.ac.jp

Abstract

In this paper, we present a new monster-collection game called “Ubiquitous Monster” as a real-world-oriented entertainment application. In Ubiquitous Monster, autonomous entities called monsters live in networked computer nodes and autonomously migrate between nodes while interacting with other monsters. These monsters are then collected by mobile users as they travel between environments. Different types of monsters appear in different types of environments, the nature of each being determined by data gathered from local sensors.

1. Introduction

Among the many achievements computer technology has produced, it would not be an exaggeration to say that computer games are among the most successful. Computer game style has evolved over time in accordance with computer technology advancement. In the ubiquitous computing environment, which integrates the real and virtual world, computer games will experience a revolutionary change as real-world information is incorporated into the virtual world of the game[1]. Computer games supported by ubiquitous computing technologies are expected to be a “killer application” in the new ubiquitous computing environment.

We propose a new monster-collection game called “Ubiquitous Monster.” In Ubiquitous Monster, autonomous monster characters live in the virtual world, the environment of which is updated dynamically to reflect information acquired by sensors embedded in the real world. Toting handheld computing devices, users move about the real world, collecting the monsters which appear depending upon the physical condition of the place. We developed Ubiquitous Monster using an adaptive network service platform. Monsters are implemented as autonomous

service components, allowing game characters to proliferate autonomously depending upon the condition of the physical environment. As a result, the game scenario can be made dynamic. In this paper, we present Ubiquitous Monster and describe the design and implementation of our system in order to realize an autonomous distributed mechanism allowing monsters to behave depending upon real-world information and a mechanism allowing users to capture monsters.

2. Ubiquitous Monster

In the ubiquitous computing environment where all entities have network connections, the real world where we live and the virtual world which is created by computer networks are interconnected. It is possible to acquire real-world information using sensors and use it to create a model of the real world in the virtual world. It is also possible to control the actuators in the real world using virtual world information. Exploiting these concepts, we aim to create a new monster collection game, the stages of which vary dynamically based upon the interactions between the real and virtual worlds.

When a user approaches an “area” in the real world, a “monster” which lives in a “field” corresponding to the given “area” appears on the user’s terminal. Users move around several areas in the real world, collecting monsters. Users may also release monsters captured in one area into a different area.

In contrast to current games in which monsters live in a predefined field, monsters in our game appear in different fields depending upon real-world information acquired by sensors embedded in areas where the game is played. For example, monsters preferring bright environments appear in areas for which the corresponding field is bright. Similarly, monsters preferring colder environments appear in areas for which the corresponding field is cold. It is also possible that

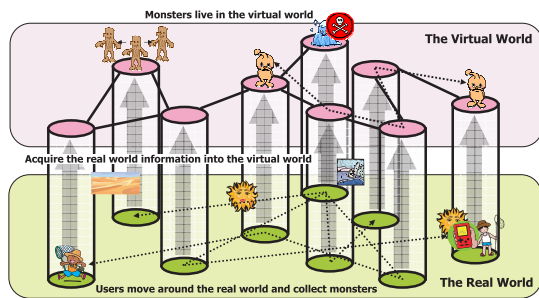


Figure 1. the whole image of “Ubiquitous Monster”

different monsters may appear in the same area of the real world depending upon the time of day.

Furthermore, by interconnecting different fields in the virtual world, monsters can move about the network of fields autonomously in search of an environment better suited to them. Thus we simulate an ecosystem of monsters in order to make game scenarios more dynamic and attractive. For example, a monster which loves bright places can survive and even multiply in a bright field, however the darker the field becomes, the more emaciated the monster becomes; eventually dying unless it is able to find a brighter field.

In this game, monsters appear in the environments they love the most. We named the game “Ubiquitous Monster” because the “monsters” are “ubiquitous” (Figure 1).

3. Design & Implementation

In order to realize Ubiquitous Monster, we designed and implemented an autonomous distributed mechanism allowing monsters to behave differently depending upon real-world information and a mechanism allowing users to capture monsters.

In order to implement their autonomous behavior, monsters are implemented as cyber-entities (CE's) which are the autonomous component of an adaptive networking service platform called “Jack-in-the-Net” (Ja-Net)[2]. We also implement real-world information acquired by sensors embedded in the area as CE's. The Ja-Net platform allows us to easily implement active applications triggered by the interaction of CE's. The monster CE's acquire real-world information by interacting with the sensor CE's and react upon it. Each monster CE contains “energy” which is necessary for living in the virtual world and a set of “preferences” representing the monster's preference for real-world information. If the environmental information matches a monster's preferences, the energy it possesses increases accordingly; otherwise its energy decreases. Real world information CE's can also migrate between network fields so that real-world

information is shared among the different fields. By interacting with the real-world information CE, monster CE's can attempt to migrate to fields better suited to their preferences.

When a user approaches an area, a monster on the field corresponding to that area appears on the user's game terminal and the user can then capture the monster. To capture the monster, the user is required to take an action which is unique to the given monster. If the user succeeds at performing the required action, the monster is captured at the user's game terminal. However, if the user fails, the monster goes back to the field. One scenario requires the user to swing a net to capture the monster. The user swings a net equipped with an acceleration sensor and the given monster is captured when the acceleration sensor reading on the net exceeds the threshold value set in each monster. We use active RFID for proximity detection and mica mote for acceleration detection of the net.

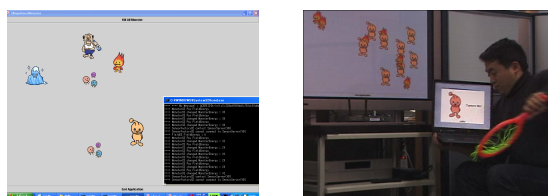


Figure 2. Screenshot of the monsters on a field (left) and user capturing monster by swinging the net (right).

4. Conclusion

In this paper, we presented a monster-collection game called Ubiquitous Monster which is realized in the ubiquitous computing environment. In Ubiquitous Monster, monsters live autonomously in the virtual world which changes dynamically depending upon real-world information acquired by sensors embedded in our living world.

Acknowledgment

This work is supported by Ministry of Public Management, Home Affairs, Posts and Telecommunications.

References

- [1] Staffan Björk, Jussi Holopainen, Peter Ljungstrand and Regan Mandryk: “Special Issue on Ubiquitous Games,” *Personal and Ubiquitous Computing*, Volume 6, Issue 5-6, pp.358-361, December 2002.
- [2] Tomoko Ito, Tetsuya Nakamura, Masato Matsuo, Tatsuya Suda and Tomonori Aoyama: “Adaptive Creation of Network Applications in the Jack-in-the-Net Architecture,” In *Proceedings of the IFIP Networking 2002*, May. 2002.

A File System for Resource Abstraction in Ubiquitous Computing

Till Riedel

Telecooperation Office (TecO)
University of Karlsruhe
76131 Karlsruhe, Germany
+49 721 6902 57
riedel@teco.edu

Christian Decker

Telecooperation Office (TecO)
University of Karlsruhe
76131 Karlsruhe, Germany
+49 721 6902 72
cdecker@teco.edu

ABSTRACT

This paper proposes a file system as an abstraction layer for a uniform way of accessing any system resource on wireless sensor nodes. Even functions and libraries can be represented and accessed in this uniform way allowing developers a novel way to design and implement applications. A light weight implementation on our Particle Computer platform is described and performance measurements indicate only a small overhead for resource accesses.

Keywords

Resource management, file system, particle computers, ubiquitous computing, system architecture

INTRODUCTION

In ubiquitous computing environments tiny networked sensor nodes are embedded in a variety of objects. Application programs on the nodes utilize many resources such as different sensors, actuators like LEDs and speakers, memory for data storage and the wireless communication interface. Other computational functionalities like algorithms are encapsulated in libraries. Various approaches were developed to ease the development of applications on wireless sensor nodes. Still, developers struggle with the diversity of different resources and their particular access methods. We propose a file system which provides a uniform name space and access model for all resources. An implementation on our Particle computer platform (<http://particle.teco.edu>) proves the feasibility for use on small sensor nodes and performance measurements indicate only a small overhead when accessing resources through this abstraction layer.

UNIFORM RESOURCE ACCESS

In general two kinds of resources can be identified. Direct resources represent the hardware on the platform, like sensors. Mediated resources abstract functional units, that may also be accessing more other resources. Accessing either kind through a uniform interface enables us to abstract from artificial boundaries like hard- and software or remote and local resources. The system programmer can add, modify and move device driver functionality transparently. At the same time an application may be developed without any knowledge about the underlying system. Inspired by the “/proc” file system introduced by Plan 9 [3] the Particle File System [2] allows to represent operating system func-

tionality along with file storage, devices and remote resources within in a common name space. Software can access any shared functionality in a system through the stream-oriented functions “read”, “write”, “open” and “close”.

Name Spaces

The abstraction of name spaces provides an intuitive way of addressing any resource. Hierarchical name spaces prove to be an adequate means to categorize any resources independent of its internal representation. Textual natural language resource identifiers specify a path into the name space that uniquely identifies a resource. Internally this resource can be represented by a fixed size machine-readable address pointing to an arbitrary implementation. Hierarchical name spaces follow strict compositional semantics. New parts of the name space can be built into the existing hierarchy via the “mount” command. This makes it easy to extend the system to react to dynamic settings. A call to “open” descends into this commonly agreed name space and couples information provider and consumer in an ad hoc manner.

Streams

As addressing will a priori occur less often than accessing the resource, we lay stress on optimizing resource access. Starting from the days of early UNIX operating system data access by using stream primitives has proven to be a scalable and powerful abstraction. Streams consist of a pair of access functions “read” and “write” as well as a state that enforces the semantics of the sequential access. We found that such semantics fit most if not all resources in our system without the need of complicated conversion. Stream drivers may invoke other streams to pass on data. This stacking of streams can extend to a flexible data processing mechanism through mediated resources.

IMPLEMENTATION

As depicted in figure 1 once a name is resolved to an address by calling “open”, a file descriptor is associated with the stream. The file descriptor table provides a cached call indirection to the driver function that provide “read” and “write” functionality. A driver specific stream state is associated with each file descriptor. The driver and the initial state are obtained by resolving the resource identifier with

“open”. Name resolution is done via a flattened tree data structure, which establishes a relation between the resources. Each 16bit tree element identifies a resource by generic 13bit addresses. Because the tree is stored in pre-order depth search can be implemented easily with only 3-bit overhead in the data structure.

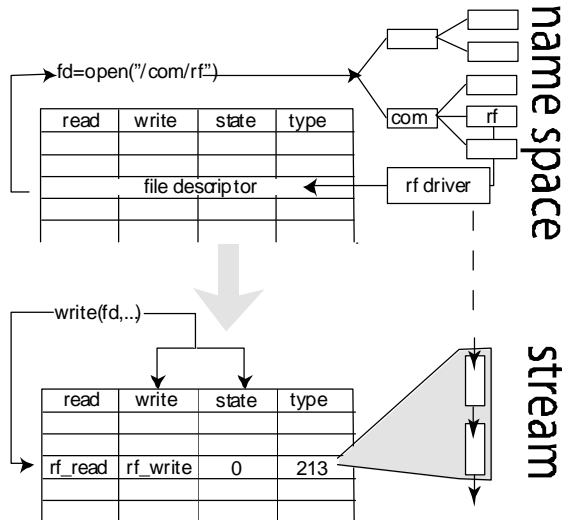


Figure 1: Resource access through file system calls

The resulting space of resource addresses is not uniform making it necessary to use specific access function for different addresses. By defining size-aligned domains within the complete address space a type system is encoded into the address. Typing allows dispatching the correct access method on a resource address. By using address translation on method invocation the access method need to have no knowledge about the address space layout. Being able to address 2^{13} distinct resources medium size storage can be integrated directly into the addressing scheme. Persistent sequential files on a 4-MBit flash chip are addressed directly. A flash driver translates stream semantics to sequential page-based file structures. Writing is done append-only via the page buffer of the flash chip allowing file size to grow dynamically. By doing explicit buffer allocation and tracking of the stream state, reading and writing over multiple streams on multiple files is possible.

APPLICATIONS

The file system serves as a runtime environment for programs running on the particle platform. Especially many existing programs written in the C programming language rely on this functionality by using the I/O functionality of the standard C Libraries. The functions “fprintf” or “getchar” are examples. We can provide full support for arbitrary resource access using such stream functions. The file system interface also provides a generic interface to remote Particles. Different gateways can be integrated into a PC Infrastructure. The PC can establish a control connection to the FTP proxy via standard browser technology, which translates control connection request to remote file system calls. The user can reprogram a specific Particle in

the network by uploading a binary image. In addition to data transfer the same remote file system interface serves as command shell to access “executable files” on the Particles via a telnet proxy.

PERFORMANCE

If we want to use the file system as a resource interface on the particle itself performance becomes a predominant issue. After resolving the name the actual overhead for using the file system for resource access is reduced to a few indirection mechanism on calling the function. The read and write functions on file descriptors expand to an indirect call via a function pointer. Also additional overhead is generated by passing length and state arguments to the driver function. Using micro benchmarking we measured a minimum of 100 processor cycles for a file system resource access on a PIC18 processor. This compares to 26 cycles when calling a static interface. Both values relate to compiler-generated code reading a one byte of instantly available sensor data. Replacing existing interfaces with the file system a second area of interest is persistent file storage. Speaking only of a few hundred Kbytes of persistent memory, the gain of file-structured storage has to relate to the overhead introduced by such structures. In our system the on page meta structures take only 5 byte of each 264-byte page. We use 256 byte of the processors EEPROM data memory to manage page allocation, which can be done without additional blocking time. Allocation of the hardware buffers of the flash chip is done explicitly using pre-emption mechanisms. When writing data to the flash through a single file system stream this mechanism achieves minimal blocking times. Write overhead is only generated by the inferior erase strategy that is necessary to keep the file system consistent. Reading the flash needs no allocation and imposes no additional overhead on the application.

CONCLUSIONS

We believe that file system functionality proves to be a powerful tool to decouple layers of software and hardware. The hierarchical name space and the standardized I/O system can be used to express many high level abstractions as system resources, while keeping the performance penalty for indirectly accessing hardware small. The file system strives to reduce design time, while maximizing the design space of the Particle Computer platform.

REFERENCES

1. Beigl, M., Zimmer, T., Krohn, A., Decker, C. and Robinson, P.. Smart-Its-Communication and Sensing Technology for UbiComp Environments, Tech. Report ISSN 1432-7864, 2003.
2. Decker, C., Beigl, M., and Krohn, A.. A file system for system programming in ubiquitous computing, LNCS 3432, 2005.
3. Presotto, D., Trickey, H., Thompson, K., Winterbottom, P. and Pike, R.. The use of Name Spaces in Plan 9. Operating Systems Review 27, 1993

Activity Model using Location and Places' Attributes for Navigation Services

Yuki Matsukura, Naohiko Kohtake, Kazunori Takashio, Hideyuki Tokuda
Graduate School of Media and Governance, Keio University
{matsu, nao, kaz, hxt}@ht.sfc.keio.ac.jp

Abstract

This paper proposes a new way to construct an activity model which represents users' habit while users move in his/her daily life. In recent years, we can acquire users' more precise location using GPS. We represent users' habit by analyzing the users' location history data. The representation of the habit enables computers predict users' destination and adjust a sight-seeing path. Existing activity models can be adapted only for a usual activity area, although users' habit is independent for usual activity area or location. In this paper, we propose a new activity model, which uses not only users' location but also attributes of the place where user stayed. This model can be used in the first visited area as well as represent users' habits.

1. Introduction

We can obtain precise location information exploiting GPS because of the elimination of restriction on the system. It is enough accurate to distinguish where the user stands in front of which shop. In addition since GIS (Geographical Information System) has been developed, we can exploit meta-information of location such as coordinates of roads, a name of buildings and their labels. Exploiting these systems, location-aware services have been developed that provides helpful information to users. In psychological domain, it is defined that there are habits in usual human activities [4]. A habit is a regularity which humans act under a particular condition such as location or time. For example, "A user goes to cafeteria after goes to the restaurant". Modeling the human habit makes computers provide helpful services corresponding to the human activity.

As an example of helpful service for daily situation: Assuming a user goes restaurant A, after that, goes cafeteria B. And now, the user is in restaurant A. Then, a system can support the user's next activity by displaying information like the shortest path to the cafeteria B, congestion degree and available time. In unusual situation, if a tour path conductor system exploits users' activity model, the system can propose unusual activity or usual activity whichever the user wants.

It has been proposed the ways to construct activity model [1] [3]. However, when users stayed the first visited area,

the users want to act same as daily activity. Like looking for cafeteria a user usually goes. Existing activity model can construct in the only area where usual acting area. So users can't be served benefits of location-aware services which use activity model in the first visited area. The problem in existing modeling methods is that they use only coordinates to represent users' location. According to the problem, in the area where users visit first, the users' activity model needs movement history in the new area.

In this paper, we propose a Model for User activity using Geographical Information (MUGI). MUGI enables us to serve location-aware service corresponding to users' habit in first visited area. Activity model treats historical movement, so system must scale and operate fast.

2. MUGI

It is needed to weaken the relation between location and activity model to exploit activity model in the first visited area. We used not only location but also places' attributes to construct activity model. Places' attributes are meta-information of location and area. The reason to focusing on places' attributes is the one of changing contexts when user moves. So we constructed activity model using those contexts.

There are several abstraction levels for places' attributes in GIS. For example, we explain *McDonald's*, which has five abstraction levels as follows; the first is the most specific, the last is the most generic.

- *McDonald's* in Ginza
- *McDonald's*
- Fast-food
- Eating and drinking
- Stores

MUGI provides activity prediction and activity evaluation. Activity prediction indicates the place where user will go for the next. Activity evaluation judges suitability of prepared paths.

3. Design and Implementation

3.1. Design

In this research, we use Hidden Markov Model (HMM)[2] to construct MUGI, which model can express probabilistic automaton, holds state transition, can output

probability. The reason of adapting HMM is to express movement patterns between the places users stayed. The places' attributes is represented as a node in HMM.

Initial state of MUGI is organized only special node. The special node saves specific attributes in the usual activity area, like school, home and working office. This special node is not used prediction and evaluation in the first visited area. The other nodes constructing MUGI is made dynamically when user visited a new attribute's place. At the same time of making nodes, make a relation between nodes.

3.2. Implementation

We implemented the CHOCO (Figure1) which constructs MUGI using Java. And we implemented a platform which enables services to exploit MUGI. Figure2 shows a snapshot of the platform. The right-bottom panel shows current MUGI status. The boxes in the panel represent nodes, line represents state transition and the numbers next to the line represent probability of the transition. Internal attributes are showed above on the node by clicking a node. The right-top panel shows map and current location. Left side panel shows applications which is developed by a application provider. Applications are changed by selecting tab.

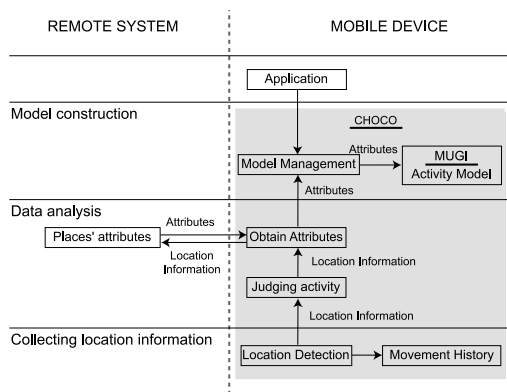


Figure 1. System design

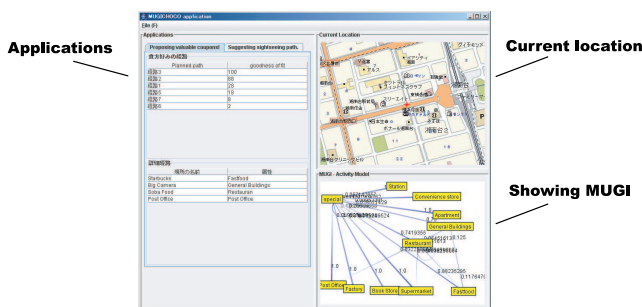


Figure 2. A snapshot of MUGI

4. Evaluation

4.1. Qualitative evaluation

Existing research to construct activity model like “Learning Significant Location and predicting User Movement with GPS[1]” and “Inferring High Level Behavior from Low Level Sensors[3]” are not able to use the first visited area. They enable precise activity prediction using movement history in a specific area. So they can't predict first visited area. MUGI constructs activity model using places' attributes which is more abstract information than location information. So MUGI can't predict users' movement precisely, but it can predict in the first visited area.

4.2. Quantitative evaluation

We evaluated CHOCO's scalability and operation performance. We have used 178 stay points of movement history, accumulated 62 days for the input for CHOCO.

We measured execution time of prediction and file size to save a MUGI image to evaluate operation performance. Maximum of execution time is 16ms, enough fast, because of the model is small, and we implemented MUGI to map on the physical memory. We have made MUGI learn for assumed 100 years movement calculated from 62 days movement to evaluate scalability. File size become about 1.1MB even input 100 years data. MUGI scales enough to save current mobile device such as PDA and cell phone.

5. Conclusions and Future work

In this paper, we proposed activity model, MUGI, using place's attributes. As an evaluation of MUGI, we confirm that MUGI scales and operates fast.

We are considering model sharing with other MUGI and making a transition weight to be more useful. Regarding model sharing, in the situation of more than two people using MUGI, each user's MUGI calculates its own prediction. MUGI should be synthesized with the other users' MUGI. Population parameter will increase with getting build up a activity model. So the one movement's effect weight will affect less with getting learned. So we need to treat transitions distinguishing old one and new one.

6. Acknowledgments

This work has been conducted in Ubila Project by Ministry of Internal Affairs and Communications (MIC).

References

- [1] D. Ashbrook and T. Starner. Learning significant locations and predicting user movement with gps. In *International Symposium on Wearable Computing*, Seattle, WA, October 2002.
- [2] L. Baum. An inequality and associated maximization technique in statistical estimation for probabilistic functions of markov processes. *Inequalities*, pages 3:1–8, 1972.
- [3] D. Patterson, L. Liao, D. Fox, and H. Kautz. Inferring high level behavior from low level sensors. In *Proceedings of the Fifth Annual Conference on Ubiquitous Computing (UBICOMP 2003)*, pages 73–89, 2003.
- [4] A. G. TOMMY GÄRLING, ROBERT GILLHOLM. Reinroducing attitude theory in travel behavior research. pages 129–146, May 1998.