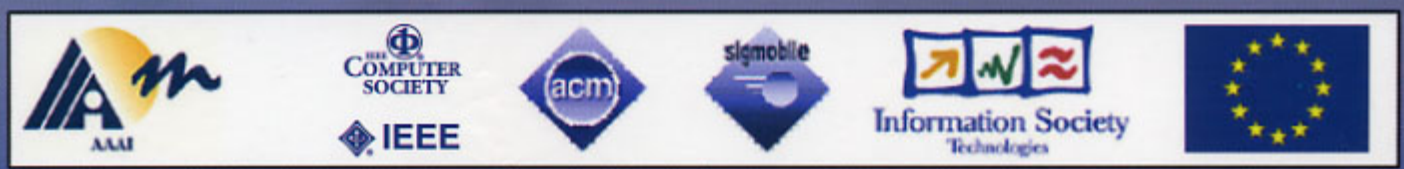
Proceedings of

MOBIQUITOUS 2004

The First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services

August 22-26, 2004 : Boston, Massachusetts, USA

General Chairs	Imrich Chlamtac, University of Texas at Dallas, USA
	Fausto Giunchiglia, University of Trento, Italy
General Vice Chairs	Valentina Tamma, University of Liverpool, UK
	Michele Zorzi, University of Padova, Italy
Program Chairs	Tim Finin, University of Maryland Baltimore County, USA
	Chiara Ghidini, ITC IRST, Trento, Italy
	Tom La Porta, Penn State University, USA
	Chiara Petrioli, University of Rome, "La Sapienza", Italy



IEEE Computer Society Order Number P2208
ISBN 0-7695-2208-4
Library of Congress 2004108490

Additional copies may be ordered from:

IEEE Computer Society
Customer Service Center
10662 Los Vaqueros Circle
P.O. Box 3014
Los Alamitos, CA 90720-1314
Tel: + 1 800 272 6657
Fax: + 1 714 821 4641
http://computer.org/cspress
csbooks@computer.org

IEEE Service Center
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331
Tel: + 1 732 981 0060
Fax: + 1 732 981 9667
http://shop.ieee.org/store/
customer-service@ieee.org

IEEE Computer Society
Asia/Pacific Office
Watanabe Bldg., 1-4-2
Minami-Aoyama
Minato-ku, Tokyo 107-0062
JAPAN
Tel: + 81 3 3408 3118
Fax: + 81 3 3408 3553
tokyo.ofc@computer.org

Individual paper REPRINTS may be ordered at: reprints@computer.org

Editorial production by Frances M. Titsworth

Cover art production by Joseph Daigle

Printed in the United States of America by The Printing House

Proc. of 1st Annual International Conference on Mobile and Ubiquitous Systems (MobiQuitous 2004), pp.392-400 (Aug. 2004).

Design and Implementation of an Extensible Rule Processing System for Wearable Computing

Masakazu Miyamae, Tsutomu Terada, Masahiko Tsukamoto and Shojiro Nishio
Graduate School of Information Science and Technology, Osaka University
Email: {miyamae, tsutomu, tuka, nishio}@ist.osaka-u.ac.jp

Abstract

In wearable computing environments, a user brings and uses his/her own computer to acquire various services wherever he/she goes. This type of computer, which acts as a service platform for wearable computing, needs autonomy and simplicity of services, flexibility of services/devices configuration, and power saving functions. Since most conventional wearable computing systems do not fulfill all of these requirements, we propose A-WEAR, which is a rule-based wearable computing system. We employ an event-driven rule as a behavior description language of A-WEAR to achieve autonomy and simplicity. Furthermore, we employ a plug-in mechanism to achieve flexibility and power saving. Using our system, we can easily provide and use various services for wearable computing environments.

1. Introduction

In recent years, the downsizing of portable computers has attracted much attention to the field of wearable computing. Wearable computing is a computing style in which a user brings and uses his/her own computer wherever he/she goes. Wearable computing has three characteristics: (1) *Hands-free*: users almost always browse information without hands because they wear the computer; (2) *Always on*: since the wearable computer is always powered while it is worn, users can use the computer whenever they want; (3) *Supporting daily life*: users utilize wearable computers to support activities in their daily life.

Figure 1 shows the appearance of a wearable computer. In the figure, the user wears a computer, a portable input device on her waist, and a HMD (Head Mounted Display). Since there is no restriction on users' behavior due to the 'Hands-free' features, many users will wear their own computer as a fashion accessory. Moreover, a wearable computer keeps watch on the user's activity and the surrounding environment because of the features of 'Always on' and 'Supporting daily life'.

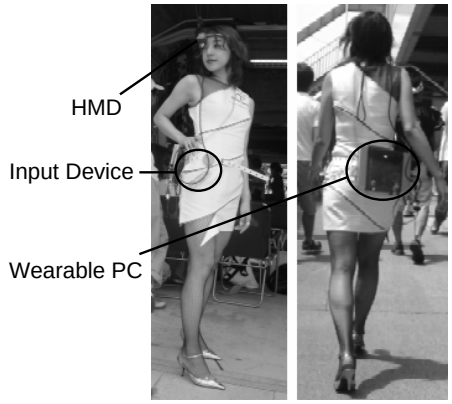


Figure 1. A style of wearable computing

Consequently, services for wearable computing will be developed as follows:

- Many facilities provide various services for wearable computers. For example, bookstores provide new book information according to users' interest, and amusement parks display waiting times of attractions by lending out high-accuracy location-aware devices.
- A user benefits from many wearable services regardless of whether the user is at work or at leisure. For example, a computer displays various information to support business as well as a shopping list when shopping.
- A user describes/customizes services such as a service that displays a message when the user meets a specific friend.
- A user customizes attached devices. For example, the user may remove a HMD when listening to music and may change old devices to new ones that are more accurate.

We define the following four properties as requirements for wearable systems to achieve the above service forms:

Autonomy: Services behave autonomously in response to information of user activities, situations, and environments around the user.

Simplicity: Users can easily create/modify services.

Flexibility: The system configuration can be changed flexibly by adding/deleting services and adding/swapping devices.

Power saving: The system has a mechanism to save battery life such as an automatic power on/off management of attached devices.

To construct a fundamental system for wearable computing that provides these properties, we propose A-WEAR (Active Wearable Engine Applying Rule-based architecture). A-WEAR offers autonomy, simplicity, and service flexibility by describing the system behavior with event-driven rules, and the device flexibility by using a plug-in mechanism to manage attached devices. Moreover, A-WEAR achieves power saving by combining rules and plug-ins to manage attached devices.

The remainder of this paper is organized as follows. We describe related works in Section 2, and Section 3 explains A-WEAR. Section 4 describes the design and implementation, and Section 5 shows the service examples using A-WEAR. Section 6 evaluates the system performance, and Section 7 presents our conclusions and future work.

2. Related Work

There has been much research on wearable systems such as MARS[1], TOWNWEAR[7] and VizWear[3]. These systems have achieved autonomy and provide high-quality services such as displaying virtual objects and annotations that overlap the real world on a HMD. On the other hand, since the services of these systems are specialized to one purpose and the device configuration is fixed, they lack flexibility of devices and services. They also lack simplicity and do not have any mechanism for power saving.

There are several service platforms for wearable computing. MEX[4] constructs an application by combining modules that provide various services and extends its functions by adding modules. NETMAN[2] is a service platform to construct applications by collaborating with other systems via a network. The system by Henk[5] is an event-driven system, which handles an input from sensors as an event. This system has power saving functions when no event occurs. Context Toolkit[6] is not for wearable computing, but it can simplify construction of application that uses context information. In Context Toolkit, since context widgets collect various information from environments and translate the information to an unified format, developers can construct context-aware applications without concern for the difference in sensors.

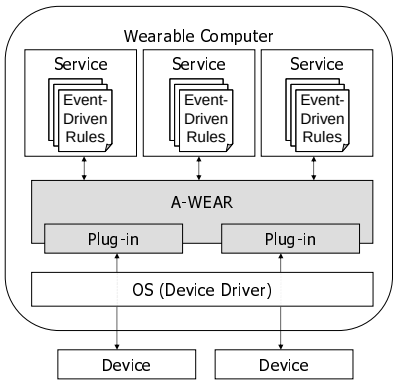


Figure 2. Block diagram of A-WEAR

Although MEX, the system by Henk, and Context Toolkit provide functions to describe autonomous services, NETMAN lacks this function. Moreover, not all of these systems can change the service configuration during execution, and only MEX can add/delete services while execution. Although MEX can also change devices dynamically, it cannot change devices that are directly attached to the computer because devices need to communicate with MEX via the network. Consequently, these systems do not have enough flexibility. The system by Henk has a mechanism for power saving. However, attached devices consume much power in wearable computing environments because this mechanism is only designed for the efficient use of a CPU. In addition, services of these systems are implemented using programming languages such as C and Java, which are not easy for general users.

3. A-WEAR

As mentioned in Section 2, there is no wearable system that provides all of the properties required for wearable systems. Consequently, we propose A-WEAR to meet this need.

As shown in Figure 2, A-WEAR manages devices via plug-ins for flexible device configuration, and services are described as a set of event-driven rules to make services autonomous and simple. We can configure services flexibly by adding/deleting/modifying rules dynamically. Moreover, A-WEAR achieves power saving functions with a combination of rules to keep watch on the system status and plug-ins for power management of attached devices.

The rest of this section explains the event-driven rules and the plug-in mechanism in detail.

DEFINE *Rule-ID*
[IN *List-of-belonging-groups*]
[FOR *Scope*]
[VAR *Variable-name* AS *Variable-type*] *
WHEN *Event-type* [(*Target-of-event*)]
IF *Conditions*
THEN DO *Actions*

Figure 3. Syntax of an ECA rule in A-WEAR

3.1. Service Description Language

All services are represented as a set of ECA rules in A-WEAR. An ECA rule is a behavior description language in an active database system, which is one of the database technologies. An active database processes the prescribed actions in response to an event arising inside/outside the database[8]. Each ECA rule consists of three parts: *Event*, *Condition*, and *Action*. The event part describes the event occurring in the system, the condition part describes the conditions for executing actions, and the action part describes the operations to be carried out. Using ECA rules, we can describe system behaviors in an event-driven manner. Moreover, since actions can generate new events, complex behaviors can be achieved by chaining ECA rules.

In conventional active databases, only database operations are considered events, such as *SELECT*, *INSERT*, *DELETE*, and *UPDATE*. Correspondingly, conventional active databases can carry out only database operations as actions. Therefore, to describe various services for wearable computing environments, we enhance the description capability of events and actions in an ECA rule. Moreover, we can manage a set of rules as a group since A-WEAR can run many services in parallel.

Figure 3 shows the syntax of an ECA rule for A-WEAR. In the figure, *Rule-ID* describes the name of the ECA rule. *List-of-belonging-groups* gives the name(s) of the group(s) that the rule belongs to. *Scope* describes the scope of rule execution. It is specified with *ENTIRE* (default) or *INTERNAL:Group-Name*. *ENTIRE* denotes the rule being evaluated in response to all occurring events, and *INTERNAL* denotes that the system processing the rule in response to occurring events that are generated by rules belonging to *Group-Name*. *Variable-name* describes the name of a local variable within this rule, and *Variable-type* specifies the type of local variable, whose designations are *I4* (32 bits integer), *R8* (64 bits floating point number), and *BSTR* (string). *Event-type* describes the name of the event that triggers this rule, and *Target-of-event* describes the target of event. For example, when *Event-type* is *Database Retrieval*, we can specify a target table for retrieval as *Target-of-event*. *Conditions* specifies conditions for executing the following actions. This part is described by the sequence of *<Exp 1> <Op 1> <Exp 2>* or *<Op 2> <Exp 3>*. *Exp 1, 2, 3* describe system parameters (described in Subsection 3.2) or constant values, and *Op 1* can be one of '==', '>', '<', '>=', '<=', and '!='. *Op 2* can be either of '?' (exists) or '!' (not exists). Complicated conditions can be described as a chain of conditions with AND and OR operators. *Actions* specifies an executing operation by describing the sequence of *<Act 1>* or *<Ret> = <Act 2>*. *Act 1, 2* describes the name of an action and its arguments, and *Ret* specifies the name of the local variable used to acquire the execution result of the action.

3.2. Plug-in Mechanism

A plug-in is an extension module for the system. In other words, we can use new events and actions by adding plug-ins. A-WEAR uses plug-ins not only for device management to defuse the differences in methods for using devices but also for enhancement of system functions such as e-mail functions and multimedia functions. Since plug-ins can be added/deleted dynamically on the system running, A-WEAR can change its functionality and device configuration dynamically (e.g. we can describe an ECA rule that rids the GPS plug-in temporarily if the user is indoors). Table 1 shows the functions provided by plug-ins that we have already implemented. In the table, *EVENT* means events provided by the plug-in and *ACTION* means provided actions. Additionally, each plug-in provides three system parameters: *NEW*, *OLD*, and *CURRENT*. NEW and OLD are provided for each occurring event, and they store a snapshot of information after/before the event occurs. Plug-ins provide current information as CURRNET in response to request from services. *CURRENT* in Table 1 means contents of CURRENT provided by the plug-in. These parameters can be used freely in the condition part and the action part of ECA rules. In the following, we describe the functions provided by plug-ins in Table 1.

Database Plug-in provides functions to access the user's database by SQL (Structured Query Language) and information about the query is set to NEW and OLD. Moreover, the system can refer to the contents of database by CURRENT. *Common Plug-in* provides functions generally used such as timer functions and loading/unloading plug-ins. *System information Plug-in* monitors the status of the wearable computer, and it raises the *SYS_POWER_CHANGED* event when the battery status changes. *Direction Plug-in* captures a user's direction using a motion sensor, and it raises the *ROTATE* event when user's direction changes. *Multimedia Plug-in* provides a function to play multimedia contents and set the current status of playing contents

Database Plug-in

Type	Name	Content
EVENT	SELECT	Acquire the data
	INSERT	Insert the tuple
	DELETE	Delete the tuple
	UPDATE	Update the tuple
ACTION	QUERY	Database operation
CURRENT	DB.*table.attribute condition*	Data matched to *condition*

Common Plug-in

Type	Name	Content
EVENT	CMN_START	System initialize
	CMN_TIMER	Timer expires
ACTION	COMM_EVENT	Generate specific event
	CMN_SET_TIMER	Set a timer
	CMN_KILL_TIMER	Kill a timer
	CMN_ADD_RULE	Insert rules
	CMN_LOAD_PLUGIN	Load a plug-in
	CMN_UNLOAD_PLUGIN	Unload a plug-in
CURRENT	NOW.TIME	Current time

System information Plug-in

Type	Name	Content
EVENT	SYS_POWER_CHANGED	Change of power status
	SYS_ADD_DEVICE	Addition of device
	SYS_REMOVE_DEVICE	Deletion of device
ACTION	SYS_STANDBY	Standby the computer
	SYS_SET_DEV_STATE	Enable/disable the device
CURRENT	SYS.BATTERY_STATUS	Remaining battery
	SYS.CPU_USAGE	CPU usage
	SYS.IS_IDLE	User's working status

Direction (motion sensor) Plug-in

Type	Name	Content
EVENT	ROTATE	Change of direction
ACTION	SET_DEFAULT	Initialize
CURRENT	DIR.ALPHA	Alpha direction
	DIR.BETA	Beta direction
	DIR.GAMMA	Gamma direction

Multimedia Plug-in

Type	Name	Content
EVENT	MM_END	Finish playing the contents
ACTION	MM_PLAY	Play the contents
	MM_STOP	Stop playing the contents
	MM_SEEK	Seek the contents
CURRENT	MM.IsPlaying	Get the status of whether the contents is playing

Camera Plug-in

Type	Name	Content
EVENT	N/A	N/A
ACTION	CAP_INIT	Initialize the Camera
	CAP_DEINIT	Release the Camera
	CAP_SAVE	Save the picture from camera
CURRENT	N/A	N/A

Current Position (GPS) Plug-in

Type	Name	Content
EVENT	MOVE	Change of position
ACTION	N/A	N/A
CURRENT	POS.LATITUDE	Latitude
	POS.LONGITUDE	Longitude

Map View Plug-in

Type	Name	Content
EVENT	N/A	N/A
ACTION	MAP_SET_CENTER	Set current position
CURRENT	MAP.LATITUDE	Latitude
	MAP.LONGITUDE	Longitude

Network Plug-in

Type	Name	Content
EVENT	NET_RECEIVE	Receive the data
	NET_FILE_RECEIVED	Finish receiving the file
	NET_FILE_SENT	Finish sending the file
ACTION	NET_UNICAST_SEND	Send the data
	NET_BROADCAST_SEND	Broadcast the data
	NET_FILE_SEND	Send the file
CURRENT	N/A	N/A

IRC Plug-in

Type	Name	Content
EVENT	IRC_RECEIVE	Receive the message
	IRC_JOIN	Join to the channel
	IRC_PART	Leave the channel
	IRC_ENUMUSER	Enumerate users
	IRC_DISCONNECTED	Disconnect the connection
ACTION	IRC_CONNECT	Connect to the server
	IRC_DISCONNECT	Disconnect the connection
	IRC_SEND	Send the message
	IRC_ENUMUSERS	Enumerate users
CURRENT	IRC.IsConnected	Connection status

E-Mail Plug-in

Type	Name	Content
EVENT	MAIL_RECEIVE	Receive the e-mail
	MAIL_SEND	Send the e-mail
ACTION	MAIL_INIT	Initialize the e-mail account
	MAIL_CHECK	Check new e-mails
	MAIL_SEND	Send the e-mail
CURRENT	N/A	N/A

Jog Dial Remote Controller Plug-in

Type	Name	Content
EVENT	JOG_ROLLUP	Roll up the jog dial
	JOG_ROLLDOWN	Roll down the jog dial
	JOG_BUTTONDOWN	Push down the jog dial
	JOG_BUTTONUP	Push up the jog dial
ACTION	JOG_SET_DISPLAY	Set the content of the display
CURRENT	N/A	N/A

Table 1. Details of plug-ins

to CURRENT. *Camera Plug-in* displays captured videos by the *CAP_INIT* action and saves captured images by the *CAP_SAVE* action. *Current Position Plug-in* tracks a user's position using GPS, and it raises the *MOVE* event when the user's position changes approximately 1.5 meters (this parameter can be changed in ECA rules). *Latitude* and *longitude* are provided as NEW and OLD. We can construct location-based services by using this plug-in. *Map View Plug-in* displays a map by the *MAP_SET_CENTER* action. *Network Plug-in* provides a function to send/receive data to/from other hosts via Internet. *IRC Plug-in* provides a function to send/receive message using the the IRC (Internet Relay Chat) protocol. *E-Mail Plug-in* provides a function to send/receive e-mail using the POP3 (Post Office Protocol version 3) and the SMTP (Simple Mail Transfer Protocol). *Jog Remote Controller Plug-in* provides a function to control the Jog Remote Controller.

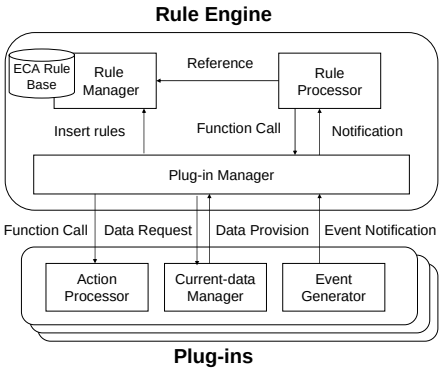


Figure 4. System structure of A-WEAR

4. Design and Implementation

4.1. System Design

Figure 4 shows the system structure of A-WEAR. *Rule Engine* is the core module of A-WEAR, and it consists of *Rule Manager*, *Plug-in Manager*, and *Rule Processor*. *Rule Manager* manages ECA rules. When new rules arrive, it analyses these rules, converts to the intermediate format, and stores them in *ECA Rule Base*. *Plug-in Manager* receives all requests for plug-ins and dispatches them to the appropriate plug-in. If a plug-in wants to notify an event to Rule Engine, Plug-in Manager receives the notification and sends the information to Rule Processor. When *Rule Processor* receives the notification of an event generated by a plug-in, it retrieves rules suited for this event from ECA Rule Base, evaluates the conditions of retrieved rules, and requests executing actions to plug-ins via Plug-in Manager. Rule Processor also requests plug-ins for getting CURRENT if necessary.

A plug-in consists of *Action Processor*, *Current-data Manager* and *Event Generator*. *Action Processor* executes actions in response to requests from Rule Engine. *Current data Manager* manages CURRENT data and provides data in response to requests from Rule Engine. *Event Generator* notifies events to Rule Engine. It also sets the contents of NEW and OLD.

4.2. Implementation

We implemented the prototype of A-WEAR and several plug-ins using Microsoft Visual C++ .NET 2003 Enterprise Architect on SONY PCG-C1MZX. We can use many programming languages such as Microsoft Visual Basic, Visual C++, and C# for implementing plug-ins because they are implemented as COM (Component Object Model) objects. Figure 5 shows a snapshot of using the building information service described in Figure 6. The user equips Xybernaut MA-V as a wearable computer, MicroOptical CO-3 as a HMD, I-O Data CFGPS as a GPS device, NEC-tokin MDP-A3U7 as a geomagnetic sensor, and a portable mouse as an input device.

Figure 5. A snapshot of using the prototype system

```
DEFINE FIND-BUILDING
WHEN MOVE
THEN DO MAP_SET_CENTER(NEW.LATITUDE,NEW.LONGITUDE)
  DO QUERY ('
   SELECT * FROM GEODATA WHERE  (
    ABS (%NEW.LATITUDE%-X) < 0.005 AND
    ABS (%NEW.LONGITUDE%-Y) < 0.005) ')

DEFINE DISPLAY-WEBPAGE
WHEN SELECT (GEODATA)
IF ?MAP.EXIST (%POS.X%;%POS.Y%;%NEW.X%;%NEW.Y%;
  %DIR.ALPHA_NORTH%;100.0;20.0)
THEN DO BROWSER_OPEN (NEW.URL)
```

Figure 6. An example of ECA rules

5. Service Examples

In this section, we describe the service examples of A-WEAR. First, we show the building information service to show simplicity of A-WEAR service. Next, we show the information delivery service to show the function for the dynamic changing of configuration of rules and plug-ins. Finally, we show other service examples in practical use.

5.1. Building Information Service

Figure 6 shows an example of ECA rules for a building information service that displays web pages of the building in front of the user. This service consists of two rules. *FIND-BUILDING* rule searches the building near the user's current position when the *MOVE* event occurs. *DISPLAY-WEBPAGE* rule extracts the buildings in the direction of the user from the search results and displays the web pages of the extracted buildings when *FIND-BUILDING* performs the data retrieval.

```
DEFINE ADD-RULE
WHEN NET_END_FILE_RECEIVE
THEN DO CMN_ADD_RULE (NEW.FILE)

DEFINE ENUM-DISABLING-DEVICE IN EnumDeviceGroup
WHEN SYS_ADD_DEVICE
THEN DO QUERY ('SELECT * FROM DeviceTable
    WHERE TYPE = %NEW.TYPE% AND DISABLED = 0')
  DO QUERY ('INSERT INTO DeviceTable (NAME, TYPE,
    DISABLED) VALUES (%NEW.NAME%, %NEW.TYPE%, 0)')

DEFINE ENUM-ENABLING-DEVICE IN EnumDeviceGroup
WHEN SYS_REMOVE_DEVICE
THEN DO QUERY ('SELECT * FROM DeviceTable
       WHERE TYPE = %NEW.TYPE% AND DISABLED = 1')
     DO QUERY ('DELETE * FROM DeviceTable
       WHERE NAME = %NEW.NAME%')

DEFINE SET-DEVICE-STATE
  FOR INTERNAL:EnumDeviceGroup
WHEN SELECT (DeviceTable)
THEN DO SYS_SET_DEV_STATE (NEW.NAME, #1 -
NEW.DISABLED#)
  DO QUERY ('UPDATE DeviceTable SET DISABLED =
    #1 - NEW.DISABLED# WHERE NAME = %NEW.NAME%')

DEFINE DISPLAY-MESSAGE
WHEN NOTIFY_MESSAGE
THEN DO CMN_DISPLAY_MESSAGE (5, NEW.MESSAGE)
```

Figure 7. Initial rules in the system

5.2. Information Delivery Service

We describe an information delivery service for an amusement park as an example of A-WEAR services. A user visits an amusement park with his wearable computer and borrows an infrared sensor when he enters the park. The park disseminates a plug-in for infrared sensor, rules for navigation, and information about attractions. There are infrared transmitters around the park, and the use can get highly detailed location information by using the infrared sensor. Based on this information, the system provides information about attractions near him.

When the computer uses an infrared sensor in this service, it does not need other location-aware devices. Therefore, it disables these devices to improve power consumption.

All of these processes are performed by ECA rules. These rules can be classified into initial rules and rules provided in the amusement park:

- **Initial rules in the system**
  Figure 7 shows the initial rules in the system. *ADD-RULE* rule stores the received rules via the network automatically. When a device is attached to the wearable computer, *ENUM-DISABLING-DEVICE* rule retrieves the devices of the same type as the attached device and records the device information to the database. Conversely, when a device is removed, *ENUM-ENABLING-DEVICE* rule retrieves the devices of same type as the removed device. *SET-DEVICE-STATE* rule is activated when attached devices are searched from the database, and it enables/disables these devices according to the content of the database. *DISPLAY-MESSAGE* rule displays messages for five seconds when other rules request notification of a message to the user.

```
DEFINE REQUEST-PLUGIN IN AmusementParkGroup
WHEN CMN_ADD_RULE (AmusementPark.eca)
THEN DO NET_UNICAST_SEND ('www.osaka-u.ac.jp:
          GET_PLUGIN;FILE:RF_SENSOR.DLL')

DEFINE LOAD-PLUGIN IN AmusementParkGroup
WHEN NET_END_FILE_RECEIVE
IF NEW.COMMAND == 'PLUGIN'
     AND NEW.FROM == 'www.osaka-u.ac.jp'
THEN DO CMN_LOAD_PLUGIN (NEW.FILE)

DEFINE SAVE-LOCATION-INFO IN AmusementParkGroup
WHEN NET_RECEIVE
IF NEW.COMMAND == 'LOCATION'
     AND NEW.FROM == 'www.osaka-u.ac.jp'
THEN DO QUERY ('INSERT INTO AmusementParkTable
           (X, Y, MESSAGE) VALUES (%NEW.X%,
            %NEW.Y%, %NEW.MESSAGE%)')

DEFINE SEARCH-LOCATION-INFO IN AmusementParkGroup
    FOR INTERNAL:AmusementParkGroup
WHEN MOVE
THEN DO QUERY ('SELECT * FROM AmusementParkTable
       WHERE %NEW.X% - 10 < X AND X < %NEW.X% + 10
       AND %NEW.Y% - 10 < Y AND Y < %NEW.Y% + 10')

DEFINE NOTIFY-USER IN AmusementParkGroup
WHEN SELECT (AmusementParkTable)
THEN DO NOTIFY_MESSAGE ('MESSAGE', NEW.MESSAGE)

DEFINE SERVICE-TERM IN AmusementParkGroup
WHEN SYS_REMOVE_DEVICE
IF NEW.NAME == 'RF_SENSOR'
THEN DO CMN_UNLOAD_PLUGIN ('RF_SENSOR.DLL')
     DO CMN_REMOVE_RULE ('AmusementParkGroup')
```

Figure 8. Rules provided in the amusement park

- **Rules provided in the amusement park**
  Figure 8 show the rules provided in the amusement park. *REQUEST-PLUGIN* rule sends the request for the infrared sensor plug-in to the server when this rule is stored in the system. The infrared sensor plug-in receives the information via the infrared transmitters, calculates the user's location, and raises a *MOVE* event. *LOAD-PLUGIN* rule installs the plug-in to the system when it receives a plug-in from the server. *SAVE-LOCATION-INFO* rule records the information to the database when the system receives the information about attractions. *SEARCH-LOCATION-INFO* rule retrieves the information of attractions within 10 meters from the user, and *NOTIFY-USER* rule notifies

Figure 9. Pit crews using the VIBRaSS

the retrieved information to the user. *SERVICE-TERM* rule unloads the infrared sensor plug-in and removes rules for the amusement park when the user removes the infrared sensor.

In this service, *NOTIFY_MESSAGE* action chains the *DISPLAY-MESSAGE* rule to display a message. This rule is for users wearing a HMD, because it displays a message on the HMD. If the user does not want to use any HMD, he can remove the HMD and change *CMN_DISPLAY_MESSAGE* to an action for reading out the contents because A-WEAR can change the service configuration dynamically.

In our system, since rules disseminated via the network can be added to the system automatically, we should filter received data to improve the security of A-WEAR. We think that it can be realized by attaching a tag that represents information on the sender and service operations. As a result, users can flexibly describe ECA rules for filtering received services by a data receiving event, tag checking conditions, and rule storing actions.

5.3. Other Applications

We have implemented many wearable applications for A-WEAR in practical use. Here, we introduce two of them.

The first application is VIBRaSS (VIsual Bike Race Support System), which helps the team managers and pit crews to show various information dynamically in a motorbike race. This application shows the status of the race such as current position, rivals' positions, remaining fuel, and expected pit-in-time calculated from the fuel consumption and lap times. Moreover, it notifies various events such as the change of current position. This application consists of approximately ten rules, and we use several plug-ins such as IRC plug-in, Network plug-in, Multimedia plug-in, and Race-Specific plug-in. This application has actually been used by two teams in the Suzuka 8 Hours World Endurance Championship Race 2003, which is a famous international motorbike race held in Japan. Figure 9 shows a screen shot of the application and photographs of pit crew members using our system.

Figure 10. MCSS used in WPC EXPO 2003

Device Type	Device Name/Spec
Wearable computer	CASIO FIVA MPC-216XL (Max. power consumption 45W)
CPU	Transmeta Crusoe TM5600 (600MHz)
GPS	I-O data CFGPS
Geomagnetism sensor	NEC-tokin MDP-A3U7
Wireless LAN adapter	Melco WLI-PCM-L11

Table 2. Specs of devices for evaluation

The second application is MCSS (Master of Ceremonies Support System), which helps the MC to facilitate his/her work. Using this application, since the director can show scripts and indications on the MC's display, the MC does not need to memorize all of the lines while on stage. This application consists of approximately fifteen rules, and we use IRC plug-in and WWW Browser plug-in to send messages via a wireless network. This application has been used at the main stage in WPC EXPO 2003, which is the premier exhibition of IT products in Asia. Figure 10 shows a snapshot of two MCs using our system in the expo.

6. Evaluation

In this section, we evaluate the effectiveness of the selective use of devices, which is one of the key features of

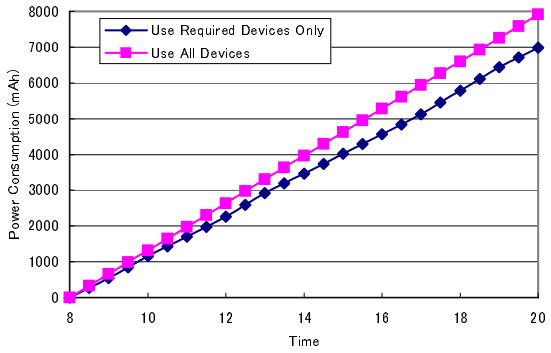


Figure 11. Effect of selective use

A-WEAR.

We assume the following scenario as an example of daily-life with a wearable computer and examine its power consumption.

- A user goes to the office by train for an hour from 8:00 a.m. At this time, his wearable computer plays the user's favorite songs automatically. The computer disables the private rules and switches to business rules automatically at 9:00 a.m., which is the start time of the user's job. He is a sales representative, and the computer automatically collects information on the customers he plans to visit today via a wireless LAN until 10:00 a.m. He takes 30 minutes to get to a customer's location and he does business for 30 minutes. The user repeats this procedure from 10:00 a.m. to 12:00 a.m. and from 1:00 p.m. to 5:00 p.m. While the user does business, the computer turns on the GPS device and navigates to the destination when it detects his movement by the acceleration sensor. If the system detects arrival at the destination, it displays the sales information to help him. At 12:00 a.m., he requests for navigation to a restaurant for lunch. From 5:00 p.m., the system navigates him to help with his shopping. While shopping, the computer provides the building information service described in Section 3 by using GPS, geomagnetism sensor, and wireless LAN. The user takes an hour to return to his home from 7:00 p.m.

We evaluated the battery consumption with/without the mechanism of selective use of devices according to the above scenario. We performed the evaluation with the devices shown in Table 2 and several 2000 mAh batteries. Figure 11 shows the results of the evaluation. The figure shows that the selective use of devices saves battery life by 1000m Ah (12%) in one day. In this evaluation, we used three devices for the target of selective use. Since generally more devices are used in wearable computing environments, the advantage of the plug-in mechanism will become even clearer.

7. Conclusion

In this paper, we proposed A-WEAR, which is an extensible rule processing system for wearable computing. A-WEAR uses ECA rules to describe autonomous services simply, and it uses plug-ins to make device configuration flexibly. We can easily construct and customize applications for wearable computing environments using A-WEAR. We also showed that A-WEAR significantly reduced power consumption.

Although we used Microsoft Windows for the platform of the prototype system in our research, our approach is independent of the operating systems used. In the future, we aim to implement an OS that can execute ECA rules directly. Moreover, we will propose strong security mechanisms. A-WEAR can receive and store ECA rules via a network. Although the system can filter undesired rules by using tags, the tags may be interpolated by the sender. Therefore, it is desirable to filter rules by referring to the contents of the received rules.

Acknowledgments

This research was supported in part by "The 21st Century Center of Excellence Program", Grant-in-Aids for Scientific Research number 13780331 from the Japan Society for the Promotion of Science and Special Coordination Funds for promoting Science and Technology of the Ministry of Education, Culture, Sports, Science and Technology, Japan.

References

[1] T. Hollerer, S. Feiner, T. Terauchi, G. Rashid, and D. Hallaway. Exploring mars: Developing indoor and outdoor user interfaces to a mobile augmented reality system. *Computers and Graphics*, 23(6):779–785, 1999.

[2] G. Kortuem, M. Bauer, and Z. Segall. Netman: The design of a collaborative wearable computer system. *Mobile Networks and Applications*, 4(1):49–58, 1999.

[3] T. Kurata, T. Okuma, M. Kourogi, T. Kato, and K. Sakaue. Vizwear: Toward human-centered interaction through wearable vision and visualization. In *PCM2001 in Beijing, China*, pages 40–47, 2001.

[4] J. Lehikoinen, J. Holopainen, M. Salimaa, and A. Aldro-Vandi. Mex: A distributed software architecture for wearable computers. In *ISWC '99 (Third Int. Symp. on Wearable Computers)*, pages 52–57, 1999.

[5] H. Muller and C. Randell. An event-driven sensor architecture for low power wearables. In *ICSE 2000, Workshop on Software Engineering for Wearable and Pervasive Computing*, pages 39–41. ACM/IEEE, 2000.

[6] D. Salber, A. K. Dey, and G. D. Abowd. The context toolkit: Aiding the development of context-enabled applications. In *CHI '99 (The 1999 Conference on Human Factors in Computing Systems)*, pages 434–441, 1999.

[7] K. Satoh, K. Hara, M. Anabuki, H.Yamamoto, and H.Tamura. Townwear: An outdoor wearable mr system with high-precision registration. In *2nd Int. Symp. on Mixed Reality*, pages 210–211, 2001.

[8] J. Widom and S. Ceri. *ACTIVE DATABASE SYSTEMS*. Morgan Kaufmann Publishers Inc, 1996.