Toward Achieving On-Site Programming

Tsutomu Terada
Kobe University
1-1 Rokkodai, Nada Ward
kobe, Hyogo 657-8501, Japan
Email: tsutomu@eedept.kobe-u.ac.jp

Masakazu Miyamae
Westunitis Co., Ltd.
Funakoshi-cho 2-4-6-3, Chuo Ward
Osaka 540-0036, Japan
Email: miyamae@westunitis.co.jp

Abstract

Wearable computing, where users can wear computers anywhere and at any time, is slowly becoming a reality because of recent technological advancements. When users access various services in wearable computing environments, they want to define new services by themselves. Therefore, we propose a new framework for constructing context-aware applications in wearable computing environments. Our framework, called Wearable Toolkit*, consists of an event-driven rule processing engine and tools to develop applications. By using our framework, we can define and customise services anywhere and at any time. We found our system helps to create context-aware wearable services from the results we obtained by evaluating it.*

1. Introduction

The down sizing of portable computers has attracted a great deal of attention to the field of wearable computing. There are many wearable computing systems such as those in health care that monitor user behaviours[1], information-presentation systems for motorbike races[2], context-aware man navigation systems[3], and systems for supporting assembly and maintenance tasks[4]. Wearable computing can provide effective and attractive services compared with the use of conventional desktop/mobile devices. People's daily lives can especially be improved by wearable computing technologies because wearable computers have detailed information about their users. However, to provide effective services that are highly personalised, the system needs new models for constructing services and programming.

We focused on end-user programming for wearable computing in this research. Wearable computer users frequently want to construct trivial services. For example, when someone in a bookstore wants to buy a book but they do not have enough money, they want to construct a reminder service where "Alert to buy it when they are next in close proximity to the bookstore". In this definition, the condition (in close proximity to the bookstore) and the action (alerts to buy the book) should be defined dynamically at the site where they discovered the idea of service. Therefore, wearable systems need a mechanism that will describe services on site. Moreover, since these services are advantageous to the general public who are not familiar with programming, the system needs to provide an interface enabling them to implement services easily.

We propose *Wearable Toolkit* for users to achieve on-site programming. It includes a rule processing engine and several tools, and we can easily construct various services without any knowledge of context awareness or programming languages. We carefully extract the requirements for the system to attain on-site programming, and we implement the extracted functions in the system. Several people actually used our mechanism to construct wearable services, and we demonstrated the effectiveness of the system through actual use and results obtained from evaluation.

The remainder of this paper is organized as follows. Section 2 describes the requirements for on-site programming and presents related work. Section 3 explains the design of the proposed framework, and Section 4 describes the context-definition tool. Section 5 explains the actual use of our system and some evaluation results that confirm its effectiveness. Section 6 is the conclusion and also discusses future work.

2. Requirements for On-site Programming

Below are other examples for *programming on site*.

- Someone wants to display a map for a destination on HMD when they go on a trip. Since it is annoying to always display the map, they construct a service where the map is only shown on HMD when they are standing.
- Since sitting for many hours is not healthy, someone can implement a service that reminds them to have a brisk workout every two hours.
- Since people often forget where they put their cell phone, they can construct a service to remember where and when they last removed it from pocket.
- When users receive an email from someone who is not important, they can stop future emails from that person being shown on their HMD.

Note that we define services as parts of applications on a wearable computer. To accomplish this kind of service programming in wearable computing environments, the system needs to fulfil three requirements:

1) *A programming model for easy implementation*
2) *Dynamic modifications to services when system running*
3) *On-site context definitions*

Since users implement services anywhere and at any time, the system should allow them to implement them simply and easily to fulfil the first requirement. The services in wearable computing are usually expressed as a set of events (e.g., arrival at a bookstore) and action (e.g., showing a map). This means that an event-driven architecture is suitable for describing services for wearable computing. Moreover, since complex descriptions are not suited to on-site programming, we should employ a simple but flexible model.

Requirement 2 means that it is important not to stop the system even when new services are installed. Users in on-site programming environments frequently add/delete/modify services. However, there are important services running in the system, such as a service for sensing vital information.

The last requirement means that the system needs a function to define a context, which becomes a trigger to activate a service, easily on site. As the above service examples, there are various contexts triggering services and it is difficult to define all possible contexts beforehand. Therefore, the system needs a function to define contexts according to current situation.

The purpose of our research was to construct a system platform that fulfils these three requirements. However, there have been many activities on toolkits for constructing context-aware applications for wearable/ubiquitous computing. One representative example is the MIThril project that was aimed at constructing a platform for wearable computing[5]. It provides APIs to support context-recognition and hardware management. In addition, the Context-Toolkit[6] is a well-known toolkit for constructing context-aware applications. Using three layerd model in Context-toolkit, a programmer can construct context-aware applications without considering the change in using sensors. TEA System[7] can generate a threshold to recognize user context from stored information by carrying with a sensor-enabled TEA board. The main purpose of these systems is to reduce the load on application programmers by abstracting acquired sensor data, and they do not fulfil our requirements. It is difficult for the general public to implement services on site even if they use such platforms.

As an example of end user programming, a CAPpella is notable system[8]. It achieves an end user context definition like our context tool. However, since this system needs to decide recognition window and to select sensors when defining a context, it is difficult to use with many sensors or complex contexts. In other words, when using a CAPpella for defining contexts supposed in this paper, the recognition rate is low as shown in *Beginner-Manual* in Table 6 because a CAPpella is like the manual mode of our context tool. Moreover, programming model and dynamic modification are not cosidered in a CAPpella.

3. Wearable Toolkit

Here, we propose a new platform that fulfils the three requirements. We call the platform *Wearable Toolkit*, which includes a rule processing engine and related tools for on-site programming, as shown in Fig. 1. *Rule Engine* is the most important part of this toolkit, which is an engine for processing event-driven rules. *GUI/Text based Rule Editor*s are tools for defining rules. *GUI builder* and *Flash GUI Loader* are tools for designing GUIs. *Debugger* is a monitoring tool for Rule Engine, and *Context Definition Tool* is an on-site context definition tool. The *Plug-in Development Kit* is a development kit for programmers who want to enhance the functions of Rule Engine directly. People usually use *Rule Engine*, *Context Definition Tool*, and *GUI based Rule Editor* for on-site programming.

3.1. Rule Engine

Rule Engine is a core part of the toolkit and Fig. 2 has a processing image of it. It works as a middleware on Windows OS, and it manages wearable devices via plug-ins to enable flexible configuration of devices. Services are described as a set of event-driven rules to make services autonomous and simple. Note that this

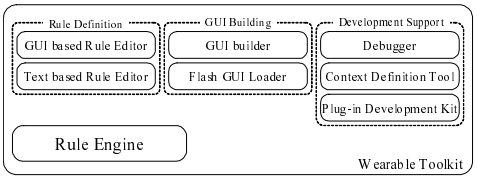

Figure 1. Structure of Wearable Toolkit

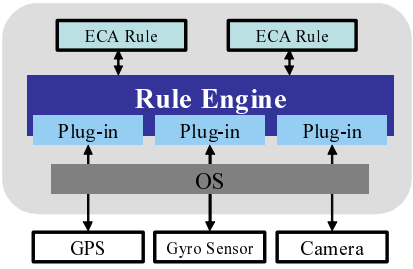

Figure 2. Processing image of Rule Engine

Rule Engine is an enhanced version of the wearable system we proposed in earlier research [9].

All services are represented as a set of ECA(Event, Condition, and Action) rules, which is a behaviour-description language in database systems. Each ECA rule consists of three parts: an *Event*, a *Condition*, and an *Action*. The first describes an event occurring in the system, the second describes the conditions for executing actions, and the third describes the operations to be carried out. Using ECA rules, we can configure services flexibly and simply by adding/deleting/modifying rules dynamically, and this means our system fulfils requirements (1) and (2). Moreover, since actions can generate new events, complex behaviours can be achieved by chaining ECA rules.

Figure 3 shows the syntax of ECA rule, where *Event-type* describes an event name that triggers this rule. *Conditions* specifies those for executing the following actions, and we can use AND/OR operators in Conditions to describe complicated conditions. *Actions* specifies executing operations. We can also use 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. The system provides current information as CURRENT. Events and actions are defined by plug-ins, which are extension modules. In other words, we can use new events and actions by adding plug-ins. By employing such a plug-in mechanism, we can enhance functions such as adaptation to a new device by adding a plug-in.

Table 1 lists some of the already implemented plug-ins, and Table 2 lists details on GPS plug-ins and System-info plug-ins as examples. Note that our plug-in mechanism allows multiple plug-ins to generate the same event. For example, all direction tracking devices (e.g., a geomagnetic sensor and a gyro sensor) can generate ROTATE events when they detect a change in directions. The system can handle user activity in a common format as a result of this abstraction mechanism. We can add functions to the system by implementing a new plug-in in C++, Java, Flash, and all .NET languages such as C#.

Figure 4 shows an example of a service description.

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

Figure 3. Syntax of ECA rule

Table 1. Implemented plug-ins

Name	Function
Common	Providing basic functions such as timer processing
GUI	Managing user defined GUI
Database	Supporting database queries
GPS	Detecting moving position
System info	Providing PC information
Multimedia	Playing multimedia content
Serial Com	Transmitting data via serial port
Camera	Capturing images by camera
Network	Transmitting data via Internet
Mail	Sending/retrieving email
Browser	Controlling WWW browser
Map view	Showing a map
VCode	Recognising a visual marker
RF-ID	Recognising an RF-ID tag
Direction	Detecting current direction
IRC	Providing IRC-based communication
Skype	Controlling Skype function

Table 2. Details on various plug-ins

Current Position (GPS) Plug-in

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

System information Plug-in

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

```
DEFINE StandingContext
 WHEN SKYPE_INCOMING
 IF GLOBAL.CONTEXT == 'stand'
 THEN DO CMN_SHOW_QUESTION(NEW.ID,
         'Call from NEW.DISPNAME.Connect?')

DEFINE SkypeAccept
 WHEN CMN_ANSWER
 IF ?NEW.RESULT
 THEN DO SKYPE_HANDLE(NEW.ID, 'INPROGRESS')

DEFINE RunningContext
 WHEN SKYPE_INCOMING
 IF GLOBAL.CONTEXT == 'running'
 THEN DO SKYPE_HANDLE(NEW.ID, 'RECORD')
```

Figure 4. Example of ECA Rule

These three rules achieve a context-aware phone service. When the system detects an incoming call and the current context of user is "stand", the *StandingContext* rule and the *SkypeAccept* rule notify of the incoming call and post a query if the user answers the call. The *RunningContext* rule permits the answering machine to pick up when the user is running.

3.2. Context Definition Tool

To implement the services on site, the system should provide a function where a user can define a new context as an event. As defined in previous researches, *context* means any information that can be used to characterise the situation of entities[6]. Context can be defined in various expressions such as 'walking', 'riding a bike', 'awake', '10 minutes from now', and 'receiving email'. Most contexts, especially in wearable computing, are determined by characteristic values from multiple sensors. Here, 'sensors' include not only hardware sensors such as acceleration sensors, but also mail clients that detect mail arriving. The characteristic value is that calculated from raw sensing data such as average, variance, maximum value, minimum value, and DP-matching results. For example, 'current position is a train station' can be translated as the current values of latitude and longitude acquired from GPS are sufficiently close to the values already registered as the position of the station. Table 3 lists examples of contexts and characteristic values that explain these contexts. A user must determine suitable sensors and the characteristic values for each sensor to define a context correctly, and it is also difficult for the general public to determine the window size for calculating characteristic values.

Our toolkit has a context definition tool to resolve the difficulty with context definitions. Fig. 5 shows a snapshot of the tool, where the acquired sensor data are shown at the left in real time. A programmer directly selects the window size in this area when defining a context, and he/she selects one or multiple characteristic values from the list at the right-hand neighbour. The other side has an interface to register context and presents the real-time recognition results. This tool can manage various types of sensors at the same time, and it is easy to add a new sensor by adding its definition. It is managing three 3-axis acceleration sensors, a direction sensor, an angular velocity sensor, a GPS, a temperature sensor, a humidity sensor, a timer, and an RF-ID tag reader in the figure.

When a user who knows about context defines the current situation as a context using this tool, the procedure for context definition is as follows:

1) The user pushes the 'stop' button to take a snapshot of the current situation.
2) The user selects appropriate sensors suited to recognizing the intended context.
3) The user determines the window size and a set of characteristic values for each selected sensor.
4) The user pushes the 'register' button and gives the context a name.

Compared with the conventional process to define a context, we can directly select sensors, window size, and characteristic values with GUI, and defined

Table 3. Contexts and characteristic values

Contexts	Sensors and characteristic values
Staying in a certain place	Current value from GPS
In transit	Variation in values from GPS
Rotation	Variation in values from gyro sensor
Standing	Average and variance values from
Walking	multiple acceleration sensors for a
Riding on bike	certain period of time
Mail reception	Current situation of email client
Removing wallet from pocket	Current value of reading RF-ID tag
Noon	Current time
Hot	Current value of temperature sensor
Getting warm	Variation in values from temp. sensor

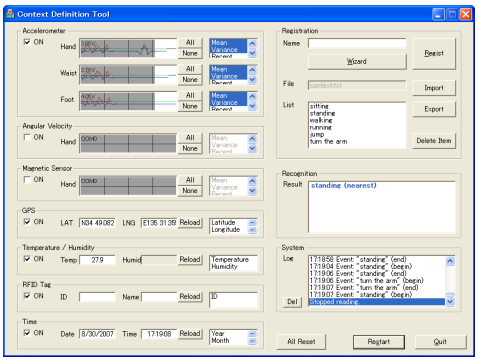

Figure 5. Snapshot of context definition tool

contexts can be used as events in the ECA rule for the rule engine. The ease with which context-aware applications are constructed is drastically improved with this tool. However, it is still difficult for the general public to define a context, especially with Steps 2 and 3. Consequently, we propose an automatic selection algorithm for sensors, characteristic values, and learning window size. Using our proposed algorithm, the procedure is reduced to the following three steps:

1) The user pushes the 'stop' button.
2) He/she answers a few questions from the system.
3) He/she gives the context name.

Note that the questions in Step 2 are not too technical. Therefore, anyone can easily define a context using our tool. This function is attained by a combination of variance-based situation clustering and a question-answering interface.

Variance-based situation clustering. The system needs to recognize the window for the user's intended context for automatic context definition. As far as the acceleration sensors are concerned, three types of situations have been considered.

1) *Static conditions* such as sitting: the sensor data does not change so much. Suitable characteristic values are the instantaneous value or the average.
2) *Continuous movements* such as walking: the sensor data changes according to certain patterns. Suitable characteristic values are the average and the variance.
3) *Short duration motions* such as jumping and gestures: the sensor data changes irregularly. Suitable characteristic values are the raw waveform for DP matching.

Considering these situations, the system extracts the window for learning in three steps:

1) Calculate variance v in the short period.

$$v(t,t-\delta)=\frac{1}{\delta}\sum_{k=t-\delta}^{t}x(k)^2-\left(\frac{1}{\delta}\sum_{k=t-\delta}^{t}x(k)\right)^2 \quad (1)$$

2) Calculate difference $d(t)$ between two sequential periods.

$$d(t)=v(t+\delta,t)-v(t,t-\delta) \quad (2)$$

The points where $d(t)$ is larger than threshold T_d become candidates for the boundary of the learning window.

3) The system extracts a minimal window larger than threshold T_w.

Next, by evaluating the length of the extracted window and the variance in data in the window, the system clusters the extracted situation into one of the three situations. More concretely, if the window is short enough, the system rates the situation as a short term motion. Otherwise, if the variance in data in the window is sufficiently small, it rates the situation as a static condition.

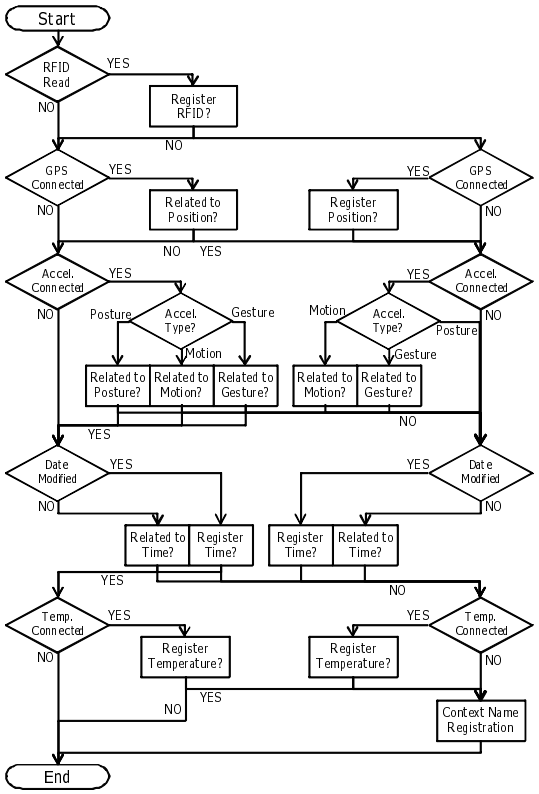

Figure 6. Example of question flow

Question-answering interface. Where a user takes some action such as making a gesture or reading an RF-ID tag, it is easy to specify the sensors for representing contexts. However, there are several sensors the system cannot find if the user's intended context includes the value of the sensors. For example, it is difficult to estimate if sensors whose values do not change drastically, such as GPS and temperature sensors, should be included for calculating contexts.

Therefore, we employed a question-answering interface. The system poses a few questions to the user to specify the sensors to be extracted. Researchers on context-awareness prepare questions, and the system dynamically configures the flow of questions taking the combination of sensors into consideration. Figure 6 shows the question flow when acceleration sensors, a GPS, a temperature sensor, and an RF-ID tag reader are used. Note that as users only need to answer Yes/No, this is not difficult for the general public.

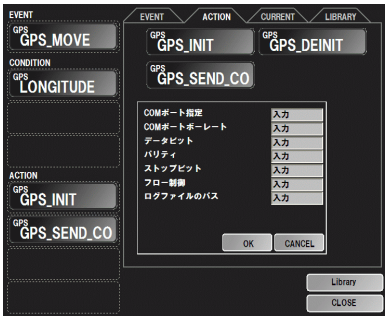

Figure 7. GUI-based ECA rule editor

Figure 8. Snapshots of Mobile Nature Rally

The system can define a context automatically by using these functions.

3.3. Rule Editor

We have provided two types of ECA rule editors, i.e., a text-based editor and a GUI-based editor. The text-based editor enables users to directly input ECA rules. Experienced people can rapidly implement applications using this editor by using several supporting functions such as detecting syntax errors in real time and pop-up displays for inputting candidates. However, a user can construct services by merely clicking a panel and inputting various parameters in a dialog box using the GUI-based editor shown in Fig. 7. This makes it easy for general public to describe rules.

4. Implementation and Evaluation

We implemented a Rule Engine and several tools mainly with Visual C++ on a Windows 2003 server edition. Most tools were implemented as plug-ins of the Rule Engine. All tools were propagated at the Wearable Toolkit Web site[10].

4.1. Actual Use in Several Events

To demonstrate the flow for developing applications, we planned one large social event called the 'Mobile Nature Rally in EXPO'80 park', where users walked around certain locations. Each location had a unique Visual Code (See Fig. 8). They read a code on their computer, and a quiz for the location was displayed. After answering the quiz, the system recommended the next destination. The purpose of this game was to answer five questions and reach the goal. Participants who reached the goal received certification that included the quiz results, the consumed calories, and the route they had walked.

We implemented this service in six steps. Note that most parts of the process were done by one student who had almost no experience in programming.

1) *Functional Design* As noted above, the system needs to use four functions.
   (a) A function to record the position to obtain his walking route
   (b) A function to recognize user movement to accurately calculate his calorie consumption.
   (c) A function to control the system by using gestures.
   (d) A function to open a Web page according to a visual code being watched.
2) *Device Configuration* Considering the functions, he employed a wearable PC, a HMD, a camera, a GPS, and two 3-axis acceleration sensors.
3) *Plug-in Selection* He picked the following plug-ins to achieve the required functions; VCode, browser, SerialCom, and GPS.
4) *Description of ECA Rules* He described 25 rules to achieve all functions. Figure 9 shows a part of the described rules
5) *Context Learning* Using the context tool, he registered several contexts such as walking, standing, sitting, and raising his right hand.
6) *Debug* He debugged the implemented service by actually using the system and debugger. He described the rules on site to modify the functions.

It took only 10 hours to complete all processes. Although a person who has no experience in programming cannot implement such a complex system with a conventional toolkit and middleware, the Wearable Toolkit enables anyone to implement the system by only describing the 25 rules. The system worked well and there were hundreds of participants in our event (it was reported in a national newspaper). From this actual demonstration of its use, it was clear that our toolkit was sufficiently practical and user-friendly.

```
//Jump to web site according to detected v-code
DEFINE VisualCodeJump
 WHEN QR_DETECT
 THEN DO GUI.Main.Browser.URL = STRING('NEW.CODE')

//Browser control by gesture
DEFINE ScrollDown
 WHEN CONTEXT_RECOGNIZED(RightHandUp)
 THEN DO WEB_SCROLL_BY('Main', 'Browser', 0, -50)

//Send context information to the server
DEFINE SendContextInfo
 WHEN CONTEXT_RECOGNIZED
 THEN DO NET_SEND(['GLOBAL.SERV', 'NEW.CONTEXT'])
```

Figure 9. Example of implemented ECA Rules

Table 4. Time to implement services (min)

Subject \ Service	1	2	3
A	< 10	< 10	25
B	30	90	55
C	20	65	85
D	20	25	30

4.2. Evaluation of Development of Services

We evaluated the time to construct three context-aware services with our toolkit to assess the cost of developing applications using the Wearable Toolkit.

1) *A musical-instrument service through dancing* that enables users to generate sounds by dancing.
2) *A context-aware mobile phone service* is shown in Fig. 4.
3) *An intelligent digital-camera service* that automatically takes photos when user shakes hands with other people, and stores the images with a timestamp.

The test subjects were four high-school students and three of them had no experience in programming. One (subject A) had two years of programming experience. After a one-hour tutorial, we assessed the time to implement services. The results are listed in Table 4. Note that the test subjects only used the reference manual and sample programs.

From the results, we found they could implement all services within a sufficiently short time using our system. There is an example of service descriptions in Fig. 10. Since services that are implemented on site are usually sufficiently simple, our toolkit has sufficient capabilities for on-site programming.

4.3. Evaluation of Context Tool

One of the remarkable features of our toolkit is its automatic definition of contexts in the context definition tool. However, if the automatically defined context is not sufficiently accurate in recognising contexts, the effectiveness of the features decreases. Therefore, we evaluated how capable the automatic definition of contexts was.

```
//Service 1: play sound by kicking
DEFINE SoundOne
 WHEN CONTEXT_RECOGNIZED(Kick)
 IF !GLOBAL.TMP
 THEN DO CMN_PLAY_SOUND('crash.wav')
  DO CMN_SET_TIMER('TMP', 500, 1)
  DO GLOBAL.TMP = BOOL(1)

DEFINE SleepWhile
 WHEN CMN_TIMER
 THEN DO GLOBAL.TEMP = BOOL(0)

//Service 3: take a photo by changing contexts
 DEFINE TakePhoto
 WHEN CONTEXT_RECOGNIZED(Handshake)
 THEN DO MM_GET_IMAGE()

DEFINE OnImage
 VAR jpeg AS BYTE_ARRAY
 WHEN MM_IMAGE
 THEN DO jpeg = MM_CONVERT_JPEG(NEW.IMAGE, 90)
  DO CMN_SAVE_FILE('NOW.HOURNOW.MINUTE.jpg', jpeg)
```

Figure 10. Example of implemented rules

Table 5. Recognising contexts for evaluation

Context	Note
Standing	Static
Sitting	Static
Walking	Continuous
Running	Continuous
Jumping	Short term motion
Turning arm	Short term motion
Reading RF-ID tag in evening	Complex
Walking around specific place	Complex
Turning arm around specific place	Complex

The test subjects were five researchers on context-awareness and eight lay people. These researchers had sufficient knowledge on context recognition and characteristic values for contexts. In the evaluation, each person defined nine contexts as listed in Table 5 in two ways, i.e., using the context tool *without* the automatic generation function and using the context tool *with* the function. After all contexts were defined, we evaluated the accuracy of context recognition by measuring the number of correctly recognised ratios when the test subjects repeatedly played these contexts.

The results of evaluation are shown in Fig. 11 and Table 6. The table lists the average recognition rates and characteristic-value matching rates for each situation. The latter evaluation criterion means the ratio between the selected characteristic values and the researchers manually selected values, which were the references to test the validity of value selection. The

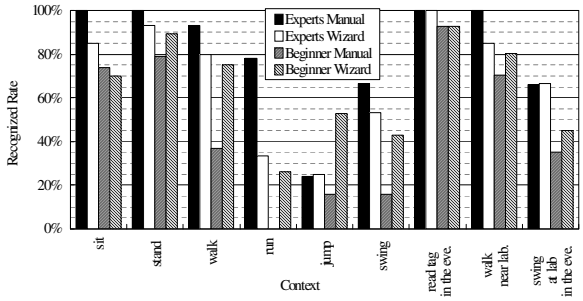

Figure 11. Evaluation results (detailed)

Table 6. Evaluation results (average)

Type	Recognition rate	Matching rate
Experts - Manual	0.81	1.0
Experts - Auto	0.69	0.85
Beginner - Manual	0.47	0.62
Beginner - Auto	0.64	0.80

figure shows the recognition rates for each context.

The results for beginners of using the automatic context definitions is clearly improved from manual definitions. This is because the higher matching rates achieve higher recognition rates. Therefore, this function can effectively be used by beginners. However, the results for automatic context definitions are still inferior to the experts' manual definitions. After carefully investigating the results, we found this was because the automatic definitions sometimes failed to extract the correct learning window. Since we employed a simple method of extracting the learning window, we should propose a more efficient method of extraction in the future.

5. Conclusion

We proposed a new framework to achieve on-site programming in wearable computing environments. We developed the *Wearable Toolkit* that includes a flexible ECA rule processing engine and several proficient tools for constructing services on site. The actual implementation of several services and the evaluation results confirmed our framework was effective. Moreover, we have started to propagate our toolkit on a Website. Its URL is http://wearable-toolkit.com. We hope the general public and not only researchers will use it to construct useful services. We plan to improve our toolkit based on its further use at actual events. Moreover, we intend to construct a repository to share plug-ins and rules on the WWW.

6. Acknowledgements

This research was supported in part by a Grant-in-Aid for Scientific Research (A)(20240009) and Priority Areas (21013034) made available by the Japanese Ministry of Education, Culture, Sports, Science and Technology.

References

[1] K. Ouchi, T. Suzuki, and M. Doi: "LifeMinder: A Wearable Healthcare Assistant," Proc. of the 2nd International Workshop on Smart Appliances and Wearable Computing (IWSAWC2002), pp. 791–792 (2002).

[2] M. Miyamae, T. Terada, M. Tsukamoto, K. Hiraoka, T. Fukuda, and S. Nishio, "An Event-driven Wearable System for Supporting Motorbike Races," Proc. of the 8th IEEE International Symposium on Wearable Computers (ISWC2004), pp. 70–76 (2004).

[3] M. Miyamae, T. Terada, Y. Kishino, M. Tsukamoto, and S. Nishio, "An Event-driven Navigation Platform for Wearable Computing Environments," Proc. of IEEE International Symposium on Wearable Computers (ISWC2005), pp. 100–107 (2005).

[4] T. Stiefmeier, G. Ogris, H. Junker, P. Lukowics, and G. Troster, "Combining Motion Sensors and Ultrasonic Hands Tracking for Continuous Activity Recognition in a Maintenance Scenario," Proc. of the 10th IEEE International Symposium on Wearable Computers (ISWC 2006), pp. 97–104 (2006).

[5] MIThril Project, http://www.media.mit.edu/wearables/mithril/.

[6] A. K. Dey, D. Salber, and G. D. Abowd, "A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-aware Applications," A Special Issue on Context-aware Computing in the Human-Computer Interaction, Vol. 16, Nos. 2–4, pp. 97–166 (2001).

[7] S. Albrecht, K. A. Aidoo, T. Antti, T. Urpo, L. V. Kristof, and V. V. Walter, "Advanced Interaction in Context," Proc. of the 1st International Symposium on Handheld and Ubiquitous Computing (HUC99), pp. 89–101 (1999).

[8] A. K. Dey, R. Hamid, C. Beckmann, I. Li, and D. Hsu, "a CAP- pella: Programming by Demonstration of Context-Aware Appli- cations," Proc. of International Conference on Human Factors in Computing Systems (CHI2004), pp. 33–40 (2004).

[9] M. Miyaame, T. Terada, M. Tsukamoto, and S. Nishio, "Design and Implementation of an Extensible Rule Processing System for Wearable Computing," Proc. of the 1st International Conference on Mobile and Ubiquitous Systems (MobiQuitous2004), pp. 392–400 (2004).

[10] Wearable Toolkit Website, http://wearable-toolkit.com/.