Gesture Profile for Web Services:
An Event-Driven Architecture to Support Gestural Interfaces for Smart Environments

Radu-Daniel Vatavu¹, Cătălin-Marian Chera², and Wei-Tek Tsai³

¹ University Stefan cel Mare of Suceava, 720229 Suceava, Romania
² Computer Science Department, Politehnica University of Bucharest, Romania
³ School of Computing, Informatics, and Decision Systems Engineering, Arizona State University, Tempe, AZ 85287, USA & Department of Computer Science and Technology, Tsinghua University, Beijing, China

vatavu@eed.usv.ro, catalin.chera@cs.pub.ro, wtsai@asu.edu

Abstract. Gestural interfaces have lately become extremely popular due to the introduction on the market of low-cost acquisition devices such as iPhone, Wii, and Kinect. Such devices allow practitioners to design, experiment, and evaluate novel interfaces and interactions for new smart environments. However, gesture recognition algorithms are currently the appanage of machine learning experts which sometimes leaves AmI practitioners dealing with complex pattern recognition techniques instead of focusing on prototyping ambient interactions. To address this problem, we propose GPWS (Gesture Profile for Web Services), a service-oriented architecture (SOA) designed to assist implementation of gestural interfaces. By providing gesture recognition as a web service, we leverage easy and fast adoption of gestural interfaces for various platforms and environments through simple service discovery and composition mechanisms. We discuss two GPWS designs based on SOA 1.0 and SOA 2.0 standards, analyze their performance, and demonstrate GPWS for a gesture-controlled smart home application.

Keywords: Gesture, gesture-based control, service-oriented computing, event-driven architecture, smart home, web services, gesture recognition, SOA, EDA.

1 Introduction

Moving into a world of intelligent ambient systems puts high demands on the interactive experience delivered to users [1][2][32]. However, traditional ways to interact with information systems have been restricted to keys and buttons, computer mice, and trackpads, while the remote control still represents the industry standard for interacting with household electrical appliances despite the potential of next-generation homes [5]. Such standard interfaces prove sometimes difficult to control (e.g., TV remotes are being perceived as confusing by most users [4]) or inappropriate for some tasks and contexts (e.g., the control of a remote screen in the kitchen [27]). As a suitable alternative, gestural interfaces are meant to deliver natural and intuitive interactions [26][37]. This angle has already been successfully exploited by the gaming industry which showed an increased interest for incorporating people’s willingness to move and gesture into playing
video games. The outcome has been a large palette of gesture acquisition devices with Nintendo Wii Remote\(^1\) and Microsoft Kinect\(^2\) being just a few popular examples which are being reused today for ambient intelligence applications [11,15,37,38].

As practitioners of ambient intelligence become more and more interested in implementing gestural interfaces into their own designs, high access to gesture recognizers is needed. One way to achieve this is to provide designers with detailed pseudocode for such recognizers [3,41] so that they can be adopted and implemented on various platforms. Another practice would be to reuse existing code, not necessarily in the form of libraries but rather as service provided over the web. The practice of service-oriented software engineering [10,14] has already shown the benefits of such software architectures in terms of platform independence, loosely coupled design, and alignment with life cycle support processes. In this context, we believe that a common framework hiding the complex details of gesture recognition algorithms and machine learning formalisms while exposing clear services would be extremely beneficial to practitioners. This way, gesture-based control could be more easily adopted by the service computing community just as any other service available on the web. Therefore, we discuss in this work web services that deliver gesture recognition by following the existing AmI practices of developing service-oriented software infrastructure [7,24,34,36].

This paper introduces GPWS (Gesture Profile for Web Services), a novel approach of presenting gesture recognition for control applications in a service-oriented event-driven manner. We highlight our main contributions:

1. We provide AmI practitioners with web services for gesture recognition in order to facilitate easy adoption and promote gesture-based interactions for smart environments.

2. GPWS is the first event-driven architecture for gesture recognition. This design choice is motivated by the fact that human gestures are naturally event-driven (they have clear start, execution, and ending timestamps) and that acquisition devices also deliver data in discrete events.

3. We motivate the need for an event-driven implementation (SOA 2.0) for gesture processing by discussing a comparison with a simpler GPWS design which uses simple request-response web services as per the standards of SOA 1.0. We show how each architecture brings benefits to practitioners in accordance to their application needs and requirements.

By introducing GPWS we hope to address the practical needs of researchers and practitioners interested in using gestures for their applications. We plan GPWS as an open-source framework with free services available to the community and refer the interested reader to [http://gpws.fcint.ro](http://gpws.fcint.ro). We demonstrate our architecture with a sample application designed to control household appliances.

\(^1\) [http://nintendo.com/wii](http://nintendo.com/wii)
2 Related Work

A large amount of work exists in the pattern recognition and human-computer interaction communities with regards to capturing motion and gestures, implementing gesture recognizers, and designing gestural interfaces [25,28,29].

Common gesture capture technologies include video cameras [28], accelerometers embedded in mobile devices [31], game controllers [33], wrist watches [21], and worn equipment such as sensor gloves [13]. Also, the recent years have seen several low-cost acquisition devices being released by the gaming industry, such as Nintendo Wii Remote and Microsoft Kinect. Many researchers have reused the motion sensing capabilities of such devices for interactive applications and for exploring gesture-based interfaces. For example, Lee [17] has inventively used the Wii Remote for 3D head tracking and touch-based interactions. Bott et al. [6] used the remote for interactive art and music applications. Vatavu [38] described an ambient interactive system using Kinect in order to demonstrate the concept of nomadic gestures, while Panger [27] used Kinect for augmenting interactions in the kitchen. Also, researchers showed interest in the technical performance of such devices such as the pointing accuracy of the Wii Remote [22] or its performance for recognizing gestures [19,33]. Wii has also found applications in controlling home appliances. For example, Pan et al. [26] used the Nunchuck controller attached to the Wii Remote to implement GeeAir, a device for controlling household appliances using gestures. Vatavu [37] explored the Wii Remote functionalities for providing WIMP-like interactions (Windows, Icons, Menus, Pointer) for working with multiple TV screens for home entertainment systems.

Whilst paying careful attention to these works in order to select the most appropriate recognition technique for our implementation, we are more interested in software architectures designed for gesture-based control as well as in gesture taxonomies and ontologies informing the design of such architectures [8]. Following this perspective, previous research of direct interest to our work can be grouped into gesture ontologies and software architectures.

Ontologies represent a powerful tool for understanding concepts and relationships with important applications in informing software design and development [18,43]. Wang et al. [39] introduced and described an ontology for multi-touch gestures. The authors discussed atomic gestures and techniques for combining them using temporal, spatial, and logical relationships. Sonntag et al. [35] defined ontological representations for pointing gestures. van Segbroeck et al. [34] described WS-Gesture which represents a framework for controlling devices based on DPWS (Devices Profile Web Services). DPWS relies on WS-* protocols initially designed for enterprise computing and then ported to the device networks that may have hardware constraints that an enterprise system does not have. Chera et al. [8] developed a gesture ontology for informing the design of service-oriented architectures for gesture controlled applications. The ontology groups gesture concepts and relationships into execution, implementation, and reflection levels. Although not explicitly developing ontologies, other researchers have classified interactive gestures by identifying common properties. For example, Wobbrock et al. [42] introduced a taxonomy of surface gestures by taking into consideration criteria such as form, nature, binding, and flow. Ruiz et al. [31] proposed a taxonomy for motion gestures in the context of mobile phones by considering gesture
mappings (nature, context, and temporal) and physical characteristics (kinematic impulse, dimension, and complexity) as classification criteria.

The practitioners of ambient intelligence have already considered service-oriented architectures for the software infrastructure needs of smart environments [36]. For example, Mingkhwan et al. [24] proposed an architecture for interconnecting home appliances and their services; Chakraborty et al. [7] discussed service composition for mobile and pervasive computing environments; and Seghbroeck et al. [34] described the first protocol of using gestures in conjunction with web services. This work adds gesture recognition to the software infrastructure layers of ambient intelligence [36] by extending previous works [34] with the first event-driven service-oriented design for gesture recognition.

3 Gesture Representation and Recognition

Throughout this work we understand by gesture a continuous motion captured using some specific acquisition device. Irrespective of the acquisition technology, a gesture can be finally represented as a time-ordered series of points: \( \{(x_i, y_i, z_i) | i = 1..n\} \), where \( n \) represents the sampling resolution. Figure 1 illustrates a “beat” gesture consisting in two rapid hand movements captured by a 3-axis accelerometer.

![Image of gesture recognition](image)

**Fig. 1.** Accelerated gesture composed of two beat-like movements captured by a 3-axis accelerometer (beat strokes can be easily identified by their high force peaks). Screen capture from our demo application (see the last section of the paper).

The pattern recognition and human-computer interaction communities have proposed many algorithms to date for recognizing gestures [3, 9, 19, 28, 29, 40, 41]. Out of these, this project employs the Dynamic Time Warping distance (DTW) [19] in the context of the Nearest-Neighbor supervised learning approach (NN). DTW computes the optimum alignment between two motion gestures by minimizing the total sum of point-to-point Euclidean distances. The NN approach classifies a candidate gesture by
comparing it with every sample stored in the training set and returns the class of the closest sample. We selected the combination of DTW/NN to perform gesture recognition as it was found to work well in the presence of user variation in gesture execution even for long term testing [19]. As the gesture recognition technique is not the main focus of this paper and we are more interested in software architectures that deliver efficient web services to the community, we refer the interested reader to Liu et al. [19] for more details on the recognition process.

The gesture terminology introduced so far only deals with single stroke gestures. However, more complex gesture commands can be imagined by composing individual gesture executions. For example, drawing a “circle” followed by drawing digit “1” could stand as a command for increasing temperature with 1 degree for an air conditioner in a smart home scenario. We therefore define a gesture sequence command as a set of one or more gestures that have been associated with meaning: \[
gesturesequence = \{\text{gesture}_1, \text{gesture}_2, \ldots\}.
\]

4 GPWS: Gesture Profile for Web Services

The motivation behind our work is to make gesture recognition available as a service for researchers and practitioners interested in prototyping gesture-based control applications. Also, we are interested in providing such services with different levels of complexity to address applications with various requirements and needs. This section presents two such service-oriented architectures. The first one implements the minimum amount of services and functionality needed to fulfill the basic needs of a practitioner interested in gesture-based processing: gesture recognition and management of the gesture set. The solution is therefore characterized by low complexity and simple usage patterns in the standard of SOA 1.0 web services [3]. The second SOA 2.0 event-driven architecture [4] is more elaborate, flexible, and therefore able to expose an increased level of functionality for subscribing clients. At the end of the paper, we discuss architecture performance and provide a comparison between the two designs.

4.1 GPWS 1.0: A Simple Architectural Design Using SOA 1.0 Response-Request Standards

This section presents our first implementation of a service-oriented GPWS using SOA 1.0 web services within the standard request-response protocol. By analyzing the practical requirements of a gesture-controlled application, we identified and implemented the following services:

1. **Recognition services.** Two services were implemented to recognize motion gestures (**GESTURE-RECOGNITION-SERVICE**) and sequences of gestures executed in order (**GESTURE-SEQUENCE-RECOGNITION-SERVICE**). The gesture service

---


implements the Nearest-Neighbor classification approach employing the DTW distance for motion trajectories, as described previously in the paper. The gesture sequence service implements a simple matching rule which compares the candidate sequence against a set of previously stored prototypes. For example, if gestures “circle” and “up” are performed consecutively by the same user in less than $X$ seconds, this sequence of gestures is interpreted as a single command.

2. Manager services for user-defined data. Gestures and commands (i.e., functions controlled in the application) represent user-defined data. Two services were implemented for storing and retrieving such data from a database repository: GESTURE-MANAGER-SERVICE and GESTURE-SEQUENCE-MANAGER-SERVICE. They serve as a simple interface between the client application and the database allowing users to manage gesture-function mappings. The first service associates gesture motions to ID values representing class names such as “circle”, “turn-on”, “help”, etc. The gesture sequence manager associates a list of gesture IDs to a command ID. For example, the command “increase the temperature of the air conditioner by 1 degree” can be stored as the set \{“up”, “digit-1”\} with each member in the set representing the class ID of an individual gesture.

Figure 2 illustrates the basic components of this first GPWS architecture: client application, gesture acquisition device, web services, and database. During setup, clients use the manager services to upload gesture-function mappings as well as gesture samples for the training set. During runtime, the client application performs requests to the gesture recognition services that respond with class IDs. Gesture recognition details are therefore hidden away into the GPWS architecture which only exposes simple functions to client subscribers.

4.2 GPWS 2.0: An Elaborated Event-Driven Architectural Design Using SOA 2.0 Standards

Although the first design satisfies the basic needs of a client application implementing gesture recognition, it is limited in terms of flexibility and scaling (e.g., only the requesting client is notified of the classification result). However, complex applications with more demanding requirements such as multiple clients that need to be informed when a specific action has occurred, need more elaborated architectures. As SOA 1.0 services use the principle of synchronous request-response calls, they can’t address such needs. SOA 2.0 goes further by allowing asynchronous calls. This way, event-driven architectures allow various components to monitor the environment, process events, and respond to changing conditions continuously. Multiple clients can be informed once a specific event occurred (e.g., the user performed the “turn-off air conditioner” gesture) in order to take specific decisions (e.g., the air conditioner client performs the command; the power consumption monitoring client updates its status and predictions; while a smart client monitoring user actions can verify whether this is typical behavior of the specific user). Also, in complex EDA [20], the traditional subscription ways in

---

5 The number of seconds represents an application setting which depends on the length of the sequence, e.g. 10 seconds for small sequences of 2-3 gestures. As a thumb rule, we add 5 seconds for each additional gesture in the sequence.
Fig. 2. 1st generation GPWS consisting in gesture managers and gesture recognition web services in the style of SOA 1.0 standards

which a filter is usually applied to a single event are replaced by more sophisticated mechanisms for which the decision is made using correlations across histories of multiple event streams.

We also note that gestures are inherently event-driven: they consist of preparation, stroke, and retraction phases [23]. Also, acquisition devices signal new incoming data at specific discrete intervals while recognizers process the acquired gestures and report recognition scores in a timely fashion. All these represent clear and distinct events from the application but also the users’ point of view. Therefore, in order to model such mechanisms, we designed an event processing architecture which we refer to as the 2nd generation GPWS. The architecture has five main components (see Figure 3):

1. **Events** are generated and processed in correspondence to gesture acquisition, gesture recognition, and gesture sequence matching key points (specific events employed by the architecture are described in detail next in the paper).

2. **Event processing services** implement event generation, processing, logging, and publishing. For example, the EVENTS-DISPATCHER-SERVICE serves as the architecture key point for generating events while the EVENTS-PROCESSING-SERVICE implements all the logic required for processing, logging, and publishing events. The specific services of GPWS 2.0 are described next in the paper.

3. **Acquisition service**: a special-purpose GESTURE-ACQUISITION-SERVICE was implemented to serve as middleware between the client application and the GPWS 2.0
architecture. This assures the independence between the acquisition device and the functionality of GPWS. The acquisition service represents the entry point of the architecture for connecting clients.

4. **Recognition services**: the same recognition services from the first generation architecture were also kept in this one. However, this time they act as event handlers and are being called by the EVENTS-PROCESSING-SERVICE when needed.

5. **Manager services for user-defined data**: they are the same as in the first generation architecture (GESTURE-MANAGER-SERVICE and GESTURE-SEQUENCE-MANAGER-SERVICE) and are used to store and retrieve user-defined gestures and commands to/from the database.

GPWS 2.0 processes three types of events relating to gesture acquisition, recognition, and sequence matching. GESTURE-ACQUIRED-EVENT is generated once the gesture was successfully acquired from the user. The client application connects to the entry point of the architecture to acquire gestures.
point of the GPWS architecture and performs a call to the GESTURE-ACQUISITION-SERVICE which will make a request to the EVENTS-DISPATCHER-SERVICE to generate a new acquisition event. The event has two arguments: client ID and the set of points acquired from the device. The GESTURE-RECOGNIZED-EVENT is generated once GESTURE-RECOGNITION-HANDLER has run the recognition algorithm. The event arguments (client ID and the class of the recognized gesture) are fed back to the EVENTS-PROCESSING-SERVICE. The gesture sequence event is produced once the GESTURE-SEQUENCE-RECOGNITION-HANDLER has run the sequence matching algorithm. This event also has two arguments: client ID and the class of the recognized command.

Most of the tasks in GPWS 2.0 are performed using the event processing service as per the following scenario. The client application acquires gesture data from a specific device. Once a gesture has been acquired, the client notifies the acquisition service that a new gesture is ready to be processed. The acquisition service represents the single entry point for the client when working with the GPWS 2.0 engine while it also separates the specific details of the acquisition device from gesture representation required by GPWS. The acquisition service uses the EVENTS-DISPATCHER-SERVICE to create a new GESTURE-ACQUIRED-EVENT having as arguments the client ID and the acquired motion. The event is passed to EVENTS-PROCESSING-SERVICE. The only function of the dispatcher service is to act as a central unified point in the architecture for creating events and for transmitting them towards processing. The event processing service implements the logic for handling incoming events:

- The event is logged in the database together with its arguments and timestamp.
- The appropriate handler is called: GESTURE-RECOGNITION-HANDLER will handle GESTURE-ACQUIRED-EVENT(s) and GESTURE-SEQUENCE-RECOGNITION-HANDLER will process GESTURE-RECOGNIZED-EVENT(s). These handlers implement gesture and sequence matching algorithms by using gesture samples and command definitions stored in the database via manager services.
- After calling and executing handlers, new events are generated as follows: GESTURE-RECOGNIZED-EVENT is generated after a call to the GESTURE-RECOGNITION-HANDLER and GESTURE-SEQUENCE-RECOGNIZED-EVENT may be generated after a call to the GESTURE-SEQUENCE-RECOGNITION-HANDLER. The processing service uses the dispatcher to generate such events (which by means of the dispatcher will finally reach the processing service later on).
- If the event should be published, the processing service will publish it to the relevant subscribers.

5 GPWS Performance Analysis and Demonstration

We report in this section GPWS performance in terms of recognition rate and response times and describe a demo application employing GPWS in the context of smart home control. Table 1 lists a summary comparison between the two GPWS architectures on implementation and performance criteria.
5.1 Recognition Rate

A simulation environment was designed for automatic testing in order to assess the recognition performance of the GPWS architectures. A custom designed application simulated users performing motion accelerated gestures. The simulation used data from a publicly available gesture set introduced by Hoffman et al. [12]. The set contains 25 types of motion gestures performed repeatedly by 17 participants for 20 times with a total number of 8,500 samples. More details on the acquisition procedure, apparatus, and participants can be found in [12]. The testing application loaded gesture samples from the set which were sent for processing to GPWS, simulating thus user behavior.

As Nearest-Neighbor recognizers perform better as more gesture samples are available in the training set, we compute and report recognition performance for varying numbers of samples per gesture type. Figure 4 (left) shows the results obtained. Recognition accuracy reached 92.5% with just 4 samples per gesture type, rose up to 95.0% with 8 training samples, and reached 96.6% with 16 examples. A fixed sampling rate of \( n=32 \) points was used during testing. A Friedman test showed a significant effect of the number of training samples on recognition accuracy (\( \chi^2(4) = 5634.966, p < .001 \)).

![Figure 4. GPWS architecture performance. Left: accuracy of the gesture recognition service (DTW and Nearest-Neighbor) vs. the number of training samples per gesture type. Right: response times for the two architecture designs (error bars show ±1 SD).](http://www.eecs.ucf.edu/isuelab/downloads.php)

5.2 Response Time

We also measured response times for the two different architectures. The simulation application measured and averaged response times for 500 consecutive gesture recognition requests. We hypothesized that the first generation GPWS would be faster due to direct request-response mechanisms. However, response times were close for both architectures: 488.6 ms (SD=28.5 ms) for the first generation GPWS and 479.2 ms (SD=33.6 ms) for the second event-driven implementation (Figure 4, right). A Wilcoxon signed-rank test showed that GPWS 2.0 was significantly faster than GPWS 1.0 (\( Z = -6.107, p < .001 \)).
Table 1. Summary comparison between the two GPWS architectural designs

<table>
<thead>
<tr>
<th>Criterion</th>
<th>GPWS 1.0</th>
<th>GPWS 2.0</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FEATURES</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Web services standards</td>
<td>SOA 1.0</td>
<td>SOA 2.0</td>
</tr>
<tr>
<td>Communication</td>
<td>synchronous</td>
<td>asynchronous</td>
</tr>
<tr>
<td></td>
<td>(request-response mechanisms)</td>
<td>(event processing)</td>
</tr>
<tr>
<td>Supports 3rd party clients</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td><strong>PERFORMANCE</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gesture recognition accuracy&lt;sup&gt;1&lt;/sup&gt;</td>
<td>95% with 8 samples per gesture type and 97% with 16 samples per gesture type</td>
<td></td>
</tr>
<tr>
<td>Response time&lt;sup&gt;2&lt;/sup&gt;</td>
<td>489 ms</td>
<td>479 ms</td>
</tr>
<tr>
<td><strong>USAGE RECOMMENDATIONS</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Should be used when</td>
<td>simple gesture recognition is needed by a single client application</td>
<td>gesture events are important for 3rd party clients</td>
</tr>
</tbody>
</table>

<sup>1</sup> Measured on a set of 8,500 3D gesture samples<sup>[12]</sup>, see the appropriate section for details.

<sup>2</sup> Measured on a 2.40 GHz Intel CoreDuo Quad CPU computer, see the appropriate section for details.

With a small Cohen effect ($r < .2$), execution times are acceptable for real-time response requirements. Times were measured on a 2.40 GHz Intel CoreDuo Quad CPU computer running Microsoft Windows with IIS hosting GPWS and MySQL as supporting data server.

### 5.3 Home Control Application

In order to demonstrate GPWS, we implemented a home control application. We chose such an application due to many works being interested in controlling household devices via gestures<sup>[16,30,37]</sup>. The testing environment was a laboratory room which contained four controllable devices: two light lamps and two air conditioners. While the lamps can only be switched on and off, the air conditioners exposed several functions: on/off; increase/decrease temperature by X degrees; select operation mode; and schedule settings. Users were able to define, edit, and publish gesture commands to the server and to associate commands to devices and functions. Figure 5 illustrates the operation of the application. Gestures were acquired with the Wii Remote which was selected for several reasons:

- The TV remote-like form factor makes the device look familiar to most users.
- Multiple options for capturing user input (a 3-axis accelerometer; an optical sensor with embedded software working in the infrared domain; 12 buttons such as +/-, home, 1/2, up/down, and left/right).
– Multiple options for providing user feedback in the form of audio through the embedded speaker, haptic through a vibromotor, and visual by using the 4 bright LEDs located at the bottom on the front side of the controller.
– Easy connectivity with PC systems via Bluetooth.
– Numerous users are buying the Wii console (over 95 million units sold before March 2012\(^7\)). Therefore, users are already familiar with how the Wii controller works and this acquired experience can be reused in our application.

Fig. 5. Operation diagram for the smart home control application

As all devices exposed similar functions (on/off), we adopted the design choice of assigning a unique gesture for common functions and prefix it by an ID gesture identifying the device. For example, the air conditioner 1 (AC1) was identified by performing a quick beat-like movement while air conditioner 2 (AC2) with two quick movements (the two force peaks gesture illustrated in Figure 1). Lamp one (L1) was identified by a small circle (suggesting the shape of the bulb) and lamp two (L2) by two consecutive circles. Once the device identification gesture was performed, all following gestures were directed to that specific device. For example, an “S”-like shape was used to start and an “X” to stop the device. Valid commands in our application were \{“Identify-AC1”, “S”\}, \{“Identify-Lamp-2”, “X”\}, \{“Identify-AC2”, “arrow-down”, “digit-1”\}, etc. We limit our description here and note that the main goal of the smart home prototype was only to demonstrate the applicability of the GPWS architecture for gesture recognition. Therefore, we don’t discuss here aspects related to gesture set design (i.e., gesture fit to function, learnability, and execution difficulty) for which special design techniques have been proposed in the literature [31,42]. We simply note the availability of the GPWS web services at [http://gpws.fcint.ro](http://gpws.fcint.ro). Also, in order to illustrate GPWS 2.0 ease of use, we present below a short section of C# code that employs the architecture to recognize a gesture candidate and receives the response as an event

\(^7\) Nintendo, Consolidated Sales Transition by Region, [http://www.nintendo.co.jp/ir/library/historical_data/pdf/consolidated_sales_e1203.pdf](http://www.nintendo.co.jp/ir/library/historical_data/pdf/consolidated_sales_e1203.pdf)
Gesture Profile for Web Services 173

(we assume that gesture samples have been submitted before by the client identified by subscriberId):

```java
public class GPWS20Example : SubscriptionService.IGestureRecognizedCallback
{
    SubscriptionService.GestureRecognizedSubscriber subscriber = null;
    InstanceContext contextOnGestureRecognized = null;
    AcquisitionService.GestureAcquisitionClient clientAcquisition = null;
    string subscriberId = "1000";

    // Subscribe to GPWS and create an acquisition client
    public void Initialize()
    {
        contextOnGestureRecognized = new InstanceContext(null, this);
        subscriber = new SubscriptionService.GestureRecognizedSubscriber( contextOnGestureRecognized );
        subscriber.Subscribe();
        clientAcquisition = new AcquisitionService.GestureAcquisitionClient();
    }

    // Data comes from the acquisition device,
    // usually as one data point at a time but we simplify here for the clarity of this example
    public void OnGestureData(Gesture gesture)
    {
        // Send gesture to GPWS for processing
        clientAcquisition.GestureWasAcquired(subscriberId, gesture);
    }

    // Gesture was recognized and event received from GPWS
    public void OnGestureRecognized(string subscriberId, string gestureName)
    {
        // Gesture was recognized as gestureName. Perform action.
    }

    // Unsubscribe from GPWS and close clients
    public void Uninitialize()
    {
        subscriber.Unsubscribe();
        subscriber.Close();
        clientAcquisition.Close();
    }
}
```

6 Conclusion

We introduced in this paper Gesture Profile for Web Services, an event-driven architecture delivering gesture recognition services. We experimented an SOA 2.0 event-driven design after we first implemented an SOA 1.0 request-response architecture. The EDA design was motivated by the event-driven nature of human gesture production and gesture acquisition devices. Performance measurements showed real-time responsiveness and high recognition accuracy of GPWS. Future work will focus on adding further processing layers on top of GPWS. One example is complex events processing [20].
for inferring advanced knowledge from simple events which will allow other service-oriented architectures \[35\] to automatically be notified when complex event patterns occur.

**Acknowledgments.** This work is supported partially by European Regional Development Fund and the Government of Romania under the grant no. 181 of 18.06.2010 and the European project Empowering Romanian Research on Intelligent Information Technologies (ERRIC), FP7-REGPOT-2010-1/264207.

**References**

5. Bonino, D., Corno, F.: What would you ask to your home if it were intelligent? Exploring user expectations about next-generation homes. J. Ambient Intell. Smart Environ. 3(2), 111–126 (2011)


