Eventcasting with a Wearable Computer

Mikael Drugge, Marcus Nilsson, Kåre Synnes, Peter Parnes
Luleå University of Technology
Department of Computer Science & Electrical Engineering
Division of Media Technology, SE-971 87 Luleå, Sweden
{*mikael.drugge, marcus.nilsson, kare.synnes, peter.parnes*}*@sm.luth.se*

Abstract

Traditionally, interaction methods for wearable computers have focused on input to the computer itself, yet little has been done when it comes to allowing interaction with the surrounding environment. Pervasive computing, on the other hand, offers access to computational power from any place all the time, yet most interaction techniques utilize either physical hardware or monitoring of the user in order to receive input.

This paper presents a novel form of interaction by which a wearable computer user can interact with and control a pervasive computing environment in a natural and intuitive manner. Using sensors, the user can be allowed to literally "throw" events into the environment as a way of interacting with devices and computers.

1. Introduction

New forms of interaction are made possible as a result of wearable and pervasive computing merging. We believe that for the user of a wearable computer, being able to interact with the surrounding environment is at least as important as interacting with the computer itself. We also believe that for a wearable computer to be commonly used and accepted by the general public, the interaction mechanisms employed must be highly natural and easy to learn, use and apply. The question is what constitutes such forms of interaction?

For a human being, some movements are more or less inherent and used unconsciously on an everyday basis. For example, when someone asks you for directions, you may use your hand to point that person in the right direction. When a mosquito flies in to bite you, maybe you fling out your arm to deflect it. Working with computers may have you familiar with the concept of pointing and clicking by use of a mouse controlled by your hand. All in all, humans tend to very much use their hands and arms in motion to achieve some desirable action. For this reason, it would be highly beneficial to extend this natural way of interacting to allow for controlling devices at a distance.

This paper presents a method by which these natural movements can be made to apply on devices in the surrounding. In short, it works by placing a position sensor near the user's hand so that a throwing motion can be detected. Once a throw is performed, devices use its trajectory to compute whether they are affected.

This interaction method, from now on dubbed *Eventcasting*, allows a user to perform a virtual throw — in essence being able to send events onto the network for devices to act upon. When the sensor is built into an everyday accessory, e.g. a wrist watch, ring or bracelet, this becomes a highly natural and unobtrusive way of interacting with devices in the surrounding. The research questions this brings forward are as follows:

- What benefit for the user of a wearable computer does this interaction technique offer?
- What differentiates this method from similar interaction methods, and what are the advantages and disadvantages of this approach?
- How are events created, specified and transmitted to the right device, and how does the device react?

It should be noted that details regarding the underlying positioning hardware used falls outside the scope of this paper. As the concept of *Eventcasting* works with any positioning system, the paper will mainly focus on the theory behind the interaction method. Nevertheless, available positioning systems will be discussed briefly where applicable in order to illustrate the viability of the concept.

The organization of this paper is as follows. Section 2 discusses related work in the area of interacting through a wearable computer. In section 3, the theory and design of the interaction method is presented. Section 4 presents our prototype implementation. Section 5 discusses this form of interaction and its implications. Finally, section 6 concludes the paper together with a discussion of future work.

2. Related Work

Many pervasive computing environments of today rely on a predefined physical infrastructure or some form of remote monitoring of the user in order to facilitate interaction with her surrounding [7]. In *Eventcasting*, the wearable computer can perform all or most of the required monitoring of its user by tracking the user's hand movements. While some form of infrastructure may still be required for positioning the user in the room, the privacy issues are not as significant as for monitoring by other means, e.g. by use of a video camera. The integrity of the user is also better maintained since it is solely the wearable computer user who decides when to disclose the information — in this case two consecutive hand positions — by issuing a virtual throw.

Much work has been done on positioning and tracking a user; both indoor[9], outdoor[2], and combinations via hybrid positioning systems[5] exist. Ultrasonic positioning can provide high accuracy at a low cost for use in e.g. an office[10]. These together with more local tracking devices, such as the acceleration glove[6] or the Fingermouse[1], can help provide the positioning which *Eventcasting* needs.

Using gestures for interaction is an area which has been studied extensively because of its naturalness for the user [11]. The gesture pendant [8] by Starner et al. is a wearable device for controlling home automation systems via hand gestures. While being able to detect gestures made in front of the camera, it lacks depth information making it unable to detect 3D motions such as a throw.

Controlling devices in the surrounding by pointing at them has been done before, typically by employing a "point-and-click" approach. In [3], Kindberg and Zhang use a handheld device equipped with laser to negotiate an association with a target device. To operate, the user presses a button to start the laser beam, then aims towards the target and pushes a second button to perform the association. In [4], Kohtake el al. use a handheld device which allows "drag-and-drop" of virtual information between real world objects. The user points the device at a marker on a real world object, clicks a button on her device to start dragging, then points at the destination marker and clicks another button to drop the information. What differentiates *Eventcasting* from [3] and [4] is the use of a throwing motion to initiate the operation. As the two actions in the "point-and-click" approach are reduced to a single action — that of the throwing motion itself — this eliminates the need for the user to hold a device in her hand and to push any buttons. As a result, the user's hands are free to operate in the real world without being encumbered by any equipment. This unobtrusiveness is what makes *Eventcasting* an important contribution to this field of research.

3. Theory and Design

In this section the theory behind *Eventcasting* is presented. The main idea is to allow a user to use the movements of her arms, e.g. flinging or throwing motions, to interact with devices in the surrounding environment. To achieve this, a sensor able to provide positional data is placed near a wearable computer user's hand. Based on position and acceleration information, a throwing motion can be detected together with its start and end position relative to the user's body. By adding a world-centric position to this, the result is a trajectory for an imaginary, or virtual, throw in the room. Using IP multicast, this positional data is then transmitted wirelessly onto the network where devices can receive it. As each device is assumed to know its own position, it can use the positional data to compute whether the virtual throw would hit it. If a device determines itself as being hit it will react on the throw by inspecting the payload received together with the positional data. The payload can be specified by the user just before issuing the throw, e.g. via voice or gesture. These steps will be explained in more detail in the following sections.

3.1. Throwing

A throw is initiated by a sudden quick movement when the hand accelerates; this is triggered by readings from the sensor rising above a certain threshold. A throwing motion includes a follow-through phase, which is the period after the release of the object being thrown. During this period, the limb movement begins to decelerate; thus the termination of the throw can be triggered by sensor readings passing below a second threshold. This way, information about positions in 3D space is retrieved, thereby giving the start and end position for the throw. Figure 1 provides a simplified illustration of this.

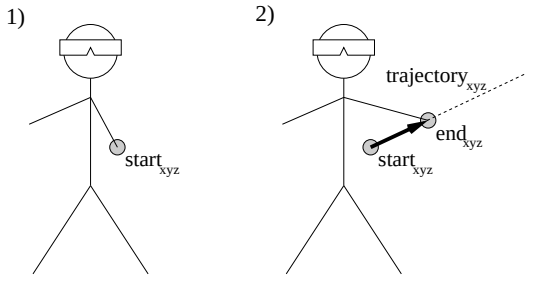

Figure 1. Start and end positions for a throw.

Based on the start and end positions, it is trivial to compute a trajectory for the virtual throw relative to the user.

$$throw\_trajectory_{xyz} = end_{xyz} - start_{xyz}$$

It should be noted that this trajectory is a straight line giving the direction for the throw. In reality, however, the user's hand most often traverses in an arc when performing a throw. An ideal example of this is illustrated in figure 2.

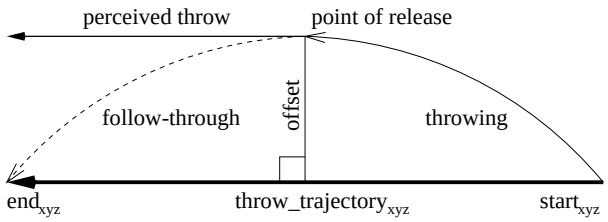

Figure 2. Arc-shaped throwing motion.

Although the user's hand traverses in an arc, the straight line trajectory is still assumed accurate enough to point out the desired object. The reason for this is that since the end position lies in the follow-through phase and not at the release point, the resulting trajectory will be more or less parallel to the throw as it is perceived by the user. The offset between the two is ideally small enough to trick the user into perceiving them as being one and the same. By adjusting the thresholds used for when to get start and end positions, each user can customize the system to suit their own pitching form. This may require a short training period to make the user proficient with performing virtual throws.

As an ordinary gesture must not be misinterpreted as a throwing motion, steps must be taken to ensure that only true throws are treated as such. The threshold to retrieve the starting position should be high enough only to react on motions similar in velocity to that of a throw. In addition, the time between retrieval of the start and end position can also be exploited so that only a reasonably swift motion is treated as a throw while the rest are discarded. Similarly, a restriction can also be imposed on the distance allowed between the start and end position, so that e.g. only very long motions are to be treated as a throw.

Thus far, only the retrieval of the throwing motion itself has been discussed, but to be of any use this must be placed in relation to the real world. Ideally, the sensor should be sophisticated enough to provide positioning in the real world without requiring a predefined infrastructure. This would require the sensor to be small enough to be wearable, yet provide indoor and outdoor positioning with centimeter level accuracy at high frequency. The problem is that such sensors are not available at the time of writing, nor are they expected to become available for the foreseeable future. To alleviate this problem, an externally given position can instead be added to the trajectory, thereby giving a trajectory in the real world. The external position can be retrieved from any source accurate enough, e.g. an ultrasonic tracking system or real-time kinematic GPS as in [2]. When this position is added, the resulting trajectory describes the throw in the real world's frame of reference.

$$trajectory_{xyz} = external_{xyz} + throw\_trajectory_{xyz}$$

Once a throw has been initiated in this way, the wearable computer sends the trajectory in an event onto the network.

3.2. Sending

The wearable computer is assumed to have some form of wireless connection to a network by which packets can reach the devices they are ultimately intended for. The devices need not be wireless enabled themselves, but can be interconnected through a fixed wired network.

The wearable computer places the start and end positions for the throw into a UDP packet, together with a payload specifying the event. Using IP multicast, the packet is sent to devices listening on a specific multicast group. A problem with this approach is that the system does not scale if multicast is deployed over a very large area, as packets from multiple users would then reach every device and consume network and CPU resources. For typical use, e.g. in a home or office environment, this is however not an issue.

An alternative to the use of unreliable UDP would be to use TCP instead and address devices directly. However, this approach becomes a service discovery problem requiring the wearable computer to monitor which devices are in the surrounding, thereby imposing a higher workload on it. The event distribution model needs further refinement, but that falls outside the scope of this paper.

3.3. Receiving

A device listening on a multicast group will receive a number of UDP packets containing events. Of these, only a subset may be relevant for that specific device. In order for the device to know whether it should react or not, it needs to compute whether it would be hit by the virtual throw based on the data received. Three parts are involved here; hit detection, reacting on the payload, and providing feedback.

3.3.1. Hit Detection. Each device knows its own position $device_{xyz}$ which can be either fixed and entered manually, or automatically retrieved through sensors similar to that which the wearable computer uses. The device also has an associated $radius$ forming a sphere containing the physical device. From an arbitrary point of view, this sphere can also be seen as a circular plane with the same radius. Therefore, from the user's point of view, she will imagine a circular plane centered at the device and that plane is the intended target for the virtual throw. This is believed to be easier for the user to imagine and aim for, rather than envisioning intersection with a sphere covering a device, especially at close range. When a virtual throw has a trajectory

that intersects with the circular plane of this sphere of influence, the device is hit.

As the start and end positions for the virtual throw, referred to as $start_{xyz}$ and end_{xyz}, are received in the packet, the hit detection mechanism now knows all it needs to compute whether the device is hit or not.

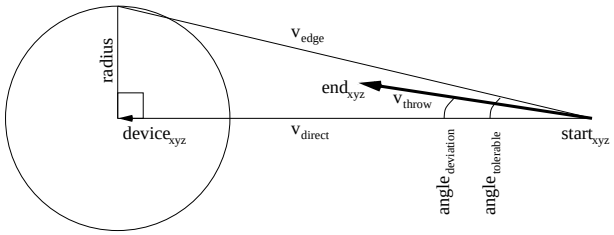

Figure 3. Hit detection calculations.

This computation is illustrated in figure 3 and done mathematically through the following steps.

- Compute a line, v_{direct}, going from $start_{xyz}$ to $device_{xyz}$, i.e. a direct line from the start of the virtual throw to the center of the device.
- Compute $angle_{tolerable}$ between a thought line, v_{edge}, and v_{direct}, where v_{edge} goes from $start_{xyz}$ to the edge of the device's sphere of influence $device_{xyz} + radius$. If we simply compute the distance between the center of the device and the start position of the throw, i.e. the length of v_{direct}, this angle is trivial to compute as shown below.

$$angle_{tolerable} = \arctan\left(\frac{radius}{||v_{direct}||}\right)$$

- Then compute another line, v_{throw}, going from $start_{xyz}$ to end_{xyz}, i.e. the line or initial trajectory for the virtual throw itself.
- Then, from the two lines v_{direct} and v_{throw}, make use of the dot product to compute the angle between the lines.

$$angle_{deviation} = \arccos\left(\frac{v_{direct} \cdot v_{throw}}{||v_{direct}||||v_{throw}||}\right)$$

- Now, $angle_{tolerable}$ specifies the angle between the start of the virtual throw and the edge of the device's sphere of influence relative to a straight line to the center of the device. Also, $angle_{deviation}$ specifies the actual angle between the throw and the center of the device. Therefore, if and only if $angle_{deviation} \leq angle_{tolerable}$ the device is hit.

If the device is not hit it ignores the packet, otherwise it inspects the payload to see how it should react.

It is important to point out that as the device is allowed to interpret the virtual throw freely, nothing stops a single device from monitoring several spheres of influence at completely different locations. This means that a single device can place spheres covering "dead" objects as well, i.e. objects that are not *Eventcasting* enabled, network enabled or even electronic devices at all. This feature is suitable for fixed infrastructures that involve lamps, doors, windows, etc. which all have a fixed physical location and are easily controllable from a single device. Also, less capable embedded devices can have a master device that performs the monitoring on their behalf.

3.3.2. Reacting on the Payload. As the content of the payload is free to vary depending on which type of device it is intended for, this section will only give some examples of classes of events and payloads that could be of use.

- **Binary event** Used for having a device switch between two states, e.g. on and off. When receiving this payload, the device simply switches to the opposite state. Typical examples are turning a monitor on/off, turning lamps on/off and locking/unlocking doors.
- **Variable event** Event containing a value within some given range. Typical examples could be to adjust the volume of a speaker or adjust a light dimmer.
- **Execution event** Used to execute an arbitrary program on the receiving device. This offers great flexibility since ordinary applications, daemons and command line programs can be set running with ease.

3.3.3. Feedback to the User. An important key issue in human-computer interaction is to provide instant feedback to the user so that she can assess the outcome of the action performed. Such feedback needs currently be implicitly provided as part of the device's reaction in the real world. E.g. a lamp reacts by going light or dark, and that is the only feedback the user will get. Still, to allow interaction in more advanced and subtle ways some other form of feedback is necessary, e.g. information presented in a user's head mounted display. However, how to provide such additional feedback is not within the scope of this paper.

3.4. Specifying the Payload

Because all except the first class of events given in section 3.3.2 require the user to specify the payload in advance, there is clearly a need for the user to be able to specify this in a similarly natural fashion before the throw is initiated. There are two ways by which this can be accomplished.

- **Voice** Just before the user initiates the throw, she utters a word that is then processed by a voice recognition system. Certain keywords can thus be used to specify what kind of payload to use for the upcoming

throw. For example, uttering the word “light” and gesturing towards a certain lamp or light switch could turn it on, while “darkness” would have the opposite effect.

- **Gesture** Before the throw, some specific hand gesture is made. This gesture can be recognized by employing hidden Markov models, similar to what is done in [8], and thus used to select a certain payload. For example, making a circular motion and then gesturing towards a fan in the ceiling could be used to selectively turn it on, while lamps nearby would not be affected.

The user must also have some way of knowing which payloads are applicable to a certain device. Currently, the idea is to use regular words describing the desired effect — a typical example is using the word “light” for turning on a lamp. Of course, the more devices can be controlled and the more advanced interaction they allow, the need for the user to be informed of acceptable payloads becomes apparent. For the time being, however, focus is on using those words and gestures which are intuitive for the user to apply on a certain device. This leverages her existing knowledge about them and allows for intuitive operation.

A useful side effect of the need for a payload specification mechanism is that it helps solve the problem of sensor drift. Some positioning sensors (e.g. inertial tracking) are sensitive to drift and need to be recalibrated or reset frequently at a known position. This calibration can be incorporated in the payload specification mechanism by enforcing the user to place her hand at a known location, and issuing the recalibration at the same moment. This makes the inertial sensor provide correct readings for a short period of time, enough to move the hand about and perform a throw accurate enough in an arbitrary direction. For example, the user could place a hand at the chest, utter a word to specify the payload and recalibrate, and then perform the throw — all in a very natural and fluent manner.

3.5. Limiting Event Scope

When a virtual throw is performed its scope is limited only by the size of the network over which the packet is sent. This means a user can control devices that are both close at hand, as well as devices that literally lie beyond the horizon. The latter would, obviously, require either a very large sphere of influence for the device or a very accurate throw to be feasible, but in theory the scope of a virtual throw is unlimited. Although useful at times, it may not be desirable to let an event reach anything further away than what is in the user’s immediate surrounding. This calls for a way by which a virtual throw can be blocked in order to limit how far it will reach.

A way of limiting the scope is to take the physical infrastructure in account so that a virtual throw can be blocked by walls and similar physical obstacles. This would require each device to know about the obstacle’s position and shape, e.g. using a model of the building’s architecture, so that it can determine whether the event can reach the device from where the throw was initiated.

Another way of limiting the scope can be to compute the distance between the user and the device, and simply ignore any events that are thrown by users located too far away. It is also possible to use the velocity of the throwing motion to specify how far an event should reach, just as the velocity of a regular throw decides how far an object is thrown.

3.5.1. Ambiguous Hits. An ambiguity can occur when more than one device lies in the trajectory for a virtual throw. This is, however, only a problem when the devices belong to the same class, e.g. a set of lamps, since such devices would all react on the same event. Ways to resolve the ambiguity is to allow devices to, somehow, negotiate among themselves to decide which one should react. However, in certain settings this is not needed. For example, in a corridor with controllable lamps it can be desirable to allow an event to pass through without being blocked, as it thereby lets a user standing in the end of the corridor affect all of them with a single virtual throw. For different types of devices, and because the payload can be specified for a certain type of device, the ambiguity is implicitly resolved as only the corresponding device will react on the event.

4. Prototype

A prototype of the *Eventcasting* system has been implemented as a proof of concept. The prototype consists of a client running on the wearable computer and servers running on standard PCs and laptops. Lacking suitable 3D positioning sensors, an ordinary computer mouse was chosen for positioning; by placing it at a known starting point in the room, the client can perform a throwing motion in 2D aligned to the ground plane. The resulting trajectory is sent via WaveLAN onto the network using IP multicast, so that the servers can react on the payload once hit.

An informal test of the prototype was conducted in our office environment where a number of devices (e.g. a monitor, PDA and telephone) and other objects (e.g. a plant, whiteboard and door) were monitored. The user testing the prototype attempted to hit these objects by issuing virtual throws, i.e. holding the mouse and performing a throwing motion along the surface. When an object was hit, the corresponding server displayed a message for the user.

The results from this testing show that the user could aim for and hit a designated object without problems. It should be stressed, however, that the prototype is very restricted by its use of 2D positioning with a mouse, thus limiting the significance of these results. Once we replace the mouse with real 3D positioning sensors, more accurate and valid conclusions can be drawn about the method’s usability.

5. Discussion

We consider *Eventcasting* a complement to other interaction methods, not a replacement for controlling every kind of device. Its advantage lies in not encumbering the user with handheld or finger operated equipment, but to instead use a throw to select and affect a device. Because performing a throwing motion is fatiguing in the long run, it is not suitable to use for devices that require repetitive actions. However, for devices needing more sporadic control, the motion required can in fact be seen as healthy exercise.

The main difference between a throw in the real world and the virtual throws employed by *Eventcasting* is that in the latter case there is no physical object being released, only an event as imagined by the user. The events being thrown are also not affected by gravity or other laws of nature. These are two significant differences that is likely to affect how users perceive the interaction method. It is unknown whether this method will offer the user an actual feeling of throwing, or whether it becomes a learned skill that requires a new way of performing a throw. To get a definite answer to these questions a user study is needed.

6. Conclusions

We have presented *Eventcasting* — a novel form of interaction that allows a wearable computer user to interact with the surrounding environment in a highly natural and intuitive fashion. The main contribution is the use of a throwing motion to affect the surrounding while leaving the user's hands unencumbered and free to operate in the real world.

The wearable computer monitors the position of the user and her hand, allowing an event to be created when the user performs a throwing motion. The event is sent wirelessly via IP multicast onto the network so that devices, knowing their own position, can detect whether they are hit. Once hit, a device reacts on the payload which the user specified in advance by use of voice, gesture or other means.

The concept will work independently of which positioning system is used, allowing it to be further refined and sophisticated as new technology becomes available. When the hardware used is truly wearable, e.g. sensors embedded in rings or wrist watches, the means for interaction becomes unobtrusive and transparent for the user. In essence, the benefit for the user is that the real world becomes the primary interface, thereby offering great potential for bridging the gap between wearable and pervasive computing.

6.1. Future Work

To determine the usability of this interaction method, a more comprehensive user study needs to be conducted. In that study, the hardware used needs to be more natural so that the subject can get a feeling for how it will be to use it in real life. This requires a usable prototype being built and incorporated in one of our wearable computers. The event distribution model also needs to be further investigated.

7. Acknowledgements

This work was funded by the Centre for Distance-spanning Technology (CDT) under the VITAL project, and by the Centre for Distance-spanning Health care (CDH). The authors thank Dr. Michael Beigl and the anonymous reviewers for insightful comments and advice given.

References

[1] P. de la Hamette, P. Lukowicz, G. Tröster, and T. Svoboda. Fingermouse: A wearable hand tracking system. In *Adjunct Proceedings of the 4th International Conference on Ubiquitous Computing*, pages 15–16, September 2002.

[2] T. Höllerer, D. Hallaway, N. Tinna, and S. Feiner. Steps toward accommodating variable position tracking accuracy in a mobile augmented reality system. In *AIMS '01: 2nd International Workshop on Artificial Intelligence in Mobile Systems*, pages 31–37, Seattle, WA, August 2001.

[3] T. Kindberg and K. Zhang. Secure spontaneous device association. In *5th International Conference on Ubiquitous Computing*, pages 124–131, October 2003.

[4] N. Kohtake, J. Rekimoto, and Y. Anzai. Infopoint: A device that provides a uniform user interface to allow appliances to work together over a network. *Personal and Ubiquitous Computing*, 5(4):264–274, 2001.

[5] J. Nord, K. Synnes, and P. Parnes. An architecture for location aware applications. In *35th Annual Hawaii International Conference on System Sciences*, volume 9, January 2002.

[6] J. K. Perng, B. Fisher, S. Hollar, and K. S. J. Pister. Acceleration sensing glove (ASG). In *International Symposium on Wearable Computers*, pages 178–180, 1999.

[7] M. Satyanarayanan. Pervasive computing: Vision and challenges. *IEEE Personal Communications*, August 2001.

[8] T. Starner, J. Auxier, D. Ashbrook, and M. Gandy. The gesture pendant: A self-illuminating, wearable, infrared computer vision system for home automation control and medical monitoring. In *International Symposium on Wearable Computers*, Atlanta, GA, October 2000.

[9] E. Vildjiounaite, E.-J. Malm, J. Kaartinen, and P. Alahuhta. Location estimation indoors by means of small computing power devices, accelerometers, magnetic sensors, and map knowledge. In *Proceedings of the 1st International Conference on Pervasive Computing*, pages 211–224. Springer-Verlag, August 2002.

[10] A. Ward, A. Jones, and A. Hopper. A new location technique for the active office. *IEEE Personal Communications*, 4(5):42–47, October 1997.

[11] Y. Wu and T. S. Huang. Vision-based gesture recognition: A review. In *Proceedings of the 3rd International Gesture Recognition Workshop*, volume 1739, pages 103–115, 1999.