Revealing the Invisible: Visualizing the Location and Event Flow of Distributed Physical Devices

Nicolai Marquardt[1], Tom Gross[2], Sheelagh Carpendale[1], Saul Greenberg[1]

[1]Dept of Computer Science, University of Calgary
Calgary, AB, CANADA, T2Y 3V1

[2]Faculty of Media, Bauhaus-University Weimar,
99423 Weimar, GERMANY

{nicolai.marquardt, sheelagh, saul.greenberg}@ucalgary.ca, tom.gross@medien.uni-weimar.de

ABSTRACT

Distributed physical user interfaces comprise networked sensors, actuators and other devices attached to a variety of computers in different locations. Developing such systems is no easy task. It is hard to track the location and status of component devices, even harder to understand, validate, test and debug how events are transmitted between devices, and hardest yet to see if the overall system behaves correctly. Our *Visual Environment Explorer* supports developers of these systems by visualizing the location and status of individual and/or aggregate devices. It visualizes the current event flow between devices as they are received and transmitted, as well as the event history. Events are displayable at various levels of detail. The visualization also shows the activity of applications that use these physical devices. The tool is highly interactive: developers can explore system behavior through spatial navigation, zooming, multiple simultaneous views, event filtering, details-on-demand, and time-dependent semantic zooming.

Author Keywords

Event flow visualization, prototyping, physical and tangible interfaces, geographical map overlays, distributed systems.

ACM Classification Keywords

D.2.6 **Programming Environments**: *Programmer workbench*; H.5.2 **User Interfaces:** Prototyping

General Terms Human Factors, Design

INTRODUCTION

Physical and tangible user interfaces [8] let people interact with and benefit from digitally-controlled physical devices situated in their everyday environment. Motivated by *ubiquitous computing* (ubicomp) ideas [25], these systems comprise devices embedded in homes and workplaces, in a manner that fuse digital and physical interaction [3,14]. They usually include a variety of *input sensors* (e.g., motion, light, temperature, or distance sensors), *input controls* (e.g., physical buttons, dials, or sliders), and *output actuators* (e.g., displays, motors, or lights). They form a distributed system when device inter-operation is computer controlled over a network.

TEI'10, January 24–27, 2010, Cambridge, Massachusetts, USA.

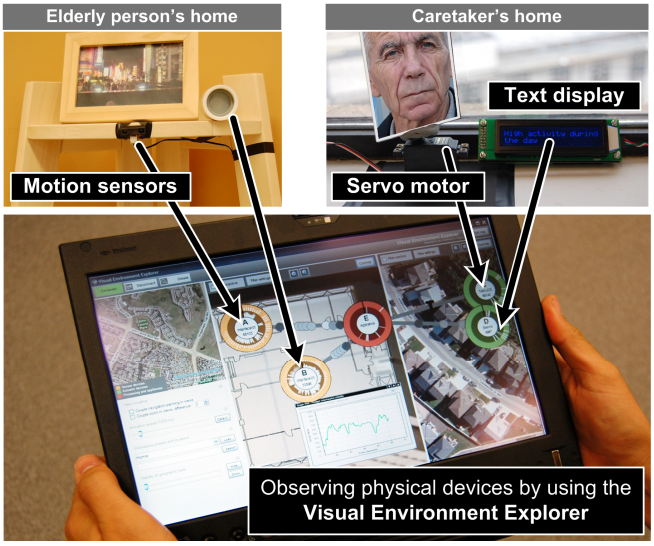

Figure 1. Exploring distributed sensors and actuators.

Developers can use various toolkits to prototype such systems (mostly research, but a few are commercial, e.g. Phidgets [www.phidgets.com] or Arduino [www.arduino.cc]). The majority facilitate programming of locally connected devices [12,13,24], although several recent toolkits allow similar access to distributed devices [16,21,22]. Such toolkits radically lower the barrier for developers to use physical devices in distributed settings. Even so, developing these systems is no easy task. It is hard to track the location and status of component devices, even harder to understand, validate, test and debug how events are transmitted between devices, and hardest yet to see if the overall system behaves correctly [15,17].

To address these issues, we developed the *Visual Environment Explorer* (VEE). Using VEE, developers can visualize ongoing activities of the distributed physical interface infrastructure (e.g., Fig. 1). Specifically, VEE displays the location and status of individual and/or aggregate devices. It visualizes the received and transmitted event flow between devices, as well as the event history. The visualization also shows the activity of active applications that control these interconnected physical devices. Developers can also interactively explore: system topology through spatial navigation and zooming; system behavior by examining event transmission/reception at various levels of detail (including filtering) and with multiple simultaneous views; and system history through time-dependent semantic zooming.

Figure 2. The VEE user interface: visual exploration of distributed sensors and actuators.

This paper describes VEE. First, we illustrate its basic workings by showing how VEE supports a person prototyping a simple distributed physical user interface. We then explain how developers interact with the various device and event visualizations offered by VEE. Subsequent sections describe its implementation, limitations and future work, and summarize related work in visualizing ubicomp system operations.

SCENARIO: PROTOTYPING AGING IN PLACE UBICOMP

We illustrate the basic visualization provided by the *Visual Environment Explorer* through a scenario, where we show how Sam (a developer) uses VEE to help prototype a distributed physical user interface. For clarity, our scenario uses a deliberately simplified set of interoperating devices.

The idea. Sam is prototyping an *aging in place* environment [4,18] containing several devices at two different locations. The elderly person's home will contain two motion sensors that monitor the elder as he moves around the home (Fig. 1, top left). At the caretaker's home (e.g., the elderly person's adult child), a picture of the elder affixed to a servo motor will wiggle to reflect the elder's activity as detected by the motion sensors. In addition, a two-line LCD display will summarize the elder's overall activity during the day (Fig. 1, top right). In this way, the caretaker will maintain remote awareness of the elder's current and daily activity; no activity is cause for concern and follow up.

Setting up the environment. Sam strategically places two Phidget motion detectors [10] in the elder's home, which communicate with a computer running the Shared Phidgets toolkit [16]. Similarly, Sam affixes the picture frame (using clay) atop a Phidget servo motor, and places it and a Phidget text display on the mantle in the caretaker's home; both are attached to another computer also running Shared Phidgets. Under the covers, Shared Phidgets seamlessly manages access and control of these distributed devices via a *distributed Model-View Controller* (dMVC) architecture [16].

Spatially locating devices. Sam starts VEE, which automatically hooks into the Shared Phidget's dMVC. Sam sees the four attached devices. He brings up VEE's geographical map view as shown in Fig. 2, and uses the map controls and/or the controls in Fig. 2b to split the view, and to navigate/zoom into the elder and caretaker's home locations (e.g., by searching for a city name, or selecting previously saved locations). Sam has crafted a floor plan of the elder's home, and adds that to the view (Fig. 2e). He then places the motion sensors and actuators at these two locations. Fig. 2e is the elder's home; its two motion sensors represented as orange circles (Fig. 2c) on the floor plan. Fig. 2f is the caretaker's home: the moving picture frame and the text display are represented as green circles. The overview map in Fig. 2a shows a zoomed-out view.

Observing sensor activity. Sam also sees a live overview of the motion sensors' activity by observing each of the orange circles (Fig. 2c). The bar charts at their centers indicate the recent motion activity in front of the corresponding sensor. Sam then opens information panels that reveal more details about each sensor, e.g., the detailed graph of sensor values in Fig. 2d. This provides valuable

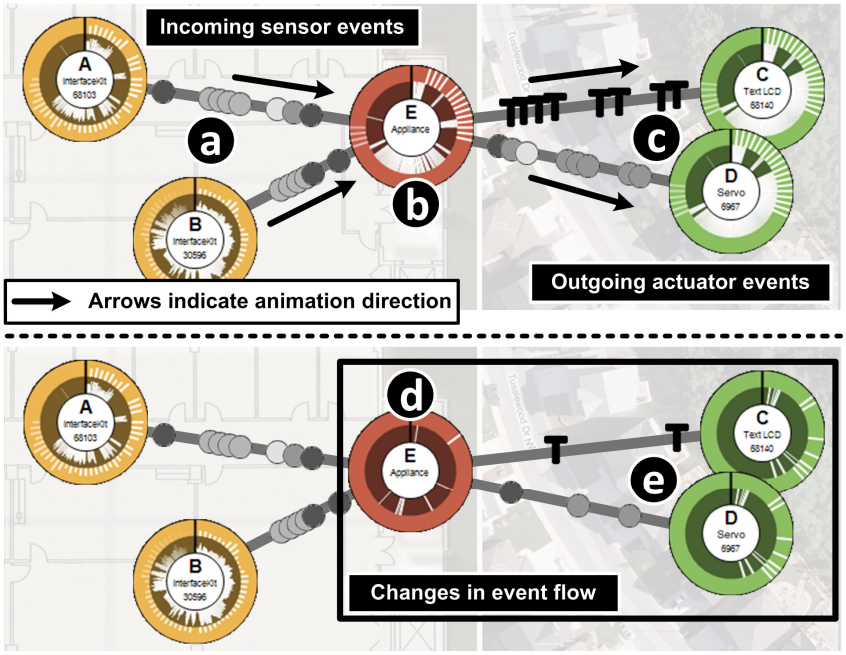

Figure 3. Observing changes in the event flow.

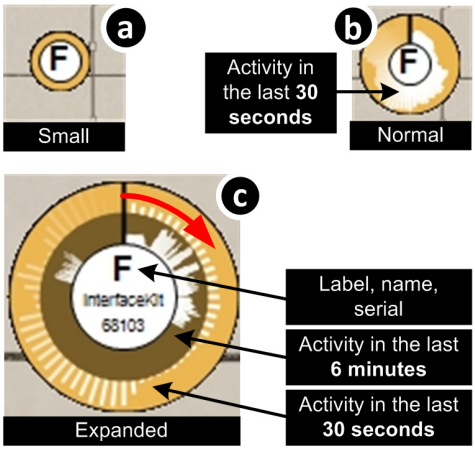

Figure 4. Visualizing device status and activity.

information for Sam; in this case, he sees the typical ranges of the sensed values produced by motion, and will use those ranges to tune the sensitivity of the servo motor's wiggle.

Control of distributed actuators. For each of the distributed actuator devices, Sam opens panels that show device status and controls for altering device properties. These panels allow both the testing of the correct device functionality and the configuration or reset of the device. With the panel in Fig. 2g Sam can change the displayed text on the LCD by typing the message in the textbox (e.g., 'High activity today'). Similarly, Sam can use the slider in Fig. 2h to change the rotation angle of the servo motor.

Programming the application. Sam now implements his software. It will estimate the elderly person's activity from the sensor values, and use those measures to control the amount of servo wiggle and content of the text display. He uses the Microsoft VisualStudio IDE, which has been augmented to include the Shared Phidgets developer library. Sam drags and drops four proxy objects into his project; each represents a particular device type. He then maps these objects to the four physical devices by selecting them on the map. The IDE then automatically generates the basic application framework: the networking of physical devices to corresponding device proxy objects is done, and all necessary event handlers are created. In the code view of the IDE, Sam adds a few lines of code to calculate the average motion activity in the elderly parent's home. He then adds two more lines of code to link the servo position and the displayed text to this average motion activity value. All this takes Sam just a few minutes.

Observing configuration and event flow. Sam starts his application. A representation for this application appears in the map view – shown as the red circle in Fig. 3b,d. Lines from this circle show that the application is connected to each of the four distributed devices. Sam immediately observes the incoming sensor events, where each transmitted event is displayed as an animated icon (Fig. 3a). In a similar way, Sam observes control events that the application sends to change the servo motor position and the displayed text (Fig. 3c).

While observing the runtime behaviour of his prototype, Sam notices that the application is generating considerable network traffic, as it transmits control events for the servo and display at a very high frequency (Fig. 3c). Sam revisits the source code and changes the thresholds that determine how the application responds to incoming sensor values. Sam then observes the runtime behaviour of this changed prototype, where he sees that his changes have, in fact, drastically reduced the number of messages sent to the display and servo (Fig. 3d,e).

While this scenario illustrated a simple distributed physical user interface, it nevertheless highlights how VEE supports various steps in the development, debugging, testing, and deployment process of such systems.

VISUALIZING DISTRIBUTED PHYSICAL DEVICES

We now dive into greater detail about how VEE visually represents distributed physical devices, and the interactions afforded by these representations.

Basic Device Visualization

VEE's basic visualization of any device contains four key characteristics: interactive icons, spatial location, views, and bar charts, as described below.

Interactive icons represent two categories of physical devices. *Sensors* (e.g., motion, distance, light, temperature) are represented as circles with *orange* background color. *Actuators* (e.g., displays, motors, lights) are circles with a *green* background.

Spatial location. Each interactive icon is located atop a 3rd party geographic map (VEE uses Microsoft Virtual Earth [http://dev.live.com/virtualearth/sdk]), or a user-supplied 2D spatial diagram (e.g., a floor plan). These locations

represent the actual or desired location of the physical device. Typically, icons are manually placed atop a location. Alternately, VEE will automatically place a GPS-enabled device at its GPS map location.

Views. Interactive icons have three views, each revealing greater detail (Fig. 4).

- *Small views* embed a unique identifier letter (associated with the device) at the icon's center. Its orange or green border indicates the device type (Fig. 4a).
- *Normal views* wrap a *bar chart* (discussed shortly) of the last 30 seconds of device activity inside the icon's center region (Fig. 4b).
- *Expanded views* include the device type and its unique hardware serial number in the icon's center. The last six minutes of device activity are shown in an additional bar chart drawn between the outer bar chart and the center region (Fig. 4c).

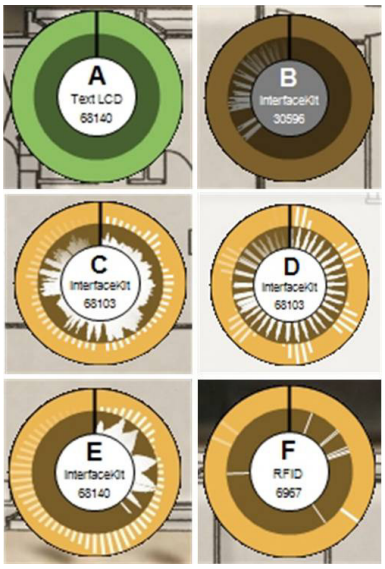

Figure 5. Device visualizations and activity patterns.

View choice is automatic or manual. When a developer zooms in or out of a spatial view, the icon's size increases or decreases in response; the view type switches at certain size thresholds. As well, a developer can manually resize an icon, where again the view type switches accordingly.

Bar chart visualizations provide information about the recent activity of each device. For sensors, the numeric values are mapped to the minimum and maximum range of the bar chart. For actuators, the peaks in the bar chart represent device activity (e.g., changed position of a motor, or changed text of a display).

Through these four visuals, developers can quickly identify particular devices by their labeling and spatial location on a map. As they zoom into particular regions, they can see event details about particular devices via the bar charts. The overview map in Fig. 2a, for example, shows each device in the small view, whereas the main window in Fig. 2c,f uses the expanded view. By manually enlarging particular icons (not shown), they can selectively see greater details about the devices they are interested in.

Recognizing Device Status and Characteristics

The icon representation shows not only the device status, but also reveals specific characteristics of the device activity. For example, Fig. 5A represents an actuator device that is currently connected and available, but that has no recent activity. Fig. 5B shows a device that has become unavailable (e.g., disconnected cable, turned off, or failure), where its color is altered to be shaded. Even so, the past activity of the device remains visible as values in the circle bar chart. Figs. 5C-F show how the bar chart activity pattern provides helpful information about device characteristics. Fig. 5C identifies a sensor with a very high rate of triggered events. Fig. 5D shows a distinctive pattern of events triggered at regular time intervals. Fig. 5E illustrates an activity pattern where its sensor value oscillates in a distinctive way. Fig. 5F, on the other hand, shows a sensor with only infrequent sporadic activity (in this example a reader hardware detecting RFID tags).

Details-on-Demand

To view and interact with device details and device settings, a developer can open specialized views for each device. Shared Phidgets provides *interface skins* that visualize sensor and actuator devices and that provide controls to change device settings [16]. VEE seamlessly integrates these interface skins into its map user interface, where the developer can select the device and choose one of several views via the *view selection dialog* panel (Fig. 6a).

- *Control interfaces (Fig. 6b-d)* show the current device properties (e.g., display resolution of a color display, Fig. 6b), show details about device activity (e.g., found RFID

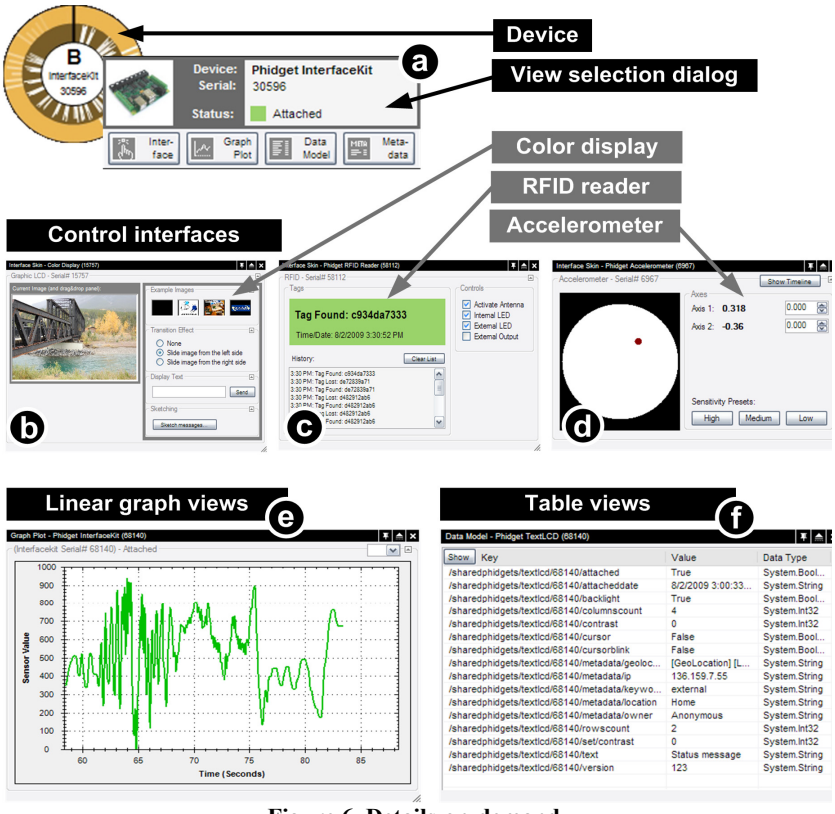

Figure 6. Details-on-demand.

tags, Fig. 6c), and allow changing of device settings (e.g., accelerometer sensitivity, Fig. 6d).

- *Linear graph views (Fig. 6e)* visualize the recent activities of sensors (e.g., temperature, acceleration) as a line graph. While this view has some similarity with the circle bar chart views of the device activity, these linear graph views are more detailed, have higher resolution, and span longer time periods.
- *Table views (Fig. 6f)* list all properties and events of a device in a table. This is a complete (albeit low-level) MVC representation of the current device status and its activities. This view includes sorting and search functionality to find specific entries in the table.

All views are visualized as floating panels atop the map. They are anchored to the current map position so that they move when the map position changes. The developer can easily rearrange and resize these windows, and collapse the views (so that only the title is visible) to occupy less space.

VISUALIZING EVENT FLOW AND INTERACTION

We now explain how our tool visualizes the real-time event flow between devices, and how users can interact with the visualization to obtain the information they need.

The Runtime Application

As shown in the scenario, VEE displays the application icon (as a red circle, e.g., 3b,d), and tracks all devices used by an application. It displays the connections between device components and the application as a node-link diagram (e.g., Fig. 3). The red application icon also displays the incoming and outgoing event activity in a manner similar to the device visualization, i.e., it shows an identification label in the centre and a bar chart portraying the current and past activity of this application (Fig. 3b,d). Thus the application visualization summarizes the activity of all its sensors and actuators. For example, developers can leverage this iconic summary by enlarging the application icon in an overview map view (leaving device icons small).

The Events

VEE animates the event activity of sensors and actuators as seen by the distributed application, where incoming and outgoing events of an application are represented as animated shapes that move along the connection lines.

Specific shapes are used for encoding different types of events, and different grey levels are used to encode values. As shown in Fig. 7 top, *boolean* events are rendered as squares, where their black/white fill represent true/false values. *Numeric* events are circles, where grey levels approximate sensor values ranging from 0 to 1000. *String* events are shapes in the form of the letter 'T', and *binary* data events are squares with rounded corners.

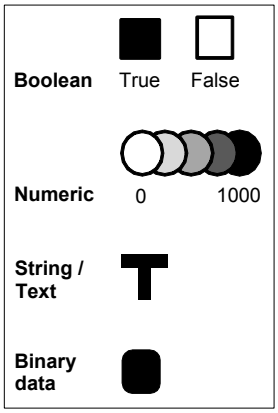

Figure 7. Event encoding.

Revisiting our aging-in-place application in Fig. 3, we see incoming numeric sensor events visualized as circles whose grayscale shading indicates its current numeric value (Fig 3a). Because they range from black to mid-grey, both sensors are detecting moderate motion. We also see outgoing events that change the servo position; here, the grayscale shading represents the transmitted servo position in degrees (Fig 3c). Similarly, we see outgoing events – the letter 'T' – that change the text on the LCD display. Thus developers not only observe the number of transmitted events, but also gain some insight into the event data.

Filtering

Some applications can be quite complex, with large numbers of devices, event types, and connections. To control complexity and visual clutter, developers can apply *filter settings* to only visualize events of a specific type. This reduces the total number of visualized events to only those of immediate interest to the developer.

In the map view, the developer opens the filter settings dialog as shown in Fig. 8b. In this example, the developer has set the filter so that only numeric events whose value ranges between 0 and 200 are shown on the map in Fig. 8a. The filter dialog lets developers filter events that fall into the following categories (or combinations of them):

- *Event data type*. Events can be filtered by their data type, i.e., Boolean, string, integer, and/or binary values.
- *Event content*. Events can be filtered to those holding specific values, i.e., *true* or *false* Boolean values, integer values in a specific range, or string events that contain a given keyword.

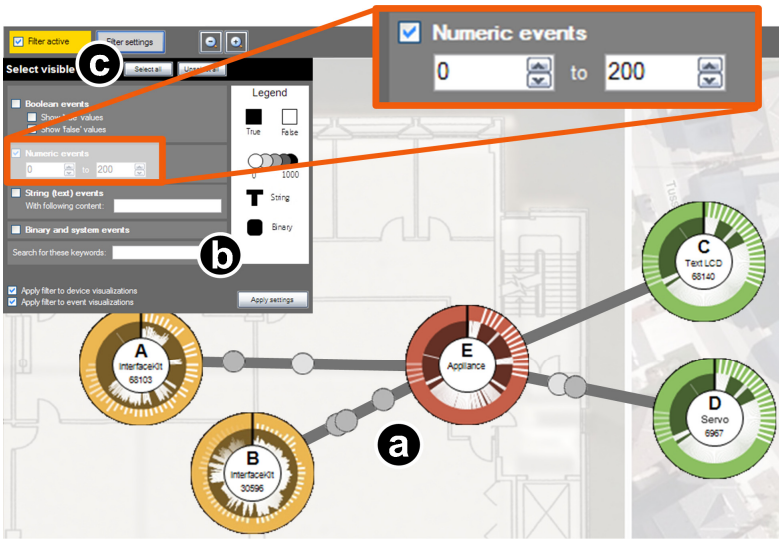

Figure 8. Applying filter settings to show only specific events.

The filter settings are by default applied globally to all current elements of the visualization. However, the filter settings can be selectively applied to the displayed devices, or to separate connections. Once a filter for any of the views is activated, an orange label (Fig. 8c top left) indicates that events in this view are filtered. A checkbox allows one to quickly activate and deactivate the filter without opening the filter settings dialog.

Viewing Multiple Maps Simultaneously

As shown earlier in Fig. 2, VEE can display more than one map view at the same time. When activating the split view of two maps, the visible part of the second map is the same as the main view by default. However, each map has independent navigation and zooming controls; thus one can navigate and zoom to different areas in each map view.

Multiple map views are very useful for visualizing distributed ubicomp, for developers can simultaneously probe different parts of the distributed system at different levels of detail. We saw this in our scenario that included two geographically distant homes (e.g., Fig. 2). Multiple map views can also be applied to the same geographical location, but in this case different filter settings can be applied to each view, e.g., one view showing only numeric events, while another view showing only binary events.

Muting the Geographical Layer

Visualizing the distributed devices at their geographical location on the map provides useful information about the context of the device (e.g., affected area of actuators, sensors nearby, separated areas by walls). At some point, the fidelity of these maps can distract the person when observing the details of a particular running application. VEE allows the developer to set the opacity of the map layer from any value between 0 and 100%. Fig. 2 shows the map with 100% opacity, and Figs. 3 and 8 with only 20% opacity. In the latter cases, the visual focus is on the distributed devices, their connections, and the animated events.

Time-dependent Semantic Zoom

VEE lets developers observe device events as they occur in real-time. This can be overwhelming when large quantities of events are transmitted at a high frequency. For example, Fig. 9a illustrates a situation in which a large quantity of numeric events (circles) is transmitted in a short interval. In such cases, it is difficult to track inter-device activities, single events of interest, or to identify errors in the stream of transmitted events.

To address this issue, the visualization lets the developer slow down the visualization speed to a specific percentage of the real time speed. All the events are still visualized on top of the device connection paths; however, they are animated at a much slower speed and include more in-depth information about the actual event (Fig. 9b). We call this *time-dependent semantic zooming*. This detailed information now includes the transmitted data type of the event, as well as the current value of the event itself (e.g., the integer value of numeric sensor events, or the transmitted text of string events), as illustrated in the close-up in Fig. 9c. This slowed-down visualization facilitates the tracking of specific events. It even allows the developer to immediately freeze the visualization in its current state (this stops all event animations and other visualizations), affording in-depth analysis of the shown (paused) visualizations. When time is set back to normal, the events immediately catch up with real time.

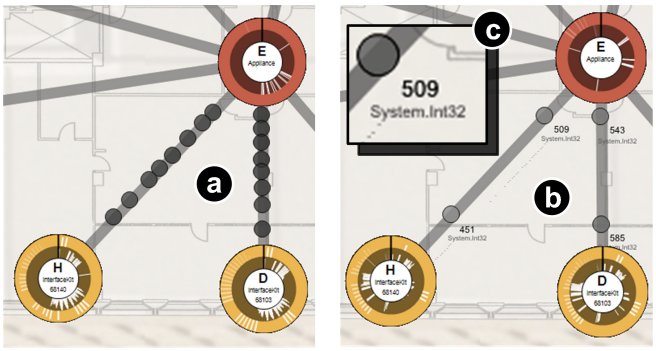

Figure 9. Time-dependent semantic zoom reveals detailed information when slowing down the visualization speed.

IMPLEMENTATION

We now briefly describe the Visual Environment Explorer architecture and how we implemented and integrated it into the existing Shared Phidgets toolkit [16].

The VEE application is implemented primarily in C# (.NET Framework 3.5), including extensions to the source of *Shared Phidgets* toolkit. VEE uses JavaScript to interact with a geographical map web service, where VEE retrieves maps from the Microsoft Virtual Earth web service SDK, and use the Microsoft Research *Map Cruncher* [http://research.microsoft.com/mapcruncher/] to integrate custom overlay maps (Fig. 10a); this tool also allows us to use higher zoom levels than in the default Virtual Earth web service. VEE connects to the Shared Phidgets server (Fig. 10b) using the Shared Phidgets kit's API and programming library [http://grouplab.cpsc.ucalgary.ca/cookbook/]. VEE then registers for updates of sensors, actuators, and running applications. Once a new physical device is connected, the tool subscribes for all events of this device (Fig. 10c).

All currently running applications of the Shared Phidgets infrastructure are monitored (Fig. 10d), and the connections in-between devices are rendered on the map. Events of all devices are forwarded to the filter (Fig. 10e) and, if they pass the filter, are forwarded to a local event buffer (Fig. 10f). The buffer then sends the events to the view triggered by a timer, whose value depends on the visualization speed set by the developer. Events are then visualized as activity indicators in the device representations (Fig. 10g), and as animated events along the links between nodes (Fig. 10h). User interaction is by direct interaction with the map views (e.g., navigation, zooming, details-on-demand), and/or by changing the visualization control settings (Fig. 10i). Here,

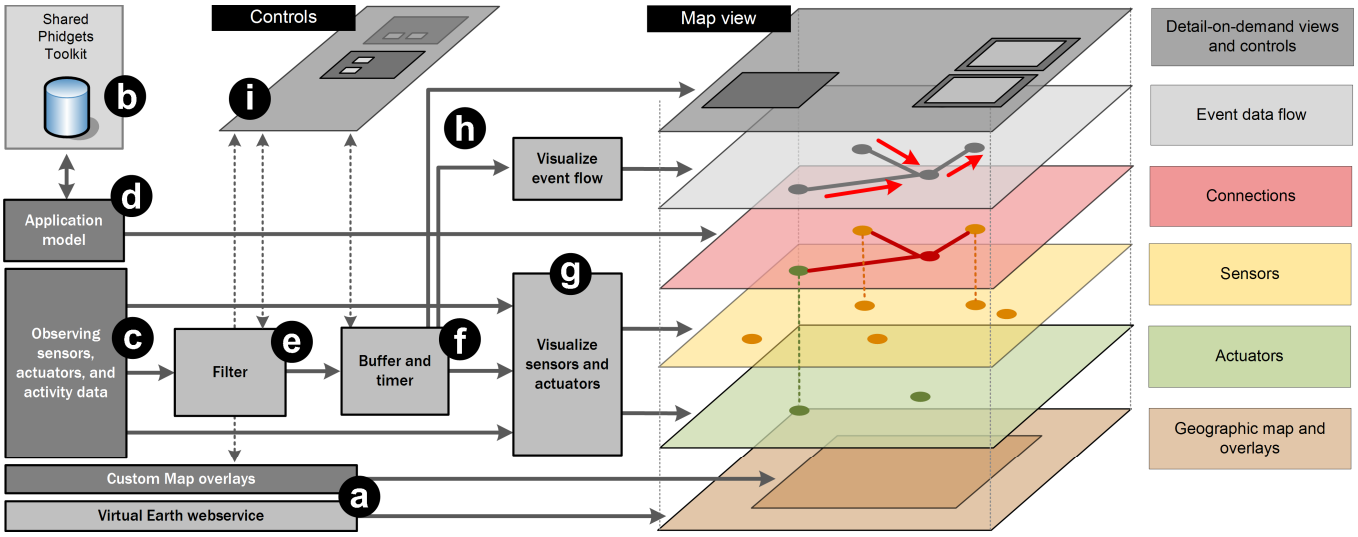

Figure 10. Architecture of the Visual Environment Explorer.

the user interface provides options to change the filter settings, the visualization speed for time-dependent semantic zooming, and the settings of the visualized layers.

DISCUSSION AND FUTURE WORK

While VEE is powerful, it could be improved even further in several ways.

Better control of event history. VEE focuses on visualizing the actual real-time activity of the devices themselves as well as the event flow between them. This information, however, only shows a few minutes of previous device activity. It should be extended to allow the *selection of time intervals,* perhaps via a dynamic query. This would let one explore past activity of the system, and could in turn provide information about the long-term device activity.

Alternative layouts. VEE positions devices on the screen by its simulated or actual physical location. While powerful, alternative 2D representations could be used to reveal other aspects of the relations between devices. For example, devices could be grouped together into clusters depending on their device properties, characteristics, and activity patterns. Alternatively, devices that have connection links with many transmitted events could be located closer to each other than to those that only sporadically communicate. Ideally, developers could switch between the visualizations to match the information they currently need.

Improved links. The connection links in our diagram are currently rendered as straight lines. Yet these links are bi-directional connections, which means that incoming and outgoing events are overlaid atop – and thus sometimes occlude – each other. This could be remedied easily by using opposing arcs that separate incoming vs. outgoing events.

End-user interfaces. Although our visualization is specifically targeting ubicomp developers' requirements, the introduced concepts of the visualization can be also applied to end-user interfaces. Considering that more and more distributed electronic devices might be available in peoples' homes in the near future, the overview and reconfiguration of these devices and their connections would become increasingly important. A user interface that includes the visualization of the spatial distribution of devices and their connections, but provides simplified controls for reconfiguration, would help people to control and manage these (often invisibly working) ubicomp applications in their changing everyday environments.

RELATED WORK

Our work relates to visualizations as used in three areas: ubiquitous computing systems, software programming, and large-scale sensor networks. We briefly review this work.

Some existing systems visualize device connections. The *Orbital Browser* [5] allows configuration of media device connections and provides visualizations of these connections. Similarly, the *Equator Component Toolkit* [6] provides a *display editor* for creating configurations of distributed physical devices and visualizing their connections. The *CollaborationBus* editor [11] also allows the graphical composition of ubicomp applications. VEE extends this work, where we specifically target requirements of ubicomp application developers, e.g., visualizing event flow between devices, or viewing details about specific devices.

Graphical view representations of developed software can facilitate the programming, debugging, and testing process of applications [1,20]. Developers can visually observe application behaviour at run-time to recognize unexpected behaviour [1], detect coordination problems in distributed multi-agent systems [20], or view the internal processes of internet routers and network infrastructures [26]. The Stanford iRoom *Event Heap* visualization [17] used visualizations for the debugging of ubicomp applications. Their *Visualizer* application provides helpful information to developers about the activities in the shared data space, so that developers and users of the systems can detect reasons

for technical breakdowns. In this spirit, VEE also visualizes run-time behaviour, albeit in a different manner.

Geographical overview visualisations have been developed to observe large scale distributed sensor network behaviour. For instance, the *SenseWeb* system [23] provides views of aggregated sensor data (e.g., temperature) through the *SensorMap* application [19]. *GRASS* [7] also visualizes environmental data, for example temperature and rainfall. These systems are optimized to provide users views of aggregated and interpreted information, such as average values of geographical regions, or changes over time. Another class of software tools focuses on visualizing the technical properties of these networks: data traffic, load balance, node attributes, as well as logical links between nodes. Instances of these systems are *EmView* [9] and *SpyGlass* [2]. While VEE includes analogous visual representations of sensors, it also displays individual device/events, actuator devices status, and the detailed event flow between devices.

SUMMARY AND CONCLUSION

We contributed the *Visual Environment Explorer* – a visualization tool that supports developers when prototyping ubicomp applications containing distributed devices such as sensors, controls, and actuators. Using VEE in combination with Shared Phidgets, developers can rapidly build such systems, and observe, test, and debug their run-time distributed behaviour.

In particular, VEE reveals the usually invisible data flow between the distributed device components. It visualizes representations for sensors and actuators on a geographical map layer, reveals the inter-connections between these devices, and shows the actual data flow of events. This allows ubicomp developers to observe the status of devices, as well as the transmission of events and their processing between devices. Various interaction techniques like *detail-on-demand*, *filtering*, and *time-dependent semantic zoom* facilitate the exploration and observation of large device infrastructures with many transmitted events between the single devices.

ACKNOWLEDGMENTS

This research is partially funded by the iCORE/NSERC/ SMART Chair in Interactive Technologies, by Alberta Ingenuity, iCORE, NSERC, and by our industrial sponsor SMART Technologies Inc. We also thank our reviewers for their valuable feedback.

REFERENCES

1. Baecker, R., DiGiano, C., and Marcus, A. Software visualization for debugging. *Commun. ACM 40*, 4, (1997), 44-54.
2. Buschmann, C., Pfisterer, D., Fischer, S., Fekete, S.P., and Kroller, A. SpyGlass: a wireless sensor network visualizer. *ACM SIGBED Review 2*, 1 (2005), 1-6.
3. Buxton, W.A.S. Living in Augmented Reality: Ubiquitous Media and Reactive Environments. In K. Finn, A. Sellen and S. Wilber, eds., *Video Mediated Communication*. Lawrence Erlbaum Associates (1997), 363–384.
4. Consolvo, S., Roessler, P., and Shelton, B.E. The CareNet Display: Lessons Learned from an In Home Evaluation of an Ambient Display. *Proc. of UbiComp 2004*, Springer (2004), 1–17.
5. Ducheneaut, N., Smith, T.F., Begole, J.B., Newman, M.W., and Beckmann, C. The Orbital Browser: Composing Ubicomp Services Using Only Rotation and Selection. *Ext. Abstracts of CHI 2006*, ACM (2006), 321–326.
6. Egglestone, S.R., Humble, J., Greenhalgh, C., Rodden, T., and Hampshire, A. The Equator Component Toolkit: Managing Digital Information Flow in the Home. *Adj. Proc. of UIST 2006*, ACM (2006).
7. Fan, F. and Biagioni, E.S. An approach to data visualization and interpretation for sensor networks. *Proc. of HICSS 2004, IEEE* (2004).
8. Fitzmaurice, G.W., Ishii, H., and Buxton, W.A.S. Bricks: Laying the Foundations for Graspable User Interfaces. *Proc. of CHI 1995*, ACM (1995), 442–449.
9. Girod, L., Stathopoulos, T., Ramanathan, N., et al. A system for simulation, emulation, and deployment of heterogeneous sensor networks. *Proc. of 2nd int. Conf. on Embedded Networked Sensor Systems*, ACM (2004), 201-213.
10. Greenberg, S. and Fitchett, C. Phidgets: Easy Development of Physical Interfaces Through Physical Widgets. *Proc. of UIST 2001*, ACM (2001), 209–218.
11. Gross, T. and Marquardt, N. CollaborationBus: An Editor for the Easy Configuration of Ubiquitous Computing Environments. *Proc. of PDP 2007*, IEEE (2007), 307–314.
12. Hartmann, B., Klemmer, S.R., and Bernstein, M. d. tools: Integrated prototyping for physical interaction design. *IEEE Pervasive Computing*, (2005).
13. Hudson, S.E. and Mankoff, J. Rapid Construction of Functioning Physical Interfaces from Cardboard, Thumbtacks, Tin Foil and Masking Tape. *Proc. of UIST 2006*, ACM (2006), 289–298.
14. Ishii, H. and Ullmer, B. Tangible Bits: Towards Seamless Interfaces Between People, Bits and Atoms. *Proc. of CHI 1997*, ACM (1997), 234–241.
15. Klemmer, S.R., Li, J., Lin, J., and Landay, J.A. Papier-Mache: Toolkit Support for Tangible Input. *Proc. of CHI 2004*, ACM (2004), 399–406.
16. Marquardt, N. and Greenberg, S. Distributed Physical Interfaces with Shared Phidgets. *Proc. of TEI 2007*, ACM (2007), 13–20.
17. Morris, M.R. Visualization for Casual Debugging and System Awareness in a Ubiquitous Computing Environment. *Adj. Proc. of UbiComp 2004*.
18. Mynatt, E.D., Rowan, J., Jacobs, A., and Craighill, S. Digital Family Portraits: Supporting Peace of Mind for Extended Family Members. *Proc. of CHI 2001*, ACM (2001), 333–340.
19. Nath, S., Liu, J., and Zhao, F. Challenges in building a portal for sensors world-wide. *First Workshop on World-Sensor-Web: Mobile Device Centric Sensory Networks and Applications (WSW)*, (2006).
20. Ndumu, D.T., Nwana, H.S., Lee, L.C., and Collis, J.C. Visualising and debugging distributed multi-agent systems. *Proc. of the third annual conf. on Autonomous Agents*, ACM (1999), 326-333.
21. Ringel, M., Tyler, J., Stone, M., Ballagas, R., and Borchers, J. iStuff: A Scalable Architecture for Lightweight, Wireless Devices for Ubicomp User Interfaces. *Proc. of UbiComp 2002*, Springer (2002).
22. Salber, D., Dey, A.K., and Abowd, G.D. The Context Toolkit: Aiding the Development of Context-Enabled Applications. *Proc. of CHI 1999*, ACM (1999), 434–441.
23. Santanche, A., Nath, S., Liu, J., Priyantha, B., and Zhao, F. SenseWeb: Browsing the Physical World in Real Time. *Proc. of IPSN 2006*, ACM/IEEE (2006).
24. Villar, N. and Gellersen, H. A Malleable Control Structure for Softwired User Interfaces. *Proc. of TEI 2007*, ACM (2007), 49–56.
25. Weiser, M. The Computer for the 21st Century. *Scientific American 265*, (1991), 94.
26. Wendlandt, D., Casado, M., Tarjan, P., and McKeown, N. The Clack graphical router: visualizing network software. *Proc. of the 2006 ACM Symposium on Software Visualization*, ACM (2006), 7-15.