Inattention-Management Middleware for Human-in-the-Loop Multi-Display Applications

Max Nicosia
Department of Engineering
University of Cambridge
Cambridge, United Kingdom
lmn27@cam.ac.uk

Per Ola Kristensson
Department of Engineering
University of Cambridge
Cambridge, United Kingdom
pok21@cam.ac.uk

***Abstract*—Operator inattention is an important and unsolved problem in mission critical multi-display systems where a single or a group of operators continuously monitor information flows on distributed displays. In this paper we present a novel system solution to this problem and a middleware for supporting flexible attention-aware applications for a variety of domains. Some of the most significant functionality includes direct querying of the application's attention state, custom callback definitions to be executed on specific attention events or application updates, inter-application message routing, and pushing custom notification with relative location information to any other registered application. We evaluate our middleware by developing three applications that both demonstrate the efficacy and versatility of the system and provide performance estimates in terms of latency as a function of payload size.**

***Index Terms*—inattention management, human-in-the-loop, multi-display, middleware**

I. Introduction

In many domains there is a need for a single or a group of operators to continuously monitor information flows on distribution displays. For example, an operator could be tasked with monitoring several concurrent video feeds of sensor data for objects of interest. Examples of application domains for the above problem specification include automatic hazard identification systems in military vehicles, closed-circuit television (CCTV) detection of suspicious activity, human-aided monitoring of stock market feeds, command and control, etc.

To aid operators in their monitoring task such human-in-the-loop systems are often coupled with machine learning algorithms that attempt to automatically flag potential objects of interest to the operator. However, this does not fully mitigate the problem. In particular, in situations when it is vital an object of interest is correctly identified, such machine learning algorithms must be configured to have a true positive rate (probability of accurate detection) near 100%. However, since machine learning is imperfect, such an extreme operating point on the Receiver Operating Curve (ROC curve), inevitably results in a very high false positive rate (probability of false alarm). In practice, this results in an operator becoming overloaded with false alarms. The problem is further exacerbated the more video streams an operator is required to attend do. For instance, if a military vehicle has six continuous visual sensor feeds scanning for suspected explosives, the operator will need to attend six concurrent visual feeds. While it is in theory possible to mitigate this problem by employing further operators, this is expensive and may not be possible due to constrained space in the vehicle and difficulty of recruiting and training skilled operators.

In this paper we present a system solution to this human-in-the-loop problem, which *complements* existing machine learning-aided systems. We view the problem as an *inattention management* problem. Instead of relying on an operator as being able to attend all incoming objects of interest immediately, we propose developing attention-aware applications that can base their behaviour on whether or not an operator attended a particular object of interest.

Our system solution couples each display with at least one sensor, currently either a commodity eye-tracker or a Kinect depth sensor. The sensors infer whether an operator is attending a display, and if so, which display region an operator is attending. This information can then be used to develop distributed attention-aware applications using an easy-to-use programming application interface (API) provided by our middleware. For example, if an operator fails to attend a flagged object of interest, this event can be used by an inattention-aware application to carry out a variety of actions, such as for instance notifying the operator on the display the operator is attending to, signalling that a particular display require operator attention, stacking unattended objects into a queue for revision later, or flagging an unattended object to another operator, etc.

II. Related Work

Displaying notifications to users and their potential adverse disruptive effects have been extensively studied in the human-computer interaction (HCI) field (e.g. [1]–[3]). Common mitigation solutions include avoid interrupting the user's current task [1] and using context-sensing and the contents of the message to infer a suitable moment to notify the user [3]. Such strategies complement our work and may inform applications running on top of our inattention management middleware.

Attention-aware systems attempt to manage users limited perceptual and cognitive abilities [4]. Such systems can manage attention in various ways, such as by developing toolkits for pushing notifications to peripheral displays [4], or for sensing users' position in relation to a set of displays and expose it for proximity-aware applications [5], [6]. Research indicates that appropriate management of user attention can increase performance [4]. Various strategies have been investigated, such as reducing interruptions, managing context switches and tagging actively used objects [7], [8]. Air traffic control research has studied effects of visualising pertinent information to operators [9]. Overall, such visualisations improve performance in the main task but can be obstructive and do not generalise well. Sometimes simple and subtle visualisations, such as pulsating objects, can be a good middle ground [9]. Recent work has used eye-tracking to highlight unseen changes in radar tasks [10]. They find that several application-dependent factors are vital for successful attention management, such as workload, task and the situation [10].

Dostal et al [11] developed Diff Displays: an inattention-aware multi-display system which used computer vision and RGB cameras to sense whether a user attended a display or not. They then developed several visualisations which dimmed unattended displays in order to reduce user distraction. However, in order to still enable the user to observe changes in unattended peripheral displays, the system provided four visualisations for visualising changes on the dimmed displays [11]. A case study in which a single user used this system as their daily programming desktop station revealed that the system substantailly reduced the number of display switches for the user [11]. The same technology was later turned into a toolkit with attention-aware graphical user interface (GUI) controls [12].

III. System Overview

The inattention management middleware consists of three software components: 1) the world renderer; 2) the local service; and 3) API and sensor data feeder. The sensor data feeder has currently drivers for three sensors: 1) Tobii eye-trackers; 2) Microsoft Kinect v2 depth sensor; and 3) Google Tango Tablet with integrated structure sensor. An operational system requires one instance of the world renderer, one local service instance per machine and one sensor data feeder per sensor for each machine connected to the sensor in question. Finally, each application will need to include the API library.

The primary objective of the middleware is to allow developers to easily deploy distributed attention-aware applications that can amplify operators ability to process visual information on multiple displays. The middleware performs this function by running distributed applications that react to information on which display the user is attending to, and by extension, which displays are left unattended. Applications can thereby for example 1) ensure an operator notices changes (reducing change blindness [13]); 2) ensure an operator does not fail to detect events or changes (reducing inattentional blindness [14]); 3) arrange so that an operator does not become overloaded with information, as unwarranted interruptions can have severe detrimental impact on memory and concentration [15]. In addition, the system supports extensive logging which can be used for instance for compliance or for collecting user attention data which can later be used to train or optimise machine learning algorithms or derive new metrics for measuring operator capacity.

The middleware abstracts information sharing across applications to allow more complex application logic, which can then be used to manage information presentation and user attention information without adding complexity to the application itself.

The middleware provides application support via two fundamental constructs.

The first is event rules, which are callbacks that are triggered on specific events, such as attention changes, application data updates, notification requests, messages received, etc. Any number of callbacks can be associated with each event type. Any type of decision logic can be associated to the event, thus providing an entry point for the application. For example, if the user fails to attend a critical update in one of the displays the application can use the API to notify the user.

The second construct is information filters, which are callbacks that visualise information in a particular way. When an event rule fires, the callback can call an information filter to display information in an appropriate way for the state change. For example, an information filter can highlight a visual object change that has surpassed a particular threshold or dim unattended display to reduce distractions.

Although the middleware is general, four use-cases have been targeted specifically:

1) Operator missed important information change: The application can use the API to insert an event rule which will evaluate the data change and highlight it to the user if necessary. The API can also be used to notify the user on an unattended event.
2) Operator must act on a time-critical event: The API can trigger an information filter from an event rule to highlight the event or use the API to push a notification immediately to an attended display, or alternatively sound an alarm.
3) Operator reacted to an event with incorrect input: If the application can identify the incorrect input, it can use the API to trigger (through an event rule) an information filter to show the incorrect input, push a notification to an attended display, or send a message to show related information that may help addressing the error.
4) Operator is making a decision that requires cross-referencing across many displays: An application can use the API's messaging call and message callbacks handler to instruct other applications to highlight or notify relevant information in other displays.

IV. System Design

The system supports three coordinate systems. The world coordinate system is situated in the physical world and relates displays and operators in metres from a fixed origin. The display coordinate system is defined within each individual display and measures on-screen objects' absolute location on an individual display in pixels. The relative coordinate

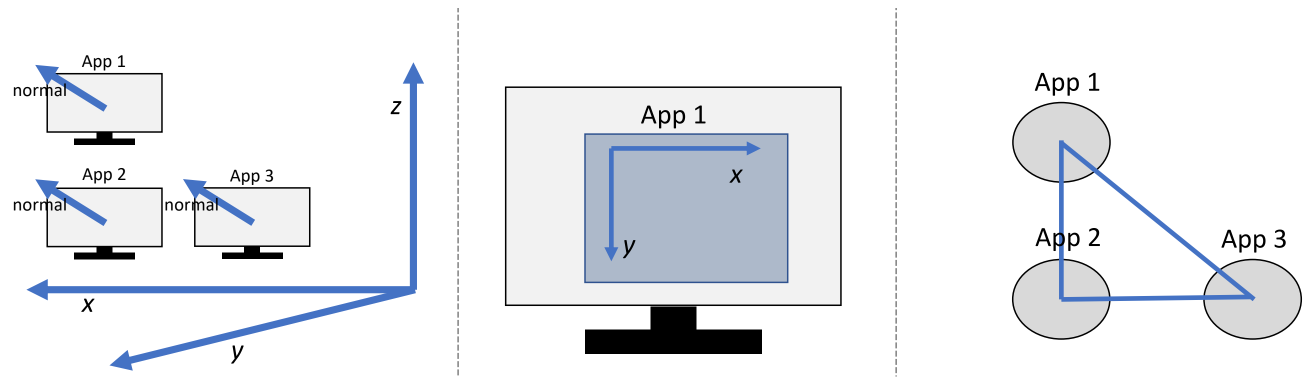


Fig. 1: The three coordinate systems the system uses to manage operator attention. The world coordinate system (left) is situated in the physical world and relates displays and operators in metres from a fixed origin. The display coordinate system (middle) is defined within each individual display and relates objects within a display in pixels relative to the upper-left corner of the display. The relative coordinate system (right) is based on adjacency.

system relates displays to each other by adjacency and these relationships are encoded in an adjacency matrix.

The middleware supports three different types of decision logic for calculating attention focus and for effectively supporting different sensor configurations and use-cases. Applications can choose any decision logic provided that the sensors support it. However, applications can only receive events in a particular coordinate system if they subscribe to it. For example, an application residing on a display that has not provided its own world coordinates in the world coordinate system can receive information from the eye-tracker attached to it (which uses the display coordinate system), but cannot receive information from a structured light sensor that registers a user approaching the display (which uses the world coordinate system). The system will also downgrade the decision logic to the best possible option in the event of sensor failure. The three types decision logic are:

1) Local Knowledge Decisions: Local decisions are made at the local service level. Sensor information is fed directly to the local service, which makes the decision on that data only. As a consequence of only having access to local sensor data, local knowledge decisions cannot handle false-positives or multiple detections from multiple sensors.

2) Global Knowledge Decisions: Global decisions are used when detection information needs to be aggregated to be accurate. Detections are forwarded to the world renderer which then builds a world model that can then be used decide the current world state. Global decisions can then be used to make a decision that for instance removes duplicated detections or false-positives by fusing information from multiple sensors connected to different local services.

3) Arbitrated Decisions: Arbitrated decisions are local knowledge decisions that are verified by global knowledge decisions. They are used to confirm previously unconfirmed local knowledge decisions. For example, a gaze detection can be marked as unconfirmed until a global decision arrives that confirms that the user was indeed attending the particular display with the gaze detection.

A. Transaction and Event-based Architecture

The only constant traffic that the system has to manage is sensor data. Applications only receive messages if either another application sent them a message or a state change affecting that application occurred, such as an attention change or a service change. Such changes will trigger unique transaction messages.

B. Display Configurations

The middleware is designed to provide support for multiple display environments and sensor configurations. This allows the system to support the following three fundamental environments.

1) Room Setup: The system can track the attention of a user, or several users, walking around in a room with displays, which are not necessarily of the same type or size. It is not feasible to use eye-tracking in this setup, currently the system relies on a set of depth sensors. The middleware can currently manage up to two tiers of displays (subject to latency limitations) around a centre point with eight depth sensors in a room measuring $3.5\,\text{m} \times 3.5\,\text{m}$. The latency restrictions are due to the noise of the sensor used (Kinect v2) and the fact that a Support Vector Machine (SVM) classifier is required to make global decisions on user attention acceptably accurate. Other configurations are possible but multi-tier setups will require training a new classifier.

2) Desk Setup: This setup consists of a user sitting in front of up to five displays. Each display is fitted with an eye-tracker. All three types of decision logic can be used in this setup.

3) Combined Room and Desk Setup: This setup is a combination of the previous two setups which allow more versatile sensing than the previous desk setup. Users can be sensed walking towards a display and when they are in range of an eye-tracker the system can seamlessly switch to close proximity detection for higher accuracy. Applications need to be subscribed to both the world and desktop coordinate systems and use either global or arbitrated decision logic.

C. Middleware Supported Features

1) Global Operator Focus of Attention Detection: Applications subscribed to the world coordinate system can receive an attended state if the world renderer infers the user is attending the display the application is running on.

2) Local Operator Focus of Attention Detection: Applications with access to eye-trackers can receive an attended state if a gaze detection intersects an application's working area. In addition, the location of the intersection is given if it intersects registered data in the application.

3) Observed-Unobserved Information Change Tagging: The middleware tracks which application updates have been observed by the user by sub-dividing the application's view-port into a grid. Display cells are marked as observed if a gaze detection intercepts the cell and the decision logic permits it.

4) Unattended Display Detection: The system detects unattended displays and makes this information available to applications.

5) Relative Position in Display Notification: Applications can push notifications to other applications using the request attention call. The notification can optionally appear on other displays in such a way that it physically directs the user to the display requesting attention by for instance an arrow pointing towards the display.

6) Service Update Notification: When a connection to a critical component or sensor is lost, affected applications are notified and all decision logic is recalculated. Once the failure is resolved the system updates all applications with the new service state.

7) Timed Event Rules: In addition to event rules, the middleware allows for callbacks to be called at configurable intervals. For example, they can be used to trigger an information filter to highlight unobserved points that have been left unattended fur a certain duration.

8) Operator Focus and Applications Event Log: The middleware can efficiently log all user and application events. This can be used to make the middleware act as a data tap for machine learning algorithms, for auditing purposes or to allow interfacing with other systems.

9) Inter-Application Messaging Support: The system provides a message delivery abstraction that applications can use to share information. This functionality can be combined with event rules. For example, when a data point change is over a threshold, the application can instruct other applications to highlight related information.

10) Legacy Application Support: Legacy applications are supported by using an overlay application that intercepts data feeds, detects changes and draws on top of the legacy application to show information filters as a translucent overlay.

V. IMPLEMENTATION

The current developer API is implemented in JavaScript. The application's callbacks must be registered before initialisation since the initialisation process depends on the coordinate systems the application is registering.

The remaining components in the system is implemented in C++, with the exception of the Tango Service, which is implemented in Java using the Google Tango API. The SVM is trained using the gesture recognition toolkit (GRT) [16]. All C++ components follow a pipeline architecture where each message is represented as a transaction going through the pipeline. Each pipeline stage run in its own thread and is connected with signal-dependant queues.

The system can be used both when a user is walking around in a display configuration within a room and in a desk environment where the user is sitting down in front of a set of displays. The latter configuration is realised by the system using eye-tracking. The former environment requires an instrumented room with a sensor network of Kinect depth sensors.

The Kinect depth sensor-generated skeletons are insufficient for correctly inferring which display a user is attending to in the vertical plane when displays are stacked vertically. To allow this feature our system complements depth sensing with a pitch classifier which infers two class labels: top display and bottom display. As a feature vector we use detection position in metres (x, y, z), pitch from each Kinect in the system, yaw, and the previous Kinect reference. We trained the SVMs based on data from 12 participants (6 male and 6 female). Participants were instructed to identify a target drawn on a single display at at time. Once the target was detected participants were instructed to gaze at it until it disappeared. The SVMs generate raw predictions of the pitch of the user every 24 frames. The system filters these predictions over a sliding window using two majority voting filters with different minimum votes. The latency for detecting an attention switch in this configuration is approximately 500 ms.

The system consists of the following implemented functions:

1) Kinect Sensor Data Collector: This collector extracts the position and rotation of the user based on the Kinect tracking skeleton (head joint) and face tracking data at 30 Hz. Rotations are filtered using an adaptive low-pass filter [17] $(\beta = 1000, fc_{min} = 0.0001)$.

2) Eye-tracker Sensor Data Collector: This collector retrieves gaze and head rotation data from the eye-tracker using the Tobii's low-level C API at maximum sample rate (90 Hz).

3) Local Service: This service is a mediator and router between the world renderer in the server and local client applications. All messages from and to the application pass through this component. Its secondary role is to process and forward eye-tracker gaze detections to the application and head rotations to the world renderer. As an optimisation, gaze data is only sent to the application if the gaze data changes the attention state or if it intercepts a cell in the application's view-port with data in it.

4) World Renderer: The world renderer calculates intersections, handles global decision logic, routes messages and propagates attention signals and notifications.

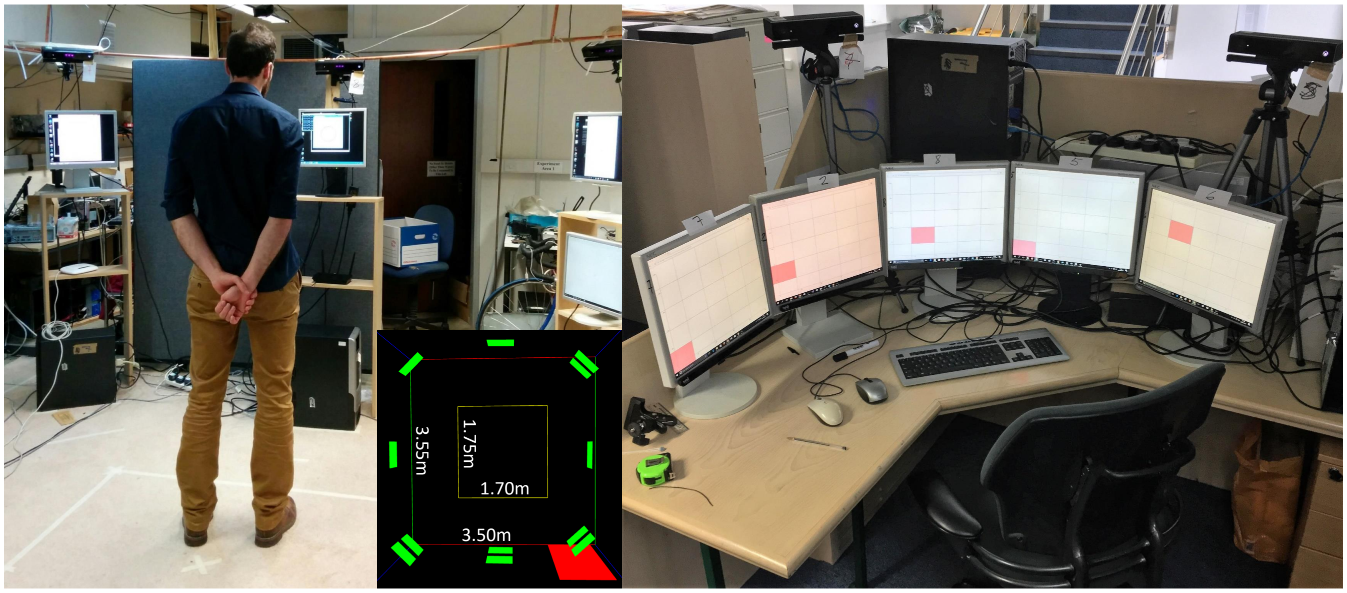


Fig. 2: Room setup with eight Kinect depth sensors (left). A combined desk setup with five displays and two Kinect depth sensors (right).

TABLE I: Latency by Payload Message Size

Size (Byte)	mean (ms)	stddev (ms)	min (ms)	max (ms)
10	41	16	13	78
100	43	15	16	74
1000	50	15	18	83
10000	51	11	33	77

VI. Evaluation

We evaluated our system in two different environments. In the first environment the displays were distributed across a room and the user could walk around freely while the system tracked the displays the user attended. In this environment the user could also optionally hold a tablet device, which the system also tracked (see Figure 2 left).

The setup occupied an area of 3.50 m × 3.55 m with two tiers (0.92 and 1.42 m) of eight displays around the area's centre and eight Kinect v2 sensors suspended from a fixture hanging from the ceiling equidistant to each other above the displays at ~1.7 m from ground, 1.5 m away from the centre.

The Kinect sensors were calibrated by taking several readings of a user standing in different locations and running them through a non-linear least square solver to minimise the error of the computed transformation matrices across the sensors.

In total there were 13 target displays, 12 displays with a display area of 35 cm×33 cm each and one Google Tango tablet with a display are of 19 cm×11 cm.

In the second environment the user sat in front of five displays set up bezel-to-bezel on a desk, each display connected to a separate computer driving a single eye-tracker. Two Kinect sensors were placed with tripods above the monitors to detect users approaching the setup (see Figure 2 right).

After verifying accurate detection of display attendance in both environments, we proceed to evaluate the latency of the system. Testing was carried out on machines with Intel i7-4790 processors and 8GB RAM running Windows 10 64-bit.

A. Switch Latency

We measured latency when the system handled display switches both between adjacent displays and displays that were the furthest away in the adjacency matrix. Mean adjacent display switch latency was 8 ms (n=44, sd=6 ms). The mean latency for display switches between displays the furthest away was 14 ms (n=17, sd=5 ms).

B. Gaze Locking Latency

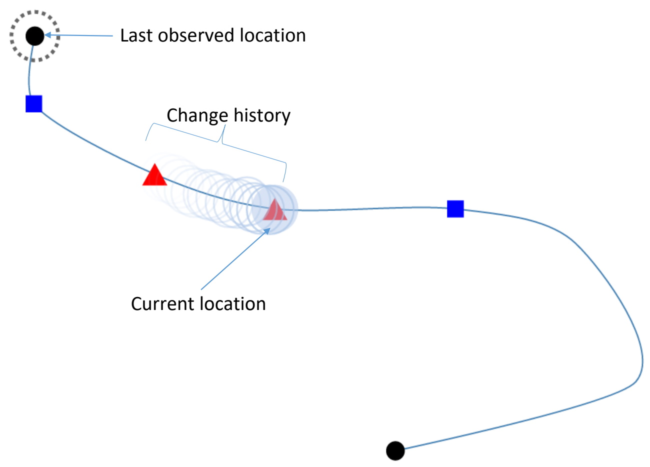
Assuming perfect operator behaviour the mean latency for a point to be detected as attended by the user was 20 ms (n=35, sd = 6 ms). In realistic operational situations the system detected a point as attended by the user in 80 ms (n=75, sd = 41 ms). The latter measurement is based on a user attending 15 points over five trials.

C. Transaction Latency

Transaction latency tests were carried out in a closed network with a high-speed switch with 50 transactions with a 50 ms delay in between each transaction. Table I shows latency by payload message size. Mean latency by request varied between 69 and 84 ms.

D. Applications

We implemented three applications to test the system's functionality and demonstrate the use of event rules and information filters. The first application was a connected graph application that used an information filter to display the history of changes in the graph in different alpha values (node and edge sizes) and an event rule that called an information filter to dim itself (which dimmed the display) when it was



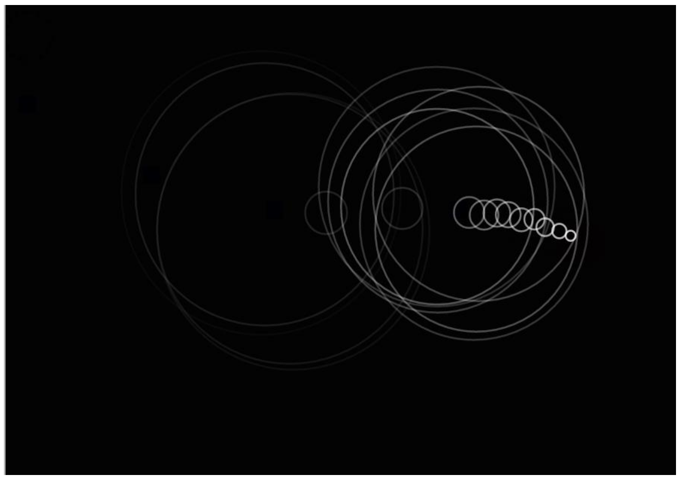


Fig. 3: The application shows an object moving along a fixed trajectory (left). When the display is unattended it is automatically dimmed to reduce operator distraction using an event rule (right). In the dimmed display, delta changes in the underlying application are made available to the operator's peripheral vision by being visualised using an information filter.

unattended by the user. The second application was a variation of the previous application, which, in addition used an event rule requesting a notification when a visual change in a display exceeded a threshold. The third application showed a point moving through checkpoints. This application used an information filter to display its movement history and an event rule to notify the user at specific checkpoints. In addition, it dimmed the display when unattended (see Figure 3).

VII. Discussion and Conclusions

Operator inattention is an important and unsolved problem in many mission-critical multi-display systems. In this paper we have presented a novel system solution for operator inattention in situations where a single or a group of operators continuously monitor information flows on distributed displays. We have presented the design of a flexible inattention-management middleware which supports deployment of versatile distributed attention-aware applications that can support multi-display operators in a variety of domains for a wide range of tasks. We implemented three applications that run on our middleware, tested our system in two different environments, and evaluated the latency of the system. We found that it had sufficiently low latency to support operational requirements. This system solution may be particularly beneficial in automatic hazard detection systems in military vehicles, closed-circuit television (CCTV) detection of suspicious activity, human-aided monitoring of stock market feeds, and command and control.

References

[1] E. Cutrell, M. Czerwinski, and E. Horvitz, "Notification, disruption, and memory: Effects of messaging interruptions on memory and performance," in *Proceedings of Interact*, 2001, pp. 263–269.

[2] S. T. Iqbal and B. P. Bailey, "Effects of intelligent notification management on users and their tasks," in *Proceedings of the ACM Conference on Human Factors in Computing Systems*, 2008, pp. 93–102.

[3] A. Mehrotra, M. Musolesi, R. Hendley, and V. Pejovic, "Designing content-driven intelligent notification mechanisms for mobile applications," in *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, 2015, pp. 813–824.

[4] C. Roda and J. Thomas, "Attention aware systems: Theories, applications, and research agenda," *Computers in Human Behavior*, vol. 22, no. 4, pp. 557–587, 2006.

[5] N. Marquardt, R. Diaz-Marino, S. Boring, and S. Greenberg, "The proximity toolkit: Prototyping proxemic interactions in ubiquitous computing ecologies," in *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, 2011, pp. 315–326.

[6] J. Dostal, U. Hinrichs, P. O. Kristensson, and A. Quigley, "Spidereyes: designing attention-and proximity-aware collaborative interfaces for wall-sized displays," in *Proceedings of the 19th International Conference on Intelligent User Interfaces*, 2014, pp. 143–152.

[7] C. Roda and T. Nabeth, "Supporting attention in learning environments: Attention support services, and information management," *Creating New Learning Experiences on a Global Scale*, pp. 277–291, 2007.

[8] S. D'Mello, A. Olney, C. Williams, and P. Hays, "Gaze tutor: A gaze-reactive intelligent tutoring system," *International Journal of Human-Computer Studies*, vol. 70, no. 5, pp. 377–398, 2012.

[9] J.-P. Imbert, H. M. Hodgetts, R. Parise, F. Vachon, F. Dehais, and S. Tremblay, "Attentional costs and failures in air traffic control notifications," *Ergonomics*, vol. 57, no. 12, pp. 1817–1832, 2014.

[10] B. R. Vallières, H. M. Hodgetts, F. Vachon, and S. Tremblay, "Supporting dynamic change detection: using the right tool for the task," *Cognitive Research: Principles and Implications*, vol. 1, no. 1, p. 32, 2016.

[11] J. Dostal, P. O. Kristensson, and A. Quigley, "Subtle gaze-dependent techniques for visualising display changes in multi-display environments," in *Proceedings of the International Conference on Intelligent User Interfaces*, 2013, pp. 137–148.

[12] J. E. Garrido, V. M. Penichet, M. D. Lozano, A. Quigley, and P. O. Kristensson, "Awtoolkit: attention-aware user interface widgets," in *Proceedings of the 2014 International Working Conference on Advanced Visual Interfaces*, 2014, pp. 9–16.

[13] D. J. Simons and R. A. Rensink, "Change blindness: Past, present, and future," *Trends in Cognitive Sciences*, vol. 9, no. 1, pp. 16–20, 2005.

[14] I. Rock, C. M. Linnett, P. Grant, and A. Mack, "Perception without attention: Results of a new method," *Cognitive Psychology*, vol. 24, no. 4, pp. 502–534, 1992.

[15] E. C. M. C. E. Horvitz, "Notification, disruption, and memory: Effects of messaging interruptions on memory and performance," in *Proceedings of Interact*, 2001, p. 263.

[16] N. E. Gillian and J. A. Paradiso, "The gesture recognition toolkit." *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 3483–3487, 2014.

[17] G. Casiez, N. Roussel, and D. Vogel, "1€ filter: A simple speed-based low-pass filter for noisy input in interactive systems," in *Proceedings of the ACM Conference on Human Factors in Computing Systems*, 2012, pp. 2527–2530.