Vizir: A Domain-Specific Graphical Language for Authoring and Operating Airport Automations

Stéphane Conversy, Jérémie Garcia, Guilhem Buisan, Mathieu Cousy, Mathieu Poirier, Nicolas Saporito, Damiano Taurino, Giuseppe Frau, Johan Debattista

Vizir: A Domain-Specific Graphical Language for Authoring and Operating Airport Automations

Stéphane Conversy, Jérémie Garcia, Guilhem Buisan, Mathieu Cousy, Mathieu Poirier, Nicolas Saporito [1], Damiano Taurino, Giuseppe Frau [2], Johan Debattista [3]

[1] ENAC - Univ. of Toulouse
France
first.last@enac.fr

[2] DeepBlue
Italy
first.last@dblue.it

[3] MATS
Malta
first.last@maltats.com

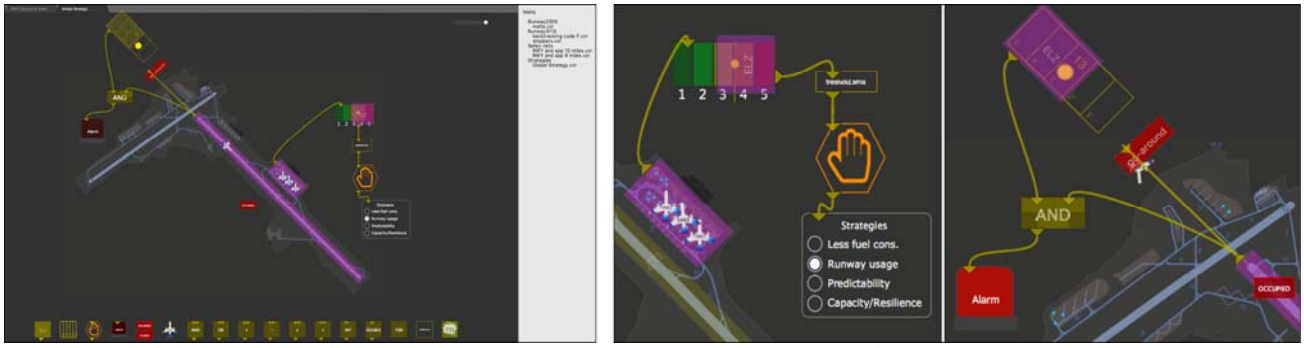

Figure 1: Vizir user interface while authoring automations (left) and examples of airport automations (center, right)

ABSTRACT

Automation is one of the key solutions proposed and adopted by international Air Transport research programs to meet the challenges of increasing air traffic. For automation to be safe and usable, it needs to be suitable to the activity it supports, both when authoring it and when operating it. Here we present Vizir, a Domain-Specific Graphical Language and an Environment for authoring and operating airport automations. We used a participatory-design process with Air Traffic Controllers to gather requirements for Vizir and to design its features. Vizir combines visual interaction-oriented programming constructs with activity-related geographic areas and events. Vizir offers explicit human-control constructs, graphical substrates and means to scale-up with multiple automations. We propose a set of guidelines to inspire designers of similar usable hybrid human-automation systems.

Author Keywords

Automation, Domain-Specific Language, Visual Programming, Air Traffic Control.

ACM Classification Keywords

H.5.2 [Information Interfaces and Presentation]: User Interfaces – Graphical User Interfaces; D.1.7 Software – Programing Techniques – Visual Programming.

INTRODUCTION

Automation is "the execution by a machine agent (usually a computer) of a function that was previously carried out by a human" [41]. The expected benefits of automating an activity include offloading monotonous tasks from human to machines, greater efficiency of the overall hybrid human-machine systems and greater safety in cases of critical activities [41,42]. However, automation raises several challenges, especially in situations in which humans retain control, as automation performances have not been assessed enough. In such hybrid systems, humans need trust and confidence in the automation ability to perform safely, or in their own ability to take control if the automation fails [42].

Air traffic control (ATC) is a complex hybrid system that includes humans and automated processes [22]. ATC aims to organize the movements of aircraft in the air or on the ground with two main objectives: safety and capacity. To face increasing traffic and complexity, further automation is one of the key solutions proposed by international air transport programs [16,29]. However, due to safety concerns, ATC still relies heavily on human operations [22,33]. For example, many clearances (i.e. instructions) from air traffic controllers (ATCOs) to pilots are still performed by voice via radio. In addition, the main problem that field studies and surveys on automation have revealed is the low sense of trust and substantial degree of confusion that pilots have regarding the operation of 'opaque' systems [42].

We believe that the opacity mentioned in [42] is related to the lack of the operator visibility on the program behavior. Since the behavior is opaque, it is difficult to predict, and the operators feel they must blindly trust automation. Operators may prefer to control the operation directly rather than rely on unpredictable automation. This mechanism leads to a dichotomy in the usage of automation (all or nothing [45]) instead of a flexible collaboration between users and automated solutions. Opacity might also prevent operators from recovering from a problematic situation. When automation fails, operators must engage into Knowledge-

based behavior, as opposed to Skills or Rules-based behavior [9,44] that is typical in a nominal situation. Operators must gather information, make sense of it, understand both the causes and the consequences to react appropriately. These tasks lead to a substantial increase of their cognitive workload. To conduct this problem-solving activity, the operator needs an accurate conceptual model of the automation, as well as a correct understanding of the state of the entire system [39]. Therefore, visibility of operations and automation states, as well as program visibility, could be of critical importance.

In this paper, we present Vizir, a 2D Domain-Specific Graphical Language (DSGL) to author airport automations and to operate them while they are running. Vizir's visual constructs lie above the airport map to make automation visible. Vizir blends interaction-oriented programming constructs borrowed from data-flow and event-based programming such as bindings or state-machines, with ATC-specific graphical components that use spatial positioning to produce or react to events. Components can be connected via wires between their inputs and outputs. For instance, a graphical zone positioned over a runway can emit an event when entered by an aircraft, which would activate a light signal accordingly. Other ATC-specific components such as aircraft or ground vehicles are dynamically positioned to reflect their current locations. Vizir also features a component that explicitly represents human control. This component allows users to specify when human validation is required to perform an action.

Visual authoring and operating would bring three expected benefits: 1) Closeness of control and action mapping [39] to foster automation understanding by authors and operators; 2) Visibility of automation status to foster awareness of current and future situation, and human intervention; 3) End-user authoring with an ATC-specific language to foster ATCO implication in the design of the automation. We hypothesize that automations designed with ATCOs would maximize relevance, minimize surprising behavior and maximize predictability.

The paper first describes the research-through-design process we used to design Vizir before detailing its contributions: a combination of visual, geographic, interaction-oriented constructs for programming hybrid human-automation systems. We conclude with design principles that might be applicable to similar work.

DESIGN PROCESS

We followed a research-through-design methodology, in "an attempt to make the right thing: a product that transforms the world from its current state to a preferred state" [48]. We worked with controllers and operational staff from Malta Airport to iteratively define the problem and the solution by designing scenarios and interactions.

Interviews, observations and work analysis workshop

We combined interviews, observation, a work analysis workshop with ATCOs and operational experts to capture and analyze real-world scenarios where automation could be useful. We met eight controllers, an ATC expert and the head of tower operations, a former ATCO with experience of all control positions. We collected documents describing Airport procedures such as the Operation Manual, and produced eight work scenarios describing nominal and non-nominal situations precisely. Participants were asked to indicate the ATCOs' intentions and challenges for each step. Later, we transcribed the scenarios into a more detailed graphical and written format (Figure 2). The outcome was refined and validated by the operational expert and the head of tower operation. This process ensured that the correct requirements were captured and that the work was grounded on the real activity.

Participatory-design workshop

We then conducted a participatory design workshop to explore and refine the DSGL concepts. We worked with two other ATCOs, together with the operational expert and the head of tower operations who formerly validated the work scenarios. We presented our design concept with an early prototype and a design scenario. After gathering feedback, we encouraged participants to identify scenarios from their previous experiences in which they could program automation using such concepts. We then used paper prototyping techniques to produce four new design scenarios involving more complex situations and/or end-user-defined automatisms. The session lasted approximately two hours.

OPERATIONAL CONTEXT AND REQUIREMENTS

The airport has two perpendicular runways, several taxiways to access the runways as well as the aprons and named points used to respectively park and guide aircraft (Figure 2). The longest runway usually accommodates commercial traffic, while the shorter and secondary runway is used for the other traffic (e.g. training flights) and engine tests. Both runways might be crossed by search-and-rescue helicopters.

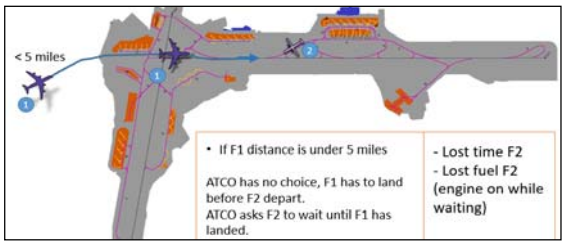

Figure 2: Map of the airport and excerpt of a work scenario consolidated by the ATCOs. The two runways form a gun-like shape, taxiways are in purple, and aprons in orange.

Three ATCOs with distinct roles and tasks operate simultaneously in the tower from adjacent desks. The tower controller is responsible for operations on the runway. The ground controller is responsible for aircraft operations on the maneuvering area. The tower coordinator clears persons and vehicles on the maneuvering area.

Controlling with physical and digital tools

In addition to an almost complete view of the airport maneuvering area, all ATCOs use several physical and

digital tools to perform their tasks. For instance, the tower controller setup is comprised of three screens, a radio and a physical map (Figure 3). The screens allow the monitoring of flights on a radar-view (a), the controlling of runway occupancy indicators and ground lights (b), and the retrieval of meteorological data (c). The radar view helps the tower controller monitor the path of approaching and departing aircraft as well as the planned landing sequence. ATCOs use the physical map to represent the states of the runways and the taxiways as well as to track aircraft positions on the ground. In this figure, a "runway occupied" token is displayed on the physical map (d) because the secondary runway has been closed. Another token reminds that the "tango" taxiway is closed. The yellow strip (e) represents an approaching aircraft.

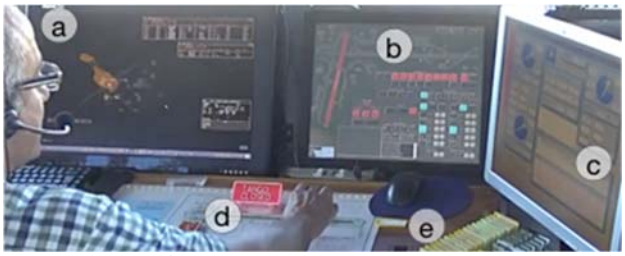

Figure 3: ATCO desk comprising a radar view (a), a map of the airport (b), meteorological data (c), a physical map with tokens indicating closed parts (d) and strips for aircrafts (e).

An already semi-automated, safety-enforced activity

ATCOs already rely on several forms of automation such as alarms detecting collision risks between flights, traffic lights or stop bars. The traffic lights are placed at road and runway intersections to allow or prevent vehicle access to the runways. They are manually set by the ATCOs and automatically raise an alarm if the set command fails. Stop bars are red lights located at the runway entrance to prevent aircraft access to the runways. To switch them off, the tower controller first clicks on the stop bar icon on the center screen. This opens a confirmation window. He then asks the coordination controller to ensure that there is no objection. If the latter agrees, he clicks the confirmation button, which switches off the stop bar. He then gives a voice clearance to the pilot. The stop bar is automatically reactivated after 90s, a delay that has been established by ATCOs and the equipment manufacturer to suit their practice.

Challenges for automations: Airport complexity factors

ATCOs and the operational experts explained that despite its medium/small size, Malta airport presents several sources of complexity mostly due to the low number of taxiways and traffic heterogeneity. The lack of taxiways available for aircraft to reach the main runway extremities from the main parking area forces ATCOs to use the runway as a taxiway ("to backtrack"). ATCOs must elaborate several strategies such as having two aircraft backtracking one behind the other or clearing a departing aircraft to backtrack when another landing aircraft has reached a specific zone of the runway.

The airport accommodates a wide range of aircraft and ground vehicles that must share some of the taxiways or restrict their parallel use. For instance, the two crossing runways prevent aircrafts from landing or departing simultaneously. Besides, depending on aircraft weight and span categories, some of the holding points around the runways cannot be used and must be vacated immediately. Thus, departing and landing automations on a runway must also consider the state of the other runway.

Requirements

Based on the user studies and on our motivation to explore visual programming, we devised a set of requirements for authoring and operating automations support. We consider *Authoring* as the programming ahead-of-time, i.e. before performing ATC, of the automations. Authoring includes creating, editing but also testing. We consider *Operating* as the monitoring and adjustments of automations just-in-time, i.e. while the automations are running. Operating includes handling real traffic data by automation, human monitoring of automations, and handover when humans want to take control over the automation.

Consider the existing scenarios (Existing)

The new technologies must support the current work practices as identified during the user studies.

Define language constructs relevant to ATC (Relevant)

ATC includes activities such as planning, monitoring, reacting to contingencies. All these activities require that humans and machines interact. Hence, the new technologies must support interaction-oriented programming: events, dataflow, state-machines, states.

Make language constructs usable by ATCOs (Usable)

An "expertise tension" exists in a two-dimensional continuum of job-related domain knowledge and system-related development knowledge [2]. ATCOs might lack systems knowledge. Thus, domain experts should be enabled to modify or extend software without having a deep understanding of a computer system or coding skills.

Foster predictability and handover (Predictable)

Beyond notification, which often occurs too late, another challenge is to offer better representations of the status of automation and its ability to handle the current traffic. This should make the system predictable and allow ATCOs to decide if they must take control over the automation.

Foster efficient and scalable authoring (Efficient)

Since the number of automations might be high, the technology should scale up i.e. should be still usable with multiple automations.

EXPLORING AND ASSESSING THE DSGL CONCEPTS

During the participatory design workshop, participants were very enthusiastic about the use of the airport map to program automations since they could instantly understand the visual programs. When we presented the design scenario, they expressed concerns about there being no human in the loop when the departing flight was granted access to the runway. In reaction to this specific automation, an ATCO stated that "*the runway is more important than my house!*", to

emphasize the importance of being effectively in control of the runway and not only supervising it. However, participants quickly understood the potential value of the concept and we explored ways for the ATCOs to either manually prevent the flight from automatically entering the runway (thus taking control back), or a means to simply notify the ATCO that the runway is available.

Assessing and updating the runway occupancy

Together, we explored a scenario to help ATCOs assess the runway occupancy status. The participants explained that the runway status is manually updated by the ATCOs which can be error-prone. The head of tower operations argued that: *"sometimes you can forget to turn it on in the system because you are busy and a vehicle wants to go on to the runway for a five-minute job"*. Participants suggested automation could provide a visual reminder of the permanent runway status by adding an area to detect aircraft and other vehicles above the runway. If the detected state differs from the system state, an alarm will be displayed to help the user update it.

To prevent possible detection failures, we suggested the use of a digital puck like the physical one they use on the current map to mimic the presence of any vehicles. The ATCOs found the idea convincing and explained this could also be very useful to indicate maintenance operations on the runway or that a runway is closed for safety reasons such as the presence of ice or traffic light maintenance.

Programming safety nets

Building upon the previous scenario, participants expressed positive feedback about the opportunity to program automations themselves that would act as safety nets, *i.e.* security measures, that they could fine test and tune. We built a design scenario with a safety net that triggers an alarm if there is an aircraft on the runway and another one approaching. To author this automation, participants suggested that they would use a zone covering an area corresponding approximately to 5min before landing and place an alarm box wired to both the runway and this zone.

Interacting with stop bars

Since participants mentioned challenges to automate the stop-bars, we explored possibilities to define their behaviors with our concepts. Participants first suggested that they could turn off the stop-bars by clicking or touching their visual representations on the map as presented in Figure 4 (left). They argued for a confirmation step involving another click to mimic existing interactions. Figure 4 (center) illustrates how we prototyped it by adding an ATCO icon (the confirmation components of the DSGL) over the stop-bar to make the user confirmation step explicit. We discussed the opportunity to make the remaining time visible as a progress bar or text superimposed over the graphical representation. This idea received positive feedback from the participants.

Figure 4: Prototypes for automating stop-bars. Left: turning off a stop-bar by tapping its representation. Center: representing the ATCO to ensure a confirmation. Right: dropping an area to join two aircraft.

VIZIR

Based on the results of the workshops, we designed Vizir, a DSGL and an environment for the visual authoring and operating of automations (Figure 1). This section incrementally introduces its features by describing several interactions based on the work scenarios (*Existing*).

The geographic airport map as the background canvas

ATCOs rely extensively on both the airport map and the radar view to accomplish their tasks. Such visual, two dimensional representations of the space they must monitor, offers clear landmarks that support reasoning. To support their tasks, automations must be visible and correlated with the airport map. Their representations should facilitate the interactive control that may be required in sensitive situations (*Usable* and *Predictable*).

At the start of the application, the map of the airport is displayed. The map is a structured SVG file, containing multiple layers of graphics implemented with SVG groups: runways, taxiways, parking, buildings. Users can freely pan and zoom the airport or hide and reveal specific layers. The background and the features of the airport are mostly dark blue. Dark images are often used in ATC as they place less stress on the eyes than luminous ones.

Authoring simple geographic-based automations

The user can add programming constructs on top of the airport to specify automations. These elements are luminous and yellow to differentiate them from the background.

Drag'n'dropping components from the toolbox

A toolbox at the bottom displays a set of components: enter/leave zone, geographic-time scale, confirmation, alarm, gauge, "occupied" and "closed" pucks, fake flight, text-to-speech, logic and arithmetic operations and properties (Figure 5). The user can drag and drop them from the toolbox to the map to create instances (*Usable*).

Figure 5: the component toolbox

Defining an Enter/Leave zone (ELZ)

The user can drag and drop a rectangular Enter/Leave zone that is activated (and colored purple) when a flight enters or leaves it. The user can move it around freely, superimpose it on a specific area of the airport, resize it and reorient it to

match the underlying area (Figure 6). The user can thus specify an area that would not match the structure of the airport, for example an area that encompasses a taxiway and a parking (*Relevant*). We will also make it possible to use airport graphics as Enter/Leave zone components (not implemented in the current prototype) (*Usable*).

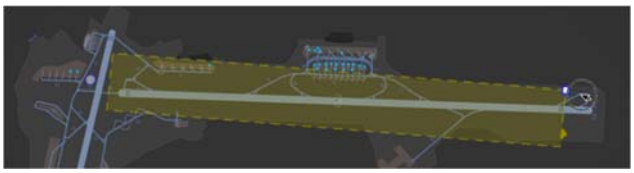

Figure 6: resizing an enter/leave zone to match runway

Connecting elements

Each component embeds a set of input and output plugs (small triangles), resembling boxes in Max and similar environments. The user can press a plug to create a wire, drag its end and drop it on the plug of another component to connect them. For example, after placing an Enter/Leave zone and a warning box, the user can connect the output of the Enter/Leave zone to the input of the warning box to specify that the Warning will turn to 'on' (e.g. red) when a flight enters the Enter/Leave zone. In addition, a light rectangle behind the triangle appears when the plug is activated and informs users of the plug activation.

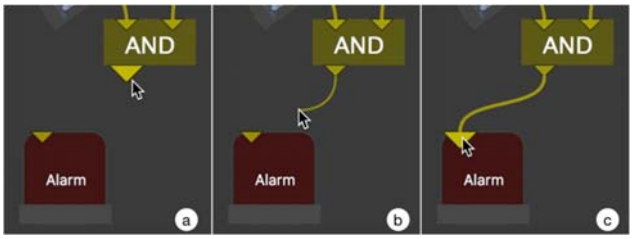

Figure 7: connecting an 'and' gate with an alarm

Authoring more complex automations

Specifying data-flow behaviors

To specify data-flow-based automations, the user can connect logical (and, or, …) and arithmetical (+, -, x …) operators between input and output plugs that emit or receive a flow of scalars. For example, users can create an Enter/Leave zone located at a position corresponding to 500m before the runway. They can then create an 'and' gate, then connect the zone to one input of the 'and' and the zone over the runway to the other input, and the result of the 'and' box to a warning box (Figure 7). This would create a warning for the ATCO that would flash if a flight is about to land while another flight is still on the runway.

Maintaining state

Users can create Boolean, integer or double properties, and connect them to other components. Properties serve as intermediate variables in dataflows and to implement states. Indeed, connectors can be of two types: data-flow or assignment (shift key pressed when connecting). While a dataflow continuously pushes data when activated, an assignment pushes only one value when activated. Connecting a component to a property with such a link specifies that the property will be set to a value when the component is activated.

Switching state

The choice of reactions of an interactive system often relies on its state. Vizir provides ways to access the inner states and transitions of the state-machine controlling a component. Figure 8 illustrates a scenario in which the user connects the transitions of two state-machines. When a transition is fired in the first state-machine, the connected one is also fired in the other state-machine. Conversely, the states of different state-machines can also be coupled. However, Vizir does not offer ways to entirely and graphically specify a state-machine, it only allows its states and transitions to be accessed. The state-machine must be written with a textual language.

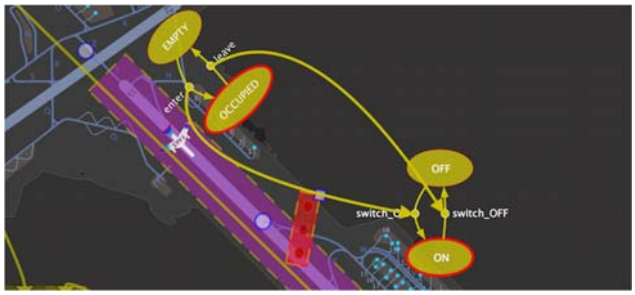

Figure 8: coupling state-machines: here, runway and stop bar

Explicitly transferring automation control to human control

To ensure that an event cannot be transmitted without explicit ATCO validation (*Predictable*), any connection can be overloaded with a confirmation decorator by dragging a 'Hand' onto the wire. When an event is sent from the source, the Hand is replaced by a glowing confirm-abort box: the user must click on it to confirm or abort the signal flow to destination (Figure 9). For example, the user can create an automation that would switch off a stop bar whenever the runway is free because the flight that landed left it for a taxiway. Since this action is critical, she can put a 'hand' on the wire between the runway zone and the stop bar to request confirmation from the operating user.

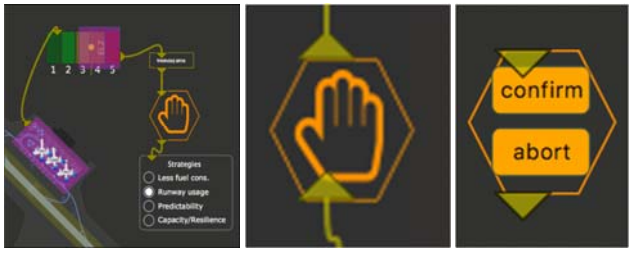

Figure 9: explicit human confirmation component

Explicitly transferring human control to automation control

Some aircraft are equipped with digital communication means to exchange mail-type messages with ATCOs, but many light or old aircraft are not. In this case, the main communication channel between ATCOs and pilots is radio. Some ATCO messages are routine, such as 'transfer

messages' in which ATCOs instruct departed flights to change their frequency to the next control sector. Other ones may be very urgent, such as a 'go around' message to a flight that must cancel its landing because the runway is unexpectedly occupied. Vizir provides programmers with a text-to-speech component that synthesizes speech over the radio when activated, for such routine or urgent message (Figure 10). This feature also shows how automation can leverage the use of older technologies, providing smooth system evolution (*Existing*).

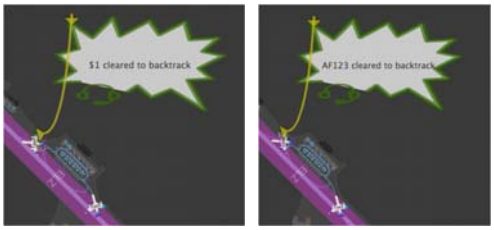

Figure 10: text-to-speech automation

Compressing distance with geographic time-scales

Coping with different orders of magnitude of distance between objects is an inherent problem with geographic maps. When zoomed to fit the entire airport, the view represents roughly a 5x4 km area. However, landing flights that are important for ATCOs may be as far as 100km away from the airport and cannot be represented without zooming out, leading to a tiny airport representation.

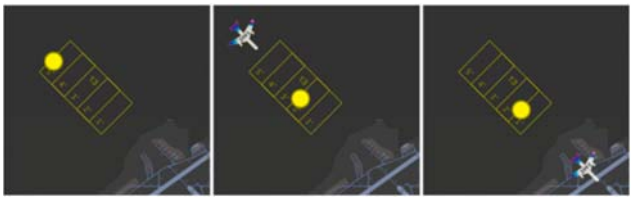

Figure 11: geographic time-scale

To mitigate this problem, we designed a geographic time-scale. A classical time-scale reflects the predicted delay before a flight lands: it projects the delay to a position in the frame of reference of the time-scale. A geographic-time scale is a time-scale substrate [18,36] whose position and orientation on the screen are meaningful with respect to the underlying geographic map. Figure 13 shows an instance of a geographic time-scale: the "entrance" of the time-scale corresponds to where the flights come from, while the "exit" of the time-scale corresponds to the "entrance" of the runway. The flight is depicted with a yellow circle. Such a representation 'compresses' the position of distant flights. Its position and orientation lend itself to smooth transition of flights movement from time to space (*Usable*). It works in this case because flights are placed in a queue before landing.

Defining meta-interaction between graphical components

The use of graphics to specify behavior can be extended from geographic-based to screen-based. For example, we have designed a gauge to reflect the 'level of traffic'. The gauge is connected to a zone comprising the taxiways and the runways and counts the number of flights in this zone. Since the value of the gauge is displayed with a yellow dot, the fact that the circle enters another zone can be used as a meta-event that triggers another behavior. As presented in Figure 9, we have connected it to a set of radio-buttons that reflect the current strategy of an external algorithm that computes a timed sequence of departures. When the level of traffic is high, the strategy is changed from 'minimize fuel consumption' to 'maximize runway usage' to make flights depart as soon as possible.

Figure 12: the gauge value is detected by an enter/leave zone

Another example is the geographic time-scale. Users can create Enter/Leave zones that detect flights inside the time-scale to trigger behavior. Figure 13 illustrates an Enter/leave zone in the range 5-3 min before landing. e.g. when flights 'enter' a 3min-delay zone. Combined with the presence of a flight on the runway, such automation is capable of triggering a warning, or even a go-around by radio if connected to the text-to-speech component.

Figure 13: meta-automation on geographic time-scale

Supporting tests

Testing is a mandatory activity of automation development (*Efficient*). Fake flights are components that the user can move around (Figure 14). Like flights, a fake flight is sensed by Enter/Leave zones. This enables the user to test parts of the automation that is under construction: she can place one or multiple flights in multiple zones and observe how the automation behaves with respect to these conditions. One can simulate a traffic with a sequence of interactions that correspond to target situations to be tested.

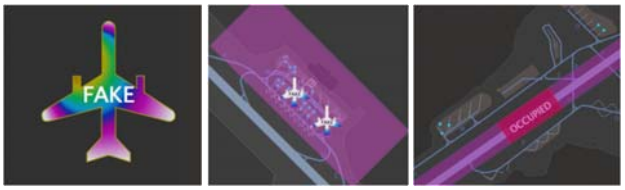

Figure 14: fake flight and occupied puck

The user can also directly propagate the activation of a component by clicking on a plug (e.g. yellow triangles in Figure 7). If the plug is an input, the component it belongs to is activated and subsequent behavior is triggered. If the plug is an output, the component connected to this plug is activated.

Managing scalability

Visual languages might suffer from scalability. Even with some simple automations, the screen might be filled up with many graphical components and crossing wires, making automation difficult to understand and edit.

We relied on progressive disclosure [24] to address this issue. Vizir distributes automations in file units organized in a folder hierarchy, as in traditional programming languages. The editor displays this hierarchy as a collapsible tree to the left of the main document, as in traditional IDE (Figure 1). The user can double-click on a file to display its content in the main document within a tab. A double-click on another file in the tree opens a new tab and displays its content on top of the radar image in lieu of the previous tab. The originality of our approach is the common use of the map in the background of all tabs, and the disclosure or the concealment of the program on top of this background. To the best of our knowledge, no other visual editor provides such a functionality.

Another possibility is to display the content of two or more tabs simultaneously, by shift-clicking on a tab, effectively turning them into layers. The content of the new tab is added to the content of the current visible tab. A slider enables control of the opacity of the components that belong to the tab. This mechanism mitigates the problem of possible overlap between components. The ability to make the content of multiple files visible and accessible to the programmer enables the user to connect the plug of components that reside in different files. This fosters modularity and visibility.

Complex components can be written in the Smala textual language [35] and directly reused in Vizir. The component appears as a box with plugs that correspond to declared properties in the code. The user can double-click on the box to open its content in a new tab. The user can control the tab opacity to see the geographical map when coding with text.

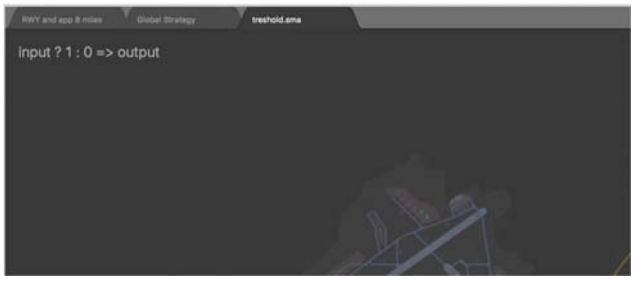

Figure 15: translucent tab to edit textual code

Operating automations

Controlling program visibility

During operation, many of the automation components are invisible to the tower ATCO. Only the visual components such as alarms, stop bars, hand and runway EL zones are visible since they are required for proper first-level interaction. However, the ATCO can progressively disclose automation components by: using a slider to control their opacity; invoking a magic lens [3] that she can move around to reveal local automation; or by clicking on a component and revealing/hiding each level of connections (considered here as a graph) using the mouse wheel.

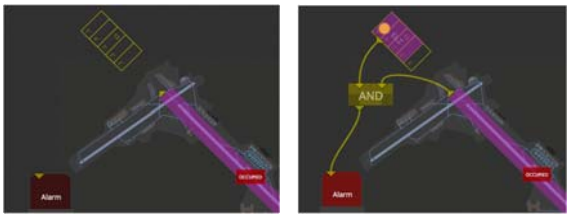

Figure 16: when the alarm is triggered, its program appears

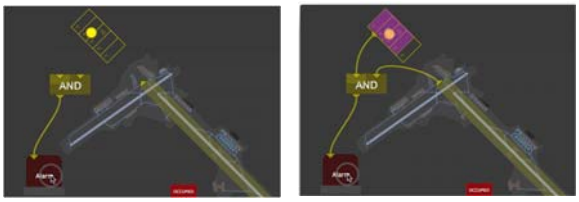

Figure 17: press+mouse wheel to gradually reveal the program

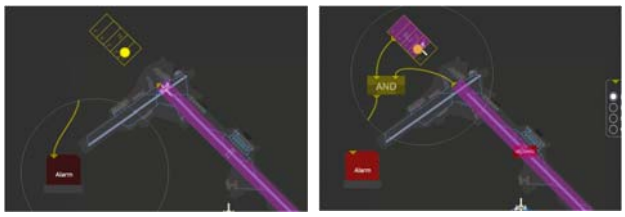

Figure 18: magic lens to inspect local automations

Triggering/Disabling automation

Some of the testing support in author mode is accessible during operation: for example, clicking a plug will activate the depending component. If an automation component is visible, the status of its plug is visible too. The user also has the possibility of temporarily disabling a coupling by simply moving the end of a connection and releasing it on the background. Instead of completely deleting it, disconnecting allows the user to retrieve a disconnected coupling and enable it again by moving it into the plug.

The ‘occupied’ and ‘closed’ pucks (Figure 14) enable the ATCO to inform the system that an Enter/leave zone is occupied as if a flight is inside, while triggering any behavior that depends on such a flight occurrence. It offers a pliant way to adapt the operation of the automation e.g. in case of an unpredicted contingency (e.g. fire) without being explicit about the causes.

Assessing future behavior of automation

To reveal the automations that will be triggered in the near future, the user can also manipulate a time slider: the flight trajectories will be extrapolated and simulate the triggering of automations as they are expected to behave. Similarly, the user can trigger a DIMP-like interaction [14] by pressing on a flight. This interaction displays the predicted trajectory and dragging the flight along the trajectory allows the user to visualize the updated positions of other flights and the

corresponding triggered automations. This enables a user to assess the behavior of automation at a critical place at the airport e.g. a crossing. The current implementation serves as a demonstration only and is not truthful, as a proper implementation requires the saving the successive states of the application to be able to backtrack in time and retrieve previous states.

Figure 19: Assessing automation behavior in the near future: pressing on the top-left flight displays its predicted route (in orange), moving the flight along its route changes the time of the whole simulation.

Implementation

We used the smala language and the djnn framework [35] to implement Vizir on top of the airport radar image. Smala and djnn unify many concepts related to interaction-oriented programming. Vizir representations of djnn unified concepts enable programmers to seamlessly combine different interaction-oriented programming constructs in a visual manner.

Since Vizir is a research project, it is not complete and safe enough to be operational. We thus simulated operations by using realistic traffic data. Simulated traffic uses a real-time player of recorded trajectories, which broadcasts them through the Ivy software bus [5]. In actual operation, the system would be fed with aircraft and vehicle locations, which is generally feasible via GPS, a ground radar or with direct communication.

DESIGN PRINCIPLES AND DISCUSSION

While designing Vizir, we identified and refined a number of design principles. The principles helped us fulfill the requirements and answer the research question. They also guided us during ideation phases. We present the guidelines as explanations based on a post-hoc analysis of Vizir's features. Their titles have been generalized from the specific activity to make them useful to designers of similar systems.

Continuum of usage and progressive disclosure

We envisage that the tool be used during a continuum of phases: ahead-of-time when authoring the map, programming rules/constraints from the operational manual of the airport and when programming local automations according to the places of the airport; or just-in-time while controlling by anticipating when quickly programming clearances in advance, reacting to behavior triggered by automation, taking control back, or simply supervising. Progressive disclosure [24] is one of the means to implement this continuum.

Space-based and event-based constructs

Since ATCOs monitor moving aircraft and vehicles on ground, most of their activity is based on the geography of the airport. However, some of their activity is time-based because time can be easily measurable. Considering that space was reliably measurable (for example using GPS), we explored how to transform time-based rules and their automation into space-based rules. Examples of features that support this principle include enter/leave zones, geographical-time scale and 'meta'-interactions between graphical components.

Make current state and future behavior visible

Visibility is one the most important properties to respect in GUIs [39] (*Usable*). We thus strived to provide 'programmers' and 'users' with means to respectively display and see all elements useful for the understanding and prediction of the behavior of the system. Examples of features to support this principle include alarms, gauges, user-triggered disclosure of DSGL constructs, visibility of plug status and means to navigate in the future. Gauges can be used to build low-visibility warnings that emit harbinger cues and inform the users of the 'load' of the automation so that they can predict their own load in an optional handover.

Seamless and seamful hybrid control

In such hybrid human-automation systems, an important property resides in the seamless interactions between humans and automations (*Predictable*). Humans should be able to easily control automation and recover all authority. Similarly, automation might be able to inform users, and request users' actions. 'Seamless' in this case means that programming constructs be designed to be indifferently performed by humans and automation. Examples of features include human-triggered and automation-triggered plugs, text-to-speech, gauges connected to strategy change. On the other hand, seamless control can also lead to dangerous behaviors such as unintended consequences of accidental actions triggered by humans or unexpected high-impact decisions made by automation. Vizir provides seamful [21] control switch between humans and automation in critical situations. Examples of such features include explicit confirmation or "occupied" puck.

Foster interaction-oriented programming

The programming model to be used should make the geographic map, spaces, events, states, behaviors and graphics first-class citizens. In other words, Vizir had to be based on an interaction-oriented programming model. We used the djnn conceptual model [39], which seamlessly combines events, reactions, activations, data-flows, state machines, graphics and states to implement interactions.

RELATED WORK

Visual authoring and operating of automation are related to End-User Programming, Graphical languages and environments, and specific authoring environments.

End-user programming and related concepts

End-user programming (EUP) is defined as "programming to achieve the result of a program primarily for personal, rather than public use." [28]. The benefits of having an end-user program lie in coupling deep knowledge of the involved activities with programming to build a system that supports such activities. A domain such as ATC is "likely to involve different types of computational patterns and different software architectures" [28]. This is what we wanted to explore: to which extent the correct conceptual model and user interface may enable stakeholders, ATCOs and Engineers to design better automated systems?

Several notions related to End-user programming (EUP) may prove important for Vizir usability. End-User Development (EUD) is a set of methods, techniques, and tools that allow users of software systems, who are acting as non-professional software developers, at some point to create, modify or extend a software artefact [34]. End-User Software Engineering (EUSE) is concerned with technologies that collaborate with end-users to improve software quality [28]. EUSE deals with other phases of the software life-cycle than programming. Programming By Example is an easier way to specify behavior for end-users [11,12,31,49]. Live Programming makes programming easier by re-executing a program continuously during editing [6,19,46]. Domain-specific languages (DSLs) are tailored to a specific application domain and offer "substantial gains in expressiveness and ease of use compared with general-purpose programming languages in their domain of application" [37].

Graphical languages and environments

All programming activities require languages and environments to specify, produce and run programs. Among them, graphical languages facilitate EUP.

Learning to program

Some IDEs are specialized in facilitating the learning phase of programming. Much of them are oriented towards children. For example Little Wizard [51] uses icons for variables, expressions, loops, conditions and logical blocks that children can drag and drop together to compose program "sentences". Squeak Etoys [52] and Scratch [53] are authoring environments to create computer programs using a scripting language based on blocks and containers. Kodu [54] is a visual programming tool to teach reactive programming to children with the "trigger – action" programming concept. Kodu builds on ideas begun with Logo [1] and other projects such as AgentSheets [55], Squeak [52] and Alice [56]. Learnable programming is a set of principles about the vocabulary, the visualization of flows and states, and on supporting coders in creating behavior by reacting to immediate feedback and abstracting [4].

Professional environments

LabVIEW [57] is a system-design platform and development environment using the G graphical language. Execution is determined by the structure of a graphical block diagram on which the programmer connects different nodes by drawing wires. VAPS XT [47] and SCADE Display [50] are products that allow users to define both object appearance and display logic in a single graphical editor for embedded systems. Most of the editing is performed in a graphical form. IFTTT [58] ("If This Then That") is an implementation of Trigger-Action for web and IoT.

Interaction-oriented languages and tools

Interaction-oriented programming [32] has requirements that necessitate support from languages, frameworks and tools [38]. For example, Fabrik is a direct manipulation user interface builder that enables a designer to specify transforms between linked widgets with a visual flow language [23]. Whizz'ed is a graphical editor for a data-flow-based programming language [17]. Whizz'ed enables programmers to draw links between blocks and is capable of triggering actions based on enter/leave events [17]. Icon is a tool to graphically specify and control data-flows of events from input to display [13]. Interstate is a tool to graphically develop state-machines that describe interactions [40]. Musical programming environments such as Max and Pure Data [43] are based on data-flow. In such languages, functions or "objects" are linked or "patched" together in a graphical environment that models the flow of the control and audio. Kitty is a tool to sketch interactive illustrations [26]. Kitty notably relies on space and time, and a relational graph that implements the causal relationships between objects in a scene. Vizir seamlessly combines several features from these works.

Specific authoring and operating environment

Several environments have been created to help develop games with high-level programming concepts that are closer to game designers' concerns. In particular, several tools such as Unreal Engine [59], Unity [60] or Euphoria [61] rely on visual programming techniques to facilitate the definition of dynamic behaviors, rules and motions. Real Time Strategy (RTS) games enable players to control multiple instances of semi-autonomous agents in a simulated world. For example, agents can go to a location, collect some elements and bring them back. Animations can also be considered automation, and several applications support their design, including those involving multiple objects [27].

A number of tools and techniques assist musical composers in programming and executing temporal plans for real-time performance. Such systems frequently combine music-specific representations such as musical notation with additional data such as control curves. The automations can also be used during live performance and modified in real-time ("operating" in Vizir). In automatic score-following tools such as Antescofo [7], a real-time listening machine reacts to musical events described in a score and interpreted by a human performer to launch predefined actions. Cont et al. propose to defer error handling strategies to the user [8]. They define two strategies, tight and loose, that respectively correspond to whether the system will necessarily trigger all actions or avoid some parts if events are missed. Vizir also relies on the user to define automation that must be validated.

However, Vizir does not yet offer such sophisticated strategies to define how to recover from user errors such as forgetting to switch off the stop bars before clearing an aircraft to enter the runway.

Automation in ATC

Automation in ATC aims at substantially reducing controller task-load per flight through automated decision support, whilst simultaneously meeting the established safety and environmental goals. However, due to safety concerns and careful integration of technologies, ATC still relies heavily on human operations [22,33]. The usability of the whole system is also important since safety and capacity heavily depend on mutual awareness [15,25] as well as fast and efficient interactions [10,33]. Still, the will to improve safety and capacity requires ATC systems to evolve and reach new levels of performance, notably with movement guidance and control systems (A-SMGCS) [30], for which automations can be designed with Vizir.

THREATS TO VALIDITY AND LIMITATIONS

The interactions that we designed might be applicable only to the specific airport we studied, due to its particularities and its low complexity. The rationale of Vizir is that airport automation will be programmed with similar constructs but with specific automation, airports and sources of complexity. We believe that many airports would benefit from location and event-based constructs like the ones introduced in the paper. The stakeholders from other airports to whom we demonstrated the system confirmed that Vizir descriptive power seems compatible with other settings. Adapting the interactions might be unavoidable, as much as adapting code to specific contexts is unavoidable in traditional software engineering.

Nevertheless, automation needs to be specific to be useful. Previous work in the design of short-term conflict alert already in use in ATC shows that algorithms must be heavily tuned to the very specific settings and traffic of the instrumented airspace [62]. Furthermore, airspace rules are continuously evolving globally and locally to make them safer and more effective, thus requiring automation that can be adapted. We believe it is more valuable to provide design tools for end-users as opposed to providing configured-in-advance tools (e.g. A-SMGCS [30]) that might not be useful at all unless they can be adjusted. Hospital studies corroborate that specifically-tailored tools are the most beneficial for patients in hospitals [20].

The interactions might also be applicable to ATC only. However, other domains share similar concerns. For example, home automation might also benefit from the space-based and event-based interactions that we designed. Similarly, designing an interactive museum experience or games might benefit from our work.

The hypothesis that visibility would help ATCOs understand automation must be experimentally tested. Another question that may arise is that of the proficiencies of the targeted users at programming their own automation. If ATCOs are professionally trained to manage traffic, they may be not literate enough in thinking about, specifying or programming software that will be sufficiently usable and safe. However, in the near future, ATCOs like most higher-grade professions will be educated in computer science during their studies. We envision they will participate in automation design in collaboration with qualified computer scientists. Indeed, after the submission of this article, we demonstrated Vizir to two ATCOS who were able to design 3 automations specific to their ATC context in 10 minutes. We never discussed these automations together before. This suggests that our hypothesis that ATCOs will be able to participate to the design of automation is not falsified.

ATCOs may already be too busy with handling the traffic and they might not be able to cope with automation. However, the whole air traffic system is already an automation-based activity from the ATCOs' point of view. ATCOs delegate a number of actions to the pilots and then must monitor whether they are performed. In some ways, pilots can be seen as equivalent to an automatic agent that might succeed or fail to comply with a clearance.

Vizir does not provide visual constructs to support automation that requires coordination with controllers that are not located in the same room. This is a frequent situation faced by controllers that should be investigated in future work. Nevertheless, if confronted to the incompleteness of the visual language, programmers can resort to the underlying textual language and can even embed algorithms coded in C/C++.

CONCLUSION

We have presented Vizir, a Domain-Specific Graphical Language and Environment to support end-user authoring and operating of airport automation. Vizir combines visual, geographic, interaction-oriented constructs to program hybrid human-automation systems. Vizir provides means to mitigate the complexity induced when scaling up the number of automations. We grounded our work by studying ATC on Malta airport, and the features were refined through a participatory design process with ATCOs.

Future work will focus on additional experimentation with ATCOs to assess how they can author and operate automation with Vizir. We will also further explore the DSGL and Vizir to adapt them to more complex airports and other domains than ATC such as home automation.

ACKNOWLEDGEMENTS

We are grateful to experts and controllers from Malta Airport that participated in our studies. This project has received funding from the SESAR Joint Undertaking under grant agreement No 699382 under European Union's Horizon 2020 research and innovation programme.

REFERENCES

1. Hal Abelson, Nat Goodman, and Lee Rudolph. 1974. LOGO Manual. Retrieved December 7, 2016 from http://dspace.mit.edu/handle/1721.1/6226

2. Joerg Beringer. 2004. Reducing expertise tension. *Communications of the ACM* 47, 9: 39. https://doi.org/10.1145/1015864.1015885

3. Eric A. Bier, Maureen C. Stone, Ken Fishkin, William Buxton, and Thomas Baudel. 1994. A Taxonomy of See-through Tools. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '94), 358–364. https://doi.org/10.1145/191666.191786

4. Victor Bret. Bret Victor, Learnable programming. *worrydream*. Retrieved July 19, 2016 from http://worrydream.com/#!/LearnableProgramming

5. Marcellin Buisson, Alexandre Bustico, Stéphane Chatty, Francois-Régis Colin, Yannick Jestin, Sébastien Maury, Christophe Mertz, and Philippe Truillet. 2002. Ivy: Un Bus Logiciel Au Service Du DéVeloppement De Prototypes De SystèMes Interactifs. In *Proceedings of the 14th Conference on L'Interaction Homme-Machine* (IHM '02), 223–226. https://doi.org/10.1145/777005.777040

6. Sebastian Burckhardt, Manuel Fahndrich, Peli de Halleux, Sean McDirmid, Michal Moskal, Nikolai Tillmann, and Jun Kato. 2013. It's Alive! Continuous Feedback in UI Programming. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation* (PLDI '13), 95–104. https://doi.org/10.1145/2491956.2462170

7. Arshia Cont. 2008. ANTESCOFO: Anticipatory Synchronization and Control of Interactive Parameters in Computer Music. In *International Computer Music Conference (ICMC)*, 33–40. Retrieved December 14, 2016 from https://hal.archives-ouvertes.fr/hal-00694803/

8. Arshia Cont, José Echeveste, Jean-Louis Giavitto, and Florent Jacquemard. 2012. Correct Automatic Accompaniment Despite Machine Listening or Human Errors in Antescofo. Retrieved November 10, 2016 from https://hal.inria.fr/hal-00718854/document

9. Stéphane Conversy, Stéphane Chatty, Hélène Gaspard-Boulinc, and Jean-Luc Vinot. 2016. The Accident of Flight 447 Rio-Paris: A Case Study for HCI Research. In *Proceedings of the International Conference on Human-Computer Interaction in Aerospace* (HCI-Aero '16), 1:1–1:8. https://doi.org/10.1145/2950112.2964586

10. Stéphane Conversy, Hélène Gaspard-Boulinc, Stéphane Chatty, Stéphane Valès, Carole Dupré, and Claire Ollagnon. 2011. Supporting air traffic control collaboration with a TableTop system. In *Proceedings of the ACM 2011 conference on Computer supported cooperative work*, 425–434.

11. Allen Cypher. 1991. EAGER: Programming Repetitive Tasks by Example. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '91), 33–39. https://doi.org/10.1145/108844.108850

12. Allen Cypher, Daniel C. Halbert, David Kurlander, Henry Lieberman, David Maulsby, Brad A. Myers, and Alan Turransky (eds.). 1993. *Watch What I Do: Programming by Demonstration*. MIT Press, Cambridge, MA, USA.

13. Pierre Dragicevic and Jean-Daniel Fekete. 2001. Input Device Selection and Interaction Configuration with ICON. In *People and Computers XV—Interaction without Frontiers*. Springer, London, 543–558. https://doi.org/10.1007/978-1-4471-0353-0_34

14. Pierre Dragicevic, Gonzalo Ramos, Jacobo Bibliowitcz, Derek Nowrouzezahrai, Ravin Balakrishnan, and Karan Singh. 2008. Video Browsing by Direct Manipulation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '08), 237–246. https://doi.org/10.1145/1357054.1357096

15. Mica R. Endsley. 2011. *Designing for Situation Awareness: An Approach to User-Centered Design, Second Edition*. CRC Press, Inc., Boca Raton, FL, USA.

16. Heinz Erzberger. 2004. (4) Transforming the NAS: The next generation air traffic control system. *NASA*, TP–2004-212828. Retrieved September 25, 2017 from https://www.researchgate.net/publication/250066237_Transforming_the_NAS_The_next_generation_air_traffic_control_system

17. Olivier Esteban, Stéphane Chatty, and Philippe Palanque. 1995. Whizz'ed: A Visual Environment for Building Highly Interactive Software. In *Human—Computer Interaction*. Springer, Boston, MA, 121–126. https://doi.org/10.1007/978-1-5041-2896-4_20

18. Jérémie Garcia, Theophanis Tsandilas, Carlos Agon, and Wendy Mackay. 2012. Interactive Paper Substrates to Support Musical Creation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '12), 1825–1828. https://doi.org/10.1145/2207676.2208316

19. Christopher Michael Hancock. 2003. Real-time programming and the big ideas of computational literacy. Massachusetts Institute of Technology. Retrieved December 18, 2016 from http://dspace.mit.edu/handle/1721.1/61549

20. David U. Himmelstein, Adam Wright, and Steffie Woolhandler. 2010. Hospital Computing and the Costs and Quality of Care: A National Study. *The American Journal of Medicine* 123, 1: 40–46. https://doi.org/10.1016/j.amjmed.2009.09.004

21. Kristina Höök and Jonas Löwgren. 2012. Strong Concepts: Intermediate-level Knowledge in Interaction Design Research. *ACM Trans. Comput.-Hum. Interact.* 19, 3: 23:1–23:18. https://doi.org/10.1145/2362364.2362371

22. V. David Hopkin. 1991. The Impact of Automation on Air Traffic Control Systems. In *Automation and Systems Issues in Air Traffic Control*. Springer, Berlin, Heidelberg, 3–19. https://doi.org/10.1007/978-3-642-76556-8_1

23. Dan Ingalls, Scott Wallace, Yu-Ying Chow, Frank Ludolph, and Ken Doyle. 1988. Fabrik: A Visual

Programming Environment. In *Conference Proceedings on Object-oriented Programming Systems, Languages and Applications* (OOPSLA '88), 176–190. https://doi.org/10.1145/62083.62100

24. J. Johnson, T. L. Roberts, W. Verplank, D. C. Smith, C. H. Irby, M. Beard, and K. Mackey. 1989. The Xerox Star: a retrospective. *Computer* 22, 9: 11–26. https://doi.org/10.1109/2.35211

25. David B. Kaber and Mica R. Endsley. 2004. The effects of level of automation and adaptive automation on human performance, situation awareness and workload in a dynamic control task. *Theoretical Issues in Ergonomics Science* 5, 2: 113–153. https://doi.org/10.1080/1463922021000054335

26. Rubaiat Habib Kazi, Fanny Chevalier, Tovi Grossman, and George Fitzmaurice. 2014. Kitty: Sketching Dynamic and Interactive Illustrations. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology* (UIST '14), 395–405. https://doi.org/10.1145/2642918.2647375

27. Rubaiat Habib Kazi, Fanny Chevalier, Tovi Grossman, Shengdong Zhao, and George Fitzmaurice. 2014. Draco: Bringing Life to Illustrations with Kinetic Textures. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '14), 351–360. https://doi.org/10.1145/2556288.2556987

28. Andrew J. Ko, Robin Abraham, Laura Beckwith, Alan Blackwell, Margaret Burnett, Martin Erwig, Chris Scaffidi, Joseph Lawrance, Henry Lieberman, Brad Myers, Mary Beth Rosson, Gregg Rothermel, Mary Shaw, and Susan Wiedenbeck. 2011. The State of the Art in End-user Software Engineering. *ACM Comput. Surv.* 43, 3: 21:1–21:44. https://doi.org/10.1145/1922649.1922658

29. P. Ky and B. Miailler. 2006. SESAR: towards the new generation of air traffic management systems in Europe. *Journal of ATC Quaterly*.

30. Roger Lane, Stephane Dubuisson, Mohamed Ellejmi, Michael Huhnold, Bert Klinkers, and Eivan Cerasi. EUROCONTROL Specification for A-SMGCS Services. 123.

31. Gilly Leshed, Eben M. Haber, Tara Matthews, and Tessa Lau. 2008. CoScripter: Automating & Sharing How-to Knowledge in the Enterprise. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '08), 1719–1728. https://doi.org/10.1145/1357054.1357323

32. Catherine Letondal, Stéphane Chatty, Greg Phillips, Fabien André, and Stéphane Conversy. 2010. Usability Requirements of User Interface Tools.

33. Catherine Letondal, Christophe Hurter, Rémi Lesbordes, Jean-Luc Vinot, and Stéphane Conversy. 2013. Flights in my hands: coherence concerns in designing Strip'TIC, a tangible space for air traffic controllers. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2175–2184.

34. Henry Lieberman, Fabio Paternò, Markus Klann, and Volker Wulf. 2006. End-User Development: An Emerging Paradigm. In *End User Development*, Henry Lieberman, Fabio Paternò and Volker Wulf (eds.). Springer Netherlands, 1–8. https://doi.org/10.1007/1-4020-5386-X_1

35. Mathieu Magnaudet, Stéphane Chatty, Stéphane Conversy, Sébastien Leriche, Celia Picard, and Daniel Prun. 2018. Djnn/Smala: A Conceptual Framework and a Language for Interaction-Oriented Programming. *Proc. ACM Hum.-Comput. Interact.* 2, EICS: 12:1–12:27. https://doi.org/10.1145/3229094

36. Nolwenn Maudet, Ghita Jalal, Philip Tchernavskij, Michel Beaudouin-Lafon, and Wendy E. Mackay. 2017. Beyond Grids: Interactive Graphical Substrates to Structure Digital Layout. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems* (CHI '17), 5053–5064. https://doi.org/10.1145/3025453.3025718

37. Marjan Mernik, Jan Heering, and Anthony M. Sloane. 2005. When and How to Develop Domain-specific Languages. *ACM Comput. Surv.* 37, 4: 316–344. https://doi.org/10.1145/1118890.1118892

38. Brad Myers, Scott E. Hudson, and Randy Pausch. 2000. Past, Present, and Future of User Interface Software Tools. *ACM Trans. Comput.-Hum. Interact.* 7, 1: 3–28. https://doi.org/10.1145/344949.344959

39. Donald A. Norman. 2013. *The Design of Everyday Things: Revised and Expanded Edition*. Basic Books.

40. Stephen Oney, Brad Myers, and Joel Brandt. 2014. InterState: A Language and Environment for Expressing Interface Behavior. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology* (UIST '14), 263–272. https://doi.org/10.1145/2642918.2647358

41. Raja Parasuraman and Victor Riley. 1997. Humans and Automation: Use, Misuse, Disuse, Abuse. *Human Factors: The Journal of the Human Factors and Ergonomics Society* 39, 2: 230–253. https://doi.org/10.1518/001872097778543886

42. Raja Parasuraman, Thomas B. Sheridan, and Christopher D. Wickens. 2000. A model for types and levels of human interaction with automation. *IEEE Transactions on systems, man, and cybernetics-Part A: Systems and Humans* 30, 3: 286–297.

43. M Puckette. 1996. Pure Data: another integrated computer music environment. *Proceedings of the Second Intercollege Computer \ldots*. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.41.3903&rep=rep1&type=pdf

44. J. Rasmussen. 1983. Skills, rules, and knowledge; signals, signs, and symbols, and other distinctions in human performance models. *IEEE Transactions on Systems, Man,*

and Cybernetics SMC-13, 3: 257–266. https://doi.org/10.1109/TSMC.1983.6313160

45. Thomas B. Sheridan and William L. Verplank. 1978. *Human and Computer Control of Undersea Teleoperators*.

46. Steven L. Tanimoto. 1990. VIVA: A Visual Language for Image Processing. *J. Vis. Lang. Comput.* 1, 2: 127–139. https://doi.org/10.1016/S1045-926X(05)80012-6

47. TMP. Avionics Display Development Software - VAPS XT™ | Presagis. Retrieved November 25, 2016 from http://www.presagis.com/products_services/products/embedded-graphics/hmi_modeling_and_display_graphics/vaps_xt/

48. John Zimmerman, Jodi Forlizzi, and Shelley Evenson. 2007. Research Through Design As a Method for Interaction Design Research in HCI. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '07), 493–502. https://doi.org/10.1145/1240624.1240704

49. 2001. *Your Wish is My Command: Programming by Example*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

50. 2012. SCADE Display®. *Esterel Technologies*. Retrieved November 25, 2016 from http://www.esterel-technologies.com/products/scade-display/

51. Little Wizard's Home Page. Retrieved November 15, 2016 from http://littlewizard.sourceforge.net/tutorial.html

52. squeakland : resources. Retrieved July 26, 2016 from http://www.squeakland.org/resources/

53. Scratch - Imagine, Program, Share. Retrieved July 21, 2016 from https://scratch.mit.edu/

54. Kodu | Home. Retrieved November 28, 2016 from http://www.kodugamelab.com/

55. Agentcubes. Retrieved November 21, 2016 from http://www.agentsheets.com/agentcubes/index.html

56. Alice.org. Retrieved December 8, 2016 from http://www.alice.org/index.php

57. Logiciel de conception de systèmes LabVIEW - National Instruments. Retrieved November 25, 2016 from http://www.ni.com/labview/f/

58. IFTTT. Retrieved November 21, 2016 from https://ifttt.com/discover

59. Unreal Engine Technology | Home. Retrieved November 16, 2016 from https://www.unrealengine.com/

60. Unity - Game Engine. *Unity*. Retrieved November 16, 2016 from https://unity3d.com

61. NaturalMotion - Home Page. Retrieved November 16, 2016 from http://www.naturalmotion.com/

62. [3] https://www.skybrary.aero/index.php/Short_Term_Conflict_Alert_(STCA)#Performance - Google Search. Retrieved July 11, 2018 from https://www.google.fr/search?q=%5B3%5D+https%3A%2F%2Fwww.skybrary.aero%2Findex.php%2FShort_Term_Conflict_Alert_(STCA)%23Performance&oq=%5B3%5D+https%3A%2F%2Fwww.skybrary.aero%2Findex.php%2FShort_Term_Conflict_Alert_(STCA)%23Performance&aqs=chrome..69i57.256j0j7&sourceid=chrome&ie=UTF-8