Towards Handling Latency in Interactive Software

Sébastien Leriche(✉), Stéphane Conversy, Celia Picard, Daniel Prun, and Mathieu Magnaudet

ENAC, University of Toulouse, Toulouse, France
{Sebastien.Leriche,Stephane.Conversy,Celia.Picard,Daniel.Prun, Mathieu.Magnaudet}@enac.fr

Abstract. Usability of an interactive software can be highly impacted by the delays of propagation of data and events and by its variations, i.e. latency and jitter. The problem is striking for applications involving tactile interactions or augmented reality, where the shifts between interaction and representation can make the system unusable. For as much, latency is often taken into account only during the validation phase of the software by means of a value which constitutes an acceptable limit. In this shor paper, we present and discuss an alternative approach: we want to handle the latency at all phases of the life cycle of the interactive software, from specification to runtime adaptation and formal validation for certification purposes. We plan to integrate and validate these ideas into Smala, our language dedicated to the development of highly interactive and visual user interfaces.

1 Introduction

An interactive software is a computer application which reacts, throughout its execution, to various sources of events. In particular, it produces a perceptible representation of its internal state [1,2]. However, the usability of an interactive application can be appreciably impacted by the delays of propagation of data and events and by its variations, i.e. latency and jitter. The problem is striking for applications involving tactile interactions or augmented reality, where the shifts between interaction and representation can make the system unusable [3,4]. Yet, while latency constraints are expressed at specification, they are often taken into account only very late in the development processes, generally by experimental a posteriori measurements, when the system is fully implemented. For instance, in some air traffic control systems such as radar visualization or remote tower, latency in the visualization of aircrafts position (and the shifts between their real position) is evaluated during experiments. Instead of redesigning the software, this may conduct to dimension the spacing limits between aircrafts [5], with direct consequences on the capacities of air traffic management.

More generally, when focusing on aeronautical software systems, the processes of certifications described in the DO-178C/ED-12C offer an important

© Springer Nature Switzerland AG 2018
M. Mazzara et al. (Eds.): STAF 2018 Workshops, LNCS 11176, pp. 233–239, 2018.
https://doi.org/10.1007/978-3-030-04771-9_18

place to formal checking. We want to take the opportunity to use formal tools and techniques to handle latency in interactive software. We are particularly interested in the Smala language, dedicated to interactive systems.

2 Djnn and Smala

Smala[1] is a language that has been designed to effectively support the development of reactive applications. Smala is built on the top of a set of C libraries named Djnn[2]. Djnn provides a core library that implements the execution engine allowing to run a tree of components [6].

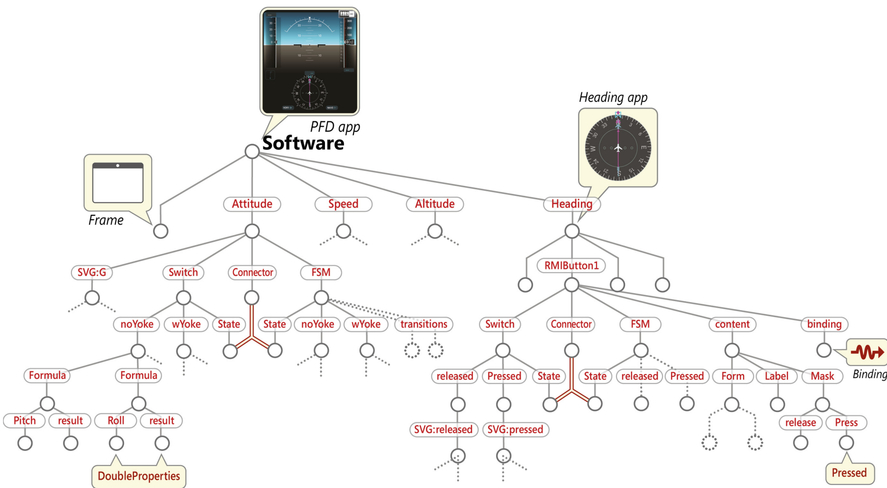

Fig. 1. Tree of Djnn components for an interactive software

Once the tree is loaded and started, the core library starts an event loop that fairly manages the events coming from the environment. On arrival, events are dispatched to the tree components. The control structures contained in this tree specify an activation graph through which the events are propagated Fig. 1.

Djnn provides libraries with various components, ranging from components for arithmetic, logic, finite state machines to graphical shapes, style components, and geometric transformations. Three rendering engines are available, one based on the Qt toolkit, another one based on Cairo, and a third one based on OpenGL.

It is possible to build a tree of components by directly using these libraries and the C language. However the task is akin to those of writing an abstract syntax tree. Thus, we designed Smala so as to provide a dedicated syntax with specific symbols that helps to visualize the interaction between components.

[1] http://smala.io.
[2] http://djnn.net.

SMALA comes with a compiler that transforms the SMALA program into a program written in the C language.

2.1 Smala Applications

Here we describe a part of a demo we developed with Smala to show some interesting points of the language. The objective was to implement a Navigation Display (ND), a standard navigation tool integrated in modern cockpits of airplanes, enhanced with interaction capacities. The ND was designed to be integrated within a full software simulator of an Airbus A320 (Prepar3D/A320 FMGS). In this paper, we will focus on the design of the interaction with a waypoint to show the expressiveness of the smala language.

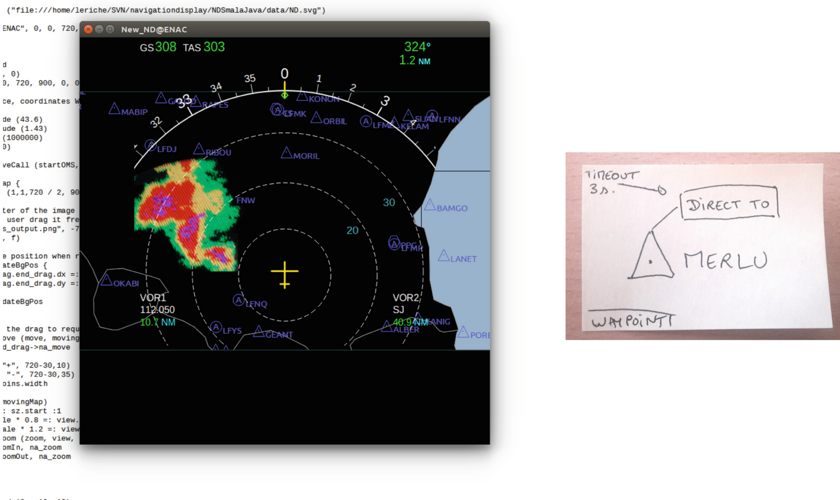

Fig. 2. Navigation Display Experiment and design of interaction with a waypoint

A flight plan mainly consists in a sequence of waypoints specifically selected for a flight before the take off. During the flight, the pilot in command might want to alter the flight plan (e.g. for meteorological reasons) and head the plane directly to another waypoint. We implemented this action on the ND with a touch command that triggers the flight management system to head to the selected waypoint. A sketch of the interaction is shown in Fig. 2: after a touch on a waypoint, a box appears with the text “DIRECT TO”, that will stay for 3 seconds. If the pilot touches this box, the command will be send to the flight management system.

The code for the smala component “waypoint” is shown in Fig. 3. This component has two states: when idle (default state) it is represented with a large triangle, including a small square at its exact position and a text label giving its

name. When selected, its color must change to green and we add the box with the text “DIRECT TO”. The basic graphical components of smala are used (Rectangle, Polygon, Colors...) and a finite state machine (FSM) is explicitely programmed to represent both states and the triggers that change the states. Thus, the component will be selected when a press will be detected inside the triangle (idle.pol.press). It will return to the idle state when either a timeout occurs or if a press is detected inside the “DIRECT TO” box (selected.rdt.press). That last action will trigger the sending of a specific message on the communication bus that links the software to the simulator.

```
1  use core
2  use base
3  use gui
4
5  import TimeOut
6
7  _define_
8  Waypoint (string name, double x_, double y_, Component view, Component frame, Component bus) {
9    Component click
10
11   Rotation r (0,x_,y_)
12   view.heading => r.a
13
14   Translation t (x_, y_)
15   x aka t.tx
16   y aka t.ty
17
18   OutlineColor oc (255, 0, 255)
19   NoFill nf
20   Rectangle p0 (-1,-1,1,1,0,0)  //big pixel center
21   FSM fsm {
22     State idle {
23         Polygon pol {
24             Point p1 (0,-20)
25             Point p2 (-20,20)
26             Point p3 (20,20)
27         }
28     }
29     State selected {
30         FillColor fcg (0, 255, 0) //green
31         Polygon pol {
32             Point p1 (0,-20)
33             Point p2 (-20,20)
34             Point p3 (20,20)
35         }
36
37         FillColor fc (0, 255, 0)
38         OutlineColor oc2 (255, 0, 0)
39         Line l (0,0,20,-20)
40         Rectangle rdt (20,-40,200,30,5,5)
41
42         FillColor fcb (0, 0, 0)
43         Text tdt (30,-20,"DIRECT TO "+name)
44         tdt.width + 20 => rdt.width
45
46         TimeOut to (3)
47         asBusOut = tdt.text =: bus.out :1
48     }
49     idle->selected (idle.pol.press)
50     selected->idle (selected.rdt.press, selected.asBusOut)
51     selected->idle (selected.to.sw.timeout)
52   }
53
54   FillColor fc2 (255, 0, 255)
55   Text label (15, 10, name)
56 }
```

Fig. 3. SMALA code for a waypoint

As a real-world example, we also completely developed the HMI of Volta, the first conventional all-electric helicopter [7]. The HMI has been built concurrently by a programmer and a graphic designer, demonstrating another powerful aspect of our approach: the strict separation of concerns between the design of the visualization and the implementation of interactions.

3 Our Approach to Handle Latency

3.1 Formal Activities Around Djnn/Smala

Although Smala is still under development, we already could experiment formal techniques for checking properties of Smala programs. For instance, we exploited the characteristics of the graph of activation [8]. This graph, deduced from the Smala code, provides all the possible activation relationships following the occurrence of an event. Thus, we managed to formally check attainability properties (i. e., an entry always ends up generating an expected exit or an alarm is always turned off in a certain configuration) or causal activation relationships (i. e., a displayed error message will never be covered by another).

In addition, with the experience gained from previous work on dedicated language and formal validation [9], we experimented in [10] the transformation of Smala code into Petri nets, with the idea to precisely define an operational semantics for Smala and to benefit from the associated formal tools and techniques. As a result, the semantics of Smala is currently under publication in a dedicated paper.

Our medium-term prospects concern the prolongation of the previous studies (based on the graph of activations) and the study of formal proof techniques applied to Smala code (translation into Caml and use of COQ, translation into Event-B).

3.2 Towards Handling Latency

We want to focus on software layers. Indeed, handling latency can be made at the hardware level with specific tools [11]. However, the end-to-end approaches existing today [12] do not allow to understand the specific issues related to software architecture choices. They are only usable to measure the latency when the system is in its validation phase.

The classical formal approach to handle latency in software systems is to consider their Worst-Case Execution-Time (WCET) [13]. WCET tools and techniques allow to verify timing properties. They are primarily made for real-time systems, and Smala programs are not. Nevertheless, since the control flow of Smala programs can be described as a tree, tree-based techniques for computing WCET could be applied. Moreover, the execution engine is being rewritten to comply with the last version of the operational semantics which is a good level to address latency issues [14].

Relying on the operational semantics, we plan to add into Smala the reification of latency properties. This should allow the programmer to add runtime adaptations (e.g. simplification/enhancement of the visualization to comply with latency constraints) and to optimize the redrawing of the graphical scene.

At last, to limit the known impacts of the operating system on latency, we are experimenting some specific versions of Djnn that can be run on OS-less ('bare') systems. This approach should result in an autonomous and complete software platform to handle latency.

4 Future Work

To achieve these goals, we aim at allowing the developer of interactive software to handle latency as a whole, during each phases of the software life cycle. Thus, this implies the conception of software tools for the measurement, visualization, specification, and formal checking of the different properties. These tools will make possible, during the design time, the objective evaluation of various software architecture solutions. At last, a methodology for designing interactive systems with latency constraints, based on these tools, should be designed.

References

1. Beaudouin-Lafon, M.: Designing interaction, not interfaces. In: Proceedings of the Working Conference on Advanced Visual Interfaces, New York, NY, USA, pp. 15–22 . ACM (2004)
2. Myers, B.A., Rosson, M.B.: Survey on user interface programming. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, New York, NY, USA, pp. 195–202 . ACM (1992)
3. MacKenzie, I.S., Ware, C.: Lag as a determinant of human performance in interactive systems. In: Proceedings of the INTERACT 1993 and CHI 1993 Conference on Human Factors in Computing Systems, New York, NY, USA, pp. 488–493. ACM (1993)
4. Ware, C., Balakrishnan, R.: Reaching for objects in VR displays: lag and frame rate. ACM Trans. Comput.-Hum. Interact. **1**(4), 331–356 (1994)
5. Cordeil, M., Dwyer, T., Hurter, C.: Immersive solutions for future air traffic control and management. In: Proceedings of the 2016 ACM Companion on Interactive Surfaces and Spaces, New York, NY, USA, pp. 25–31. ACM (2016)
6. Chatty, S., Magnaudet, M., Prun, D., Conversy, S., Rey, S., Poirier, M.: Designing, developing and verifying interactive components iteratively with djnn. In: proceedings of ERTS 2016, TOULOUSE, France, January 2016
7. Antoine, P., Conversy, S.: Volta: the first all-electric conventional helicopter. In: MEA 2017, More Electric Aircraft, Bordeaux, France, February 2017
8. Chatty, S., Magnaudet, M., Prun, D.: Verification of properties of interactive components from their executable code. In: Proc of EICS 2015, New York, NY, USA, pp. 276–285. ACM (2015)
9. Matougui, M.E.A., Leriche, S.: Validation of COSMOS DSL programs. The 2010 International Conference on Computer Engineering & Systems, pp. 307–313 (2010)
10. Prun, D., Magnaudet, M., Chatty, S.: Towards support for verification of adaptative systems with djnn. Proc. Cogn. **(03 2015)**, 191–194 (2015)
11. Zabolotny, W.M.: Automatic latency balancing in VHDL-implemented complex pipelined systems. CoRR abs/1509.08111 (2015)
12. Casiez, G., Pietrzak, T., Marchal, D., Poulmane, S., Falce, M., Roussel, N.: Characterizing latency in touch and button-equipped interactive systems. In: Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology, New York, NY, USA, pp. 29–39. ACM (2017)

13. Puschner, P., Burns, A.: Guest editorial: a review of worst-case execution-time analysis. Real-Time Syst. **18**, 115–128 (2000)
14. Asavoae, M., Maiza, C., Raymond, P.: Program semantics in model-based WCET analysis: a state of the art perspective. In: Maiza, C. (ed.) 13th International Workshop on Worst-Case Execution Time Analysis, vol. 30, pp. 32–41. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, Wadern, Germany (2013)