Building Animations of Distributed Algorithms for Educational Purposes

Boris Koldehofe
Computer Science Department
Saarlandes Un., 66123 Saarbrücken, Germany
& Chalmers Un., 412 96 Göteborg, Sweden
Email: khofer@cs.chalmers.se

Marina Papatriantafilou Philippas Tsigas
Computing Science Department
Chalmers Un. of Technology and Göteborg Un.
S-412 96 Göteborg, Sweden
Email: (ptrianta,tsigas)@cs.chalmers.se

April 1998

Summary

Distributed algorithms are by nature complicated to understand and to teach; Threads of control compete for resources, try to synchronize and dynamically change execution behaviour. Each execution involves many processes, a large amount of data of processes' local state to describe the system state and an even larger amount of date to describe complex interactions between the processes. Moreover, different executions of the same algorithm, even if they start from the same initial system configuration, may not result in the same output, due to asynchrony.

Animation of distributed algorithms graphically shows an execution of the distributed algorithm given. It can assist the teacher to illuminate the description of a distributed algorithm (including its time and communication complexity analysis) and to graphically show material that she/he has explained on the board.

This poster paper is on our work in building animations of distributed algorithms to demonstrate *(i)* the "key ideas" of the functionality of the algorithms, *(ii)* their behaviour under different timing and workload of the system, *(iii)* their communication and time complexities. The visualisation takes as input any possible execution trace of the respective algorithm, so that students (users) can view it in any possible execution that they can select. We propose the use of a set of views, which also take into account two inherrent difficulties in understanding distributed algorithms executions. These difficulties stem from the absence of *global time* in the system, which implies:

- that processes need to rely on their knowlege of *causal relations* among events in the system,
- and that in order to measure the length of an execution in time, we need to employ some mechanism related to the *dependecies* induced by each algorithm.

In the next section we describe the views that we have decided to provide for each animation and also motivate our decisions, by explaining the role each one plays in assisting the understanding of the algorithms. The code for all but one ("special") view is modularly used by all algorithms, as they are to assist in understanding issues which are common in all distributed algorithms. The idea behind the "special" view is to illustrate the *special concepts* for each algorithm (therefore the view needs to be different for each algorithm).

For our animation programs we use the Polka library [4], which is highly portable, friendly to use and has very good features for visualization, including possibility for multiple views, speed tunning, step-by-step execution and callback events to assist interactive animation.

This effort is within our project Lydian [3], which involves the development and optimization of an integraded environment to enable *(i)* development and maintainance of an archive of distributed algorithms and protocols (together with their animation programs), *(ii)* development and maintenance of an archive of system specification/configuration files, to specify different types of systems to be used for the simulation and animation of the algorithms, and *(iii)* simulation of the algorithms in appropriate systems, in order to produce traces of a variety of executions, which can be used for the animation and for the evaluation of the algorithms.

To the best of our knowledge, there has been only one attempt towards a a set of animations of distributed protocols for educational purposes, ZADA [2], on the animation package Zeus, a Modula-3 based system for specialized platforms, —not as highly portable. The effort resulted in a small archive of protocols, for each of which the set of views is fixed and the implementation is the same program as the animation (this implies

essentially fixed timing, workload, etc). Of relevance is also a nice work described by Ben-Ari in [?], where the focus is on the choice of a language to be used for implementations of distributed algorithms for demonstration and laboratory exploration.

Animation Views

It should be noted that all views evolve continuously as the execution of the algorithm evolves (continuous motion). The accompanying figures[1] illustrate a snapshot of the animation of an execution of the ECHO algorithm (broadcast with acknowledgements) [5]. The problem and the algorithm are as follows: One process(or) needs to broadcast a message to all the others and to also know when all have received it. It can only communicate with its neighbours in the network, so it sends the message to them. Each process, upon receiving the broadcast message for the first time, it propagates it to its other neighbours and it waits to receive acknowledgements from all of them before sending its own acknowledgement to the one where it got the message from for the first time. Any process receiving the broadcast message again acknowledges immediately to that sender and does not propagate it again.

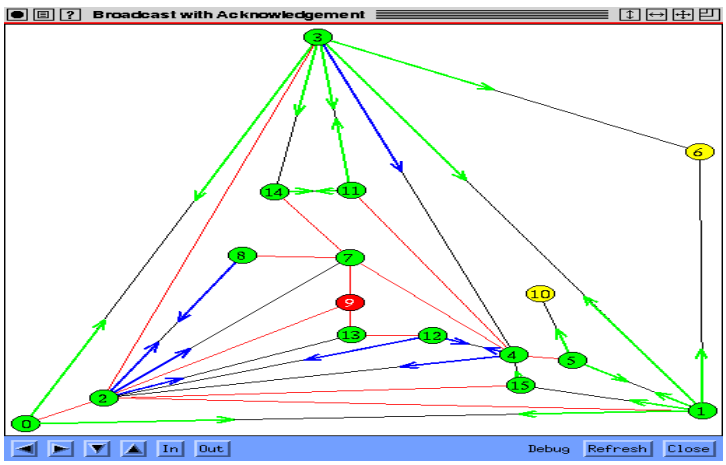

Figure 1: Basic view of the broadcast-with-acknowledgements algorithm animation

Basic View (cf. Fig 1) It illustartes the basic idea of the algorithm, hence it can look quite different for different algorithms. In the particular algorithm it shows the communication network, the propagation of the broadcast and the acknowledgement messages (arrows in green and blue respectively) and colors (green or blue) the nodes (processes) that have received the broadcast message and/or the acknowledgements, accordingly. (initially all nodes are yellow, except from the one that initiates the broadcast, which is always shown in red). As the algorithm execution evolves, *waiting chains* are formed among processes (each process in the chain waits for an acknowlegement from its next one in the chain); these chains also determine the *time complexity* of the algorithm. The edges involved processes in the chains are marked in red (in this particular algorithm they also form a spanning tree of the network at the end).

Communication View (cf. Fig. 2) It shows the contribution of each process(or) in the traffic (messages) induced by the algorithm's execution. It also shows the average number of messages per process(or) during the execution. In this example it is easy to observe that the amount of traffic that each process(or) is responsible for is proportional to its degree in the communication graph (shown in figure **??**).

Causality View (cf. Fig 2) It illustrates the causal relation between events in the system execution (arrows represent message transmission). It also shows how the processes logical clocks are incremented during the execution. Even though logical clocks are not used in all algorithm, the view is always available; its purpose is to show how would a monitoring process view the execution, based on traces as would be

[1] they are in color, hence the reader may find them more explanatory if the file is printed in color

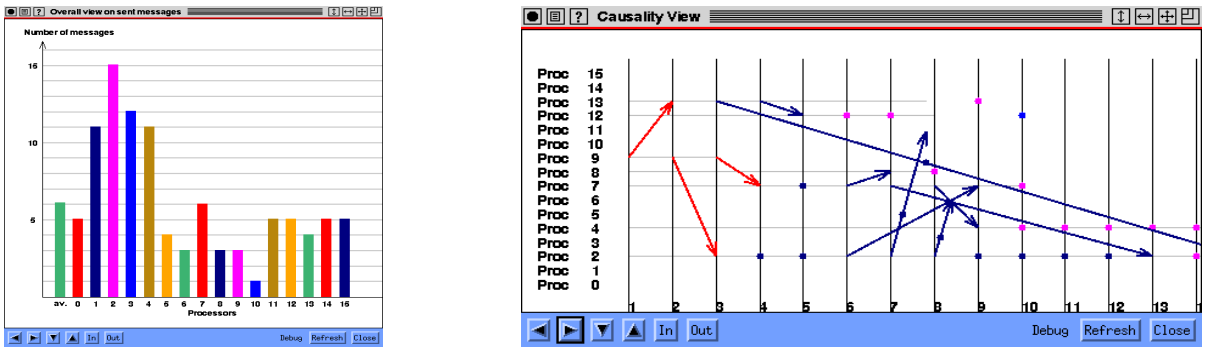

Figure 2: Views showing (a) the communication induced by each procees(or) and the average figure; (b) causality and logical times (e.g. as would be seen by a monitoring process)

given by each process seperately. This is important, as the processes in a distributed system do not have global knowledge of time. Besides, as consecutive causally related events change color, overlapping arrows with different colors visualize the degree of asynchrony in the execution. It should be noted that showing the maximum directed path in the resulting graph shows the length of the execution in units of message transmission times.

Naturally, only part of the "view" can be shown in the window, but the user is able to go back and return (as well as to zomm in and out), like with all other views.

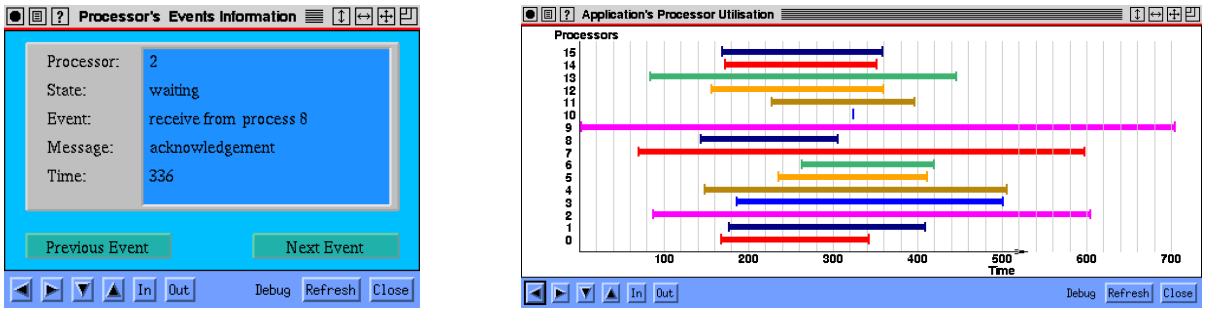

Figure 3: Process(or) step and occupation views

Process Step View (cf. Fig. 3) This gives the user the possibility to click (on- or off-line) on any node in the basic view window, to get information about its status (state, last event processed, last message received/sent, ...). It is also possible to view *interactively* the whole execution of the selected process. This can be done for any process in the system.

Process Occupation View (cf. Fig. 3) It shows in actual times (i.e. as given by the simulation trace) the period that each process is kept buzy by the algorithm during the animated execution. In this example, it can be easily observed that the initiator of the broadcast is the first to start and the last to finish; by getting the acknowledgements from its neighbours —i.e. its children in the induced spanning tree— it knows that the broadcast message reached everybody, hence it terminates.

References

[1] M. Ben-Ari "Distributed Algorithms in Java" *ACM Conference on Integrating Technology into Computer Science Education – ITiCSE '97*, p. 62-64, 1997.

[2] A. Mester, P. Herrmann, D. Jager, V. Mattick, M. Sensken, R. Kukasch, A. Ritter, S. Bunemann, P. Unflath, M. Bernhard, F. Austel, T. Alders, A. Rohrbach ZADA: Zeus-based animations of distributed algorithms and communication protocols. *T.R. Universität Dortmund*, 1995.

[3] M. Papatriantafilou, Ph. Tsigas "Lydian: a Library of Distributed Algorithms Animations" *4th International Conference on Computer Aided Learning and Instruction in Science and Engineering – CALISCE '98*, 1998 (to appear).

[4] John Stasko "POLKA Animation Designer's Package" *Technical Report*, Georgia Institute of Technology, 1995.

[5] G. Tel "Introduction to Distributed Algorithms" *Cambridge University Press*, 1994.