LYDIAN: An Extensible Educational Animation Environment for Distributed Algorithms

BORIS KOLDEHOFE
MARINA PAPATRIANTAFILOU
and
PHILIPPAS TSIGAS
Chalmers University of Technology

LYDIAN is an environment to support the teaching and learning of distributed algorithms. It provides a collection of distributed algorithms as well as continuous animations. Users can combine algorithms and animations with arbitrary network structures defining the interconnection and behaviour of the distributed algorithm. Further, it facilitates the creation of own algorithm descriptions as well as the creation of own network structures. This makes LYDIAN a flexible tool to be used with students of different skills and backgrounds.

This article gives an overview about various ideas and concepts behind LYDIAN by describing in detail the framework for an educational visualisation and simulation environment for learning/teaching distributed algorithms as well as discussing possible extensions which may improve possibilities for user interaction. Moreover, in our effort to understand better what visualisation and simulation environments such as LYDIAN need to provide we show results taken from a case study integrating LYDIAN in an undergradute distributed systems course.

Categories and Subject Descriptors: K.3.1 [**Computers and Education**]: Computer Uses in Education—*distributed algorithms, visualisation*; *simulation*

General Terms: Visualisation, Simulation, Animation

Additional Key Words and Phrases: distributed algorithms

1. INTRODUCTION

Distributed algorithms are algorithms that run concurrently on many interconnected processing elements called processors or processes. The algorithms are supposed to work correctly independent from the speed of the communication links and the structure of the network. Understanding such algorithms including their performance analysis and their correctness plays an important role in courses related to distributed systems, operating systems and computer networks. Mostly, students try to achieve an understanding of the algorithm's control flow and its performance by following the explanations on the board and the pseudo-code description presented in a technical book or paper. This approach often suffers from the large amount of data describing local state and complex interaction between processes. Simulation and animation of distributed algorithms give students the possibility to experience how the distributed algorithms evolves over time and test the algorithm under different system behaviour. Compared to executing the algorithms on a real system, the animation and simulation allows the student to interact with the system state, pause the animation and execute an instance of an algorithm multiple times. A simulator also enables students to trace behaviour which under real circumstances rarely occurs, but is important to understand the correctness or

asymptotic behaviour of the algorithm.

In this article we present LYDIAN, an environment to support the learning of distributed algorithms. LYDIAN provides a database of distributed algorithms and respective continuous animations. Students can write their own algorithm implementation in a high level language and test it with any arbitrary interconnection of processes. The provided animation framework allows interactive demonstrations of distributed algorithms. The animations do not use a fixed interconnection of processes, but allow the students to create their own networks descriptions in a visual way and apply them to the respective algorithm and animation. This way teachers can use LYDIAN in various ways depending on the level and background of their students.

Related Work. Compared to advanced system simulation tools like [Khanvilkar and Shatz 2001] LYDIAN focuses on the educational aspects of algorithm visualisation which are mainly to support the student in reasoning on the analysis of the distributed algorithm. At the time we introduced the concepts behind LYDIAN [Papatriantafilou and Tsigas 1998] and our work on building an animation framework [Koldehofe et al. 1999; Koldehofe 1999] for distributed algorithms there was only one known attempt towards a a set of animations of distributed protocols for educational purposes, ZADA [Mester et al. 1995], based on the animation package Zeus, a Modula-3 based system for specialised platforms. 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 was also the interesting work by Ben-Ari in [Ben-Ari 1997] and [Ben-Ari 2001]. There, the focus is on providing a framework for writing distributed algorithms (in a portable language) that allows students to interact with the states of a process and this way understand state changes and data structures of the algorithm. Subsequently, more tools with emphasis on different educational aspects evolved.

VADE [Moses et al. 1998] is a system that supports algorithms to be executed as Java processes on a server, and providing the client with a consistent view on algorithm events that happens on the server. The visualisation is based on WEB approach where users can view on a web page the visualisation of a selected algorithm by downloading the respective Java client. The animations supports multiple views, but makes no distinction between views for special educational purposes. The approach is mainly designed to make students view a prepared protocol, but not to implement protocols on their own. It seems that it was even thought to prevent an observer from viewing the code behind an algorithm. To the best of our knowledge, there is no recent development of this tool.

In the contrary ViSiDiA [ViSiDiA 2000] supports, like LYDIAN, an integrated approach of simulation and animation of algorithms and in this respect covers closest the aspects addressed by LYDIAN. The interconnection of processes is abstracted by a communication graph model, which can be created interactively by the user. The provided algorithms implemented in Java can be run on top of the selected network. Hereby, the code for processes is simulated with Java threads. Users can also create own protocols by using the provided library functions. However, the

user has no influence on defining timing behaviour for communication links of the network. The animations show the graph model visualising events and states by displaying labels attached to links and processes. The visualisation mainly addresses to visualise the current states, but does not provide the user with information on other issues, such as causal relations and message complexity.

The work presented in [Schreiner 2002] provides a nice object oriented framework which allows a simple specification of protocols in Java. The specification protocols reflect the automaton model presented in textbooks on distributed algorithms such as [Lynch 1996]. The animation, because of its non-continuous nature, cannot give the user a picture about actions that happen concurrently and it does not address other aspects in educational visualisation. Moreover, the network is specified with the definition of a process, i.e. the code for each process identifies the respective neighbours of processes.

Organisation of this article. This article is organised to give first a general overview of LYDIAN's components. In the following sections we present in detail the framework used for the provided animations of LYDIAN, where we give special attention to the educational aspects of the animations (see Section 3). Further, in Section 4 we introduce how LYDIAN supports the creation of own protocols with LYDIAN's simulator. The concepts behind are explained on a simple example, presenting how to implement a broadcast algorithm with LYDIAN. Section 5 describes a case study integrating LYDIAN in a basic distributed systems course. We evaluate a distributed system assignment in which students used LYDIAN to implement their algorithms. In our study neither the teachers nor the students had earlier class experience with LYDIAN. In Section 6 we discuss an extension of the visualisation framework based on Virtual Reality technology to increase possibilities for user interactivity. Finally, in Section 7 we present our conclusions and future work.

2. AN OVERVIEW ON LYDIAN

LYDIAN is intended to serve a wide range of educational purposes. For instance, one can use LYDIAN (i) to give a demonstration of prepared animations, (ii) to let students create their own network structures, which can be linked to the respective animations, or (iii) to let students create own protocols, which can be executed on LYDIAN's simulator. The interaction with LYDIAN is based on a graphical user interface (GUI) written in TCL/TK [Ousterhout 1994] which allows to access LYDIAN's archive of created created resources as well as to create own resources.

Since in the execution of a protocol there are many components involved, LYDIAN introduced the concept of *experiments* in which the user can describe the properties for relevant components. An experiment contains information about

—the *protocol* the user wants to execute,

—the underlying *network structure* describing how processes are interconnected and the characteristics of the timing behaviour,

—a *trace file* in which during the execution of the algorithm significant events are stored which can be traced by the user,

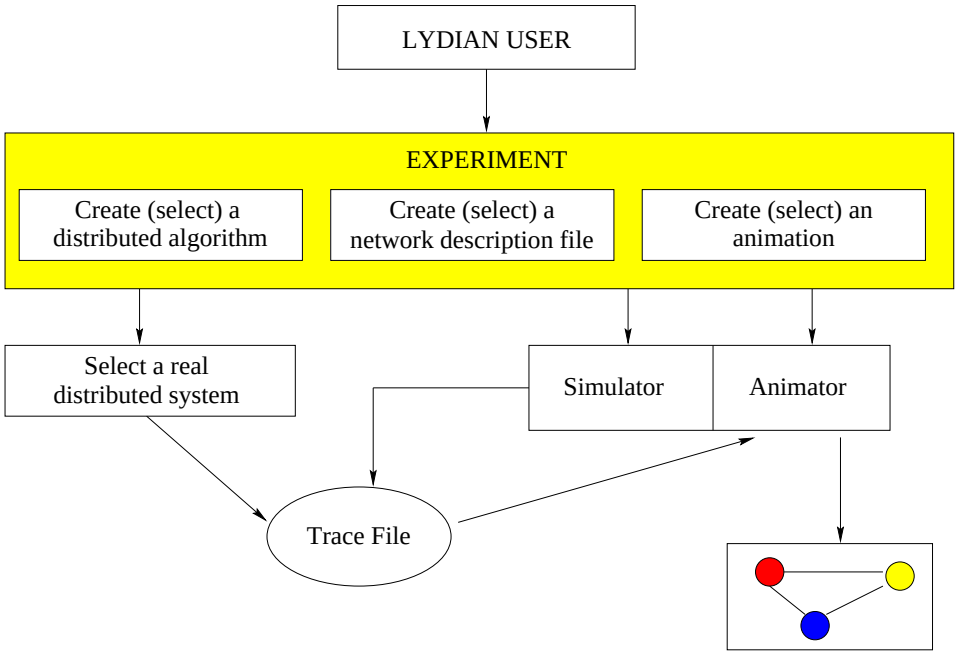

Fig. 1. An overview on LYDIAN's functionality from a users perspective.

—and an *animation* which can give a graphical representation of the events in the trace file.

The experiment is abstracted by a single window containing all experiment specific information. The user interacts with the experiment by pressing buttons representing different actions or modification choices. There are two actions a user can perform on an experiment. It is possible to "run" the experiment, i.e. the respective protocol will be executed as specified in the experiment, or one can "animate" an experiment, i.e. an animation will be shown which respects the specification of the experiment.

When a user selects to "run" the experiment, the simulator of LYDIAN will be started and in the protocol defined events written into a trace file. The user can specify in its experiment further data evaluating the protocol. It is even possible to add own events by adding debug lines to the protocol. In order to view the data the user can choose between graphical or text output for the animation. The text output is useful if users added own events and wish to see all information created by the simulator. However, most users will prefer the graphical visualisation which provides an animation with respect to the components selected within the experiments. We will describe the framework used in LYDIAN to show animations in more detail in Section 3.

The GUI of LYDIAN is designed such that the user can view all relevant information in one window and can change components of a selected experiment on the fly. For instance, in order to change the network, a user simply selects another

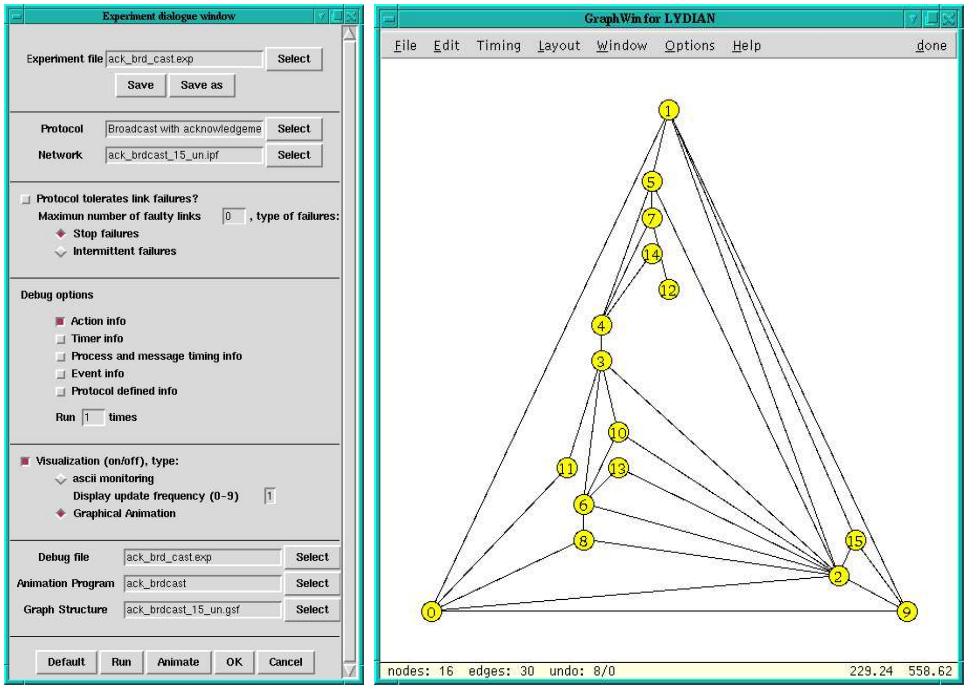

Fig. 2. The experiment dialogue and the graphwin drawing tool.

network description file in the experiment and can run the experiment and view the animation as before, but with respect to the new network structure.

It is important for students to experiment with different network structures. LYDIAN provides an easy visual way to create their own network descriptions. This component is based on LEDA [Mehlhorn and Näher 1999], a library for efficient data structures and algorithms, providing many algorithms to manipulate graphs and draw them efficiently. In LYDIAN the user simply draws a network based on a graph in which vertices represent processes and edges links between processes. Besides moving the vertices in order to achieve a pleasant layout, the user can apply a wide range of layout algorithms. For specifying the timing behaviour of the network, it is possible to choose among many different distributions valid for all processes, but also define a specific behaviour for a link or process. When saving the graph the files will be available for the simulator as well as for the animator. This way the user can see the same network in the animation as it was drawn in the graph editor.

3. THE ANIMATION FRAMEWORK OF LYDIAN

This section 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 inherent 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 knowledge 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 *dependencies* induced by each algorithm.

In the following we describe the set views that we 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 [Stasko 1995], which is highly portable, friendly to use and has very good features for visualisation, including possibility for multiple views, speed tuning, step-by-step execution and callback events to assist interactive animation.

Animation Views

It should be noted that all views evolve continuously as the execution of the algorithm evolves (continuous motion). The user can decide which views should be shown. The views can be selected by a menu window. Also a further control window enables the user in changing the speed or even halt animations in order to watch interesting parts or skip uninteresting parts of the algorithms execution. Moreover, the user has the possibility in zooming into interesting parts of the animations as he/she can move to any area of a view. This is important since by nature some animations will not be able to take place in a bounded window frame because the animator has not any previous knowledge of further executions of the algorithm. With exception of the basic view, in which an individual animation for each algorithm was developed, the offered views were designed such that they are transferable for any distributed algorithm for a message passing system although they allow some specifications. Thus further development will have to concentrate only on the main ideas of algorithms.

The accompanying figures[1] illustrate a snapshot of the animation of an execution of the ECHO algorithm (broadcast with acknowledgements) [Tel 1994]. 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, propagates the message to its other neighbours and waits to receive acknowledgements from all of them. Once, a process received all acknowledgements, it starts sending its own acknowledgement

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

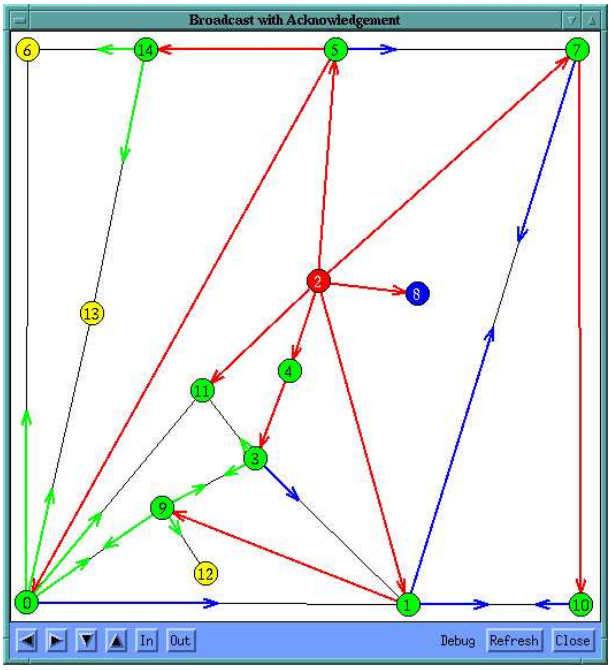

Fig. 3. Basic view of the broadcast-with-acknowledgements algorithm animation.

to the one process from which it received the message for the first time. Any process receiving the broadcast message again acknowledges immediately to that sender and does not propagate it again.

Basic View (cf. Figure 3). It illustrates the basic idea of the algorithm, hence Basic Views of different algorithms most likely look different. However, for many algorithms it is of interest to see the state of processes and messages which are sent along links. This can be achieved by showing the communication network, by colouring its nodes (processes) according to their state, and by showing moving arrows which are coloured according to the kind of message sent along an edge (link). In the particular algorithm the Basic View shows the communication network, the propagation of the broadcast and the acknowledgement messages (arrows in green and blue respectively) and colours (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 acknowledgement from its next one in the chain. These chains also determine the *time complexity* of the algorithm. The edges between two consecutive processes in the chains are marked in red. In this particular algorithm they also form a spanning tree of the network at the end of the algorithm.

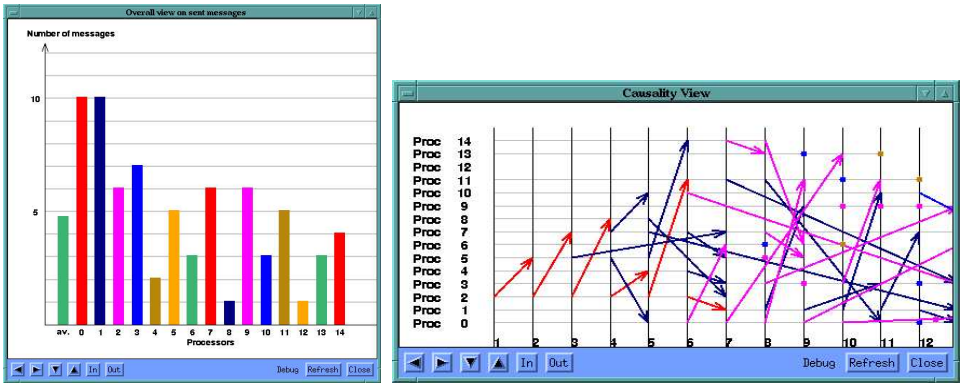

Fig. 4. Views showing (a) the communication induced by each process(or) the average and (b) the causality and logical times (e.g. as would be seen by a monitoring process.

Communication View (cf. Figure 4). This view assists in measuring the communication complexity of the algorithm and is often helpful in finding relationships between communication complexity and the structure of the communication graph. It shows the contribution of each process(or) in the traffic (messages) induced by the algorithms execution and it also shows the average number of messages per process(or) during the execution. The number of messages are displayed in a bar chart where bars grow online with the number of messages sent by a process(or). In this example it is easy to observe that the amount of traffic induced by each process(or) is proportional to its degree in the communication graph (shown in figure 3).

For some algorithm it is also of interest to have a measure of the bit complexity of messages. The actual known maximum size of a message (represented in bits) is displayed below every processes bar. The size of a message is represented by a circle of which its area content is proportional to its message size. As the message size increases online the user is able to observe how fast message sizes are increasing.

In our example algorithm the bit complexity of a message was constant so that the bit complexity is not of any interest and thus not shown in Figure 4.

Causality View (cf. Figure 4). 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 algorithms, the view is always available. Its purpose is to show how would a monitoring process view the execution, based on traces as would be given by each process separately. This is important, as the processes in a distributed system do not have global knowledge of time. Besides, as consecutive causally related events change colour, overlapping arrows with different colours visualise 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 a part of the whole view can be shown in the window, but the user is able to go back and return (as well as to zoom in and out), as it is possible in all other views.

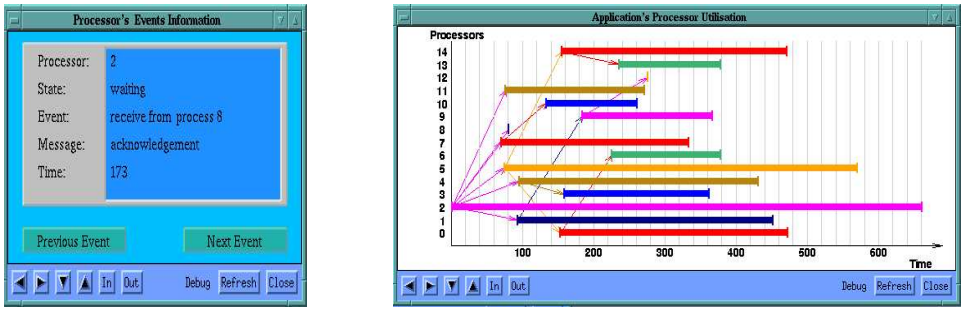

Fig. 5. Process step view and process occupation view.

Process Step View (cf. Figure 5). This gives the user the possibility to click on any node in the basic view window, to receive information about its status (state, last event processed, last message received/sent, etc.) at any point during the animation (or even after it has completed). 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. Figure 5). It shows in actual times, i.e. as given by the simulation trace, the period that each process is kept busy by the algorithm during the animated execution. If it is required by the algorithm it is also possible to distinguish how long a process was kept busy in a certain state. Therefore, a user may come to a better understanding of the algorithms time complexity by retracing, for instance with the Process Step View, why a specific process was kept busy for a long time. 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 receiving 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.

4. WRITING OWN PROTOCOLS

In courses assignments often require to write own protocols since the implementation helps the student to reflect in more detail on the main concepts behind an algorithm. However, writing own protocols on "real systems", i.e. based on libraries as MPI [Gropp et al. 1996] can become a complex task requiring a lot of time for the students to understand the respective library. LYDIAN offers a simulator including a simple language and data structures that allow to implement quickly own protocols. Although the language is based on C syntax, it requires the user to know only about the most elementary commands, e.g. needed for loops and case analysis. The simulator helps the students to implement their ideas closely to the way they are represented in ordinary textbooks and avoids much of the overhead needed when learning a real system. The simulation also gives the possibility to test the program under user defined behaviour, needed to test cases which rarely occur, while in a real system a user may be unlikely to encounter the same situation. After the execution the user can trace significant information about the execution of the algorithm within the user experiment.

LYDIAN's simulator called *DIAS* is based on the work for *DSS* [Spirakis et al. 1992]. The simulator is implemented by using the concept of Communicating Finite State Machines which allows to model the distributed system as a collection of processes communicating via communication links with messages. The simulation is event driven, i.e. when an event takes place the process performs a computation depending on its state.

As a first crucial step a programmer has to consider the states, messages and events of the algorithm and understand how the algorithm is supposed to behave on the occurrence of an event. Therefore the programmer creates a table of transitions which associate with each pair, consisting out of a state and an event, the appropriate function call to be executed. The defined code associated with a function call is valid for all processes of the network observing the same event in the same state. This way it is not required to create special code for a process, and the programmer is supported to create code valid for any interconnection of processes.

The table of events is created interactively when a user chooses to create a new protocol within LYDIAN. Depending on the states and messages of the protocol, LYDIAN will ask for each possible transition the respective function call. Finally, the user is asked to specify a file which contains the function calls for the defined transitions. LYDIAN links all this information together and creates a protocol, which a user can select within an experiment and run it with different network structures.

In the following we will illustrate the concept behind the simulator of LYDIAN by outlining how to implement a simple broadcast algorithm. The main idea behind the algorithm is to send a piece of information from one source to all processes of the network. The process starting the algorithm sends a message containing this information to all its neighbours. A process receiving the information for the first time, sends this message to all its neighbours, while a process that received the information before ignores the message.

Recall that the first step to implement an algorithm is to identify the states and messages needed. In order to distinguish between the state where a process has not received a message yet and the state where a process can ignore the information, we introduce two states: *sleeping* and *received*. For propagation of the information only one message denoted by *broadcast* is needed.

From this information LYDIAN creates the following transitions, where on the right hand side of the arrow we have to specify corresponding actions inform of a function call:

```
sleeping   ×      INITPROTOCOL       →      start()
sleeping   ×   RECMES(BROADCAST)     →     forward()
received   ×         init            →     illegal()
received   ×   RECMES(BROADCAST      →     ignore()
```

Since *ignore()* and *illegal()* are just place holders for doing nothing or throwing an exception because the algorithm entered an illegal state, the only functions which remain to be implemented are *start()* and *forward()* (cf. Figure 6).

The function *start()* is called when the protocol is initialised. We assume that the protocol is executed on a network such that only one single process is woken up by receiving the event *INITPROTOCOL*. To create such a network with the

```
start()
{
    MESSAGE * mess;
    int i;

    debug(DEBUG,''START %d %d'',me,get_time());

    for (i=0; i< PCB[me].adjacents; i++) {
        mess = create_message();
        mess→kind = BROADCAST;
        send_to(mess, i);
        debug(DEBUG,''SEND_BROADCAST %d %d %d'',me,PCB[me].adjust[i].id,
              get_time());
    }

    new_state = RECEIVED;
}


forward()
{
    MESSAGE * mess;
    int i;

    debug(DEBUG, ''REC_BROADCAST %d %d %d'', me, CURMESS→from, get_time());

    for (i=0; i< PCB[me].adjacents; i++) {
        if (CURMESS→port ≠ i) {
            mess = create_message();
            mess→kind = BROADCAST;
            send_to(mess, i);
            debug(DEBUG, ''SEND_BROADCAST %d %d %d'', me,
                  PCB[me].adjust[i].id, get_time());
        }
    }
   new_state = RECEIVED;
}
```

Fig. 6. The code implementing the simple broadcast algorithm.

respective properties we can use LYDIAN's graph drawing tool (see also section 2). The process receiving the event *INITPROTOCOL* is the initiator of the broadcast algorithm. It creates a new message and sends it to all adjacent vertices. Creating and sending a message is done by using the commands *create_message()* and *send_message()* from the simulators library. The function calls require to use the data structure *Message*, which allows us to define the type of the message, but also to send information with the message. In order to communicate with its neighbours a process must be able to know about the link information. Therefore, the simulator provides a data structure called *PCB* from which a process can extract

information which must be known locally to a process, for instance the number of adjacent processes. Finally, the process changes its state by setting the variable *new_state* to the new valid state of the process.

The function *forward()* behaves similar to the function start(), however it occurs when a process receives a message. In our description the message will not be forwarded to the sender. The contents of the received message is available in the variable CURMESS.

For evaluation of the protocol in both functions we defined debug messages. After running the protocol the user can view within the trace file of the respective experience the global order of events as they were executed within the simulator.

The example demonstrates only the most elementary functionality of the simulator, sufficient for most introductory assignments though. However, the simulator of LYDIAN provides a wide range of functions as timeout events and the possibility to specify parameters with the protocol which also makes LYDIAN usable in graduate education and could even assist researchers to implement and test ideas of their own.

5. COURSE INTEGRATION

In this section we present a case study of integrating a simulation-visualisation environment into a distributed system course. We present the evaluation of a distributed system assignment in which students used LYDIAN to implement their algorithms. In our study neither the teachers nor the students had earlier class experience with LYDIAN. The feedback received gives valuable information on what simulation-visualisation environments for distributed algorithms need to provide in order to be successfully used in class. We are not aware of any similar study in the area of distributed computing. However, the feedback we have received shows the significance of such evaluations to help users improve their performance and help them to acknowledge the wealth of tools they are provided.

The study is based on results taken from a compulsory basic undergraduate course in computer science and engineering -distributed systems-, at our university. The teachers taking part in this study have not used LYDIAN in class before, but were positive in using it from what they heard and read about it. The feedback received shows that students succeeded well in the implementation of an algorithm. Many students experienced some behaviour of the algorithm they did not expect before, and this helped them in better understanding the algorithm. The feedback also shows that students should be asked to test their implementations by exploiting various parameters inherent in network topologies to improve their knowledge. Naturally, good simulation-visualisation environments should provide such possibilities. One should also remark that animation, as expected, is of good help. For students being able to successfully exploit features of these environments, places also high demands on the documentation of the respective tool.

5.1 Description of Study

The evaluation is based on results taken from a basic undergraduate compulsory distributed system course at our university. We received answers from 50 students of the course. The questionnaire was anonymous. The main subject of most students was computer science and engineering, however there were also some students with

other major subjects. Most students were in their final year of studies, but all of them had studied for at least two years in a program at our university. Hence, most of the students were experienced in programming. The percentage of female students participating in the study was around 10%. The age of students varied from 21 to 40, where most of the students were younger than 25.

For our study it is important to mention that neither the students nor the teachers had earlier class experience with LYDIAN. The teachers had to work out their own assignment, which would correspond to approximately one week of work for each student. The idea was to use LYDIAN for a programming assignment in which students had to implement some distributed algorithm based on elementary algorithms introduced in the course, i.e. the echo broadcast algorithm, logical clocks and voting. The students could choose to implement one of the following algorithms:

—leader election, based on an echo-broadcast approach,

—leader election, based on a voting approach,

—resource allocation, based on logical clocks.

The outline of the algorithms was given with the assignments, so the students essentially had to understand the algorithm and try to implement it. Parts of the algorithms, as the echo broadcast, were also available together with an animation, but needed to be changed to be usable within the algorithm. Most students decided to implement the algorithm based on the echo broadcast approach. This is probably because this concept seemed easier to realise. The algorithms were supposed to work on any arbitrary network structure.

For our study and evaluation our interest was focused on the following aspects:

—the students' performance in implementing an algorithm,

—how students test and reason about their implementation,

—whether/how much can LYDIAN help the students get an insight into distributed algorithms and their behaviour,

—whether students consider LYDIAN to be helpful,

—general feedback on the tools administration and maintenance.

5.2 Outcome and observations

The questions of this study and the answers received are summarised in Table I and Table II. In Table III, we examine the correlation between answers of students, in order to examine the relation between:

—the factors that helped the students most in getting a better insight into distributed algorithms,

—the help that the students got from using LYDIAN and their performance in carrying out the assignment,

—and the students performance in the assignment and their appreciation of LYDIAN.

Because of the anonymity of the answered questionnaire, we cannot associate the success of the students in their assignment with the answers that they gave to

(1) Approximately how long have you used LYDIAN?

hours	0-4	5-8	9-12	13-16	17-20	21-40
students	2	11	17	7	9	4

(2) Which algorithm did you select to implement?

Election with Echo	Election with Voting	Resource Allocation
41	5	4

(3) Approximately how long did it take you to understand LYDIAN's interface?

hours	0-4	5-8	9-12	13-16	∞
students	30	8	7	2	4

(4) Approximately how long time did you spend on studying the algorithm that you had to implement?

hours	1	2	3	4	15
students	31	11	4	3	1

(5) Approximately how long did it take you to implement and test the same algorithm in LYDIAN?

hours	0-4	5-8	9-12	13-16	17-20	21-30
students	13	17	9	5	2	4

(6) Did you try to use the animation part of LYDIAN?

yes	no
24	26

Table I. Questions and Answers

our questions. We simply trust their answers regarding their understanding of the assignment material. Below we discuss the results of our study.

—About 60% of the students were done with understanding how to use LYDIAN and with implementing and testing their solution in 1.5 working-days[2], while 80% of them were done in 2.5 to 3 working days. It should be mentioned here that the whole assignment was intended to take a maximum of 5 working days.

—Nearly half of the students tried the animation part —although it was not required in the assignment. As expected, animation stimulates the students' interest in studying.

—Every third student experienced some behaviour/property of the algorithm they implemented, which they had not thought about before. Of those students the majority thought that this experience helped them to understand the algorithm better. The animation part of LYDIAN can be even more beneficial in this aspect, since the same execution can be seen multiple times, and difficult scenario can be "scrutinised" and digested better by the students' minds.

—Approximately 60% of the students tested their implementations on more than one network structures.

—Although some students experienced some difficulties with using specific parts of the tool –mainly where documentation was not sufficiently detailed– the overall impression is that the class found the tool to be useful or relatively useful for understanding distributed algorithms.

[2]measured with 8 hours per working day

(7) When or after you implemented the distributed algorithm in LYDIAN, did you experience any behaviour of the algorithm that you did not think about before?

yes	no
17	33

(8) If yes, did this help you understand better the algorithm or other material discussed in the course?

yes	no
12	5

(9) How many different network topologies did you use when testing your implementation?

number	0-1	2-5	15
students	21	28	1

(10) LYDIAN is useful for understanding algorithm in distributed computing.

Strongly disagree	Disagree	Neutral	Agree	Strongly agree
4	10	14	21	1

(11) LYDIAN is easy to use.

Strongly disagree	Disagree	Neutral	Agree	Strongly agree
17	22	7	4	0

(12) Which parts of LYDIAN do you think need to be improved?

Documentation in general	Example implementations	Interface
39	17	27

Stability	Documentation on network creation	No comment
14	21	5

(13) What is your year of study?

year	3	4	5	6	?
students	9	30	1	1	9

(14) Age.

age	21-23	24-26	older
students	27	13	5

Table II. Questions and Answers

—The majority of students who got better insight into their algorithm also tried the animation part. Maybe these students were more interested in the subject. However, the use of animations does not show any relation to the number of network topologies students used for testing purposes.

—Conforming to our expectation, the students who got more insight into the algorithm they implemented, tried more network structures in their testing. It seems worth the effort to try to stimulate students to test more and experiment more with their implementations. It is also good that LYDIAN's supported simulator provides this possibility.

—One main observation is that students who experienced unexpected behaviour of their algorithm mainly thought LYDIAN to be helpful.

—Students who used many network topologies did not think that LYDIAN is more helpful than those who did not use this feature. However the opinion is more

Students who experienced some behaviour of their algorithm they did not expect before, gave the following answers to these questions:

(1) Did you try to use the animation part of LYDIAN?

yes	no
10	7

(2) Approximately how long did it take you to implement and test the same algorithm in LYDIAN?

hours	0-4	5-8	9-12	13-16	17-20	22
students	4	8	2	2	0	1

(3) How many different network topologies did you use when testing your implementation?

number	0-1	2-5	15
students	7	10	1

(4) LYDIAN is useful for understanding algorithm in distributed computing.

Strongly disagree	Disagree	Neutral	Agree	Strongly agree
1	1	3	12	0

The following groups of students thought as follows about LYDIAN being helpful:

(1) Students tested their algorithm only with one network topology or were not aware of them.

Strongly disagree	Disagree	Neutral	Agree	Strongly agree
2	2	9	8	0

(2) Students tested their algorithm with multiple network topologies.

Strongly disagree	Disagree	Neutral	Agree	Strongly agree
2	8	5	13	1

(3) Students who tested the animation part of LYDIAN.

Strongly disagree	Disagree	Neutral	Agree	Strongly agree
0	7	4	13	0

(4) Students who experienced behaviour they did not expect before.

Strongly disagree	Disagree	Neutral	Agree	Strongly agree
1	1	3	12	0

Table III. Correlation between answers

biased, i.e. students who experienced with this feature have a stronger attitude whether they like or dislike LYDIAN in a course, while students who did not use it tended to be more neutral.

—A similar observation can be made for people who used the animation feature of LYDIAN.

An analysis of these results, from the perspective of seeing what such simulation-visualisation tools need to provide in order to be successfully used in class and how users (teachers, students) can be helped to improve the performance of the learning process, leads to the following lessons:

Tools. Since the "bottleneck" in understanding distributed algorithms is the actual concurrency and the parameters that can affect the step interleaving in each

execution, it is very important for a tool which aims at facilitating the learning process in this area, to provide:

—means for the user to experiment by varying all parameters (this helps in revealing a scenario that the user had not thought about before),
—a good way to visualise concurrency,
—the possibility for a user to observe the same execution multiple times,
—a good documentation and good user guides.

Users. Since more experimentation is shown to be effective, it is important that instructors are explicit –e.g. as part of an assignment– in asking the students to use the visualisation/animation possibilities, as well as to experiment by changing the parameters of the system and by coming up with "special", unusual constellations.

5.3 Discussion

Teaching is improved by pointing out unexpected properties/instances of the taught material. This is especially important in teaching distributed algorithms and systems, where there is a large number of parameters that affect the sequence of steps that a process will follow in each execution. LYDIAN was shown to be of good value in this respect, since it helped students to observe such instances, in an efficient manner.

Furthermore, teachers can help and get helped by using such tools in class to provide special case studies, to test cases, and to stimulate students to do own experimentation. Simulation and animation environments such as LYDIAN are shown to be useful in this respect, as well. Animation helps in understanding and also stimulates students.

LYDIAN helped in providing insight into the taught material, even though it had not been used in class by the teachers before and even though there were parts of the documentation which were not complete. The effort to improve on the supporting material –improved manuals, more examples– is expected to be appreciated and further increase the tools use-basis.

6. ENVIDIA: A VIRTUAL REALITY EXTENSION

In this section we describe an extension of LYDIAN's visualisation framework, which is intended to improve the interaction with the users. In difference to the approach described in Section 3 EnViDiA represents the communication structure in a 3D-model in which users are immersed. This way a natural interaction based on real world behaviour is possible. As it it is the case for the 2D-animations of LYDIAN, the algorithms are required to work correctly using any arbitrary interconnection of processes represented by a communication graph. However, in contrast to ordinary 2D-worlds, complex non-planar graph models can be nicely represented in three dimensions with the perspective adapting to the movements of the user. Further, within such a world the orientation is facilitated providing spatial sound. It assists the user becoming aware of the important system events.

Students working within such an environment can be more active since they walk or fly through the distributed system world in a game like scenario. EnViDiA

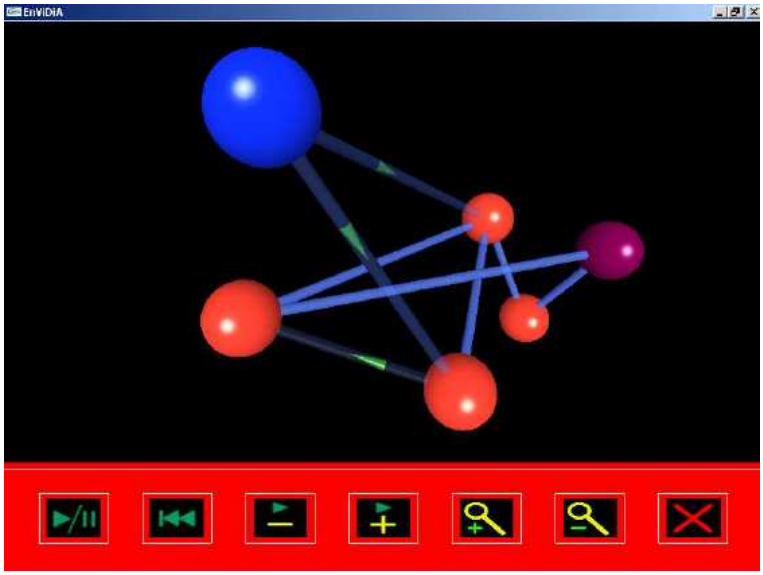

Fig. 7. This screenshot shows the EnViDiA interface on a desktop computer executing an algorithm based on resource allocation.

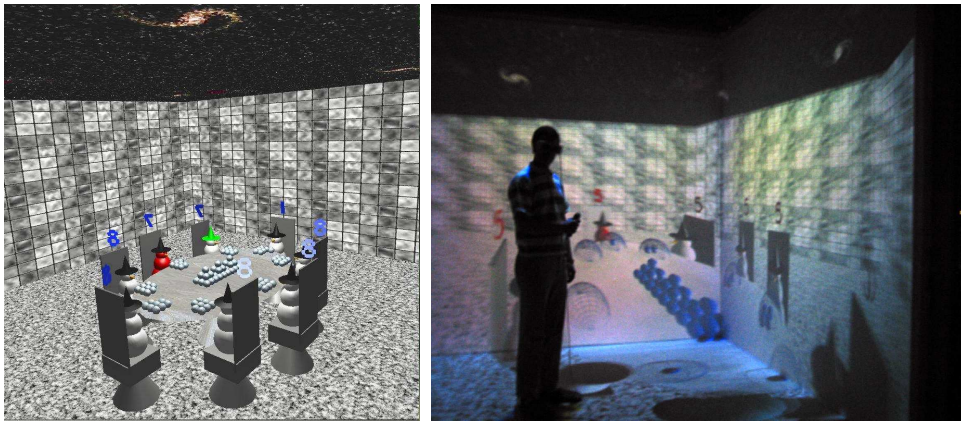

Fig. 8. An animation based on a student project allowing multiple users to collaborate on the concept of self-stabilisation.

has been developed by undergraduate students within the CAVE, an immerse VR environment. The animation framework is based on the problems the students experienced themselves when studying the distributed algorithms for the first time. Although EnViDiA is intended to be used in immerse VR environments, the tool can also be used in a simpler version on ordinary desktop computers supporting 3D-graphics. At its current state EnViDiA supports three distributed algorithms namely simple broadcast, broadcast with acknowledgement and resource allocation

based on the algorithm by Ricart and Agrawala. These algorithms are taught in a basic distributed system course at Chalmers University of Technology. Besides adding more algorithms and evaluating the tool at its current state, the main focus is on providing features to support multiple user collaboration, which are tested at the distributed concept of self-stabilisation [Koldehofe and Tsigas 2001]. The focus here is on removing the constraints of the traditional communication model in order to allow more user interaction with the distributed concept and among multiple users themselves (cf. Figure 8).

7. CONCLUSION

Although simulation and animations cannot be a replacement for students to study carefully material presented in textbooks or classes, it can well assist the student in perceiving a better understanding on the functionality of the algorithm as well as to reflect on its performance behaviour and correctness. LYDIAN provides an extensible framework which includes a wide range of material sufficient to cover a big part taught in a distributed system course. LYDIAN has been successfully tested at various universities in courses on distributed systems for students of various backgrounds.

The results presented in Section 5 show that LYDIAN can help students by pointing out unexpected instances of the taught material. This is consistent with our own experience in using LYDIAN in class. We find that programming assignments with LYDIAN can help to improve the quality of students solutions, especially if the students were encouraged to test their solutions using different network topologies. Similar effects we could observe from students adapting animations provided with LYDIAN to work with their own protocols. The input from the simulator and the visual execution of the protocol facilitates to observe bad behaviour of a protocol execution and this way also improved the quality of students assignments. We have also good experience with encouraging students to execute prepared protocols and their animations in combination with tasks in which students should verify properties, e.g. by finding an execution which matches the worst case behaviour of an algorithm. We think that this helps students to understand proof ideas and concepts and can help students to develop competence in creating own proofs.

LYDIAN is freely available and easy to deploy for Linux and UNIX platforms. Recent work has dealt with removing barriers for using LYDIAN in class by facilitating the installation, increasing portability, and allowing users to adapt LYDIAN according to their needs.

The future development of LYDIAN will happen within an open source initiative [LYDIAN 2005]. We see this step in line with the suggestions proposed in [Naps et al. 2003] and hope that this will help to improve LYDIAN in two ways:

(1) remove barriers for using LYDIAN by gathering more feedback and provide discussion forums
(2) increase the amount of educational resources and teaching material for LYDIAN.

In addition we are working on several extensions to LYDIAN which aim on the one hand to facilitate the development of own animations, but on the other hand also support collaborative work among students to learn distributed concepts.

Acknowledgements

We would like thank Phuong Hoai Ha for his contributions to teaching material distributed with LYDIAN, especially for his work on the LYDIAN manual. Moreover, we are grateful to the teachers who participated in the evaluation as well as students who have supported the development of LYDIAN.

REFERENCES

Ben-Ari, M. 1997. Distributed algorithms in Java. In *Proceedings of the 1st Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education (ITiCSE'97)*. ACM Press, 62–64.

Ben-Ari, M. 2001. Interactive execution of distributed algorithms. *ACM Journal of Educational Resources in Computing (JERIC) 1,* 2es, 2–8.

Gropp, W., Lusk, E., Doss, N., and Skjellum, A. 1996. A high-performance, portable implementation of the MPI Message Passing Interface Standard. *Parallel Computing 22,* 6 (Sept.), 789–828.

Khanvilkar, S. and Shatz, S. M. 2001. Tool integration for flexible simulation of distributed algorithms. *Software: Practice and Experience 31,* 14 (Nov.), 1363–1380.

Koldehofe, B. 1999. Animation and analysis of distributed algorithms. M.S. thesis, Universität des Saarlandes.

Koldehofe, B., Papatriantafilou, M., and Tsigas, P. 1999. Distributed algorithms visualisation for educational purposes. In *Proceedings of the 4th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education (ITiCSE'99)*. ACM Press, 103–106.

Koldehofe, B. and Tsigas, P. 2001. Using actors for an interactive animation in a graduate distributed system course. In *Proceedings of the 6th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE'2001)*. ACM press, 149–152.

LYDIAN 2005. The LYDIAN open source project. *http://sourceforge.net/projects/lydian/* .

Lynch, N. 1996. *Distributed Algorithms*. Morgan Kaufmann.

Mehlhorn, K. and Näher, S. 1999. *LEDA: A Platform of Combinatorial and Geometric Computing*. Cambridge University Press, Cambridge, England.

Mester, A., Herrmann, P., Jager, D., Mattick, V., Sensken, M., Kukasch, R., Ritter, A., Bunemann, S., Unflath, P., Bernhard, M., Austel, F., Alders, T., and Rohrbach, A. 1995. Zada: Zeus-based animations of distributed algorithms and communication protocols. *http://ls4-www.cs.uni-dortmund.de/RVS/zada.html*.

Moses, Y., Polunsky, Z., Tal, A., and Ulitsky, L. 1998. Algorithm visualization for distributed environments. In *IEEE Symposium on Information Visualization*. 71–78.

Naps, T., Cooper, S., Koldehofe, B., Leska, C., Rling, G., Dann, W., Korhonen, A., Malmi, L., Rantakokko, J., Ross, R. J., Anderson, J., Fleischer, R., Kuittinen, M., and McNally, M. 2003. Evaluating the educational impact of visualization. *ACM SIGCSE Bulletin 35,* 4, 124–136.

Ousterhout, J. K. 1994. *Tcl and Tk Toolkit*. Addison-Wesley.

Papatriantafilou, M. and Tsigas, P. 1998. Towards a library of distributed algorithms and animations. In *4th International Conference on Computer Aided Learning and Instruction in Science and Engineering (CALISCE '98)*. Gothenborg, Sweden, 407–410.

Schreiner, W. 2002. A Java toolkit for teaching distributed algorithms. In *Proceedings of the 7th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE'2002)*. ACM press, 111–115.

Spirakis, P., Tampakas, B., Papatriantafillou, M., Konstantoulis, K., Vlaxodimitropoulos, K., Antonopoulos, V., Kazazis, P., Metallidou, T., and Spartiotis, S. 1992. Distributed system simulator (DSS). In *Proceedings of Symposion on Theoretical Aspects of Computer Science (STACS '92)*. LNCS, vol. 577. Springer-Verlag, 615–616.

Stasko, J. 1995. POLKA animation designer's package. Tech. rep., Georgia Institute of Technology.

Tel, G. 1994. *Introduction to Distibuted Algorithms*. Cambridge Press.

ViSiDiA 2000. Visidia project: Visualization and simulation of distributed algorithms (2000). *http://www.labri.fr/Recherche/LLA/visidia/*.