Using Actors in an Interactive Animation in a Graduate Course on Distributed System*

Boris Koldehofe and Philippas Tsigas
Chalmers University of Technology
Computer Science Department
41296 Göteborg
Sweden
{khofer, tsigas}@cs.chalmers.se

Abstract

We describe and evaluate an experiment where actors were used to simulate the behaviour of processes in a distributed system in order to explain the concept of *self-stabilisation* in a graduate course on distributed systems.

A self-stabilising system is one that ensures that the system's behaviour eventually stabilises to a safe subset of states regardless of the initial state. Protocols satisfying this elegant property, which enables a system to recover from transient failures that can alter the state of the system, are often hard to understand, especially for students that have not studied distributed computing and systems before.

The experiment was part of an introductory course on distributed computing and systems for graduates in October 2000. The purpose of this interactive animation was to introduce to the students the basic concepts behind self-stabilisation (eligible states, transient faults, execution convergence) before their formal introduction.

All of the students had a degree either in mathematics or computing science and had taken a course on algorithms before. However, most of the students did not have a background in distributed systems or distributed algorithms. The latter was not only the motivation for preparing this method of presentation but also what made this a challenging effort.

The feedback from the class was that the concept and this teaching method were very well received. We could observe that their understanding evolved to the point that they were able to successfully come up with ideas for solutions and argue for/prove their correctness. As suggested in [1], dramatisation of executions can help the students to understand new issues and complications. This work shows that this is true even for graduate level courses. In our experiment we could conclude that dramatisation can be almost as powerful as a programming exercise in the teaching process; sometimes even more efficient, especially when we need to teach new concepts to an audience with diverse educational backgrounds. In analysing the results of our method we make a combination of the *qualitative* and *quantitative* approaches [4].

* This work is partially supported by the Swedish Research Council for Engineering Sciences.

1 Introduction to self-stabilisation

The self-stabilisation paradigm, first introduced by Dijkstra [2], defines a system as a self-stabilising one if it can recover following the occurrence of a fault that puts the system in an arbitrary state. The self-stabilising system will stabilise to a legal system state within finite amount of steps when faults stop. Hence, even though the system might be negatively affected by a failure, e.g. a power failure or a malicious process, once the failure ceases the system will start functioning again as desired after a finite number of system steps. This property is of big importance for systems like the Mars Polar Lander; for these systems it is desirable that they have the capability to fulfil their mission, in a timely manner, in the presence of failures, or accidents with no need for human interaction.

Formally, we define self-stabilisation, for a system *S*, as a property with respect to a predicate *P* over its set of configurations. The predicate *P* depends on the task that the system is executing. For instance, when the task is mutual *exclusion*, the predicate *P* is: there is at most one processor in the critical section. A system is self-stabilising with respect to predicate *P* if starting in any configuration of *S*, the system is guaranteed to reach a configuration satisfying *P* within a finite number of state transitions and *P* is a stable property (closed) under the execution of *S* [3][7].

Although self-stabilisation was introduced in 1973 its importance was not realised until 1983 when Lesley Lamport emphasised in [4] the importance of this work by Dijkstra. Since then self-stabilisation has evolved to one of the most active research fields in distributed computing and became an important concept to theoreticians and practitioners.

Traditionally, students dealing with distributed protocols and in particular self-stabilising protocols as introduced in this paper,

have problems understanding the representation of the algorithms because of state explosion and the lack of a real-life metaphor.

2 Dijkstra's self-stabilising token-passing algorithm

The algorithm that was presented to the students, first presented in [2], assumes that processes are interconnected in a *ring* and each one of them can access only information that is *local* or *shared* with its *direct neighbours* (left, right). The algorithm has to ensure mutual exclusion among the processes, i.e. that only one process at a time can perform a special computation.

The solution uses the *token passing* method; there is one special entity, called token, that processes can circulate among themselves. A process has the privilege to perform this special computation whenever it holds the token. After finishing, it passes the token to one of its neighbours. There is a consistent direction of flow of the token (say, anticlockwise), to guarantee fairness among the way that the processes receive the token.

In an arbitrary state, the system may contain no tokens or more than one tokens. A self-stabilising solution guarantees convergence to the behaviour described in the previous paragraph, with only one token in the system that flows anticlockwise. The solution is as follows:

1. P_1: **do** forever
2. **if** $x_1 = x_n$ **then** $x_1 := (x_1 + 1) \bmod (n+1)$
3. P_i $(i \neq 1)$: **do** forever
4. **if** $x_i \neq x_{i-1}$ **then** $x_i = x_{i-1}$

The token is realised by the shared memory variables $x_1,\ldots,x_n$. The processes whose if-condition is true is the process holding the token. The algorithm assumes the existence of a *leader* and that processes have consistent sense of orientation that cannot be affected by failures. Inconsistency due to faults i.e. more than one processes are holding a token is resolved by all non-*leader* processes P_i equalising the two values shared with their respective neighbours. Hence, the algorithm converges to a situation where $x_1=x_2,\ldots,x_{n-1}=x_n$ meaning only P_1 holds the token i.e. the system stabilises.

3 Dramatising Dijkstra's algorithm

The idea of the experiment was inspired by a children's game where children seated in a cycle use apples to synchronise in order to ensure that only one child is speaking at a time. The experiment consisted out of several acts. Each act used the same representation of the distributed system: a table with three actors (processes), in which apples by each actress[1] could be placed between herself and the two neighbouring actresses into two small baskets (variables x_i, x_{i-1}).

Act 1 (*Perfect system)*

In this act the actresses aimed at: i) introducing the non-stabilising algorithm, ii) explaining the token-passing idea and iii) motivate the problem. Initially, one apple was placed between two actresses. The actress that found the apple in her left hand side basket took the apple and started speaking (i.e. the actress holds the token). While speaking the actress explained to the students why she was allowed to speak and what were the next steps of the protocol that she was going to follow. Having finished the actress put the apple to her right hand side i.e. between herself and the anticlockwise-next actress, thus forwarding the token. The next actress was then enabled to act and repeated the same actions as the previous actress. The three actresses continued this act for some time.

Act 2 (*Introducing transient faults into the system)*

After a while the play moved on to the next act where one of the actresses – we call her the *evil actress* - started maliciously either adding new apples or removing apples thus bringing the system into arbitrary states. The purpose was to introduce the problems that arise when transient faults occur. Finally, the lecturer discussed with the students these problems that occurred because of the *evil actress* and the inability of the previous algorithm to cope with them.

Act 3 (*An attempt for stabilisation*)

In the third act the actresses proposed a solution that could potentially guarantee self-stabilisation (c.f. Figure 1). One (predefined) actress became the *leader*, a property which could not be affected by the evil actress. Initially, between each pair of actresses there were no apples while in the middle of the table there was a central basket with many apples (the latter is a technicality, in order to give to the actresses an "unlimited" number of apples to apply their rules of the game, so that they do not have to worry for an additional constraint in attempt to come up with stabilising rules). Recall that each actress can *only* see the apples that are placed between herself and her immediate neighbours. The *leader* first checked the number of apples in each of her hand side baskets. If there were *equally* many apples in both baskets (e.g. no apples at all, as it was initially the case in our

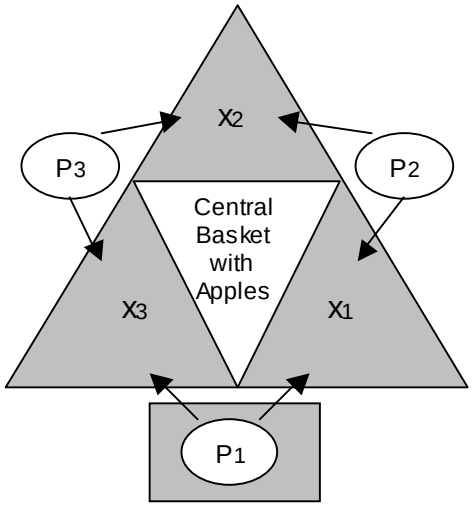


Figure 1. The table with actresses P_1, P_2 and P_3 where X_1,X_2 and X_3 denote the number of apples. Actress P_1 is the leader.

[1] We assume in this paper all our actors to be female.

experiment) the *leader* could start speaking i.e. she was holding the token (in our experiment every actress was explaining to the class what she was doing, the new rules in the system, etc). When the *leader* finished her story, she took an apple from the middle of the table and put it into the basket to her right, thus forwarding the token to the right. Each of the other actresses was allowed to speak only when the number of apples in her right side basket was *not equal* to the number of apples in her left side basket. The rest of the rules for them were, however, the same; after an actress finished speaking (explained her acting), she had to add an apple (taken from the pile in the middle) into the basket at her right hand side. The actresses continued for a while with this behaviour in order to give the audience the possibility to understand the algorithm so far.

Next, the evil actress started adding/removing apples to/from her left (*evil actions*), being able to speak at the same time while another actress was speaking. However, the system could converge to a stable state i.e. within some amount of time after the evil actions stopped, only one actress at a time was able to speak.

Act 4: *(Students find the "bug" and finalise the solution themselves)*

In this act the students had the ability to interact with the system and introduce the faults themselves. After a short discussion, the lecturer invited the students to place apples in a way that the system would fail. However, the students could observe that the system managed to recover apart from one critical scenario: a student placed between the leader and her left neighbour apples such that the *leader* had more apples to her left and the neighbour had more to her right (c.f. Figure 2). For the system to stabilise, the left neighbour of the *leader* should equalise the number of apples on her left and right side. However, at the beginning we used a simplified, "buggy" version of the algorithm not working in this case. The students managed to encounter the problem and fixed the "bug" by correcting the rule. Any non-*leader* actress was now allowed to speak whenever the number of apples at her left side was not equal to the number of apples at her right side, and then the actress had to adjust the number of apples at her right side to be equal to the number of apples at her left side. Having

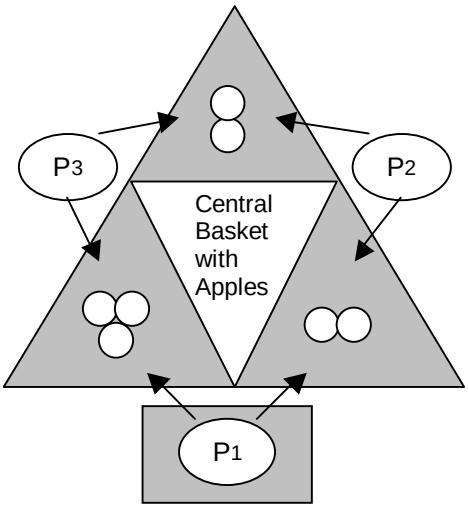


Figure 2. Deadlock situation due to wrong algorithm used by P_3.

the possibility to interact directly with the system, the students were also able to argue about the stabilising properties of the algorithm also in presence of concurrent actions.

4 Evaluation

In analysing the results of our method we make a combination of the *qualitative* and *quantitative* approaches.

In the course we had 13 students participating 11 of them handed back the questionnaire that was given to them.

All of the students had a degree either in mathematics or computing science and all of the students have had a course on algorithms before. However, the majority of the students did not study distributed systems or distributed algorithms before.

Our feedback from the class has been that the concept and the method were very well received. We could observe that the students' understanding evolved to the point that they were able to successfully *come up with ideas for solutions* and argue for/prove their *correctness.*

Knowing that many of the students were used to formal representations and had a strong background on formal methods, at the beginning, we were wondering whether our method could help or would be appreciated. Seven students answered that the dramatisation helped them a lot to understand the new concept, while only three answered that the dramatisation helped them fairly. None of them thought that it did not help at all or only a little.

As a next step, we were interested to know which methods used in the animation were the significant ones that helped the students understanding. Before executing the experiment, we were guessing that interactiveness would increase the effectiveness of the animation.

The students' answers (eight said that the interactive part helped, while two said it helped only a little) seem to confirm this hypothesis. One student wrote that the bug, which is described in the previous section, leading to the correct and final version of the algorithm helped the most.

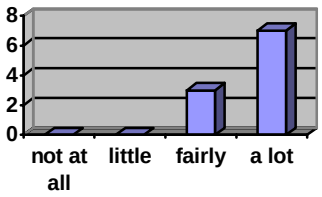
Another aspect that we were interested in evaluating, was, how much more efficient it was to have a dramatisation with human beings, compared to an interactive computer animation like the ones presented in [6][8][9]. To be able to answer this question we also asked whether the students have seen any computer animations before, which was true for seven and six of them were also familiar with educational computer animations. The answers (c.f. Figure 3), show that the majority was in favour of an animation with human beings. Some of the comments indicate that the spontaneous and not predefined reactions of human beings provide more information.

When designing the experiment, we thought that by representing the system and the protocol as a children's game we could enhance the effectiveness of the animation. The students seem to disagree: seven students answered that the children's game representation did not increase the effectiveness of the dramatisation, while only two answered that this representation actually helped them.

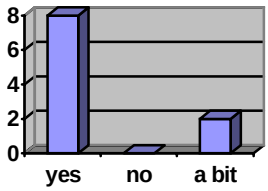
Since most of the students had a very positive attitude towards this dramatisation, it was not a surprise that all students answered that they believe that animations in general can be of big help in

understanding the behaviour of algorithms or distributed algorithms.

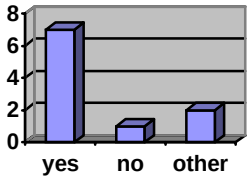
1. How much did the animation help you to understand the algorithm and the concept of self-stabilisation?



2. During the animation you had the ability to interact with the animation. Do you think that this helped you significantly to understand the algorithm and the new concepts?



3. Do you think that the fact that the animation was run by human beings instead of programs helped you significantly to understand the algorithm and the new concepts?



4. Did the fact that the animation was presented as children's game helped you?

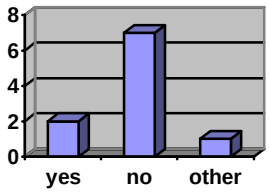


Figure 3. Some of the questions and results from the questionnaire handed out to the students.

5 Conclusion

Analysing further our experiment and the way it was received by the students, we can conclude that dramatisation can be almost as useful and powerful in the learning process as a programming exercise even in graduate courses; sometimes even more efficient, especially when we need to teach concepts to an audience with different backgrounds. New ideas are transmitted faster, while the students, by being active (and interactive) participants, have the possibility to point out the issues, which they find confusing, and to obtain more viable knowledge.

Although students seemed to favour a human animation we cannot say for sure that computer animations cannot be as powerful. We would like to continue this project by implementing the dramatisation in a virtual environment in which students can interact with virtual actors and apples.

Acknowledgements

We would like to thank: i) H. Sundell and Yi Zhang for being such great actors, ii) M. Papatriantafilou for her big help during the writing phase of the paper, and iii) the graduate students that enthusiastically participated in the project.

References

[1] M. Ben-Ari, Y.B-D. Kolikant. Thinking Parallel: The Process of Learning Concurrency. *Proceedings of ACM ITiCSE 1999*, pp. 13-16.

[2] E.W. Dijkstra. Self-stabilizing Systems in Spite of Distributed Control. *Communications of the ACM*, Vol. 17, No. 11, Nov. 1974.

[3] S. Dolev. Self-stabilization, *MIT Press*, 2000.

[4] Y.B-D. Kolikant, M. Ben-Ari, S. Pollack. The Anthropology of Semaphores. *Proceedings of ACM ITiCSE 2000*, pp. 21-24.

[5] L. Lamport. Solved problems, unsolved problems and non-problems in Concurrency. PODC84, pages 63-67, 1983.

[6] Lydian - An Educational Animation Environment for Distributed Algorithms and Protocols (2000). http://www.cs.chalmers.se/~lydian/

[7] M. Schneider. Self-Stabilization, *ACM Computing Surveys*, 1993.

[8] The VADE project (2000). http://www.wisdom.weizmann.ac.il/~ulitsky/java/proj/

[9] ViSiDiA Project: Visualization and Simulation of Distributed Algorithms (2000). http://dept-info.labri.u-bordeaux.fr/~stefan/