EWD391
Self-Stabilization in Spite of Distributed Control

A systematic way for finding the algorithm ensuring some desired form of co-operation between a set of loosely coupled sequential processes can in general terms be described as follows: the relation "the system is in a legitimate state" is kept invariant. As a consequence, each intended individual process step that could possibly cause violation of that invariant relation has to be preceded by a test that it won't do so, and depending on the outcome of that test the critical process step is either caused to take place or it —and with it the process of which it is a part— is delayed until a more favourable system state has been reached. With a suitable choice of the set of legitimate states one can indeed introduce the rule that a critical process step will be delayed only as long as its execution would lead to violation of the corresponding invariant relation.

The resulting design is readily implemented if the different sequential processes can be granted mutually exclusive access to a common store in which the current system state is recorded. Then a relation between (the values of) the variables in that commonly accessible store is the core of what we could call "the centralized control".

A complication arises when there is no such commonly accessible store and "the system state" must be recorded in variables distributed over the various processes, and furthermore the communication facilities are limited in the sense that each process can only exchange information with "its neighbours", a (possibly small) subset of the total set of processes. (We can view the processes as nodes of a connected graph in which each of the (sparse) set of edges denotes the neighbour relation.) The complication is that a node's behaviour can only be influenced by the part of the total system state description that is available in that node: local actions taken on account of local information must accomplish a global objective. Such

systems (with what is quite aptly called "distributed control") have been designed, but all such designs I am familiar with are unstable in the sense that, when once in an illegitimate state, they could remain so forever. I call a system "self-stabilizing" when, regardless of its initial state, it is guaranteed to arrive at a legitimate state in a finite number of steps. (Whether the property of self-stabilization is interesting as a start procedure, for the sake of system robustness, or merely as an intriguing problem, is a question that falls outside the scope of this article.)

Unable to decide on theoretical grounds whether non-trivial self-stabilizing systems with distributed control could exist at all, I decided to try to design one under the following constraints and objectives.

We consider a system built from $N + 1$ finite state machines numbered from 0 through N. (The state space for the total system is then the Cartesian product of the $N + 1$ individual state spaces of the respective machines.) The machines are arranged in a ring, i.e. for $0 \leqslant i < N$, machine $nr.i$ has machine $i + 1$ as its right-hand neighbour, and machine N has machine 0 as its right-hand neighbour.

In the middle of the ring stands a demon, each time giving, in "fair random order", one of the machines the command "to adjust itself". (In "fair random order" means that in each infinite sequence of successive commands issued by the demon, each machine receives the command to adjust itself infinitely often.) Upon "adjustment" a machine goes into a (new) state, which must be a function of its own (old) state and the current states of its (two) neighbours.

Furthermore, as a function of its own state (and possibly of the states of its neighbours) a machine may be "privileged". The legitimate states are defined as those states in which exactly one machine is privileged and for which all possible successor states are legitimate as well; furthermore it is required that then the privilege will rotate around the ring.

SIDE REMARK. I was hoping for an existence proof of self-stabilizing systems with distributed control: a ring is then one of the most natural, simple connection graphs. My choice of legitimate states, viz. requiring convergence towards a solution of the mutual exclusion problem, is understandable for historical reasons [1], [2], [3], [4], it is also justified by its central position in the whole field of controlling co-operation between loosely coupled processes. Finally, the choice of the demon was suggested by a recent experience with a cyclic relaxation problem in which "fair random relaxation" would converge to a limit, while simultaneous relaxation could lead to oscillation (EWD386, unpublished). So much for the justification of the problem choice.

Again I beg my intrigued readers to stop reading here and to try to solve the stated problem themselves, for only then will they (slowly!) build up some sympathy with my difficulties: the problem has been with me for many months, while I was oscillating between trying to find a solution

lled "distributed control") have been familiar with are unstable in the sense e, they could remain so forever. I call a dless of its initial state, it is guaranteed finite number of steps. (Whether the sting as a start procedure, for the sake n intriguing problem, is a question that .)

rounds whether non-trivial self-stabiliz- l could exist at all, I decided to try to traints and objectives.

$N + 1$ finite state machines numbered or the total system is then the Cartesian ate spaces of the respective machines.) ng, i.e. for $0 \leqslant i < N$, machine $nr.i$ has hbour, and machine N has machine 0 as

ds a demon, each time giving, in "fair es the command "to adjust itself". (In in each infinite sequence of successive each machine receives the command to n "adjustment" a machine goes into a ion of its own (old) state and the current

s own state (and possibly of the states of e "privileged". The legitimate states are xactly one machine is privileged and for are legitimate as well; furthermore it is ll rotate around the ring.

existence proof of self-stabilizing systems is then one of the most natural, simple f legitimate states, viz. requiring conver- mutual exclusion problem, is understand- , [3], [4], it is also justified by its central controlling co-operation between loosely choice of the demon was suggested by a elaxation problem in which "fair random limit, while simultaneous relaxation could ublished). So much for the justification of

lers to stop reading here and to try to solve for only then will they (slowly!) build up ulties: the problem has been with me for illating between trying to find a solution

—and many an at first sight plausible construction turned out to be wrong! — and trying to prove the non-existence of a solution. And all the time I had no indication in which of the two directions to aim, nor of the simplicity or complexity of the argument —if any!— that would settle the question.

* * *

The crucial observation is that, in general, the problem cannot be solved if, in addition, we require our machines to be identical. For if the number of machines is non-prime, our starting situation can have a cyclic symmetry of degree n ($2 \leqslant n \leqslant N/2$) and if then the demon —and he is free to do so!— gives his first n commands equally spaced around the ring, the cyclic symmetry will not have been destroyed. If the demon continues with such fair (but nasty) behaviour, we shall never reach the state after which, forever, a single machine will be privileged. Making not all machines identical can be accomplished in two extreme ways: either by making them all different or by making one exceptional. In view of our obligation to enforce asymmetry, one exceptional machine and all others mutually equal seems the most promising choice.

Secondly, it is not a priori excluded that the nett effect of the command "adjust yourself" is nil, viz. that the new state of the machine to which the command was given equals its old state. In a legitimate state we have no particular desire to let the adjustment command have any effect when given to a machine far away from the privileged one. To simplify matters we can look for a solution in which the adjustment command has only effect when directed towards a machine that at that moment is privileged, and the result of whose adjustment will be that it loses its privilege. When now the function "privileged" is chosen such that at least one machine must be privileged, then "dead ends" are excluded a priori: the ring will remain alive, and we can concentrate on the requirement that the system converge to the state from which a single privilege will rotate past all machines.

Thirdly, we may feel tempted to introduce some sort of counter, but because we are confined to finite machines, true counters are excluded and the best we can hope for are counters counting modulo K, where K is some sufficiently large constant (certainly > 1). For two counter values modulo K, the maximum or minimum is not defined and we cannot hope to establish progress towards the legitimate state because some "maximum counter value" decreases. But equality and a successor function that can be applied a limited number of times without leading to ambiguity are well-defined. This suggests defining the function "being privileged" in terms of equality of states.

In terms of equality we can define a function "being privileged" such that at least one machine is privileged quite easily when bearing in mind that one machine —let it be machine 0— should be exceptional. Let for $1 \leqslant i \leqslant N$ machine i be privileged when its state differs from that of

machine $i - 1$, i.e. when $x[i] \neq x[i-1]$. We choose this —rather than the other way round— because now non-privileged implies $x[i] = x[i-1]$ and equality is transitive: in other words, when all machines except machine 0 are non-privileged, $x[0] = x[N]$ and when we define this as the condition for machine 0 being privileged, our requirement of at least one machine being privileged is therefore met.

Furthermore we had suggested that adjustment would cause the machine in question to loose its privilege. For the normal machines ($1 \leqslant i \leqslant N$) we have no freedom anymore: adjustment of machine i means

"**if** $x[i] \neq x[i-1]$ **then** $x[i] := x[i-1]$ **fi**" .

For the exceptional machine, 0, I now suggest

"**if** $x[0] = x[N]$ **then** $x[0] := (x[0] + 1)$**mod** K **fi**"

and it is only here, where a new state has to be generated, that it becomes significant that we consider the machine states x as a counter modulo K.

To start with, we remark that when a machine "fires" —if we may use that term for the non-nil adjustment that takes place when the demon gives the command to a privileged machine— it loses its privilege, it may give the privilege to its righthand neighbour and to no one else. Because at least one machine must be privileged, firing of the only privileged machine will always give the only privilege to its righthand neighbour: once in a legitimate state the system will remain in a legitimate state and the privilege will rotate around the ring.

Furthermore: suppose that the exceptional machine is not privileged, i.e. $x[0] \neq x[N]$, then in a finite number of commands it will become privileged. For let j be the minimum value such that $x[j] \neq x[0]$; because j is the minimum value, $x[j-1] = x[0]$ and therefore $x[j] \neq x[j-1]$, i.e. machine j is privileged. In a finite number of commands the demon will point to it, thus increasing j if $j < N$ or making $x[N] = x[0]$ if $j = N$, i.e. making the exceptional machine privileged. So the exceptional machine will continue forever to get the opportunity to fire.

Let us now investigate what happens when we start the system in an arbitrary state. When the exceptional machine fires for the first time, we colour its new state blue and all other states white; from then onwards each state created by the exceptional machine or copied from a blue state by a normal machine will be blue as well. If h is the number of times the exceptional machine fires while $x[N]$ is still white, then —because $K > 1$— h will satisfy $h \leqslant N$: after the first firing, the copying process along the chain of normal machines can supply machine N at most with another $N - 1$ further white states, differing in succession.

Without loss of generality we could have chosen initially $x[0] = K - 1$. If $K > N$, then the first N firings of the exceptional machine have created the blue states from 0 through $N - 1$, and scanning the blue states, starting at the exceptional machine and going to the right, we find a sequence of

$= x[i-1]$. We choose this —rather than the
v non-privileged implies $x[i] = x[i-1]$ and
words, when all machines except machine 0
] and when we define this as the condition
l, our requirement of at least one machine
t.
ed that adjustment would cause the machine
e. For the normal machines ($1 \leqslant i \leqslant N$) we
ıstment of machine i means

$:= x[i-1]$ **fi**"

I now suggest

$= (x[0] + 1)$**mod** K **fi**"

w state has to be generated, that it becomes
machine states x as a counter modulo K.
at when a machine "fires" —if we may use
ment that takes place when the demon gives
achine— it loses its privilege, it may give the
our and to no one else. Because at least one
firing of the only privileged machine will
to its righthand neighbour: once in a legiti-
in in a legitimate state and the privilege will

he exceptional machine is not privileged, i.e.
number of commands it will become privi-
value such that $x[j] \neq x[0]$; because j is the
$x[0]$ and therefore $x[j] \neq x[j-1]$, i.e.
inite number of commands the demon will
$j < N$ or making $x[N] = x[0]$ if $j = N$, i.e.
e privileged. So the exceptional machine will
ortunity to fire.
t happens when we start the system in an
ptional machine fires for the first time, we
l other states white; from then onwards each
l machine or copied from a blue state by a
as well. If h is the number of times the
$x[N]$ is still white, then —because $K > 1$—
first firing, the copying process along the
supply machine N at most with another
ering in succession.
could have chosen initially $x[0] = K - 1$. If
of the exceptional machine have created the
1, and scanning the blue states, starting at
going to the right, we find a sequence of

non-increasing blue x-values. At the next firing of the exceptional machine with $x[0] = N - 1$, also $x[N] = N - 1$ must hold. At that moment, however, $x[N]$ must be blue as well and therefore *all* states must be $= N - 1$, i.e. the system has arrived in one of its legitimate states. And this completes the proof for self-stabilization provided $K > N$ (and, for smaller values of K, counter examples kill the assumption of self-stabilization).

* * *

So far, so good, but one may object to using a rather powerful demon that may be very awkward to implement. Can we eliminate that centralized agency, can we replace it by "a distributed demon"?

Each variable $x[i]$ is only inspected and assigned to by machine i and only inspected by its right-hand neighbour. We assume each variable $x[i]$ equipped with its own, private, two-way switch, which excludes simultaneous access by the two neighbours it connects. And we assume that the machines will adjust themselves with a finite speed and a finite frequency, instead of waiting for the demon's command. Does it work? Amazingly it does without any further refinements.

Two simultaneous adjustments of non-neighbouring machines have no mutual interference at all. An adjustment by the exceptional machine cannot suffer from simultaneous activity of its lefthand neighbour N, because $x[N]$ is inspected only once per adjustment. But adjustment of a normal machine i, although possibly inspecting $x[i-1]$ twice during a single adjustment, cannot suffer from its lefthand neighbour activity either: if $x[i-1]$ changes its value between the two inspections, the first value differed from $x[i]$; if the second value differs from $x[i]$ as well, the program behaves as if this value was also offered the first time, while if the second value equals $x[i]$, the assignment has no effect and it is as if the adjustment had not taken place at all!

Conclusion

Self-stabilizing systems with distributed control do exist in the sense that local decisions force the system towards satisfying and then maintaining a global requirement. In particular, local mutual exclusion is a sufficient building block for eventually achieving mutual exclusion globally.

References

[1] Dijkstra, E.W. Solution of a problem in concurrent programming control. *Comm.* ACM 8, 9 (Sept. 1965), 569

[2] Knuth, D.E. Additional comments on a problem in concurrent programming control. *Comm.* ACM 9, 5 (May 1966), 321–322

[3] de Bruijn, N.G. Additional comments on a problem in concurrent programming control. *Comm*. ACM 10, 3 (March, 1967)
[4] Eisenberg, M.A. and McGuire, M.R. Further comments on Dijkstra's concurrent control problem. *Comm*. ACM 15, 11 (Nov. 1972), 999