Assuring adaptive behaviour in self-organising systems

J.W. Sanders
International Institute for Software Technology
United Nations University
Macao SAR China
Email: jeff@iist.unu.edu

Graeme Smith
School of Information Technology and Electrical Engineering
The University of Queensland
Australia
Email: smith@itee.uq.edu.au

***Abstract*—The important notion of adaptivity of a distributed information system is formalised, extending Dijkstra's idea of self stabilisation. The formalisation quantifies the extent to which a system adapts, enabling degrees of adaptivity to be specified and hence assured in an implementation. The ideas are expressed without commitment to any particular formal notation and demonstrated on a cluster formation algorithm for mobile *ad hoc* networks.**

***Keywords*-adaptivity, self-organising, self-$*$, formal methods.**

I. Introduction

Many contemporary distributed systems are required to be 'self-$*$': able autonomously to configure, organise, manage and repair in response to change or disturbance either from the environment or internally, caused by malfunction or failure. In some cases, this is done through machine learning techniques such as genetic algorithms or reinforcement learning. In others, the adaptivity mechanisms are explicitly engineered into the system. It is this latter case that is of primary interest in this paper.

The global behaviour of such systems depends on certain constraints being met. For example, if the components can communicate to only immediate neighbours then broadcasting a message (to all components in the system) requires that all components be (transitively) connected. Such constraints do not correspond to the global behaviour desired from the system, but are sufficient for it to hold.

The ability to adapt to a given change or disturbance depends on (a) whether these constraints hold after the change or disturbance and, if not, (b) the ability of the system to restore the constraints *via* a finite number of system actions. In fact, adaptivity can be quantified; firstly with respect to the functionality of the disturbance and secondly with respect to the number of actions it takes for the system to restore the constraints.

Similar observations have been made before. Dijkstra [4] uses them to capture the notion of *self stabilisation* and shows that it is possible to have distributed self stabilisation. For the enormous impact of Dijkstra's ideas we refer to Schneider's survey [15] and Herman's bibliography [9].

Güdemann *et al.* [7] present an architecture for Organic Computing [18] which divides a system into a collection of agents and an observer/controller (OC) which monitors the agents and tells them when to reconfigure. Although the approach is not completely decentralised, it is argued that the OC could be distributed amongst the agents. In this specific context, the authors introduce a notion of invariant which is basically an assignment of roles to agents taking into account agent capabilities. The OC monitors this invariant and when it no longer holds (*e.g.*, because an agent loses a particular capability), calculates a new configuration to make it hold. A SAT solver, *i.e.*, an automatic constraint solving tool, is proposed to calculate the new reconfiguration [14].

This differs from our approach by concentrating on only a certain kind of invariant (role allocation) for a specific architecture. It also defines an implementation strategy which explicitly checks the invariant in order to invoke an explicit reconfiguration. We, on the other hand, are not interested in defining an implementation strategy for adaptivity but reasoning about arbitrary implementations.

Georgiadis *et al.* [5] use constraints on the system architecture to reconfigure a system when a component is added, removed or changed. Again the idea is to restore certain constraints (this time on the architecture) to ensure the system functions correctly. It is closer to the approach of Güdemann *et al.* than our work since it relies on explicit monitoring of the constraints and explicit invocation of reconfiguration in the implementation.

The aim of our work is to provide formal assurance that a proposed system design adapts to particular changes and disturbances. While the approaches of Güdemann *et al.* and Georgiadis *et al.* provide a level of confidence that a system will adapt, they do not provide guarantees. In Section II we provide a formal definition of adaptivity based on those ideas. In Section III, we introduce a case study based on a passive clustering algorithm for a mobile *ad-hoc* network [6]. We formally derive results concerning the ability of the algorithm to adapt to new connections being formed, and existing connections being broken, in Sections V and VI respectively. We conclude in Section VII.

II. A FORMAL NOTION OF ADAPTIVITY

A. Systems

The systems targeted in this paper are multi-agent systems and their abstractions. A *multi-agent system* is viewed as a distributed system whose components, the agents, are autonomous in the sense of being responsible for the actions on their own state: the only actions that directly change an agent's state are the actions invoked by that agent. There are no system actions other than those of the agents. Agents communicate with each other by either shared variables or agent-to-agent channels.

In summary, system state comprises the states of the agents together with a global component (reflecting either shared variables or channel communication), and each agent's state is accessed (read and written) by only that agent, using invocation of named actions. However such a system is conveniently *specified* using actions that change the states of several agents. So this paper considers more general distributed systems, even when the eventual target of a formal analysis using these techniques is a multi-agent system.

A *system* consists of a quadruple $\mathcal{S} = (S, S_0, S_1, A)$ in which: S is the type of states; $S_0 \subseteq S$ are the *initial states*; $S_1 \subseteq S$, such that $S_0 \subseteq S_1$, are the states satisfying the *system invariant*; and A is an indexed family of actions, called *system actions*, each of which maintains the system invariant S_1. If a system action is enabled in some state then it may accept input (either by reading global variables or from its input channels) and, if it does not diverge, terminates (nondeterministically, in general) in some final state, delivering output (either by writing to global variables or outputting on its output channels).

Actions in the distributed system may be concurrent, but they are assumed to be atomic so that from the point of view of functional behaviour it suffices to consider a linear semantics of behaviours, in which concurrency is captured by interleaving.

Each action has two predicates in its initial state: its *guard*, which holds at just those states where the action is enabled; and its *termination condition*, which holds at just those states satisfying the guard from which termination is sure (since the action need not be deterministic, the termination condition holds at x only if the action terminates *whenever* enabled in x). Finally it has a binary predicate in its initial and final states, its *postcondition*, describing the effect when the action is enabled and terminates. *Refinement* of actions, $\alpha \sqsubseteq \alpha'$, holds if α' is less enabled but more terminating and more deterministic [3].

In this section no particular formalism is presumed for describing $\mathcal{S}$. It is thus appropriate to capture (linear) system semantics by sequences of actions and the state transitions they achieve (as in say [15]). If $E \subseteq S$ and σ is a well-defined sequence of actions (*i.e.*, each enabled and terminating) $E \underline{after}\, \sigma$ denotes the states obtainable from those in E by executing the sequence of actions σ. By $\mathcal{S} \underline{after}\, \sigma$ is meant $S_0 \underline{after}\, \sigma$. Throughout we assume well-defined sequences.

If $\mathcal{S}$ and $\mathcal{S}'$ are systems, recall that $\mathcal{S}$ is said to be *refined by* $\mathcal{S}'$, $\mathcal{S} \sqsubseteq \mathcal{S}'$, if the behaviours of $\mathcal{S}'$ are included in those of $\mathcal{S}$. This includes data refinement [3] but not action refinement [1] (in which the actions of the concrete system may be more numerous). The latter is particularly useful for developing multi-agent systems [16] and we intend to investigate its relationship to adaptivity in future work.

The $\underline{after}$ operation is monotonic in several senses.

Theorem 1: Let $\mathcal{S}, \mathcal{S}'$ be systems, let $E, F \subseteq S$, let $\sigma, \tau : A^*$ and let α, α' be actions. The catenation of σ and τ is written $\sigma.\tau$. Then (straight from the above definitions)

1) If $\sigma : A^*$ then $\mathcal{S} \underline{after}\, \sigma \subseteq S_1$
2) $(E \underline{after}\, \sigma) \underline{after}\, \tau \;=\; E \underline{after}\, \sigma.\tau$
3) If $\alpha \sqsubseteq \alpha'$ then $E \underline{after}\, \alpha \supseteq E \underline{after}\, \alpha'$
4) If $E \subseteq F$ then $E \underline{after}\, \sigma \subseteq F \underline{after}\, \sigma$
5) If $\mathcal{S} \sqsubseteq \mathcal{S}'$ then $\mathcal{S} \underline{after}\, \sigma \supseteq \mathcal{S}' \underline{after}\, \sigma$.

B. Adaptivity

Environmental duress on a system $\mathcal{S} = (S, S_0, S_1, A)$ is captured (and quantified functionally) by an action Z, called *external* to distinguish it from the system actions (*i.e.*, the elements of A, which by definition preserve the system invariant), whose initial state is in S_1 but whose final state need not satisfy the system invariant and so is merely in S.

A system $\mathcal{S}$ is said to be *Z-adaptive*, for external action Z, if after an occurrence of Z the system invariant is always restored by some finite number of system actions, under the assumption that no further external actions occur during that time. Writing $\trianglelefteq$ for prefix, it is expressed

$$\forall\, \sigma : A^* \;\cdot\; \forall\, \tau : A^\omega \;\cdot\; \sigma.Z.\tau \text{ is well defined } \Rightarrow$$
$$\exists\, \upsilon : A^* \;\cdot\; \upsilon \trianglelefteq \tau \;\wedge\; \mathcal{S} \underline{after}\, \sigma.Z.\upsilon \subseteq S_1\,. \quad (1)$$

When we wish to focus on the number of actions required to restore the invariant, we say instead that the system $\mathcal{S}$ is *n-Z-adaptive*, where n is the least number of system actions required to restore the invariant S_1, *i.e.*, the length of the shortest such υ. In particular, if $\mathcal{S}$ is 0-Z-adaptive then the external action Z preserves the system invariant S_1, and so $\mathcal{S}$ need do nothing to 'recover from Z'. The larger the value of n, the longer (in terms of actions) it takes $\mathcal{S}$ to recover from Z (measured with respect to a linearised semantics).

C. Examples

A comprehensive, rigorous example is provided in Section III. In this section two informal examples are presented to show the kind of system to which the ideas apply; both can be developed in the style of Section III.

Free flight [10] is the term used to refer to the distributed system of aircraft control in which, rather than having a centralised control tower, each aircraft is responsible for maintaining an object-free sleeve around itself in order to avoid

collision. Typical actions Z might consist of injecting a new airborne object into the system, or instantaneously changing weather conditions (hence constraining flight paths). The algorithm being used for free flight is adaptive to such a Z if after some number of adjustment-actions on the part of the aircraft, the sleeves are restored.

Self stabilisation [4] is the (extreme) case in which the external action Z is unconstrained. Continuing with Dijkstra's example, in a *token ring* Z might add a finite number of extra tokens to the ring. It is a nice property of Herman's probabilistic algorithm [8], [13] that it adapts quickly, with respect to a probabilistic semantic model: with n processors it adapts on average in $\Theta(n^2)$ steps.

D. Results

The notion of adaptivity is monotone under refinement of the external action.

Theorem 2: If $Z \sqsubseteq Z'$ and $\mathcal{S}$ is n-Z-adaptive then $\mathcal{S}$ is n'-Z'-adaptive for some $n' \leq n$.

Proof: By definition (1), for any $\sigma : A^*$ and $\tau : A^\omega$ such that $\sigma.Z.\tau$ is well defined, there is some $\upsilon \trianglelefteq \tau$ of length n for which $\mathcal{S} \underline{after}\, \sigma.Z.\upsilon \subseteq S_1$. So

$$
\begin{array}{ll}
& (\mathcal{S} \underline{after}\, \sigma.Z') \underline{after}\, \upsilon \\
\subseteq & \text{Theorem 1, parts 2, 3 and 4} \\
& \mathcal{S} \underline{after}\, \sigma.Z.\upsilon \\
\subseteq & \text{assumption} \\
& S_1 \,.
\end{array}
$$

Hence n' is at most n. ∎

A system that is adaptive to a given external action at one level of abstraction may not be adaptive to it at a more abstract level. However adaptivity is monotone under system refinement:

Theorem 3: If $\mathcal{S}$ is n-Z-adaptive and $\mathcal{S} \sqsubseteq \mathcal{S}'$ then $\mathcal{S}'$ is n'-Z-adaptive for some $n' \leq n$.

The proof follows that of Theorem 2. This result shows that the method of top-down incremental development can be used for adaptive systems.

III. Cluster-based routing

As a typical example of an adaptive self-organising system, we consider the cluster-formation process in a cluster-based routing protocol designed for use in mobile *ad hoc* networks (MANETs) [12], [11], [6].

MANETs have a dynamically changing topology due to the movement of mobile nodes. Hence, message routing protocols need to be able to dynamically discover routes. A *naïve* approach is to use a *flooding* protocol to propagate the routing request to every node in the network. However, this is highly inefficient both in terms of the energy and network bandwidth required, and can lead to the network becoming congested.

For this reason, cluster-based routing protocols have been developed. The idea is to generate a topology based around groups of nodes called clusters. Each cluster has a head node which is *adjacent*, *i.e.*, directly connected, to every other node in the cluster. Only the head nodes deal with routing requests, which significantly reduces the network traffic compared with flooding. The maintenance of clusters, as nodes move within a MANET, itself creates overhead: information about connections between nodes in a distributed setting requires the exchange of messages. Passive clustering approaches, such as that in [6], avoid this extra overhead by piggy-backing a 'hello' message (around 2 bits capturing the node's clustering state) onto MAC (Medium Access Control) layer communications.

Cluster-based routing works by having every non-head node adjacent to a head node. Also, for efficiency, no two head nodes should be adjacent. Those conditions need to be established during cluster-formation.

There are a number of ways that clusters can be established. One approach is to assign the node with the minimum identifier amongst all its neighbours as the head. As pointed out by Gerla *et al.* [6], this approach can lead to inefficiencies in establishing stable clusters, and also requires a *quasi-stationary* assumption that nodes are not mobile during cluster formation and maintenance.

Gerla *et al.* suggest an alternative approach called the "first declaration wins" rule: essentially, the first node in a neighbourhood to declare itself the head becomes the head. Then some means of dealing with contention, *i.e.*, when two nodes simultaneously declare themselves head, is required. But that is solved by standard techniques such as the node with the minimum (or maximum) identifier backing off, or both nodes backing off for random times.

In Section IV, we provide a formal specification of the cluster formation process using a "first declaration wins" strategy. Our specification is abstract, ignoring details such as contention handling. In Sections V and VI, we illustrate how we can reason about adaptivity of the specified system using our definition of Section II.

IV. A formal model of cluster formation

Our approach is independent of any particular formalism. For presentation purposes, however, we employ the Z notation [17]. We begin our formalisation with the definition of a type *Node* denoting the set of all nodes in a network, and a type *State* denoting the possible states of a given node.

$$
\begin{array}{l}
[\mathit{Node}] \\
\mathit{State} ::= \mathit{undecided} \mid \mathit{member} \mid \mathit{head}
\end{array}
$$

The state of a network is captured by the states and connectivity of nodes. A node cannot be connected to itself, and must be transitively connected to all other nodes. These constraints are captured as invariants in the state schema *Network*. (Here $^+$ denotes the transitive closure).

Network

$$state : Node \to State$$
$$conn : Node \leftrightarrow Node$$

$$\forall n : Node \bullet (n,n) \notin conn \quad \forall m : Node \bullet n \neq m \Rightarrow (n,m) \in conn^+$$

For a fixed network, *i.e.*, one in which nodes are not mobile and in which connectivity is never changed, there would be two additional invariants. Firstly, there are no adjacent heads.

$$I_1 \;\widehat{=}\; \nexists n, m : Node \bullet state(n) = head \land state(m) = head \land (n,m) \in conn$$

Secondly, a member node is always connected to a head and only member nodes are connected to heads.

$$I_2 \;\widehat{=}\; \forall n : Node \bullet state(n) = member \Leftrightarrow (\exists m : Node \bullet (n,m) \in conn \land state(m) = head)$$

These conditions, although necessary for the intended operation of the network, are not invariants when we allow mobility (as discussed in Sections V and VI).

Initially, all nodes of a network are in *undecided* state and the connection topology is arbitrary.

Init

$$Network$$

$$\forall n : Node \bullet state(n) = undecided$$

This schema establishes the additional invariants for fixed networks which are trivially satisfied when all nodes are in the *undecided* state. That is,

$$Init \Rightarrow I_1 \land I_2 \,. \tag{2}$$

We model the cluster formation algorithm abstractly by a single action which may occur repeatedly for as long as it remains enabled. This action models a node n in *undecided* state broadcasting that it changes to *head* state. On receiving such a broadcast, all neighbouring nodes enter *member* state. This captures the "first declaration wins" strategy.

DeclareHead

$$\Delta Network$$
$$n : Node$$

$$state(n) = undecided$$
$$state'(n) = head$$
$$\forall m : Node \bullet (n,m) \in conn \Rightarrow state'(m) = member \quad (n,m) \notin conn \Rightarrow state'(m) = state(m)$$
$$conn' = conn$$

This action maintains the additional invariants required for a fixed network. If I_1 is true when *DeclareHead* occurs then there are no adjacent heads. The action makes node n a head, but also makes all of its neighbouring nodes members. Hence, I_1 continues to hold.

Similarly, if I_2 is true when *DeclareHead* occurs then (a) all nodes connected to a head are members, and (b) all member nodes are connected to a head. Hence n, which is undecided, is not connected to a head (only to nodes which are undecided or members). The action makes n a head and all of its neighbours members. Hence, condition (a) continues to hold. Also, since no neighbour of n is a head before the action, the action does not change the state of an existing head and hence condition (b) also continues to hold.

That is, for any $i : \mathbb{N}$ we have

$$Init \underline{after}\, DeclareHead^i \Rightarrow I_1 \land I_2 \,. \tag{3}$$

For the cluster-based routing protocol, we require that eventually every non-head node is adjacent to a head.

$$I_3 \;\widehat{=}\; \forall n : Node \bullet state(n) \neq head \Rightarrow (\exists m : Node \bullet (n,m) \in conn \land state(m) = head)$$

This is not an invariant of the specification since, for example, it is not true initially. However, we can show that it is eventually true for a finite, fixed network as follows.

The number of *undecided* nodes are decreased by at least one (the broadcasting node n) every time *DeclareHead* occurs. Since there are no other actions, for a finite network, this number eventually reaches zero. Hence, given that I_2 holds, we can deduce that I_3 will eventually be true.

For a network with more than one node, since each head node must have at least one member (since the network is transitively connected), a maximum of $N-1$ occurrences of *DeclareHead* are required, where N is the number of nodes in the network. Hence, for some $i \leqslant N-1$,

$$Init \underline{after}\, DeclareHead^i \Rightarrow I_3 \tag{4}$$

This is an example of *self configuration*, *i.e.*, where a system configures itself for a given purpose.

V. Adapting to new links

In a mobile *ad hoc* network, connections between nodes are formed and lost as the nodes move. In this section, we will consider the ability of the system specified in Section III to adapt to new connections being formed. In Section VI, we will consider the case of connections being lost.

A new link being formed between two nodes that are not both heads maintains the invariants I_1 and I_2 so it suffices to consider an external action that links only heads.

NewLink

$\Delta Network$
$n, m : Node$

$state(n) = state(m) = head$
$conn' = conn \cup \{(n, m)\}$
$state' = state$

If we add this action to the specification of Section III, I_1 is no longer invariant: a node that is a head can be connected to another head. Since head nodes never change their state, the specification does not restore the condition. Hence, we deduce that the system is not adaptive to the action *NewLink*.

To make the system adaptive, we could add an action *Hello* to the specification which models head nodes periodically broadcasting a message to their neighbours. All neighbours on receiving such a broadcast, change their state to *member*. Hence, if two head nodes become connected, the first to broadcast its "hello" message remains head, the other becomes a (non-head) member node.

Hello

$\Delta Network$
$n : Node$

$state(n) = head$
$\forall m : Node \bullet$
$\quad (n, m) \in conn \Rightarrow state'(m) = member$
$\quad (n, m) \notin conn \Rightarrow state'(m) = state(m)$
$conn' = conn$

Given this new action, we can show that the system restores I_1. The number of interconnected head nodes decreases by at least two every time a "hello" message is sent by such a node. Hence, after a single occurrence of *NewLink*, a *Hello* from one of the interconnected heads will reduce the number of interconnected head nodes to zero. So for all $i \geqslant 2$ and $n_1, n_2 : Node$

$$Init \underline{after} \rho \Rightarrow I_1 \tag{5}$$

where ρ is the catenated sequence

$$DeclareHead^i.NewLink[n1, n2/n, m].\tau$$

and τ is any sequence of actions containing either $Hello[n1/n]$ or $Hello[n2/n]$ and not containing *NewLink*.

Now consider I_2. It is not affected by the addition of the *NewLink* action. However, it is affected by the action *Hello* which can cause a head node to become a member node, leaving some member nodes without a head. For example in Figure 1, node 2 becomes a member after receiving a "hello" message from node 3. Node 1 is left without an adjacent head.

The system as specified has no means of restoring I_2. Hence, we add a further action. The action *Timeout* models a member node timing out after not having received a "hello" message from its head for a specified amount of time. The action abstracts time and requires only that the member node no longer has a head.

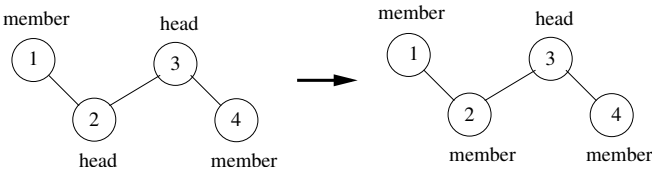

Figure 1.

Timeout

$\Delta Network$
$n : Node$

$state(n) = member$
$\forall m : Node \bullet$
$\quad (n, m) \in conn \Rightarrow state(m) \neq head$
$state'(n) = undecided$
$\forall m : Node \bullet$
$\quad m \neq n \Rightarrow state'(m) = state(m)$
$conn' = conn$

Given the new action, we can show that I_2, although no longer an invariant, is restored by application of *Timeout*. The number of member nodes without an adjacent head node decreases by one (the node n) every time *Timeout* occurs. Hence, if after a single occurrence of *NewLink* followed by a *Hello* message, there are j member nodes without a head, I_2 is restored after j occurrences of *Timeout*. The node which remains head (node 3 in Figure 1) has at least two members after the action (its original member(s) and the node which changes from a head to a member). Therefore, the maximum number of *Timeout* actions required is $N - 3$, where N is the number of nodes in the network. That is, for all $i \geqslant 2$ and $n1, n2 : Node$, there is some $j \leqslant N - 3$ such that

$$Init \underline{after} \rho \Rightarrow I_2 \tag{6}$$

where catenation ρ is $DeclareHead^i.NewLink[n1, n2/n, m].\tau$ and τ is any sequence of actions containing the subsequence $Hello[n1/n].Timeout^j$ or $Hello[n2/n].Timeout^j$.

Since both *Hello* and *Timeout* occur within a finite time of being enabled (this is only stated informally here, but could be formalised), any sequence of actions following a single occurrence of a *NewLink* action will contain this subsequence. Hence, the system is j+1-*NewLink*-adaptive.

Once I_2 is restored it is not violated by any of the system actions (unless *NewLink* occurs again). Hence, following the reasoning in the previous section, the system self configures restoring I_3 after a number of *DeclareHead* actions. In the worst case, all of the member nodes which timed out become heads. That is, for all $i \geqslant 2$ and $n1, n2 : Node$, there is some $j \leqslant N - 3$ and some $k \leqslant j$ such that

$$Init \underline{after}\ \rho.DeclareHead^k \Rightarrow I_3 \tag{7}$$

where catenation ρ is defined as above.

VI. Adapting to broken links

A link being broken is modelled by the following action.

BreakLink

$\Delta Network$
$n, m : Node$

$(n, m) \in conn$
$conn' = conn \setminus \{(n, m)\}$
$state' = state$

I_1 is not affected by this action. However, I_2 may no longer hold since the connection between a member node and its only head may be removed.

This situation is analogous to that in Figure 1 and I_2, and hence I_3, will be restored if we add the actions *Hello* and *Timeout*. The reasoning follows that in the previous section.

VII. Conclusion and further work

In this paper, we have formally defined adaptivity to a given external action Z in a manner which allows us to quantify the system's adaptive response. Our formalisation makes proofs of adaptivity amenable to Formal Methods, in particular to incremental development. We would also like to be able to use abstraction to perform model checking [2, Ch. 13]. Determining its relationship to our definition of adaptivity is an important area of future work.

Our formal definition is not limited to a particular specification paradigm. However to make it more suited to multi-agent systems, we would like to consider it in the context of action refinement which would allow us to capture the concurrent behaviour of such systems. In particular, this would allow us to reason more explicitly about notions such as contention between agents. Action systems [1] is one formalism that supports action refinement and we will consider using (a variant of) it in future work.

Acknowledgements

The first author acknowledges support from the Macao Science and Technology Development Fund under the PEARL project, grant number 041/2007/A3 and the second author acknowledges support from the Macao Science and Technology Development Fund under the EAE project, grant number 072/2009/A3.

References

[1] R.J.R. Back and R. Kurki-Suonio. Decentralization of process nets with centralized control. *Distributed Computing*, 3(2):73–87, 1989.

[2] E.M. Clarke, O. Grumberg, and D.A. Peled. *Model Checking*. MIT Press, 1999.

[3] W.-P. de Roever and K. Engelhardt. *Data refinement: model-oriented proof methods and their comparison*. CUP, 1998.

[4] E.W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, **17**:643–644, 1974.

[5] I. Georgiadis, J. Magee, and J. Kramer. Self-organising software architectures for distributed systems. In *1st Workshop on Self-healing Systems (WOSS '02)*, pages 33–38, 2002.

[6] M. Gerla, T.J. Kwon, and G. Pei. On demand routing in large ad hoc wireless networks with passive clustering. In *Proceedings of IEEE Wireless Communications and Networking Conference (WCNC 2000)*, volume **1**, pages 100–105, 2000.

[7] M. Güdemann, F. Nafz, F. Ortmeier, H. Seebach, and W. Reif. A specification and construction paradigm for organic computing systems. In *Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2008)*, pages 233–242. IEEE Computer Society Press, 2008.

[8] T. Herman. Probabilistic self-stabilization. *Information Processing Letters*, **35**:63–67, 1990.

[9] T. Herman. A self-stabilization bibliography. Available at www.cs.uiowa.edu/ftp/selfstab/bibliography/, 1992.

[10] J.M. Hoekstra. *Designing for Safety: the Free Flight Air Traffic Management Concept*. PhD thesis, Technical University of Delft, 2001.

[11] M. Jiang, J. Li, and Y.C. Tay. Cluster Based Routing Protocol (CBRP). Technical report, Internet draft, National University of Singapore, 1999.

[12] C.R. Lin and M. Gerla. Adaptive clustering for mobile wireless networks. *IEEE Journal on Selected Areas in Communications*, **15**(7):1265–1275, 1997.

[13] A.K. McIver and C.C. Morgan. An elementary proof that Herman's Ring is $\Theta(N^2)$. *Information Processing Letters*, **94**:79–84, 2005.

[14] F. Nafz, F. Ortmeier, H. Seebach. J.-P. Steghöfer, and W. Reif. A universal self-organisation mechanism for role-based organic computing systems. In *International Conference on Autonomic and Trusted Computing (ATC'09)*, volume **5586** of *Lecture Notes in Computer Science*, pages 17–31. Springer-Verlag, 2009.

[15] M. Schneider. Self stabilization. *ACM Computing Surveys*, **25**:45–67, 1993.

[16] G. Smith and J. W. Sanders. Formal development of self-organising systems. In *International Conference on Autonomic and Trusted Computing (ATC'09)*, volume 5586 of *Lecture Notes in Computer Science*, pages 90–104. Springer-Verlag, 2009.

[17] J.M. Spivey. *The Z Notation: A Reference Manual*. Prentice Hall, 2nd edition, 1992.

[18] R.P. Würtz, editor. *Organic Computing: Understanding Complex Systems*. Springer-Verlag, 2008.