Assuring adaptive behaviour in self-organising systems

J. W. Sanders and Graeme Smith

July 2010
UNU-IIST and UNU-IIST Reports

UNU-IIST (United Nations University International Institute for Software Technology) is a Research and Training Centre of the United Nations University (UNU). It is based in Macao, and was founded in 1991. It started operations in July 1992. UNU-IIST is jointly funded by the government of Macao and the governments of the People’s Republic of China and Portugal through a contribution to the UNU Endowment Fund. As well as providing two-thirds of the endowment fund, the Macao authorities also supply UNU-IIST with its office premises and furniture and subsidise fellow accommodation.

The mission of UNU-IIST is to assist developing countries in the application and development of software technology.

UNU-IIST contributes through its programmatic activities:

1. Advanced development projects, in which software techniques supported by tools are applied,
2. Research projects, in which new techniques for software development are investigated,
3. Curriculum development projects, in which courses of software technology for universities in developing countries are developed,
4. University development projects, which complement the curriculum development projects by aiming to strengthen all aspects of computer science teaching in universities in developing countries,
5. Schools and Courses, which typically teach advanced software development techniques,
6. Events, in which conferences and workshops are organised or supported by UNU-IIST, and
7. Dissemination, in which UNU-IIST regularly distributes to developing countries information on international progress of software technology.

Fellows, who are young scientists and engineers from developing countries, are invited to actively participate in all these projects. By doing the projects they are trained.

At present, the technical focus of UNU-IIST is on formal methods for software development. UNU-IIST is an internationally recognised center in the area of formal methods. However, no software technique is universally applicable. We are prepared to choose complementary techniques for our projects, if necessary.

UNU-IIST produces a report series. Reports are either Research [R], Technical [T], Compendia [C] or Administrative [A]. They are records of UNU-IIST activities and research and development achievements. Many of the reports are also published in conference proceedings and journals.

Please write to UNU-IIST at P.O. Box 3058, Macao or visit UNU-IIST’s home page: http://www.iist.unu.edu, if you would like to know more about UNU-IIST and its report series.

Peter Haddawy, Director
Assuring adaptive behaviour in self-organising systems

J. W. Sanders and Graeme Smith

Abstract

The important notion of adaptivity of a distributed information system is formalised, following Dijkstra's idea of self stabilisation. Moreover the formalisation quantifies the extent to which a system adapts, enabling degrees of adaptivity to be specified and hence assured of an implementation. The ideas are expressed without commitment to any particular semantic formalism and demonstrated on the cluster-based routing of messages in mobile ad hoc networks.
Jeff Sanders is Principal Research Fellow at UNU-IIST. His interests lie largely in Formal Methods.

Graeme Smith is a senior lecturer in the School of Information Technology and Electrical Engineering at the University of Queensland, Australia. He is well known for his work on Formal Methods, Z and object orientation.

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

1 Introduction 7

2 A formal notion of adaptivity 8
   2.1 Systems 8
   2.2 Adaptivity 9
   2.3 Examples 10
   2.4 Results 10

3 Cluster-based routing 11

4 A formal model of cluster formation 12

5 Adapting to new links 14

6 Adapting to broken links 17

7 Conclusion and further work 17
1 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 with respect to the number of actions it takes for the system to restore the constraints.

Similar observations have been made before. Dijkstra [3] 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 [14] and Herman’s bibliography [8].

Güdemann et al. [6] present an architecture for Organic Computing [16] 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 [13].

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. [4] 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. Like our work, though, it is interested in only completely decentralised systems.

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. For that, the environmental ‘perturbation’ must be made precise and the rate at which the system adapts must be quantified. In Section 2 we provide a formal definition of adaptivity based on those ideas. In Section 3, we introduce a case study based on a passive clustering algorithm for a mobile ad-hoc network [5]. We formally derive results concerning the ability of the algorithm to adapt to new connections being formed, and existing connections being broken, in Sections 5 and 6 respectively. We conclude with a discussion of related and future work in Section 7.

2 A formal notion of adaptivity

2.1 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 \( S = (S, S_0, S_1, A) \) in which: \( S \) is the type of states; \( S_0 \subseteq S \) are the initial states; \( S_1 \subseteq S \) are the states satisfying the system invariant, assumed to be established by the initial states: \( S_1 \supseteq S_0 \); 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. That setting is not in general appropriate for an analysis of contention between concurrent actions, a point which is discussed further in Section 7.

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
A formal notion of adaptivity

9

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 \( \alpha' \) is less enabled but more terminating and more deterministic.

In this section no particular formalism is presumed for describing \( S \). It is thus appropriate to capture (linear) system semantics by sequences of actions and the state transitions they achieve (as in say [14]). If \( E \subseteq S \) and \( \sigma \) is a well-defined sequence of actions (i.e., each enabled and terminating) \( E \) after \( \sigma \) denotes the states obtainable from those in \( E \) by executing the sequence of actions \( \sigma \). By \( S \) after \( \sigma \) is meant \( \text{Init after } \sigma \). Throughout we assume well-defined sequences.

If \( S \) and \( S' \) are systems, recall that \( S \) is said to be refined by \( S' \), \( S \sqsubseteq S' \), if the behaviours of \( S' \) are included in those of \( S \). This includes data refinement but not action refinement (in which the actions of the concrete system may be more numerous). For details of refinement of systems and actions we refer to de Roever and Engelhardt [2].

The after operation is monotonic in several senses.

**Theorem 1** Let \( S, S' \) be systems, let \( E, F \subseteq S \), let \( \sigma, \tau : A^* \) and let \( \alpha, \alpha' \) be actions. The catenation of \( \sigma \) and \( \tau \) is written \( \sigma \cdot \tau \). Then

1. If \( \sigma : A^* \) then \( S \) after \( \sigma \) \( \subseteq S_1 \)
2. \( (E \) after \( \sigma \) after \( \tau = E \) after \( \sigma \cdot \tau \)
3. If \( \alpha \sqsubseteq \alpha' \) then \( E \) after \( \alpha \supseteq E \) after \( \alpha' \)
4. If \( E \subseteq F \) then \( E \) after \( \sigma \subseteq F \) after \( \sigma \)
5. If \( S \sqsubseteq S' \) then \( S \) after \( \sigma \supseteq S' \) after \( \sigma \).

### 2.2 Adaptivity

Environmental duress on a system \( S = (S, S_0, S_1, A) \) is captured (and quantified) 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 \( S \) 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 \( \sqsubseteq \) for subsequence, it is expressed

\[
\forall \sigma : A^* \cdot \forall \tau : A^\omega \cdot \sigma.Z.\tau \text{ is well defined } \Rightarrow \exists \upsilon : A^* \cdot \upsilon \sqsubseteq \tau \wedge S \text{ after } \sigma.Z.\upsilon \sqsubseteq S_1.
\]
When we wish to focus on the number of actions required to restore the invariant, we say instead that the system $S$ is $n$-Z-adaptive, where $n$ is the least number of system actions required to restore the invariant $S_i$, i.e., the length of the shortest such $\upsilon$. In particular, if $S$ is 0-Z-adaptive then the external action $Z$ preserves the system invariant $S_1$, and so $S$ need do nothing to 'recover from $Z$'. The larger the value of $n$, the longer (in terms of actions) it takes $S$ to recover from $Z$ (measured with respect to a linearised semantics).

The maximum length of $\upsilon$, when it is well defined, is important in bounding that ‘recovery time’. In general it is not well defined; however in certain fortunate circumstances, one of which occurs in Section 4, an upper bound on the length of $\upsilon$ is possible.

### 2.3 Examples

A comprehensive, rigorous example is provided in Section 3. 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 3.

**Free flight** [9] 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** [3] 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 [7, 12] that it adapts quickly, with respect to a probabilistic semantic model: with $n$ processors it adapts on average in $\Theta(n^2)$ steps.

### 2.4 Results

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

**Theorem 2** If $Z \subseteq Z'$ and $S$ is $n$-Z-adaptive then $S$ is $n'$-Z'-adaptive for some $n' \leq n$. 
Cluster-based routing

Proof
By definition (1), for any \(\sigma : A^*\) and \(\tau : A^0\) for which \(\sigma . Z . \tau\) is well defined, there is some \(\nu \subseteq \tau\) of length \(n\) for which \(S_{after} \sigma . Z . \nu \subseteq S_1\). So

\[
(S_{after} \sigma . Z')_{after} \nu \\
\subseteq \text{Theorem 1, parts 2, 3 and 4}
\]

\[
S_{after} \sigma . Z . \nu \\
\subseteq \text{assumption}
\]

\[
S_1.
\]

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 \(S\) is \(n\)-\(Z\)-adaptive and \(S \subseteq S'\) then \(S'\) is \(n\)-\(Z\)-adaptive.

This result shows that the method of top-down incremental development can be used for adaptive systems.

3 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) [11, 10, 5].

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
A formal model of cluster formation

[5], 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. [5], 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 4, 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 5 and 6, we illustrate how we can reason about adaptivity of the specified system using our definition of Section 2.

4 A formal model of cluster formation

Our approach is independent of any particular formalism. For presentation purposes, however, we employ the Z notation [15]. 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.

\[
\begin{align*}
\text{State} & := \text{undecided} \mid \text{member} \mid \text{head}
\end{align*}
\]

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 \(\ast\) denotes the reflective transitive closure.)

\[
\begin{align*}
\text{Network} & := \text{Node} \rightharpoonup \text{State} \\
\text{conn} & := \text{Node} \rightharpoonup \text{Node}
\end{align*}
\]

\[
\forall n : \text{Node} \bullet \\
\quad (n, n) \notin \text{conn} \\
\forall m : \text{node} \bullet (n, m) \in \text{conn}^\ast
\]
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.

\[ I_1 \triangleq \exists n, m : \text{Node} \bullet state(n) = \text{head} \land state(m) = \text{head} \land (n, m) \in \text{conn} \]

Secondly, a member node is always connected to a head.

\[ I_2 \triangleq \forall n : \text{Node} \bullet state(n) = \text{member} \iff \exists m : \text{Node} \bullet (n, m) \in \text{conn} \land state(m) = \text{head} \]

Those conditions, although necessary for the intended operation of the network, are not invariants when we allow mobility (as discussed in Sections 5 and 6).

Initially, all nodes of a network are in \textit{undecided} state and the connection topology is arbitrary.

\[
\begin{align*}
\text{Init} & \\
\text{Network} & \\
\forall n : \text{Node} \bullet state(n) = \text{undecided}
\end{align*}
\]

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

\[ \text{Init} \Rightarrow I_1 \land I_2. \]  

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 \textit{undecided} state broadcasting that it changes to \textit{head} state. On receiving such a broadcast, all neighbouring nodes enter \textit{member} state. This captures the “first declaration wins” strategy.

\[
\begin{align*}
\text{DeclareHead} & \\
\Delta \text{Network} & \\
n : \text{Node} & \\
state(n) = \text{undecided} & \\
state'(n) = \text{head} & \\
\forall m : \text{Node} \bullet & \\
(n, m) \in \text{conn} \Rightarrow state'(m) = \text{member} & \\
(n, m) \notin \text{conn} \Rightarrow state'(m) = state(m) & \\
\text{connected}' & = \text{connected}
\end{align*}
\]

This action maintains the additional invariants required for a fixed network. If \( I_1 \) is true when \texttt{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
\[
\text{Init after DeclareHead}^i \Rightarrow I_1 \land I_2.
\] (3)

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

\[
I_3 \equiv \forall n : \text{Node} \mid \text{state}(n) \neq \text{head} \bullet \exists m : \text{Node} \bullet (n,m) \in \text{conn} \land \text{state}(m) = \text{head}
\]

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.

In fact, 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 $i \leq N - 1$,
\[
\text{Init after DeclareHead}^i \Rightarrow I_3
\] (4)

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

5  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 3 to adapt to new connections being formed. In Section 6, 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 $I_1, I_2$ so it suffices to consider an external action that links only heads.
Adapting to new links

If we add this action to the specification of Section 3, $I_1$ 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 $\text{NewLink}$.

To make the system adaptive, we could add an action $\text{Hello}$ 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 $\text{member}$. Hence, if two head nodes become connected, the first to broadcast its “hello” message remains head, the other becomes a (non-head) member node.

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 $\text{Hello}$ occurs. Hence, after a single occurrence of $\text{NewLink}$, a $\text{Hello}$ from one of the interconnected heads will reduce the number of interconnected head nodes to zero. So for all $i \geq 2$ and $n_1, n_2 : \text{Node}$

$$\text{Init after } \rho \Rightarrow I_1 \quad (5)$$

where $\rho$ is the catenated sequence

$\text{DeclareHead}^i \cdot \text{NewLink}[n1, n2 / n, m], \tau$

and $\tau$ is any sequence of actions containing $\text{Hello}[n1 / n]$ or $\text{Hello}[n2 / n]$.

Now consider $I_2$. It is not affected by the addition of the $\text{NewLink}$ action. However, it is affected by the action $\text{Hello}$ 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.
Adapting to new links

Figure 1: A MANET in which nodes 2 and 3 are originally adjacent and so, by both being heads, inviolate the invariant $I_1$. After a Hello action $I_1$ is restored by node 2 changing state.

The system as specified has no means of restoring $I_2$. Hence, we add a further action. The action $\text{Timeout}$ 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.

$$
\begin{align*}
\text{Timeout} \\
\Delta \text{Network} \\
\text{n : Node} \\
\text{state(n) = member} \\
\forall m : \text{Node} \bullet (n,m) \in \text{connected} \Rightarrow \text{state(m) \neq head} \\
\text{state'}(n) = \text{undecided} \\
\forall m : \text{Node} \bullet m \neq n \Rightarrow \text{state'}(m) = \text{state}(m) \\
\text{connected'} = \text{connected}
\end{align*}
$$

Given the new action, we can show that $I_2$, although no longer an invariant, is restored by application of $\text{Timeout}$. The number of member nodes without an adjacent head node decreases by one (the node $n$) every time $\text{Timeout}$ occurs. Hence, if after a single occurrence of $\text{NewLink}$ followed by a Hello message, there are $j$ member nodes without a head, $I_2$ is restored after $j$ occurrences of $\text{Timeout}$. 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 $\text{Timeout}$ actions required is $N - 3$, where $N$ is the number of nodes in the network. That is, for all $i \geq 2$ and $n_1,n_2 : \text{Node}$, there is some $j \leq N - 3$ such that

$$
\text{Init after } \varphi \Rightarrow I_2
$$

where catenation $\varphi$ is $\text{DeclareHead'}'.\text{NewLink}[n_1,n_2/n,m].\tau$ and $\tau$ is any sequence of actions containing the subsequence $\text{Hello}[n_1/n].\text{Timeout}^i$ or $\text{Hello}[n_2/n].\text{Timeout}^i$.

Since both $\text{Hello}$ and $\text{Timeout}$ 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 $\text{NewLink}$ action will contain this subsequence. Hence, the system is $j+1-\text{NewLink}$-adaptive.

Once $I_2$ is restored it is not violated by any of the system actions (unless $\text{NewLink}$ 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 \( i \geq 2 \) and \( n1, n2 : \text{Node} \), there is some \( j \leq 3 \) and some \( k \leq j \) such that

\[
\text{Init after } \rho.\text{DeclareHead}^k \Rightarrow I_3
\]  

(7)

where catenation \( \rho \) is defined as above.

6 Adapting to broken links

A link being broken is modelled by the following action.

\[
\begin{array}{l}
\text{BreakLink} \\
\Delta \text{Network} \\
\quad n, m : \text{Node} \\
\quad (n, m) \in \text{connected} \\
\quad \text{connected}' = \text{connected} \setminus \{(n, m)\} \\
\quad \text{state}' = \text{state}
\end{array}
\]

\( I_1 \) 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.

7 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 action refinement in the incremental approach to refinement to develop adaptive systems, and to use abstraction to perform model checking [1, Ch. 13]. Developing such theories 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 partial-order semantics which would allow us to capture the truly concurrent behaviour of such systems. In particular, this would allow us to reason more explicitly about notions such as contention between agents.
References


