# Designing Adaptive Systems Using Teleo-Reactive Agents

Graeme Smith[1(✉)], J.W. Sanders[2,3], and Kirsten Winter[1]

[1] School of Information Technology and Electrical Engineering, The University of Queensland, Brisbane, Australia
smith@itee.uq.edu.au
[2] African Institute for Mathematical Sciences (AIMS), Cape Town, South Africa
[3] Department of Mathematical Sciences, Stellenbosch University, Stellenbosch, South Africa

**Abstract.** Although adaptivity is a central feature of agents and multi-agent systems (MAS), there is no precise definition of it in the literature. What does it mean for an agent or for a MAS to be adaptive? How can we reason about and measure the ability of agents and MAS to adapt? How can we systematically design adaptive systems? In this paper, we provide a formal definition of adaptivity, and a framework for designing adaptive systems aimed at addressing these issues.

The definition of adaptivity, based on Dijkstra's notion of self stabilisation, is independent of any particular mechanism for ensuring adaptivity, and any particular specification notation. The framework for designing adaptive systems is similarly independent of both implementation mechanisms and specification notation. It is based on the paradigm of teleo-reactive agents proposed by Nilsson: a paradigm in which agents move towards their goal in the presence of a continually changing environment.

## 1 Introduction

Adaptivity is central to the functionality of many multi-agents systems (MAS) [36]. Agents adapt gradually to their environment using, for example, machine learning techniques [20], and the distributed nature of MAS is exploited to make them robust against both external disturbances and agent failures. The inherent complexity of such systems has led to much interest in systematic approaches to their development in the area of agent-oriented software engineering (AOSE) [37], and growing interest in the use of formal methods to complement assurance approaches based primarily on simulation [6,17].

These development and assurance approaches rely on precise behavioural specifications of the agents or MAS being engineered. Such specifications can be developed in an *ad-hoc*, system-by-system fashion, but a more systematic approach is hindered by the lack of a precise definition of adaptivity and a general framework for specifying adaptive behaviour. As Mohyeldin *et al.* [21] state


R. Kowalczyk and N.T. Nguyen (Eds.): TCCI XVI, LNCS 8780, pp. 34–61, 2014.
DOI: 10.1007/978-3-662-44871-7_2

> "Still open is a semantically well defined process to design adaptive systems, while concentrating on the adaptive behaviour rather than discussing implementation details."

In this paper, we address this issue by

1. providing a formal definition of adaptivity which is independent of any particular adaptivity mechanism and general enough to use with any specification notation, and
2. based on this definition, a framework for specifying the behavioural requirements of adaptive agents and MAS in a systematic way.

We begin in Sect. 2 by motivating our definition of adaptivity (which is based on our previous work [26,29]) with respect to representative informal notions of adaptivity arising in the literature. We then provide a formal model of MAS in Sect. 3 as a basis for formalising our adaptivity definition in Sect. 4. In Sect. 5, we introduce a specification framework for modelling adaptive systems based on Nilsson's teleo-reactive agents paradigm [23,24]. The approach is applied to a case study in Sect. 6 before we conclude in Sect. 7.

## 2 What is Adaptivity?

Various informal notions of adaptivity can be found in the literature. Below are some representative examples.

- *Adaptivity should allow change in system functionality* [13]. The view is that, by adapting to environmental change, a system or agent may offer new operations on new states.
- *Adaptivity should include self-organisation* [12,16]. Systems should be able to reconfigure to adjust for changes in the environment or agent capabilities.
- *Adaptivity should include self-optimisation* [3]. In response to a change in externally-set parameters (*i.e.*, global variables) the agents are able spontaneously and autonomously to perform calculations (viewed as optimising certain local variables) which result in the system returning to a desired state.
- *Adaptivity should include (machine) learning* [20,33]. It should, for example, enable improved response to change so that when confronted with the same situation in the future, the system or agent adapts more quickly and efficiently.

Our goal is to find a general definition which covers each of the notions above.

The most fundamental feature of an adaptive system is its ability to change behaviour (*i.e.*, functionality) in response to environmental change. The prospect of changing behaviour at first conjures up visions of systems which are somehow more advanced than standard computer programs. However, changing behaviour is illusory since a system, when viewed at a certain level of abstraction as a simple state machine, does not actually change what it is capable of doing. As shown in [13], it simply moves to a new state in which different actions and

environmental interactions are possible. This is true even of approaches to evolutionary computing [10] and machine learning [20]: underlying such systems is just a computer program. Given this observation, what distinguishes an adaptive system from a merely reactive one? Indeed, a number of approaches for adaptive systems seem to support designs which respond to environmental change without further consideration for what makes those designs adaptive [4,5].

To answer this question, we appeal to the notion of *legitimate states* of a system introduced by Dijkstra [8]. In this work, 'legitimacy' is defined by a state invariant capturing those states in which the system behaves as intended. A typical reactive system would operate only in such legitimate states changing its behaviour to reflect environmental interactions and, importantly, always operating as intended.

Dijkstra's paper is concerned however with self-stabilisation of distributed systems. His examples consist of token ring networks in which a legitimate state is one in which there is exactly one token present. He presents several algorithms which, given an arbitrary number of tokens initially, end up in a legitimate state.

An important feature of Dijkstra's self-stabilising networks is that even if they start in states which are not legitimate states, they are guaranteed to reach legitimate states after a finite number of system actions. A typical reactive system may not operate at all when not in a legitimate state. Also, it is not designed to perform actions which allow it to reach a legitimate state. An adaptive system, on the other hand, is required to be a reactive system which, like Dijkstra's self-stabilising token rings, can reach legitimate states from illegitimate ones.

In other words, an adaptive system is one which, when placed in a particular environment, has a defined set of legitimate states and when in an illegitimate state reaches a legitimate state again. Indeed, the period before the system reaches the legitimate state is the time when the system is adapting.

Following Dijkstra's definition, we do not allow a system in a legitimate state to enter an illegitimate state of its own accord. That is, defined transitions of the system from legitimate states enter only other legitimate states. A system is placed in an illegitimate state by an *external* action, *i.e.*, an action that is not regarded as part of the system's specification. This action may represent a change in the environment due to an unforeseen disturbance, or the passing of a threshold point in an environment that is gradually changing over time. Alternatively, an external action may represent a change to the system itself. In the case where the system is an agent, this may be caused, for example, by the action of a software virus changing internal data. In the case where the system is a MAS, it may be caused by the failure of a component agent.

In each case we can reason about adaptivity as the ability to 'recover' from the external event, *i.e.*, the ability of the system to reach a legitimate state. Since systems will, in general, be adaptive to only a subset of all possible external actions, we qualify our definition of adaptivity with respect to a specific external action. Furthermore, we quantify adaptivity with respect to the number of actions required for the system to adapt. At a given level of abstraction, this provides us with a metric for comparing different adaptivity mechanisms.

Dijkstra's systems are *closed* in the sense that they do not interact with an external environment. The notion of legitimate states must for our purposes be extended to *open* systems by considering the state to include both that of the system and its environment. Then our definition of adaptivity covers each of the informal definitions above.

- *Adaptivity should allow change in system functionality.* Since the environment state is part of each legitimate state, a change in the environment will result in a change in the system states which constitute a legitimate state. Moving to such a system state will result in a different behaviour (and hence functionality).
- *Adaptivity should include self-organisation.* Self-organisation of a system can be seen as moving towards a legitimate state. Indeed, the approaches of both Güdemann *et al.* [16] and Georgiadis *et al.* [12] are based on satisfying certain system constraints, or invariants. Using the terminology of complex systems, self-organisation may be viewed as autonomous convergence to attracting states [25]. Under our definition the attracting states are the legitimate states.
- *Adaptivity should include self-optimisation.* With respect to self-optimisation, our definition would identify optimal states with legitimate states. While this shows the applicability of our definition in theory, it may not always be possible to specify the optimal states in practice.
- *Adaptivity should include (machine) learning.* In supervised machine learning (positive and negative) examples of a concept $Q$ are provided to enable subsequent approximate classification of specimens into those satisfying $Q$ and those not satisfying $Q$. The set of legitimate states expresses approximate classification of $Q$ (for example as formalised by Valiant in probably approximately correct (PAC) learnability [35]). Initialisation of the learning protocol is regarded as an external action, and the agent 'learns $Q$' if, and only if, it reaches a legitimate state after such initialisation.
  Our quantification of adaptivity with respect to the number of actions required for the system to adapt enables us to specify improved response to external actions.

## 3 A Formal Model of Multi-Agent Systems

In order to formalise our definition of adaptivity, we begin by providing formal representations of agents and MAS. Since we consider agents as being artifacts that are realised by software, they can – on a low level of abstraction – be represented as labelled transition systems (LTS). An LTS comprises a (possibly infinite) set of states, a (possibly infinite) set of initial states, and a collection of actions which cause (possibly nondeterministic) state transitions. Similar concepts have been used in the agent literature before. For example, formalisms with an underlying transition systems semantics such as Z and Object-Z have been suggested for modelling agents and MAS [9,14]. Also, Hunter and Delgrande [18] use transition systems which they extend with a metric function to capture

"plausibility" amongst belief states. Without loss of generality, we assume such a metric can be encoded in the transition system.

**Definition 1.** *An LTS is a 4-tuple* $S = (Q, I, \Sigma, \delta)$ *where*

- $Q$ *is the (possibly infinite) set of states.*
- $I \subseteq Q$ *is the non-empty set of initial states.*
- $\Sigma$ *is the set of actions (or labels).*
- $\delta \subseteq Q \times \Sigma \times Q$ *is the set of labelled transitions.*

A *behaviour* of an LTS, $S$, is a possibly infinite sequence alternating between states and actions $q_0\ a_1\ q_1\ a_2\ q_2\ \ldots$ where for all $i > 0$, $a_i \in \Sigma$ such that $(q_{i-1}, a_i, q_i) \in \delta$.

Let $\mathcal{B}(S)$ denote the behaviours of $S$ starting from an initial state of $S$, *i.e.*, where $q_0 \in I$, and $\mathcal{B}(S, Q')$ denote behaviour of $S$ starting in a state $q_0 \in Q'$ where $Q' \subseteq Q$. Let $st(b, i)$ denote the $i$th state of a behaviour $b$, and let $act(b, i)$ denote the *ith* action.

To facilitate reasoning about environmental interaction, we use a simple extension of LTS in which actions are partitioned into three sets: *internal* actions, *input* actions (externally observable actions controlled by the environment), and *output* actions (externally observable actions controlled by the component).

Such a partitioning of actions has been proposed for modelling reactive systems. It is central to the *I/O automata* approach of Lynch and Tuttle [19], and *interface automata* of de Alfaro and Henzinger [7]. In each of these approaches, combined automata interact by synchronising on common-named input and output actions. All automata with a given action are involved in each synchronisation on that action.

The main difference between I/O automata and interface automata is that the former are *input-enabled* meaning that input actions can never be refused. This is not the case with interface automata where the restrictions on the type of input actions and when they can occur is used to model assumptions on the system's environment.

An approach similar to interface automata has also been proposed for modelling groupware systems by Ellis [11]. This approach, inspired by Smith's work on collective intelligence in computer-based collaboration [31], has been formalised and further developed by ter Beek *et al.* [34]. The automata are referred to as *component automata*, and component automata which are formed as the composition of other component automata as *team automata*. The major difference with the aforementioned approaches to reactive systems is that in a team automaton not all of the composed automata with a given action need to synchronise on that action. This flexibility has been shown to be well suited to formalising notions of coordination, cooperation and collaboration in a distributed setting [34]. In the remainder of this section, we show how agents and MAS can be modelled using component and team automata.

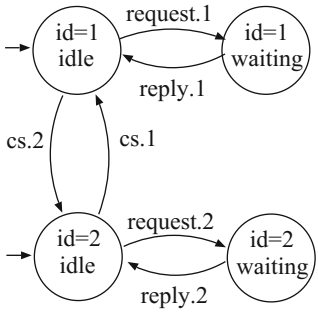


**Fig. 1.** Component automaton of the client agent

### 3.1 Agents as Component Automata

Agents are modelled as component automata [34].

**Definition 2.** *An agent is an LTS,* $A = (Q, I, \Sigma = \Sigma_{int} \cup \Sigma_{inp} \cup \Sigma_{out}, \delta)$*, where* $\Sigma_{int}$*,* $\Sigma_{inp}$ *and* $\Sigma_{out}$ *are pairwise disjoint, and*

- $\Sigma_{int}$ *is the set of internal actions. Such actions are controlled by the agent and are not externally observable.*
- $\Sigma_{inp}$ *is the set of input actions. Such actions are externally observable and are controlled by the agent's environment.*
- $\Sigma_{out}$ *is the set of output actions. Such actions are externally observable and are controlled by the agent.*

*Example 1.* Consider an agent *Client* which is aware of a number of servers in its environment with which it can interact. The state of the client includes the set of server identifiers and the identifier of the server with which it is currently interacting. It has a set of internal actions $cs.id$ which allows it to change the server with which it is interacting to that with identifier $id$ (initially the client is interacting with any server of which the agent is aware), a set of output actions $request.id$ representing a request to the server with identifier $id$, and a set of input actions $reply.id$ representing a reply from the server with identifier $id$.

Assume there are two available servers with identifiers 1 and 2. An LTS that models the client can be depicted as in Fig. 1, where incoming arrows mark initial states. In this model, the client changes server only when it is not awaiting a reply.

To represent this system as a component automaton, we simply partition its actions as follows.

$$\begin{aligned}\Sigma_{int} &= \{cs.i \mid i \in 1\,..\,2\}\\ \Sigma_{inp} &= \{reply.i \mid i \in 1\,..\,2\}\\ \Sigma_{out} &= \{request.i \mid i \in 1\,..\,2\}\end{aligned}$$

◇

Given this partitioning, the fact that the input action *reply.id* occurs only after *request.id*, for $id \in 1..2$, is an assumption that has been made about the client's environment. It is not something the client could itself enforce.

### 3.2 Multi-Agent Systems as Team Automata

When component automata are composed, they potentially synchronise on common-named actions. Hence to prevent unwanted synchronisations, a precondition for composing a group of agents is that no internal action of one agent is present as an action (internal or external) of another agent.

Let $A_i$ denote the agent $(Q_i, I_i, \Sigma_i = \Sigma_{i,int} \cup \Sigma_{i,inp} \cup \Sigma_{i,out}, \delta_i)$, for $i \in 0..n$. A composition of the agents $A_0, \ldots, A_n$ is possible if

$$\forall\, i \in 0..n \bullet (\Sigma_{i,int} \cap \bigcup_{j:0..n \setminus \{i\}} \Sigma_j) = \varnothing. \tag{1}$$

Given such a composable set of agents, a multi-agent system (MAS) is modelled as a special kind of component automaton called a *team automaton* [34]. A state $q$ of the team automaton is a tuple of the possible states of the agents, $q \in \prod_{i:0..n} Q_i$. We let $q_j$, for $j \in 0..n$, denote the $j$th element of the tuple $q$.

**Definition 3.** *A MAS comprising agents $A_0, \ldots, A_n$ is an LTS, $M = (Q, I, \Sigma = \Sigma_{int} \cup \Sigma_{inp} \cup \Sigma_{out}, \delta)$, where $\Sigma_{int}$, $\Sigma_{inp}$ and $\Sigma_{out}$ are pairwise disjoint, and*

- $Q = \prod_{i:0..n} Q_i$.
- $I = \prod_{i:0..n} I_i$.
- $\Sigma_{int} = \bigcup_{i:0..n} \Sigma_{i,int}$.
- $\Sigma_{out} = \bigcup_{i:0..n} \Sigma_{i,out}$.
- $\Sigma_{inp} = (\bigcup_{i:0..n} \Sigma_{i,inp}) \setminus \Sigma_{out}$.
- $\delta \subseteq Q \times \Sigma \times Q$ *such that*
  - *for all $(q, a, q') \in \delta$, there exists a $j \in 0..n$ such that $(q_j, a, q'_j) \in \delta_j$ and for all $i \in 0..n$ with $i \neq j$, $(q_i, a, q'_i) \in \delta_i$ or $q_i = q'_i$*
  - *for all $q, q' \in Q$ and $a \in \Sigma_{int}$, if there exists a $j \in 0..n$ such that $(q_j, a, q'_j) \in \delta_j$, then $(q, a, q') \in \delta$.*

The internal and output actions of $M$ are those of the agents. The input actions are those of the agents which are not also output actions. In the case where an input action of one agent is the same as an output action of another agent, the input is assumed to be caused by the output action and hence is not an input action for the MAS. The fact that the output action is not also removed from the system allows team automata to be further composed with other component or team automata, *e.g.*, to act as the environment of a component in a further composition.

The transitions of $M$ are such that the following hold.

(i) Each transition involves a non-empty subset of agents engaging in the action $a$. The state of each agent not involved in the action remains unchanged.
(ii) There is a MAS transition for each agent transition corresponding to an internal action.

Not all agents with action $a$ need to be involved in a system transition corresponding to $a$. This allows different interaction strategies to be captured [34]. However for consistency, we require that any output action of the system involves at least one agent output action, *i.e.*, there should not be a system action $a$ involving only an agent which has $a$ as an input action when there are other agents which have $a$ as an output action. More formally

$$\forall (q, a, q') \in \delta \bullet a \in \Sigma_{out} \Rightarrow \exists\, j \in 0\,..\,n \bullet a \in \Sigma_{j,out} \wedge (q_j, a, q'_j) \in \delta_j. \quad (2)$$

Furthermore, given a transition $(q, a, q')$ of a MAS such that $q_j = q'_j$ for some $j \in 0..n$, if the $j$th agent has a transition $(q_j, a, q_j)$ then the agent undergoes this action, otherwise (*i.e.*, if $(q_j, a, q_j) \notin \delta_j$) it undergoes no action. This *maximal* interpretation suggested by ter Beek *et al.* [34] removes any ambiguity concerning which agents participate in a particular MAS transition.

*Example 2.* To continue Example 1 above we assume that each server is defined as $Server_i = (Q_i, I_i, \Sigma_{i,int} \cup \Sigma_{i,inp} \cup \Sigma_{i,out}, \delta_i)$ with $\Sigma_{i,int} = \varnothing$, $\Sigma_{i,out} = \{reply.i \mid i \in 1\,..\,2\}$ and $\Sigma_{i,inp} = \{request.i \mid i \in 1\,..\,2\}$. The behaviour of the two servers is modelled abstractly in Fig. 2. (For simplicity, we assume that a server deals with only one client at a time).


**Fig. 2.** Component automata of the server agents

The agents *Client*, $Server_1$ and $Server_2$ can be composed since (1) holds.

Given $Client = (Q_{Client}, I_{Client}, \Sigma_{Client}, \delta_{Client})$, one team automaton that can be composed from *Client* and the servers $Server_1$ and $Server_2$ is $M = (Q, I, \Sigma_{int} \cup \Sigma_{inp} \cup \Sigma_{out}, \delta)$ with

- $Q = Q_{Client} \times \prod_{i:1..2} Q_i$.
- $I = I_{Client} \times \prod_{i:1..2} I_i$.
- $\Sigma_{int} = \{cs.i \mid i \in 1\,..\,2\}$.
- $\Sigma_{inp} = \varnothing$.
- $\Sigma_{out} = \{reply.i, request.i \mid i \in 1\,..\,2\}$.
- $\delta = \{(q, a, q') \in Q \times \Sigma \times Q \mid (q_0, a, q'_0) \in \delta_{Client} \wedge$
  $a \notin \Sigma_{int} \Rightarrow (\exists\, i \in 1\,..\,2 \bullet (q_i, a, q'_i) \in \delta_i \wedge (\forall\, j \neq i \bullet q_j = q'_j)) \wedge$
  $a \in \Sigma_{int} \Rightarrow (\forall\, i \in 1\,..\,2 \bullet q_i = q'_i))\}$.

Since all common-named actions synchronise, and all actions which are enabled in a component can occur, the definition satisfies (2).

It is possible, by restricting $\delta$ in such compositions, to limit when operations are enabled, or to limit the agents which synchronise on an action. For example, if we had included two client agents, then we would expect only one to be involved in each request, reply and change-server action. ◇

## 4 Adaptivity Defined

In this section we provide formal definitions of adaptivity for agents and MAS based on their team automata representations as defined in Sect. 3. We base our definitions on Dijkstra's notion of legitimate states [8] which we extend to include both the state of the system under consideration (agent or MAS) and its environment. The definitions qualify adaptivity with respect to the external action to which the system adapts, and quantify it with respect to the number of actions required to adapt.

Since agents and MAS are represented by automata, the definition of adaptivity for each of them is identical. We begin by defining adaptivity in the special case of *closed systems*, *i.e.*, where the system does not interact with its environment, in Sect. 4.1. This definition is applicable to MAS which do not rely on environmental interaction for their operation. We then extend the definition to *open systems*, *i.e.*, where interaction with the environment is central to the system's operation, in Sect. 4.2. This definition is applicable to agents as well as MAS that interact with their environment.

### 4.1 Adaptivity of Closed Systems

A closed MAS $M$ can be modelled by a team automaton with no input actions. The team automaton of Example 2 is an example of such a closed system. Let $\mathcal{Q}(M)$ denote the set of legitimate states of $M$. By definition, all transitions from legitimate states lead to legitimate states. That is, given $M = (Q, I, \Sigma, \delta)$

$$(q, a, q') \in \delta \wedge q \in \mathcal{Q}(M) \Rightarrow q' \in \mathcal{Q}(M). \tag{3}$$

A MAS is *well-formed* if the initial states of $M$ are legitimate states, or if $M$ is guaranteed to reach a legitimate state in a finite number of actions. That is,

$$I \subseteq \mathcal{Q}(M) \vee (\forall\, b \in \mathcal{B}(M) \bullet \exists\, i \geq 0 \bullet st(b, i) \in \mathcal{Q}(M)). \tag{4}$$

In the case where a finite number of actions are required to reach a legitimate state, the MAS undergoes an initial self-configuration process.[1]

Let $Q$ be the set of states of $M$ and $Z$ be an external action defining the set of transitions $\zeta \subseteq Q \times Z \times Q$ on $M$. Such an external action can move the MAS from a legitimate state to an illegitimate one. $M$ can adapt to the external action, if it can return to a legitimate state.

[1] Self-configuration can itself be viewed as a type of adaptivity in which the external action is the system initialisation.

**Definition 4.** *A closed MAS $M$ is $Z$-adaptive if, after an occurrence of $Z$ which places the MAS in an illegitimate state, the MAS is guaranteed to reach a legitimate state in a finite number of transitions under the assumption of no further occurrences of $Z$.*

*That is, for all $b \in \mathcal{B}(M)$ such that there exists an $i \geq 0$ such that $st(b, i) = q$ and for all illegitimate states $q'$ such that $(q, Z, q') \in \zeta$ the following holds.*

$$\forall\, b' \in B(M, \{q'\}) \bullet \exists\, j > 0 \bullet st(b', j) \in \mathcal{Q}(M) \tag{5}$$

Note that we are concerned only with cases where $Z$ places the MAS in an illegitimate state. If $Z$ places the MAS in a legitimate state, the MAS is robust against $Z$, but we do not regard this as adapting (since there is no deflection from its normal behaviour).

A closed MAS $M$ is $n$-$Z$-adaptive for some $n > 0$, if it can adapt within at most $n$ transitions. That is, for all $b \in \mathcal{B}(M)$ such that there exists an $i \geq 0$ with $st(b, i) = q$ and for all illegitimate states $q'$ such that $(q, Z, q') \in \zeta$ the following holds.

$$\forall\, b' \in B(M, \{q'\}) \bullet \exists\, j \in 1 \,..\, n \bullet st(b', j) \in \mathcal{Q}(M) \tag{6}$$

The following theorems follow directly from these definitions.

**Theorem 1.** *If $M$ is $n$-$Z$-adaptive, it is also $m$-$Z$-adaptive for any $m \geq n$.*

**Theorem 2.** *If $M$ is $n$-$Z$-adaptive for some $n > 0$, then it is also $Z$-adaptive.*

Note that the inverse of Theorem 2 does not hold. It is possible, due to nondeterminism in a MAS, that there is no minimum number of transitions required to reach a legitimate state. For example, consider a MAS that after $Z$ is repeatedly able to choose between two actions $a$ and $b$, and reaches a legitimate state after choosing $b$. If we assume fairness (so that $b$ must eventually be chosen) the MAS is $Z$-adaptive. However, there is no $n$ for which it is $n$-$Z$-adaptive.

### 4.2 Adaptivity of Open Systems

An agent provides an example of an *open* system since its behaviour typically depends on its environment. The environment controls the agent's input actions, and may restrict the occurrence of its output actions when synchronisation is required. Similarly, a MAS can be an open system. In this section we will discuss adaptivity of agents, although the results are also directly applicable to open MAS.

To reason about an agent, we need to model the interactions with its environment. In interface automata, this is taken care of by the restrictions placed on the types of observable (input and output) actions and when they can occur [7]. The same is true for component and team automata.

With open systems, there are two kinds of external actions. The first change the state of the agent. They are identical to the external actions of closed systems and adaptivity to these actions can be reasoned about in the same way.

The second kind of external actions change the state of the system's environment, and possibly also the system's state. In the setting of component automata, this would manifest itself as (possibly) different restrictions on the observable actions.[2]

To facilitate modelling such an external action, we need to extend the agent's state with one or more auxiliary variables which the external action changes. These auxiliary variables, representing some facet of the environment, can be used to restrict when particular actions can occur. They are also used in defining the legitimate states.

*Example 3.* Consider the client agent of Sect. 3. A possible external action that could occur in its environment is a server going down. Let $Z_1$ be the external action that causes a server with which the client is interacting to go down when the client is in the state where it has not performed a request. Let $Z_2$ be an external action similar to $Z_1$ which occurs when the client has performed a request and is waiting for a reply.

To reason about the adaptivity of the client, we extend it with an auxiliary variable *down* which is the set of servers which are currently down. This set is initially empty. The transitions are restricted such that if the extended client is in a state where server $i$ is down, transitions corresponding to $cs.i$, $request.i$ and $reply.i$ cannot occur. If it is in a state where server $i$ is not down, the transitions can occur. The transitions do not change whether a given server is up or down.

Figure 3 shows part of the restricted client automaton. We show the states in which both servers are up, and also the states in which server 1 is down. We choose the legitimate states to be those where the client can perform request or reply actions, *i.e.*, $id \notin down$. These states are shaded in the figure. The external actions are shown using dotted arrows between states.

After a single occurrence of $Z_1$, the client is able to perform the change-server action restoring it to a legitimate state. Hence, the client is $Z_1$-adaptive. In fact, since it requires only one action to reach a legitimate state, it is 1-$Z_1$-adaptive. In a more detailed specification where, for example, the client was required to log in to the new server, the client would be $n$-$Z_1$-adaptive for some $n > 1$. Hence, the quantification of adaptivity with respect to number of actions is dependent on the level of abstraction. It should, therefore, be used only for comparing adaptive responses at the same level of abstraction.

In the case of $Z_2$, there is no possibility of performing the change-server action. The client is therefore not $Z_2$-adaptive. This may correspond to a design flaw which our reasoning allows us to detect and rectify if desired, *e.g.*, by introducing a timeout when waiting for a response. ◇

As can be seen from Example 3, the specifier must, based on an understanding of the system and its environment, determine the available transitions from states corresponding to different values of the auxiliary variables. It is possible that

[2] We assume all input actions possible in the environment are included in the agent automaton, as are all of the agent's possible output actions. Hence, no observable actions will be introduced or removed by such an external action.

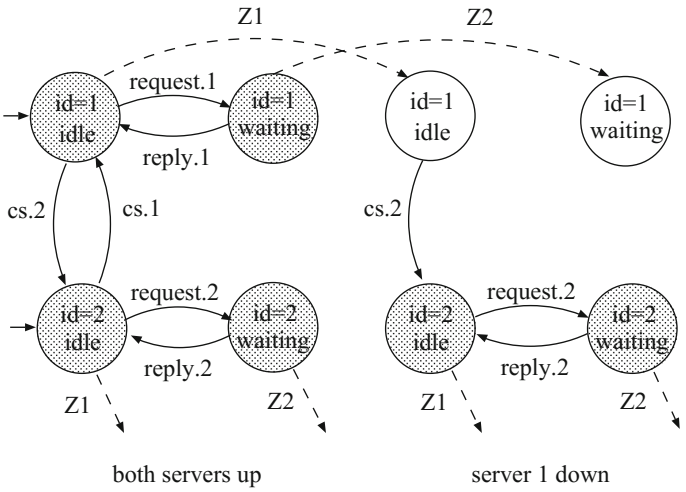


**Fig. 3.** Team automaton of the restricted client

these transitions change the values of the auxiliary variables. Although the agent cannot change these variables directly, it can interact with its environment to instigate their change. For example, if the agent of the above example was able to call a maintenance agent to fix the server, then as a consequence of this call it would return to a state where the server was no longer down. Whether the agent can do this and how the environment responds is up to the specifier.

To be adaptive to an external action $Z$, an agent must (i) be guaranteed to reach a legitimate state in a finite number of transitions, and (ii) have at least one behaviour which reaches a legitimate state without the auxiliary variables being changed. The second condition precludes agents which rely on the auxiliary variables changing to reach a legitimate state. For example, an agent may call a maintenance agent but have no other strategy for dealing with a server that is down. We would not regard such an agent as adaptive.

The approach is formalised as follows. We enhance the agent $A = (Q, I, \Sigma = \Sigma_{int} \cup \Sigma_{inp} \cup \Sigma_{out}, \delta)$ with a set of auxiliary variables describing a set of states $E$. The cross product $Q \times E$ captures the state space of $A$ extended with these auxiliary variables. Thus, the enhanced version of the agent is defined as $A' = (Q \times E, I \times I_E, \Sigma, \delta')$ where $I_E \subseteq E$ and $\delta' \subseteq (Q \times E) \times \Sigma \times (Q \times E)$. We require that the behaviours of $A'$ when restricted to $Q$ correspond to behaviours of the original agent $A$. That is,

$$\forall\, b \in \mathcal{B}(A') \bullet b\,|_Q \in \mathcal{B}(A) \tag{7}$$

where $b\,|_Q$ denotes the behaviour $b$ restricted to the state $Q$ of the original agent $A$. Hence, $A'$ behaves identically to $A$ in the absence of external actions. This is true in Example 3 since initially no servers are down.

Let $Z$ be an external action defining the set of transitions $\zeta \subseteq (Q \times E) \times Z \times (Q \times E)$ on $A'$.

**Definition 5.** *An agent (or open MAS) $A$ is $Z$-adaptive, if its enhancement $A'$ on which $Z$ is defined is, after an occurrence of $Z$ which places it in an illegitimate state, able to reach a legitimate state in a finite number of transitions, and at least one behaviour reaches a legitimate state without changing the extension to the state of $A$.*

*That is, for all $b \in \mathcal{B}(A')$ such that there exists an $i \geq 0$ such that $st(b, i) = q$ and $(q, Z, q') \in \zeta$ the following holds.*

$$\forall\, b' \in \mathcal{B}(A', \{q'\}) \bullet \exists\, j > 0 \bullet st(b', j) \in \mathcal{Q}(A') \tag{8}$$

*and*

$$\begin{aligned} &\exists\, b' \in \mathcal{B}(A', \{q'\}) \bullet \\ &\quad \exists\, j > 0 \bullet st(b', j) \in \mathcal{Q}(A') \wedge (\forall\, k \leq j \bullet st(b', k)\,|_E = q'\,|_E) \end{aligned} \tag{9}$$

*where $s\,|_E$ restricts a state $s$ of $A'$ to the variables of $E$.*

An agent (or open MAS) $A$ which is $Z$-adaptive is $n$-$Z$-adaptive for some $n > 0$, if it can adapt within at most $n$ transitions. That is, for all $b \in \mathcal{B}(A')$ such that there exists an $i \geq 0$ such that $st(b, i) = q$ and $(q, Z, q') \in \zeta$, the following holds along with condition (9) above.

$$\forall\, b' \in \mathcal{B}(A', \{q'\}) \bullet \exists\, j \in 1\,..\,n \bullet st(b, j) \in \mathcal{Q}(A') \tag{10}$$

Theorems 1 and 2 of Sect. 4.1 remain true and follow directly from these definitions.

## 5 A Teleo-Reactive Development Framework

The notions of component and team automata introduced in Sect. 3 have provided a convenient setting for the definition of adaptivity in Sect. 4. We now turn to the question of designing adaptive systems: specifically, to the generation of high-level (behavioural) specifications of such systems which can be developed to suitable implementations using any of the range of techniques discussed in Sect. 2.

Our approach is to make the adaptive behaviour of the system explicit in the specification, and thereby simplify its validation. This is achieved using a specification framework based on the teleo-reactive agent paradigm of Nilsson [23,24]. This paradigm, which was developed to facilitate

> "robustly [directing] an agent toward a goal in a manner that continuously takes into account the agent's changing perceptions of a dynamic environment" [24],

is well suited to our task.

A teleo-reactive agent is represented by an ordered list of production rules of the form

$$C \longrightarrow P$$

where $C$ is a condition evaluated with respect to the agent's world model (comprising its perception of its own state and that of the environment), and $P$ is a non-terminating program (referred to as a *durative action*). The behaviour of such an agent is to continuously evaluate the conditions in the list and, at any time, perform the durative action associated with the *first* production rule in the list whose condition is true. If no conditions are true, the behaviour of the teleo-reactive agent is undefined.

**Definition 6.** *A teleo-reactive agent is a 6-tuple* $T = (Q, I, \Sigma, \delta, C, \rho)$ *where*

- $(Q, I, \Sigma, \delta)$ *is a component automaton.*
- $C \subseteq 2^Q$ *is the set of conditions whose elements form a total order* $\leq$ *capturing the priority amongst production rules.*
- $\rho \subseteq C \times 2^\delta$ *is the agent's set of production rules and satisfies the following constraint. For all conditions* $c$ *in* $C$*, for all* $q \in c$ *such that* $q \notin c'$ *for any* $c' \leq c$*, there exists a transition* $(q, a, q') \in \rho(c)$*. This constraint ensures the set of actions associated with a condition represents a non-terminating program, or durative action.*

Assuming transitions in $\delta$ are atomic and that conditions are evaluated before each transition, the behaviour of a teleo-reactive agent, $T$, is a possibly infinite sequence alternating between states and actions $q_0\ a_1\ q_1\ a_2\ q_2\ \ldots$ where for all $i \geq 0$, if there exists a condition $c \in C$ such that $q_i \in c$ and, for all $c' \leq c$, $q_i \notin c'$ then $(q_i, a_i, q_{i+1}) \in \rho(c)$. If there does not exist a condition $c \in C$ such that $q_i \in c$ then the action and post state of the next transition are undefined; they can be any action and state including those not in the sets $Q$ and $\Sigma$ respectively. This would be used to leave implementation flexibility in an abstract design.

Teleo-reactive agents are usually designed to have a *regression* property whereby executing a durative action $P_i$ when condition $c$ holds, will eventually result in a condition $c'$ occurring earlier in the list, *i.e.*, $c' \leq c$. This enables an agent to eventually reach its goal corresponding to the first production rule in the list. In our context, we use the regression property to ensure an adaptive agent reaches a legitimate state.

### 5.1 Designing Adaptivity

In keeping with the generality of our results, our teleo-reactive framework is independent of the specification notation used to capture the behaviour of the agent's state and actions. We illustrate its use on a case study with a particular specification language, Object-Z [28], in Sect. 6.

To motivate our approach, we return to the client-server example of Sects. 3 and 4. In Fig. 3 we extended the original component automaton for a client agent to include a modified behaviour when server 1 is down (corresponding

to a particular value of the auxiliary variable *down*). Although not shown in Fig. 3, similar modified behaviours would exist for when server 2 is down and for when both servers are down. The client can therefore be thought of as being in one of four *modes*; the particular mode being determined by the state of the environment, *i.e.*, which servers are currently down.

As our definition of adaptivity precludes the agent changing its environment, reasoning about adaptivity following an external action $Z$ is reduced to reasoning within the mode to which $Z$ takes the agent. For example, when server 1 goes down we need consider only the behaviour within the four (out of 16) states corresponding to this server being down. We therefore define a teleo-reactive specification corresponding to each mode. This considerably simplifies the process of reasoning about adaptivity.

For the normal operational behaviour of the agent, *i.e.*, when no servers are down, the specification is as in Fig. 1. A teleo-reactive version of this specification is presented below.

$$Client \mathrel{\widehat{=}} true \longrightarrow \{request, respond, cs\}$$

where $\{request, respond, cs\}$ denotes a program which repeatedly chooses to perform one of the agent actions *request*, *respond* or *cs*. The choice when more than one action is enabled is nondeterministic. For this program to represent a durative action, it is necessary that at least one of the agent actions is always enabled (which is the case in the example). In general, we need to prove

$$\begin{aligned} \forall\, c \in C \bullet \forall\, q \in c \bullet (\forall\, c' \neq c \bullet c' \leq c \Rightarrow q \notin c') \Rightarrow \\ \exists\, q' \in Q, a \in \Sigma \bullet (q, a, q') \in \rho(c). \end{aligned} \tag{11}$$

In the teleo-reactive specification of *Client*, there are no illegitimate states and hence the teleo-reactive specification has only one production rule. In general, an agent may start in an illegitimate state and need to self-configure before normal operation begins. In that case, there would be one production rule whose condition describes the legitimate states followed by one or more production rules whose conditions describe illegitimate states. Self-configuration would be proved by showing that the specification has the previously mentioned regression property. Similarly, adaptivity to external duress can be shown in this way.

Consider the mode of the client when server 1 is down. The teleo-reactive specification is

$$\begin{aligned} Server1Down \mathrel{\widehat{=}}\ & id = s2 \longrightarrow \{request, respond\} \\ & id = s1 \wedge state = idle \longrightarrow \{cs\} \\ & id = s1 \wedge state = waiting \longrightarrow \{skip\} \end{aligned}$$

where the special action *skip* corresponds to the agent doing nothing.

The legitimate states are those satisfying $id = s2$. We divide the illegitimate states into those that result from $Z1$ (satisfying $id = s1 \wedge state = idle$) and those that result from $Z2$ (satisfying $id = s1 \wedge state = waiting$). To prove

adaptivity to $Z1$, we need to simply prove that $cs$ changes $id$ from $s1$ to $s2$ (since this will make the earlier condition $id = 2$ true moving the agent to a legitimate state). It is also obvious from the specification that the agent cannot adapt to $Z2$ (since *skip* does not change the agent's state). If this were not desirable, we could change the specification at this point to add, for example, the timeout action suggested in Sect. 4.

A similar teleo-reactive specification $Server2Down$ is given for the mode in which server 2 is down. For the mode where both servers are down the teleo-reactive specification is

$$BothServersDown \mathrel{\widehat{=}} true \longrightarrow \{skip\}$$

indicating there is no means for the agent to adapt; external (maintenance) actions are required for continued operation in this case.

By considering each mode in isolation as a teleo-reactive system, we can readily reason about the adaptivity of our design and modify the design when necessary. The final step of our approach combines the mode specifications as follows.

$$\begin{aligned} TR\_Client \mathrel{\widehat{=}}\ & down = \varnothing \longrightarrow Client \\ & down = \{s1\} \longrightarrow Server1Down \\ & down = \{s2\} \longrightarrow Server2Down \\ & down = \{s1, s2\} \longrightarrow BothServersDown \end{aligned}$$

where the occurrence of a teleo-reactive specification on the right-hand side of a production rule is a syntactic convention which expands as follows: A rule $C \longrightarrow TR$, where $TR$ is a teleo-reactive specification, is replaced by a sequence of rules $C \wedge C_i \longrightarrow P_i$ corresponding to the sequence of rules $C_i \longrightarrow P_i$ in $TR$. For example, $down = \{s1\} \longrightarrow Server1Down$ is replaced by

$$\begin{aligned} & down = \{s1\} \wedge id = s2 \longrightarrow \{request, respond\} \\ & down = \{s1\} \wedge id = s1 \wedge state = idle \longrightarrow \{cs\} \\ & down = \{s1\} \wedge id = s1 \wedge state = waiting \longrightarrow \{skip\}. \end{aligned}$$

Formally, a composed teleo-reactive system is defined as follows.

**Definition 7.** *Let a system have $n$ modes on the state space defined by states $Q$ and initial states $I$. Let $(Q, I, \Sigma_i, \delta_i, C_i, \rho_i)$ where $i \in 1..n$ be the teleo-reactive specification of the $i$th mode. Let $E$ denote the set of states described by the auxiliary variables, and $E_i \subseteq E$ be those states corresponding to the $i$th mode. Given the first mode corresponds to normal behaviour (in the absence of external duress), the teleo-reactive specification of the entire system is $(Q', I', \Sigma, \delta, C, \rho)$ where*

- $Q' = Q \times E$.
- $I' = I \times E_1$.
- $\Sigma = \bigcup_{i:1..n} \Sigma_i$.

- $\delta = \{(q, a, q') \in Q \times \Sigma \times Q \mid \exists\, i \in 1\,..\,n \bullet q\ |_E \in E_i \wedge (q|_Q, a, q'\ |_Q) \in \delta_i\}$.
- $C = C_1 \times E_1 \cup \ldots \cup C_n \times E_n$ *such that*
  - *for all* $i \in 1\,..\,n$ *and* $c, c' : C_i$ *where* $c' \leq c$, $c' \times E_i \leq c \times E_i$
  - *for all* $i, j \in 1\,..\,n$ *where* $i \leq j$ *and* $c' : C_i$ *and* $c : C_j$, $c' \times E_i \leq c \times E_j$.
- $\rho \subseteq C \times 2^{\delta}$ *such that for all* $c \in C_i \times E_i$, $(q, a, q') \in \rho(c)$ *iff* $(q\ |_Q, a, q'\ |_Q) \in \rho_i(c\ |_Q)$.

The composed teleo-reactive system begins in the first mode and transitions to other modes depending on the values of the auxiliary variables. The behaviour in a given mode is identical to that of the mode considered in isolation. Since the first mode corresponds to the behaviour in the absence of external actions, (7) is guaranteed to hold.

It should be noted that the assumption that modes operate on the same state space is not overly restrictive. If it is necessary to specify modes of a system which operate on different states, *i.e.*, different sets of variables, to combine these modes we first extend the states of each mode with the variables of the others to provide a unified state.

### 5.2 From Teleo-Reactive Specification to Component Automaton

While teleo-reactive implementation approaches exist [15], in general we may not wish to implement our system in a teleo-reactive style. We therefore provide a mapping from a teleo-reactive specification to the more general form of a component automaton.

**Definition 8.** *Let* $(Q, I, \Sigma, \delta, C, \rho)$*be a teleo-reactive system. The equivalent component automaton is* $(Q, I, \Sigma, \delta')$ *where*

$$\delta' = \bigcup_{c:C} \{(q, a, q') \in \rho(c) \mid q \in c \wedge \forall\, c' \leq c \bullet c' \neq c \Rightarrow q \notin c'\}.$$

A transition from the teleo-reactive system is enabled only when a condition under which it may occur is true, and all conditions of earlier production rules are false.

### 5.3 Adapting to Internal Disturbances

The approach proposed above assumes a set of auxiliary variables which capture the effect of an external action. In the client example these variables represent a state of the environment. The auxiliary variables can, however, also be used to represent a condition of the internal state of the system. The main difference is that it may be possible in this case for a system to move from one mode to another during adaptation. Hence, for showing that a system adapts when in a given mode may require showing that, rather than reaching a legitimate state in that mode, the system transitions to another mode from which it can adapt.

As a result the approach can be used in the development of systems (including closed systems) which adapt to changes to internal state. This is illustrated by the case study in Sect. 6.

## 6 Case Study: The Self-Adaptive Production Cell

In this section we illustrate our approach and its use with a specific formal notation, Object-Z [28]. Object-Z is an object-oriented extension of the well known Z specification language [32]. Its notions of classes and objects are ideal for specifying MAS [30]. Both Z and Object-Z have been advocated for the description of agents by other researchers in the field [9,14].

Our case study is a self-adaptive, multi-robot production cell based on that described by Nafz *et al.* [22]. The production cell comprises a number of robots which are capable of taking on various roles in the production of an item. These roles may involve the use of tools such as drills and screwdrivers. Since changing tools, and hence changing roles, is very time-consuming, the robots in the production cell take on different roles from each other and collaborate on the production of items; passing the items between them in the required order. For example, a typical scenario is a production cell with three robots: the first robot uses a drill to drill a hole in the item, the second inserts a screw, and the third tightens the screw with a screwdriver [22].

In the design of the system below, we consider both the self-configuration of the system from an initial state where no robot has a role, as well as the ability of the system to adapt to a robot losing a capability, *e.g.*, if a robot in the above scenario breaks its drill or screwdriver, or runs out of screws.

### 6.1 Normal Behaviour and Self-Configuration

We begin by specifying the normal behaviour of the system, in which all robots possess all capabilities. A robot is specified using an Object-Z class which, like a class in an object-oriented programming language, encapsulates state variables, their initial values and all operations which can change their values.

The robot has two state variables: *available* denoting the roles which are available for the robot to take on (corresponding to an internal model of its environment), and *roles* the robot's current roles (we will limit the number of roles to one in the specification, but, in general, cases where a robot could take on more than one role could be considered). Each variable is assigned a value which is a subset of a given type *Role* comprising all possible roles ($\mathbb{P}\, S$ denotes the power set of $S$). Initially, all roles are available and the robot has no role.

*Robot*

$$available : \mathbb{P}\, Role$$
$$roles : \mathbb{P}\, Role$$

---

$$roles \cap available = \varnothing$$
$$\#roles \leq 1$$

*INIT*

$available = Role$
$roles = \varnothing$

. . . operations detailed below

The operations of a class are named boxes with

- a $\Delta$-list (read "delta list") listing the state variables which the operation may change; all other variables remain unchanged. An operation without a $\Delta$-list cannot change any state variables.
- a number of declarations of local variables (such as inputs and outputs).
- a predicate restricting the values of the state variables both before and after the operation, and the values of the local variables. State variables after an operation are denoted by the variable name decorated with a prime, *e.g.*, $available'$.

The operations of class *Robot* are as follows.

A robot without a role may choose one from the available roles. The robot's new role is communicated to the environment via the output variable $r!$. At the level of abstraction of our specification we are assuming that two robots will not choose an available role simultaneously. At a lower level of abstraction such an occurrence would need to be resolved using a suitable contention mechanism such as the robot with the minimum (or maximum) identifier backing off, or both robots backing off for random amounts of time.

*ChooseRole*

$\Delta(available, roles)$
$r! : Role$

---

$role = \varnothing$
$r! \in available$
$available' = available \setminus \{r!\}$
$roles' = \{r!\}$

A robot receiving an input $r?$ corresponding to another robot taking on an available role, updates its environmental model accordingly.

*RemoveAvailable*

$\Delta(available)$
$r? : Role$

---

$r? \in available$
$available' = available \setminus \{r?\}$

A robot with a role may operate according to that role. We abstract from what the robot actually does including the passing of the item between robots: these aspects of the production cell have no effect on the system's adaptivity.

*Operate*

$$roles \neq \varnothing$$

The system is specified by another class *System* whose state comprises a set of robots; one for each role. Initially, each robot is in its initial state, *i.e.*, has no role and believes all roles are available.

*System*

$$robots : \mathbb{P}\, Robot$$

$$\#robots = \#Role$$

*INIT*

$$\forall\, r : robots \bullet r.INIT$$

$$ChooseRole \mathrel{\widehat=} [\!]\ r_0 : robots \bullet r_0.ChooseAvailable \parallel (\wedge r : robots \setminus \{r_0\} \bullet r.RemoveAvailable)$$

$$Operate \mathrel{\widehat=} \wedge r : robots \bullet r.Operate$$

The operations use the familiar dot notation from object-oriented programming to denote robots undergoing operations. In *ChooseRole* the choice operator $[\!]$ is used to select one robot $r_0$ to undergo operation *ChooseAvailable*, and the conjunction operator $\wedge$ to specify all other robots undergoing *UpdateAvailable*. The parallel composition operator $\parallel$ equates the inputs $r?$ of the latter robots' operations with the output $r!$ of $r_0$'s operation. A precise semantics of these operators is given by Smith [28]. In *Operate*, the conjunction operator is used to specify all robots undergoing their *Operate* operation.

Following the approach in the previous section and assuming that legitimate states are those in which all robots have a role, the teleo-reactive specification of the system is as follows.

$$NormalOperation \mathrel{\widehat=} \forall\, r : robots \bullet r.role \neq \varnothing \longrightarrow \{Operate\}$$
$$true \longrightarrow \{ChooseRole\}$$

Note that the second production rule's condition is implicitly $\exists\, r : robots \bullet r.role = \varnothing$ since this production rule is only considered when the condition of the first production rule is false.

Showing that (11) holds is straightforward. *ChooseRole* will be enabled whenever there is a robot that can undergo *ChooseAvailable*, *i.e.*, whenever there is a robot without a role (which is the implicit condition of the second production rule). Similarly, *Operate* will be enabled whenever all robots can undergo

*Operate*, *i.e.*, whenever all robots have a role (which is the condition of the first production rule).

To prove the system self-configures, we need to show that the teleo-reactive specification has the regression property, *i.e.*, that the condition of the first production rule will become true. Choosing the number of robots without a role as a variant, we can show that this variant will decrease by one with each occurrence of *ChooseRole*. Hence, with a finite number of robots the variant will decrease to zero and the condition of the first production rule will be true.

### 6.2 Adapting to Loss of Capabilities

We now consider two external actions that cause a single robot to lose a capability. The first action $Z1$ causes it to lose the capability to perform its current role. The second $Z2$ causes it to lose the capability to perform a role other than its current role, if any. We assume that legitimate states are those in which all robots are assigned a role that they can perform.

To model the effects of these external actions, we introduce an auxiliary variable $capabilities : Robot \pfun \mathbb{P}\, Role$ which maps robots in the system to those roles they are capable of performing. Given this variable, we will consider two modes of operation: the mode where all robots can perform all roles $\forall\, r : robots \bullet capabilities(r) = Role$, and the mode where a robot $r_0 : robots$ has lost a capability $c : Role$, $capabilities(r_0) = Role \setminus \{c\} \land (\forall\, r : robots \setminus \{r_0\} \bullet capabilities(r) = Role)$.

The behaviour of the former mode is captured by the teleo-reactive specification *NormalOperation* above. The behaviour of the latter is captured by the following teleo-reactive specification.

$$\begin{aligned} Loss(r_0, c) \mathrel{\widehat=} \; & r_0.role \neq \{c\} \longrightarrow NormalOperation \\ & true \longrightarrow \{ChooseRole, skip\} \end{aligned}$$

That is, when $r_0$'s role is not $c$ the system behaves as in mode *NormalOperation*, and when $r_0$'s role is $c$ the system may perform *ChooseRole* (when robots other than $r_0$ need to choose a role) and otherwise does nothing, *i.e.*, *skip*. Again it is straightforward to show (11) holds.

It is immediately obvious from the final production rule that the system is not $Z_1$-adaptive. To be able to adapt, it would need a way for the robot $r_0$ to release its current role in order to choose a new role; something we have not included in our specification. Further analysis shows that the system is also not $Z2$-adaptive: when $r_0$ has no role, it may choose $c$ causing the condition of the final production rule to be the only one enabled.

**Modifying the Design.** In the interests of making the production cell adaptive, we modify the original specification as follows. Firstly, we add a variable $capable\_of : \mathbb{P}\, Role$ to the class *Robot* to make robots self-aware of their own capabilities. Initially, this variable would be assigned the value *Role* and no operation would change this value (it would only be changed by external actions

such as $Z1$ and $Z2$). The operation *ChooseRole* would be modified so that a robot would only choose a role it was capable of performing, *i.e.*, we would add the predicate $r! \in capable\_of$.

We would also add a new operation *ReleaseRole* which allows a robot to release a role it is not capable of performing.

*ReleaseRole*

$$\Delta(available, role)$$
$$r! : Role$$

---

$$role = \{r!\}$$
$$r! \notin capable\_of$$
$$available' = available \cup \{r!\}$$
$$role' = \varnothing$$

Other robots would need to update their beliefs about the available roles using the following operations.

*AddAvailable*

$$\Delta(available)$$
$$r? : Role$$

---

$$r? \notin available$$
$$available' = available \cup \{r?\}$$

The corresponding operation added to class *System* would be

$$ReleaseRole \mathrel{\widehat=} [\!] \; r_0 : robots \bullet r_0.ReleaseRole \parallel (\wedge r : robots \setminus \{r_0\} \bullet r.AddAvailable)$$

Before reanalysing the mode $Loss(r_0, c)$ with this new specification, we first re-examine *NormalOperation* to make sure that the changes have not affected the system's ability to self-configure. *NormalOperation* is in fact unchanged since, in the absence of external actions, $capable\_of = Role$ for all robots, and *ReleaseRole* is never enabled.

A naive reinterpretation of mode $Loss(r_0, c)$ is

$$\begin{aligned} Loss(r_0, c) \mathrel{\widehat=} \; & r_0.role \neq \{c\} \longrightarrow NormalOperation \\ & true \longrightarrow \{ChooseRole, ReleaseRole\} \end{aligned}$$

However, it is easy to show that (11) no longer holds. When $r_0$ is the only robot without a role, and $c$ is the only available role then *ChooseRole* is not enabled. Hence, $Loss(r_0, c)$ needs to be modified to

$$\begin{aligned} Loss(r_0, c) \mathrel{\widehat=} \; & \forall\, r : robots \bullet r.role \neq \varnothing \longrightarrow \{Operate\} \\ & r_0.role \neq \{c\} \longrightarrow \{ChooseRole, skip\} \\ & true \longrightarrow \{ChooseRole, ReleaseRole\} \end{aligned}$$

from which it is easy to see that adaptivity is not assured: since *skip* is the only action enabled when $r_0$ is the only robot without a role, and $c$ is the only available role.

**Modifying the Design Further.** To remedy the above situation, we need robots to be able to release their current role for another robot to take on. This should only occur when the latter robot has lost the capability of a role and no other roles are available.

We add a boolean variable *reconfig* to class *Robot* to denote when a robot without a role is unable to choose any available role. This variable acts as a shared flag indicating that (partial) reconfiguration is required. This variable would initially be false. An operation *InitiateReconfig* sets it to true when a robot without a role cannot choose an available role.

*InitiateReconfig*

$$\Delta(reconfig)$$

---

$$role = \varnothing$$
$$available \cap capable\_of = \varnothing$$
$$reconfig'$$

Other robots would need to update their *reconfig* variable using the following operation.

*SetReconfig*

$$\Delta(reconfig)$$

---

$$reconfig'$$

The corresponding operation added to class *System* would be

$$InitiateReconfig \mathrel{\widehat=} [] \; r_0 : robots \bullet r_0.InitiateReconfig \wedge (\wedge r : robots \setminus \{r_0\} \bullet r.SetReconfig)$$

Note that conjunction is used in place of parallel composition here as there are no input and output variables in the combined operations.

A further operation *ChangeRole* is added to *Robot*. This operation is enabled when *reconfig* is true and the robot is capable of performing a role that is available. The robot releases its current role and takes on an available role. It also sets its *reconfig* value to false.

*ChangeRole*

$$\Delta(available, role, reconfig)$$
$$r\_old! : Role$$
$$r\_new! : Role$$

---

$$reconfig$$
$$r\_new! \in available \cap capable\_of$$
$$role = \{r\_old!\}$$
$$available' = (available \cup \{r\_old!\}) \setminus \{r\_new!\}$$
$$role' = \{r\_new!\}$$
$$\neg\, reconfig'$$

Other robots would need to update their beliefs about the available roles and *reconfig* using the following operation.

*ChangeAvailable*

$$\Delta(available, reconfig)$$
$$r\_old? : Role$$
$$r\_new? : Role$$

---

$$reconfig$$
$$r\_old? \notin available$$
$$r\_new? \in available$$
$$available' = (available \cup \{r\_old!\}) \setminus \{r\_new!\}$$
$$\neg\ reconfig'$$

The corresponding operation added to class *System* would be

$$ChangeRole \mathrel{\widehat=} [\!]\ r_0 : robots \bullet r_0.ChangeRole \parallel (\wedge r : robots \setminus \{r_0\} \bullet r.ChangeAvailable)$$

Again we need to re-examine the mode *NormalOperation* in light of the changes. As the new operations become enabled only when a robot loses a capability, there is no change to behaviour. Next we need to reinterpret mode $Loss(r_0, c)$ as a teleo-reactive system. One possible specification is

$$\begin{aligned} Loss(r_0, c) \mathrel{\widehat=} {} & r_0.role \neq \{c\} \wedge r_0.role \neq \varnothing \longrightarrow NormalOperation \\ & r_0.role = \varnothing \longrightarrow Reconfigure(r_0) \\ & r_0.role = \{c\} \longrightarrow \{ChooseRole, ReleaseRole\} \end{aligned}$$

$$\begin{aligned} Reconfigure(r_0) \mathrel{\widehat=} {} & r_0.available \cap r_0.capable\_of \neq \varnothing \longrightarrow \{ChooseRole\} \\ & r_0.reconfig \longrightarrow \{ChooseRole, ChangeRole\} \\ & true \longrightarrow \{ChooseRole, InitiateReconfig\} \end{aligned}$$

It is now possible to show that (11) holds: each of the conditions ensures that the associated atomic actions form a durative action. It is also possible to show that the system is both $Z_1$-adaptive and $Z_2$-adaptive. We may further quantify the adaptivity.

$Z_1$ causes a robot $r_0$ to lose the capability to perform its current role. Hence, it will need to perform *ReleaseRole*. The worst case, in terms of number of actions until it is in a legitimate state, occurs when all other robots have no role, and then choose all roles apart from the one $r_0$ has just released. This would force $r_0$ to perform *InitiateReconfig* after which another robot would perform *ChangeRole* before $r_0$ could finally choose a role. In this scenario each robot in the system performs *ChooseRole*, $r_0$ additionally performs *ReleaseRole* and *InitiateReconfig*, and another robot performs *ChangeRole*. Hence, for $n$ robots the system is $(n + 3)$-$Z1$-adaptive.

$Z_2$ causes a robot $r_0$ to lose a capability other than its current role, if any. The worst case occurs when all robots, including $r_0$, have no roles and then all

robots other than $r_0$ choose a role other than the one $r_0$ has just released. Again $r_0$ would be forced to perform *InitiateReconfig* and another robot would have to perform *ChangeRole* before $r_0$ could choose a role. Hence, for $n$ robots the system is $(n+2)$-$Z2$-adaptive.

Examining worst case scenarios in order to quantify adaptivity can highlight inefficiencies in the design. Above it can be seen that if the robots were aware of which role had been released, they could choose that role with priority thereby avoiding a later reconfiguration. At this stage the designer may make a decision to change the design to reflect this.

Further external actions and associated modes can then be considered in a similar fashion. For example, we might like to consider where a robot $r_0$ loses more than one capability. In this case, the boolean variable *reconfig* is not enough: robots changing their role cannot be certain that the role they are giving up is one which $r_0$ is capable of performing. Hence, *reconfig* would need to be replaced by a set of roles that $r_0$ cannot perform, for example. We could also consider multiple robots losing capabilities, or a single robot losing all capabilities. In the latter case, for the system to adapt robots would need to be able to take on more than one role simultaneously. This would allow the system to keep functioning but, due to the time needed to swap tools, in a degraded fashion. If multiple robots could fail completely then to reduce this degradation in performance, the functioning robots would need to share the load as evenly as possible. Our approach helps us to realise these requirements and prove that suitable collaborative mechanisms satisfy them.

### 6.3 Combining the Modes

With just the two modes we have considered, the teleo-reactive specification of the production cell would be

$$
\begin{aligned}
&\mathit{ProductionCell} \mathrel{\widehat=} \\
&\quad \forall\, r : \mathit{robots} \bullet \mathit{capabilities}(r) = \mathit{Role} \longrightarrow \mathit{NormalOperation} \\
&\quad \exists\, r_0 : \mathit{robots};\ c : \mathit{Role} \bullet \mathit{capabilities}(r_0) = \mathit{Role} \setminus \{c\} \land \\
&\qquad (\forall\, r : \mathit{robots} \setminus \{r_0\} \bullet \mathit{capabilities}(r) = \mathit{Role}) \longrightarrow \mathit{Loss}(r_0, c)
\end{aligned}
$$

It is straightforward to show that the equivalent component automaton has the same behaviour as the modified Object-Z specification. In this case study, the effect of the auxiliary variable *capabilities* to restrict actions in a given mode is captured by the introduced variable *capable_of* of the individual robots, and so does not need to appear in the final specification.

## 7 Conclusion

In this paper we have presented a formal definition of adaptivity in terms of Dijskstra's notion of self stabilisation [8], and based on this definition, a framework for designing adaptive systems. The latter is based on Nilsson's notion of

teleo-reactive agents [23,24]. Both our definition and framework are independent of any specific implementation mechanism for adaptivity, and any specific specification notation. We illustrated our approach using the Object-Z specification notation [28] on a case study involving a multi-robot production cell [22].

Formalisms by which adaptivity can be specified and verified are scant. Anceaume *et al.* [1] concentrate on self-organisation in dynamic networks. They argue that a definition of self-organisation cast in terms of convergence to predefined legitimate states is inadequate for two reasons: (a) the identification of legitimacy may be impractical (it may be emergent; the status of batteries in sensor networks is quoted as an example); and (b) due to the dynamic nature of the network, convergence to a legitimate state may not be possible. As a result they consider instead structural properties reflecting high interactivity between nodes, node mobility and heterogeneity, and view local self-organisation as reducing system entropy. Criticisms (a) and (b) may well be valid if only 'static' system descriptions are considered but are overcome if the dynamic features of the system are specified. Generally that requires non-local specifications; but a specification is not constrained to be local like the implementation. A simple but typical example is the glider in Conway's *Game of Life* where global time may be introduced as a 'specification device' to capture a property which emerges in the implementation from local interactions [27].

By pursuing that approach we have here offered a formalism complementary to that of [1], able to consider directly the system functionality of entirely general systems rather than secondary structural properties of dynamic networks.

An alternative to our approach should be mentioned, in which the specification of an adaptive MAS is not static but thought to change with time, reflecting adaptivity. That is the approach considered by Artikis [2] who defines a dynamic specification language C+, a kind of action language for expressing changing properties, and a 'causal calculator' for their execution. Our view is that a specification which by its definition changes with time offers none of the advantages expected of a specification. Much of the detail incurred by the present work solves the problem of how to capture, in a static specification, dynamic changes which are environmentally triggered.

**Acknowledgments.** This work was supported by Australian Research Council (ARC) Discovery Grant DP110101211 and the Macao Science and Technology Development Fund under the EAE project, grant number 072/2009/A3.

## References

1. Anceaume, E., Défago, X., Potop-Butucaru, M., Roy, M.: A framework for proving the self-organization of dynamic systems. CoRR, abs/1011.2312 (2010)
2. Artikis, A.: A formal specification of dynamic protocols for open agent systems. CoRR, abs/1005.4815 (2010)
3. Böcker, J., Schulz, B., Knoke, T., Fröhleke, N.: Self-optimization as a framework for advanced control systems. In: Industrial Electronics Conference (IECON 2006), pp. 4671–4675. IEEE (2006)

4. Bruni, R., Corradini, A., Gadducci, F., Lluch Lafuente, A., Vandin, A.: A conceptual framework for adaptation. In: de Lara, J., Zisman, A. (eds.) FASE 2012. LNCS, vol. 7212, pp. 240–254. Springer, Heidelberg (2012)
5. Bucchiarone, A., Lafuente, A.L., Marconi, A., Pistore, M.: A formalisation of adaptable pervasive flows. In: Laneve, C., Su, J. (eds.) WS-FM 2009. LNCS, vol. 6194, pp. 61–75. Springer, Heidelberg (2010)
6. Dastani, M., Hindriks, K.V., Meyer, J.-J.C. (eds.): Specification and Verification of Multi-agent Systems. Springer, Heidelberg (2010)
7. de Alfaro, L., Henzinger, T.A.: Interface automata. In: Symposium on Foundations of Software Engineering, pp. 109–120. ACM Press (2001)
8. Dijkstra, E.W.: Self-stabilizing systems in spite of distributed control. Commun. ACM **17**, 643–644 (1974)
9. d'Inverno, M., Luck, M.: Development and application of a formal agent framework. In: International Conference on Formal Engineering Methods (ICFEM'97), pp. 222–231. IEEE Press (1997)
10. Eiben, A.E., Smith, J.E.: Introduction to Evolutionary Computing. Natural Computing Series. Springer, Heidelberg (2003)
11. Ellis, C.A.: Team automata for groupware systems. In: Hayne, S., Prinz, W. (eds.) International ACM SIGGROUP Conference on Supporting Group Work: The Integration Challenge, pp. 415–424. ACM Press (1997)
12. Georgiadis, I., Magee, J., Kramer. J.: Self-organising software architectures for distributed systems. In: Workshop on Self-healing Systems (WOSS '02), pp. 33–38 (2002)
13. Gouda, M.G., Herman, T.: Adaptive programming. IEEE Trans. Softw. Eng. **17**(9), 911–921 (1991)
14. Gruer, P., Hilaire, V., Koukam, A., Cetnarowicz, K.: A formal framework for multiagent systems analysis and design. Expert Syst. Appl. **23**(4), 349–355 (2002)
15. Gubisch, G., Steinbauer, G., Weiglhofer, M., Wotawa, F.: A teleo-reactive architecture for fast, reactive and robust control of mobile robots. In: Nguyen, N.T., Borzemski, L., Grzech, A., Ali, M. (eds.) IEA/AIE 2008. LNCS (LNAI), vol. 5027, pp. 541–550. Springer, Heidelberg (2008)
16. Güdemann, M., Nafz, F., Ortmeier, F., Seebach, H., Reif, W.: A specification and construction paradigm for organic computing systems. In: IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2008), pp. 233–242. IEEE Computer Society Press (2008)
17. Hölzl, M., Wirsing, M.: Towards a system model for ensembles. In: Agha, G., Danvy, O., Meseguer, J. (eds.) Formal Modeling: Actors, Open Systems, Biological Systems. LNCS, vol. 7000, pp. 241–261. Springer, Heidelberg (2011)
18. Hunter, A., Delgrande, J.P.: Iterated belief change: a transition system approach. In: International Joint Conference on Artificial Intelligence (IJCAI05), pp. 460–465 (2005)
19. Lynch, N., Tuttle, M.: An introduction to Input/Output automata. CWI Q. **2**(3), 219–246 (1989)
20. Mitchell, T.: Machine Learning. McGraw Hill, New York (1997)
21. Mohyeldin, E., Fahrmair, M., Sitou, W., Spanfelner, B.: A generic framework for context aware and adaptation behaviour of reconfigurable systems. In: IEEE International Symposium on Personal Indoor and Mobile Radio Communications (PIMRC05). IEEE Press (2005)

22. Nafz, F., Ortmeier, F., Seebach, H., Steghöfer, J.-P., Reif, W.: A universal self-organization mechanism for role-based organic computing systems. In: González Nieto, J., Reif, W., Wang, G., Indulska, J. (eds.) ATC 2009. LNCS, vol. 5586, pp. 17–31. Springer, Heidelberg (2009)
23. Nilsson, N.: Teleo-reactive programs for agent control. J. Artif. Intell. Res. **1**, 139–158 (1994)
24. Nilsson, N.: Teleo-reactive programs and the triple-tower architecture. Electron. Trans. Artif. Intell. **5**, 99–110 (2001)
25. Polani, D.: Foundations and formalizations of self-organization. In: Prokopenko, M. (ed.) Advances in Applied Self-organizing Systems. Advanced Information and Knowledge Processing, pp. 19–37. Springer, Heidelberg (2008)
26. Sanders, J.W., Smith, G.: Assuring adaptive behaviour in self-organising systems. In: Self-Organising and Self-Adaptive Systems Workshop (SASOW 2010), pp. 172–177. IEEE Computer Society Press (2010)
27. Sanders, J.W., Smith, G.: Emergence and refinement. Formal Aspects Comput. **24**(1), 45–65 (2012)
28. Smith, G.: The Object-Z Specification Language. Kluwer, Norwell (2000)
29. Smith, G., Sanders, J.W., Winter, K.: Reasoning about adaptivity of agents and multi-agent systems. In: International Conference on Engineering of Complex Computer Systems (ICECCS 2012). IEEE Computer Society Press (2012)
30. Smith, G., Winter, K.: Incremental development of multi-agent systems in Object-Z. In: Software Engineering Workshop (SEW-35). IEEE Computer Society Press (2012)
31. Smith, J.B.: Collective Intelligence in Computer-Based Collaboration. Lawrence Erlbaum Associates, Hillsdale (1994)
32. Spivey, J.M.: The Z Notation: A Reference Manual, 2nd edn. Prentice-Hall International, Englewood Cliffs (1992)
33. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge (1998)
34. ter Beek, M., Ellis, C., Kleijn, J., Rozenberg, G.: Synchronizations in team automata for groupware systems. Comput. Support. Coop. Work: J. Collab. Comput. **12**(1), 21–69 (2003)
35. Valiant, L.: A theory of the learnable. Commun. ACM **27**, 1134–1142 (1984)
36. Wooldridge, M.: An Introduction to Multiagent Systems. Wiley, New York (2002)
37. Zambonelli, F., Omicini, A.: Challenges and research directions in agent-oriented software engineering. Auton. Agent. Multi-Agent Syst. **9**(3), 253–283 (2004)