Adaptive Technology and Its Applications

João José Neto
Universidade de São Paulo, Brazil

INTRODUCTION

Before the advent of software engineering, the lack of memory space in computers and the absence of established programming methodologies led early programmers to use self-modification as a regular coding strategy.

Although unavoidable and valuable for that class of software, solutions using self-modification proved inadequate while programs grew in size and complexity, and security and reliability became major requirements.

Software engineering, in the 70's, almost led to the vanishing of self-modifying software, whose occurrence was afterwards limited to small low-level machine-language programs with very special requirements.

Nevertheless, recent research developed in this area, and the modern needs for powerful and effective ways to represent and handle complex phenomena in high-technology computers are leading self-modification to be considered again as an implementation choice in several situations.

Artificial intelligence strongly contributed for this scenario by developing and applying non-conventional approaches, e.g. heuristics, knowledge representation and handling, inference methods, evolving software/hardware, genetic algorithms, neural networks, fuzzy systems, expert systems, machine learning, etc.

In this publication, another alternative is proposed for developing Artificial Intelligence applications: the use of adaptive devices, a special class of abstractions whose practical application in the solution of current problems is called Adaptive Technology.

The behavior of adaptive devices is defined by a dynamic set of rules. In this case, knowledge may be represented, stored and handled within that set of rules by adding and removing rules that represent the addition or elimination of the information they represent.

Because of the explicit way adopted for representing and acquiring knowledge, adaptivity provides a very simple abstraction for the implementation of artificial learning mechanisms: knowledge may be comfortably gathered by inserting and removing rules, and handled by tracking the evolution of the set of rules and by interpreting the collected information as the representation of the knowledge encoded in the rule set.

MAIN FOCUS OF THIS ARTICLE

This article provides concepts and foundations on adaptivity and adaptive technology, gives a general formulation for adaptive abstractions in use and indicates their main applications.

It shows how rule-driven devices may turn into adaptive devices to be applied in learning systems modeling, and introduces a recently formulated kind of adaptive abstractions having adaptive subjacent devices. This novel feature may be valuable for implementing meta-learning, since it enables adaptive devices to change dynamically the way they modify their own set of defining rules.

A significant amount of information concerning adaptivity and related subjects may be found at the (LTA Web site).

BACKGROUND

This section summarizes the foundations of adaptivity and establishes a general formulation for adaptive rule-driven devices (Neto, 2001), non-adaptivity being the only restriction imposed to the subjacent device.

Some theoretical background is desirable for the study and research on adaptivity and Adaptive Technology: formal languages, grammars, automata, computation models, rule-driven abstractions and related subjects.

Nevertheless, either for programming purposes or for an initial contact with the theme, it may be unproblematic to catch the basics of adaptivity even having no prior expertise with computer-theoretical subjects.

In adaptive abstractions, adaptivity may be achieved by attaching adaptive actions to selected rules chosen

from the rule set defining some subjacent non-adaptive device.

Adaptive actions enable adaptive devices to dynamically change their behavior without external help, by modifying their own set of defining rules whenever their subjacent rule is executed.

For practical reasons, up to two adaptive actions are allowed: one to be performed prior to the execution of its underlying rule, and the other, after it.

An adaptive device behaves just as it were piecewise non-adaptive: starting with the configuration of its initial underlying device, it iterates the following two steps, until reaching some well-defined final configuration:

- While no adaptive action is executed, run the underlying device;
- Modify the set of rules defining the device by executing an adaptive action.

Rule-Driven Devices

A *rule-driven device* is any formal abstraction whose behavior is described by a rule set that maps each possible configuration of the device into a corresponding next one.

A device is *deterministic* when, for any configuration and any input, a single next configuration is possible. Otherwise, it is said *non-deterministic*.

Non-deterministic devices allow multiple valid possibilities for each move, and require backtracking, so deterministic equivalents are usually preferable in practice.

Assume that:

- D is some rule-driven device, defined as $D = (C, R, S, c_0, A)$.
- C is its set of possible configurations.
- $R \subseteq C \times (S \cup \{\varepsilon\}) \times C$ is the set of rules describing its behavior, where ε denotes empty stimulus, representing no events at all.
- S is its set of valid input stimuli.
- $c_0 \in C$ is its initial configuration.
- $A \subseteq C$ is its set of final configurations.

Let $c_i \Rightarrow^{(r)} c_{i+1}$ (for short, $c_i \Rightarrow c_{i+1}$) denote the application of some rule $r = (c_i, s, c_{i+1}) \in R$ to the current configuration c_i in response to some input stimulus $s \in S \cup \{\varepsilon\}$, yielding its next configuration c_{i+1}.

Successive applications of rules in response to a stream $w \in S^*$ of input stimuli, starting from the initial configuration c_0 and leading to some final configuration $c \in A$ is denoted $c_0 \Rightarrow_w^* c$ (The star postfix operator in the formulae denotes the Kleene closure: its preceding element may be re-instantiated or reapplied an arbitrary number of times).

We say that D defines a sentence w if, and only if, $c_0 \Rightarrow_w^* c$ holds for some $c \in A$. The collection $L(D)$ of all such sentences is called the language defined by D:

$$L(D) = \{w \in S^* \mid c_0 \Rightarrow_w^* c, c \in A\}.$$

Adaptive (Rule-Driven) Devices

An adaptive rule-driven device $AD = (ND_0, AM)$ associates an initial subjacent rule-driven device $ND_0 = (C, NR_0, S, c_0, A)$, to some adaptive mechanism AM, that can dynamically change its behavior by modifying its defining rules.

That is accomplished by executing non-null adaptive actions chosen from a set AA of *adaptive actions*, which includes the *null adaptive action* a^0.

A built-in counter t starts at 0 and is self-incremented upon any adaptive actions' execution. Let X_j denote the value of X after j executions of adaptive actions by AD.

Adaptive actions in AA call functions that map AD current set AR_t of adaptive rules into AR_{t+1} by inserting to and removing adaptive rules ar from AM.

Let **AR** be the set of all possible sets of adaptive rules for AD. Any $a^k \in A$ maps the current set of rules $AR_t \in \mathbf{AR}$ into $AR_{t+1} \in \mathbf{AR}$:

$$a^k : \mathbf{AR} \rightarrow \mathbf{AR}$$

AM associates to each rule $nr^p \in NR$ of AD underlying device ND a pair of adaptive actions $ba^p, aa^p \in AA$:

$$AM \subseteq AA \times NR \times AA$$