

Paper:

Anytime System Scheduler for Insufficient Resource Availability

Annamária R. Várkonyi-Kóczy, and Gábor Samu

Department of Measurement and Information Systems
Integrated Intelligent Systems Japanese-Hungarian Joint Laboratory
Budapest University of Technology and Economics
H-1521, Magyar Tudósok krt. 2., Hungary
E-mail: koczy@mit.bme.hu
E-mail: sg222@hszk.bme.hu
[Received August 31, 2003; accepted April 20, 2004]

Anytime systems are advantageous when resource and/or data availability changes during operation and some kind of intelligent reconfiguration of the system is needed to cope with temporal data and resource access conditions. Such systems may provide an optimal tradeoff between time/resource needs and computational complexity and the quality (accuracy) of results and are designed using special models, methods, and algorithms together with applying active monitoring for being able to supervise the operation of the system on-line and making intelligent decisions based on sensory information of so-called shortage indicators. Since the monitor operates under prescribed response time requirements and the number and complexity of the executable tasks may be very high, especially in complex systems, new considerations must constantly be made to achieve optimal or acceptable performance. In software terms, this requires the application of special compilation methods dealing also with timing considerations and constraints of the underlying operating system and even with the run-time characteristics of the monitor. This must also be supported by anytime development tools and special anytime description languages. In this paper a hierarchical compilation method is introduced together with theoretical considerations about a possible anytime development tool and the basics of the anytime description meta-language ATDL are presented.

Keywords: anytime systems, anytime compilation, hierarchical compilation, active monitoring, anytime development platform, anytime description language

1. Introduction

Monitoring and diagnostics systems applied in continuous technologies are able to quickly identify run-time failures arising in the given technology and can neutralize them within certain limits. This behavior assumes varied information processing tasks solved by a computer in real-time at a well-defined response time. It is an obvious

requirement to provide enough computational power but the achievable processing speed is highly influenced by the precedence, timing, and data access conditions of the processing itself. It thus appears inevitable, even in extremely careful design, to run into situations involving a serious shortage of necessary data and/or processing time, resulting in a critical breakdown of the monitoring and/or diagnostic systems [1]. The concept of “anytime” processing tries to handle cases in which too many abrupt changes and their consequences occur in larger scale embedded systems [2], the idea being that if there is a temporal shortage of computational power and/or there is a loss of some data the actual operations should be continued to maintain overall performance at a “lower price,” i.e., information processing based on algorithms and/or models of simpler complexity should provide outputs of quality acceptable enough to continue the operation of the complete monitoring system. Processing accuracy may be temporarily lower but possibly still high enough to produce data for qualitative evaluations and supporting decisions [3].

Systems possessing this (anytime) property need an intelligent monitor as an add-in which tries to optimize the operation of the complete controlled system by using sensory information about the current state. Since the monitor also operates under prescribed response time requirements and the number and complexity of executable tasks may be very high, especially in complex systems, new considerations must be made to achieve optimal or acceptable performance. At software level, this requires the application of special compilation dealing also with timing considerations and constraints of the underlying operating system and even with the run-time characteristics of the monitor.

Complex systems usually consist of smaller subsystems which can be viewed as elementary modules. For complex anytime systems, optimization means, among other things, distribution of time allocations to the above elementary modules (computational elements) with respect to the optimization criteria of the system as a whole. For this, special databases are constructed for the monitor by compilation that deals with special properties of the elementary modules. Some properties are common,

Table 1. Performance profiles.

	Function	Name
Deterministical	$q(t)$	PP
	$q(q_{in}, t)$	CPP
Statistical	$p(t)[q_{out}]$	PDP
	$p(q_{in}, t)[q_{out}]$	CPDP
Expected	$q_E(t)$	EPP
	$q_E(q_{in}, t)$	ECPP

so compilation can be accelerated by algorithms based on the consequences of the properties. Anytime development tools and special anytime description languages can also support these tasks.

In this paper, we introduce a hierarchical compilation method and theoretical considerations about a possible anytime development tool together with the basics of the anytime description meta-language ATDL. The paper is organized as follows: Section 2 gives an overview about basic terms of anytime computation. Section 3 briefly discusses the compilation of anytime algorithms. Sections 4 and 5 summarize the main ideas and operation of hierarchical compilation. Section 6 describes the theoretical basics of an anytime development tool and proposes a new anytime description language, called ATDL.

2. Anytime Computation

If an algorithm can be executed for different running times for which a longer time allocation causes the algorithm yield output of a better quality for some aspect then it is called an *anytime algorithm*. Anytime algorithms are characterized by relations in which execution time and the quality of the input and output may be involved. Relations are represented as one-, two-, and sometimes multi-dimensional tables. These relations, or rather mappings, are called *performance profiles* [4,5].

Several types of *profiles* can be defined and constructed, the main distinction being which parameter is present as part of the input in the relation: input quality and/or output quality. If input quality is used, then the profile is *conditional*. The input-output relation may be treated statistically, in which case the output of the profile is a *distribution* of probability. Output quality is the parameter of this distribution and thus of the profile itself. Time is a parameter of every profile. The most important profile types are the Performance Profile (PP), the Conditional Performance Profile (CPP), the Performance Distribution Profile (PDP), the Conditional Performance Distribution Profile (CPDP), the Expected Performance Profile (EPP), and the Conditional Expected Performance Profile (CEPP) (**Table 1**). (Note that anytime literature uses the notation CPP mostly for Conditional Performance Distribution Profiles.)

Quality is the measure of the “goodness” of any given object in some aspect based on the possible values of the

given object. Quality may be defined for simple numbers and for complex and abstract structures.

The *quality function* is mapping from object values to quality values. Together with this, the (time dependent) output function, and a measurement method, performance profiles can be constructed.

Since using algorithms with decreasing quality (or expected quality) by increasing time allocation is pointless, profiles must possess an *increasing monotonic* property along time allocation. Similarly, any *increasing input quality* is assumed to cause *nondecreasing output quality*.

Other profile types are also definable. For details on dynamic profiles and time-dependent planning under uncertainty, see [4,6–8].

3. Compilation of Anytime Algorithms

Complex anytime systems, like other complex systems, can be decomposed into elementary anytime or nonanytime modules to facilitate design. Systems operate as whole entities, therefore compilation is needed to handle and operate these systems. Compilation compiles numerous elementary algorithms into a so-called *composite* that acts as an elementary module and has one or more profiles and a special structure containing elementary allocation times for modules involved in compilation. Because a composite and an elementary module have the same type of description and, usually, the same behavior, a composite may be both the output of a compilation step and its input.

The *compilation process* compiles elementary modules into one composite. This process may involve one or more *compilation steps*, which make transitional composites and contain composites in their input.

Let expression $C\{S_1, S_2, \dots, S_k, B_1, B_2, \dots, B_n\}$ denote a single compilation step in which elements of sets S_i ($i = 1, \dots, k$) and all elements B_i ($i = 1, \dots, n$) are compiled into one composite.

Modules having at least one input driven by an output of another module must have conditional profile(s), meaning only CPPs (ECPPs) and CPDPs can be used.

For systems with two single-input-single-output (SISO) anytime modules connected serially and using CPPs, output quality is formulated as follows:

$$q_2(q_1[q_{in}, T_1], T_2) \cdot \dots \cdot \dots \cdot \dots \quad (1)$$

where q_{in} denotes the (omittable) quality of the system input, the second module (with index 2) is connected after the first one, q denotes conditional performance profiles, and T stands for time allocations.

The composite profile is constructed so that for every composite allocation and composite input quality, the elementary allocation pair with the highest composite output quality is selected from possible pairs.

Using CPPs:

$$q(q_{in}, T) = \max_{T_1} q_2(q_1[q_{in}, T_1], T - T_1) \quad \dots \quad (2)$$