Intelligent Monitoring System for the Optimization of the Operation of Systems in Resource Insufficient Environment

Gábor Samu, Annamária R. Várkonyi-Kóczy

Integrated Intelligent Systems Japanese-Hungarian Laboratory
Department of Measurement and Information Systems,
Budapest University of Technology and Economics,
Magyar tudósok körútja 2., H-1117 Budapest, Hungary
sg222@hszk.bme.hu, koczy@mit.bme.hu

Abstract: Agents operating in real-time domain have to encounter resource insufficiency in some cases and applications even with deliberative and aware design. The resource demands may vary depending on the operation phases, change in/of the environment and especially in alert conditions and in situations caused by sudden events. A strong hardware base can be required for operating in dynamically changing environment. Some real-time agents applying sophisticated AI algorithms have to comply with timing constraints and continuous operation as well, leaving them to make a compromise between the consumption of the continuous and sufficient computation time and one or more performance properties. Anytime algorithms provide a tradeoff between the time resource and computational performance, namely the quality of the results. This kind of behaviour is favorable in some domains, i.e. a less optimal decision can be better than a missing one. In case of complex systems usually a decomposition is applied during the design process, thus for the construction of a usable system the compilation of the operational units (modules) is also needed. The use of anytime algorithms as operational units assumes a complex software frame, an operation system which contains a so-called scheduler capable of distributing the computation time among the anytime and non-anytime algorithms. This scheduler which is called as monitor in anytime systems has to deal with the timing considerations of many portions of the underlying operation system and even its own run-time characteristics to determine the time allocations while tracking the performance of the algorithms and the state of the environment. There is a gap between the implementation of these anytime systems and the theoretical results as the compilation and the monitoring have. A survey of a new compilation technique, a monitoring scheme, and an implementation approach are covered by this paper.

Keywords: Anytime systems, anytime compilation, hierarchical compilation, active monitoring, implementation

1 Introduction

The state of the art of the diagnostics systems applied in continuous technologies is able to quickly identify run-time failures arising in the given technology and can neutralize them between certain limits. This behavior assumes various information processing tasks solved by a computer at real-time respectively at a well defined response time. Serious time and/or data deficiency appearing in just the critical operational phases is unavoidable even by a deliberative design, causing malfunctions in the diagnostics system. Applying generalized anytime algorithms may offer the proper get-out from the problems above. They can provide results with less but still acceptable quality by adapting to the available time and processing capacities at the actual state in order to sustain continuous operation even in critical situations. Systems possessing this property need an intelligent monitor as an add-in which tries to optimize the operation of the complete supervised system by using sensory information about the current state. Complex systems usually consist of smaller subsystems which are called as elementary modules. For complex systems optimization means the distribution of the time allocations given to the above-mentioned elementary modules (computational elements) as to optimize the overall performance of the whole system. For this purpose special databases are constructed for the monitor by compilation methods which deal with special properties of the elementary modules. Some properties are common hence compilation methods can be accelerated by algorithms based on the consequences of these properties. In this paper the theoretical basics of a new compilation method, the hierarchical compilation, and the active monitoring scheme based on this proposed technique are introduced beside a rough description of an implementation approach.

2 Anytime computation

If an algorithm can be executed for different running times and a longer time allocation causes the algorithm giving an output of better quality in some aspect then it is called as an *anytime algorithm*. Anytime algorithms are characterized by relations in which the execution time, the quality of the input(s) and output(s) may be involved. The relations are represented as one-, two-, and sometimes multidimensional tables. These relations (or rather mappings) are called *performance profiles*. (See [1].)

Quality is the measure of the “goodness” of any given object in some aspect according to the possible values of the given object. Quality can be defined not only for simple numbers but also for complex and abstract structures. E.g. a path given by a path-planning algorithm may be qualified in several ways. One can define a quality based on the length of the actual path by simply using the ratio of

the minimal and the actual lengths. (Minimal length may be obtained from maximal speed.)

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

2.1 Profile types

Several types of profiles can be defined and constructed. The main aspect of distinction is that which parameter is present as a part of the input in the relation: input quality and/or output quality. If input quality is used then the profile will be *conditional*. The input-output relation may be treated in statistical manner. In this case the output of the profile is a *probability distribution*. The output quality is the parameter of this distribution and thus of the profile itself too. The time is a parameter of every profile.

Table 1 summarizes the different profile types, where the notations are: t : time, q_{in} : input quality, q_{out} : output quality, q : function with quality value output (deterministical), p : function giving a probability distribution output (statistical). q_E : some type of expected profile.

Table 1: Performance profiles

	Function	*Acronym*	*Name*
Deterministical	$q(t)$	PP	Performance Profile
	$q(q_{in}, t)$	CPP *	Conditional Performance Profile
Statistical	$p(t)[q_{out}]$	PDP	Performance Distribution Profile
	$p(q_{in}, t)[q_{out}]$	CPDP	Conditional Perf. Distr. Profile *
Expected	$q_E(t)$	EPP	Expected Performance Profile
	$q_E(q_{in}, t)$	ECPP	Expected Cond. Perf. Profile

* *Anytime literature uses the notation CPP mostly for Conditional Performance Distribution Profiles (CPDPs in this paper).*

Expected profiles are usually constructed from the appropriate statistical profiles by computing expected values, along the output quality.

2.2 Profile properties

There is no point in using algorithms with decreasing quality (or expected quality) by increasing time allocation, so profiles have to possess the *increasing monotonic* property along time allocation.

Similarly it can be assumed that any *increasing input quality* causes *non-decreasing output qualities.*

Other profile types can also be defined, see [1], [4] and [8] for further details of dynamic profiles and time-dependent planning under uncertainty.

3 Compilation of anytime algorithms

Complex anytime systems (like other complex systems) can be decomposed to elementary anytime (or non-anytime) modules in order to allow the design process. On the other hand, systems work as whole entities therefore some kind of compilation is needed to handle and to operate these systems. By compilation many elementary algorithms can be compiled to a so called *composite* which acts like an elementary module and has one or more profiles and a special structure containing elementary allocation times for modules involved in the compilation. Since a composite and an elementary module have the same type of description (and indistinguishable behaviour in usual cases), a composite may not only be an output of a compilation step but it can also be an input of it.

The compilation of elementary modules to one composite is the *compilation process*. The compilation process may consist of one or more *compilation step*s. The letter can make transitional composites and contains composites in its input.

Later in this paper let the expression $C\{S_1, S_2, ..., S_k, B_1, B_2, ..., B_n\}$ denote a single compilation step where the elements of sets S_i $(i = 1..k)$ and all the elements B_i $(i = 1..n)$ are compiled to one composite.

3.1 Compilation of two anytime modules

It is obvious that those modules which have at least one input driven by an output of an other module have to be characterized by conditional profile(s). Consequently, only CPPs (ECPPs) and CPDPs can be used. For a system with two single-input single-output (SISO) anytime modules connected in serial and using CPPs the output quality can be formulated as follows:

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

where q_{in} denotes the (omittable) quality of the system input and the second module (using index 2) is connected *after* the first one. T_i and q_i stand for the time allocation and for the CPP of the i-th module, respectively.

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 the possible pairs.

Using CPPs:

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

where $T = T_1 + T_2$, obviously.

If modules are characterized by CPDPs then better output quality means better expected output quality since statistical profiles operate with probability distributions:

$$q(q_{in}, T)[q_{out}] = \sum_{i=1}^{N_{q_1}} \left(p_1(q_{in}, T_1)[q_{1out,i}] \cdot p_2(q_{1out,i}, T - T_1)[q_{out}] \right), \tag{3}$$

where q_{in} and q_{out} denote the system input and output, respectively. N_{q_1} and N_{q_2} nominate the extent of elementary profiles among the quality dimension.

Only compilation with CPPs will be discussed hereinafter in this paper. See [1] for further details of working with CPDPs.

3.2 Global Compilation

The method described in the previous subsection can be extended to multimodular systems but here we have to note that the complexity of the algorithm is growing exponentially by the number of modules which limits its practical applicability [1].

Assume a system Y with N elementary modules $M_1,...,M_N$ connected in *any structure* and characterized by their CPPs. Let M_N be the notation of the module at the system output. The output quality of the composite can be expressed as follows:

$$q_C(q_{in}, T_1,...,T_M) = q_M(...,q_1[...,T_1],...,q_{M-1}[...,T_{M-1}],...,T_M), \tag{4}$$

where $q_i(...,T_i)$ denotes the CPP of the i th module. q_C is a compound function in which the elementary profiles and compound functions are the subfunctions.

The composite profile can be built by the formula below:

$$q(q_{in}, T) = \max_{T_1,...,T_{N-1}} q_C(q_{in}, T_1,...,T_{N-1}, T - \sum_{i=1}^{N-1} T_i) \tag{5}$$

It is apparent that this method is not applicable for more than 3 modules therefore better methods should be applied to eliminate the complexity problem. Fortunately profiles have some special properties as stated above.

3.3 Modules with multiple inputs

By using SISO modules only chain structures can be created. More general, tree structures can be made of multiple-input-single-output (MISO) modules (see Figure 1). Compilation of these tree-structured systems is effectively solved by the *local compilation* method introduced by Zilberstein [1] and discussed in the next subsection.

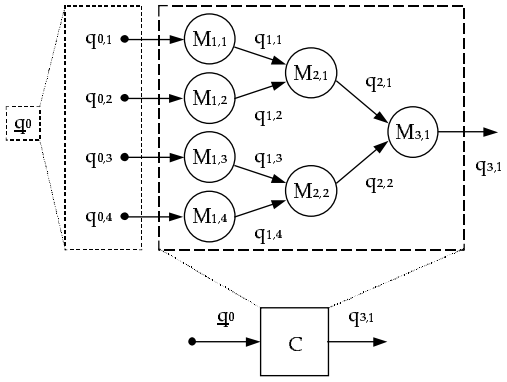

Figure 1: Tree structured system

3.4 Local Compilation

Local compilation compiles a module and modules connected on its immediate inputs to one composite. Local compilation produces global optimum (with pseudo-polynomial complexity) under the assumptions below:

1. Modules have conditional profiles
2. Profiles are monotonic non-decreasing functions of the input quality
3. System has a tree structure with bounded degree
4. CPDPs hold the input linearity property

Proof is given in [1]. Tree and chain structured anytime systems are discussed briefly in [3].

The example in Figure 1 may be compiled by the *compilation process* shown below:

1. $C_{1,1} = C^{(L)}\{M_{1,1}, M_{1,2}, M_{2,1}\}$
2. $C_{1,2} = C^{(L)}\{M_{1,3}, M_{1,4}, M_{2,2}\}$
3. $C = C^{(L)}\{C_{1,1}, C_{1,2}, M_{3,1}\}$

where operator $C^{(L)}$ denotes a local *compilation step* which can be expressed for the compilation of modules $M_{1,1}$, $M_{1,2}$ and $M_{2,1}$ for example in the following form:

$$q_{C,2,1}(q_{0,1}, q_{0,2}, T_{2,1}) = \max_{T_{1,1}, T_{1,2}} q_{2,1}[q_{1,1}(q_{0,1}, T_{1,1}), q_{1,2}(q_{0,2}, T_{1,2}), T_{2,1} - T_{1,1} - T_{1,2}] \quad (6)$$

This formula differs from the general formula of global compilation (5) only by the arrangement of the subfunctions and not by the complexity hence a tree-structured system with degree of more than 3 possesses the same complexity as a chain of as many modules as the degree.

If the degree of the tree is quite high and/or the system has any structure different than a tree then the usage of local compilation raises difficulties. Local compilation processes the graph *from the inputs* therefore modules with multiple output can not be compiled in such a way.

System graphs can also be processed *from the output*, moreover subsystems with special interfaces can be compiled independently and then to the rest of the system. The idea of processing subsystems as independent systems leads to the new idea of hierarchical compilation and to the Output Based Incremental Compilation process described in the following chapters.

3.5 Hierarchical Compilation

Definition 1: Selecting a subsystem *S* with a *single output* and *multiple inputs* from any given *DAG* (Directed Acyclic Graph) represented anytime system *Y* and replacing the subsystem by its global, local, or hierarchical compilated composite is called *hierarchical compilation.* See Figure 2.

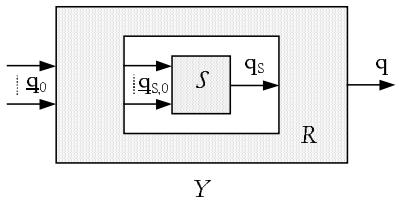

Figure 2: Hierarchical compilation

Hierarchical compilation may yield the optimality of global compilation if certain assumptions are made:

1. *DAG represented system.*
2. *Input monotonity (along both the time and the input quality dimensions).*
3. *Modules described by CPPs.*

Theorem 1: Hierarchical compilation provides global optimum beside the assumptions above.

The proof of **Theorem 1** is discussed briefly in [2].

4 Monitoring contract algorithms

There are two types of anytime algorithms: *interruptible* and *contract* algorithms. Interruptible algorithms may be interrupted after any elapsed running time so that they provide valid output. Contract algorithms do not give useful output values before the expiration of the *contract time*. Contract time is an allocation calculated before the activation of the algorithm. From contract algorithms, interruptible algorithms can be constructed, see [6] for further details.

Anytime monitoring is the process of distributing allocations for elementary algorithms so as to optimize the operation of the anytime system using information about the current state. The distribution itself is a part of the *scheduling process* which is responsible for the management of the executable code under timing considerations (like in real-time systems).

Anytime monitoring can be *passive* and *active*. Passive monitoring assigns allocation times *before* the activation of an anytime (sub-)system. Active monitoring assigns the allocation times *during* the execution of the modules and subsystems (therefore interruptible anytime operation applies active monitoring).

4.1 Output based incremental compilation

Let $M_1,...,M_N$ be a set of elementary contract modules characterized by CPPs and connected in some structure. C denotes the system composite (created by compiling all M_i-s into one composite). Assume one system output and a total system allocation obtained from the current state (e. g. by a utility driven computation) nominated as T. The purpose of a system is to perform certain tasks which is done by executing elementary operations in a well defined order called as *execution order*. These operations in our anytime system are the implementation functions of the anytime modules. The task is accomplished when the output value appears on the system output.

When the scheduler is about to execute the subsequent (actual) elementary module it has to compute an allocation for the module. This allocation depends only on the remainder of the modules to schedule and the input quality of the actual module, i. e. the remainder may be treated as an individual anytime module or rather a composite. This remainder is called as *residual composite*. The residual composite is used to obtain an allocation for it by the above-mentioned total allocation computations and then the allocation for the actual module is given by the compilation. This mode of scheduling requires the creation of these residual composites.

Let $\{E_1,...,E_N\}$ denote the execution order, formally a list of modules sorted by the order of activation. The

$$\bigcup_{i=1}^{N} E_i \equiv \bigcup_{i=1}^{N} M_i \tag{7}$$

expression is true, obviously.

If the system has only one output then compilation steps implemented by hierarchical compilation can be used to build residual composites. The first composite will be the last module in the activation list. The second composite will be the result of the last two modules in the list, etc. This compilation process is the *Output Based Incremental Compilation (OBIC)*, since the actual residual composite and the forthcoming module are compiled together at each compilation step. The order of modules is the reverse ordered activation list. Assuming a chain structure the OBIC and the scheduling method of active monitoring based on the OBIC can be formulated, as follows (the formulas would be too complicated for any other structure):

$$\begin{aligned} &T_i = T_{i,q,T}(q_{0,i}, T_{R,i}) \\ &T_{i,q,T}(q_{0,i}, T_{R,i}) \in C_{R,i} \\ &T_{R,i} = F_T[S, q_{R,i}(q_{0,i}, T_{R,i}), t_i, i, ...] \\ &t_1 = 0 \end{aligned} \tag{8}$$

where:

i :	Step variable, $i = 1..N$.
T_i :	Allocation for the module executed at the i-th step
$T_{i,q,T}(q_{0,i}, T_{R,i})$:	Time allocation table of the next module
$T_{R,i}$:	Residual allocation
F_T :	Function or algorithm giving the residual allocation
S :	Set of state parameters
t_i :	The elapsed absolute time from the system activation at the i-th scheduling step

$C_{R,i}$: The i-th residual composite

The formulas of qualities are:

$$\begin{aligned} & q_{0,i} = q_{i-1}(q_{0,i-1}, T_{i-1}) \\ & q_1(q_{0,1}, T_1) \equiv \begin{cases} q_1(T_1) & : P_1 \equiv PP \\ q_1(q_0, T_1) & : P_1 \equiv CPP \end{cases}, \\ & q_i(q_{0,i}, T_i) \in C_{R,i} \end{aligned} \tag{9}$$

where P_i is the performance profile of module E_i, and q_0 is the quality of the system input.

$$\begin{aligned} & C_{R,i} = C^{(H)}\{E_i, C_{R,i+1}\} \\ & C_{R,N} \equiv E_N \end{aligned}, \tag{10}$$

where $C^{(H)}\{\}$ denotes the hierarchical compilation operation.

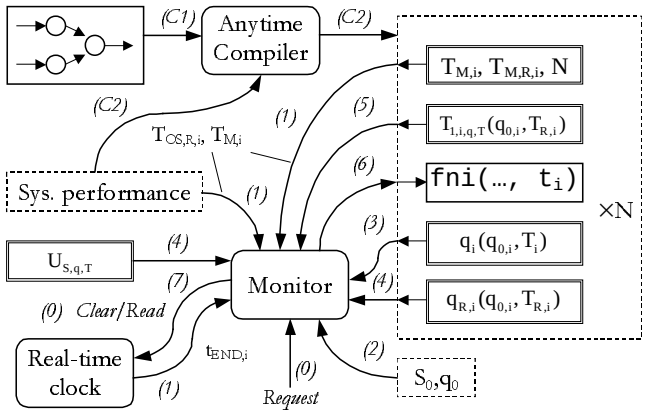

Figure 3: Block diagram of active monitoring

Figure 3 shows the block diagram of the monitoring system. The numbers in braces denote the sequence of the monitoring steps:

(0) Arrived request, real-time clock is set to zero.

(1) Obtaining the time needed by the monitor for the scheduling of the residual composite. Calculating the approximate values of the current state (using the initial values). Computing the time allocation needed by the operating system in the interval of executing the residual composite (e.g. interrupts).

(2) Determining the input quality of the first module to execute and the initial state values.

(3) Counting the input quality of the next elementary module by using the input quality and time allocation of the previously scheduled module (The value is simply read from the CPP of that module).

(4) Computing the optimal allocation ($T_{R,i}$) for the residual composite. (Utility driven computation may be used.) The input quality computed in step *(3)* is applied.

(5) Obtaining the allocation for the forthcoming elementary module by using the input quality and the residual allocation ($T_{1,i,q,T}(q_{0,i}, T_{R,i})$ is used.)

(6) Executing the implementation function of the module with allocation obtained by the previous step.

(7) Saving the current value of the real-time clock ($t_{END,i-1}$ for the next scheduling step).

5 Implementation by using development tools

As it can be seen in the previous chapters, the design of an anytime system has the main parts listed below:

1 (System specification/system decomposition) [D]
2 (Creation of anytime/non-anytime algorithms) [D]
3 Determination of the profiles [D] [C]
4 Compilation of the anytime algorithms [C]
5 Monitoring/scheduling [R]

The points surrounded by round brackets are out of the scope of this paper. It can be easily noticed that the points marked by [D] belong to the design process, mark [C] is for activities done by the Anytime Compiler which is a part of the Anytime Development Tool and is responsible for the creation of the run-time information database of the system used by the Monitor during the operation (see *(C1)* and *(C2)* in Figure 3). The Monitor operates in run-time [R], obviously.

Analyzing the operation of the anytime systems, it can be realized that on one hand, there is a gap between the Compiler and the Monitor in that sense that the compilation occurs before the activation of the system in contrast to the monitoring which works after the activation, during the process of the system operation. On the other hand, compilation and the monitoring are strongly related

together, a particular monitoring scheme requires certain compilation modes and methods.

The operation of the elementary modules is implemented by functions written in some programming language and the execution of these algorithms is started by function-call-like code fragments. The gap itself is caused by the simple fact that it is not trivial (or even impossible) to “dig out” the execution order from a source code, especially off-line. (E.g. if these function calls are located inside *if* or *for* structures.)

The way of the description of the anytime algorithms is not as easy in practice as it comes from the theoretical results and from the abstract handling of the profiles at first glance because the profiles are tables with certain sampling properties and dimensions. It is unlikely to have profiles with fitting dimensions and appropriate sampling intervals. Different profiles are valid for different definitions of the output qualities and they have to be adjusted properly for different target platforms. The complete solution of these problems is beyond this paper and under development.

A simple system architecture of an Anytime Development Tool shown in Figure 4 is presented in the followings.

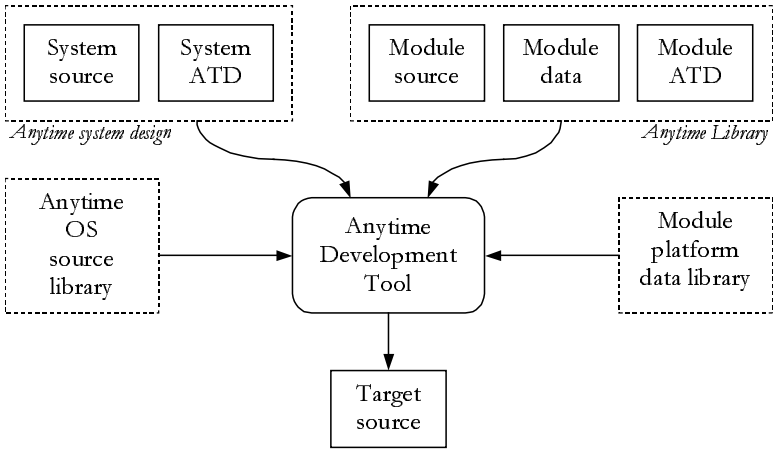

Figure 4: Block diagram of an anytime development architecture

5.1 Anytime Library

An anytime algorithm can be completely described by the following objects:

1. Source code(s) (written in certain language(s))
2. Profiles
3. Description

4 Other information

The source code can be written in several languages and compiled to intermediate files like object files or can be included into the source of the system code.

Profiles bring the essential information about the time dependent performance of the algorithms. Different platforms have different timing properties therefore the profiles have to be transformed. A reference computer had been suggested by Zilberstein which could be compared to a given platform in order to connect profile times with the real time. However, it is apparent that various possible execution times (allocations) of an algorithm are resulted from conditional commands in its code therefore choosing timing properties of a specific platform for parametrisation of the profiles is not the most efficient way. An algorithm can have only finite number of possible allocations and if the profiles would have the index of these allocation times as their time parameter then they would be platform independent. This allocation time index is called as *step time*. The quality in the profiles can be defined in several ways so a particular profile has to store its *quality function* (a mapping from output values to quality values or distribution) to keep all the information consistent.

The platform independent anytime algorithms can be collected in an Anytime Library which can provide basic or complex elements for the system design. Figure 5 shows what kind of data are needed to completely characterize an anytime algorithm.

Algorithm Characterization	
Source code(s): + language info., compiler/compilation options	
Profiles	profile type
	profile data
	quality function(s)
	source code of the profile maker fn.
Timer function source	
Module description (Module ATD)	
Other information	

Figure 5: Anytime algorithm characterization

The Module description is responsible for supporting the linkage between the pure source, the profile, and timing database (see later), and the target source. It can also define calling conventions of the implementation function. The *profile maker function* creates the profiles of the algorithm by executing it for appropriate inputs, determining the output qualities or distributions in the proper way, and collecting the performance data. The profile data is an array containing the samples of the mapping implemented by the profile.

5.2 Anytime system design

The Anytime system design collects all the results of the design process. It contains source code portions, the architectural description of the anytime system, and many option settings.

5.3 Anytime Operation System source library

This part of the development system contains template-like source code portions for the several types of anytime systems. It supports various monitoring schemes and an individual operation system can be created from it for the given anytime system design. This result is a set of compilable and/or linkable *target sources* and data files which are processed by an extant language compiler (C++ compiler, for example) to get an executable software as the final result of the anytime system design process.

5.4 Platform data

Platform data object is a collection of platform properties, mostly the timing data. This database is built by measurement performed by the Anytime Development Tool with the aid of the so-called *timer functions* (see Figure 5). This database is not necessarily created in compilation time and at every design iteration (step) since working with the same platform means the platform database is needed to be created only once.

5.5 Description of anytime systems

The Module ATD and System ATD sub-blocks in Figure 4 can be given in a meta-language. A simple example of a proposed a language is shown in this chapter. The AnyTime Description Language (ATDL) can be used to define modules, tasks, and systems. From the ATDL description a C++ code is generated which operates as an extension to the anytime operating system. This extension can be included in the operating system on source code level and results in the target source code. Thus, to each anytime system a separate anytime operating system is created.

Assume a simple anytime system shown in Figure 6.

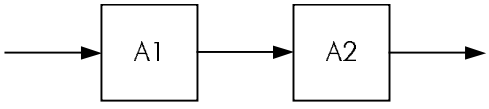

Figure 6: A simple anytime chain

The ATDL description of the block A1 can be in the form shown in Figure 7, where the profile is stored in a .prf file, the header (and included by the #include directive) of the module functions is "A1.h", the timing data are located in the file "A1.tim". Figure 8 shows an example of a header. Note that the algorithm's identifier (class) will be "MA1".

```
// A1.atd
module MA1(A1_INPUTSTRUCT input – double output)
{
      profile "A1.prf";
      header "A1.h";
      timing "A1.tim";
      implementation A1(input, output);
      profile_maker A1_profile_maker;
      timing_meter A1_timing_meter;
};
```

Figure 7: ATDL Example for an algorithm ATDL

```
// A1.h
#if !defined(_A1_H_INCLUDED_)
#define _A1_H_INCLUDED_
#include "atdefs.h"

      struct A1_INPUTSTRUCT
      {
            int a, b;
      };

      void A1(A1_INPUTSTRUCT &, double &, unsigned);
      CProfile* A1_profile_maker();
      CTiming* A1_timing_meter();

#endif
```

Figure 8: ATDL Example for an algorithm header

The request of the task consisting of the execution of the two modules can also be described by the ATDL. This task description determines the structure of the system. (Look at Figure 9.)

```
// T1.atd
task T1
{
      utility "T1.utl";
      include "A1.atd";
      include "A2.atd";

      MA1 a1;
      MA2 a2;

      connections a1.output – a2.input;

      external a1.input T1_input;
      external a2.output T1_output;
};
```

Figure 9: ATDL example for a Task

The two module definitions (A1 and A2) have to be included to instantiate the algorithm `a1` of type `MA1`, and the `a2` which is an `MA2` typed anytime algorithm. The single connection between the modules is defined by the `connections` keyword. The output and the input of the system can be referred by the identifiers `T1_input` and `T1_output`.

The system definition has the form shown in Figure 10. The system can run only one task, the `T1`.

```
system Sys
{
      include "T1.atd";
      header "Sys.inl";
      monitor active;
};
```

Figure 10: ATDL example for a system

Conclusions

An active monitoring method supported by the Output Based Incremental Compilation has been introduced in this paper. The basics of a development tool and description language have also been proposed. These tools are capable to operate systems consisting of anytime algorithms characterized by conditional performance profiles and arranged in a directed acyclic graph. Such systems can very advantageously be used when the resource and/or data availability is insufficient or changing during the operation. The OBIC is based on the theoretical results of hierarchical compilation.

Acknowledgement

This work was sponsored by the Hungarian Fund for Scientific Research (OTKA T 035190).

References

[1] S. Zilberstein, Operational Rationality through Compilation of Anytime Algorithms, Dissertation for the degree of doctor of philosophy in computer science, 1993.

[2] Samu, G., Várkonyi-Kóczy, A. R., "Intelligent Monitor for Anytime Systems", In Proc. of the IEEE int. Symposium on Intelligent Signal Processing, WISP'2003, Budapest, Hungary, Sep. 4-6, 2003.

[3] J. Grass, S. Zilberstein, "Anytime Algorithm Development Tools", UM-CS-1995-094, 1995.

[4] E. A. Hansen, S. Zilberstein, "Monitoring the Progress of Anytime Problem-Solving", *Proceedings of the Thirteenth National Conference on Artificial Intelligence* (AAAI-96), 1996.

[5] E. A. Hansen, S. Zilberstein, "Monitoring and control of anytime algorithms: A dynamic programming approach", *Artificial Intelligence 126* (2001) 139–157.

[6] S. Zilberstein, F. Charpillet, P. Chassaing, "Optimal Sequencing of Contract Algorithms", *Annals of Mathematics and Artificial Intelligence*, 2002.

[7] S. Zilberstein, "Optimizing Decision Quality with Contract Algorithms", *14th International Joint Conference on Artificial Intelligence*, Montreal, Canada, aug. 1995.

[8] Garvey, V. Lesser, "Design-to-time Scheduling with Uncertainty", *SIGART Bulletin*, jan. 9, 1995.

[9] Garvey, V. Lesser, "A Survey of Research in Deliberative Real-Time Artificial Intelligence", *UMass Computer Science Technical Report* 93–84, nov. 19, 1993.

[10] S. Zilberstein, S. Russell, "Approximate Reasoning Using Anytime Algorithms", In S. Natarajan, editor, *Imprecise and Approximate Computation*. Kluwer Academic Publishers, Dordrecht, 1995. 11.

[11] S. Zilberstein, S. Russell, "Optimal Composition of real-time systems", *Artificial Intelligence*, 82(1-2):181--213, 1996.

[12] M. Boddy, T. L. Dean, "Solving time-dependent planning problems", *In Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, Michigan (1989) 979-984.

[13] Labrosse, J. J. (1992-1995), "uC/OS, The Real-Time Kernel", source codes. R&D Books, 1992., ISBN 0-13-031352-1.

[14] Waite, M. W.,Goos, G. (1984), "Compiler Construction", ISBN 0-387-90821-8.