In Proc. of the IEEE International Conference on Computational Cybernetics, ICCC'2003, Siófok, Hungary, Aug. 29-31, 2003, pp. 205-210

Anytime System Scheduler for Insufficient Resource Availability

Annamária R. Várkonyi-Kóczy
Department of Measurement and Information Systems
Budapest University of Technology and Economics
H-1521, Magyar Tudósok krt. 2.
Hungary
koczy@mit.bme.hu

Gábor Samu
Department of Measurement and Information Systems
Budapest University of Technology and Economics
H-1521, Magyar Tudósok krt. 2.
Hungary
sg222@hszk.bme.hu

Abstract – **Anytime systems can very advantageously be used when the resource and/or data availability is changing during the operation and some kind of intelligent reconfiguration of the system is needed to cope with the temporal data and resource access conditions. These schemes may provide an optimal tradeoff between the time/resource need and computational complexity and the quality (accuracy) of the results. Such systems can be achieved by using special types of models, methods, and algorithms together with applying active monitoring being able to supervise the operation of the system on-line and making intelligent decisions based on the sensory information of the so-called shortage indicators. Since the monitor operates at prescribed response time requirements and the number and complexity of the executable tasks can be very high especially in case of complex systems new considerations are needed to achieve optimal or acceptable performance. At computer level it means to application of special compilation methods dealing also with timing considerations and constraints of the underlying operating system and even their own run-time characteristics. This can 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 ATDL anytime description meta-language are presented.**

Keywords – Anytime systems, anytime compilation, hierarchical compilation, active monitoring, anytime development platform, anytime description language.

I. 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 between certain limits. This behavior assumes varied information processing tasks solved by a computer at 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 seems to be unavoidable even in the case of extremely careful design to get into situations where the shortage of necessary data and/or processing time becomes serious. Such situations may result a critical breakdown of the monitoring and/or diagnostic systems [1]. The concept of "anytime" processing tries to handle the case of too many abrupt changes and their consequences in larger scale embedded systems [2]. The idea is 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 the overall performance "at lower price", i.e., information processing based on algorithms and/or models of simpler complexity should provide outputs of acceptable quality to continue the operation of the complete monitoring system. The accuracy of the processing will be temporarily lower but possibly still 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 at prescribed response time requirements and the number and complexity of the executable tasks can be very high especially in case of complex systems new considerations are needed to achieve optimal or acceptable performance. At computer level it means to application of special compilation methods dealing also with timing considerations and constraints of the underlying operating system and even their own run-time characteristics.

Complex systems usually consist of smaller subsystems which can be viewed as elementary modules. For complex anytime systems, optimization means, among others, the distribution of the time allocations to the above-mentioned elementary modules (computational elements) with respect to the optimization criteria of the whole system. For this, 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 the properties. Anytime development tools and special anytime description languages can also support these tasks.

In this paper a hierarchical compilation method and theoretical considerations about a possible anytime development tool and the basics of the ATDL anytime description meta-language are introduced. The paper is organized as follows: Section II. gives an overview about basic terms of anytime computation. In Section III. the compilation methods of anytime algorithms are discussed briefly. Section IV. And V. summarizes the main ideas and operation of hierarchical compilation. Section VI. describes the theoretical basics of an anytime development tool and the ATDL anytime description languages is proposed.

II. 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

TABLE I: 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

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 [4], [5]).

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 *distribution* of probability. The output quality is the parameter of this distribution and thus of the profile itself too. 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). The different profile types are summarized in Table I. (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 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.

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.

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, for details of dynamic profiles and time-dependent planning under uncertainty see [4], [6], [7] and [8].

III. COMPILATION OF ANYTIME ALGORITHMS

Complex anytime systems (like other complex systems) can be decomposed into 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 may 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 behavior 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.

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.

It is obvious that those modules that 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 systems 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, the second module (with index 2) is connected after the first one, q denotes the conditional performance profiles, and T stands for the time allocations.

The composite profile is constructed so that for every composite allocation and composite input quality the elementary allocation pair with 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:

$$\begin{aligned} &q(q_{in},T)[q_{out}]= \\ &=\arg\max_{E}\max_{T_1} E\{p(q_{in},T_1,T-T_1)[q_{out}]\} \\ &p(q_{in},T_1,T_2)[q_{out}]= \\ &=\sum_{i=1}^{N_{q_1}}\left(p_1(q_{in},T_1)[q_{1out,i}]\cdot\right. \\ &\left.\cdot\, p_2(q_{1out,i},T-T_1)[q_{out}]\right) \end{aligned} \tag{3}$$

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.

In this paper compilation with CPPs will be discussed only.

The above method described for to modules 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.

Assume a system Y with N number of 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:

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

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 formulated as

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

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

By using SISO modules only chain structures can be created. More general, tree structured systems can be made of multiple-input-single-output (MISO) modules (see Fig.1.). The compilation of these tree-structured systems is effectively solved by the *local compilation* method introduced by Zilberstein ([4]) and discussed in the next subsection.

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 input quality
3. System has a tree structure with bounded degree
4. CPDPs have the input linearity property

Proof is given in [4]. Tree and chain structured anytime systems are discussed briefly in [9].

The example on Fig. 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* what is 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 a 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 different structure 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.

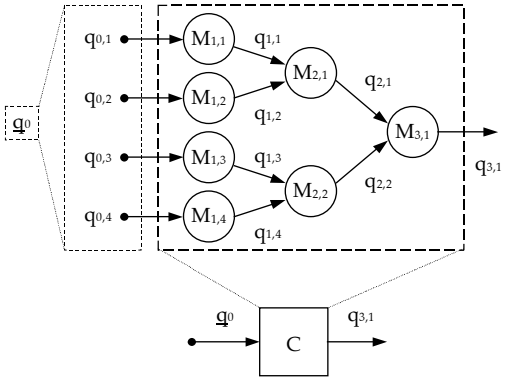

Fig. 1: Tree structured system

However 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 hierarchical compilation and the output based incremental compilation.

IV. HIERARCHICAL COMPILATION

Selecting a subsystem *S* with *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 compiled composite is called *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 time and input quality dimensions).*
3. *Modules described by CPPs.*

Hierarchical compilation provides global optimum beside the assumptions above (for the proof of this theorem, see [10]) and it means the following:

Let system *Y* divide into two parts: a MISO subsystem *S* and a remainder *R* (see Fig. 2.). Thus the number of modules in *Y* (N) is divided into two sets: $M_{S,1}, ..., M_{S,NS}$ are in *S* and $M_{R,1}, ..., M_{R,NR}$ are in *R*. $NR + NS = N$ is true, obviously. The time allocations for modules in S and R are $T_{S,1}, ..., T_{S,NS}$ and $T_{R,1}, ..., T_{R,NR}$, respectively. The global compilation of *Y* is:

$$q^{(G)}(\underline{q}_0, T) = \\ = \max_{T_1, ..., T_{N-1}} q\left(\underline{q}_0, T_1, ..., T_{N-1}, T - \sum_{i=1}^{N-1} T_i\right), \quad (7)$$

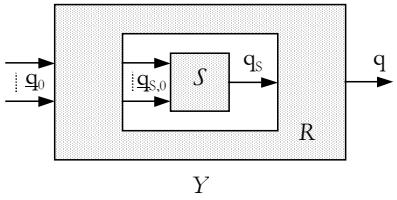

Fig. 2: Hierarchical compilation

By separating the allocations of *S* and *R*:

$$q^{(G)}(\underline{q}_0, T) = \max_{T_{S,1},...,T_{S,NS-1},T_{R,1},...,T_{R,NR}} q\left(\underline{q}_0, T_{S,1},..., T_{S,NS-1},, T - \sum_{i=1}^{NS-1} T_{S,i} - \sum_{i=1}^{NR} T_{R,i}, T_{R,1},..., T_{R,NR}\right) \quad (8)$$

where T_S is expressed by the other allocations. Hierarchical compilation of *Y* using *S* is:

$$q^{(H)}(\underline{q}_0, T) = \max_{T_{R,1},...,T_{R,NR}} q\left(\underline{q}_0,..., q_S^{(G)}\left[\underline{q}_{S,0}, T - \sum_{i=1}^{NR} T_{R,i}\right],, ,..., T_{R,1},..., T_{R,NR}\right) \quad (9)$$

where T_S is expressed by T and T_R-s. Expand (9) using (8):

$$q^{(H)}(\underline{q}_0, T) = \max_{T_{R,1},...,T_{R,NR}} q\left(\underline{q}_0,..., \max_{T_{S,1},...,T_{S,NS-1}} q_S\left[\underline{q}_{S,0}, , T_{S,1},..., T_{S,NS-1}, T - \sum_{i=1}^{NR} T_{R,i} - \sum_{i=1}^{NS-1} T_{S,i}\right], ,..., T_{R,1},..., T_{R,NR}\right) \quad (10)$$

q and q_S are monotonic non-decreasing functions of their arguments, every $T_{S,i}$ is in the inner maximum function, and $\underline{q}_{S,0}$ depends on $T_{R,i}$-s only.

$$q^{(H)}(\underline{q}_0, T) = \max_{T_{R,1},...,T_{R,NR}} \max_{T_{S,1},...,T_{S,NS-1}} q\left(\underline{q}_0,..., q_S\left[\underline{q}_{S,0}, , T_{S,1},..., T_{S,NS-1}, T - \sum_{i=1}^{NR} T_{R,i} - \sum_{i=1}^{NS-1} T_{S,i}\right], ,..., T_{R,1},..., T_{R,NR}\right) \quad (11)$$

Merging the maximum functions and eliminating the decomposition the formula below can be obtained:

$$q^{(G)}(\underline{q}_0, T) = \max_{T_{S,1},...,T_{S,NS-1},T_{R,1},...,T_{R,NR}} q\left(\underline{q}_0, T_{S,1},..., T_{S,NS-1},, T - \sum_{i=1}^{NS-1} T_{S,i} - \sum_{i=1}^{NR} T_{R,i}, T_{R,1},..., T_{R,NR}\right) \quad (12)$$

V. 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 don't 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 [7]).

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* that 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).

Let $M_1,..., M_N$ be a set of elementary contract modules characterized by their 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 be scheduled 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 *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

$$\bigcup_{i=1}^{N} E_i \equiv \bigcup_{i=1}^{N} M_i , \quad (13)$$

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 called *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):

$$\begin{aligned} &T_i = T_{i,q,T}(q_{0,i}, T_{R,i}) \\ &T_{i,q,T}(q_{0,i}, T_{R,i}) \in C_{R,i} \\ &T_{R,i} = F_T[S, q_{R,i}(q_{0,i}, T_{R,i}), t_i, i, ...] \\ &t_1 = 0 \end{aligned} \quad , \tag{14}$$

where:

i :	Step variable, $i = 1..N$.
T_i :	Allocation for the module executed at the i-th step
$T_{i,q,T}(q_{0,i}, T_{R,i})$:	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:

$$\begin{aligned} &q_{0,i} = q_{i-1}(q_{0,i-1}, T_{i-1}) \\ &q_1(q_{0,1}, T_1) \equiv \begin{cases} q_1(T_1) & : P_1 \equiv PP \\ q_1(q_0, T_1) & : P_1 \equiv CPP \end{cases} \\ &q_i(q_{0,i}, T_i) \in C_{R,i} \end{aligned} \quad , \tag{15}$$

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

$$\begin{aligned} &C_{R,i} = C^{(H)}\{E_i, C_{R,i+1}\} \\ &C_{R,N} \equiv E_N \end{aligned} \quad , \tag{16}$$

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

VI. ANYTIME DEVELOPMENT ENVIRONMENT AND THE ATDL ANYTIME DESCRIPTION LANGUAGE

There is a well recognizable "gap" between the compilation (fitting) and the monitoring. From system point of view the previous can be best down in compilation time while the latter should be executed during run time. The problem is that for the correct compilation we need a lot of information about the modules that are not trivial to "dig out" from the source files (execution order, etc.). The result of the anytime development has to be an executable file written in a given programming language but on source code level. This can not be simply fulfilled in case of the high level languages. The solution can be that from the anytime algorithms, from their description, and from the system source code the anytime development tool prepares a so called *target source code* which is compiled to an executable file by a special compiler. This is supported by a meta-language which helps the anytime development tool to connect the anytime algorithms and their description tables stored in files.

For the above purposes, a simple AnyTime Description meta-Language (ATDL) is proposed in this paper which makes possible the full description of anytime systems in cooperation with the C++ high level programming language. 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. The block diagram of this architecture is shown in Fig. 3.

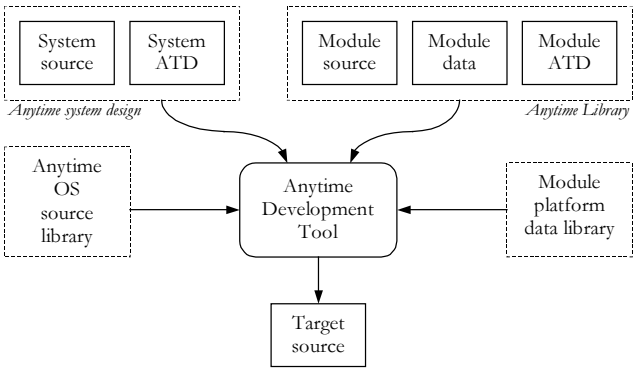

Fig. 3: Block diagram of the architecture of anytime development

The anytime system design block covers the source code of the system, where the system features and properties are given in ATDL language. The anytime library contains the source codes, profiles of the anytime algorithms, and their ATDL descriptions. The anytime platform data library includes the timing data. The anytime operation and the monitor is described in the anytime OS source library. The anytime development tool ensures the development environment. The target source is the result of the development (compilable C++ source and data files).

In the followings as examples, parts of a system, a subtask, and a module description are given in Figs. 4-6. to illustrate the simplicity and plasticity of the proposed ATDL language. The **system**, **task**, and **module** commands define the anytime system, task-units, and anytime modules, respectively. Tasks can be included in the system with the help of the **include** command. With the **monitor** command the type of monitoring (**active**, **passive**) can be set and the **header** points to the header file of the system. The utility function file can be pointed by the **utility** command. The connections among the modules can be defined by the **connection** command. **External_use** means that the given inputs and outputs are the external inputs and outputs of the task-unit. The **profile**, **header**, and **timing** elements give the corresponding file names. The **implementation** is followed by the interface of the implementation function of the algorithm. The profile functions can be given after the **profile_maker** or **maker** commands.

The ATDL code parts can easily be merged with C++ codes resulting in an anytime operating system.

VII. CONCLUSION

An active monitoring method supported by 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 anytime 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 output based incremental compilation is based on the theoretical results of hierarchical compilation.

ACKNOWLEDGMENT

This work was sponsored by the Hungarian Fund for Scientific Research (OTKA T 035190) and the I^2S Integrated Intelligent Systems Japanese-Hungarian Laboratory.

REFERENCES

[1] Várkonyi-Kóczy, A.R., T. Kovácsházy, "Anytime Algorithms in Embedded Signal Processing Systems," *In Proc. of the IX. European Signal Processing Conference, EUSIPCO-98*, Rhodes, Greece, Sep. 8-11, 1998, Vol. 1, pp. 169-172.

[2] Baron,C., J.-C Geffroy, G. Motet ed., *Embedded System Applications*, Kluwer Academic Publishers, 1997.

[3] Várkonyi-Kóczy, A.R., A. Ruano, P. Baranyi, O.Takács, "Anytime Information Processsing Based on Fuzzy and Neural Network Models," *IEEE Trans. on Instrumentation and Measurement*, accepted for publication.

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

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

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

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

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

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

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

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

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

[13] Zilberstein, S., "Optimizing Decision Quality with Contract Algorithms," *In Proc, of the 14th International Joint Conference on Artificial Intelligence*, Montreal, Canada, Aug. 1995.

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

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

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

Fig. 4: System definition: ATDL source code-part

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

      MA1 a1;
      MA2 a2;

      connections a1.output - a2.input;

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

Fig. 5: Task definition: ATDL source code-part

```
// 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;
};
```

Fig. 6: Module definition: ATDL source code-part