524-61
136898

N93-17523

P4

RT-Syn: A Real-Time Software System Generator

Dorothy E. Setliff
Electrical Engineering Department
University of Pittsburgh
Pittsburgh, PA 15261

Abstract

This paper presents research into providing highly reusable and maintainable components by using automatic software synthesis techniques. This proposal uses domain knowledge combined with automatic software synthesis techniques to engineer large-scale mission-critical real-time software. The hypothesis centers on a software synthesis architecture that specifically incorporates application-specific (in this case real-time) knowledge. This architecture synthesizes complex system software to meet a behavioral specification and external interaction design constraints. Some examples of these external constraints are communication protocols, precisions, timing and space limitations. The incorporation of application-specific knowledge facilitates the generation of mathematical software metrics which are used to narrow the design space, thereby making software synthesis tractable. Success has the potential to dramatically reduce mission-critical system life-cycle costs not only by reducing development time, but more importantly facilitating maintenance, modifications and extensions of complex mission-critical software systems which are currently dominating life-cycle costs.

1 Introduction

The software development process is time consuming, expensive, and fraught with errors. These characteristics hinder the reuse of software. This paper presents an approach that seeks to reduce software development time while simultaneously increasing software reuse. This approach, called RT-Syn, uses characteristics of the software domain to synthesize software. Although there is a wealth of structural and syntactic knowledge that can be brought to bear for software synthesis, the lack of success in generalized software synthesis [15] argues that this knowledge is insufficient, and that domain-specific knowledge is necessary for successful software synthesis.

The chosen domain is real-time software. Software for real-time applications can be characterized as a set of time-constrained tasks. Real-time system failure is defined as when any task misses its hard deadline. Software development and subsequent redevelopment is one of the major bottlenecks of real-time system design and maintenance. One method to remove this bottleneck is to automatically synthesize real-time software to meet all system platform and task timing requirements. We argue that the presence of strict operation requirements, such as task-level timing constraints and compiler and target platform constraints, can be used to guide synthesis.

RT-Syn builds off of previous work by Setliff [11, 12], Barstow [1], and Kant [9] among others. Generality is supported by the use of a visual language as an algorithm specification. Only that application-specific knowledge relevant to programming-in-the-small synthesis is currently incorporated within the RT-Syn synthesis architecture. Within the vernacular of this particular application domain, this current project focuses on task-level synthesis issues. Future work encompasses system-level (programming-in-the-large) synthesis issues.

Section 2 reviews the current RT-Syn 1.0 architecture and presents a brief overview of each component within RT-Syn 1.0, including an enumeration of what knowledge is required to perform task-level synthesis and how that knowledge is represented. Section 3 describes RT-Syn 2.0, an approach to system-level software synthesis. This section describes the various components envisioned within RT-Syn 2.0 and their interactions. Finally, Section 4 presents an overview of our progress and a plan for future work.

2 RT-Syn 1.0 Architecture

This section describes the architecture of our initial version of RT-Syn called RT-Syn 1.0. This version incorporates knowledge of the effects of a specific platform, the DLX processor [4], on task-level synthesis, as well as an algorithm that uses this knowledge to formulate timing and space requirements estimates for different implementation possibilities for a task. We chose the simulated DLX processor for this initial version because it is completely modelled and DLXsim (a program which models the operation of the DLX processor) keeps statistics useful for checking our design decisions. Current work in predictable platforms [18] seeks to design and develop real-world platforms which exhibit the similar predictable characteristics exhibited by the DLX processor.

RT-Syn 1.0 integrates platform characteristics to synthesize a real-time software task to meet hard deadline requirements. The RT-Syn 1.0 automatic synthesis system has five key features. First, a visual graphical user interface, called Intuition, captures application algorithms without implementation specifications. Second, RT-Syn 1.0 analyzes the task-level data and control flows to produce worst-case timing and space predictions. Third, RT-Syn 1.0 uses these predictions to select abstract representations of data structures and algorithm implementations to meet required timing and space constraints. Fourth, RT-Syn 1.0 synthesizes C code from the selected implementations. Fifth, RT-Syn 1.0 validates the execution of the code to meet the predicted timing and space utilization values.

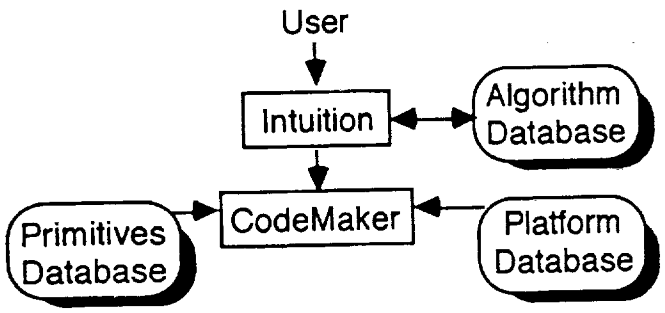

Figure 1: RT-Syn 1.0 Synthesis Architecture

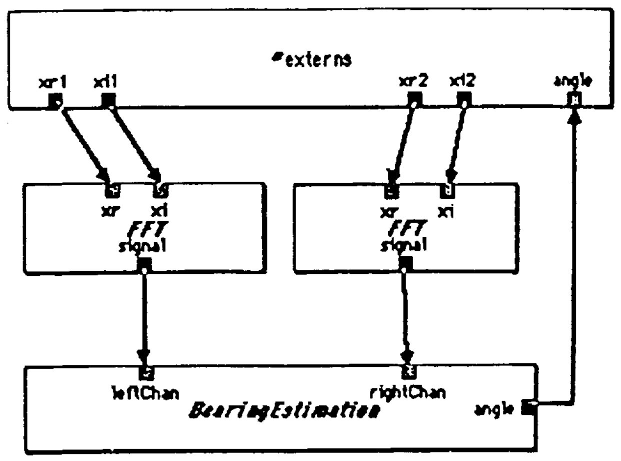

Figure 2: Representative Intuition Algorithm Representation

RT-Syn 1.0 (See Figure 1) is composed of several databases and two tools: Intuition, a graphical programming language, and CodeMaker, a real-time software directed program synthesis system.

A visual programming language is a way to program a system using a visual paradigm. Visual programming languages vary in the level of represented detail [5]. Intuition provides a semantic-level, user-friendly, hierarchical visual programming environment. This environment offers several advantages over text-based environments. It allows for the user-friendly input of RT-Syn specifications with minimum effort. Its hierarchical nature allows the user to easily modify the description. There is also less learning time involved. The concepts behind these advantages are present in a variety of existing visual programming research efforts [2, 3, 10, 13, 14].

Intuition acts as a user input system as well as a high level visual algorithm description. Intuition descriptions capture both data flow and control flow information. Intuition is similar in form and function to the popular object-oriented drawing program that exists on personal computers, such as MacDraw. The user has a palette of tools, and a window in which to draw a schematic representation of algorithms. Figure 2 is an example of a Intuition screen representation of a FFT signal processing algorithm. Each rectangular cell on the screen represents a basic building block of the algorithm. The building block may be direct computation or a hierarchical reference to another Intuition file representing quantities of computations. Data and control flow is captured by the connecting lines between the cells. There is no data typing or language representation implied within Intuition. Groups of Intuition algorithmic representation formulate the Algorithm Database. This database organizes the algorithms by function and by hierarchical components.

The cornerstone of the RT-Syn 1.0 schema is a program synthesis system which is capable of producing extremely well-modelled executable code. CodeMaker, a program synthesis system targeting real-time signal processing software, guarantees the validity of the generated code using the following methodology. First, CodeMaker analyzes all algorithmic representations available within Intuition that can result in the required behavior and produces an internal representation of the algorithmic data and control flow. Second, CodeMaker uses knowledge of the platform to produce implementation ranges for each specifiable resource (currently, speed and space). All implementation possibilities for a particular task are guaranteed to be within the implementation range. Third, CodeMaker analyzes the implementation ranges to select the algorithmic approach most likely to satisfy all of the required specifiable resource quantities for a particular task. Fourth, CodeMaker then synthesizes an implementation of the algorithm that is guaranteed to meet the required specifications.

The following example illustrates the operations within CodeMaker, what knowledge is applied, and how knowledge is applied. In this example, CodeMaker is directed (via Intuition) to synthesize a FFT algorithm that executes in X cycles and may consume no more than Y bytes of data memory. There exist numerous algorithms that exemplify the FFT behavior [6]. Each algorithm, regardless of the implementation, places a differing strain on computation and memory resources. CodeMaker must select a particular algorithm and then synthesize an implementation of the algorithm to specifically meet the requirements for that task.

CodeMaker first analyzes all algorithmic possibilities and produces a control and data flow graph for each. The implementation ranges are formed by noting a fairly simple fact. Timing limitations generally adversely affect space utilization (less time requires greater space requirements). To discover the minimum speed characteristic for any algorithm, synthesize the algorithm while attempting to reach a 'time = 0' constraint. Of course, this is impossible so synthesis will fail, but the resultant implementation specification will be the closest to zero and thus the minimum time and maximum data memory. Setting a 'space = 0' constraint and synthesizing to meet that constraint find the implementation that uses the least data memory and maximum time. Program memory requirements are not considered currently in CodeMaker. Each algorithm that satisfied the required behavior is synthesized twice to produced implementation ranges for that algorithm. The synthesis process incorporates knowledge of the platform and maintains any dependencies within an algorithm. This approach is used to prune out those possibilities that will fail early on in the process. The algorithm with a design space closest to the resource requirements is selected. It is important to note that the design space is not convex. There exist points in the design space that are not reachable. The synthesis process is capable of backtracking back to this point if the requirements are not attainable during the final synthesis process.

Synthesis of an implementation requires access to specific platform-dependent data. The Platform Database contains modelling information needed to generate accurate predictions about the timing and space utilization characteristics of tasks for a given platform. The bulk of the information in this database consists of data about primitives. Primitives are the lowest-level building blocks used by RT-Syn when synthesizing C code. Typically, primitives represent single C operations, such as addition, multiplication, and branching operations. Information stored in the platform database includes: the timing and space characteristics of all primitives on this platform, formula to account of eccentricities in the behavior of the model of the platform, characterization of any operating system calls that will be used, and a characterization of the C compiler to be used when generating executable code. The behavior of platforms and compilers vary widely. As a basic example, some computers utilize a separate floating-point math unit, which makes floating-point math a faster alternative to integer math. At a compiler level, different compilers perform different optimization techniques, so that the same piece of C code compiled on two different compilers and then run on the same platform will have different characteristics. An important feature of the RT-Syn system that is a result of the Platform Database is that a given set of tasks can be completely resynthesized for different platforms by changing only the platform selection.

CodeMaker uses the data and control flow graphs, in tandem with knowledge of the tradeoffs requisite in a particular target platform (e.g., operating system, hardware) and target language primitives, to select implementations. The selection process is performed bottom-up (or correspondingly output-back-to-input) on the data flow graph. This graph walking approach specifically acknowledges the impact of external requirements (the required outputs) on the implementation selections. In this way, the external requirements are applied as early as possible and are used to reject portions of the design space that are unworkable. Only that knowledge pertinent to the task-level prediction and selection is required. Analysis, prediction, then selection continues until the task implementation is fully specified. At this point, the implementation code is constructed.

Experimentation using Intuition and CodeMaker [16] demonstrate both the efficient synthesis (100's of lines of code is less than 30 seconds on a MAC II) possible when application-specific knowledge is incorporated within synthesis. Numerous experimentation [16, 17] also shows the range of implementations attainable merely by modifying the task-level constraints. These experiments show the power of using application-specific knowledge to synthesize software to meet a set of specifications and thereby provide for software reusability. Only knowledge of platforms and primitives particular to data structure and algorithm selection is incorporated and used within CodeMaker. This knowledge prunes the design space, while providing solutions for time and space constrained signal processing tasks.

RT-Syn 1.0, a real-time task set synthesis architecture, synthesizes viable C code solutions based on a user's high-level task set specification. RT-Syn 1.0 takes as input a high-level description of the real-time tasks to be generated, along with information about the equipment on which the system will be run and generates code to implement each task. The code is guaranteed to meet any resource use requirements. The next section describes work in progress towards system-level synthesis using the successes of the RT-Syn 1.0 system as a foundation.

3 Towards System-Level Synthesis

RT-Syn 2.0 focuses on system-level synthesis. RT-Syn 2.0 performs all of the synthesis provided by RT-Syn 1.0 and also generates a scheduler to coordinate the execution of the individual tasks. The entire system will be guaranteed to produce the desired outputs within the given deadlines and without exceeding the host equipment's resources.

The RT-Syn 2.0 real-time software synthesis architecture is shown in Figure 3. Each block in the diagram represents a component in the synthesis architecture. There are two types of components in the synthesis architecture: design and database components. We first introduce each component, then describe the advantages of this architecture. We then describe in detail the functionality of each component

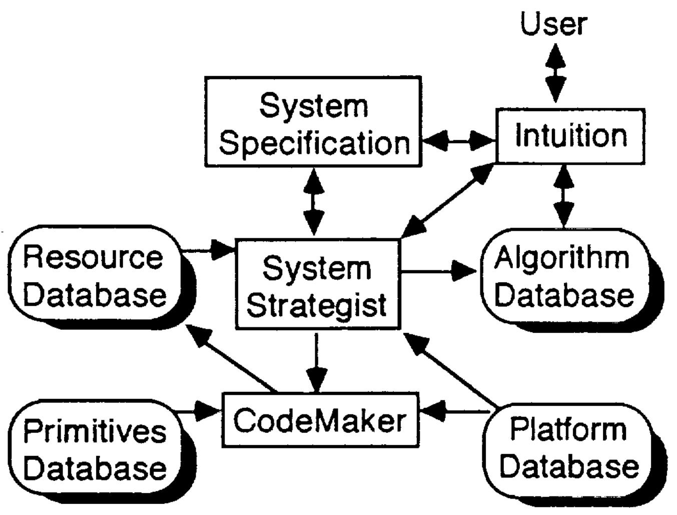

Figure 3: RT-Syn 2.0 Synthesis System

in the order that synthesis follows: from user specification of a set of tasks to the successful synthesis of code for each task. The software synthesis process is not monolithic; rather, the synthesis process is composed of a succession of abstraction levels. Breaking the software synthesis architecture into the various abstraction levels illustrates the design hierarchy. Each design abstraction level is composed of a design component with database components as required. The highest abstraction level is the user interaction with the synthesis architecture. The System Specification component interacts with the user to form a system-level behavior description. The user interacts with the synthesis architecture via Intuition.

The second abstraction level decomposes the system behavior descriptions into task level specifications. The System Strategist component analyzes the system-level behavior description, determines what tasks are required, how the tasks are to be scheduled, and how resources are to be allocated to each task. All of these operations require knowledge. This knowledge is captured within three database components: the Resource Management Database, the Algorithm Database, and the Platform Database. The Resource Management Database contains knowledge of current system resource allocations and results from prior software synthesis operations. This information is used within the System Strategist to aid the design and synthesis process. The Algorithm Database contains knowledge about a variety of useful algorithms and the methods in which they are implemented. Each algorithm in this database is represented using Intuition. The Platform Database contains modelling information for various computer/compiler platforms. This information is used in the analysis tools within the System Strategist component and CodeMaker tool.

The third abstraction level decomposes the task level specifications into code implementations. This abstraction level has already been developed and discussed in a prior section of this proposal.

There are several advantages to this architecture. The decomposition via abstraction level provides a design focus and limits the amount of design search. The decomposition also closely mirrors the current development process of a system analyst and programmer. By mirroring this development process, it is possible to allow both system analysts and programmers to interact with the synthesis architecture. The distinction between design and database components provides a growth mechanism.

Isolating the information into three distinct databases facilitates the expansion of RT-Syn 2.0 to work with a variety of systems. Extending the knowledge of the Algorithm Database enables the system to synthesize a wider array of tasks. Adding information about more systems to the Platform Database allows RT-Syn 2.0 to model new hardware/compiler platforms. Finally, by updating the Primitives Database, RT-Syn 2.0 gains the ability to synthesize code in different languages.

The System Specification component serves three functions. The first function is to enable the user to specify the set of tasks to be synthesized. The user specifies a type of task set (out of a list of tasks of which the system has knowledge), provides information about the number of and type of inputs and outputs to the set of tasks, and chooses a target platform (from a list of machines for which there exist timing models). From this information RT-Syn 2.0 constructs a suite of tasks which will perform the desired function and run within the confines of the target platform's constraints. The output is information to the System Strategist about what tasks are required and what platform is being used. This task specification information can be very coarse-grained (e.g., system-level input) or extremely detailed (e.g., implementation-level input), depending on the level of user interaction. The second function of the System Specification component is to interact with the user during the design and synthesis process. Interaction takes the form of behavior simulation. In this way, the user can validate that the set of tasks given in the input do indeed satisfy the required mathematical functionality before synthesizing to meet the needs of the application.

The Resource Management Database provides information about system resource allocation results to the System Strategist. As individual tasks are synthesized, the amount of resources required will solidify from coarse estimates to accurate predictions. During the synthesis process, the Resource Management Database is updated to reflect the current state of a task's resource allocation needs. These needs are represented as the set of task descriptors Ci, Ti and Ui (which represent worst-case execution time, execution period, and utilization, respectively) [8, 7].

The System Strategist acts as the systems analyst for the synthesis of the set of tasks. Its initial function is to determine what task implementations will be used in the desired system and to choose and develop a real-time scheduler to manage these tasks. To choose a scheduler the System Strategist must have knowledge of a variety of scheduling schemas and have heuristics for deciding which one is most applicable to the current situation. These heuristics consist of rules derived from real-time scheduling theory. The following describes the algorithm database and provides a detailed discussion of the operations the System Strategist performs once the synthesis process is underway.

There are two main characteristics of the Algorithm Database: organization and database entry contents. The algorithms are organized hierarchically by behavior. Figure 4 illustrates the organization of the Algorithm Database:

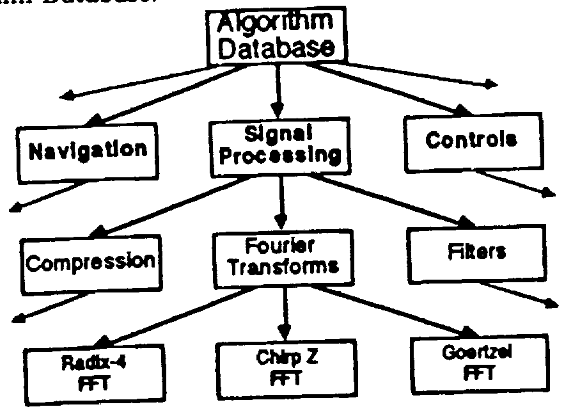

Figure 4: Design Organization of the Algorithm Database

The advantage of this organization is the reduction of search required to find all algorithms with a certain behavior. Organization by behavior places each algorithm in a specific behavior class. Each entry in a particular class possesses the identical behavior but differs in the corresponding functionality. A behavior hierarchy corresponds to the desired system-level user interaction. A typical system level specification consists of a set of behavior descriptions. These behavior descriptions match particular class and subclass behaviors in the algorithm hierarchy. All algorithms corresponding to the specified behavior may be immediately retrieved for analysis. Each entry in the database is a 3-tuple: algorithm representation, resource constraint ranges, synthesis history. Each algorithm is represented in Intuition. Each entry contains coarse-grained resource characteristics. These coarse-grained characteristics define the implementation design space for that algorithm entry. Currently the Algorithm Database contains time and space resource characteristics.

The System Strategist analyzes the system-level behavior description as a set of tasks, schedules each task, and allocates resources to each task in the system. The input to the System Strategist is a system-level behavior description. The output is a system scheduler algorithm selection, and a set of task-level descriptions. A task-level description details the behavior algorithm and desired time and space characteristics for each task. Analysis is required to synthesize the scheduler algorithm and task-level descriptions. The System Strategist first accesses the implementation range information in the Algorithm Database for each task in the system. The set of implementation ranges defines the design space that the scheduling algorithm must guarantee. There is an enumerable set of scheduling algorithms. The System Strategist attempts to synthesize all known scheduling algorithms. The System Strategist applies the underlying mathematics of each scheduling algorithm to the set of implementation ranges. A task specification is chosen within each range that will result in guaranteed schedulability. The set of task specifications may not overutilize system memory constraints. A potential scheduling algorithm is removed from consideration (pruned) when it fails to guarantee the task specification set.

Scheduling algorithm analysis iterates with the synthesis of each task in the system. Task synthesis results in more exact information than the implementation range information in the Algorithm Database. More exact information prunes the design space and allows the reallocation of resources.

4 Conclusions

The real-time software development process is time consuming. This paper presents work in progress towards alleviating the costs of real-time software system design, development, and maintenance. This work incorporates state-of-the-art scheduling theory within a software synthesis architecture. This architecture leverages off of past research into the successful synthesis of software. Results illustrating the ability to accurately predict the time and space characteristics of a task and then synthesize an implementation to meet the prediction can be found in [16, 17]. This synthesis system successfully generates functional C code from high-level algorithmic descriptions. The generated code can be modeled for speed and space requirements, and these predictions prove to be reasonably accurate for a variety of platforms. The ability to generate predictable code is the cornerstone of the RT-SYN system. By demonstrating the validity of the synthesis system, we validate the premise of automated real-time task set synthesis.

The scope of this paper presents work in the initial phase of the design and development of a real-time software synthesis architecture. We target the synthesis of uni-task systems with a focus on the use of prediction to aid synthesis. Future work in this phase encompasses expanding the repertoire of algorithms within the Algorithm Database and verifying efficient synthesis of these algorithms. The second phase of this work targets on multi-task synthesis. Specifically, we will incorporate the RT Mach operating system into the Platform Database. We see this second phase as proving the predictability of RT Mach by accurately predicting synthesis results. The third and final phase of this work targets system-level synthesis. At this point, we will introduce real-time network characteristics into the System Synthesis schedulability analysis. These characteristics encompass adding a real-time database and system-level application scenarios.

References

[1] D. Barstow. *Automatic Program Construction Techniques*, chapter The roles of knowledge and deduction in algorithm design, pages 201–222. McMillan, 1984.

[2] S.K. Chang. *Principles of Visual Programming Systems*. Prentice Hall, Englewood Cliffs New Jersey, 1990.

[3] S.K. Chang. A visual language compiler for information retrieval by visual reasoning. *IEEE Transactions on Software Engineering*, 16:1136–1149, October 1990.

[4] J. L. Hennessy and D. A. Patterson. *Computer Architecture A Quantitative Approach*. Morgan Kaufmann, San Mateo California, 1990.

[5] S.K. Chang T. Ichikawa and P.A. Ligomenides, editors. *Visual Languages*. Plenum Press, New York, 1986.

[6] S.M. Kay. *Model Spectral Estimation Theory and Application*. Prentice Hall Signal Processing Series, 1988.

[7] L. Sha J.P. Lehoczky and R. Rajkumar. Scheduling algorithms for real-time operating systems. Technical report, Department of Computer Science, Carnegie-Mellon University, December 1986.

[8] C.L. Liu and J.W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *JACM*, 20 (1):46 – 61, 1973.

[9] E. Kant F. Daube W. MacGregor and J. Wald. *Automating Software Design*, chapter 8: Scientific Programming by Automated Synthesis. AAAI Press, 1991.

[10] M. Eisenstadt J. Domingue T. Rajan and E. Motta. Visual knowledge engineering. *IEEE Transactions on Software Engineering*, 16:1164–1177, October 1990.

[11] D. Setliff and R. Rutenbar. On the feasibility of synthesizing cad software from specifications: Generating maze router tools in elf. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 10(6):783–801, June 1991.

[12] D. Setliff and R. Rutenbar. Knowledge representation and reasoning in a software synthesis architecture. *IEEE Transactions on Software Engineering*, Accepted for publication in special issue on Knowledge Representation to appear in September 1992.

[13] B. Shneiderman. Direct manipulation: A step beyond programming languages. *IEEE Transactions on Computers*, 16:57–69, August 1983.

[14] D. Harel H. Lachover A. Naamad A. Pnueli M. Politi R. Sherman A. Shtull-Trauring and M. Trakhtenbrot. Statemate: A working environment for the development of complex reactive systems. *IEEE Transactions on Software Engineering*, 16:403–414, April 1990.

[15] H.A. Simon. Whether software engineering needs to be artificially intelligent. *IEEE Transactions on Software Engineering*, SE-12(7):726–732, July 1986.

[16] T.E. Smith and D.E. Setliff. Towards an automatic synthesis system for real-time software. In *Proceedings of the 12th IEEE Real-Time Systems Symposium*. IEEE, December 1991.

[17] Tobiah E. Smith. Constraint-driven synthesis of signal processing algorithms. Master's thesis, Elec. Eng. Department, Univ. Of Pittsburgh, Pittsburgh, PA, November 1991.

[18] H. Tokuda and M. Kotera. A real-time tools set for the arts kernel. In *Proceedings of 9th Real-Time Systems Symposium*. IEEE, December 1988.