

## Analysis of Guaranteed Service Times of Distributed Real-Time Objects

K. H. (Kane) Kim

University of California  
Irvine, CA 92697, U.S.A.  
Kane@Ece.Uci.Edu  
(Position paper)

As the demands for real-time distributed systems increase, the needs for programming tools useful in development of such application systems are becoming increasingly acute. An issue that the research community has long recognized as an important technological challenge but has not shown much progress in meeting the challenge is to *guarantee response times* of real-time distributed systems. Two basic problems must be solved to effectively meet this challenge:

- (1) To establish the distributed real-time program structure and the system infrastructure structure, i.e., the structure of the operating system (OS) and the communication infrastructure, that enable systematic analysis of the worst-case time behavior of the application systems; and
- (2) To establish a tool which performs automated analysis of the worst-case time behavior while leaving only minimal work to the designers.

The problem area (1) has been recognized as a research area for a long time but only in recent years, a skeleton of a usable technical foundation started emerging. In other words, there has long been the lack of fully general and yet easily analyzable distributed real-time program structures and also the lack of useful OS timing models. However, on the basis of the skeleton emerged, a rapid progress is expected in the future toward establishment of a full technical foundation. Therefore, time seems ripe also for launching new larger-scale attacks on the problem (2) on the basis of those recent developments in handling the problem (1).

### 1. Challenging nature of the timing analysis of distributed real-time objects

Ensuring that the real-time distributed computer systems produced meet all the applicable hard deadlines is a big challenge. First of all, one cannot just rely on a certain number of test-runs of the implemented system since devising a test-case corresponding to the worst possible execution case is often very difficult. In determining the *execution safety*, i.e., the absence of the possibility of violating hard deadlines, an analysis of the source code and other design specifications along with descriptions of the chosen execution environment, is an inevitable approach. Then, the contributions of an OS and

message communication facilities to the system response time are often difficult to determine with good accuracy.

General-purpose OSs have long been known to show unpredictable timing behavior. On the other hand, specialized real-time OSs have been showing continuous improvements in predictable timing behavior. This improvement will continue for years to come and moreover, will influence the evolution of general-purpose OSs as well.

Moreover, in recent years, the understanding about idealistic structures for real-time OS kernels and middleware that yield easy and systematic analysis of their worst-case service times, has been expanded substantially. Our experiences indicate that even a popular commercial general-purpose OS such as Windows NT can be configured and extended with middleware to approach the idealistic structure to a considerable extent (e.g., one capable of supporting application actions with the 10ms-level timing accuracy) [Kim99c].

The understanding about the timing behavior of the communication facilities has also increased substantially in recent years. The complexity and scale of the communication infrastructure varies widely among the application environments. In a local area network (LAN) environment, the timing behavior of the network is under considerable influence of the OS. The research on the effect of the combination of a communication infrastructure and a OS on the response time of an application system has increased steadily in recent years [Kim99a, Kop97]. In fact, an important consideration in designing and configuring a real-time OS is to properly influence / control the communication network. Although it is not yet a mature part of the state of the art in real-time OS design, its advances are expected to be accelerated in coming years.

Even if the impacts of OSs and communication infrastructure are well understood, distributed application computations often exhibit complex forms of synchronizations and competitions for accessing shared data and obtaining OS services. Because of the logical complexity of sizable real-time distributed applications and execution engines (aggregates of hardware, OS, and communication infrastructure), the analysis of execution safety becomes easily an unmanageable problem if the applications are not well structured. Conventional process-network structuring of real-time distributed

applications leaves too wide a door open for incorporating various difficult-to-analyze competing modes for accessing shared data and obtaining OS services, compared to recently emerged real-time object structuring. In other words, *guaranteeing timely services* (i.e., guaranteeing the abilities to meet hard deadlines) of real-time distributed computing applications becomes much more feasible with proper real-time object structuring than with the conventional use of processes or process-segments as modules.

One such real-time object structuring approach is the *time-triggered message-triggered object* (TMO) structuring scheme [Kim97, Kim99b]. The TMO incorporates several rules for execution of its components that make the analysis of the worst-case time behavior of TMOs to be systematic and relatively easy while not reducing the programming power in any way. The TMO scheme was formulated from the beginning with the objective of enabling *design-time guaranteeing of timely actions*.

## 2. Systematic timing analysis process desired

The systematic analysis procedure basically involves determining

- (1) the worst-case execution time of every program-segment that requires the CPU (+memory) resource only and
- (2) the worst-case completion time of every call for an OS service including every call for a message communication service involving the communication infrastructure.

The tightest upper bound on the worst-case execution time of a program-segment requiring the CPU resource only that can be determined at the compile-time without executing the program-segment is called the *tightest execution time bound* (TETB) of the program-segment. Although a substantial amount of research efforts have been invested into analytical and experimental research on the TETBs of program-segments, research attempts on the analysis of the execution time of an overall distributed program spent for delivery of a specific application service have been sporadic and have never reached an experimental research phase.

Pragmatic considerations dictate that a desirable automated analysis tool must be based on an existing

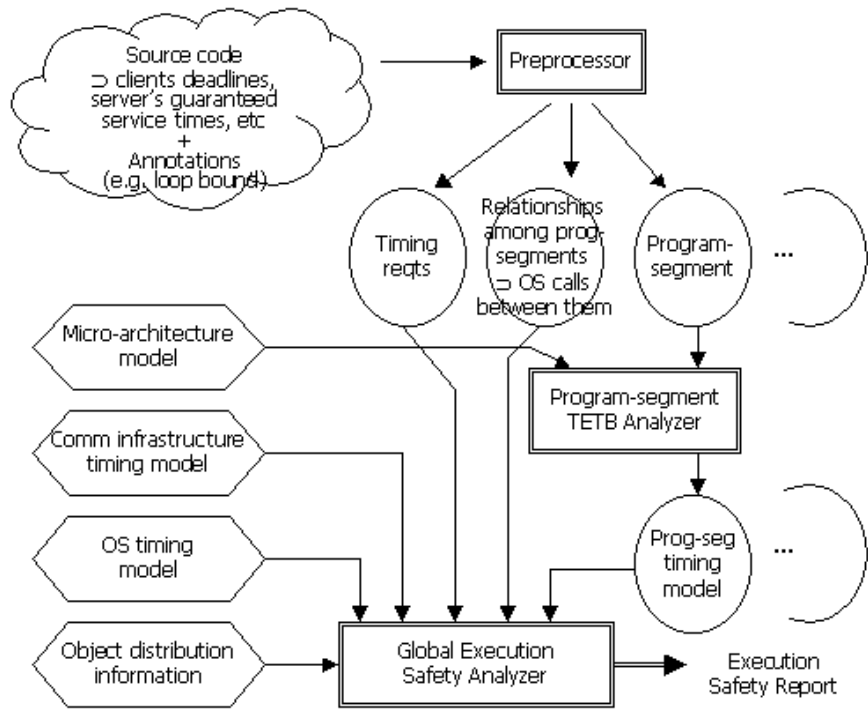


Figure 1. The analysis process and the support tools desired

compiler for a predominant programming language such as C++ and Java. It appears that no extended C++ compiler which performs the TETB analysis of program-segments is currently available, let alone a tool for automated analysis of distributed program execution times.

Our view of the analysis process and support tools needed is depicted in Figure 1. The source code of a distributed computing application program structured as a network of real-time objects such as TMOs contains specifications of clients' deadlines, servers' guaranteed service times, and output time-windows. In addition, the program source contains additional useful information in the form of *annotations* and the examples of such information are upper bounds on loop iterations expressed as functions of input data, ranges of input data values, etc. From this source information, the *preprocessor* generates the following three types of properly formatted information:

- (1) Program-segments in source code forms, each requiring the CPU resource only;
- (2) Relationships among program-segments including the OS calls linking them;
- (3) Timing requirements enforced at various execution points within and between program-segments

The program-segments generated and formatted by the preprocessor is then fed into the *Program-segment TETB Analyzer* (PTETBA) along with a chosen *micro-architecture model* which is a parametric representation of

the characteristics of the chosen CPU resource relevant to the TETB analysis. PTETBA in turn generates a *timing model for each program-segment* which is in general a TETB representation which may contain some parameters to be used during the global analysis of the overall application program.

The TETB analysis of program-segments generally involves an inter-procedural TETB analysis which in turn involves both a bottom-up pass and a top-down pass of a procedure call graph [Pus97]. The execution time of each invocation of a procedure can vary significantly depending on the parameters and their types.

The global analysis is performed by the *Global Execution Safety Analyzer* (GESA). GESA requires not only program-segment timing models but also a communication infrastructure timing model, an OS timing model, and information on *object distribution* as input. GESA then produces a judgment on the execution safety, i.e., the absence of the possibility of deadline violations.

Both the *OS timing model* and the *communication infrastructure timing model* desired here are the characterizations of the timing behaviors of OSs and communication infrastructure in forms enabling parameterized quantitative analysis. The object distribution information indicates an object-to-node mapping.

When GESA produces a negative report "unable to guarantee execution safety", it does not necessarily mean that deadline violations can occur during the execution. It just says its inability to guarantee execution safety. Even with the negative report from GESA, the system engineer may still want to take a chance with the chosen execution environment and the implemented program, especially after seeing that the system survives through all the test-cases prepared. Or the system engineer may want to play more safely and try to change the program or the execution environment.

In fact, guaranteed service times generated with help of the tools in Figure 1 may often turn out to be of poor quality, i.e., much larger than the service times observed during all of their service executions or test-runs. This is due to many factors, including fluctuations in the service times of the execution engine and loose timing design of the application. Therefore, the system engineers may often end up examining both the guaranteed service times and the *statistical performance indicators* in deciding on the acceptability of the systems produced. For example, an application system can be put through a number, say 1000, of test-runs while the service time is measured in each test-run. Then the statistical distribution of the service time for each type of service can be attached to the application system as a statistical performance indicator.

Moreover, when guaranteed service times of poor quality are produced by the tools, the system engineer may perform refined analysis to derive guaranteed service

times of better quality. Avoiding or minimizing this is of course the main goal of research in this area.

Figure 1 represents a target that will require an enormous amount of research and development efforts. An interesting question here is whether a reasonable practical implementation of the tool set will appear in a decade.

**Acknowledgments:** The research work reported here was supported in part by the US Defense Advanced Research Project Agency under Contract N66001-97-C-8516 monitored by SPAWAR, and in part by the NSF Next-Generation Software (NGS) Program under Grant 99-75053.

## References

- [Kim97] Kim, K. H., "Object Structures for Real-Time Systems and Simulators", *IEEE Computer*, Vol. 30, No. 8, August 1997, pp. 62-70.
- [Kim99a] Kim, K.H. and Subbaraman, C., "Dynamic Configuration Management in Reliable Distributed Real-Time Information Systems", *IEEE Trans. on Knowledge and Data Engr.*, Vol.11, No.1, Jan./Feb. 1999, pp.239-254.
- [Kim99b] Kim, K.H., "Real-Time Object-Oriented Distributed Software Engineering and the TMO Scheme", *Int'l Jour. of Software Engineering & Knowledge Engineering*, Vol. No.2, April 1999, pp.251-276.
- [Kim99c] Kim, K.H. Ishida, Masaki, Liu, Juqiang, "An Efficient Middleware Architecture Supporting Time-Triggered Message-Triggered Objects and an NT-based Implementation", *Proc. 2nd IEEE CS Int'l Symp. on Object-Oriented Real-time Distributed Computing (ISORC '99)*, St. Malo, France, May, 1999, pp.54-63.
- [Kop97] Kopetz, H., *Real-Time Systems*, Kluwer Academic Pub., 1997.
- [Pus97] Puschner, P. and Schedl, A., "Computing Maximum Task Execution Times – A Graph-Based Approach", *Real-Time Systems*, Vol. 13, No. 1, pp. 67-91, July 1997.

Proceedings

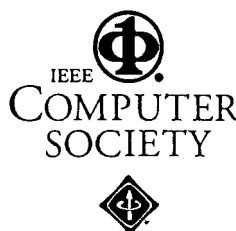
**Third IEEE International Symposium on  
Object-Oriented Real-Time  
Distributed Computing  
(ISORC 2000)**

March 15-17, 2000  
Newport Beach, California, USA

*Sponsored by*  
IEEE Computer Society

*In Cooperation with*  
IFIP WG 10.  
OMG  
IBM

*Financial Contributions from*  
IBM



Los Alamitos, California

Washington • Brussels • Tokyo

---