High-Level High-Precision Design and Programming of Real-Time Distributed Computing Components

K. H. (Kane) Kim
ECE Dept.
University of California
Irvine, CA 92697, U.S.A.
khkim@uci.edu

Abstract:

Current conditions in the information technology industry exhibit a compelling need for a real-time (RT) distributed programming and software engineering method which is at least *multiple times more effective* than currently widely practiced programming techniques [J-C00, Jen00, OMG99a, RTE00]. This new-generation RT distributed programming method must be based on a general high-level programming style which can be accommodated with minimal efforts by current-generation business application programmers (using C++ and Java) rather than on a style that has been practiced by assembly language or low-level language programmers. Ideally, designers must be required to specify both the interactions among distributed program components and the timing requirements of various actions in natural intuitively appealing forms only.

At the same time, a desirable RT distributed software engineering method must allow some system engineers dealing with safety-critical applications to confidently produce *certifiable* RT distributed computing systems. The state of the art in this area is quite inadequate. The general public which has witnessed conspicuous improvements in the reliability of the desk-top computer systems in 1990's will demand in this new century a different level of reliability for the systems in safety-critical applications. They will demand sufficiently trustable certifications of the designs and implementations. *Design-time guaranteeing of response times* of computing components / systems is considered a major technological requirement that must be fulfilled before such certification becomes a common practice.

Particularly important directions to pursue toward these general goals are the following:

(1) High-level real-time distributed objects which do not reflect low-level program abstractions such as threads, semaphores, and socket protocols but contain high-level intuitive and yet precise expressions of timing requirements.

(2) Graphic user interface (GUI) based interactive design of real-time distributed computing software structures and components and automated generation of their code-frameworks.

(3) End-to-end timing analyzability: Easy and mostly automated analysis of real-time distributed computing software components and systems to verify the *execution safety*, i.e., absence of possible timing failures, and support systematic design modification to remove timing failure possibilities, must be enabled. This can be met only if the execution engines including kernels and middleware are also properly structured and analyzed.

The author believes that approaches 1 and 2, if properly developed, will lead to dramatic improvements (at least several times improvement rather than say 20% improvement) over current practices in terms of the new-generation application development productivity and the reliability of new-generation applications produced. The approach 3 is essential if we were to guarantee certain levels of quality of service (QoS) in new-generation safety-critical distributed computing applications and certify such applications.

The author believes that a reasonable body of research results that has established the feasibilities of these approaches beyond any reasonable doubt exists now [IEE00, ISO, WOR]. Moreover, about 10 years of sustained support for research in these areas will result in conversion of most of those feasibilities into daily practices. In this workshop, the author proposes to give an overview of some of those feasibility indicators and the steps and the estimates of the efforts needed to convert those feasibilities into the technologies relevant to daily practices by the engineers of new-generation application systems.

More details of the author's views outlined above and his views on important research issues in fault-tolerant real-time distributed computing technologies are discussed in [Kim00a] and [Kim00b]. Together with several colleagues such as Hermann Kopetz, the author founded the *WORDS* (*IEEE CS's Workshop on Object-oriented Real-time Dependable Systems*) Series and the *ISORC* (*IEEE CS Int'l Symp. on Object-oriented Real-time distributed Computing*) Series to stimulate R&D in some of the areas mentioned above. Some more elaborations on the motivations for pursuing the approaches mentioned above are contained in Appendix A.

References

[IEE00] 'A special issue of Computer (a magazine of IEEE Computer Society) on Object-oriented Real-time distributed Computing', June 2000.

[ISO] *ISORC* (*IEEE CS Int'l Symp. on Object-oriented Real-time distributed Computing*) Series; 1st held in April 1998, Kyoto, Japan; 2nd in May 1999, St. Malo, France; 3rd in March 2000, Newport Beach, CA; 4th in May 2001, Magdeburg, Germany. Proceedings are available from IEEE CS Press.

[Kim00a] Kim, K.H., "Object-Oriented Real-Time Distributed Programming and Support Middleware", *Proc. ICPADS 2000 (7th Int'l Conf. on Parallel & Distributed Systems)*, Iwate, Japan, July 2000, pp.10-20, (Keynote paper).

[Kim00b] Kim, K.H., "Issues Insufficiently Resolved in Century 20 in the Fault-Tolerant Distributed Computing Field", Proc. SRDS 2000 (19th IEEE CS Symp. on Reliable Distributed Systems), Nuremberg, Oct. 2000, pp.106-115 (Invited paper).

[WOR] *WORDS* (*IEEE CS's Workshop on Object-oriented Real-time Dependable Systems*) Series; 1st held in Oct. '94, Dana Point; 2nd in Feb. 1996, Laguna Beach; 3rd in Feb. 1997, Newport Beach; 4th in Jan. 1999, Santa Barbara; 5th in Nov. 1999, Monterey; 6th in Jan. 2001, Rome, Italy. Proceedings are available from IEEE CS Press.

Appendix A

Motivations for using the OO RT programming approach

Among several cutting-edge technology movements initiated in 1990's in software engineering is the *high-precision real-time (RT) object-oriented (OO) programming* movement. In this author's view, the most important goal of that movement has been to instigate a quantum productivity jump in software engineering for RT computing application systems. Particularly targeted application domains have been those challenging large-scale distributed / parallel computing applications in fields such as factory automation, telecommunication, defense, intelligent transportation, emergency management, etc.

The movement is still in its youthful stage and its impact has just started surfacing up. However, its great potential is now much more clearly and widely recognized than it was in mid-

1990's.

A.1 Complexity and costs of RT distributed programming

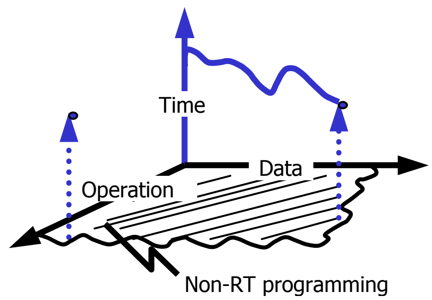

Figure 1. RT programming as a generalization of non-RT programming

Starting a few years ago, the field of RT computing applications has been showing a rapid growth pattern. Computer systems in those application domains are generally responsible for *RT control* of physical devices, *RT storage and* search for information, and *RT communication and display* of information. In addition, they are often tasked to perform *RT simulation* of their application environments. The field of computer-embedded communication-equipped system engineering has been growing particularly fast in recent years.

As a result, industry has felt an acute need for RT distributed programming and software engineering methods which are at least *multiple times more effective* than currently widely practiced programming techniques. This new-generation RT distributed software engineering method must be based on a "general high-level programming style" which can be accommodated with minimal efforts by current-generation business application programmers (using C++ and Java) rather than on a style that has been practiced by assembly language or low-level language programmers. Continuous use of old low-level programming styles is not economically viable for dealing with increasing demands for new RT application systems.

Designers must be required to specify both the interactions among distributed program components and the timing requirements of various actions in natural intuitively appealing forms only. Designers should not be forced to deal directly with notions such as priorities for the sake of meeting application timing requirements since priorities are usually associated with low-level computation units such as processes and threads in manners reflecting the application semantics poorly.

The fact that *distributed objects* represent a higher-level structure for distributed programs than *distributed processes* do have been widely recognized by the industry in the past 10 years, e.g., technology movements such as CORBA, DCOM, and RMI. Naturally, researchers started searching for extensions of distributed objects that allow unambiguous specification of timing requirements imposed on various computations units [IEE00, ISO, WOR].

A.2 Should RT programming remain an esoteric branch of computer science and engineering ?

It is fair to say that up to now, RT programming has been treated as an esoteric branch of computer science and engineering. Very few universities have courses on RT programming and even those few existing courses are almost entirely graduate courses.

The main reason is that RT programming has been practiced largely as an ad hoc art in a form looking quite alien to the vast number of business and scientific application programmer. On the other hand, there is no reason why future RT computing cannot be realized in the form of a generalization of the non-RT computing, rather than the other way around. Figure 1 depicts this. If the main-stream (traditional) programming science is viewed as a study of the two-dimensional space, (data X operation), then a proper form of RT programming should be practiced as work within the three-dimensional space, (data X operation X time). Of course, the less the programmer is burdened with the work on the time dimension, the better off. We just need a powerful programming scheme capable of dealing with all practically useful RT and non-RT computing requirements in uniform manners. Under such a properly established RT programming methodology, every practically useful non-RT program must be realizable by simply filling the time constraint specification part with the default value "unconstrained".

A.3 Reliability of RT distributed programs

RT programs have been notoriously difficult to analyze. It is well known that testing alone does not ensure sufficiently high reliability of RT programs. Given rapidly increasing demands for RT application systems and the fact that complexities of RT distributed programs are far greater than those of single-node programs, the practice of relying solely on testing for reliability assurance is becoming less and less tolerable.

A new-generation RT distributed software engineering method must allow some system engineers dealing with safety-critical applications to confidently produce certifiable RT distributed computing systems. The general public which has witnessed conspicuous improvements in the reliability of the desk-top computer systems in 1990's will demand in this new century a different level of reliability for the systems in safety-critical applications. They will demand sufficiently trustable certifications of the designs and implementations. *Design-time guaranteeing of response times of computing components / systems* is considered a major technological requirement that must be fulfilled before such certification becomes a common practice.

Moreover, distributed computer systems (DCSs) used in safety-critical RT applications must possess some degrees of *RT fault tolerance* capabilities. The current reality is that widely used node operating systems (OSs) do not show easily analyzable and predictable timing behavior. Whether we like it or not, there are certain *hard deadlines* in human societies and violation of these deadlines have severe consequences. For example, suppose cars are to be driven by automated drivers (robots). If such cars are heading toward a collision course, then the collision can be avoided only if at least one driver detects the danger and takes an avoidance action within a certain hard deadline. Applications subject to hard deadlines are called *hard-real-time (HRT) applications*.

Therefore, both unreliable hardware components and node OSs with erratic timing behavior can lead to violation of hard deadlines. This makes employment of RT fault tolerance mechanisms in safety-critical RT DCSs to be imperative.

Again, whether hardware and node OSs possess RT fault tolerance mechanisms or not, distributed HRT applications must be designed with *response time guarantees*. It cannot be done if application software is structured in undisciplined manners. That is, easily analyzable structuring of application software must be pursued to the maximum extent possible. Research in recent years has made it clear that high-level structuring in the form of distributed RT objects has significant advantages in this regard in comparison to somewhat lower-level distributed process structuring.

A.4 HRT program components

From the viewpoint of realizing systematic modular construction of sizable HRT distributed computing applications, one highly desirable approach is to use *HRT program components*, each of which is associated with a *guaranteed service time* (GST), also called *guaranteed completion time*, for every service method that it provides. If a program component provides multiple service methods, it is associated with multiple GSTs. If every program component contributing to the computation subject to a hard deadline has a GST associated with it, then meeting the hard deadline becomes the trivial problem of checking whether the sum of the GSTs of the contributing components is indeed less than the hard deadline. The real problem then is to ensure that every GST associated with every program component is credible. In a sense, each GST is a hard deadline that the designer of the program component has decided to impose on the program component. Therefore, implementation of such a program component may involve the use of fault tolerance mechanisms.

If a program component fails to meet a GST, then a number of other program components designed to be dependent on the former component may also be treated as failed components unless the latter were also designed to handle reports about the failure of the former component. Any attempt to replace a GST of a program component by a "soft" deadline to be imposed on the component is expected to lead to a complicated methodology which does not enable modular systematic construction of HRT systems. Special situations where it might be worth augmenting HRT program components with statistical performance indicators were discussed in the literature.

Therefore, the HRT program component possessing GST attributes is a key to cost-effective systematic construction of HRT distributed computing applications. This *HRT component based construction approach* is conservative in nature but highly cost-effective due to its systematic and modular characteristics. Here the designer/implementer of an HRT component announces its GST to all potential designers of client components. We expect that this HRT component based construction approach will meet increasing acceptance in the practicing field in the future.

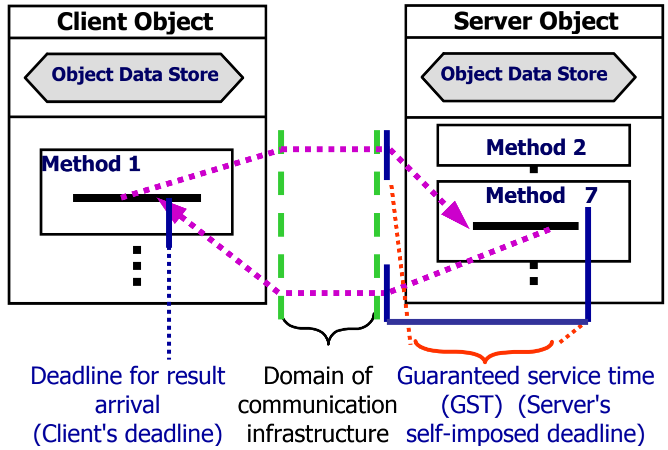

Figure 2. Client's deadline vs. Server's GST

Figure 2 depicts the relationship between a client and a server component in a system composed of HRT components which are structured as distributed computing objects. The client object in the middle of executing its method, Method 1, calls for a service, Method 7 service, from the server object. In order to complete its execution of Method 1 within a certain target amount of time, the client must obtain the service result from the server within a certain deadline. During the design of this client object, the designer searches for a server object with a GST acceptable to him/her. The designer must also consider the time to be consumed by the communication infrastructure in judging the acceptability of the GST of a candidate server object.