



S
SERI
JOURNAL

SERI JOURNAL

1 Toward New-Generation Object-Oriented
Real-Time Software and System Engineering
by Kwang Hae(Kane) Kim, University of California, Irvine

24 Semantics-based Transaction Management
for Distributed Multidatabases
by Pyeong S. Mah

44 A New Dynamic Signature File Method for
Efficient Information Retrieval
by Jeong-Ki Kim, Choon-Hee Lee and Jae-Woo Chang

63 Texture Classification Using Wavelet Frame and
Neural Networks
by Kyung-Ok Kim and Young-Kyu Yang

Systems
Engineering
Research
Institute

January 1997
Volume 1
Number 1

Subscriptions

The SERI Journal is distributed free of charge to SERI researchers, selected non-SERI individuals, industries, governmental organizations, libraries, and educational institutions world-wide. Please address your subscription requests or change of address on a printed letterhead to SERI Library, P.O. Box 1 Yoosung-Gu, Taejon, 305-600, Korea.
Telephone: +82-42-869-1162 Facsimile: +82-42-869-1169

TOWARD NEW-GENERATION OBJECT-ORIENTED REAL-TIME SOFTWARE AND SYSTEM ENGINEERING (INVITED PAPER)

KWANG HAE (KANE) KIM*

Abstract. In recent years, the market conditions for the real-time computer system (RTCS) technology and the hardware economy have been showing significant changes and thus this author feels that time is ripe for vigorously pursuing new paradigms in designing and structuring RTCS's. The new design paradigm discussed in this paper for new-generation RTCS's is called the general-form timeliness-guaranteed (GT) design paradigm and this approach has the flavor of an idealistic perfectionist approach. The essence of the GT design paradigm is to realize real-time computing in a general manner not alienating the main-stream computing industry and yet allowing system engineers to confidently produce certifiable RTCS's for safety-critical applications. After a discussion on the GT design paradigms, a specific approach devised by the author and his collaborators for following the paradigm, the RTO.k object structuring scheme, is discussed along with the established and desired support tools. The RTO.k scheme is compared with other approaches for extending the conventional object structuring scheme to suit real-time applications. Among other things, the RTO.k object scheme provides the uniformity and a wide range of controlled accuracy in representation of both application environments and control computer system designs during multiple system engineering phases. It thus facilitates a uniform and integrated design of real-time distributed computer systems together with the real-time simulators of their application environments. The potential advantages of this approach including the strong requirements-design traceability, the feasibility of thorough and cost-effective validation, the ease of maintenance, etc., are discussed. Much of the desired tools discussed are considered highly meaningful targets for future research in this area.

Key words. real-time, real-time object, RTO.k, guaranteed timely service, DREAM kernel, micro-kernel, thread, process

1. Introduction. Significant improvements achieved in recent years in hardware economy and component reliability have spurred the growth in both the volume and the variety of the market for computer applications. One of the computer application fields which started showing noticeable new growth trends is the real-time computing application field.

The state of the art for engineering of real-time computer systems (RTCS's) is inadequate for coping with this increasing application demand because of the following weaknesses and others.

(1) Little impacts of *object oriented* (OO) design approaches

Over the past 15 years OO design approaches have become a common practice in development of non-real-time business data processing software due to the modularity, generality, and natural abstraction benefits that the OO approaches bring in [5, 3, 26, 27]. On the other hand, OO-structuring has had minimal impacts in RTCS engineering in contrast to its pervasive use in non-RTCS engineering. This means that much of the capabilities existing in the vast business data processing software field is currently not utilized in development of RTCS's. It also means that the RTCS engineering process and the real-time application software themselves take peculiar forms unfamiliar to the vast main-stream software engineering community. The consequence is the poor economy of scale in RTCS development and the relatively low reliability

*Department of Electrical and Computer Engineering, University of California, Irvine, CA 92697, USA, (Kane@Ece.Uci.Edu)

©1997 SERI

SERI Journal, Vol. 1, No. 1, January 1997, pp. 1-23.

of the software products except in cases of small-scale simplistic phase-locked loop control types of applications.

(2) Low reliability of large-scale RTCS's

Various representation schemes, analysis and synthesis techniques, and tools have been developed for use in each phase of RTCS engineering such as specification, design, implementation, validation, and maintenance. By and large these techniques have not evolved in sufficiently integrated forms up to now. As a result, the current RTCS engineering practices suffer from the following problems: (a) weak traceability among various system models used, (b) lack of rigor in requirements specification, and (c) lack of integration in design techniques. Moreover, experiences have shown that real-time distributed computing software is notoriously difficult to test. All these lead to the fact that the reliability of the large-scale real-time software produced under the state of the art is not sufficiently high.

Therefore, this author feels that it is time for a so-called *paradigm shift* in RTCS design. Recently, this author started advocating pursuit of a futuristic design paradigm called the **general-form timeliness-guaranteed** (GT) design paradigm [14, 22]. The essence of the GT design paradigm is to realize real-time computing in a general manner not alienating the main-stream computing industry and yet allowing system engineers to confidently produce certifiable RTCS's for safety-critical applications. The GT design paradigm will be fully discussed in Section 2. Arguments are then made in Section 2 for exploring new real-time extensions of the conventional OO structuring scheme to establish tools for exercising the GT design paradigm.

A tool formulated by this author together with his research collaborators for exercising the GT design paradigm is called the *RTO.k object structuring scheme* [11, 12, 22]. After presenting the essence of the RTO.k scheme in Section 3, we discuss the differences between the RTO.k scheme and other real-time extensions of the conventional object structuring scheme in Section 4.

The RTO.k object model provides the uniformity and a wide range of controlled accuracy in representation of both application environments and control computer system designs that evolve through multiple system engineering phases. In fact, we recently started advocating a *uniform and integrated design* (UID) of real-time distributed computer systems (DCS's) together with the *real-time simulators* of their application environments. Under this approach, the integrated design of a control computer system and its application environment simulator takes the form of a network of RTO.k objects. The UID approach has been put through several experiments in the UCI DREAM Laboratory directed by the author [11, 21]. Several other experiments are under way in other organizations. The experiments conducted include the development of a military command control application, a steel mill control application, etc. All the implemented RTO.k structured applications run on PC's interconnected by an Ethernet LAN. The steel mill control design case is used here to illustrate the essence of the UID approach. The capabilities of the RTO.k object model for uniform representation of various designs and specifications evolving through multiple CSE phases are demonstrated. The UID approach is discussed in Section 5.

In order for the RTO.k objects to provide guaranteed timely services to external clients, the engine that supports the execution of RTO.k objects must provide guaranteed timely responses to service requests from the executing RTO.k objects. A model of an operating system kernel which can support RTO.k objects and conventional real-time processes with guaranteed timely services was formulated by the

author [14, 15]. The model is called the *DREAM (Distributed Real-time Ever Available Micro-computing) kernel*. A series of prototype implementations of the DREAM kernel have been produced by the author and his collaborating researchers to run on networks of PC's connected by the Ethernet local area network (LAN). A friendly C++ based application-programmer-interface (API) to the DREAM kernel, called the *DREAM library*, has also been developed [16, 20]. A brief overview of the DREAM kernel and the DREAM library is given in Section 6.

In order to firmly establish the RTO.k object based GT design as a technology usable by common practitioners, many more tools in addition to the DREAM kernel and the DREAM library need to be developed. A desirable software engineering environment (SEE) for RTO.k object based engineering of RTCS's is discussed in Section 7. In this author's view, the tools which appear as the to-be-developed components of the SEE are highly meaningful targets for future research. The paper concludes in Section 8.

2. Futuristic Design Paradigm and Desirable OO Structuring. In order to achieve noticeable progress in the design efficiency and the system reliability attained, this author believes that the *general-form timeliness-guaranteed* (GT) design paradigm which consists of the elements discussed below must be vigorously pursued [12, 14, 22]:

(1) *General-form design*: Future real-time computing must be realized in the form of a generalization of the non-real-time computing, rather than in a form looking like an esoteric specialization.

(2) *Application of the same system/subsystem model through multiple engineering phases*: In order to achieve significant improvement in seamless integration of design tools and techniques, one of the key issues to be resolved is the *uniformity* and *the range of controlled accuracy* in representation of various levels of system designs evolving through multiple design phases. The uniformity must be attained by using a powerful system/subsystem modeling scheme consistently during multiple phases. Moreover, a desired representation (or modeling) scheme should be effective not only in the abstraction of RTCS's under design but also in the representation of the application environments to the degree of accuracy desired in each phase of system development. Ideally, the modeling scheme should be capable of handling not only logical value manipulation capabilities but also temporal characteristics at various degrees of accuracy.

The general-form design is important with respect to utilizing the capabilities existing in the vast business data processing software field in development of RTCS's, which is currently not done. Under a properly established RTCS design methodology, it should be possible to realize every practically useful non-RTCS by simply filling the time constraint specification part with unconstrained default values. Such design becomes a reality only when a powerful structuring scheme capable of dealing with all practically useful real-time and non-real-time computing requirements is established. This approach is the only practical way to meet the growing application demands for real-time computing with the acceptable economy of scale and significantly improve the reliability and other quality attributes of the real-time software products.

The general-form design approach and the approach of using the same powerful modeling scheme during multiple engineering phases may seem unrelated but this author believes that both can be met by the same structuring technique of OO type. In other words, searching for a proper real-time extension of the conventional OO structuring scheme is a natural way not only for formulating a general-form design

method but also for finding a powerful system/subsystem model. This is all because of the natural modularity characteristics and the multi-level abstraction capabilities of the basic object structure. Indeed in the last several years there has been a growing trend of research activities aimed for extending the conventional OO-structuring approaches to support RTCS design [1, 2, 8, 11, 23, 27, 28, 31]. Although not included in many of those real-time extensions of the basic object, this author believes that two of the most essential new elements in a properly formulated *real-time object* are the facilities for *time-triggered activation* of computing actions and the *deadline enforcement* facilities. In particular, applying the time-triggered activation to object methods, rather than to individual statements, and structuring them in explicit forms clearly distinguished from the basic object methods which are triggered by messages from the clients, are considered to be the most effective approaches in this area.

(3) *Design-time guarantee of timely service capabilities of subsystems*: To meet the demands of the general public on the assured reliability of future RTCS's in safety-critical applications, there does not appear to be any adequate way but to require the system engineer to produce design-time guarantees for timely service capabilities of various subsystems (which will take the form of objects in object-oriented system designs).

Experiences of practicing engineers indicate that testing alone is not sufficient for assuring the level of reliability of RTCS's which the customers have started demanding. Even in late 1960's when the real-time distributed computing software field was at best in its embryonic stage, Dijkstra uttered the famous words, "Program testing can show the presence of bugs but never their absence" [6]. It is also known that in general the program verification approach is not practical for application to any sizable application software. However, this author believes that even if verifying the full logical behavior of a sizable real-time software is not practical, verification of the timing behavior must be pursued. This is because it is the timing behavior which presents the biggest difficulties to the system engineer relying on the testing for assuring proper behavior to a reasonable degree. The ease or difficulty of verifying the timing behavior depends on the way time constraints are specified in real-time objects.

So far the excuse used by the RTCS engineering community for not doing the timeliness-guaranteed design of sizable real-time software has been that the operating system (OS) available does not give guaranteed timely services and thus it is hopeless to realize application objects providing guaranteed timely services. However, such an OS characteristics is turning out to be a research issue of short-term nature, not a fundamental obstacle. Changes in this regard have already started occurring in industry.

In order to support all three elements of the GT design paradigm mentioned above, a real-time object structure that not only provides facilities for time-triggered activation of object methods and deadline enforcement but also eases the problem of producing design-time guarantees of timely service capabilities of the objects, needs to be established. Most of the previous works on real-time objects have not been aimed for supporting the design-time guarantee of timely service capabilities of objects, which is one of two most fundamental parts of the GT design paradigm.

3. The Essence of the RTO.k Object Structuring Scheme. An initial abstract framework of the **RTO.k object model**, also called the **time-triggered real-time object (TT-RTO) model**, came out of the attempt by this author and Hermann Kopetz at the Technical University of Vienna to find a proper extension of

the basic object model which is highly cost-effective in development of *hard-real-time* application systems. Based on the initial abstract framework formulated in late 1980's [23], a concrete syntactic structure and execution semantics was developed to support the GT design paradigm in recent years [11, 12, 16, 22]. The underlying motivations of the RTO.k object structuring scheme are depicted in Figure 3.1.

The basic structure of an RTO.k object is depicted in Figure 3.2. It is an extension of the conventional object model(s) and two most important and unique extensions are the following:

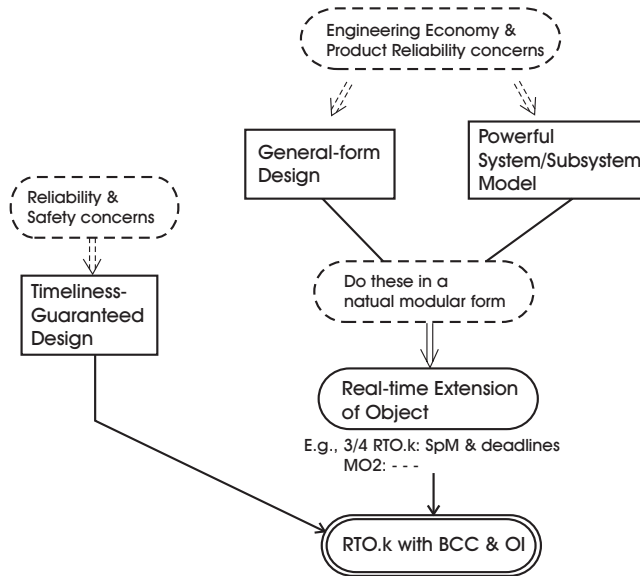


FIG. 3.1. Motivating factors for the RTO.k object structuring

(a) *Two clearly separated groups of methods:*

For some methods of an RTO.k object, a real-time clock serves as the mechanism for triggering the method executions as the clock reaches some values specified at design time and such methods are called **time-triggered (TT-) methods**, also called the **spontaneous methods** (SpM's), and *clearly* separated from the conventional **service methods** (SvM's) triggered by messages from clients. The two types of methods in an RTO.k object are different not only in the way their executions are triggered but also in that

“actions to be taken at real times *which can be determined at the design time* can appear only in SpM's”.

Therefore, actions of the type “at constant-clock-value do S” or the type “sleep-until constant-clock-value” can appear only in SpM's. Incorporation of SpM's means introducing the potential for the following two new types of concurrent executions of object methods in addition to the potential for concurrent executions of SvM's that exist in conventional objects:

Type I Concurrency among SpM executions: This concurrency is specified in an implicit but natural manner, e.g., two SpM's designed to be triggered at 10 am.

Type II Concurrency between SpM executions and SvM executions.

(b) *Basic concurrency constraint (BCC)*:

In order to dramatically reduce the designer's efforts in guaranteeing timely service capabilities of RTO.k objects, the execution rule which prevents conflicts between SpM's and SvM's is incorporated. Basically, *activation of an SvM triggered by a message from an external client is allowed only when potentially conflicting SpM executions are not in place*. To be exact, when a message-triggered SvM is not free of conflict with a SpM in accessing the same portion of the object data space (ODS), execution of the former method (SvM) must not be allowed in a time zone earmarked for a TT-execution of the latter method (SpM). This restriction is called the **basic concurrency constraint** (BCC). Therefore, SpM's are given higher priorities for execution over the SvM's. Note that this BCC does not impose any restriction on concurrent execution of SpM's or concurrent execution of SvM's. Therefore, executions of SpM's are not disturbed by SvM executions and triggering times of SpM's are fixed at the design time. At least this makes it very easy to analyze the execution time behavior of SpM's. For example, if a statement of the type "at 10am do S" appears in an SpM, its reliable execution can be easily assured.

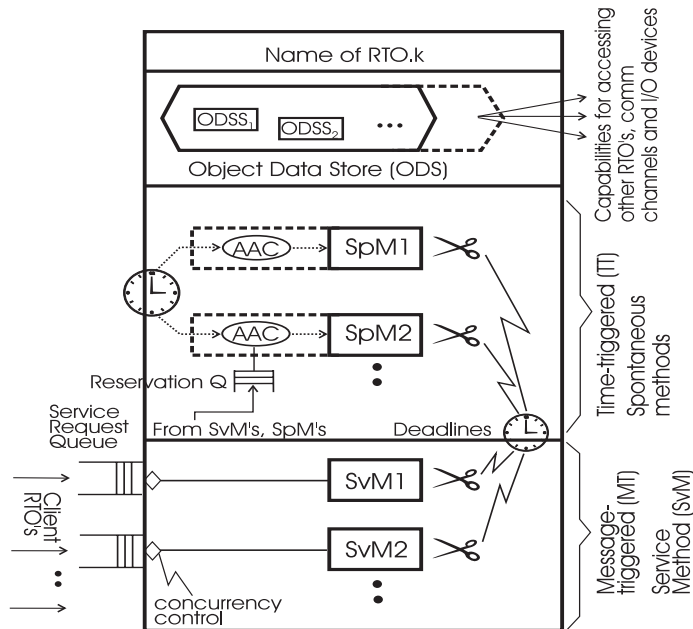


FIG. 3.2. Structure of the RTO.k object model (Adapted from [11])

The above two features make the RTO.k object model clearly distinguished from other proposed real-time object models [1, 2, 8, 28, 31]. In addition, the RTO.k object contains the following features not found in the conventional object model(s):

- (c) For each execution of an output action designed as a part of a method of an RTO.k object, a **deadline** is imposed;
- (d) Real-time data contained in an RTO.k object become invalid after the interval called the **maximum validity duration** passes; and
- (e) **Variables of time-value type**: There are two different time-value types of

variables which may appear in SpM's as well as in SvM's: absolute-time type and time-duration type. The practically effective granularity of the absolute time or time-interval values depends on the granularity of the clock maintained by the execution engine.

Triggering times for SpM's must be fully specified as constants during the design time. Those real-time constants appear in the first clause of an SpM specification called the **autonomous activation condition** (AAC) section. An example of an AAC is

“for t = from 10am to 10:50am every 30min
start-during (t, t+5min) finish-by t+10min”

which has the same effect as

{“start-during (10am, 10:05am) finish-by 10:10am”,
“start-during (10:30am, 10:35am) finish-by 10:40am”}.

A provision is also made for making the AAC section of an SpM contain only candidate triggering times, not actual triggering times, so that a subset of the candidate triggering times indicated in the AAC section may be dynamically chosen for actual triggering. Such a dynamic selection occurs when an SvM within the same RTO.k object requests future executions of a specific SpM. The AAC specifying candidate triggering times rather than actual triggering times starts with a declaration “if-demanded”. Therefore, there are two different modes of determining triggering times for SpM's:

- (a) fully determined during the system design, in which case the SpM is said to be *statically scheduled*, and
- (b) determined during the run time when an SvM requests executions of the SpM and designates a subset of the candidate triggering times prepared during the design time as actual triggering times, in which case the SpM is said to be *partially dynamically scheduled*.

An underlying design philosophy of the RTO.k object model is that an RTCS will always take the form of a network of RTO.k objects. RTO.k objects interact via calls by client objects for service methods in server objects. The caller may be an SpM or an SvM in the client object. In order to facilitate highly concurrent operations of client and server objects, non-blocking (sometimes called asynchronous) types of calls (i.e., service requests) can be made to SvM's.

The designer of each RTO.k object provides a guarantee of timely service capabilities of the object by indicating the *deadline for every output* produced by each SvM (and each SpM which may be executed on requests from SvM's) in the specification of the SvM (and some relevant SpM's) advertised to the designers of potential client objects. Before determining the deadline specification, the server object designer must convince himself/herself that with the *object execution engine* (hardware plus operating system) available, the server object can be implemented to always execute the SvM such that the output action is performed within the deadline. Again, the BCC contributes to major reduction of these burdens imposed on the designer [17].

4. Differences among the Real-time Extensions of Objects . In this section, major differences between the RTO.k object structuring scheme and other proposed real-time extensions of the basic object structuring scheme are discussed along with the costs and benefits which each distinct feature brings to the RTO.k scheme.

4.1. Clear-cut distinction between SpM and SvM. All real-time extensions of the basic object model proposed so far are *active objects* with some time constraints added. An active object is an object with its own thread of execution

control. Therefore, multiple active objects can exhibit concurrency among their activities. Of all the models proposed as real-time extensions of the basic object model, about half of them do not provide anything resembling SpM's. Such models will not be discussed any further in this paper [2, 4, 33]. In the object model adopted in the RTC++ project [8] and other models [1, 7, 28], the clear-cut distinction made in the RTO.k model between SpM's and SvM's is not done. That is, the rule, "actions to be taken when the real-time clock reaches values which can be determined at the design time can appear only in SpM's", is not adopted in any of those models. This rule was adopted in the RTO.k to simplify the task of guaranteeing the timely services offered by the RTO.k object. Also, the number of SpM executions that can proceed in parallel has no fixed limit in the RTO.k model unlike in models such as the MO2 model [1] and others [28].

In some models providing something similar to SpM's, interactions between those corresponding to SpM's and SvM's are not facilitated [7, 28, 31].

4.2. Basic concurrency constraint. The MO2 model proposed in [1] is the only other model which contains something resembling the basic concurrency constraint. However, the MO2 model allows only one SpM in an object. Also, an SvM in an RTO.k object is not initiated if it has the potential of running into data conflicts with any SpM in execution or with any SpM scheduled. On the other hand, SvM's and the SpM in an MO2 object may be in concurrent execution with intermittent competition for accessing data in the ODS. Therefore, the RTO.k model sacrifices some fine degree of parallelism for the sake of ease in guaranteeing timely services of objects.

4.3. Design-time guarantee of timely service of each object. The RTO.k object model was formulated with this specific objective of facilitating design-time guarantee of timely services of each object in mind. For example, the AAC section in the SpM declaration is restricted to be an expression which can be fully evaluated at design time. The execution engine which contains the operating system and both the inter-object communication facility and the intra-object inter-method communication facility, is required to yield an easy analysis of the worst-case execution time for any local or remote method execution. It appears that this "conservative" policy was not adopted in any other model, or at least not pursued to the extent done in the RTO.k model. However, the conservative policy of the RTO.k model can result in some sacrifice of hardware utilization in comparison to the case of using "liberal" policies adopted in other models.

4.4. Interactions among objects. In some models, only the blocking type of service call is facilitated although the developers probably considered the issue of allowing non-blocking service calls a minor issue. In the MO2 model [1], the non-blocking service call is facilitated. The *client-transfer call* mechanism in the RTO.k object model [12] is not available in any other model. The need for allowing client-transfer calls in the RTO.k model arose mainly due to the adoption of the basic concurrency constraint and the approach of design-time guarantee of timely services.

4.5. Main differences between the RTO model in RTC++ and the RTO.k object model. Overall, the object model in RTC++ [8] and the MO2 model [1] are closer in nature to the RTO.k model than other models are. Since the differences between the RTO.k model and the MO2 model have been pointed out clearly in the above discussion, the RTC++ approach is reviewed in more detail here and compared against the RTO.k object model.

In RTC++, an RTO is declared as an active object and contains one or more locally possessed threads called *master threads* and incorporates specifications of timing constraints imposed on object methods and individual statements. It also defines a finite set of *slave threads* responsible for executing object methods called by clients. Each object is assigned a fixed priority and the priority of a client object is *inherited* by the server object during the execution of the method called by the client.

Main differences between the RTO model in RTC++ and the RTO.k model are the following:

(a) In RTC++, master threads, which are counterparts for the SpM's (spontaneous methods) in the RTO.k object, are not clearly separated in their roles from SvM's to the extent that SpM's in the RTO.k object are separated from SvM's. For example, SvM's in RTC++ cannot directly request executions of master threads but instead can perform by themselves all computing actions which would be done by SpM's in an RTO.k object.

(b) The basic concurrency constraint (BCC) can not be adopted in RTC++ since the approach of assigning fixed priorities to RTO's and the priority inheritance approach were adopted. Therefore, master threads cannot have higher priorities than SvM's in RTC++, which is the opposite of the BCC approach in the RTO.k model. (c) No priorities are assigned to RTO.k objects. How to order competing accesses by object methods in execution for the same portion of the ODS is left to the execution engine which should utilize in its ordering decision the current information on time constraints associated with the competing methods. A special rule called the *ordered isolation rule* [17] can also be incorporated into the execution engine in order to make the design-time guarantee of timely services easier.

5. RTO.k Object Based Uniform Integrated Design of RTCS's and Real-time Simulators of Their Application Environments. In order to demonstrate the potential power of the RTO.k object structuring scheme, we developed several real-time distributed computing application systems including experimental prototypes of a military command control system and a factory control system. One of them is an experimental prototype of an *Automatic Gauge Control (AGC)* system which controls a steel *rolling and pressing process (RPP)* and the prototype also includes a real-time simulator of the controlled steel factory facilities. The implemented prototype was structured as a network of both *computing RTO's* and *environment simulator RTO's*, all distributed among PC's which are connected by Ethernet and equipped with the DREAM kernel. In this section, we present the steps involved in the top-down design of the AGC system as an illustration of the RTO.k object based approach for *uniform and integrated design (UID)* of an RTCS and its application environment simulator.

The steel factory application environment considered is shown in Figure 5.1. A roll of steel sheet of non-uniform thickness is first loaded onto a cylinder called the *pay-off reel (POR)*. This POR is rotated by an attached POR motor drive as shown in the figure. When the POR is rotated in a clockwise manner by the attached motor drive, the steel sheet wrapped around the reel advances forward and to the right along a guiding rail (not shown in the figure) and goes in between two solid cylinders, known as the *work roll (WR)*. The WR cylinders can not only be rotated by an attached WR motor drive but can be pressed together by a hydraulic actuator. Such a pressure will be transferred to the steel sheet that passes in between the WR cylinders. The net effect is to make the thickness of the steel sheet more uniform than before. When the WR motor drive rotates the WR cylinders, this causes the steel sheet that comes in

between the cylinders to advance forward further to the right. Finally, the steel sheet follows the guiding rails (not shown in the figure) and wraps around a cylinder called the *tension reel (TR)*. As in the case of the POR and the WR, the TR is also rotated by an attached motor drive, thereby causing the steel sheet coil around it.

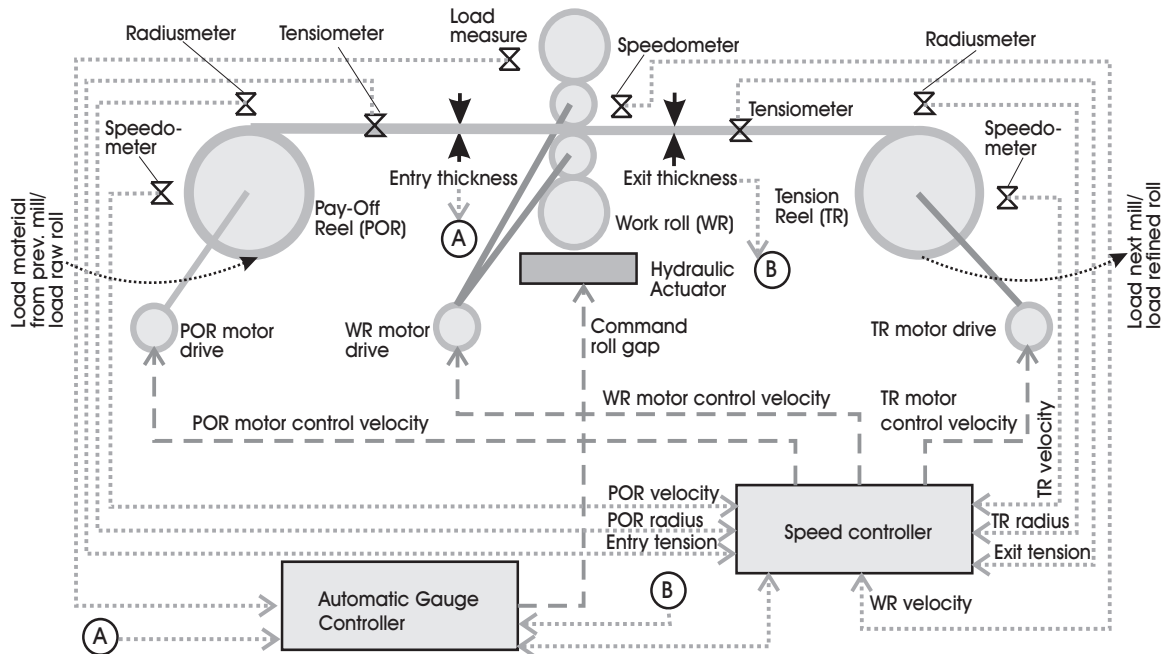


FIG. 5.1. Steel rolling and pressing mill application environment

Figure 5.1 shows just one RPP in detail. In general, the steel factory can have n different RPP's, all working concurrently. In this case, a roll of raw steel is first loaded on the POR of the first RPP. After being processed in the first RPP, the steel is loaded to the POR of the next RPP, and so on. The steel sheet thus moves from one RPP to the next in a pipeline fashion until it gets processed by the n th RPP and comes out refined from the pipeline.

In each RPP, the speeds of all the three motor drives as well as the load applied by the hydraulic actuator need to be carefully controlled. For instance, if the WR does not have a velocity sufficient enough to always keep the sheet between the POR and the WR taut, then the material between POR and WR could bend, resulting in an undesirable situation. The speed controller (SCT) shown in the figure, which is to be designed by a computer engineer (team), should control the three motor speeds while the automatic gauge controller (AGC) to be designed should control the load applied to the hydraulic actuator. The three speedometers shown in the figure supply the SCT with the current speeds of the motor drives, the two radius-meters supply the SCT with both the current radius of the POR and the radius of the TR, and the two tensiometers supply the SCT with the steel tension just before and the tension after the WR. The SCT can use these input parameters to calculate the new speeds of the three motors. The AGC receives the current load applied to the hydraulic actuator and the entry thickness measure of the steel sheet as the input and can use these values to calculate the desired gap between the WR cylinders.

Initially the high-level requirement is given by the customer who places an order for the AGC system:

“A rectangular sheet of raw steel of non-uniform thickness, length l m (meter), width w m, and material physical attribute set S , should be refined to a rectangular sheet of steel with at least l m in length, at least w m in width, and thickness in the range of t mm $\pm k$ μ m. The minimum thickness of the raw steel is greater than or equal to t mm $- k$ μ m.”

5.1. Step 0: High-level specification of the application environment of the AGC as an RTO.k object and its real-time simulation. Initially, computer-based controllers do not exist and neither do sensors such as speedometers, tensiometers, etc., and actuators such as hydraulic actuators and motor drives because the system engineer has not decided which types to use. As the first step, the system engineer (team) may describe the application environment of the AGC system (i.e., steel pressing factory) as an RTO.k object as depicted in Figure 5.2. This RTO.k is called the *Steel Pressing Factory RTO*. As we mentioned before, the steel factory consists of up to n RPP’s. Hence, the object data store (ODS) of the Steel Pressing Factory RTO consists of the state descriptors for $(0 - n)$ RPP’s. The information kept in all these state descriptors constitutes the information kept in the Steel Pressing Factory RTO.

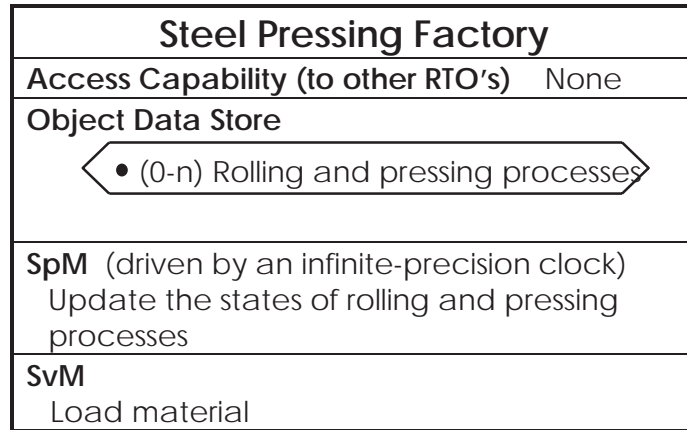


FIG. 5.2. *The steel pressing factory RTO*

The RPP state descriptors are periodically updated by a spontaneous method (SpM). Conceptually, this SpM in the Steel Pressing Factory RTO is *activated continuously* and each of its executions is *completed instantly*. The SpM thus represents the continuous state changes that occur naturally in the real RPP’s. The SvM in the Steel Pressing Factory RTO functions as an interface to “external clients”. The only conceivable client here is the mechanism that inputs the material (i.e. raw steel sheet of non- uniform thickness) to (the POR of) the first RPP.

So far, the Steel Pressing Factory RTO in Figure 5.2 was interpreted as a mere description of the application environment. However, if the activation frequency of each SpM is chosen such that it can be supported by an object execution engine, then the resulting RTO becomes a simulation model. The behavior of the application environment is represented by this simulation model somewhat less accurately than

by the aforementioned description model based on continuous activation of SpM's. In general, the accuracy of an RTO.k object structured simulation is a function of the chosen activation frequencies of SpM's. Note that this style of simulation is **real-time simulation** in which the simulation objects are designed to show the same timing behavior that the simulation targets do [18].

Therefore, the RTO.k object model is effective not only in the multiple-level abstraction of real-time (computer) control systems under design but also in the accurate representation and simulation of the application environments. This means considerable potential benefits to the system engineers.

5.2. Step 1: High-level design of an RPP using the RTO.k object model. After creating the high level specification of the application environment, the system engineer (team) now decides to produce a high-level design of each RPP. For this, the engineer first decomposes the Steel Factory RTO into multiple *RPP RTO's*. Such a decomposition would also involve the introduction of one or more SvM's in each RPP RTO to establish some connections among the RTO's and between the RTO's and the clients outside the factory.

Next, the engineer decides on the types of sensors and actuators to be used. Once those devices are chosen, then the control algorithms for operating the devices and controlling the RPP will be determined. Figure 5.1 already showed all the sensors and actuators chosen. The RPP augmented with chosen sensors and actuators and an imaginary controller can be described as an RTO shown in Figure 5.3. The ODS of this RTO contains state descriptors for the pressing mill and the controller which performs the automatic gauge (thickness) control and the motor speed control. The sensors and actuators are treated as a part of the pressing mill instead of separating them out since their functionality is simple. Therefore, the requirements to be imposed on the computer based controller have been specified in a concrete form, i.e., RPP RTO in Figure 5.3.

The SpM "Update the state of Pressing mill" keeps track of the rotary motions of the POR, WR, and TR, and the linear motion of the steel sheet through the guiding rails between the POR and the TR. It also keeps track of the action of the hydraulic actuator aimed for setting the roll gap between the WR cylinders to the target value ordered by the controller.

The SpM "Update the state of Controller" keeps track of the action of the controller and thus it can be viewed as a core part of the requirement specification for the computer based controller at this state of the controller development.

Again, the RPP RTO contains an SvM representing the operation controlled by an external client of loading material on to the POR. Also, if an RPP has the mechanism for sending processed still rolls to another RPP, then the RTO representing the former RPP should contain the capability for making service calls (calls for SvM "Load material") to the RTO representing the latter RPP. This is the reason for including "NextRPP" in the access capability section of the RPP RTO in Figure 5.3.

5.3. Step 2: Decomposition of the RPP RTO into a pressing mill RTO and a computer-based controller RTO. As the system engineer decomposes the single RTO.k representation of the RPP RTO in Figure 5.3 as a part of more detailed design, a component of the ODS becomes a new RTO.k object. When these new RTO.k objects are created, the SvM's that serve as the front-end interfaces of these new RTO.k objects should also be created. After the decomposition, the AGC system may be composed of a network of two RTO.k objects: the *Pressing Mill RTO* and the *Controller RTO* which represents the desired actions of both SCT and AGCT.

Rolling and Pressing Process
Access Capability (to other RTO's) NextRPP
Object Data Store <ul style="list-style-type: none"> • Pressing mill including POR, WR, TR, sensors, and actuators • Controller (= automatic gauge control + motor speed control)
SpM "Update the state descriptors in ODS" Update the state of Pressing mill Update the state of Controller
SvM Load material

FIG. 5.3. The rolling and pressing process (RPP) RTO

In this process, the requirement specifications associated with the controller may be refined. The Pressing Mill RTO may now describe or simulate the actions of the sensors and actuators, the rotary motions of the POR, WR, and the TR, and the linear motion of the steel sheet between the POR and the TR, more accurately than the RPP RTO did. Similarly, the Controller RTO can describe or simulate the desired gauge control and motor speed control actions more accurately than the RPP RTO did. Now, the Pressing Mill RTO can be viewed as a description or a real-time simulator of the application environment and the Controller RTO as an abstract design or requirement specification of the computer-based controller to be implemented by the computer engineer (team).

5.4. Step 3: Further decomposition of the Pressing Mill RTO and detailed design of the computer-based controller. Further decomposition of the Pressing Mill RTO may produce a network of three different RTO's: the POR RTO, the WR RTO, and the TR RTO as shown in Figure 5.4. Now, these three RTO's can be hosted on three different nodes, if a high frequency simulation is sought after. Here, the WR RTO describes or simulates the rotary motion of the WR, the pressing action of the cylinders, the linear motion of steel through the gap between the WR cylinders, and the operations of the sensors and actuators located in the vicinity of the WR. The POR RTO and the TR RTO describe or simulate their corresponding facilities in similar fashions. Each of these environment RTO's should have SvM's which interface not only with the computer-based controller but also with other environment RTO's.

During this step the computer engineer (team) may produce a more detailed design specification of the computer-based controller by expanding the single RTO design into a network of two RTO's as shown in Figure 5.4: the *Automatic Gauge Controller (AGCT) RTO* and the *Speed Controller (SCT) RTO*. These two RTO's may be hosted on two different computer nodes or on the same computer node. The

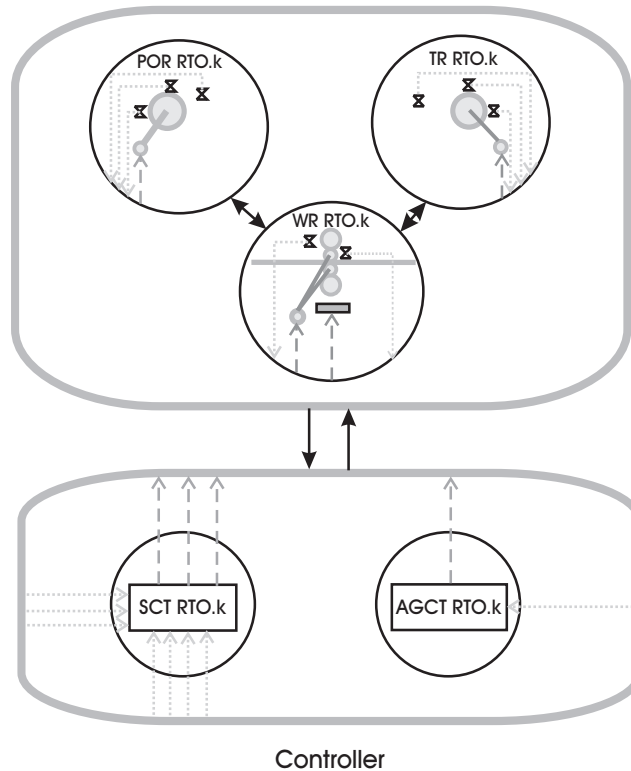


FIG. 5.4. Network of RTO's in a rolling and pressing process

detailed design specification of the SCT RTO is shown in Figure 5.5. The SCT RTO receives from the POR RTO the sensor readings, namely, the current POR velocity, the current POR radius, and the current steel tension in the measurement point between the POR and the WR. It also receives the current WR velocity from the WR RTO. In addition, it receives from the TR RTO some sensor readings such as the current TR velocity, the current TR radius, and the current steel tension in the measurement point between the WR and the TR. The SCT then calculates the velocity required for the POR motor drive, the WR motor drive, and the TR motor drive that satisfies the requirement specification, and outputs these values to appropriate motor drives.

The AGCT RTO receives from the WR RTO some sensor readings such as the current value of load applied to the WR cylinders and the thickness of the steel sheet measured in the entry point of the WR. This RTO then calculates the desired load to be applied to the cylinders.

The AGC system in Figure 5.4 thus consists of five RTO.k objects interconnected. The upper portion depicts the three RTO.k objects that compose the real-time simulator of the application environment. The lower portion depicts the two RTO.k objects that compose the computer-based controller.

The prototype DREAM kernel version 3.0 (to be discussed in the next section) was used in the experimental implementation of this AGC system. The implemented real-time distributed application software consisting of five easily modifiable RTO.k objects runs on a network of two PC's. In this experimental effort, the RTO.k objects were implemented in C++ with the support of the DREAM library v3.0b [16]. The

implemented application software consists of about 2,500 lines of C++ code and the DREAM kernel v3.0 consists of about 15,000 lines of C++ code and an assembly module of about 700 instructions. Other application prototypes implemented in the author's laboratory are much larger. For example, the military command control prototype implementation consists of 9 RTO's and about 20,000 lines of C++ code [21].

The AGC system implemented can be easily expanded with functions such as handling of various alarm conditions including a safe shutdown of the pressing mill. During the development of this prototype and the military command control prototype, we observed that the debugging efforts required were less than 20% of the efforts required during our earlier development of similar but simpler prototypes using the conventional real-time process-structured design methods. The language compilers and the PC platforms used this time were more powerful than the tools used earlier but we feel that still the RTO.k object structuring scheme and the predictable DREAM kernel are the major factors that have caused this significant, if not dramatic, improvement in the design and implementation efficiency.

5.5. Advantages Of the RTO.k Object-Based Design Of Complex Systems. Under the RTO.k object based UID approach, both the computer engineer (team) who produces control computer systems and the system engineer (team) who interfaces with the customers of the computer-embedded application systems and produces precise specifications of requirements to be met by the computer engineer, use the same structuring approach during their systematic construction of specifications.

The computer engineer receives an RTO.k object structured requirement specification as discussed in Section 5.2 and 5.3. The computer engineer initially produces an abstract single RTO.k object design and then proceeds to refine it into a more detailed design which has the structure of an RTO.k object network (e.g., Figure 5.4).

On the other hand, the system engineer first starts with a single RTO.k object representation of the application environment (the RPP) (e.g., Figure 5.2) in which sensors, actuators, and control computer systems are to be embedded. The system engineer gradually refines this single RTO.k representation into an RTO.k network representation (e.g., Figure 5.3), which can be given to the computer engineer as a requirement specification.

Thereafter, the system engineer (or another team) can optionally refine the environment portion of the RTO.k network representation into a real-time simulator of the application environment, e.g., simulator depicted in (the upper half of) Figure 5.4. The environment simulator can then be used for testing the control computer system produced. The environment simulator is a *real-time simulator* which produces sensor data in real-time and takes real-time commands for actuators and simulates the subsequent operations of actuators in real-time. Obviously, this kind of testing yields better coverage than the testing based on non-real-time simulation of the application environment.

6. An Overview of the DREAM Kernel and the DREAM Library.

6.1. The DREAM Kernel. In order for the RTO.k objects to provide guaranteed timely services to external clients, the engine that supports the execution of RTO.k objects must obviously provide guaranteed timely services to the requesting RTO.k objects. In addition to containing a hardware platform, such an execution engine should be formed by a *timeliness-guaranteed operating system*. Existing commercial operating systems are not yet capable of providing guaranteed timely services

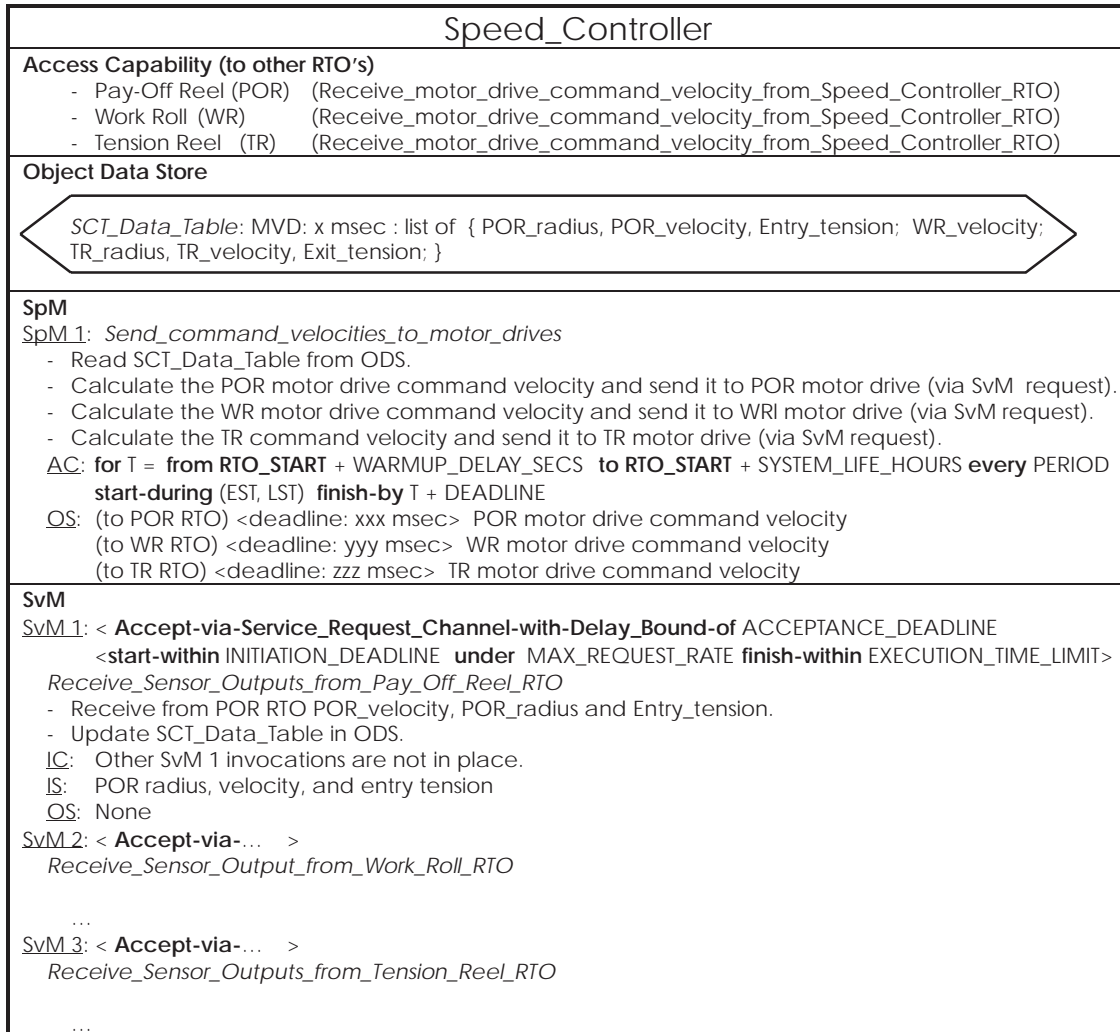


FIG. 5.5. The speed controller RTO

needed by a great deal of real-time application software.

The DREAM kernel has been formulated as a model of an operating system kernel that can support guaranteed timely services not only for RTO.k object structured real-time applications but also for conventional process-structured real-time applications [14, 15, 16]. It thus contains full features of a general-purpose kernel. Therefore, the architectural principles adopted in the DREAM kernel to enable guaranteeing timely services to its clients, i.e., processes and RTO.k objects, are broadly applicable to any situations where real-time operating systems are developed. The DREAM kernel rigidly manages execution times of concurrency units and interrupt handlers in order to provide guaranteed timely services to its clients, i.e., RTO.k objects and real-time processes.

The DREAM kernel guarantees the timely response to service requests by adopt-

ing a unique approach for the layering of its components [14, 15]. This approach is based on the organizational principle called the **time-leasing machine layering**, which is of fundamental nature. Under the time-leasing machine layering principle, the bottom layer (L0 in the DREAM kernel) owns the full power of the hardware machine. So, the bottom layer uses the hardware machine at its own will. The remainder of the hardware machine time after the bottom layer use of the hardware machine, is “leased” to the next upper layer (L1 in the DREAM kernel). The time leasing relationship is recursive, i.e., it is maintained through all the layers that compose the kernel (in the DREAM kernel layers L0-L4). Moreover, there is a tight bound on the amount of machine time that each layer uses. This sometimes implies a bound on the frequency of certain types of interrupts that are accepted.

For supporting the components of RTO.k structured application programs, the DREAM kernel utilizes the same support provided to the components of conventional process-structured real-time application programs. In other words, the DREAM kernel uses the support provided for *processes*, *concurrent-read-&-exclusive-write (CREW) monitors* which are shared data structure monitors for intra-node inter-process communication, and *data field channels (DFC's)* [13, 16, 24] which are real-time logical multicast channels for both intra-node and inter-node inter-process-group message communication to execute the components of an RTO.k-structured application program.

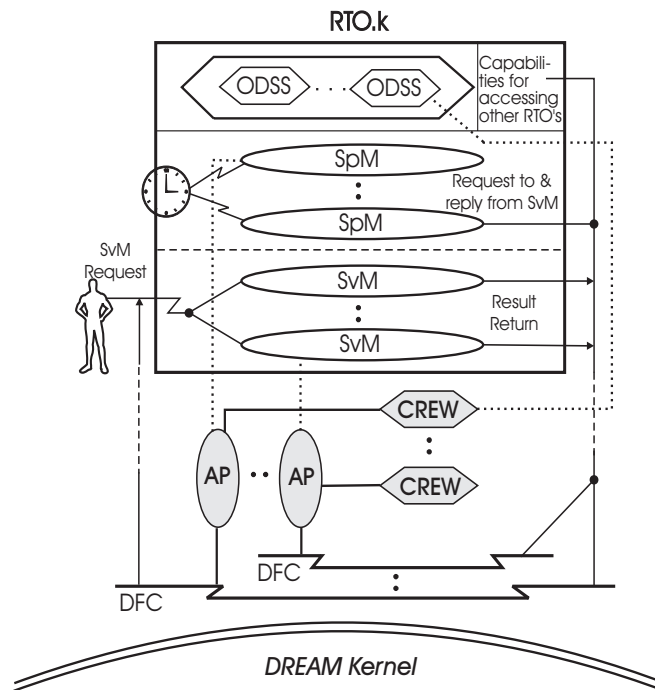


FIG. 6.1. Mapping of an RTO.k object to a conventional process-structured program (Adapted from [15])

The DREAM kernel executes RTO.k object components as follows (as shown in Figure 6.1)

- (1) ODS segments (ODSS's) as CREW monitors,
- (2) Both SpM's and SvM's as application-specific program bodies of processes (called

method execution processes or MEP's),

- (3) Paths for accessing SvM's in the form of DFC's monitored by the SvM's,
- (4) Result-return paths to clients in the form of DFC's monitored by the clients, and
- (5) Capabilities for accessing SvM's in other RTO's in the form of ID's of the DFC's monitored by the SvM's.

6.2. DREAM library support for RTO.k structured programming in C++. The DREAM library greatly reduces the burden of the C++ RTO.k application programmer in the construction of RTO.k objects and in designing kernel service calls (KSC's) to the DREAM kernel [16, 20]. In order to create an RTO.k object, the user should first define an *RTOClass* that contains

- (1) an instance of *RTOAccessCapabilityClass* (representing access rights for SvM's in other RTO's),
- (2) instances of each *ODSSClass* (representing the ODS segments in this RTO),
- (3) instances of each *SpMClass* (representing the SpM's of this RTO), and
- (4) instances of each *SvMClass* (representing the SvM's of this RTO).

The user then creates an instance of the *RTOClass* inside the *application initial process* (AIP). The AIP is the starting point of the entire application in a node.

In order to ease the creation of these user-defined classes, the DREAM library provides a set of pre-defined classes (templates) which can be inherited by the user-defined classes. The set of pre-defined classes includes

- (1) the *BasicRTOAccessCapabilityClass*,
- (2) the *BasicODSSClass* for constructing an ODSS,
- (3) the *BasicSpMClass*, for creating an SpM, and
- (4) the *BasicSvMClass*, for creating an SvM.

Thus, a user-defined *RTOAccessCapabilityClass* can be created easily by inheriting the *BasicRTOAccessCapabilityClass* from the DREAM library and then adding a certain amount of application-specific program components. Other user-defined classes can also be created in similar manners.

Both the latest prototype implementation of the DREAM kernel and the DREAM library will be freely available from the Web site (<http://www.eng.uci.edu/dream/dream-lab.html>) by the time this article is printed.

7. Desirable SEE for RTO.k Object Based Engineering of RTCS's.

7.1. Specification of requirements and abstract designs. The requirement specification which is to be given to the computer engineer in Section 5.2 and 5.3 suggests a new structure for requirement specification. This new structure is based on the RTO.k object structure and practically indistinguishable from the RTO.k structured specification of high-level designs. Also, it enables embedding variable-accuracy representation of application environments in the requirement specifications. Some tools for template-based input will be useful. Simulator engines are another type of valuable tools. A flexible simulator engine should provide both the real-time simulation mode and the logical clock driven simulation mode.

Although the RTO.k object structuring enables modular organization and simulation of specifications, the simulation requires manual design because the RTO.k structuring does not provide sufficient formal specification mechanisms. Integration of the RTO.k structuring and the established formal specification methods (e.g., LOTOS scheme [32], SDL scheme [32], SPECS methodology [25], and state transition model based methods) along with the development of support tools is considered a highly worthwhile effort. Figure 7.1 depicts the symbiotic relationship between the

RTO.k structuring scheme and the formal specification methods.

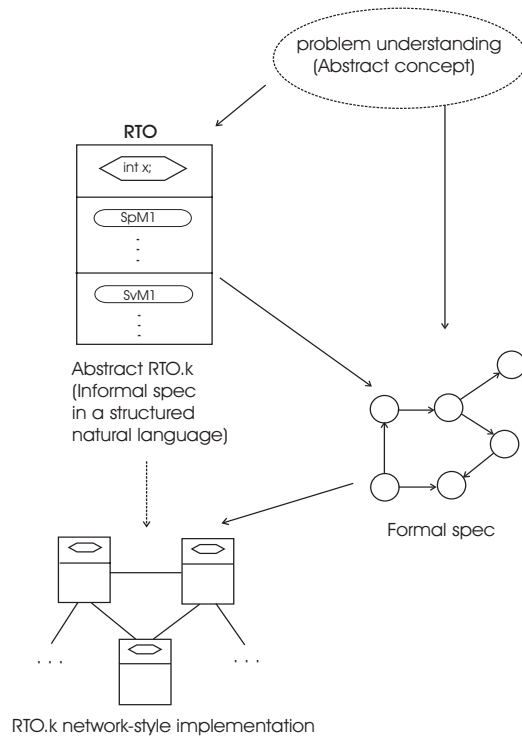


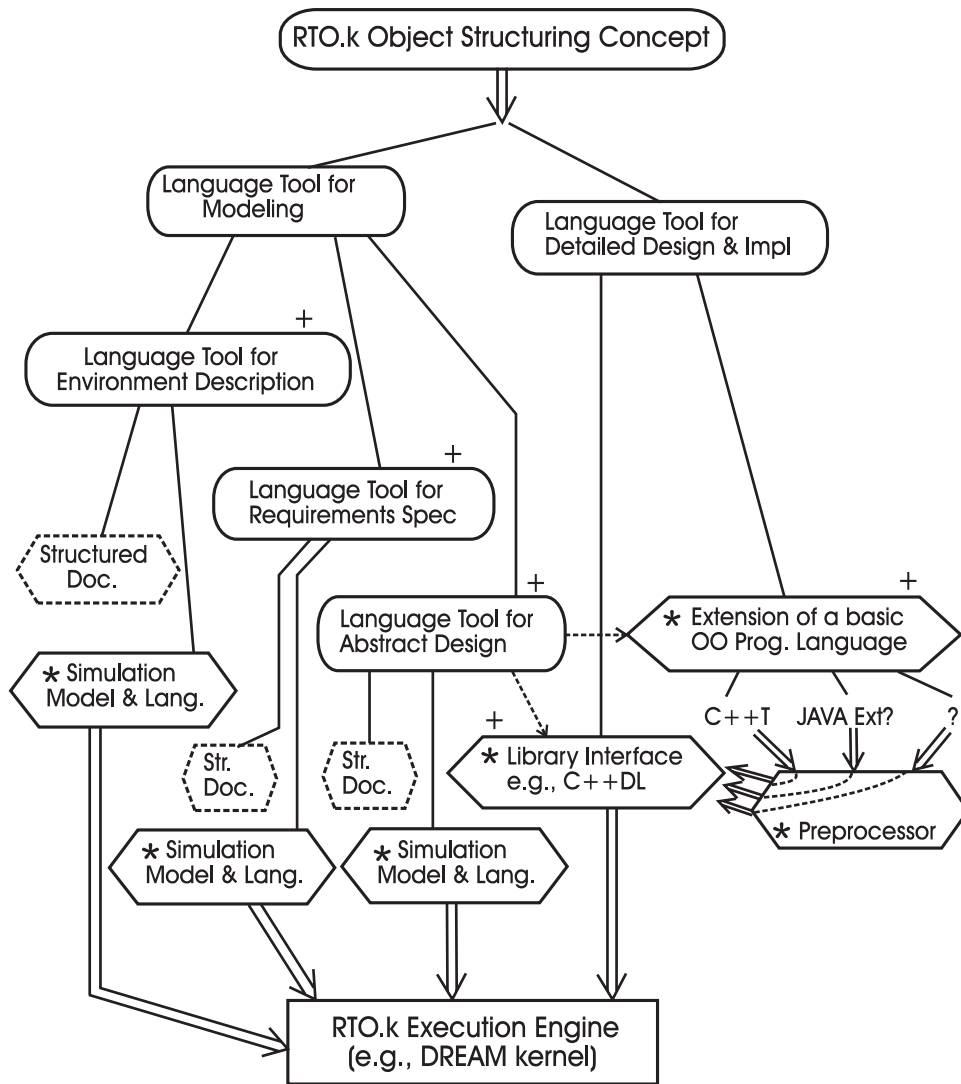
FIG. 7.1. *Symbiotic relationship between the RTO.k structuring scheme and formal specification methods*

7.2. Analysis of the worst-case service time. The most urgently needed among various software engineering tools desired are those for assisting the RTO.k object designer in the process of determining the response time to be guaranteed [17]. Such tools must be capable of making good estimates for worst-case execution times of various segments of object methods. Such tools can also be useful in checking the *execution feasibility* of RTO.k objects, e.g., checking if a group of RTO.k objects can be loaded onto a DCS node without introducing the possibility of any violation of the timing constraints associated with the RTO.k objects.

7.3. Programming language tools. Various language tools that can provide useful aid in exercising the RTO.k object based top-down GT design are listed in Figure 7.2.

An extension of C++ to support RTO.k object programming is one of the most urgent development targets in this area. The author and his research associates are currently developing one such extension named C++T. Our approach will be to convert C++T programs into C++ programs containing DREAM library calls first and then to convert the latter programs into machine programs by use of commercial C++ compilers. Figure 7.2 indicates that there are numerous tools worth developing.

7.4. Operating system (OS) support. At present, the DREAM kernel is little more than a minimal-functionality model of a timeliness-guaranteed operating system kernel supporting RTO.k objects. It can be extended in several major direc-



- + Additional software tools aiding analysis and conversion are desirable
 * Tools for producing executable programs

FIG. 7.2. Language tools aiding the RTO.k object based GT design

tions. Adapting the DREAM kernel to commercial micro-kernel environments is a meaningful direction to pursue. Without investment by industry of sizable efforts to revise and extend their existing operating system products to support RTO.k objects, execution engines usable by common practitioners will not be available anytime soon.

CORBA-compliant [30] support for RTO.k objects is under development at SoHaR, Inc., a small company in Los Angeles, under the support of the US Air Force [29].

Recently a basic scheme for realizing real-time object replicas which possess *time-bounded fault tolerance* capabilities has been formulated [22]. This object-based real-time fault tolerance scheme is called the **primary-shadow RTO.k replication (PSRR)** scheme and has been partially validated through several real-time computing application development experiments conducted in the author's laboratory. The PSRR scheme is an integration of the RTO.k object scheme and the primary-shadow active replication approach established in development of the process-structured real-time fault tolerance schemes [10]. Considering the broadly applicable nature of the PSRR scheme, efficient OS support for the PSRR scheme is considered to be another important topic for future research.

This author believes that in order to achieve significant progress in resource allocation in real-time DCS's, a systematic approach for obtaining from the system engineer a specification of not only the *urgency* but also the *criticality* (i.e., the degree of damage to be incurred to the application mission if not done properly) of each output action of the DCS to the application environment and utilizing the information effectively in allocating resources, must be explored. The **risk incursion function (RIF)**, also called the *benefit loss function (BLF)*, was proposed recently by this author as a guide for globally optimal dynamic resource allocation and trade-off among multiple service quality attributes in real-time DCS's [19]. The RIF is the quantitative specification by the system engineer of the relationship between the loss in the timed value accuracy of each output action from the DCS and the resultant damage/risk incurred to the application mission. The time-value function proposed in [9] can be viewed as one special case of the RIF. Any tool for aiding the system engineer in determining the RIF will be a valuable addition to the SEE for future RTCS engineering. Also, OS mechanisms (e.g., process scheduler) capable of effectively utilizing the RIF in resource allocation are another open topic for future research.

7.5. Testing tools. Although the approach of design-time guarantee of timely services of objects was advocated and the importance of developing the tools for analysis of the worst-case service times was emphasized in this paper, we still need to go through the testing phase. It is obviously needed to check out various logical value manipulation aspects. The testing can even show sometimes the wide gap between the worst-case service time determined through an analysis and the worst-case service time observed during a reasonable length of execution period. In developing testing tools effective in validating real-time OO programs implemented following the GT design paradigm, an interesting question is how to exploit the knowledge generated during the analysis of the worst-case service times. Another issue of greater urgency is how to instrument the real-time OO programs to facilitate systematic testing.

7.6. Reuse support. If RTO.k objects are designed and implemented in conformance with some common interface standards such as CORBA, it will go a long way toward facilitating their reuse. Therefore, development of tools for generating and executing RTO.k objects of which the interfaces conform to the widely accepted standards is a highly meaningful topic for future research.

The inheritance and the polymorphism aspects of the RTO.k object have one more dimension that those of the conventional object has: dealing with timing attributes. For example, when an RTO.k class is inherited, the autonomous activation conditions, deadlines, and other specifications of action timings inside the RTO.k class are all passed on. If the timing attributes are not acceptable in a new application situation, then attempts can be made to exploit polymorphism to effectively replace some parts of the class. A tool aiding the user in checking whether such manipulation of an

RTO.k class introduces inconsistency or execution infeasibility will be of great value and thus a good target of development.

Finally, graphics oriented tools aiding the reuse process are highly desirable.

8. Conclusion. The essence of the GT design paradigm advocated in this paper and also in earlier publications by the author and the RTO.k object structuring scheme for its realization, is to realize real-time computing in a general manner not alienating the main-stream computing industry and yet enabling the system engineer to confidently produce certifiable RTCS's for safety-critical applications. This author believes that time is ripe for vigorously pursuing this idealistic approach. We expect that active development of real-time OO system structuring techniques and operating systems with guaranteed timely service capabilities, such as the DREAM kernel, will become conspicuous industry trends before year 2000. However, maturity of these ingredients for realization of the GT design paradigm will be achieved much later than year 2000.

The development of software engineering tools discussed in Section 7 will require multiple years of massive efforts but it is essential to make the GT design a common practice.

Acknowledgment. The research work reported here was supported in part by US Navy, NSWC Dahlgren Division under Contract No. N60921-92-C-0204, in part by the University of California MICRO Program under Grant No. 96-169, in part by the California Transportation Department via the UCI Institute for Transportation Studies, in part by Hitachi, Ltd, in part by ETRI/SERI, and in part by LG Electronics.

REFERENCES

- [1] A. ATTOUI AND M. SCHNEIDER, *An Object Oriented Model for Parallel and Reactive Systems*, Proc. IEEE CS 12th Real-Time Systems Symp., pp. 84-93, 1991.
- [2] T. BIHARI, P. GOPINATH, AND K. SCHWAN, *Object-Oriented Design of Real-Time Software*, Proc. IEEE CS 10th Real-Time Systems Symp., pp. 194-201, 1989.
- [3] G. BOOCH, *Object-Oriented Design*, Benjamin Cummings, CA, 1991.
- [4] M. CHAMPLAIN, *Synapse: A Small and Expressive Object-based Real-time Programming Language*, ACM SIGPLAN Notices, Vol. 25, No. 5, pp. 124-134, 1990.
- [5] O. J. DAHL, *Hierarchical Program Structuring*, in Dahl, Dijkstra, & Hoare eds., *Structured Programming*, Academic Press, NY, 1972.
- [6] E. W. DIJKSTRA, *Notes on Structured Programming*, in Dahl, Dijkstra, & Hoare eds., *Structured Programming*, Academic Press, NY, 1972.
- [7] J. HERNANDEZ AND J. A. SANCHEZ, *RT-MODULA2: An embedded in MODULA2 Language for writing Concurrent and Real Time programs*, ACM SIGPLAN Notices, Vol. 27, No. 2, pp. 26-36, Feb. 1992.
- [8] ISHIKAWA, Y., TOKUDA, H., AND C. W. MERCER, *An Object-Oriented Real-Time Programming Language*, IEEE Computer, pp. 66-73, Oct. 1992.
- [9] E. D. JENSEN, C. D. LOCKE, AND H. TOKUDA, *A Time-Value Driven Scheduling Model for Real-Time Operating Systems*, Proc. IEEE CS Symp. on Real-Time Systems, Nov. 1985.
- [10] K.H. KIM, *Action-level Fault Tolerance*, in S. H. Son ed., *Advances in Real-Time Systems*, Ch. 17, Prentice Hall, 1994, pp. 415-434.
- [11] K.H. KIM AND H. KOPETZ, *A Real-Time Object Model RTO.k and an Experimental Investigation of Its Potentials*, Proc. 1994 IEEE CS Computer Software and Applications Conf. (COMPSAC), Taipei, pp. 392-402, Nov. 1994.
- [12] KIM, K.H. ET AL., *Distinguishing Features and Potential Roles of the RTO.k Object Model*, Proc. WORDS '94 (IEEE Computer Society's 1st Workshop on Object-oriented Real-Time Dependable Systems), Dana Point, pp. 36-45, Oct. 1994. (the proceedings published in 1995).
- [13] K.H. KIM, K. MORI, AND H. NAKANISHI, *Realization of Autonomous Decentralized Computing with the RTO.k Object Structuring Scheme and the HU-DF Inter-Process-Group Commu-*

- nication Scheme*, Proc. 1995 IEEE CS's 2nd Int'l Symp. on Autonomous Decentralized (ISADS 95), Phoenix, AZ, pp. 305-312, Apr. 1995.
- [14] K.H. KIM, *Towards New-Generation Real-Time Object-Oriented Computing*, Proc. FTDCS '95 (IEEE Computer Society's 5th Workshop on Future Trends of Distributed Computing Systems), Cheju Island, pp.520-529, Aug. 1995.
- [15] K.H. KIM ET AL., *A Timeliness-Guaranteed Kernel Model - DREAM Kernel and Implementation Techniques*, Proc. 1995 Int'l Workshop on Real-Time Computing Systems and Applications (RTCSA 95), Tokyo, Japan, pp.80-87, Oct. 1995.
- [16] K.H. KIM, K.H., C. SUBBARAMAN, AND Y. KIM, *The DREAM Library Support for PCD and RTO.k programming in C++*, Proc. WORDS '96 (IEEE Computer Society 2nd Workshop on Object-oriented Real-Time Dependable Systems), Laguna Beach, pp. 59-68, Feb. 1996.
- [17] K.H. KIM, *Towards Designing RTO.k Structured Server Objects with Service Time Guarantee*, Proc. SEKE '96 (8th Int'l Conf. on Software Engineering & Knowledge Engineering), Lake Tahoe, NV, pp. 522-528, June 1996.
- [18] K.H. KIM, C. NGUYENM, AND C. PARK, *Real-Time Simulation Techniques Based on the RTO.k Object Modeling*, Proc. COMPSAC '96 (IEEE CS Software & Applications Conf.), Seoul, pp. 176-183, Aug. 1996.
- [19] K.H. KIM ET AL., *An Experimental Investigation of the Potential of BLF-driven Scheduling of Real-time Threads*, Proc. 2nd IEEE Int'l Conf. on Engineering of Complex Computer Systems (jointly with CSES AW '96, RTAW '96, & SES '96), Montreal, Canada, pp. 60-67, Oct. 1996.
- [20] K.H. KIM, L. BACELLAR, AND C. SUBBARAMAN, *Support for RTO.k Object Structured Programming in C++*, Proc. 21st IFAC/IFIP Workshop on Real-Time Programming (WRTP '96), Gramado, RS, Brazil, pp. 17-22, Nov. 1996 (An expanded version to appear in Control Engineering Practice, an IFAC Journal, 1997).
- [21] K.H. KIM, Y.S. KIM, AND H.J. KIM, "An RTO.k Object Based Uniform Integrated Design of Real-Time Computing Systems and their Application Environment Simulators, Proc. 2nd World Conf. on Integrated Design and Process Technology, IDPT-Vol.2, Austin, TX, pp. 106-113, Dec.1996.
- [22] K.H. KIM AND C. SUBBARAMAN, *Fault-Tolerant Real-Time Objects*, to appear in Communications of ACM, Jan. 1997.
- [23] H. KOPETZ AND K.H. KIM, *Temporal Uncertainties in Interactions among Real-Time Objects*, Proc. IEEE Computer Society's 9th Symp. on Reliable Distributed Systems, Huntsville, AL, pp. 165-174, Oct. 1990.
- [24] K. MORI, *Autonomous Decentralized Systems: Concept, Data Field Architecture, and Future Trends*, Proc. IEEE CS Int'l Symp. on Autonomous Decentralized Systems (ISADS 93), Kawasaki, Japan, pp. 28-34, Mar. 1993.
- [25] R. REED, W. BOUMA, J.D. EVANS, M. DAUPHIN, AND M. MICHEL EDS., *SPECS - Specification and Programming Environment for Communication Software*, North-Holland, 1993.
- [26] J. RUMBAUGH, ET AL., *Object-Oriented Modeling and Design*, Prentice Hall, NJ, 1991.
- [27] B. SELIC, G. GULLEKSON, AND P.T. WARD, *Real-Time Object-Oriented Modeling*, John Wiley & Sons, New York, 1994.
- [28] S.K. SHRIVASTAVA AND A. WATERWORTH, *Using Objects and Actions to provide Fault Tolerance in Distributed, Real-Time Applications*, Proc. IEEE CS 12th Real-Time Systems Symp., pp. 276-285, 1991.
- [29] E.H. SHOKRI ET AL., *An Approach for Adaptive Fault-Tolerance in Object-Oriented Open Distributed Systems*, to appear in Proc. WORDS '97 (IEEE Computer Society 3rd Workshop on Object-oriented Real-Time Dependable Systems), Newport Beach, Feb. 1997.
- [30] R. SOLEY ED., *Object Management Architecture Guide*, 3rd edition, John Wiley & Sons, New York, 1995.
- [31] K. TAKASHIO AND M. TOKORO, *DROL: An Object-Oriented Programming Language for Distributed Real-Time Systems*, Proc. OOPSLA, pp. 276-294, 1992.
- [32] K.J. TURNER ED., *Using Formal Description Techniques - An Introduction to Estelle, LOTOS, and SDL*, Wiley Interscience, 1993.
- [33] V. WOLFE, S. DAVIDSON, AND I. LEE, *RTC: Language Support For Real-Time Concurrency*, Proc. IEEE CS 12th Real-Time Systems Symposium, pp. 43-52, 1991.

Received Dec. 13, 1996