Object-Oriented Application Framework Design for Real-Time Systems

Pao-Ann Hsiung
Institute of Information Science
Academia Sinica, Taipei, Taiwan.

Abstract

In this work, we present our experiences in designing an object-oriented application framework for real-time systems, called SESAG. A components-patterns view is provided so that application developers may easily use the framework to design his/her applications. SESAG consists of five components: *Specifier*, *Extractor*, *Scheduler*, *Allocator*, and *Generator*. Within SESAG, several design patterns have been proposed for real-time systems. Experiences of using SESAG show a significant increase in design productivity through design reuse, and a decrease in design time and effort.

1 Introduction

Real-time systems such as telecommunications, avionics, multimedia, robotics, ... impose timing constraints on executing tasks. Violation of such constraints usually result in an incorrect system. Hence, appropriate task scheduling and resource allocation become a crucial part of all real-time systems design. It is here that object-oriented application frameworks can be taken advantage of for reusing classes and design strategies. Corresponding to different specification styles and scheduling policies, different design patterns can be proposed. Combining design patterns and components, a novel object-oriented application framework called SESAG is proposed for the design of real-time systems. SESAG helps designers of real-time systems increase productivity, decrease design time and effort, enhance manageability, and increase design reuse.

A *real-time system* is generally specified as a collection of *tasks* which might share resources. The tasks are usually independent and periodic. Execution time, period, deadline, type of priority and resource requirements are specified for each task. Hard real-time systems do not allow the violation of any timing constraint. Soft real-time systems strive to minimize deadline violations. To statically guarantee satisfaction of all timing constraints, the tasks must be scheduled using *priority-based* scheduling algorithms such as rate-monotonic (RM) [16], earliest-dealine first (EDF) [16], mixed-priority (MP) [16], pin-wheel, etc. or using *timed-based* scheduling algorithms.

An object-oriented application framework (OOAF) is a reusable, “semi-complete” application that can be specialized to produce custom applications [12]. OOAF are application-domain specific reuse methods, such as user interfaces or real-time avionics. Examples include MacApp, ET++, Interviews, ACE, Microsoft’s MFC and DCOM, Javasoft’s RMI and implementation of OMG’s CORBA. Frameworks can be further distinguished by their scope into *system infrastructure frameworks* (used internally within a software organization), *middleware integration frameworks* (used to integrate distributed applications and components), and *enterprise application frameworks* (used in application-specific domains) [5].

The rest of this paper is organized as follows. Section 2 gives some previous and related work on applying object-oriented technology to real-time system design. Section 3 describes the SESAG framework, application development strategy, and an example application. Section 4 gives the final conclusion.

2 Previous Work

Object-oriented technology has been used in the development of real-time systems for quite some time now. Research literatures have shown how the concept of objects in real-time systems can be useful. Object-oriented real-time (OORT) system models such as MO2 [2], evaluation taxonomy such as in [9], object-oriented real-time language design [11, 1], concurrency exploitation in OORT systems using metrics-driven approach [24], checking time constraints [7], and verification of function and performance for OORT systems [4] are some of the recent work on applying OO technology to real-time system design.

Although object-oriented technology has been applied to the design of real-time systems in several proposed work, but there has been little work on the development of an OOAF for real-time system application design, except for a single one called *Object-Oriented Real-Time Sys-*

tem Framework (OORTSF) [21, 14]. OORTSF is a simple framework and only lists the classes used in the development of real-time applications. No design patterns have been proposed specific to real-time system application design. This results in a difficult comprehension of the collaboration among the classes. How exactly is a real-time application developed using OORTSF is not clear from the work. It is also unclear how is the application code generated from OORTSF, how are the resources allocated, and how are the tasks scheduled. Further, the flexibility of specifying real-time objects, the ease of using OORTSF, the benefits of applying OORTSF, and other issues related to OOAFs are not described in the work. No example application was given for OORTSF in [21, 14]. According to the knowledge of the author, besides OORTSF, there is no other work on *enterprise application frameworks* devoted to *general* real-time system application development. As far as *middleware integration frameworks* are concerned, there has been a TAO Real-Time Object Request Broker (ORB) proposed by Schmidt recently [20]. We mainly compare our work with OORTSF since they belong to the same class of enterprise application frameworks.

In contrast, the SESAG framework proposed in this paper proposes design patterns specific to real-time applications, provides flexibility in specification of application objects, describes how the tasks are scheduled, how the resources are allocated, the hierarchy of code generated is explained clearly, and lastly the application development strategy is also described. The specific classes used and the interface are described in a related work [10]. An example application is given at the end of the paper to support our claims on reduced design efforts through SESAG.

3 The SESAG Framework

SESAG derives its name from the five components that constitute our object-oriented application framework for real-time system application design. The five components, ordered by the sequence in which they are used, are *Specifier*, *Extractor*, *Scheduler*, *Allocator*, and *Generator*. SESAG is illustrated in Fig. 1. The notation used in this paper is based on that proposed in Rumbaugh's Object Modeling Technique (OMT) [19]. The order of the SESAG components used by a designer to design a system is: Specifier, Extractor, Scheduler, Allocator, and Generator. Application domain objects are specified using the Specifier. Real-time constraints are either specified separately or coupled with the application domain objects. In the latter case, Extractor is used for extracting constraints. Extractor is also used to extract tasks from the given domain objects. Scheduler schedules the tasks using some scheduling algorithm and Allocator allocates resources among the tasks that are running concurrently. Finally, Generator is used to generate the application code based on the decisions made in the other components.

3.1 Specifier

This component is the main interface between application domain objects and SESAG. *Application domain objects* are those objects that constitute the application that the designer desires to design. Three design patterns are used in this component: *Objects-with-Constraints*, *Objects-without-Constraints*, and *Tasks-with-Constraints*. We call them design patterns because often the real-world objects are not specified as tasks, whereas a real-time system application is generally described in terms of a set of canonical tasks. This semantic gap has become a design issue [7] and topic of research [1] for the object-oriented model of real-time systems. The three design-patterns correspond to how SESAG provides the designer with the flexibility of choosing either (1) to specify application domain *objects* with constraints *coupled* to the objects, or (2) to specify application domain *objects* with constraints as a *separate* entity, or (3) to specify real-time *tasks* with constraints. The first two design patterns do not explicitly specify what the tasks are. This specification is left to SESAG and SESAG accomplishes it using Extractor as will be described in the next subsection. Timing constraints coupled to the objects are specified as *annotations* to methods [7]. The specification language could be C++ with annotations [7], a real-time extension of C++ called RTC++ [11], ARTS/C and ARTS/C++ used in the ARTS real-time distributed operating system kernel [23], real-time Euclid [13], real-time Mentat [8], or FLEX [15]. Currently, only C++ with annotations is supported in SESAG. Future implementations will include RTC++ and other languages.

3.2 Extractor

This component transforms the specification provided by the designer within Specifier into a uniform intermediate format suitable for schedulability analysis. The intermediate format is necessary since Specifier allows the designer several choices of specifying his/her application. The main job of Extractor is to extract important information from the objects and formulate them into two parts: a Task-Table object and a Call-Graph object. All task-related information are instantiated into a Task-Table object for future reference. This object consists of the task index, the method name, its execution time, period, deadline, type of priority (fixed or dynamic), and its resource requirements. The resource requirement is specified as an real-numbered vector, where each element corresponds to some system resource such as memory, processor utilization, . . . and the real-number corresponds to the amount of each resource required by the particular task. System resources are specified by the designer within Specifier through instantiation of application domain objects. Extractor also generates a Call-Graph object which is a directed graph $G = (V, E)$, nodes in V represent tasks and arcs in E represent the call relationships between two tasks. This graph is useful

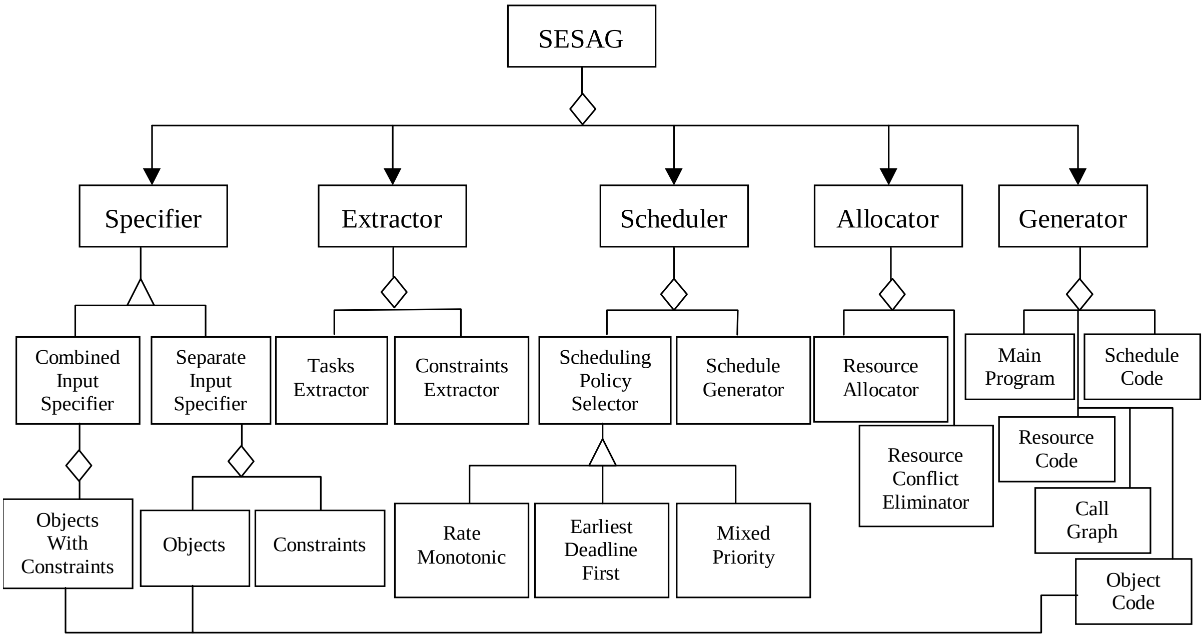

Figure 1: Components-Patterns View of SESAG

for schedulability test, resource allocation, scheduling, and conflict resolution.

3.3 Scheduler

The Call-Graph and the Task-Table objects instantiated in Extractor is scheduled into a feasible application by Scheduler. As described in Section 2, there are many priority-based scheduling policies such as rate-monotonic, earliest-deadline first, mixed priority, pin-wheel, et al. It is sometimes evident from the application, as to which scheduling policy should be applied. But, in most applications, the prime concern is the satisfaction of the timing constraints, irrespective of which scheduling algorithm is applied. Scheduler includes a design pattern similar to the *Strategy Pattern* [6] adapted to real-time systems. In this pattern, one and only one scheduling algorithm must be chosen from the set of all scheduling algorithms for scheduling the tasks in the Call-Graph and the Task-Table.

This component mainly consists of two parts: a Policy Selector (PS) and a Schedule Generator (SG). The designer can choose to assign a particular scheduling policy he deems fit or the designer can also choose to allow SESAG determine automatically the right choice. The choice is made by performing schedulability tests with respect to each scheduling algorithm. One of the scheduling algorithms in the successful cases is then selected as the automatic decision result. This selection can be arbitrary or based on some criteria such as the shortest schedule length (i.e., the shortest scheduled time). Currently, SESAG leaves this option to the designer. Schedule Generator generates the actual start/end timing of each task based on the schedule policy chosen and on the Call-Graph constraints such as precedence relationships.

Before moving on to the next component, one must note that the set of tasks in the Task-Table object are not independent, they have precedence/succession relationships as given in the Call-Graph, so the set of tasks that are scheduled to run concurrently is, in fact, not a fixed one. In the scheduling terminology, this means that the *initial phase* of each task is constrained by the latest end-time of all of its predecessors in the Call-Graph. Thus, another well-known problem arises here, namely the *priority inversion problem*. When higher priority tasks are blocked from execution due to resources being held by lower priority tasks. We adopt the priority inheritance approach [22] to solve this problem. The tasks are all assumed to be preemptive as required by the scheduling policies.

3.4 Allocator

Resource allocation is handled by this component, which is composed of two modules: a Resource Allocator and a Conflict Eliminator. The scheduled Call-Graph does not yet contain any resource information. It is in this component that resources are allocated to each task based on its resource requirements recorded in the Task-Table. It may happen that certain tasks, scheduled to be simultaneously executing, conflict in their resource requirements. The last arriving task (i.e., the task with the latest starting

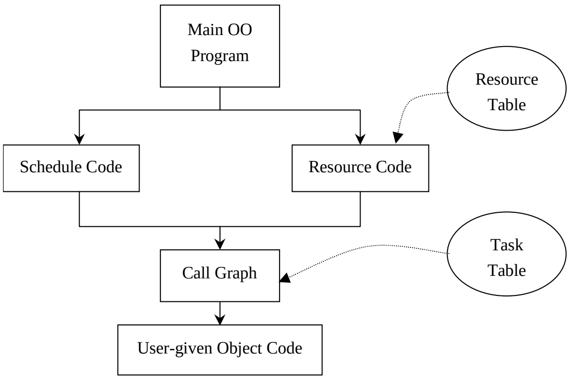

Figure 2: Code Hierarchy in SESAG

phase) is delayed and rescheduled. If more than one conflicting tasks start at the same time, then the task that has a smaller requirement of the conflicting resources is delayed for rescheduling. If no partial order can be assigned to the resource requirements of two or more conflicting tasks then an arbitrary choice of task is made for delay and rescheduling. Conflicts in resource requirements of scheduled tasks are thus eliminated and the resulting scheduled Call-Graph is a feasible one for code generation.

3.5 Generator

The final component of SESAG is responsible for generating the OO code for the real-time application under development by the designer. The hierarchical structure of the generated code is shown in Fig. 2. It consists of mainly five parts: the main OO program, the schedule code, the resource allocation code, the tasks Call-Graph code and the user-given domain object code. The hierarchy is a *calling* hierarchy, that is, the main OO program calls schedule code functions and the resource allocation code functions, which in turn call the call-graph code methods, and finally the user-given object code is called for execution. Two auxiliary codes used for reference are Resource-Table and Task-Table, which contain all the information for system resources and tasks.

The main OO program maintains a global clock which is used for recording progress in the developed system or application. It also contains exception handles to error recovery, fault handling, and other mechanisms to handle exceptions such as constraint violations. The main program ensures that the system is always in an acceptable state. This is achieved by calling the schedule code functions and resource handling functions in the resource allocation code.

The schedule code is an implementation of the actual scheduling of the tasks in the Call-Graph and after any resource conflicts are eliminated. This code depends on the scheduling policy selected either by the user or by SESAG. The schedule code consists of the actual time a task must start execution and the time it should terminate. Everything is settled statically for optimal performance and satisfaction of timing constraints in real-time systems.

Resources related information are all recorded in the Resource-Table object. Resource allocation code is responsible for accepting resource allocation/deallocation requests, transmitting them to the resource codes (or the actual physical resources in case of a hardware real-time system), handling exceptional situations such as dynamic resource failure and recovery, and resource conflict handling. Although resource conflicts among tasks were eliminated statically in the scheduled Call-Graph (refer to the Allocator component of SESAG), yet conflicts may still arise in exceptional situations when resources become faulty, when tasks violate timing constraints due to external environmental disruptions, ...

The Call-Graph code is an implementation of the resource-allocated, scheduled Call-Graph object. This code is required in spite of the calling scheme of tasks (method calls) being implicit in the user-given domain object code, because scheduling information and resource requests are, respectively, not given and not explicit in the object code. The object code is just the user-given code. The hierarchical structure of the code generated by SESAG has many advantages including easy debugging, code modularization, and explicit interface with user-given code.

Due to page-limit, the actual classes used in SESAG are not presented here. Figure 3 illustrates the development strategy. A user of SESAG (also called the designer) begins with specifying his/her system. Either application domain objects may be defined or tasks directly input by instantiating the Task-Table class and the Call-Graph class. When application domain objects are specified, Task-Table and Call-Graph are instantiated by SESAG through the Extractor component. An example of real-time applications has been developed using SESAG.

3.6 Avionics Example

An illustrative example is an avionics system application: digital flight control [3]. The 24 tasks in this example are specified as shown in Table 1 [18, 3, 17]. The hardware resource for executing these tasks is the SIFT (Software-Implemented Fault-Tolerance) computer [18]. We consider 8 processors, with each processor having an instruction execution rate of 0.5 MIPS and an address space of 64 Kbytes. Since the total utilization [3] of each task does not exceed the rate-monotonic scheduling upper bound of 0.693 [16], rate-monotonic scheduling is used. This application when developed with SESAG took only one week for a real-time system designer. The same designer took approximately five weeks to design the same application. This is because a lot of things need only be specified into SESAG without caring for the details such as how tasks are scheduled, how resources are allocated, etc.

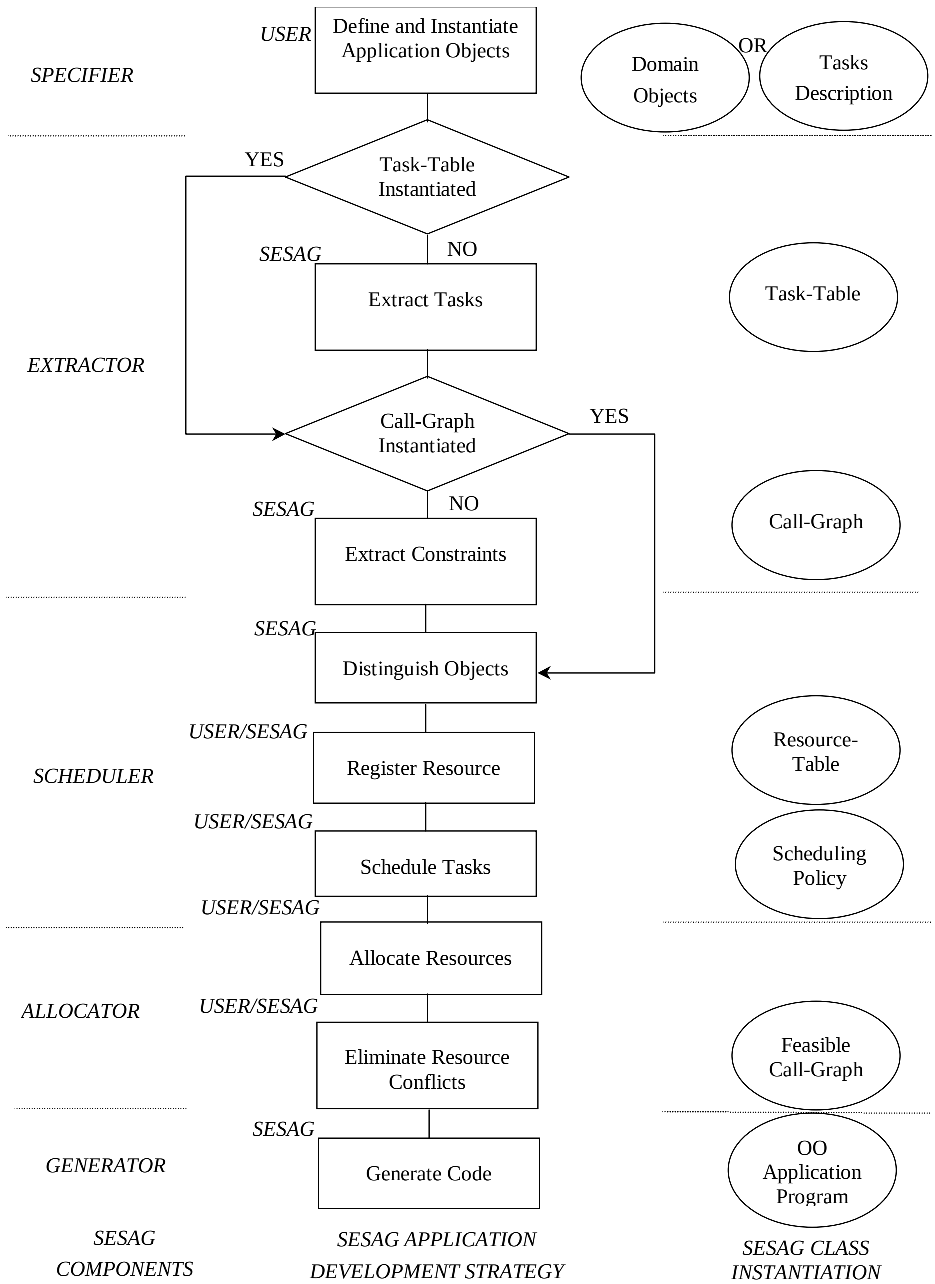

Figure 3: Application Development using SESAG

Table 1: Avionics Example: Digital Flight Control Tasks [3]

Index	Task Description	Execution Time (ms)	Period (ms)	Utilization	Memory
1	Attitude Control	2.456	50.00	0.04912	2,075
2	Flutter Control	0.276	4.00	0.06900	92
3	Gust Control	0.116	4.17	0.02784	60
4	Autoland	0.684	6.25	0.10944	1,025
5	Autopilot	0.400	200.00	0.00200	250
6	Attitude Director	5.120	33.33	0.15360	1,310
7	Inertial Navigation	2.700	40.00	0.06750	2,250
8	VOR/DME	1.540	200.00	0.00700	300
9	Omega	1.600	200.00	0.00800	505
10	Air Data	0.400	200.00	0.00200	135
11	Signal Processing	3.500	5000.00	0.00070	315
12	Flight Data	11.040	200.00	0.05520	550
13	Airspeed	1.098	62.50	0.01757	430
14	Graphics Display	7.950	125.00	0.06360	6,250
15	Text Display	3.800	100.00	0.03800	9,340
16	Collision Avoidance	0.064	1.49	0.04288	1,150
17	Onboard Communication	0.056	4.00	0.01400	705
18	Offboard Communication	0.310	250.00	0.00124	687
19	Data Integration	0.720	250.00	0.00288	1,300
20	Instrumentation	5.584	200.00	0.02792	1,900
21	System Management	4.640	2000.00	0.00232	950
22	Life Support	4.640	2000.00	0.00232	950
23	Engine Control	7.194	30.30	0.23740	1,500
24	Executive	0.400	200.00	0.00200	1,100

Recalculated from [3]

4 Conclusion

An object-oriented application framework, called SESAG, was proposed for real-time systems application development. A components-patterns view of SESAG was presented. Design patterns related to real-time system design were proposed and implemented in SESAG. Several application examples were developed using SESAG. Due to page-limit, only one has been described in this paper. The example has shown how design time is significantly reduced due to a large extent of object and code reuse from SESAG. Besides reuse, SESAG also automates several design phases of real-time system application development, including tasks extraction, constraints extraction, scheduling, and resource allocation. All of these design phases were very pain-staking and laborious originally when a designer had to develop either from scratch or using different specialized tools such as scheduling analysis tool, allocation tool, etc.

SESAG can be easily extended since new specification languages, scheduling algorithms, etc. can always and easily be integrated into it. Future extensions will include RTC++ support and other scheduling algorithms. More examples will also be developed using SESAG. One major work to be accomplished in this field is the formal definition of *design effort* and how the metric could be used to compare different frameworks. This research work is still on-going.

References

[1] B. Achauer. Objects in real-time systems: Issues for language implementors. *ACM OOPS Messenger*, 7(1):21–27, January 1996.

[2] A. Attoui and M. Schneider. An object oriented model for parallel and reactive systems. In *Proc. Real-Time Systems Symposium*, pages 84–93, December 1991.

[3] J. A. Bannister and K. S. Trivedi. Task allocation in fault-tolerant distributed systems. *Acta Informatica*, 20(3):261–281, 1983.

[4] J.C. Browne. Object-oriented development of real-time systems: Verification of functionality and performance. *ACM OOPS Messenger, Special Issue on Object-Oriented Real-Time Systems*, 7(1):59–62, January 1996.

[5] M. Fayad and D.C. Schmidt. Object-oriented application frameworks. *Communications of the ACM, Special Issue on Object-Oriented Application Frameworks*, 40(10), October 1997.

[6] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA: Addison-Wesley, 1995.

[7] M. Gergeleit, J. Kaiser, and H. Streich. Checking timing constraints in distributed object-oriented programs. *ACM OOPS Messenger, Special Issue on Object-Oriented Real-Time Systems*, 7(1):51–58, January 1996.

[8] A.S. Grimshaw, A. Silberman, and J.W.S. Liu. Real-time mentat, a data-driven object-oriented system. In *Proc. of IEEE Globecom*, pages 141–147, November 1989.

[9] D.K. Hammer, L.R. Welch, and O.S. van Roosmalen. A taxonomy for distributed object-oriented real-time systems. *ACM OOPS Messenger, Special Issue on Object-Oriented Real-Time Systems*, 7(1):78–85, January 1996.

[10] P.-A. Hsiung. RTFrame: An object-oriented application framework for real-time systems. In *Proc. of the 27th International Conference on Technology of Object-Oriented Languages and Systems (accepted for presentation)*, September 1998.

[11] Y. Ishikawa, H. Tokuda, and C. W. Mercer. Object-oriented real-time language design: Constructs for timing constraints. *ACM SIGPLAN Notices, ECOOP/OOPSLA'90 Proceedings*, 25(10):289–298, October 1990.

[12] R. Johnson and B. Foote. Designing reusable classes. *Journal of Object-Oriented Programming*, 1(5):22–35, June 1988.

[13] E. Kligerman and A.D. Stoyenko. Real-time euclid: A language for reliable real-time systems. *IEEE Trans. on Software Engineering*, SE-12(9):941–949, September 1986.

[14] T.Y. Kuan, W.B. See, and S.J. Chen. An object-oriented real-time framework and development environment. In *Proc. OOPSLA'95 Workshop #18*, 1995.

[15] K.-J. Lin and S. Natarajan. Expressing and maintaining timing constraints in flex. In *Proc. Real-Time Systems Symposium*, pages 96–105, 1988.

[16] C.L. Liu and J.W. Layland. Scheduling algorithms for multiprogramming in a hard-real time environment. *Journal of the Association for Computing Machinery*, 20(1):46–61, January 1973.

[17] M. Potkonjak and W. Wolf. A methodology and algorithms for the design of hard real-time multi-tasking asics. *ACM Trans. on Design Automation of Electronic Systems*, 5(1):to appear, January 2000.

[18] R.S. Ratner, E.B. Shapiro, H.M. Zeidler, S.E. Wahlstrom, C.B. Clark, and J. Goldberg. *Design of a Fault-Tolerant Airborne Digital Computer, vol. 2, Computational Requirements and Technology*. SRI Final Report, NASA Contract NAS1-10920, 1973.

[19] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modeling and Design*. Prentice-Hall, 1991.

[20] D.C. Schmidt. Applying design patterns and frameworks to develop object-oriented communication software. *Handbook of Programming Languages*, I, 1997.

[21] W.-B. See and S.-J. Chen. High-level reuse in the design of an object-oriented real-time system framework. In *Proc. International Computer Symposium*, pages 363–370, December 1996.

[22] L. Sha, R. Rajkumar, and J.P. Lehoczky. *Priority Inheritance Protocols: An Approach to Real-Time Synchronization*. Technical Report CMU-CS-87-181, Computer Science Department, Carnegie Mellon University, November 1987.

[23] H. Tokuda and C.W. Mercer. Arts: A distributed real-time kernel. *Operating Systems Review*, 23(3):29–53, July 1989.

[24] L. R. Welch. A metrics-driven approach for utilizing concurrency in object-oriented real-time systems. *ACM OOPS Messenger*, 7(1):70–77, January 1996.