Schedulability in Model-based Software Development for Distributed Real-Time Systems

Stephen S. Yau and Xiaoyong Zhou
Computer Science and Engineering Department
Arizona State University
Tempe, AZ 85287, USA
{yau, xzhou}@asu.edu

Abstract

Schedulability of distributed real-time system has been studied extensively. However, due to the discrepancies the reference model used in scheduling analysis and the object-oriented model, the results cannot be easily used for distributed real-time object-oriented software development. Even if object-oriented models can be used to conduct schedulability analysis successfully, there still lacks an effective approach to validate the implementation with the design. Model-based approach alleviates the discrepancies between models of different stages and the implementation by automatically transforming models at different granularity until the code generation for the implementation from the generated design. In this paper, an approach to incorporating schedulability analysis reference diagrams into the existing framework for model-based software development will be presented. It will improve the predictability of a distributed real-time system as well as increase the capability of model refinements and code generation using the new reference diagrams and schedulability analysis results to generate code for implementing scheduling and synchronization aspects of the distributed real-time system.

Keywords: Distributed real-time system, end-to-end timing constraint, aspect-oriented programming, UML for real-time, Schedulability analysis, model-based development

1. Introduction

Next-generation software systems have growing demands for satisfying various distributed end-to-end timing constraints in distributed real-time system [1]. Extensive research for different types of schedulability analysis to check the feasibility of the tasks competing for resources has been conducted [2, 3, and 4]. However, it is extremely difficult to use the research results of schedulability analysis in the object-oriented design due to the differences between underlying computational model of object-oriented design and the task model used in the real-time scheduling research. The underlying computational model of the object-oriented design is event-handler based, where thread is part of multiple end-to-end computations and timing constraints, whereas in the real-time scheduling research, each thread is dedicated to specific task [5]. Even if we could manage to apply existing schedulability analysis techniques on the object-oriented models, manual implementation according to these techniques is difficult and error prone because the locations related to the scheduling aspects are scattered around the system and developer needs to manually assign priorities on individual objects as well as use proper synchronization mechanism at proper locations.

The standardized UML Real-Time Profile provides an extended set of modeling concepts for real-time software [6]. It addresses timeliness, dynamic internal structure, reactiveness and concurrency, and distribution aspects of often very complicated distributed real-time systems. Other than that, graphic notations used in the UML Real-Time Profile are strictly defined and have formal semantic so that it not only increases the understandability of the system structure and behavior, but also can be used in the system analysis and code generation process. With such model-based process, the discontinuities in the system development process can be reduced and the benefits of system modeling extend through the software development life-cycle [7].

However, some critical issues regarding distributed embedded real-time system are not well addressed by the current UML-RT model-based approaches, especially because schedulability analysis for distributed real-time systems has not been effectively incorporated. Although some researchers [5, 6] have addressed this problem by providing code synthesis of scheduling aspects and functionality aspects models, they have mainly focused on stand-alone systems. Furthermore, in the current model-based development approaches to real-time systems development [6, 12], the functional aspects and scheduling aspects of the generated code are interwoven with each other and tightly coupled with proprietary run-time libraries, which provide necessary timing, scheduling, message passing and synchronization run-time services. This dependency and tightly coupling of generated code with proprietary run-time libraries have caused poor portability, scalability, interoperability, and high maintenance cost. It is also very difficult to adapt the run-time libraries and code generation process to achieve better performance or for special domain specific needs.

Real-Time CORBA middleware provides an integrated architecture and a standard set of services that can help developers deliver end-to-end timing constraints and quality of service support for distributed embedded real-time system [8]. But without an overall development environment support, it is impractical to capture complicated real-time requirements and let the developer control and configure fine grained end-to-end properties of the system by using the services provided by Real-Time CORBA.

In this paper, we will present an approach to incorporating schedulability analysis in a UML-RT model-based development process. Using this approach, satisfaction of the distributed end-to-end timing constraints of a distributed real-time system can be verified and the schedulability analysis results will be used for aspect-oriented code generation in the model transformation and automatic code generation.

2. Our Approach

Our approach focuses on the aspect-oriented code generation for scheduling concerns based on schedulability analysis of distributed real-time systems. It enables schedulability analyses in the model-based distributed real-time system development process and using aspect-oriented program to address scheduling, quality of service and synchronization issues.

Our approach to the distributed real-time system development focuses on the following two aspects:

- Schedulability:
  Generate general scheduling reference diagrams to represent distributed end-to-end timing constraints, resources requirements and quality of services properties of objects in the UML-RT development environment.
  In addition to scheduling reference diagrams, our reference model is closely related to and tightly coupled with other functional models. Reference diagrams can be synthesized from other functional models as well as facilitate further code generation.

- Aspect-oriented code generation:
  Generate code for both the functional models in UML-RT as well as the schedulability reference diagrams.
  In this process, the code generation for schedulability will generate aspect-oriented program to address end-to-end scheduling, synchronization, distribution and quality of services aspects of a distributed real-time application. The aspect-oriented code can be further extended and customized by the developer to fine tune the code generation behavior and optimize the generated application system. The implementation will be generated through the code weaving of functional code and scheduling aspect-oriented code.

In our approach, schedulability analysis and aspect-oriented code generation aspects will be incorporated into the existing UML-RT model-based process. The new process consists of the following steps:

1. Conduct object-oriented design and development using UML-RT based on functional requirements, which generate class diagrams, structure diagrams, collaboration diagrams and state diagrams. In the meantime, timing requirements and estimated timing properties of model elements will be associated with collaboration diagrams and state diagrams.
2. Generate scheduling reference diagrams based on timing requirements and collaboration diagrams, and conduct schedulability analysis which derives priority for each action in state diagrams, synchronization mechanism and threading policies of active objects.
3. Generate code based on functional models, and threading policies from schedulability analysis results.
4. Generate scheduling aspects-oriented code based on the schedulability analysis results and insert developer's specific scheduling aspect code.
5. Weave scheduling aspect code with functional aspect code.

In Step 1, use cases and scenarios are used to capture functional requirements. Sequential diagrams elaborate on the message flows in the use case. Static class hierarchy and object relations are represented by class and structure diagrams. The dynamic behavior of objects are specified through the embedded C or C++ code that designers create to specify the actions performed during state transitions, developers can also include calls to legacy code or externally defined libraries [13, 14, 21].

Code generation based on functional models in Step 3 consists of structural and behavioral code generation [15]. Structural code generation includes the skeleton code of an application through mapping the structure models to class definitions using predefined rules. In the behavioral code generation process, translators customize a pre-coded state machine template for each state machine. This state machine implements states, transitions and communication between peer state machines. The designer's inserted source code will be invoked to perform specific actions for state transitions to achieve overall functionality [7].

The code weaving of Step 5 uses scheduling aspect point sets defined in Step 4 to identify transitions in state machines and insert scheduling aspect action code around the state machine transition code defined by the designer to implement the schedulability analysis results obtained in Step 2.

Below, we will elaborate Steps 2 and 4 in Sections 3 and 4. In Section 5, we will give a simple illustrative example. Because we use UML-RT and Rational Rose Real-Time [8] in our model-based distributed real-time system design, we will use UML-RT Rational Rose Real-Time terminology as needed.

3. Schedulability Reference Diagrams for UML-RT

In our approach, we construct scheduling reference diagrams to facilitate schedulability analysis and scheduling aspect-oriented code generation in the UML-RT based distributed real-time systems development. Our scheduling reference diagrams consist of a distributed task graph for distributed end-to-end tasks in the system and a set of subtask-resource graphs for subtasks and resources on each processor. These two types of diagrams both have graphic form and text form.

Distributed task graph is a graph $G = \{g_{1,} g_{2,} g_{3,} \ldots g_n\}$, which consists of a set of sub-graph g_i. The sub-graph can be as simple as a node for a stand-alone task. It can also be a linked list or a tree, which represents a chain of non-repetitive subtasks forming a distributed task. In its most complicated form, it can be an arbitrary graph, such as a graph with loops, which reflects some repetitions of subtasks execution. In the UML-RT based object-oriented design, each distributed task diagram maps to one specific scenario in the requirements or use cases, and forms a one-to-one mapping with collaboration diagrams. There is an extra node for each sub-graph to represent the overall properties of distributed task. Other nodes in the sub-graphs represent subtasks of the distributed task. In the UML-RT context, nodes in the distributed task diagram represent two types of action, 1) a transition of a capsule's state machine, or 2) a remote message passing through network. The edges in the distributed task graph represent precedence between subtasks in an end-to-end processing.

A local subtask-resource graph is a bipartite graph $G = \{V_{a,} V_{r,} E_d\}$. Left-hand side vertices V_a represent subtasks on a specific processor. Vertices on the right-hand side V_r represent resources on a processor. The edges E_d in the subtask-resource graph represent resource dependencies between the subtasks and resources. The subtask nodes have a one-to-one mapping to the nodes in the distributed task graph. The processor and resource types are specialized for the UML-RT. The processor types can be a physical processor or a network link. To accurately represent the run-to-completion semantics of the ROOM state machine used in UML-RT [7], each active object will need to be modeled as a semaphore type of resource. So is the prioritized network connection. Together with other resource types such as shared data and interrupt handler, they form the resource set for a particular processor.

The text form of a distributed task diagram and subtask-resource diagrams are used to store detailed timing specifications and timing properties of each node in the distributed task graph and subtask-resource graphs. They also keep information that represents mappings between nodes of schedulability task diagrams and functional models units. Each type of node in the graphic model has their counterpart in the text form model. They are defined as classes using C++ syntax. For head nodes of a distributed task diagram, the associated class will hold the end-to-end timing constraint for the task. It further contains a function pointer, which points to the function that implements a derivation algorithm to derive subtasks' timing constraint from the end-to-end timing constraints. We will provide a set of default heuristic functions so that developer can reuse existing approaches as well as define their own heuristic algorithms [4, 16].

For the remote massage passing type of task nodes in a distributed task diagram, the quality of service properties such as delay, roundtrip deadline and jitter are included in the associated classes of sub-graphs in the distributed task diagram. It also has a property to indicate whether it is a synchronous message passing or an asynchronous one. For the transition type of task nodes in the distributed task diagram, properties included are timing properties and dependent resource list. Timing properties are period, jitter, deadline, criticality, preempt type, laxity type, and laxity function. Resources information includes its location, processor, network environment, shared data and passive objects needed.

Because scheduling reference diagrams will be used for code generation process, they store connector identification or capsule and state machine action identification correspondingly to realize the one-to-one mapping between nodes of diagrams to the specific location in the functional model.

Based on the definition of our scheduling reference diagrams, steps to generate scheduling reference diagrams and conduct schedulability analysis can be summarized as below:

1. Construct distributed task diagrams and set up mappings between vertices of distributed task diagram and elements of functional models from timing annotated sequential diagrams which are generated by the functional design and association of timing specification with functional models.
2. Decide distributed scheduling heuristics, threading policy and synchronization mechanism, then derive subtask-resource diagram for individual processor based on these decisions and the distributed task diagram.
3. Conduct schedulability analysis using subtask-resource diagram of each processor and store the results of priority assignments for each subtask in subtask-resource diagrams.
4. If schedulability of the system cannot be satisfied, repeat Step 2 and change choice of heuristics for generating subtask-resource diagrams from the distributed diagram or threading and synchronization policies.

There are following advantages of using our scheduling reference diagrams: 1) The distributed end-to-end timing constraints can be systematically captured, clearly represented, and derivation of individual processor's timing constraints can be easily conducted; 2) The semantic of UML-RT can be accurately reflected in the schedulability reference diagrams; 3) Tightly coupling of our scheduling reference diagrams and other models will facilitate scheduling aspect-oriented code generation process and code weaving of functional code and scheduling aspect-oriented code.

4. Aspect-oriented Code Generation for Incorporating Scheduling Analysis Results

After generating functional code based on structure diagrams, state diagrams, and threading policies. We will generate aspects-oriented code based on the schedulability analysis results stored in the scheduling reference diagrams.

Our approach extends the existing code generation process of ROOM [7] using aspect-oriented programming and code weaving, which is shown in Figure 1. We use aspect-oriented programming principles [9, 10, 17, and 18] to address different aspects of scheduling concerns.

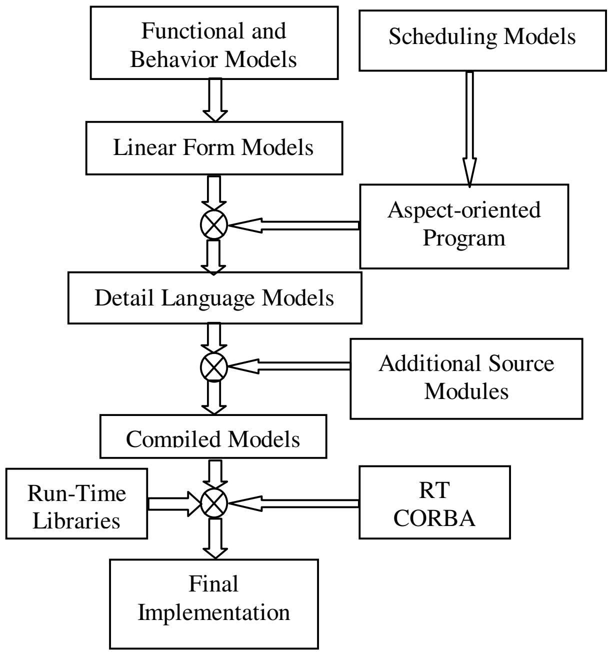

Figure 1 Code Generation Process

The scheduling aspect-oriented language has two parts: qualification statements and actions statements. The qualification statements consist of a set of predicates and operators, which are used to identify scheduling point sets in the UML-RT functional models. In UML-RT models, scheduling points are locations where developer specified action statements to influence scheduling of the system. Under the context of UML-RT semantic, scheduling points are trigger actions execution points, asynchronous and synchronous message passing points and run-time service libraries access points in UML-RT behavior models. Predicates for qualification statements to identify scheduling points are trigger(), send(), invoke() and system_call(). The operators can be used in the qualification statements are *and*, *or, not* and quantifiers *exist* and *any*.

Action statements of our scheduling aspect-oriented language are a subset of procedural language that can be used to specify actions that will be performed around the scheduling points. In the UML-RT context, action statements can be assignment of priorities to the messages, current thread priority adjustment statement as well as statements to create threads or access control objects.

Using scheduling point predicates, we can define four types of scheduling aspect point sets. Trigger processing scheduling aspect point sets are locations before or after trigger actions. Message passing scheduling aspect point sets are asynchronous or synchronous message passing points. Distributed task chain scheduling aspect point sets are points in a distributed task chain. System service access scheduling aspect point sets are places that system services such as creating and deleting of threads will be invoked.

Based on the definition of our scheduling aspect language, scheduling aspect-oriented code can be generated using distributed task diagrams and schedulability analysis results in following steps:

1. Generate qualification statements to identify trigger processing, message passing and distributed task chain scheduling aspect point sets for each subgraph in the distributed task diagram. Because nodes in our scheduling reference diagrams have one-to-one mapping with elements of functional models, qualification statements to identify these scheduling aspect point sets can be automatically generated.
2. Generate scheduling aspect action statements at trigger scheduling aspect point sets to adjust priority of execution thread according to the message priority, generate assignment statements to adjust message priority before message passing scheduling points, and insert synchronization code at the distributed task chain scheduling aspect point sets if the heuristic defined in the header node of the distributed task diagram is predefined.
3. Generate qualification statements to identify system service access scheduling aspect point sets according to threading policy and resources information in the subtask-resources diagrams. Generate appropriate code to reflect threading policies.
4. Insert developer specific code for the user defined heuristic algorithms for distributed scheduling and synchronization.

By using aspect-oriented code generation based on schedulability reference diagrams, our code generation process incorporates the schedulability analysis results in the code generation process and has following advantages:

1. Generated code can satisfy both the functional requirements as well as various timing constraints, which has been centralized represented as scheduling reference diagrams.
2. Code generation process for the scheduling aspect is flexible so that the non-deterministic nature of the distributed schedulability analysis is addressed by allowing designer to apply different distributed schedulability analysis heuristics.
3. Generation of scheduling aspect code is separated with the generation of functional code so that it can work with different implementation strategies of functional code generation.
4. Separation of scheduling aspect-oriented code generation with functional code generation reduces intertwining of functional code with scheduling aspect code which avoids over complicated code and reduces maintenance cost.

Finally, after we have generated scheduling aspect-oriented code to address scheduling related issues and insert developer specific scheduling code, a code weaver will combine aspect-oriented code with functional code. It inserts scheduling aspect action code into the proper position in the functional model generated code based on scheduling aspect point sets. It can even work against the functional model directly so that different implementation strategies can all benefit from scheduling aspect-oriented code.

5. An Example

5.1 Requirements

Below we will show an example that consists of a group of robots which conduct specific tasking according to the commands from a central control system. The system also includes a real-time database to keep track of information obtained from each robot.

Each robot has two processors, command processing and control processor P1 and I/O processor P2. We will focus on the analyzing of following two distributed end-to-end tasks:

Robot control task, which consists of three subtasks on two processors, it is a periodic task with a period of 100ms and a deadline of 200ms

- Rt1 is strictly periodic task to take input from a motor sensor on the I/O processor P2 then send it to the control processor P1
- Rt2 is a command processing task running on the control processor P1 based on new command and parameters received from the central controller, then send detailed instructions to the I/O processor P2
- Rt3 is motor control also running on the I/O processor P2, it sends out signal for actuator based on output from Rt2

Measurement task, which consists of three subtasks on three processors, it is a periodic task with a period of 200ms and a deadline of 200ms

- Mt1 is strictly periodic task that reads sensor and preprocesses it on the I/O processor P2, it then sends it to the command processing processor P1
- Mt2 is some more complicated digital signal processing on the control processor P1 and send processed data to the real-time database system
- RDt2 is the real-time DB update task, after the real-time database system receive updated tracking information from a robot, it updates its record and send display updates to the central control system

5.2 Object-Oriented Design

Based on the functional requirements, timing requirements and estimated timing properties, we can conduct object-oriented design and construct class diagrams, structure diagrams, sequential diagrams and state diagrams. Figure 2 shows the sequential diagram of the first distributed end-to-end task in the robot control system.

5.3 Construct scheduling diagrams and conduct schedulability analysis

Using the sequential diagrams and associated timing specification, we can construct distributed end-to-end tasks diagrams which are shown below in Figure 3.

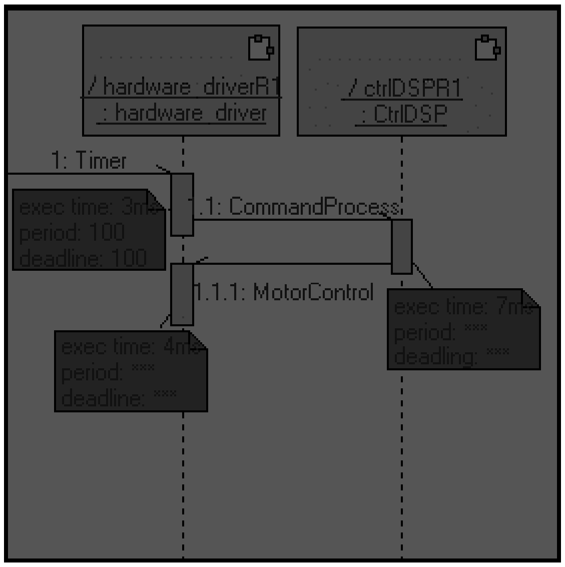

Figure 2 Sequential Diagram

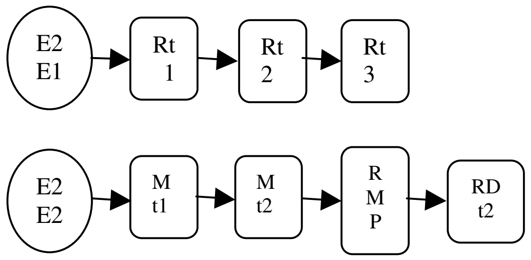

Figure 3 End-to-end task diagrams

Here we can see a remote message passing from the robot to the real-time database is represented as a subtask that is not originally in the requirement specification. This task will be scheduled on the network resource which can be modeled as a processor.

Each nodes in a distributed end-to-end task graph represents a transaction in a specific state machine, this one-to-one mapping relation will be stored in the text form of the graph.

The derivation of individual nodes' task diagram is generally a NP hard problem and we need to choose different heuristic algorithms. Based on the algorithms we chosen, we can construct the individual processor's task-resource dependency diagrams and use it to conduct mature schedulability analysis such as rate monotonic analysis. Results of the schedulability analysis will be stored in a table which is show in Figure 4.

action	Heuristic	Thread	E/D	Priority
Rt1	Proportion	Single	3/100	200
Mt1	Effective	Single	10/200	100

Figure 4 Schedulability Analysis Result Table

5.4 Code generation

Functional based code generation will generation state machine skeleton code. A simplified robot I/O control capsule's state machine skeleton code is shown in Figure 5

```
    Capsule: I_OR1
    State: Command_Process{
        entry action:
              { (entryCommand_Process());}
        exit action:
              { exitCommandProcess());}
        transitions:{
        {transition command_input:
           triggered by: {signal:timeout(100)}
        action: {transition1()}
     }
```

Figure 5 Skeleton Code of Robot Control Capsule State Machine

Using our aspect-oriented language syntax, scheduling point sets can be specified as below in Figure 6:

```
Pointset: adjust_points = {
        Trigger( Rt1) || Trigger( Rt2) || Trigger
        (Rt3);
        Trigger( Mt1) || Trigger( Mt2) || Trigger
        (RDt2);
        }

Pointset: assign_points = {
        Send(Mt2); || Invoke(Rt2);
        …
        };
…
```

Figure 6 scheduling point sets specification

Then based on the schedulability analysis results, following priority settings statements can be generated as shown in Figure 7

```
Aspect : adjust_thread_priority {
        At: adjust_points
        Action:
        System.thread.setPriority(signal.priority);
        }
Aspect : set_signal_priority {
        At: assign_points
        Action:
        Signal.priority = x;
        }
        …
```

Figure 7 scheduling aspect action specifications

5.5 Code weaving

Finally based on the predicates of scheduling aspect point sets, we can identify scheduling points and insert scheduling aspect action code around scheduling points. For the state transition code shown in Figure 5, the weaved code is shown in Figure 8.

```
Capsule: hardware_driverR1
State: Wait_Command{
        var: old_priority;
        entry action:{
        old_priority
           = system.Thread.getPriority();
        system.Thread.adjust(signal.priority);
        (entryWait_Command());}
        exit action:{
        exitWait_command());
        system.Thread.adjustPriority(old_priority);}
        …}
```

Figure 8 Weaved Code

6. Discussion

In this paper, we have presented a model-based development approach to address the schedulability in a distributed real-time system. By incorporation of scheduling reference diagrams into the existing models, and conduct schedulability analysis, the scheduling aspects of a distributed system is accurately captured and analyzed. Through the further scheduling aspect-oriented code generation and code weaving of different aspects, the schedulability analysis results are accurately realized in the implementation.

Future research in the area needs to be done includes extending the aspect-oriented program by meta-object protocols to optimize code generation process as well as adapt run-time libraries for different domains [19, 22]. Adaptation of run-time libraries should also be explored as link time static adaptation or run-time adaptation. Because of the tightly coupling of generated code and the run-time libraries, enabling its customization will further improve the quality of the implementation and extend the usability of model-based development approach to more resource restricted domains [20].

References

[1] C. D. Gill, F. Kuhns, D Levine, D. C. Schmidt, etc., “Applying Adaptive Real-time Middleware to Address Grand Challenges of COTS-based Mission-Critical Real-Time Systems” *Proceedings of the 1st International Workshop on Real-Time Mission-Critical Systems: Grand Challenge Problems*, IEEE, Phoenix, Arizona, November 30, 1999.

[2] Liu, C.L. and Layland J.W. “Scheduling Algorithms for Multiprogramming in a Hard Real Time Environment”. JACM 20(1) (1973), 46-61

[3] M. H. Klein, T. Ralya, B. Pollak, R. Obenza, *Practitioner's handbook for real-time analysis: guide to rate monotonic analysis for real-time systems*, Kluwer Academic Pub., Boston, MA, 1993

[4] J. W. S. Liu., *Real-Time Systems*, Prentice Hall, Upper Saddle River, NJ, 2000

[5] M. Saksena, P. karvelas, Y. Wang, “Designing for schedulability: integrating schedulability analysis with object-oriented design”, *Proceedings. Third IEEE International Symposium on Object-Oriented Real-time Distributed Computing*, 2000

[6] M. Saksena, P. Karvelas, “Automatic synthesis of multi-tasking implementations from real-time object-oriented models”, Euromicro RTS 2000. 12th Euromicro Conference on Real-Time Systems, 2000 Page(s): 101 –108

[7] B. Selic, G. Gullekson, P. T. Ward. *Real-time object-oriented modeling*, Wiley & Sons, New York, c1994.

[8] Object Management Group, *the Common Object Request Broker: Architecture and Specification*, 2.4 ed., Oct. 2000.

[9] T. Elrad, R. E. Filman, A. Bader., "Aspect-Oriented Programming: Introduction", Communication of ACM Volume 44, Issue 10 (October 2001)

[10] C. Zimmermann, ed. *Advances in Object-Oriented Metalevel Architectures and Reflection,* CRC Press, Boca Raton, Fla., c1996.

[11] A. B. Arulanthu, C. O’Ryan, D. C. Schmidt, and M. Kircher, “Applying C++, Patterns, and Components to Develop and IDL compiler for CORBA AMI Callbacks”, C++ Report, vol. 12 Mar 2000.

[12] P. Martins. “Integrating Real-Time UML Models with Schedulability Analysis”, http://www.tripac.com/html/whitepapers

[13] J. Rumbaugh, I. Jacobson, G. Booch. *The Unified Modeling Language User Guide*, Addison Wesley, 1999

[14] J. Rumbaugh, I. Jacobson, G. Booch, *The Unified Modeling Language Reference Manual*, Addison Wesley, 1999

[15] R. Bell, “Code Generation from Object Models”, Embedded Systems Programming, March, 1998, http://www.embedded.com/98/9803fe3.htm

[16] J. Sun, and J. W. S. Liu, “End-to-end synchronization protocols of fixed priority periodic tasks,” Proceedings of the 16th IEEE International Conference on Distributed Computing Systems, Hong Kong, June 1996

[17] R. E. Filman and D.P. Friedman, “Aspect-Oriented Programming is Quantification and Obliviousness”, Workshop on Advanced Separation of Concerns, OOPSLA 2000, October 2000, Minneapolis.

[18] R. Filman, “Applying Aspect-Oriented Programming to Intelligent Synthesis.” Workshop on Aspects and Dimensions of Concern at ECOOP’2000, Cannes, France, June 2000.

[19] D.C. Schmidt, D.L. Levine, and S. Mungee, “The design and Performance of a Real-Time Object Request Brokers,” Computer Communications, vol. 21, pp. 294-324, Apr. 1998

[20] S. Shlaer, S. J. Mellor, *Object lifecycles : modeling the world in states*, Engliewood Cliffs, NJ : Yourdon Press, c1992

[21] G. Gullekson, “Design for Concurrency and Distribution with Rational Rose RealTime”, http://www.rational.com/

[22] S. S. Yau, and F. Karim, "Component Customization for Object-Oriented Distributed Real-time Software Development", *Proceedings of 3rd IEEE International Symposium on Object-oriented Real-time Distributed Computing*, March 15-17,2000, pp 156-163