Analysis of Concurrent and Real-Time Systems

W. Richards Adrion
Computer and Information Science Department
University of Massachusetts

COINS Technical Report 90-87

ANALYSIS OF CONCURRENT AND REAL-TIME SYSTEMS

W. Richards Adrion
Department of Computer and Information Science
University of Massachusetts, Amherst
Amherst, MA 01003
adrion@cs.umass.edu

1 INTRODUCTION

Computers are becoming an essential embedded system component in everything from life critical applications, such as medical devices and nuclear power plants, to the everyday appliance. These applications are naturally "real-time" and amenable to concurrent programming solutions. Dealing with concurrency and real-time constraints pose many interesting and challenging problems. Since it is much harder for developers to reason about concurrent behavior and real-time constraints than unconstrained sequential behavior, it is likely that more errors will be introduced into these systems. Furthermore, because of the added complexity and difficulties with reproducing results and simulating realistic scenarios, it is important that powerful testing techniques be developed to evaluate the reliability of such systems.

Based on our experience with developing testing techniques for sequential systems, it seems most likely that a variety of techniques will need to be developed and integrated together. We believe that both static analysis techniques, which evaluate systems independent of their execution, and dynamic techniques, which require the execution of the system on test cases, will be necessary to achieve the desired confidence levels. With static analysis, errors and anomalous conditions can be detected directly for certain classes of errors. Dynamic analysis can be used to develop test cases or scenarios based on error revealing criteria or to evaluate the selected test cases or scenarios to see how well they satisfy those criteria.

In studying and developing testing techniques for real-time systems, we first investigated concurrent systems and now we have begun to examine whatever additional concurrency and complexity may result from the real-time constraints. Our work on the testing of concurrent systems is an extension and improvement upon the work of Taylor[Tayl83a, Tayl83b]. Using a reduced control flow graph representation of each task in a system, Taylor defines a *concurrency graph* that models the behavior of the total system. Concurrency graphs capture all the possible states of a concurrent system and have been used along with specialized dataflow techniques to discover sequences of suspicious events [Tayl80]. Unfortunately concurrency graphs are analytically intractable, i.e., in the worst case, where each node of a task's control flow graph can interact with any node of any other task in the system, the total number of states in the concurrency graph is the product of the number of nodes in the graph for each task.

We have been developing a model of interacting tasks that appears to reduce considerably the number of states in the concurrency graph representation. We call our representation

a Task Interaction Concurrency Graph, since it is derived from a Task Interaction Graph instead of from a control flow representation. In all cases we have examined, Task Interaction Concurrency Graphs have proved to be a substantial reduction over typical concurrency graph representations and, as a result, will have a major impact on the kinds of analysis that can be applied and on the kinds of programs that can be analyzed. Moreover, this reduction comes with no loss of information.

In a real-time environment, a sizable number of additional tasks will be needed to represent the run-time system, the scheduler, the interrupt handler, various remote processes, and interaction with the external environment. The complexity of the graph representations would increase accordingly. In addition, the mechanisms to allow this interaction, such as added task activations and increased use of global variables, will increase both the complexity of the representation and the potential for undesirable performance.

On the other hand, real-time constraints will provide some potential for optimizing the representations. Take the simple example of the dining philosophers, the race condition between any two philosophers has two bad outcomes: deadlock and starvation (livelock). Deadlock has a number of solutions involving both changes to the implementation and external constraints on access to shared resources. A scheduler might be employed to reduce the chances for starvation. Assume a simple algorithm that queues up requests for a fork in such a way that the philosopher that has least recently had access to a fork is given priority (least-recently serviced scheduling). Certain task interactions would no longer be possible thus simplifying the representation Unfortunately, scheduling algorithms will not always lead to reductions.

Besides introducing additional complexity, the real-time constraints will make it likely that certain tasks will be interrupted before completion. In some cases these tasks will never be resumed. If these are iterative tasks, for example involving numerical optimization or refinement, then we must consider intermediate levels of performance. In the dining philosophers problems, one could introduce the notion of "hungry" as a less desirable but adequate performance criteria. A philosopher could be interrupted before finishing dining, thus not starving, but not full either.

2 CONCURRENCY ANALYSIS TECHNIQUES

There are several graph models for representing concurrent behavior. Among these are Petri Nets[Mura83], control flow graphs, concurrency graphs[Tayl83a], Task Interaction Concurrency Graphs[Long89a], and Constrained Expressions[Avru85, Avru86, Dill88]. Petri nets have been used widely for performance analysis as well as for static analysis. All of these representations are basically equivalent, differing primarily in their expressiveness for particular domains of analysis and in their complexity.

Our interests in graph representations are as a tool to model concurrency which can be used to guide the application of a variety of static and dynamic testing techniques.

2.1 Task Interaction Graphs and Task Interaction Concurrency Graphs

Task Interaction Graphs (TIG's) and Task Interaction Concurrency Graphs (TICG's) are structures used in an analysis method developed by Clarke and Long[Long89a] for concurrent programs. The method is an extension of work by Taylor[Tayl83a, Tayl83b]. In both the

Clarke/Long and Taylor, et al techniques, graphical representations of concurrent behavior are developed as an aid in partitioning the verification and testing problem into domains, for example, separating concurrency issues from sequential behavior, which can be analyzed independently. Another primary motivation to both of these coordinated activities is to develop "inexpensive" methods to detect portions of the code in which various behaviors were possible, e.g., potential dangerous parallel execution, that could then be used to guide analysis employing more costly techniques, e.g., global data flow analysis.

The details of the Clarke/Long method will be omitted here, but we will provide some insight by giving an example of the method applied to the dining philosopher problem. What is interesting about dining philosophers in this context is that it appears to be nearly a worse-case scenario for any concurrency analysis technique. All of these techniques, including Petri Nets[Mura89, Shat88] and Constrained Expressions[Avru85, Avru86, Dill88], build a representation of concurrency by embedding representations of tasks at the point of invocation. Such representations are naturally exponential in the number of tasks. All methods attempt to reduce the potential complexity of representation by constraining the problem, by exploiting regularity and/or by hiding details. The dining philosophers problem results in solutions employing multiple processes contending for shared resources whose structure prevents significant reductions in in the complexity of the representation.

In Figure 1, the TIG's for the two dining philosopher problem are given. TIG's are developed using a rendezvous model of concurrency[Burn85] and hide all sequential behavior by partioning tasks into possibly overlapping regions not affecting task interactions. The explicit psuedocode for the regions are expressed as annotations to individual nodes. Edges in TIG's represent specific task interactions and indicate a transition from region to another. For example, the TIG's representing the philosopher tasks show the entries to the associated fork tasks as edges, where F1.U is the start or end (depending on the subscript) of an entry at point U in fork task F1. Multiple edges exiting a node represent decision points in the code, e.g., select, loop or if-then-else statements, which control task interactions. Control statements in the actual tasks which do not affect task interactions are hidden in the node annotations. Entries and accepts are modeled as two task interactions (TIG edges) that divide the task into three regions. The two interactions represent the beginning and end of an accept or entry, while the intermediate node represents the region where parameters may be passed between the calling and accepting task. Under certain circumstances these two edges and intermediate node can be collapsed into a single edge.

Task Interaction Concurrency Graphs are derived from TIG's. Each node of a TICG is a k-tuple over the combined nodes of all k TIG's. An edge in a TICG represents the start or end of a rendezvous between exactly two tasks. For example, in Figure 2, the transition $(1,1,1,1)$ to $(3,1,3,1)$ represents Philosopher 1 beginning to pick up Fork 1(his right fork). Clarke and Long present various techniques[Long89a, Long89b] to reduce the TICG. Actually, Figure 2 is a reduced TICG obtained by compacting the TIG's initially and then pruning states from the TICG. Note that $(3,3,3,3)$ represents a deadlock condition in this TICG.

In summary, the TICG and TIG models:

- have been designed to capture the rendezvous-like synchronization found in languages like Ada[1] [Ada83], Distributed Processes [Brin78], and CSP [Hoar85].
- reduce the complexity of representing concurrent behavior when compared to other

[1] Ada is a registered trademark of the DoD/AJPO.

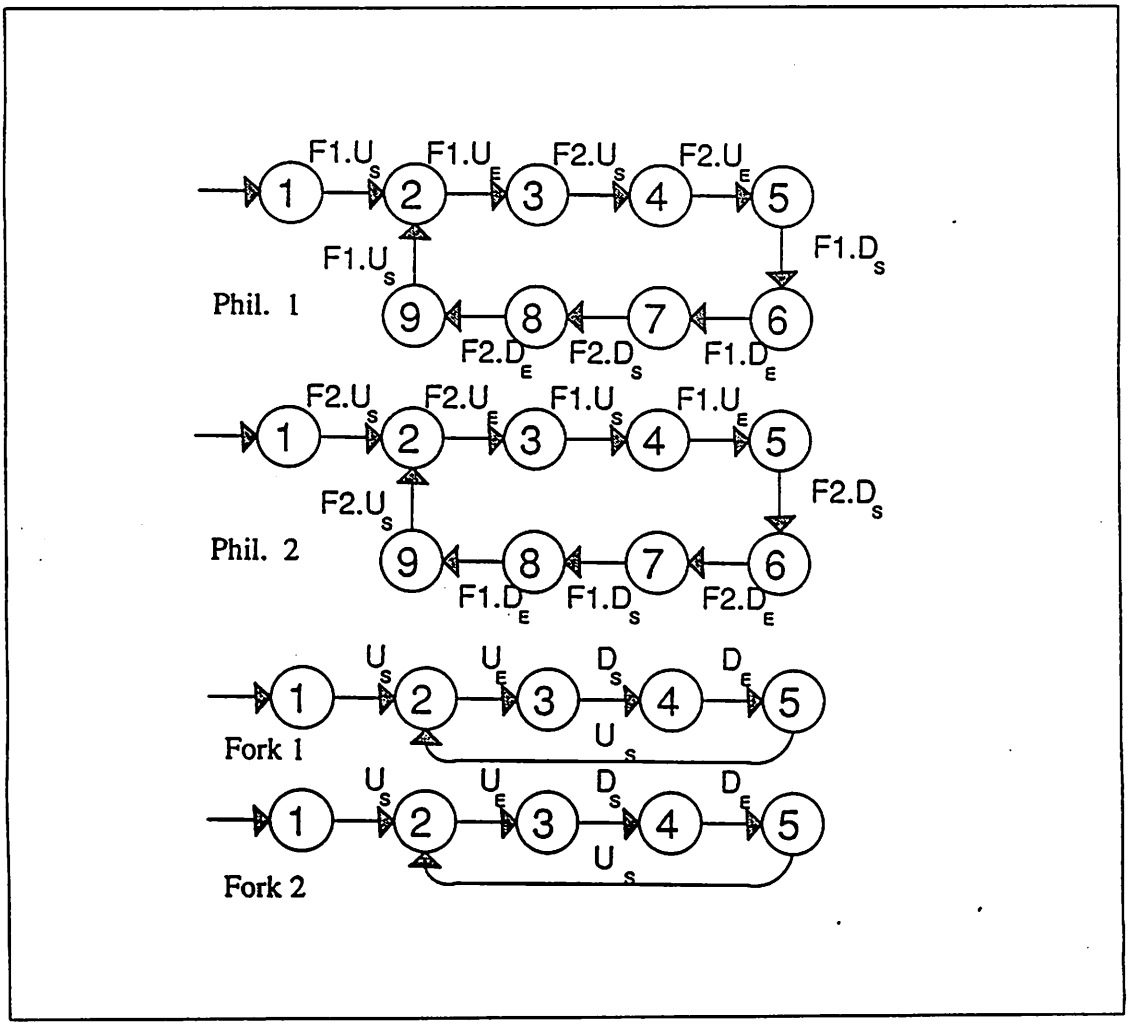

Figure 1: TIG*'s for the Two Dining Philosopher Problem*

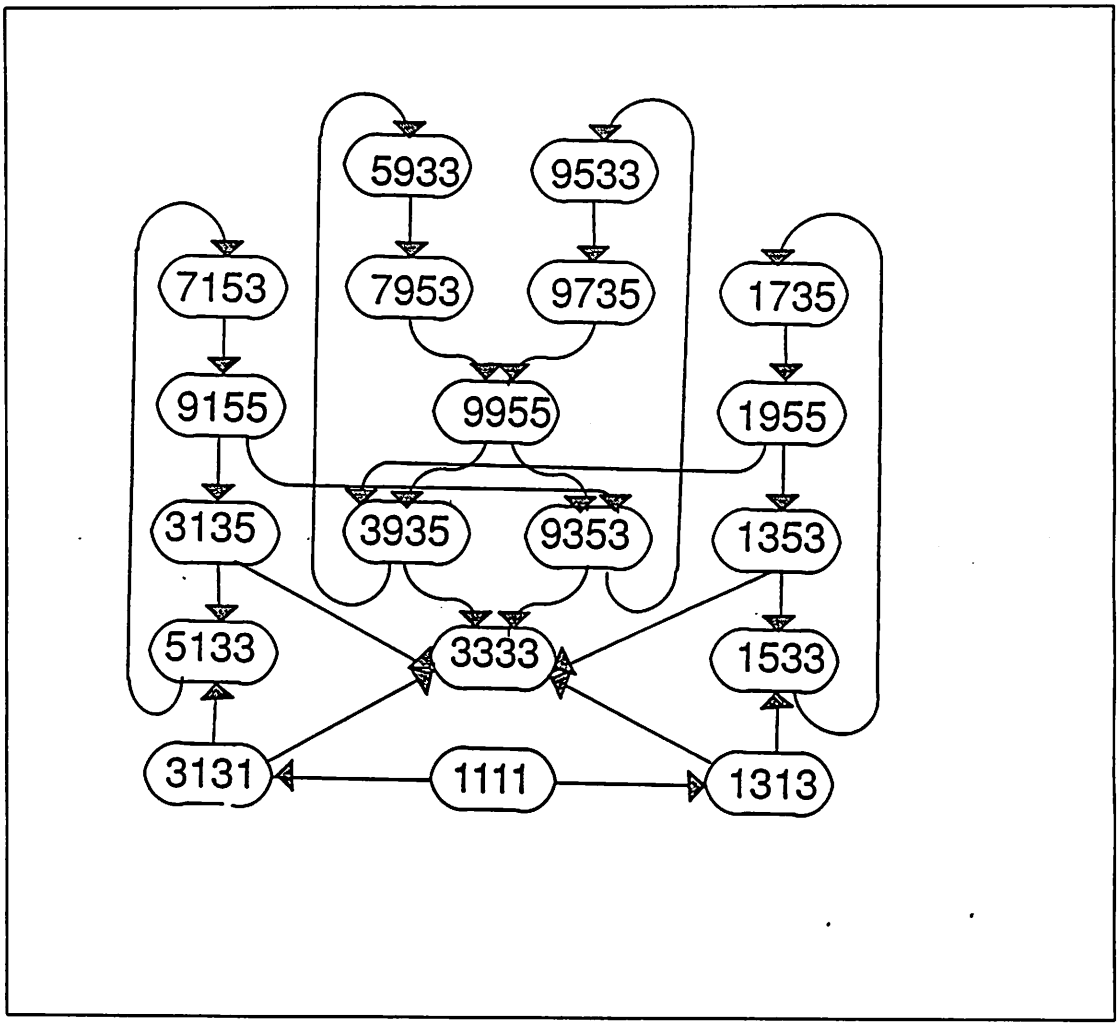

Figure 2: *Reduced* TICG *for the Two Dining Philosopher Problem*

such techniques[Tayl83a, Shat88],

- lend themselves to partitioning and parceling to reduce analysis
- can be constrained through language restrictions which can in turn be enforced (at a cost),
- and can include certain simple kinds of analysis as part of their construction, for example, detection deadlock and "dangerous" parallelism.

The TIG and TICG models are clearly exponential and we are carrying out a mathematical analysis of the expected complexity of typical applications. Experience to date with actual applications shows that we are always inside the elbow of the exponential curve. For example, analysis begins to be very difficult only for applications where the concurrency is greater than that exhibited by the six dining philosophers problem. Since such behavior is nearly worse case, we do not expect to see many difficulties in practice. Clarke, Long and Fialli have suggested some reasonable programming practices[Long89b] that result in significant reductions in the TICG complexity.

2.2 Relationship to the Arcadia Project

The work of Clarke and Long, as well as the work described herein, is a part of the analysis tools and techniques being developed under the Arcadia Project[Tayl86, Tayl87]. As part of Arcadia, we are developing a suite of tools, called CATS, the Concurrency Analysis Tool Suite. Figure 3 shows the structure of CATS and its relationship to other Arcadia tools in the Arcadia-1 prototype development environment. In CATS, Ada programs are complied into an intermediate representation, IRIS. IRIS represents programs as attributed graphs. TIG's can be derived directly from the IRIS representation or from control flow graphs. The TICG's are then derived from the TIG's.

Figure 3 shows how CATS will be used in conjunction with various data flow and temporal logic assertion techniques[Youn88]. We are planning to use the TIG/TICG representation in conjunction with a variety of testing techniques[Clar82, Clar85, Rich86a, Rich86b].

2.3 Real-Time Aspects

The introduction of real-time constraints in applications amendable to concurrent programming solutions adds considerable complexity to the analysis problem. We take as a definition for real-time, the following[Stan88]:

> *Hard real-time systems* are defined as those systems in which the correctness of the system depends not only on the logical result of the computation, but also on the time at which the results are produced.

The TIG and TICG models have no inherent notion of time. Where they are useful is in identifying regions of tasks and task interactions which may cause certain real-time specifications to fail. One approach is to identify these and apply other analysis techniques. Young in his thesis[Youn88] develops techniques for creating and then inserting temporal logical assertions in tasks to aid dynamic analysis. We will not discuss the actual analysis and testing of real-time systems here, but will concentrate on what modifications to the TIG

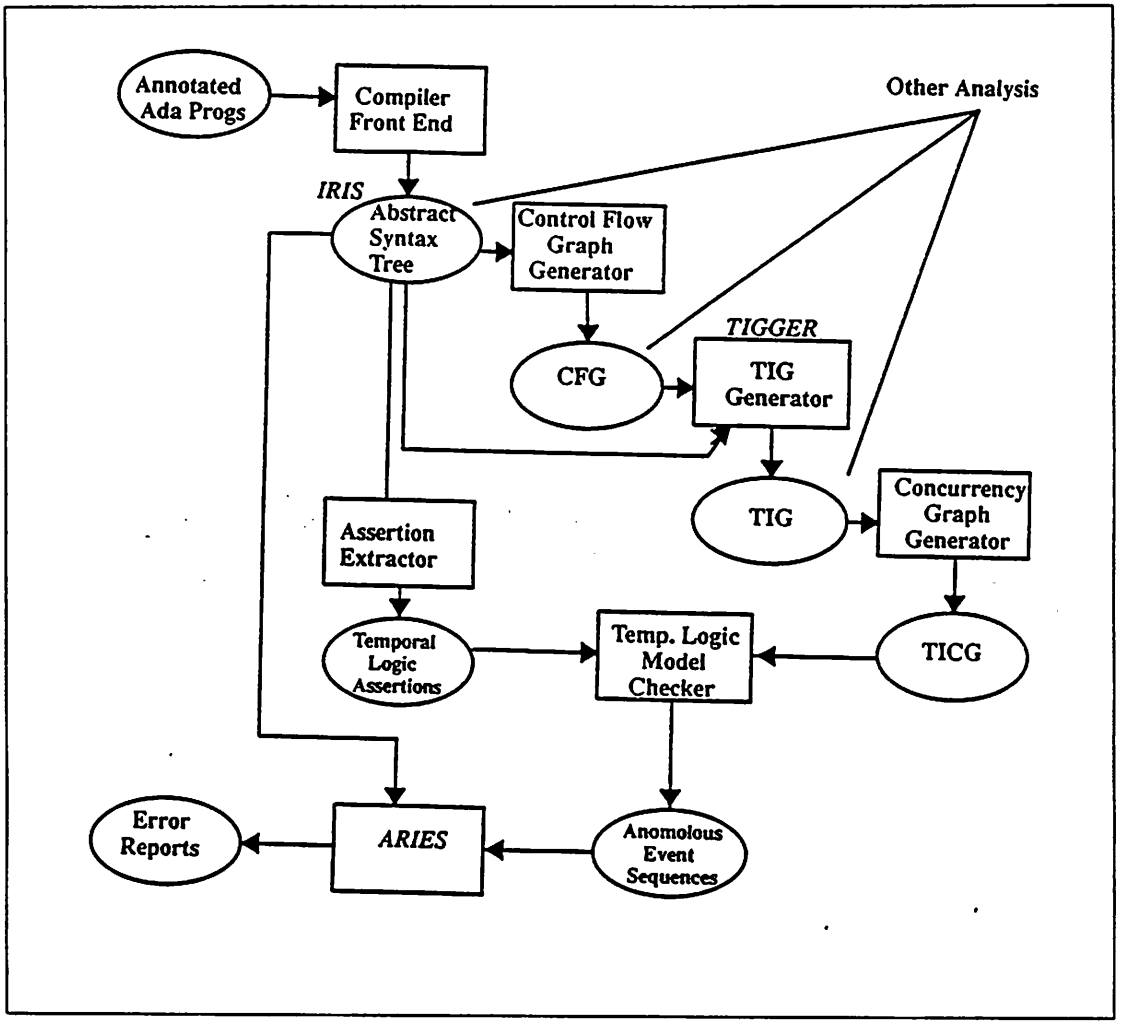

Figure 3: *CATS: The Concurrency Analysis Tool Suite*

and TICG models result from applying them to real-time applications. In this paper, we will concentrate on three central issues: the representation of the run-time behavior of a multi-tasking application, the impact of scheduling regimes, and the constraints imposed by various real-time designs.

3 APPLICATION TO REAL-TIME SYSTEMS

We shall use as an example, the various modifications to the dining philosophers problem introduced in [Stan88] called the dying, dining philosophers. We will consider three variations suggested by [Stan88]: the introduction of a start and end time by which each philosopher must begin and complete a meal; resources which cannot be relinquished at any time during a meal (in contrast with forks which may be put down and later picked up); and the possibility that philosophers will arrive and depart the table. The basic dying, dining philosophers problem differs from the usual problem in that each philosopher must begin eating within a certain time period or starve. In addition, a time to complete a meal is given for each philosopher. Each of the variants can be described as follows:

- **DDP1**: Associated with each philosopher is a start time by which the philosopher must begin eating or starve. In addition, each philosopher has a hard deadline by which the philosopher must leave for a lecture. If a philosopher has not completed a meal(held the fork resources for the specified meal period), the philosopher can be considered to have starved or, as a weaker condition, to have left hungry.
- **DDP2**: In addition to the meal at each philosopher's place, there are non sharable resources, e.g., chairs or condiments, which are necessary for dining, but which cannot be released for the duration of an individual's meal. If there are fewer than n chairs, some philosophers may not sit down until others have completed their meals.
- **DDP3**: As philosophers leave the table (through starvation or to go to their lectures) they are replaced (some times immediately and sometimes not) by another philosopher.

We shall consider the effect of possible solutions, both schedules and other constraints, on the TIG and TICG models.

3.1 Techniques for Managing Real-Time Constraints

The DDP1 problem can be constrained in two obvious ways: control the number of philosophers at the table and control how each philosopher may request resources. In the first case, we could restrict the number of philosophers so that at least one chair was empty (DDP1e), thus guaranteeing that one philosopher will access two forks and deadlock will be avoided. Alternatively we could require a philosopher to request forks only if both are available (DDP1b) or we could require that odd philosophers pick their left fork first (DDP1o) and even philosophers choose their right fork first. Because each of these solutions to the deadlock problem restrict the possible task interactions, the resulting task interaction graphs are much reduced. Examples of the the two philosopher TICG's for these three solutions are given in Figure 4. Note that the TICG's for DDP1b and DDP1o are identical as one would expect. The TICG for DDP1e is much less complex because it places a fixed ordering on

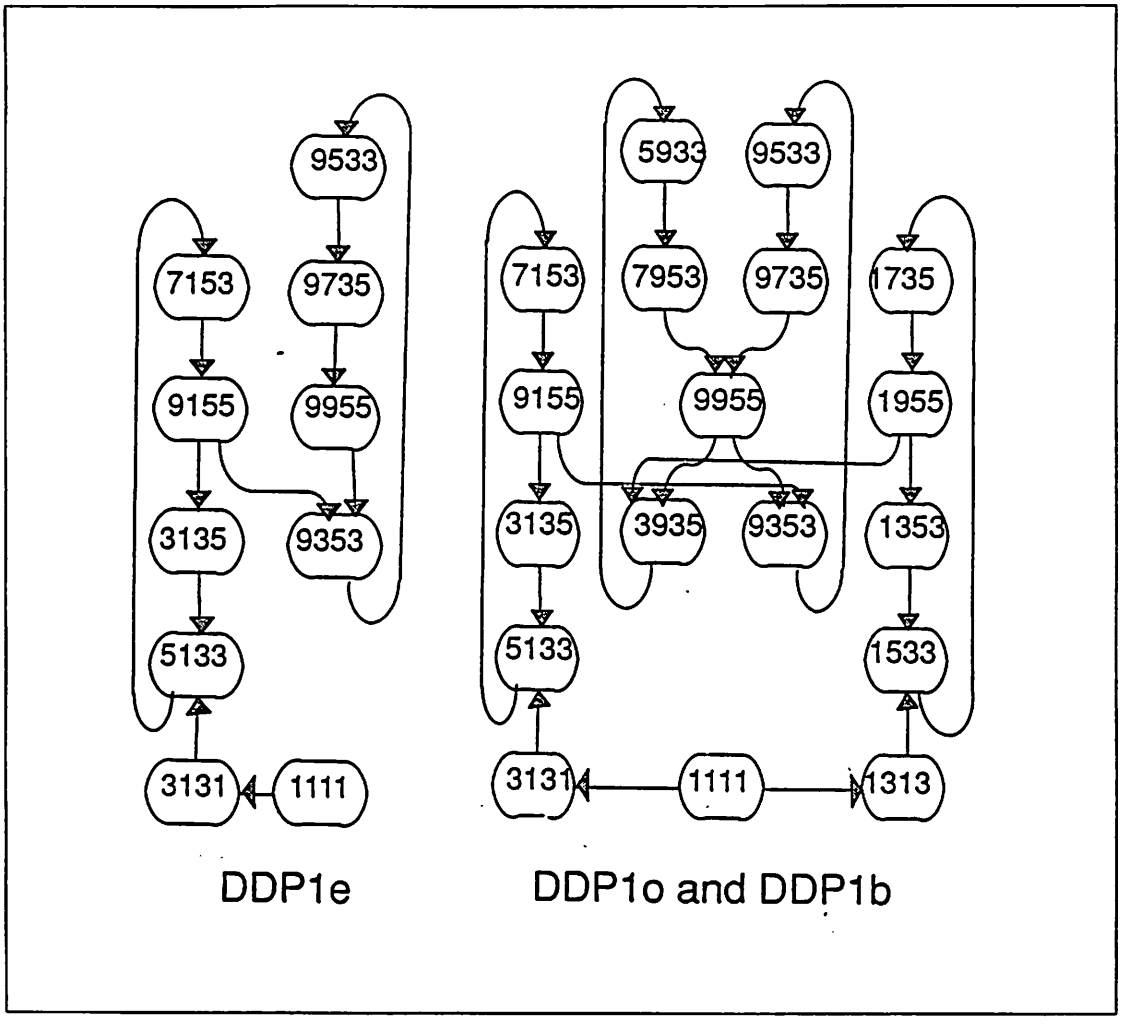

Figure 4: TICG*'s for the three DDP2 Solutions*

the execution of tasks. In both cases, how the constraints are imposed are hidden, but in practice they may need to be made explicit.

While each constraint avoids deadlock, none deal with the starvation problem. Starvation is a problem even without start/completion deadlines. With deadlines in each solution, a philosopher may wait to start eating until after the required start time and there is no guarantee that a philosopher will complete a meal before the deadline.

Without the additional constraint of start and completion times, starvation can be managed by scheduling the philosophers. Let us look at two regimes: round-robin and least recently serviced. In the former, the TIG's and TICG will grow in complexity to represent the imposed ordering. In both, additional states will have to be added to represent interaction with the scheduler. When start and completion deadlines are imposed, a *feasible* schedule will have be to determined either statically, by analyzing the order of events and absolute time, or heuristically by attempting to schedule the tasks dynamically. In the first case, $n!$ possible orderings will have to be analyzed, but the overhead in the TIG and TICG

models will not be great. In the latter, time will have to be managed directly in the implementation adding significant additional task interactions and additional tasks. Obvious implementations of dynamic scheduling involve interrupting an executing task. Interrupts are not currently representable in the TICG model since interrupts are difficult to represent as rendezvous.

In the case of DDP2, the addition of non-sharable resources complicates the solution, but adds no additional complexity to the TICG model beyond that necessary to represent the more complex solutions and scheduling regimes. It is obvious that, with less than n chairs, the solution is identical to restricting the number of philosophers which may be at the table.

For the DDP3 scenario, the additional difficulty is in dealing with dynamic arrivals and departures of philosophers. While it complicates the scheduling somewhat, it is not really much different from the original problem from a scheduling point of view. The TIG and TICG models, however, can model only predetermined task activity. Analysis would have to be partitioned into the original case and separate additional tasks for monitoring arrivals and departures.

In summary, strategies which would normally be employed to avoid deadlocks will reduce the number of possible concurrent states, and thus reduce the complexity of the TICG. On the other hand, techniques to avoid livelock and to insure deadlines are met will increase the complexity of and the number of TIG's and thus the complexity of the TICG. Figure 5 shows a TICG for the DDP3 two philosophers problem where deadlock is avoided by requiring both forks be available to a requesting task. Scheduling is based on a static heuristic. Both constraints are implemented by additional tasks bringing the total to five interacting tasks in this implementation.

3.2 Scheduling

An alternative strategy to reprogramming the application to manage the real-time constraints is to embed the real-time management strategies in the operating and run-time systems. We are currently looking at means of augmenting the intermediate nodes in entry and accept subgraphs of the TIG's to represent operating and run-time system interactions with the application tasks. Clocks will clearly be managed by the run-time system and TIG structures will have to be developed to represent this interaction. In addition, scheduling regimes, interrupts, accept/entry queue management and other systems calls will have to be represented.

We also looking carefully at how we might use the TICG representation as a design tool for developing scheduling regimes and deadlock prevention mechanisms. For example, in Figure 4 the DDP1 solutions could have been derived from the original TICG and compared for ease of analysis.

4 CONCLUSIONS AND FUTURE RESEARCH

We have shown how the TIG and TICG models can be used to analyze real-time applications. We have also identified a number of areas in which modifications and augmentations to the model will be necessary to represent real-time behavior. In summary, future directions include:

- representing time-dependencies and interrupts in the TIG/TICG and rendezvous models,

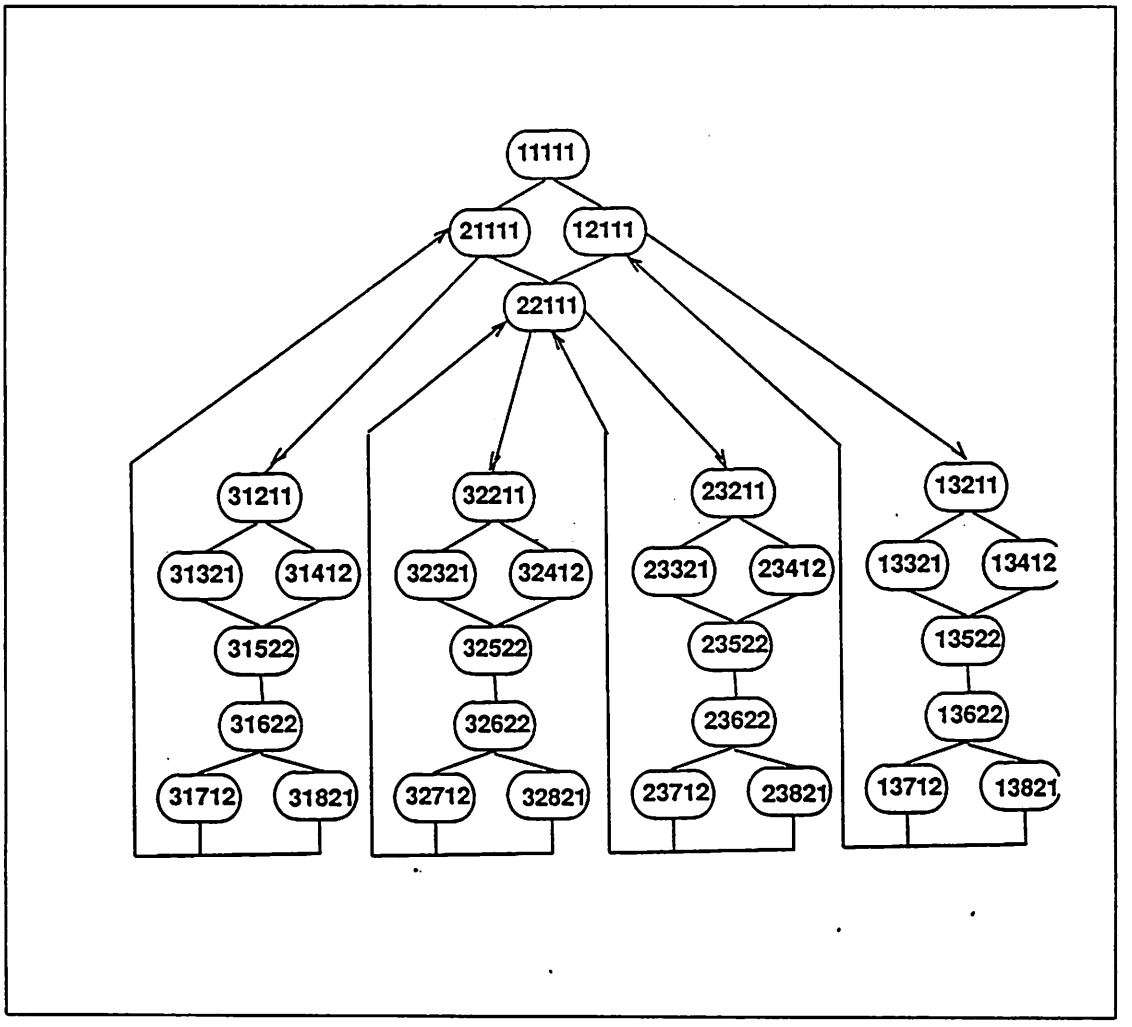

Figure 5: TICG *for the DDP3 Problem*

- representing operating and run-time system interactions,
- employing TICG's as a design tool for real-time scheduling and deadlock/livelock prevention
- analyzing the probable (expected) complexity of TIG and TICG models and comparing with the known asymptotic complexity, and
- studying the tradeoff between TIG and TICG complexity and various scheduling regimes and real-time constraints.

References

[Ada83] **Reference Manual for the Ada Programming Language (ANSI/MIL–STD–1815A)**, United States Department of Defense, Washington, D.C., January 1983.

[Adri89b] W. Adrion, "Testing of Concurrent and Real-Time Systems", *Proceedings of the ONR Workshop on Testing, Analysis and Verification*, San Diego, August 1989.

[Avru85] Avrunin, G.S. and Wileden, J.C., *Describing and Analyzing Distributed Software System Designs*, **ACM Transactions on Programming Languages and Systems**, Vol.7, No.3, July 1985, pp.380–403.

[Avru86] Avrunin, G.S., Dillon, L.K., Wileden, J.C., and Riddle, W.E., *Constrained Expressions: Adding Analysis Capabilities to Design Methods for Concurrent Software Systems*, **IEEE Transactions on Software Engineering**, February 1986, pp.278–292.

[Brin78] Brinch Hanson, Per, *Distributed Processes: A Concurrent Programming Concept*, **Communications of the ACM**, Vol.21, No.11, November 1978, pp.934–941.

[Burn85] Burns, A., **Concurrent Programming in Ada** Cambridge University Press, New York, 1985.

[Clar82] Clarke, L.A., Hassell, J., and Richardson, D.J., *A Close Look at Domain Testing*, **IEEE Transactions on Software Engineering**, Vol.SE-8, No.4, July 1982, pp.380–390.

[Clar85] Clarke, L.A., Podgurski, H.A., Richardson, D.J., and Zeil, S.J., *A Comparison of Data Flow Path Selection Criteria*, **Eighth International Conference on Software Engineering**, London, England, August 1985, pp.244–251.

[Dill88] Dillon, L.K., Avrunin, G.S., and Wileden, J.C., *Constrained Expressions: Toward Broad Applicability of Analysis Methods for Distributed Software Systems*, to appear in **ACM Transactions on Programming Languages and Systems.**

[Gutt82] Guttag, J.V., Horning, J.J., and Wing, J.M., *Some Notes on Putting Formal Specifications to Productive Use*, **Science of Computer Programming**, Vol2, December 1982, pp.53–68.

[Hoar85] Hoare, C.A.R., *Communicating Sequential Processes*, **Communications of the ACM** Vol.21, No.8, 1978, pp.666–677.

[Long89a] D.L. Long and L.A. Clarke. *Task Interaction Graphs For Concurrency Analysis*, **11th International Conference on Software Engineering**, Pittsburgh, Pennsylvania, May 1989, pp.44–52.

[Long89b] D.L. Long, L.A. Clarke, and J.M. Fialli. *Ada Language Considerations for Concurrency Analysis*, **6th Washington Ada Symposium**, McLean, Virginia, June 1989, pp.75–80.

[Mura83] Murata, T., *Modeling and Analysis of Concurrent Systems*, in **Handbook of Software Engineering**, C.R. Vick and C.V. Ramamoorthy, editors, New York, Von Nostrand Press, 1983, Chapt. 3.

[Mura89] Murata, T., Shenker, B. and Shatz, S.M., *Detection of Ada Static deadlocks Using Petri Net Invariants*, **IEEE Trans. on Soft. Engr.**, Vol.15, No.3., March 1989, pp.314-326.

[Olen86] Olender, K.M. and Osterweil, L.J., *Specification and Static Evaluation of Sequencing Constraints in Software*, **University of Colorado, Boulder, Department of Computer Science, CU–CS–335–86**, June 1986.

[Rich86a] Richardson, D.J. and Thompson, M.C., *A New Model for Error Detection*, **University of Massachusetts, Department of Computer and Information Science, Technical Report 86–64**, Amherst, Massachusetts, December 1986.

[Rich86b] Richardson, D.J. and Thompson, M.C., *An Analysis of Test Data Selection Criteria Using the RELAY Model of Error Detection*, **University of Massachusetts, Department of Computer and Information Science, Technical Report 86–65**, Amherst, Massachusetts, December 1986.

[Stan88] Stankovic, J.A. and Ramamitham, K. **Hard Real-Time Systems**, Computer Society Press, Silver Spring, 1988

[Shat88] Shatz, S.M., Mai, K., Moorthi, D., and Woodward, J., *A Toolkit for Automated Support of Ada Tasking*, **Proceedings of the 9th International Conference on Distributed Computing Systems**, 1988, pp.595-602.

[Tayl80] Taylor, R.N. and Osterweil, L.J., *Anomaly Detection in Concurrent Software by Static Data Flow Analysis* **IEEE Transactions on Software Engineering**, SE-6,3, May 1980, pp.265–278.

[Tayl83a] Taylor, R.N., *A General–Purpose Algorithm for Analyzing Concurrent Programs*, **Communications of the ACM**, Vol.26, No.5, May 1983, pp.362–376.

[Tayl83b] Taylor, R.N., *Complexity of Analyzing the Synchronization Structure of Concurrent Programs*, **Acta Informatica**, Vol.19, 1983, pp.57–84.

[Tayl86] Taylor, R.N., Clarke, L.A., Osterweil, L.J., Selby, R.W., Wileden, J.C., Wolf, A.L., and Young, M., *Arcadia: A Software Development Environment Research Project*,

Second International Conference on Ada Applications and Environments, Miami, Florida, April 1986, pp.137–149.

[Tayl87] Taylor, R.N., Baker, D.A., Belz, F., Boehm, B.W., Clarke, L.A., Fisher, D.A., Osterweil, L.J., Selby, R.W., Wileden, J.C., Wolf, A.L., and Young, M., *Next Generation Software Environments: Principles, Problems, and Research Direction*, Arcadia Document, **University of Massachusetts, Department of Computer and Information Science, Technical Report 87–63**, Amherst, Massachusetts, July 1987.

[Youn86] Young, M. and Taylor, R.N., *Combining Static Concurrency Analysis with Symbolic Execution*, **University of California, Irvine, Department of Information and Computer Science**, September 1986.

[Youn88] Young, M., *How to Leave Out Details: Error-Preserving Abstractions of State-Space Models*, Proceedings of the 2nd Workshop on Software Testing, Verification and Analysis, Banff, Canada, July 1988, pp.63-70.