IDRIS : Interactive Design of Reactive Information Systems

Peter Osmon and Philip Sleat*

Systems Architecture Research Centre
Computer Science Department
City University
London EC1V 0HB

Abstract. The IDRIS project is developing a model, a methodology, a notation, and a CASE tool for the interactive design of real-time, or reactive, systems. Increasingly such designs are implemented on parallel hardware, for performance and reliability reasons. The work described here addresses performance, but not reliability, issues.

The model describes a real time system in terms of a database and a set of asynchronous tasks. Each task is the system's response to an external stimulus event. A task decomposes into atomic transactions, acting on entities in the database, connected by a control structure which is determined by entity access constraints.

Based on a static analysis of these data dependencies, the methodology is able to rely on scheduling rather than conventional locking to control access to the database. The model describes real time systems generally. The methodology concentrates on the class of hard real time systems designs - those which must guarantee to meet the response latency and throughput constraints specified by the requirements. The methodology uses scheduling and allocation heuristics to map tasks and their transactions onto physical processors.

The methodology has a number of stages and the designer is assisted by four graphical notations: event context diagrams; data dependency rings for capturing both intra task and inter task data dependencies; precedence graphs for expressing the intra task control structure implied by the data dependencies; Gantt charts for expressing worst case schedules. The methodology assumes a run time environment with a hierarchy of schedulers to control processing and access to data entities. An interactive CASE tool helps the designer capture and manipulate the design information, display partial designs, and perform allocation and scheduling trials against the specification. An example real time system design, for a ship control system, generated using the methodology and the CASE tool, is briefly described.

* present address: Marex Technology Ltd., Marex House, 88 The High Street, Cowes, I.O.W., U.K.

1. Introduction

The paper introduces a real time systems model and then describes, a CASE tool and the associated methodology for designing hard real time systems, and the run time environment assumed by the methodology. These are described in more detail in [Sle 91].

Hard real time systems are reactive systems where the specification includes "non functional" performance criteria, for at least some of the stimuli, to the effect that- the responses must be guaranteed to occur within particular time intervals and minimum throughput rates must be supported. Evidently the design of hard real time systems must include analysis of worst case behaviour in order to verify conformance with the specification. (An alternative approach to real time system design involves a statistical analysis and probabilistic statements about conformance. This approach can indeed deliver real time system designs, but not "hard" real time system designs. Statistical designs, where the probability of failure to meet the stimulus-response latency and throughput constraints is below certain limits, may be called "firm" real time designs.)

For performance reasons many hard real time systems must be implemented on parallel hardware. The allocation of data and processing across the system and scheduling of the processing are then major design problems. These problems are generally too complex to be solved dynamically, while guaranteeing to meet the hard real time constraints, and so a static analysis must be performed off line. Capture of specification, optimisation, information and subsequent worst case analysis of designs containing many interacting parts is extremely tedious and prone to error and so mechanisation using a CASE tool is very desirable.

2. The Model

2.1 Introduction

The model is consistent with the current practice of basing real time designs on a database. The model is, however, unusual in making two strong assertions about the behaviour of systems so as to simplify the design problem. The first is that systems can generally be partitioned into a set of highly decoupled subsystems ("tasks") where each task is associated with a distinct stimulus, or stream of stimuli. I.e. tasks have different stimuli and so there is no control coupling between them- they are asynchronous with respect to one another. The second assertion is an extension of the first: although there is no control coupling between tasks, they may share data, subject only to the constraints that writes are serialised and each task has a consistent view of the data. The only problem these assertions seem to cause is how to handle "freshness" which is treated as a constraint and introduced into the design at an iteration stage. As an example to illustrate these two asumptions consider a control system for a chemical plant. A particular chemical vat may respond to changes in temperature and pressure. A task could be activated in response to critical temperature conditions. A separate task could be activated to handle critical pressure conditions. Although decoupled by independent triggering events, these two tasks may share system data; for example each task may consult the same data set describing the operating condition for the chemical vat.

The model provides two levels of system decomposition (task and transaction) and two viewpoints at each level (control and data entity).

2.2 Decomposition

The criterion for decomposition of the specified system into tasks has been given above. Tasks are composed of transactions: each one is a function applied to stored data entities. Consider the "temperature task" controlling the chemical vat. This may be decomposed into transactions to examine the current temperature; check the desired temperature; generate changes to heaters; issue warnings to operators etc.

2.3 Control Viewpoint (Intra Task)

A control flow graph, or precedence graph, in which the nodes are the transactions and called the Transaction Precedence Graph (TPG), describes the order in which transactions respond to an external stimulus incident on the task. This ordering is based on constraints imposed by the system specification (for example: the temperature control task must strive to stabilise the temperature before informing the operator of critical conditions) as well as constraints necessary to protect the integrity of the database (for example, where two transactions both update the same entity, there must be "write serialisation").

2.4 Data Entity Viewpoint (Intra Task)

Transactions occur concurrently except where requirements specifies a partial ordering or serialisation of writes imposes a partial ordering. This ordering information, of transactions within a task, defines the precedence graph for each task referred to above.

Determinism requires that the value of a data entity accessed by a task instance (see next section) should only be changed by the transactions within that task instance.

For the purposes of the next section it is generally convenient to overlay the precedence graph with "regions" associated with each data entity. An *entity region* encloses all transactions accessing the entity. A *critical region* encloses transactions accessing the entity up to the one making the last write access.

2.5 Control Viewpoint (Task Level)

A definition of each task (effectively definitions of the constituent transactions plus the precedence graph) is stored in the machine. The effect of a stimulus event arriving is to "peel off" an instance of the task which then executes independently of other instances. This execution is guided by some static representation of the precedence graph.

2.6 Data Entity Viewpoint (Inter Task)

During its lifetime an entity has a succession of values or *versions* (identified by *version numbers* N). To ensure determinism, critical regions on an entity occurring in different instances (of the same or different tasks) must be serialised. To achieve this, versions must be "write once" and, after an instance has left the critical region, the version number should be one greater than on entry i.e. should change to N+1.

In the case of read-only access it may be desirable that the task instance is given access to the "latest" version of an entity, in the same way that writes are. However, in general, this will constrain performance unnecessarily and it is preferable to satisfy explicit "freshness" constraints which should be specified in the requirements.

3. The Methodology

3.1 Introduction

The design methodology, which rests on the model, relies on scheduling to implement write serialisation- both intra task and inter task. Further, writes to an entity are only committed when a task instance leaves its critical region on the entity, allowing the scheduler to abort all transactions within the critical region up to this moment.

The methodology makes an assumption about the streams of stimuli incident on the system, without which a worst case analysis is impossible: the shortest interval between consecutive stimuli in a stream is finite and its value (called the minimum repetition time- MRT) is included in the specification. Consider the temperature control task for the chemical vat. The MRT for this task is guided by the physical behaviour of the devices that read the temperature; each device has a minimum time after being used before it can be used again. Consequently, there is a minimum time after "triggering" before the temperature task can trigger again.

In order to generate schedules by static analysis, when the analysis is performed it is necessary to know the trigger times for tasks. Since aperiodic tasks are likely to be included in the task set, as well as periodic ones, this is not generally possible. However, for purposes of worst case static analysis, which is what is required for hard real time systems, aperiodic tasks can be treated as though they are periodic with a period of their MRT.

For purposes of static analysis, the temporal properties of a task are encapsulated in the triple Task ID (execution time, deadline, MRT) the three components of the tuple being measured in the time units (TU's) of the design (eg microseconds). Execution time (ET) is the elapsed time to complete the task. Deadline is the worst case latency (stimulus-response delay) and MRT (minimum repetition time) is the worst case throughput of this task specified by the requirements. (Note that the execution time must be less than the deadline. Note also that MRT may be less than the deadline, implying that a sequence of instances of the same task, at different stages of processing, may be present concurrently in the system.)

The static analysis must be based on some scheduling heuristic. The scheduling policy used in IDRIS is earliest deadline first (EDF), but this choice is not fundamental and another could be substituted. IDRIS offers the designer a choice of EDF with, or without, preemption. (Preemptive designs involve tasks backing off to the beginning of critical regions and hence are more complex. However, they are sometimes necessary.)

The static analysis must also assume some heuristic for allocation of tasks and transactions to processors. The heuristic presently being used has two parts. First, each task is divided into subtasks, with sub tasks being allocated to different processors within a "cluster". Graphically, a sub task is a vertical slice of the task's TPG. This division into sub tasks, and allocation to different procesors, maximises the potential for concurrency, and may indeed be necessary in order to achieve an execution time less than the deadline. Second, the set of clusters of logical procesors is mapped onto the number of physical processors allowed by the requirements. In this mapping, transactions, from different tasks which share the same data entity, are, as far as possible, placed on the same processor, so as to minimise the traffic in data entities. This allocation scheme has similarities to [MA84] and [MLT82].

The worst case static analysis, making use of transaction ET's, is performed to determine whether all tasks can complete within their deadlines, without preemption. In the case of failure, preemption can be tried or attempts can be made to improve the allocation. Continued failure means the design is not feasible without a relaxation of the specified hard real time constraints.

3.2 Procedure

The methodology contains the following sequence of steps:

1 Identification of triggers (stimuli)
2 Decomposition of system into tasks
3a Identification of data entities, and
3b Decomposition of tasks into transactions
4 Representation of database interactions using DDR's
5 Representaion of tasks using TPG's
6 Conversion of aperiodic tasks to periodic
7 Allocation of tasks and data entities to processors
8 Static analysis of schedulability.

The content of some of these steps has been outlined already. In the first step, as the triggers are identified in the statement of requirements they are given unique names and sorted into periodic and none periodic sets. At the second step, independent tasks are named and identified one-to-one with the triggers. Each task is then decomposed into transactions (of some granularity appropriate to the implementation platform) which manipulate a set of data entities. This is the third step. The designer is expected to derive definitions of the transactions and the data entities within each task from the specification.

The definition of each task is not completed until the fifth step: derivation of the control structure, expressed as a Transaction Precedence Graph (TPG) for each task, linking the transactions in the task, followed by identification of the critical regions within the task.

But before this, the designer is required to make the fourth step- drawing data dependency rings (DDR's) for each entity. A DDR displays all the interactions with that entity by identifying the transactions in all tasks that access it. The DDR shows up the conflicting accesses by transactions within a task. These conflicts will be resolved by serialisation, thereby imposing a partial ordering on the transactions within a task. (There may be additional constraints on transaction order specified in the requirements.) The DDR also shows up data entity access conflicts at the task level. These too are resolved by serialisation, but not at the task level since this is unduly cautious: it is critical regions not whole tasks that set limits on sharing.

The sixth and seventh and eighth steps were discussed in the previous sub-section. If the procedure fails to generate a feasible design, then steps seven and eight are iterated. Continued failure means that it is not possible to meet the requirements, and these must be relaxed. Relaxation might allow more processors, or more powerful processors, to be used.

4. Run Time Environment

The methodology assumes the existence of a run time environment on the implementation hardware, consisting of a hierarchy of three schedulers, to support the design. The highest level creates a task instance in response to a trigger. The middle level manages access to critical regions. the lowest level manages the individual transactions. Figure 1 shows the scheduler hierarchy and a system fragment consisting of three processors and two tasks.

The Task Scheduler is distributed across all processors. It is invoked by triggers (stimuli) and by "task finished" events. It maintains a table of task instances and ensures that successive instances of a task stay in sequence. The Critical Region Scheduler is invoked when a task enters a critical region and when it leaves. It maintains a lock table and a queue of waiting tasks and makes access and pre-emption (including back-off) decisions. The Transaction Scheduler (one for each task instance) is invoked by the Task Scheduler but subject to control by the Critical Region Scheduler. It includes a representation of the TPG for the task instance. The three levels of scheduling are arranged to cooperate to manage the processing in accordance with the EDF policy, while ensuring serialised writes and the atomicity of critical regions.

5. Graphical Notations

5.1 Introduction

A number of graphical notations are used by the IDRIS methodology and CASE tool. They

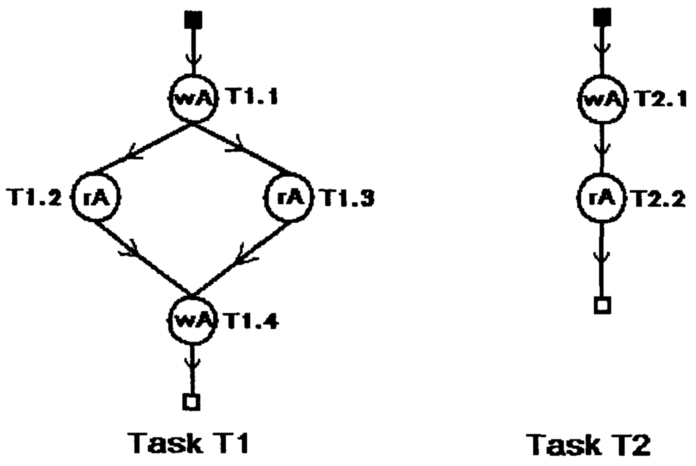

Fig. 1(a): Transaction Procedure Graphs for tasks T1 and T2

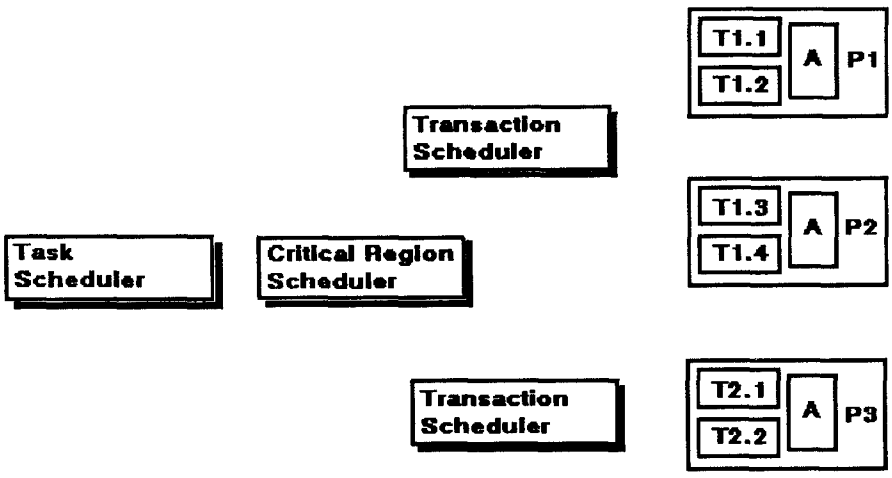

Fig. 1(b): Scheduler Hierarchy and Allocation to Processors are summarised below.

5.2 Event Context Diagram (ECD)

This is a graphical notation adapted from dataflow diagrams. Its purpose is to capture all the stimuli and response events specified for the system as a preliminary to associating a distinct task with each stimulus-response pair. An example ECD displayed by the IDRIS tool is shown in Figure 2.

5.3 Data Dependency Rings (DDR's)

This is a graphical notation introduced to help the designer grapple with the complexities, referred to above, of the competition between transactions and between tasks for access to data entities. An example DDR is shown in Figure 3. Notice the tasks arranged round the ring, and the transactions involving the entity shown within each task. There is a separate DDR for every data entity in the system. When they have all been drawn the information content of all the precedence diagams for the system has been captured and these may then be drawn automatically. Ordering constraints imposed on transactions within a task may alse be described with an annotation around the edge of the DDR. For example in the first constraint described in section 2.3 we could indicate on the transaction which informs the operator of ciritcal temperature conditions that it must "wait for" the transaction that stabilises the temperature to finish.

5.4 Transaction Precedence Graphs (TPG's)

Figure 4 shows the TPG's for some of the tasks in the Ship Control example introduced in section 7. The dotted arcs represent run time control choices (which may be expressed as If...Then...Else statements in the implementation code for the task). The width of a TPG measures the maximum amount of concurrency available in the task for the design to exploit. TPG's may be annotated with Execution Times for the task as a whole or for individual transactions or for critical regions. the TPGs are automatically generated from the information described within the DDRs. The TPGs exhibit the maximum possible concurrency for a task while at the same time respecting the "write serialisation" constraints described within the DDRs.

5.5 Scheduler Gantt Charts (SGC's)

This notation is introduced to present earliest deadline first scheduling information from the design so that satisfaction of the hard real time constraints for the system may be verified by inspection. The Gantt chart has time along the "x-axis" and tasks along the "y-axis". The TPGs determine the maximum execution time for a task. The Gantt chart has a line for each task. On this line, those regions of time where the task is active are marked off, assuming the task is initially triggered at T_0 and each task is continuously retriggered at its MRT. An example of this is shown in figure 5. The Gantt chart can then be used to verify the feasibility of scheduling decisions.

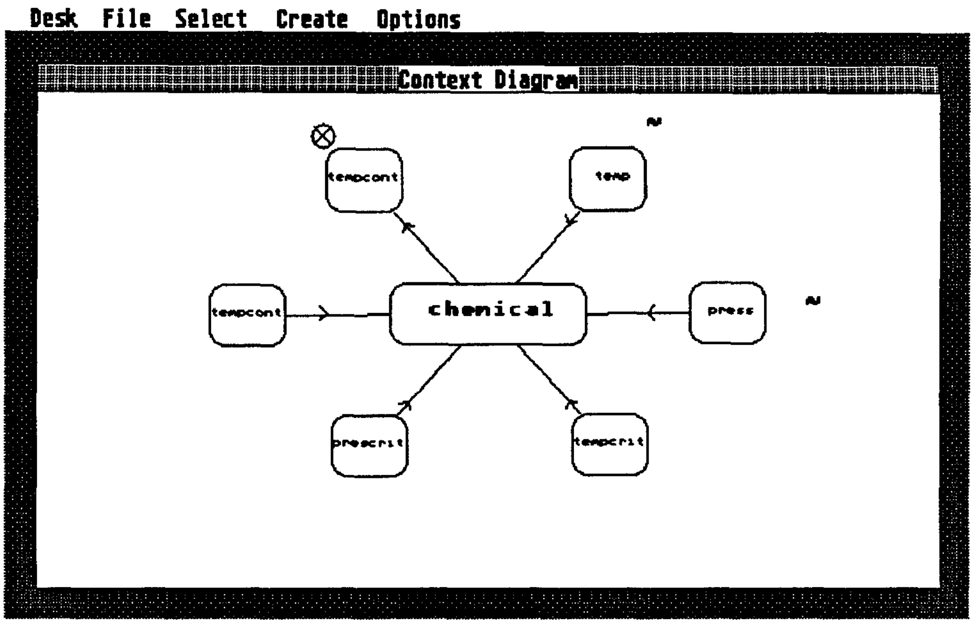

Fig. 2: An example Event Context Diagram

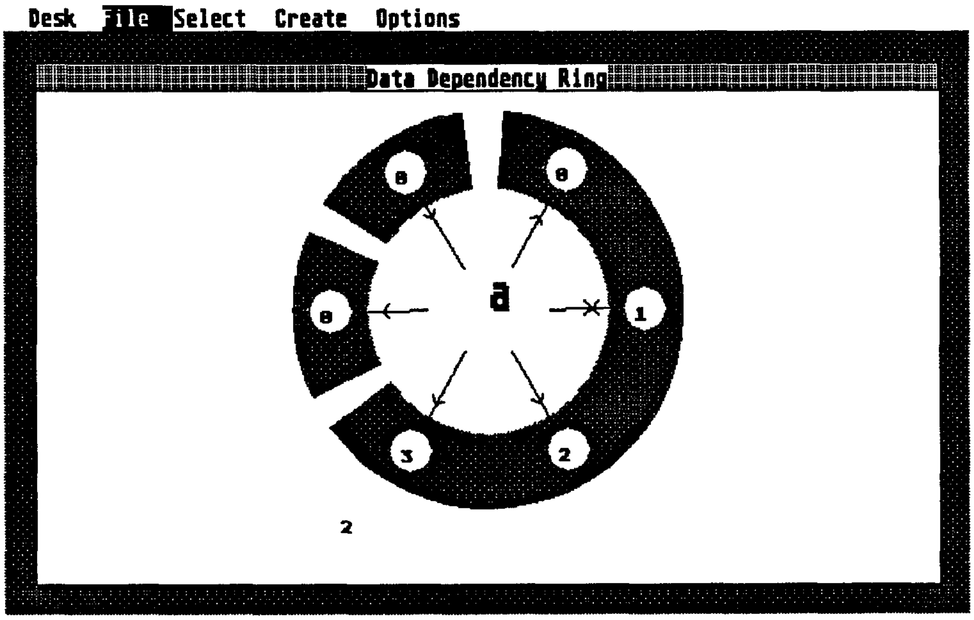

Fig. 3: An example Data Dependency Ring

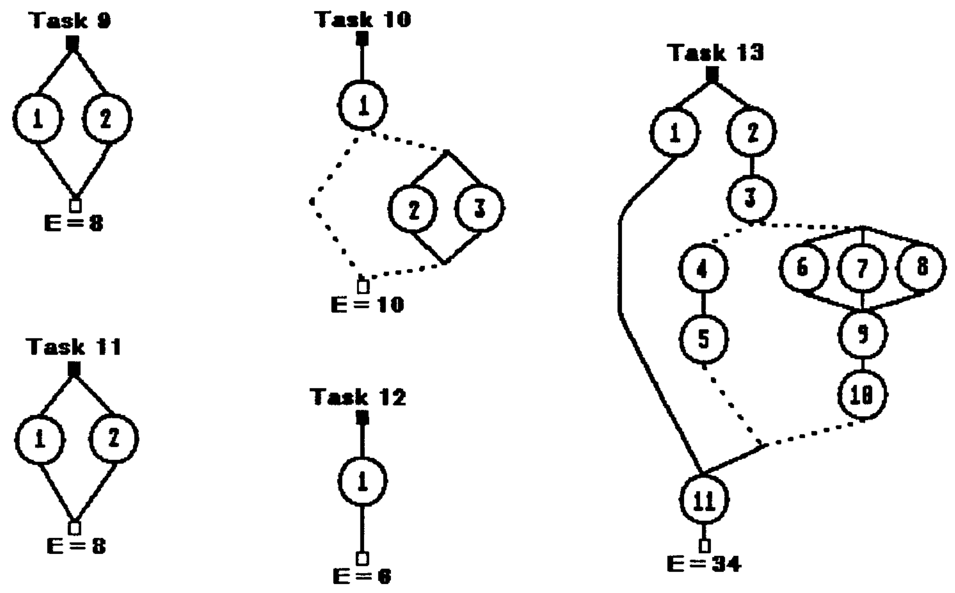

Fig. 4: Transaction Precedence Graphs, Tasks 9 to 13, Ship Control System

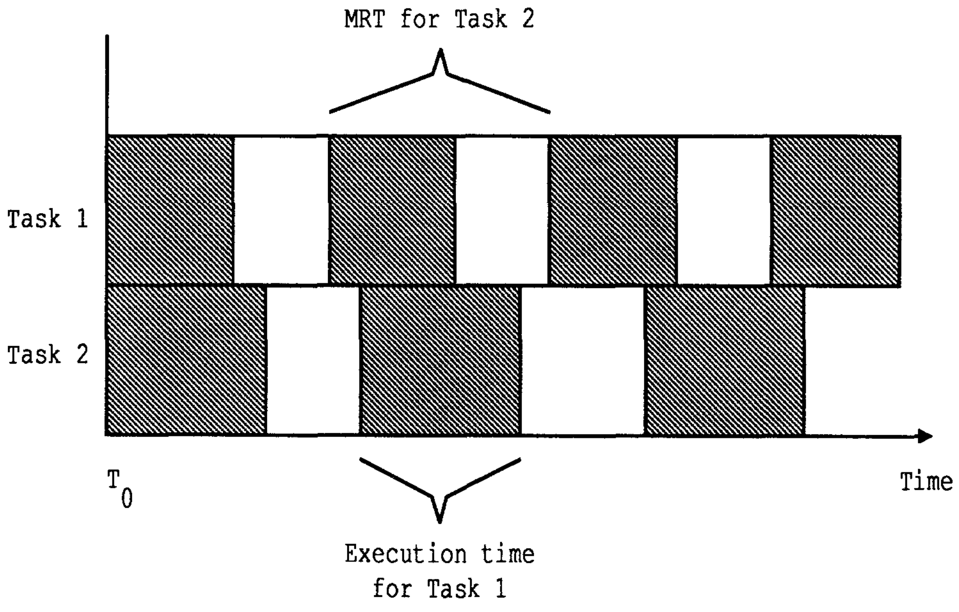

Fig. 5: An example Scheduler Gantt Chart

6. The CASE tool

An interactive CASE tool has been written to assist designers using the IDRIS methodology. The user is prompted by the tool to input information in the order needed by the methodology and the tool presents partial designs using the various IDRIS graphical notations.

The tool has been written in C under a GEM environment to provide a user friendly "WIMP" (Windows, Icons, Menus, and Pointers) interface consisting of

- Drop Down Menus for selecting actions
- Forms for entering data
- Windows in which information and results are displayed.

Besides capturing design information, the tool relieves the designer of two tedious and error prone stages in the design process. The first is extraction of TPG's for the system once the set of DDR's has been defined. This involves scanning the DDR's and use of a concurrency optimising heuristic. The second is a detailed static analysis of the three levels of scheduling with reporting on the feasibility of the design.

7. An Example System Design

IDRIS has been used in the design of a ship control system. This example is larger than many design examples described in the literature. The system is an embedded computer control system, for a fleet of commercial ships, which is responsible for automatically monitoring and controlling the state of a ship's engines, guiding the ship between destinations, accepting new courses and commands from the operator, monitoring and sending communications between ships of the fleet.

The method and CASE tool have been used to generate

- context diagram,
- DDR's,
- TPG's,
- allocation and schedule,

for a design for this system. The tool was then used to help the designer verify that with this allocation and schedule the system would satisfy the hard real time deadlines.

Analysis revealed 23 asynchronous tasks and hence 23 TPG's. Worst case task retrigger times specified in the requirements were entered. Critical regions were identified and worst case execution times calculated by summing transaction times on logical processors. 13 database tables +5 sub tables + 15 peripheral devices treated as database entities were identified and hence 33 DDR's were elicited from the requirements and scanned by the tool to determine the TPG's.

Allocation turned out to require 54 logical processors. Requirements had specified 6 physical processors. The logical to physical mapping was performed using the allocation heuristics outlined in this paper. Scheduler Gantt Charts for the physical processors, using worst case retrigger times for all the tasks, were generated to confirm that the requirements were met.

8. Conclusions and Future Development

8.1 Conclusions

IDRIS emphasises an important property of reactive systems that appears to be undervalued in other design methods (for example [WM86] and [YC78]). This is the role of the shared data entity and its effect on both latency and throughput deadlines. The implicit control flow imposed on otherwise independent real time tasks is taken into account from the early stages of the design.

Essentially, the design method depends on a static analysis of the implicit control flows so that scheduling can take the place of conventional locking. The static analysis of the scheduling problem doesn't generate a schedule directly, as in other static scheduling techniques; the analysis simply determines the effectiveness of the dynamic heuristic (Earliest Deadline First). If the analysis shows that the heuristic is successful, then at run time the heuristic will correctly decide what to do with newly triggered tasks such that all deadlines are met. This approach is similar to [LL73].

8.2 The Future

The model implies a hierarchical run time environment for real time systems, involving three levels of scheduling, which appears to provide a good platform for addressing reliability issues in the next stage of development of the model.

At present the methodology stands alone and lacks a definite interface to the requirements specification stage of a project. We intend to investigate the possibility of tying the methodology to a particular model of requirements. [NS90] might provide a suitable starting point.

The first version of the IDRIS CASE tool is written in C and runs under GEM. It is currently being ported, and partially rewritten, to run on Unix platforms under X-Windows. The new version will provide better support for allocation and simulation of the run time environment.

References

[KN84] H. Kasahara and S. Narita. Practical Multiprocessor Scheduling Algorithms for Efficient Parallel Processing. *IEEE Transactions on Computers,* C-33(11), Nov. 1984.

[LL73] C.L. Liu and J.W. Layland. Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. *Journal of the ACM,* 20(1), Jan. 1973.

[Ma84] R.P.Y. Ma. A Model to Solve Timing Critical Application Problems in Distributed Computer Systems. *IEEE Computer,* 1984.

[MLT82] R. Ma E.Y.S. Lee and M. Tsuchiya. A Task Allocation Model For Distributed Computing Systems. *IEEE Transactions on Computers,* C-31(1), Jan. 1982.

[NS90] M. Nejad-Sattary. An Extended Data Flow Diagram Notation for Specification of Real-Time Systems. *PhD Thesis, City University, 1990.*

[Sle91] P.M. Sleat. A Static, Transaction Based, Design Methodology, for Hard Real-Time Systems. *PhD thesis, City University,* 1991.

[SO91] P.M. Sleat and P.E. Osmon. A Methodology for Real-Time Database System Construction. *In Proceedings of the Third International Conference on Software Engineering for Real-Time Systems.IEE* Sept. 1991.

[WM86] P.T. Ward and S.J. Mellor. Structured Development for Real-Time Systems, volume 1,2 and 3. *Yourdon Press, New Jersey,* 1986.

[YC78] E. Yourdon and L. Constantine. Structured Design. *Yourdon Press,* 1978.