Efficient development of real-time fault-tolerant controllers version 0.95

Claudio Pinello
Luigi Palopoli

21st July 2001

1 Introduction

This research plan deals with the synthesis of real-time embedded controllers, taking into account constraints and design goals pertaining to the domain of the control application and that of its implementation. A particular concern is on real-time scheduling and fault tolerance, two critical requirements of an emerging class of embedded applications (e.g. "by-wire" automotive controllers). We perceive that systematic approaches in this discipline would enable shorter development cycles and more efficient designs. Designers should concentrate on high level decisions and have large support of synthesis and analysis tools to take care of detailed tuning and validation. Most of the detailed decisions should be automatically derived as solution to optimization problems or should be the result of synthesis based on rich libraries of debugged components. Both flows should give sufficient insight to the designer and should allow for his guidance.

This document is organized as follows. Section 2 offers an overview of the proposed methodology summarized in Figure 1; the three design activities and the performance evaluation activity are briefly illustrated. Section 2 also presents some possible design flows supported by the methodology. Particular emphasis is given to iteration loops involving the use of performance metrics to drive architecture and/or behavior design. Finally, in section 3 we identify the research tasks that we want to pursue, with references to relevant bibliography.

2 The envisioned methodology: an overview

A pictorial representation of the methodological approach we advocate is shown in Figure 1 where ovals represent activities, boxes represent artifacts or components and round cornered boxes represent design goals. An arrow drawn from an activity into a box is used to denote that the content of the box is a result of the activity. Conversely, arrows drawn from boxes denote inputs to activities. The graph also includes boxes which are not results of any activity, referred to inputs from the designer. At a first glance, it is possible to recognize that the envisioned approach is inspired to a common conception in modern embedded systems design: behavioral and architectural design should be as much as possible orthogonal activities. By leveraging this principle, it is possible to transform a confused sequence of trial-and error iterations into a well defined engineering process, unfolding unprecedented opportunities in the search for optimal performance/cost trade-offs [Bo97].

Inputs to the design process. In the assumed scenario, inputs to the design process are the following:

- plant model: it consists of a mathematical description of the physical plant the controller will be applied on;
- application constraints: they derive from physical considerations on the plant and on the application which any design solution has to meet: e.g. the maximum lateral acceleration during a maneuver has to be less than a given value for the tire traction of a vehicle to endure it;

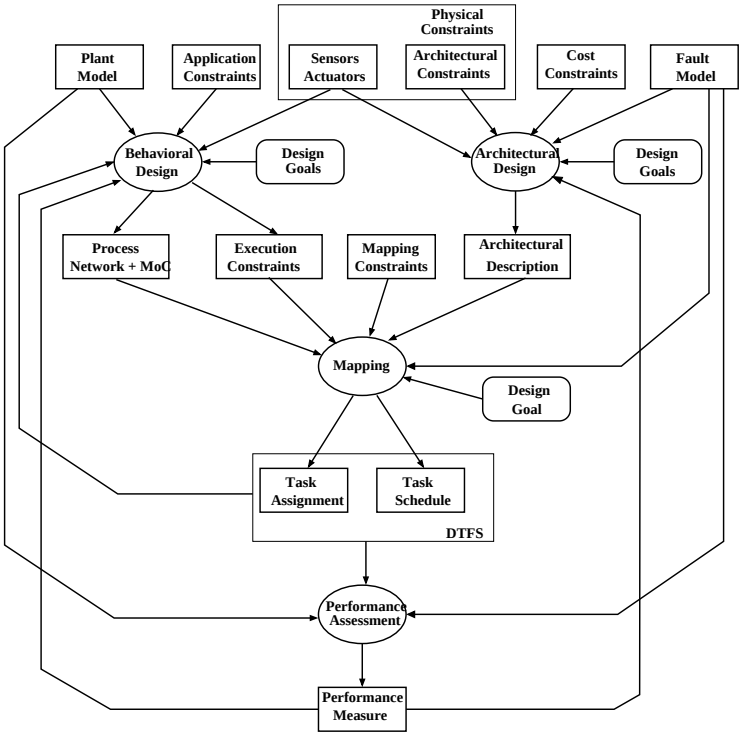

Figure 1: Pictorial representation of the envisioned methodology: ovals represent activities, boxes represent artifacts and round cornered boxes represent goals.

- physical constraints: this class of constraints includes the choice of sensors and actuators and their physical deployment on the plant; e.g. if the objective of the design is a brake "by wire" system, a designer may wish *a priori* to deploy a computing unit in the proximity of each wheel;
- cost constraints: for the system production to be affordable the total cost of the used hardware architecture is bounded to an extent decided by the project management;
- fault model: this defines the type and numbers of faults, along with their pattern of occurrence, that the implemented controller is required to withstand.

In addition to the above described constraints, further inputs to the activities are the design goals, which are usually expressed using domain specific performance metrics. For a performance metric to be useful, its relevance to the development process has to be matched with a certain degree of analytical tractability. In the sequel, we will introduce examples of suitable performance metrics for each of the activities composing the design process.

Design activities The activity marked as **behavioral design** produces as outputs a description of the controller expressed in terms of a *network of processes* endowed with a *Model of Computation*. These terms are borrowed from the Metropolis lingo, where they are used respectively to characterize a purely behavioral description of a system and the activation (*firing*) and interaction rules for the behavioral components (see web site `http://www.gigascale.org/metropolis/`). Another outcome of the behavioral design is a set of execution constraints for the processes, which may be expressed in a different way according to the adopted model of computation. For instance, a precedence constraint is to be considered as an execution constraint. Other types of execution constraints for a network of processes are end-to-end timing constraints [GHS95] or even activation periods for time triggered processes [TH01]. Inputs to this phase are:

- a physical model of the plant;

- application constraints;
- knowledge on sensors and actuators to be used.

The choice of the physical model of the plant will be restricted to linear time invariant plants. Particular classes of nonlinear models could be considered for their particular relevance in practical applications or for their analytical tractability (e.g. chained form). Typical application constraints include stability, guaranteed rates of exponential convergence, robustness etc. As far as sensors and actuators are concerned, we will consider heterogeneous flows of information typically leading to multirate digital control schemes. The design goals leading this activity, as shown below, could either belong to the control domain (e.g. optimal LQG performance) or to the implementation domain (e.g. find a stabilizing controller that allows for the loosest possible requirements in terms of timing behavior to enable the use of low cost architectures).

The activity marked as **architectural design** produces a description of HW/SW components to be used for the implementation. This design activity aims at selecting the distributed architecture (hardware, RTOS, device drivers, protocols etc.) to support the given application process network. The selection process can be driven by different goals and constraints. The constraints include

- maximum total cost (sum of the cost of each component)
- architectural constraints (e.g. dictated by the location of sensors and actuators on a large plant)
- fault model (e.g. the fault model may exclude any resource with less than k processors or p channels).

The design goals may include

- minimize total cost
- minimize number of processors and/or channels
- minimize resource utilization factors.

This design activity takes as feedback the performance assessment from the mapping results for further iterations, if needed.

The **mapping** activity aims at generating a distributed fault-tolerant scheduling (DFTS) encompassing both a process and communication assignment and a schedule. This activity, in our view, is an automated *synthesis* process realized in different steps: 1) augmenting the behavioral description with replicated and additional components to meet the fault tolerance constraints, 2) allocating architectural resources to behavioral components, 3) deciding the scheduling policy to enforce the execution and fault tolerance constraints. Input data and constraints include

- the application process network, with the associated MOC,
- the architecture model, including performance model,
- execution constraints, (e.g. timing constraints, precedence constraints),
- the fault model.

In this phase the designer may wish to impose further constraints (for instance it is reasonable that behavioral components devoted to preprocessing sensor data be located in the neighborhood of their related sensors). Moreover, specific design goals can be introduced to drive the mapping phase toward a region of the solution space regarded as particularly promising. Relevant examples can be

- minimize resource utilization,
- minimize the response time (input-to-output maximum delay),
- minimize the time to react to a fault.

Another outcome of the mapping phase is the analytical (worst-case) evaluation of the timing behavior of tasks (CPU utilization, communications' workload etc.).

The DFTS, the architecture, the fault model and the plant model can be used as inputs to another activity, denoted here as **performance assessment**. This activity can be developed along two different directions: simulation and prototyping. We are particularly interested in the performance assessment by simulation. The simulation tool operates on a heterogeneous aggregation of models (continuous time plants, communication links, real-time schedulers) and permits the fault-injection according to pre-specified deterministic/stochastic patterns. The result of this phase are performance measures which can include:

- control theoretic performance (overshoots, rise time, linear quadratic functions, etc.),
- dependability measures (number of tolerated resource failures, response time to a failure, etc.)
- timing behavior (CPU utilization, end-to-end delays etc.).

Although the synthesis of the schedule carries some meaningful information concerning the timing behavior of the system, we observe that the information collected through simulation can be far richer.

In fact, the schedule is usually synthesized under worst case assumptions about the resource requirements of tasks. On the contrary, simulation can be performed considering stochastic models for the resource requirements (e.g. probability distributions, or histograms) and, therefore, it is possible not only to determine the worst case utilization of resources but also its statistical profile. This information could be used, for instance, to justify the introduction of soft/non real-time activities (such as diagnostic logging) to be executed when the system is underloaded.

Design Flows The diagram in Figure 1 depicts several design paths with different possible iterations. Possible design flows could try to answer the following questions.

Q: what is the least expensive architecture that meets all requirements and constraints?
A: The solution to this problem can found be as follows. The behavioral design produces a model and a set of execution constraints ensuring a low resource demand. The architecture design selects an architecture trying to minimize the cost. The mapping design searches for a DFTS such that the resource utilization factor are minimized. A new iteration may very well involve only the architecture design and mapping activities. The performance assessment (again resource utilization factors) can be used to select a low cost architecture by reducing the capacity of underutilized components and increasing that of overloaded components. Even though some efficient methods could be devised to direct this search (e.g. branch and bound) under particular assumptions, in the general case this process seems hard to automate. Nonetheless manual exploration can benefit from the formalization of the design flow.

Q: what are the best control parameters for a given architecture?
A: This question arises after an architecture is chosen and an assignment of tasks is devised. If the design is parametric in the tasks execution constraints, a behavioral refinement may achieve a better tuning of the application to the architecture. For instance, the design could allow to adapt gains to different activation rates. In this case, the refinement may yield better control performance, if the solution produced by the original mapping is overly conservative. Clearly, goals driving this design activity are chosen in the domain of the control application (e.g. settling time, rate of convergence, etc.) and the initial mapping is used to specify a set of resource constraints of the system.

Q: what does it buy me to use fail stop versus conventional processors?
A: Some recent work on VLSI has focused on self-checking hardware that is capable of detecting some types of error, i.e. deviation from normal behavior. When applied to processors, these techniques allow the production of fail-stop processors (e.g. [BLS97]). Fail stop processors halt execution upon detection of an error, greatly reducing the need of software for fault containment. The goal of the question is to determine whether the cost of this expensive hardware is really out-weighted by the reduction in system complexity. In order to address this point, we need to repeat several times the design cycle assuming a different fault model. The partial or complete automation of the different design activities makes this exploration viable.

3 Tasks

3.1 Resource aware control synthesis

In the context of the proposed design methodology, the behavioral design is expected to accomplish two distinct (and to some extent conflicting) goals. On the one hand, we want to select/devise a set of models and algorithms able to ensure a good level of orthogonalization between the design of the behavior and that of the architecture. On the other, we want that the information contained in artifacts circulating between the different activities be sufficiently rich to ensure the actual respect of the design constraints and the fulfillment of the design goals. It is our persuasion that classical design flows, based on a rigid separation between the work of the control engineer and that of the computer engineer, fail into achieving the latter goal. Since limitations related to the implementation platform are not taken into account during the early phase of the design, they manifest themselves only during the prototyping of the system compelling the developers to expensive trial-and-error iterations.

Such limitations arise from different practical issues which, in a general sense, impose upper bounds on the amount of information that can be circulated in unit time from sensors to actuators.

A good deal of recent control literature has been trying to attack these emerging issues from many viewpoints. Important connections between control theory and information theory have been exposed considering the bandwidth limits of communication links. The problem of stabilization of bandwidth constrained systems is addressed in [WB99], while the state estimation problem for linear systems under bit-rate constraints is analyzed in [WB97, NE98]. The application of LQG techniques to control synthesis with communication constraints is illustrated in [TSM98].

A related area of research is the effect of quantization which is commonplace in all digital controllers. Traditionally quantization has been considered as disturbance to be rejected. Recently, a thread of novel works [BMP01, EM01, Del89] is unveiling new perspectives on this phenomenon, to the point of investigating its unsuspected opportunities.

Effects deriving from the concurrent use of common resources from multiple independent control tasks are analyzed in [SLSS96, RS00]. In [HM99] important results are reported as to the problem formulation and solution when multiple control activities share the same communication media to control a plant. This work is particularly noteworthy for our purposes, since we are interested in unfolding the possible trade-offs between the priorities given to each path in the flow of sensor information through the controller.

As said above, we want to address two different design flows: 1) providing a functional design and a set of execution constraints to be used during the mapping activity, 2) refining the functional design once the architecture and the mapping have been fixed. Two prongs of research activity will be generated from these distinct, though related, design flows.

Control synthesis minimizing the resource requirements The basic problem we want to tackle in this research activity can be summarized as follows: what is the minimal description enabling the realization of a real-time controller with guaranteed performance? Saksena and other in [KRH+96] tried to address this problem in the framework of period calibration design methodology [GHS95]. They assume that a plant is controlled by a graph of tasks sharing the same CPU. Their methodology is in two steps: first, the constraints of the control application (expressed by usual metrics: rise time, overshoot etc) are translated into end-to-end requirements, then the period calibration procedure permits to find a feasible schedule which meets the end-to-end constraints and minimizes the CPU occupation. The main limitations of this work are: 1) only "classical" single rate systems are considered, 2) the methodology enforces a specific architectural mapping (based on a single CPU) which inhibits an authentic exploitation of a concurrent implementation. However, this idea contains some interesting cues which could be generalized and cast into our framework, provided that some basic issues are addressed. The first issue regards the model of computation to be used for specifying the system's behavior. More specifically we should come up with a set of behaviors and execution constraints such that: 1) they enable compliance with the application constraints, 2) they permit the mapping on multiple architectural choices, 3) they are the result of a well-settled and "certifiable" analytical procedures. The basic question is "what is the most abstract known model of computa-

tion" permitting to accomplish these goals? A first possibility we want to explore is the use the time triggered approach [Kop97], as in the Giotto framework [TH01]. The advantages of this model is that it permits to find a deterministic unifying analytical framework for dealing with the control synthesis: Periodically Time Varying Linear systems [SPG84, S.B90, CRN92, MH99]. Moreover it permits to abstract from low-level implementation issues allowing for a relatively free exploration of different architectural alternatives. Following this research direction, an open problem is the formulation of clear design goals to drive the solution search. Notice that, at this level, we have no concept of resource constraints, so a concrete risk is that the problem be ill-posed. In example, the absence of resource constraints easily leads the solution towards infinite activation rates. To cope with this problem, in this research thread, we think of execution constraints as a design goal. In other terms, the synthesis procedure searches for the loosest execution constraints that guarantee a given level of performance, thus easing the work of the mapping activity. In the context of the time triggered model, a possible design goal could be: $\min \sum c_i \frac{1}{T_i}$ where T_i are the activation periods of the tasks and c_i are "tentative" *a priori* evaluations of relative costs. Clearly, the cost evaluation could be subject to refinements based on the feedback of other activities.

Limitations of the time triggered MOC are: 1) the data release jitter is eliminated by making very pessimistic assumptions (output data are released by processes at the end of a period even when they are available before), 2) it imposes tight guidelines on the mapping and architectural design activities. We want to explore possible alternatives based on a revisited form of "end-to-end" constraints [KRH+96, GHS95].

Control synthesis with resource constraints. The scenario is reversed with respect to the previous research activity. In this case we start from a mapped system and we want to refine the parametric behavioral design to maximize control performance. The mapped system provides us with a well defined set of resource constraints (e.g. tasks schedulability). This problem has important similarities with the one addressed in some of the works cited above [HM99, SLSS96, RS00]. Moreover, unlike in the situation described above, the introduction of explicit resource constraints makes the control optimization problem well posed. A basic issue to be addressed is how to represent the resource constraints in a tractable analytical form. At this regard, different models of computation present different difficulties. Once again, the time triggered model of computation offers interesting opportunities which deserve being explored in depth. However, we want to extend the analysis to less restrictive models of computation. Another important point to cope with is: "what are useful performance metrics amenable to analytical treatment in the context of real-time concurrent systems?".

The overall result should be a problem formulation enabling the use of standard optimization techniques (e.g. convex optimization).

3.2 Synthesis of Distributed Fault Tolerant Schedules

Some applications are so critical that their failure may have very high or unacceptable costs (e.g. property, financial, personal damage). In these cases the applications need be resilient to faults[1] in the implementation hardware or even in the software itself. Given the extreme variety of possible faults, one usually needs to make restrictive assumptions. Such assumptions are referred to as *fault model*. A common assumption is fail silence: components either provide a correct result or do not provide any result at all. Usually the fail silence assumption can be relaxed if invalid results are otherwise detected, as in the example of CRC-protected communications. One of the loosest fault models is Byzantine failures [LSP82] in which components can have any behavior, including malicious ones. In addition to type of faults, assumptions in a fault model also regard the number (or even the mix) of faults to be tolerated. Typically adopted fault models assume a single failure, but this number can be higher in case of short MTBF (mean time between faults) or of safety critical applications.

Once the fault model has been settled, a developer is confronted with two challenging activities:

- augmentation of the behavioral description,

[1] Following the classification given in [Lap92]: A fault is the cause of an error; an error is the part of the system state which may cause a failure; a failure is the deviation of the system from the specification.

- scheduling on a distributed architecture.

Starting from a network of process, the augmentation task corresponds to replicating some or all of the processes and the data they exchange. Moreover, additional processes may be needed for voting on the results of different replicas of a same process to establish a common result: the consensus problem [BMD93].

The scheduling task determines an assignment and schedule of the augmented network onto the distributed architecture.

These two tasks are both tedious and error prone, thus introducing the risk of faults in the controller due to poor programming of the fault tolerance mechanisms. Hence it seems profitable to relief the designer from this burden and opt for an off-the-shelf solution or full-synthesis.

As a reference we sketch how these tasks are tackled in Kopetz's time triggered architecture (TTA). One of the greatest advantages in the TTA approach is that the application needs not be aware of the fault tolerant mechanisms which are transparently provided by the architecture [KM99]. This is achieved by fully replicating the hardware and the processes that run on it, the consensus problem being solved within the network interface. The search for an effective scheduling solution is simplified by the time triggered paradigm: an immediate solution is to simply use the same schedule for each replica. Since all replicas of a process are always running (active replicas) even in the absence of faults, the architecture turns out to be potentially underutilized. Another rigidity of the model is that it does not scale well in the number of communication channels: a fault model requiring resilience to more than one permanent fault in the communication system requires a complex design dissipating part of the advantages of the TTA.

We advocate that approaches based on synthesis rather that of–the-shelf solutions promises to overcome the shortcomings highlighted above. As far as the problem of resource underutilization is concerned, we want to allow a flexible use of passive replicas (replicas of a process that run only when the main replica undergoes a fault). Preliminary results have shown the usefulness of this technique in achieving higher schedulability by "reclaiming" resources from non-running replicas [AKH97, CB98]. A further venue of improvement may arise in the context of gracefully degrading applications, where replicas are not an exact copy of the original process. Rather there may be simpler versions with reduced functionality and/or accuracy and likely less resource requirements [CBS00]. This exposes an opportunity to achieve higher schedulability, by requiring strong fault resilience only of the light-weight versions.

Moreover we want to allow general architectures, removing the strict boundaries of the modules and busses found in the TTA. This enables more general fault models also for the communication subsystem. The resulting architecture is a full-fledged distributed multiprocessor system, where each node can be used per se and not as a mere duplicate of another one. All the parallelism in the hardware can then be exploited to speed up the execution of parallel processes of the application [DGLS01, AH00] without affecting the degree of fault tolerance. An important source of inspiration could be some results from the multiprocessor scheduling community [HTK97].

We note that most of the results cited above have been derived under very restrictive assumption on the fault model. We believe some of their founding principles can be rephrased in a more general framework. The expected outcome of this research is a systematization of a set of design techniques which could allow for an easy exploration of design alternatives arising from different fault models.

3.3 Possible Links to Metropolis

Scheduling point-tool In our view, the activity of devising a distributed fault tolerant schedule (DFTS) in an automatic or semi-automatic fashion would be a valuable point-tool in the Metropolis flow. This tool, starting from the behavioral model and from the architectural model, would permit an easy selection of fault scenarios and design goals, generating an optimized mapping, in the sense described above. In essence the tool would provide an implementation of the synthesis procedures outlined in the previous section.

Plant and controller cosimulation We believe that Metropolis would benefit from a native support for continuous/discrete simulations. As a first argument, we observe that continuous time dy-

namical systems are just another model of computation. Therefore, it makes perfectly sense to think them as a particular instance of the general framework for defining different models of computation, advocated by Metropolis.

Secondly, in feedback control applications, closed loop performance is the most important measure of the quality of the design. The need for tools providing this facility is clearly recognized by a consistent thread of novel research and industrial works. For instance, in [EC99] the authors propose a toolbox to simulate a real-time scheduler in a simulink block.

Even tools like VCC (from Cadence) have been extended (see `http://www.parades.rm.cnr.it/ aferrari`), to include support for the simulation of continuous time plants (derived from simulink schematics) into the context of a real-time controller.

Instead, a native support for this type of cosimulation in the design of the library, would permit to exploit synergies with the other components achieving better results in terms of modeling ease and efficiency of the simulation. An example of this approach can be found in the RTSIM tool described in [LP+01]. RTSIM natively permits to model a plant, and a real-time controller, specified according to two different viewpoints: its functional behavior and its HW/SW architecture.

Another feature which we propose to include in the simulation environment of Metropolis is the support for fault tolerance. More specifically, we see two necessary additions: the fault injection and the collection of dependability measures. We want to investigate and formalize the type of fault injection(see [CP95, JAC+93, CJS+99]) mechanisms that should be included to Metropolis as modeling primitives or as simulation parameters. Both deterministic and stochastic injection schemes should be used to test and validate the dependability of the system. In particular deterministic fault injection can help us test and debug the system under specific or worst case fault combinations (typically rare events). On the other hand stochastic models often permit to simulate more realistic behaviors. This is particularly useful when simulating graceful degradation systems, so that one can evaluate the quality of the behavior that will be achieved. To this end, the tool should permit to easily gather statistics on various types of performance metrics related to the dependability of the system.

References

[AH00] J. Aguilar and M. Hernandez. Fault tolerance protocols for parallel programs based on tasks replication. In *Proceedings of MASCOTS*, San Francisco, CA, 2000.

[AKH97] KapDae Ahn, Jong Kim, and SungJe Hong. Fault-tolerant real-time scheduling using passive replicas. In *Proceedings Pacific Rim International Symposium on Fault-Tolerant Systems*, Taipei, Taiwan, 1997.

[BLS97] E. Böhl, Th. Lindenkreuz, and R. Stephan. The fail-stop controller ae11. In *Proc. International Test Conference*, July 1997.

[BMD93] M. Barborak, M. Malek, and A. Dahbura. The consensus problem in fault-tolerant computing. *ACM Computing Surveys*, 25(2):171–220, June 1993.

[BMP01] A. Bicchi, A. Marigo, and B. Piccoli. Quantized control systems and discrete nonholonomy. *IEEE Trans. on Automatic Control*, 2001.

[Bo97] F. Balarin and other. *Hardaware-Software Co-Design of Embedded Systems: the polic approach*. Kluwer Academic Publishers, 1997.

[CB98] M. Caccamo and G. Buttazzo. Optimal scheduling for fault-tolerant and firm real-time systems. In *Proc. IEEE Conference on Real-Time Computing Systems and Applications*, Hiroshima, Japan, 1998.

[CBS00] M. Caccamo, G. Buttazzo, and L. Sha. Capacity sharing for overrun control. In *Proc. IEEE Real-Time Systems Symposium*, Orlando FL, 2000.

[CJS+99] P.E. Chung, Woei-Lee Jyh, J. Shih, S. Yajnik, and Huang Yennun. Fault-injection experiments for distributed objects. In *Proceedings of the International Symposium on Distributed Objects and Applications*, Edinburgh, UK, 1999.

[CP95] J. A. Clark and D.K. Pradhan. Fault injection: a method for validating computer-system dependability. *Computer*, 28(6):47–56, June 1995.

[CRN92] P .Colaneri, R.Scattolini, and N.Schiavoni. The lqg problem for multirate sampled-data systems. *IEEE Trans. Aut. Contr.*, 37(5), 1992.

[Del89] David F. Delchamps. Extracting state information from a quantized output record. *Systems and Control Letters*, 1989.

[DGLS01] C. Dima, A. Girault, C. Lavarenne, and Y. Sorel. Off-line real-time fault-tolerant scheduling. In *Proceedings Ninth Euromicro Workshop on Parallel and Distributed Processing*, Mantova, Italy, 2001.

[EC99] J. Eker and A. Cervin. A matlab toolbox for real-time and control systems co-design. In *Proc. of The Real-Time Computiong Systems and Applications*, Hong Kong, China, December 1999.

[EM01] N. Elia and S. Mitter. Stabiliztion of linear systems with limited information. *IEEE Trans. on Automatic Control*, 2001.

[GHS95] R. Gerber, S. Hong, and M. Saksena. Guaranteeing real-time requirements with resource-based calibration of periodic processes. *IEEE Transaction on Software Engineering*, 21(27), 1995.

[HM99] Dimitris Hristu and Kristi Moransen. Limited communication control. *System and Control Letters*, 37(4):193–205, July 1999.

[HTK97] Koji Hashimoto, Tatsuhiro Tsuchiya, and Tohru Kikuno. A new approach to realizing fault-tolerant multiprocessor scheduling by exploiting implicit redundancy. In *Proc. Twenty-Seventh Annual International Symposium on Fault-Tolerant Computing*, December 1997.

[JAC+93] Arlat J., Costes A., Y. Crouzet, J.C. Laprie, and D. Powell. Fault injection and dependability evaluation of fault-tolerant systems. *IEEE Transactions on Computers*, 42(8):913–23, Aug. 1993.

[KM99] H. Kopetz and D. Millinger. The transparent implementation of fault tolerance in the time-triggered architecture. In *Dependable Computing for Critical Applications*, San Jose, CA, 1999.

[Kop97] H. Kopetz. *Real-Time systems: Design Principles for Distributed Embedded Applications.* Kluwer, 1997.

[KRH+96] N. Kim, M. Ryu, S. Hong, M. Saksena, C. Choi, and H. Shin. Visual asessment of a real-time system design: a case study on a cnc controller. In *Proceedings of the IEEE Real-time Systems Symposium*, 1996.

[Lap92] J.C. Laprie, editor. *Dependability : basic concepts and terminology in English, French, German, Italian, and Japanese*, volume 5 of *Series title: Dependable computing and fault-tolerant systems.* Springer–Verlag, New York, 1992.

[LP+01] Giuseppe Lipari Luig Palopoli et al. A tool for simulation and fast prototyping of embedded control systems. In *Proc. of ACM SIGPLAN 2001 Workshop on Languages, Compilers, and Tools for Embedded Systems (LCTES'2001)*, June 2001.

[LSP82] L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, July 1982.

[MH99] Anthony N. Michel and Bo Hu. Towards a stability theory of general hybrid dynamical systems. *Automatica*, 35, 1999.

[NE98] G. N. Nair and R. J. Evans. State estimation under bit rate constraints. In *Proc. of the 37th IEEE Conference on Decision and Control*, Tampa, Florida, December 1998.

[RS00] H. Rehbinder and M. Sanfridson. Scheduling of a limited communication channel for optimal control. In *Proc. of the 39th IEEE Conference on Decision and Control*, Sidney, Australia, December 2000.

[S.B90] G.De Nicolao S.Bittanti, P.Colaneri. An algebraic riccati equation for the discrete-time periodic prediction problem. *Systems and Control Letters*, 14, 1990.

[SLSS96] D. Seto, J.P. Lehoczky, L. Sha, and K.G. Shin. On task schedulability in real-time control systems. In *IEEE Real Time System Symposium*, December 1996.

[SPG84] S.Bittanti, P.Colaneri, and G.Guardabassi. Periodic solutions of periodic riccati equations. *IEEE Trans. Aut. Contr.*, 29, 1984.

[TH01] C.M. Kirsch T. Henzinger, B. Horowitzm. Embedded control systems development with giotto. In *Proc. of ACM SIGPLAN 2001 Workshop on Languages, Compilers, and Tools for Embedded Systems (LCTES'2001)*, June 2001.

[TSM98] S. Tatikonda, A. Sahaim, and S. Mitter. Control of lqg systems under communication constraints. In *Proc. of the 37th IEEE Conference on Decision and Control*, Tampa, Florida, December 1998.

[WB97] W.S. Wong and R. Brockett. Systems with finite bandwidth constraints - part i: State estimation problems. *IEEE Trans. on Automatic Control*, 42(9), 1997.

[WB99] W.S. Wong and R. Brockett. Systems with finite bandwidth constraints - part ii: Stabilization with limited infromation feedback. *IEEE Trans. on Automatic Control*, 44(5), 1999.