A G-Net Based Environment for Logical and Timing Analysis of Software Systems

Angelo Perkusich[1] and Jorge C.A. de Figueiredo[2]

[1]Departmento de Engenharia Elétrica

[2]Departamento de Sistemas e Computação

Universidade Federal da Paraíba

Caixa Postal 10105

58109-970 - Campina Grande - PB - Brazil

Fone: +55 83 333 1000 R-412, Fax: +55 83 333 1650

e-mail: perkusic@dee.ufpb.br, abrantes@dsc.ufpb.br

Abstract

The application of Petri nets for the modeling and verification of systems, at specification and design levels are well know. Despite of powerful structuring mechanisms available in the Petri nets theory for the construction of the model of complex systems, the designer is still likely to face the problem of state explosion, when analyzing and verifying large systems. Also, when dealing with real-time systems, the verification of timing properties is necessary. A model, named *G-Nets*, and a time extension, named *Fuzzy Time G-Nets*, were introduced to the the modular analysis of complex real-time software systems. In this work we introduce an environment for logical and timing analysis based on this two kind of Petri nets.

1 Introduction

One of the most important aspect that must be considering when specifying and designing a complex distributed software system is its inherent concurrency. Besides this, one must consider structural properties of the system, as well as the behaviors of the components of the system and the communication messages exchanged among these components. Formal techniques must be employed, because they allow the specifier to write unambiguous, clear, and concise specifications, providing a foundation for analyzing specifications for correctness.

The *G-Net* model [7, 14] and the *Fuzzy Time G-Nets* [5, 4, 17, 6], that explores the possibility of using the natural decomposition of distributed systems, are used to the design and implementation of a verification environment to the modular logical and temporal analysis of complex software systems. The logical analysis is based on the so called assume/guarantee paradigm [10]. This modular analysis methodology allows the designer to reason about components, processes or software modules, and their interactions with an environment. When designing a component, assumptions are made about the behavior of the environment so that the local behavior of a component can be specified and verified. When designing the components that compose the environment of another component, the designer must guarantee that these components behave as assumed. Indeed, the designer guarantees the commitment of the environment with respect to the assumed environmental behavior.

In the case of real-time systems, besides the logical analysis, the analysis of timing aspects must be taken into account. Considering Petri, a variety of modifications have been proposed in order to extend Petri nets [11, 20, 21] and/or stochastic [12]. One of the major problem with this approaches when analyzing complex systems is the lack of modular techniques.

In order to perform timing analysis of complex real-time software systems, we introduced an extension for the Petri Net model in order to characterize timing constraints [1, 4, 6, 17]. In this extension, the positive aspects of the deterministic and stochastic approaches are combined in a complementary fashion, i.e., the proposed extended Petri net is amenable to model real-time systems and to make performance analysis of systems. The extension, called *Fuzzy Time Petri Net (FTPN)*, uses a fuzzy approach based in the fuzzy set theory introduced by Zadeh [22]. In this fuzzy approach, the tokens carry a fuzzy time function

that characterizes the possibility of there being a token in a place in a given instant of time. Also, fuzzy time intervals are associated with the transitions to provide ability to represent timing restrictions. The fuzzy time intervals associated with the transitions allows the representation of the three general categories of timing constraints that are considering in real-time systems [3]: maximum timing constraints, minimum timing constraints and durational timing constraints. This extension is good to evaluate performance of systems modeled by a Petri net. For example, for a given system it is possible to compute the minimum, maximum and most probable response time needed to reach a given state from an initial state. Also, aggregate performance indices may be computed.

We integrated the proposed *FTPN* together with *G-Nets*, in order to define a modular approach to perform timing analysis of systems. The integration is called *Fuzzy Time G-Nets*. Then, using the concept of *Fuzzy Time G-Nets*, we can divide a complex system in subsystems which are studied in isolation and the results later combined in order to compute the global solution for the entire system [6].

Besides the theoretical foundation, it is necessary to provide to designers environments, e.g. graphic editors, simulators, and so on. In this paper we introduce the concepts behind *G-Nets* and *Fuzzy Time G-Nets*, and describe an environment to help designers to develop the design of real-time distributed software systems. In this work the emphasis is given to logical and timing analysis modules implemented.

The rest of this paper is structured as follows. Section 2 presents the background and the application of *G-Nets* and *G-Net* systems. Section 3 outlines the logical analysis. Section 4 discusses the timing analysis. In Section 5 we briefly introduce the verification environment for *G-Net* systems. Finally, in Section 6, we present some discussion and the conclusion of this paper.

2 G-Nets and G-Net Systems

In [7] the concept of *G-Nets* and *G-Net* systems were introduced. *G-Nets* are a Petri net based framework for the modular design and specification of distributed information systems. The framework is an integration of Petri net theory with the object oriented software

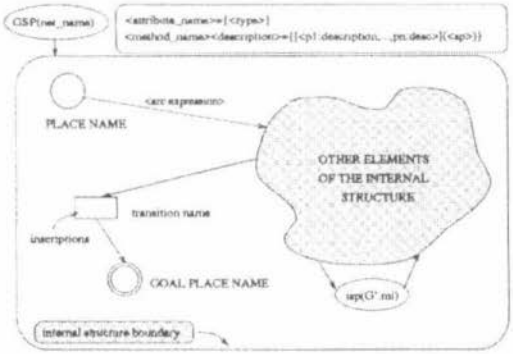

Figure 1: Notations used to represent a *G-Net*

engineering approach for system design. The motivation of this integration is to bridge the gap between the formal treatment of Petri nets and a modular, object-oriented approach for the specification and prototyping of complex software systems. The *G-Net* notation incorporates the notions of module and system structure into Petri nets, and promotes abstraction, encapsulation and loose coupling among the modules.

A specification or design based on *G-Nets* consists of a set of independent and loosely-coupled modules (*G-Nets*) organized in terms of various system structures. A *G-Net* is encapsulated in such a way that a module can only access another module through a well defined mechanism called *G-Net abstraction*, avoiding interference in the internal structure of another module.

A *G-Net G*, is composed of two parts: a special place called *Generic Switch Place (GSP)* and an *Internal Structure (IS)*. The *GSP* provides the abstraction of the module, and serves as an interface between the *G-Net* and other modules. The internal structure is a modified Petri net, and represents the detailed internal realization of the modeled application. The notation for *G-Nets* is very close to the Petri net notation [13]. Among other features, this notation allows the user to indicate communication among *G-Nets* and termination. The notation for *G-Nets* is shown in Figure 1, and is explained as follows:

The *IS* of the net is enclosed by a rounded corner rectangle, defining the internal

structure boundary. The GSP is indicated by the ellipse in the left upper corner of the rectangle defining the IS boundary. The inscription $GSP(net\_name)$ defines the name of the net to be referred by other *G-Nets*. The rounded corner rectangle in the upper right corner of the IS boundary is used to identify the methods and attributes for the net, where: $\langle attribute\_name \rangle = \{\langle type \rangle\}$ defines the attribute for the net where: $\langle attribute\_name \rangle$ is the name of the attributes, and $\langle type \rangle$ is a type for the attribute. $\langle method\_name \rangle$ is the name for a method, $\langle description \rangle$ is a description for the method. $\langle p1 : description, \cdots, pn : description \rangle$ is a list of arguments for the method. Finally, $\langle sp \rangle$ is the name of the initial place for the method. A circle represents a normal place. An ellipse in the internal structure represents an *instantiated switching place (isp)*. The *isp* is used to provide *inter-G-Net* communication. The inscription $isp(G'.mi)$ indicates the invocation of the net G' with method mi. A rectangle represents a transition, that may have an inscription associated with it. This inscriptions may be either an attribution or a firing restriction. We will use the standard Language C notation for both attributions and firing restrictions. A double circle represents the termination place or *goal place*. Places and transitions are connected through arcs that may carry an expression.

The GSP of a *G-Net* G, denoted by $GSP(net\_name)$ in the ellipse of Figure 1, uniquely identifies the module. The rounded-corner rectangle in the GSP side contains a description of one or more methods, which specify the functions, operations or services defined by the net, and a set of attributes specifying the passive properties of the module (if any). The detailed structures and information flows of each method are defined by a modified high-level net in the internal structure. More specifically, a method defines the input parameters, the initial marking of the corresponding internal high-level net (the initial state of the execution). The collection of the methods and the attributes (if any) provides the abstraction or the external view of the module.

In the internal structure, places represent *primitives*, and transitions, together with arcs, represent connections or relations among the primitives. These primitives may be actions, predicates, data entities, and *instantiated switch places (isp's)*. A set of special places called *Goal Places* represents the final state of the execution, and the results (if any) to be returned.

A transition, together with arcs, defines the synchronization and coordinates the information transfer between its input and output places.

Given a *G-Net* G, an *isp* of G is denoted by $isp(G_{name}.mtd)$ (or simply $isp(G)$ if no ambiguity occurs), where G_{name} is the unique identification of G, and mtd is a defined method for G. An $isp(G_{name}.mtd)$ denotes an instantiation of the *G-Net* G, i.e., an instance of invocation of G based on the method mtd. Therefore, executing the *isp* primitive implies invoking G (by sending a token to G) based on the specified method. This token contains the parameters needed to define the tokens for the initial marking of the invoked net. This interaction between *G-Nets* can be compared to the mechanism of remote procedure call. The *isp* notation serves as the primary mechanism for specifying the connections or relationships between different *G-Nets* (modules). Embedding an *isp* of a lower level *G-Net* into the internal structure of a higher level *G-Net* specifies a hierarchical configuration. A formal introduction to *G-Nets* can be found in [7].

3 Logical Analysis

In this section we outline a modular logical analysis methodology for *G-Net* systems The modular logical analysis allows the composition of a system in a modular fashion. Therefore, providing mechanisms by which it is possible to consider different components of a model, based on a rigorous structure, allowing the designer to have a better control of the complexity of the system. Hence, different parts of the model might be independently considered. Moreover, analysis, reuse, and correction should be localized and performed at the component level, as long as the component interface remains unchanged. To be able to take advantage of the benefits of such a modular approach, a component must present two characteristics:

- The external view of the components must be loosely coupled, so that independence among components can be as high as possible and only a few well defined relationships are allowed.

- Externally a component must present a very high-level of functional cohesion, so that the role and the contribution of each component to the entire system are clearly defined.

G-Nets and *G-Net* systems possess the above characteristics. In order to take advantage of the above two general concepts, inherent to *G-Nets*, we have to define how *G-Nets* communicate with each other by means of a high-level protocol, which determines how *G-Nets* are connected. As discussed before, the *GSP* of a *G-Net* provides the abstraction with explicit definition of the methods (services) available to other nets. Also the *isp* and the *goal places* provide means by which a *G-Net* can be invoked and the processed result (if any) is returned. The general principles that must be satisfied when *G-Nets* are communicating is very similar to the client-server protocol, and consists of the following steps:

1. The requester *G-Net* requests a service.
2. The called *G-Net* accepts or not the requested service.
3. Upon acceptance of the service, the called *G-Net* attends the request and provides the results, otherwise the requester *G-Net* must issue another request.
4. The requester *G-Net* retrieves the result.

3.1 G-Net Systems Analysis

For analysis purpose a *G-Net* system will be considered as obtained by composing certain number of *G-Nets*. Having in mind the objective of avoiding state explosion, an evident method to analyze *G-Net systems* is decomposition. The intent is to verify properties of individual components, validate if these properties hold for the entire system, and use them to deduce additional properties of the system.

The methodology is a combination of behavioral and logical analysis. Behavioral analysis is applied to verify the local behavior. In this step we can either use invariant analysis or reachability tree analysis [13]. We use the reachability tree analysis for the reason that we can also extract from the reachability tree the external or observed behavior of a *G-Net*.

Logical analysis is based on the so called *assume/guarantee paradigm* for transition systems [19]. The objective is to verify properties of individual components, infer if these properties hold in the complete system, and use them to deduce additional properties of the

system. Furthermore, when verifying properties of the components, it may also be necessary to make assumptions about the behavior of the environment.

Assumptions and commitments may be given by temporal logic formulae. Generally, first order temporal logic is necessary to describe the services of a *G-Net*, as the realization of the methods are given by a modified *Predicate/Transition Net* [9], but we may separate the pure interaction (the protocol), which may be described using only propositional temporal logic, from the proper service specification dealing with the computation of data, where first order logic is necessary. Thus, in most cases, it is possible to separate the propositional part, describing the pure interaction, and the first order part describing the data transformations. Then the analysis or correctness verification can be done in two steps: first verifying the pure interaction, e.g. by applying a model checking procedure, and second verifying the correctness of the data transformations by the realization of the internal structure by applying well know analysis methodologies for *Predicate/Transition Nets* like reachability and invariant analysis.

To verify whether the implementation part satisfies the properties imposed in the interaction between a *G-Net* and its environment, we have to construct a model (in this case the an abstracted reachability graph) representing the implementation, in which the specification or the desired properties can be interpreted.

Our target is to consider software components modeled by means of *G-Nets*. We consider that these *G-Nets* are asynchronous components that communicates by means of synchronized actions as in CSP. Therefore, we assume that there is no dependency among the clock units of different components, i.e., there is no global clock.

4 Timing Analysis

For timing analysis we use *Fuzzy Time Petri Nets*, that are an extension of Petri nets which use a fuzzy approach to introduce time into Petri nets. The model is intended to serve as a tool which is suitable for modeling real-time systems and for computing selected performance indices, i.e., this model combines the ability to model real-time systems of the deterministic extensions and the ability to make performance evaluation of stochastic extensions. As

shown later, this is accomplished through the combination of time intervals and fuzzy set theory [8].

In the *Fuzzy Time Petri Net* model, tokens carry a fuzzy time function representing the possibility of there being a token in a given place from a reference set. Besides the fuzzy time function, two fuzzy intervals are associated with each transition. The enable interval E represents the minimum and maximum time that must elapse between the enabling of a transition and its firing time. The delay interval D represents the firing time, i.e., the delay necessary to execute the action represented by the corresponding transition. The existence of these time intervals associated with transitions is necessary in order to allow the modeling of two different timing concepts: time-outs and processing delays. Moreover, these two types of intervals are important when dealing with real-time systems. We can represent three general categories of timing constraints for real-time systems, that are: maximum timing constraints, minimum timing constraints and durational timing constraints [3].

The timing analysis is based on the fuzzy time function associated with the tokens. The idea is to use the fuzzy reachability graph, where we have the complete state space, in order to compute the $FTFs$.

The fuzzy reachability graph is a modified reachability graph which incorporates the fuzzy time concepts defined in the $FTPN$ model. The basic idea is to extend the concept of reachability graph to include the Fuzzy Time Function (FTF) carried by the token, which is computed after each transition firing, starting from the initial marking. Also, the modified reachability graph considers the fuzzy time intervals associated with the transitions. Thus, the fuzzy reachability graph is characterized by:

1) Assigning a FTF to each token on places in a given state.

2) Attaching to each edge of the graph the timing intervals associated with the corresponding firing transition representing the time restrictions when firing a transition.

4.1 Timing Analysis Issues

We assume that the FTFs associated with the initial tokens are known[1] and defined in the fuzzy reachability graph by the $FSET$ assigned to the root (initial state). Thus, from the initial state in the fuzzy reachability graph and the initial fuzzy time functions, we can compute the FTFs in the others states of the fuzzy reachability graph. Informally speaking, to compute these functions, it is necessary identify differences among the $FSET$s of immediate reachable states. Considering two connected states in the FRG and their respective $FSET$s, we have to consider two cases during the analysis:

1. common elements between $FSET$s.

2. different elements between $FSET$s.

Focusing on the dynamic behavior of a $FTPN$, item 1 indicates that tokens were not removed after firing a transition. Item 2 may represent two things: tokens removed from input places and tokens deposited in output places. According to the fuzzy function computation rule, input tokens are combined to determine the FTF of the output tokens.

4.2 Timing Analysis Using Fuzzy Time G-Nets

For timing analysis purpose, we assume that the internal structure of a *G-Net* will be a $FTPN$. When one *G-Net* invokes another *G-Net* with a given method m, this method defines the initial marking for the invoked net. Since the internal structure is a *Fuzzy Time Petri Net*, the token that arrives in an *isp* is a fuzzy token. Thus, the initial marking for the invoked net should be composed by fuzzy tokens. Once our main objective is the modular analysis, we have to have conditions to study each *G-Net* in isolation, i.e., the individual *G-Net* timing analysis must be independent of the analysis of the *G-Net* which is invoking it. Due to the commutativity and distributivity properties of fuzzy sets, it is possible to study each invoked *G-Net* considering that the tokens of the initial marking for the invoked net are not fuzzy. The results are then combined via the *isp*, i.e., a fuzzy interval that

[1]The initial FTF can be set to compute specific indices or may represent some different situations.

corresponds to the minimum and maximum execution time of the invoked net is computed by combining the enable and delay intervals attached to its transitions. The computation of this fuzzy interval resumes when a *goal place* is reached, indicating that the invocation finished. Finally, this fuzzy interval is combined with the fuzzy time function assigned to the *isp's* token. The analysis of the invoked net depends of the method used in the invocation.

In order to define the *Fuzzy Time G-Net* (*FTG-Net*), we consider the elements that take part in the interaction among *G-Nets*, that are the *isp* and the *GSP*, and the fuzzy time intervals, defined delays, associated with the firing of transitions.

5 Verification Environment and Application

In this section we introduce the verification environment developed for *G-Net* systems. The system was implemented in language C, and is portable to different machines. The actual implementation runs on DECStations and SPARCstations under UNIX. The interface was implemented based on the XWindows system.

In Figure 2 the block diagram of the verification system is shown. From this figure, we can identify the execution of two different kind of analysis: performance analysis and behavioral analysis.

As we show in Figure 2, the verification system has as input a *G-Net* specification, actually this is a text description as introduced by Chen in [2]. This text description is loaded by the reachability tree generator, and is used to generate the reachability tree. Based on this reachability tree and some additional information, we generate the reachability graph for the specified net. The reachability graph is then used to perform re-entrance verification, local structure and behavior verification, and interaction analysis. The performance analysis is performed based on the concepts previously introduced for *Fuzzy Time G-Nets Systems*, and the *fuzzy reachability graph* is derived from the reachability graph, generated in the logical analysis.

The approach introduced in this paper has been used to model two different kind of systems: a track vehicle transport system [15, 16] and for a flexible manufacturing systems [18].

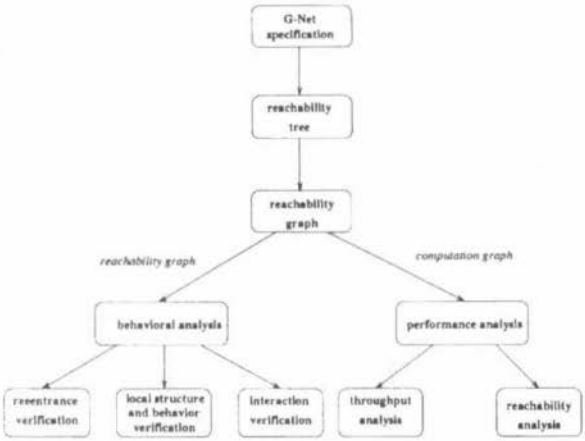

Figure 2: Block diagram of the verification system

In Figure 3 we show the screen-dump for the execution of the analysis procedure. Note that not all of the analysis are implemented. We show in this figure the transitive closure for the reachability graph. Based on the transitive closure we perform behavioral and structural verification of the internal net implementing each method. At the end of the window we present the places belonging to the interface of the net. With these information one can verify the external behavior of one *G-Net*, and can also, based on the reachability tree and reachability graph verify the internal behavior of the net. Details about the analysis procedure can be found in [14].

The screen-dump depicted in Figure 4, presents a screen with the analytical results obtained from the timing analysis algorithm as well as the corresponding *Timing G-Nets Interaction Graph (TGIG)* [6]. The *TGIG* is a 2D graph that depicts the timing and interaction aspects between different *G-Nets* in a *G-Net* system. The *TGIG* can be used as an alternative approach in order to determine some performance indices. From these two screens, we can see that the time chart reflects the results obtained by using the algorithm. For example, the FTF value associated with $isp(C.ms)$ can be derived from the *TGIG* by observing maximum values in the horizontal lines corresponding to the best and worst cases

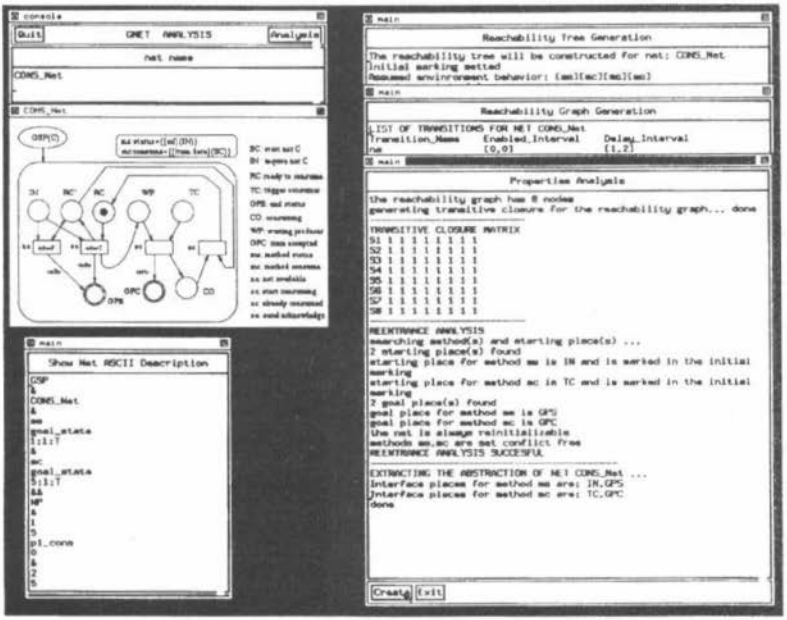

Figure 3: Screen dump for the consumer net showing the analysis

for *G-Net/method G(C).ms*.

6 Conclusion

In this paper we described a system for the modular analysis for a class of high-level Petri net named *G-Nets*. We described the motivation for such a modular analysis approach: the necessity to avoid the state explosion problem when analyzing a large complex software system.

The emphasis given on this paper is on the engineering aspects of the methodology. We believe that the marriage between formal methods and engineering approaches is the best way to face the difficulties associated with the design and verification of complex software

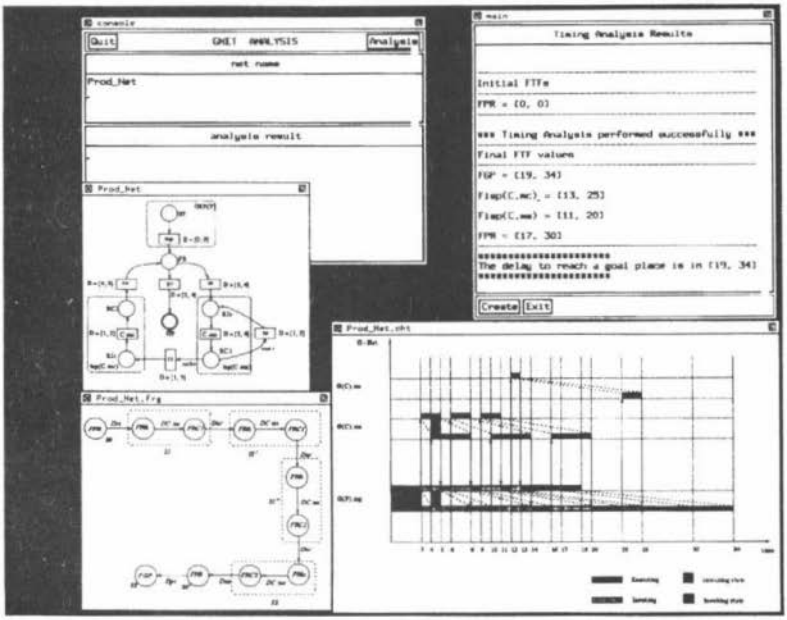

Figure 4: Screen-dump for the timing analysis of the producer

systems. The advantages of this convenient marriage are evident. From one aspect it is possible to share the methodological aspects inherent to an engineering approach for the design and verification of complex software systems. On the other hand, powerful formal methods are then available for the designer, so that early in the design phase of a system it is possible to verify whether the system possesses or not the specified properties.

Actually we are developing a graphical editor and animator to be coupled to a simulator for *G-Net* systems. Also we are working on the integration of the verification system presented in this paper with the editor/animator.

References

[1] S.K. Chang, A. Perkusich, J.C.A. de Figueiredo, B. Yu, and M.J. Ehrenberger. The design of real-time distributed information systems with object-oriented and fault-tolerant characteristics. In *Proc. of The Fifth International Conference on Software Engineering and Knowledge Engineering*, San Francisco, California, June 1993.

[2] T.C. Chen, Y. Deng, and S.K. Chang. A simulator for distributed systems using g-nets. In *Proceedings of 1992 Pittsburgh Simulation Conference*, Pittsburgh, PA, USA, May 1992.

[3] B. Dasarathy. Timing constraints of real-time systems: Constructs for expressing them, methods of validating them. *IEEE Transactions on Software Engineering*, 11(1):80 – 86, January 1985.

[4] J.C.A de Figeiredo, A. Perkusich, and S.K. Chang. Anticipated faults in real-time distributed systems. In *Proc. of The Seventh International Conference on Software Engineering and Knowledge Engineerinag, SEKE'95*, June 1995.

[5] J.C.A de Figueiredo and A. Perkusich. Fault tolerance in real-time distributed systems using petri nets extension. In *Proc. of 7th International Conference on Computing and Information, ICCI'95*, Trent University, Peterborough, Canada, July 1995.

[6] J.C.A. de Figueiredo, A. Perkusich, and S.K. Chang. Timing analysis of real-time software systems using fuzzy time petri nets. In *Proc. of The Sixth International Conference on Software Engineering and Knowledge Engineering*, pages 243–253, Riga, Latvia, June 1994.

[7] Y. Deng, S.K. Chang, J.C.A. de Figueiredo, and A. Perkusich. Integrating software engineering methods and petri nets for the specification and prototyping of complex software systems. In M. Ajmone Marsan, editor, *Application and Theory of Petri Nets 1993*, volume 691 of *Lecture Notes in Computer Science*, pages 206 – 223. Springer-Verlag, Chicago, USA, June 1993.

[8] D. Dubois and H. Prade. Processing fuzzy temporal knowledge. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(4):729–744, July 1989.

[9] H.J. Genrich. Predicate/Transition nets. In W. Brauer, W. Reisig, and G. Rozemberg, editors, *Petri Nets: Central Models and Their Properties*, volume 254 of *Lecture Notes in Computer Science*, pages 207–247. Springer-Verlag, 1987.

[10] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer Verlag, New York, NJ, 1992.

[11] P.M. Merlin and D.J. Farber. Recoverability of communication protocols - implications of a theoretical study. *IEEE Transactions on Communication*, COM-24(9):1036–1043, September 1976.

[12] M.K. Molloy. *On the Integration of Dealy and Throughput Measures in Distributed Processing Models*. PhD thesis, UCLA, 1981.

[13] T. Murata. Petri nets: Properties, analysis and applications. *Proc. of the IEEE*, 77(4):541–580, April 1989.

[14] A. Perkusich. *Analysis of G-Net Systems Based Upon Decomposition*. PhD thesis, Department of Electrical Engineering, Federal University of Paraiba, Campina Grande, PB, Brazil, August 1994.

[15] A. Perkusich and J.C.A de Figeiredo. Object oriented design of a track-vehicle system. In *Proc. of The Seventh International Conference on Software Engineering and Knowledge Engineerinag, SEKE'95*, June 1995.

[16] A. Perkusich and J.C.A. de Figeiredo. A petri net based approach to model objects for a trach-vehicle control system. In *Proc. of 7th International Conference on Computing and Information, ICCI'95*, Trent University, Peterborough, Canada, July 1995.

[17] A. Perkusich, J.C.A. de Figueiredo, and S.K Chang. Embedding fault-tolerant properties in the design of complex systems. *Journal of Systems and Software*, 2(25):23–37, 1994.

[18] M.L.B. Perkusich, A. Perkusich, and U. Schiel. Object-oriented real-time database design and hierarchical control systems. In *Proc. of International Workshop on Active and Real-Time Databases, ARTDB-95*, Skovde, SE, June 1995.

[19] A. Pnueli. In transition from global to modular temporal reasoning about programs. In K.R. Apt, editor, *Logics and Models of Concurrent Systems, NATO ASI series, Series F, Computer and Systems Sciences*, volume 13. Springer-Verlag, 1984.

[20] C. Ramchandani. Analysis of asynchronous concurrent systems by petri nets. Technical Report Project MAC-TR120, M.I.T., Cambridge, MA, 1974.

[21] J. Sifakis. Performance evaluation of systems using nets. In *Net Theory and Applications*, volume 84 of *Lecture Notes in Computer Science*. Springer-Verlag, 1980.

[22] L.A. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.