International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:2, No:2, 2008

Modeling and Analysis of a Cruise Control System

Anthony Spiteri Staines

***Abstract*—**This paper examines the modeling and analysis of a cruise control system using a Petri net based approach, task graphs, invariant analysis and behavioral properties. It shows how the structures used can be verified and optimized.

***Keywords*—**Software Engineering, Real Time Analysis and Design, Petri Nets, Task Graphs, Parallelism.

I. Introduction

CRITICAL systems [3]–[7], [12], [24] and embedded systems have special real time requirements. Various formalisms like logics, temporal logics, ASMs, calculus, CSP, CCS, algebras, process algebras, automata, and formal languages like Z, VDM, B, Haskell, LOTOS etc. have been created in the past decades to express different views of these systems and aspects of the design process. There are many diagrams and notations found in real time methods like UML-RT, JSD, DARTS, CODARTS, ROOM [3],[7] and the UML. These are useful to represent different aspects of dynamic behavior.

Formal methods definitely help towards producing better models because in the design process more thinking and reasoning is applied. Unfortunately many formal methods are specific to a particular issue e.g. CSPs and CCS are focused on component communication, Z is used to represent the system using schemas. Another problem is that most formal methods do not offer proper visualization. Some formal methods have limited CASE tool support. Formal representation can be difficult to understand, time consuming to produce and to amend.

Many diagrams and notations found in software methods E.g. UML communication diagrams, sequence diagrams, ROOMcharts are informal [1]–[2]. Research has been devoted to formalize these diagrams [11]–[13].

Petri nets [12]–[13],[17]–[19] are a convenient formalism for behavior modeling, experimentation, visualization and reasoning about real time system properties. Petri nets support concurrency, synchronization and resource sharing, both formally and diagrammatically [19],[21]. Both the structural and dynamic properties of Petri nets can be represented and analyzed mathematically [14]–[18], [21]–[22]. Petri nets have over three decades of coverage. There are several classes of Petri nets ranging from Elementary nets [9] to Object Oriented nets and Colored Petri nets [14]. Various CASE tools support Petri net modeling e.g. CPN Tool, HPsim, ExSpect [23], etc. Petri net structures can be supported and translated into other formalisms like automata and algebras. Often the simple properties of Petri nets are overlooked. These present detailed analysis methods as is discussed in this paper.

A. Spiteri Staines is an assistant lecturer at the Department of Computer Information Systems, Faculty of Science, University of Malta, Msida, MSD 2080, Malta, Europe (phone: 00356-21373402; fax: 21312110; e-mail: toni_staines@yahoo.com, tony.spiteri-staines@um.edu.mt).

II. Cruise Control

A. Basic Description

A typical cruise control system [5]-[6],[24] is composed of several components or classes interacting amongst one another in real time. Cruise control is subject to control law computations, with certain components having high computational requirements and redundancy issues. Communication between the 'actors' might require the support of adequate protocols and communication channels. Some parts of this system clearly exhibit closed loop highly cyclical behavior typical of certain types of controllers. In the cruise control user data and sensor data is read in. The input data is compared with the desired speed. This comparison is used to compute the output adjustment value for the actuator. The actuator automatically performs the adjustment. The RT system also functions as a controller. Two types of tasks can be identified i) periodic tasks and ii) user initiated tasks. In a system like cruise control these two types of tasks can be combined in a single process.

B. Simplified Algorithm

The text below explains the basic algorithm for the cruise control system. The algorithm is derived from the diagrams in section C and [24].

Set Timer to interrupt periodically in a period (T)
at each interrupt do
1) Sensor Scan process (
GPS,UI,Brake,Accel,Engine)
2) Get current speed
3) Compute control values
4) Update parameters
5) Send adjustment value to throttle
enddo

Steps 3 and 4 have the most significant time requirements. The reading in sensor inputs tasks 1 and 2 can be carried out in any order. This is because the final computation is based on these values. It is also possible to execute these tasks concurrently. Tasks 3 and 4 have precedence constraints requiring certain ordering. The current speed measured from the wheel rotation is compared with the desired speed and the data from the other sensors. The computed adjustment value is sent to the throttle actuator. Normally this would be i) reduce speed or ii) increment speed or iii) maintain current speed. Tasks 4 and 5 can be executed concurrently. This system behavior can be classified as deterministic, exhibiting a repeated pattern behavior. Control-law computations are involved. Sensor data is read to obtain accurate estimates of state variables to be monitored and controlled. Input values are used to compute an adjustment value. Part of the system can be scheduled differently.

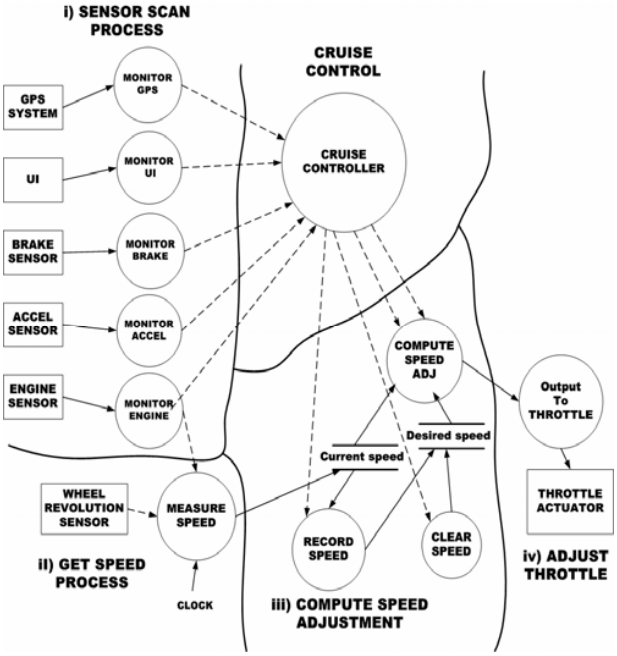

Fig. 1 Cruise System Control Flow DFD adapted from [20]

C. Diagrams

The diagram in Fig. 1 illustrates the main cruise control system events. This diagram is commonly used in RTSAD methods. Extensions to DFDs are used to add details for event flows and control transformations like discrete, continuous, triggered, enable/disable etc. The diagram has been partitioned into four main sections to illustrate the four main tasks. All activities are controlled and synchronized by the cruise controller.

Fig. 2 depicts the top-level network diagram for the cruise control using MASCOT notation [3]. This diagram can be used to obtain a full system template with bindings and interfaces e.g.

Server Disp out:Digital_out(ow=USER..op);

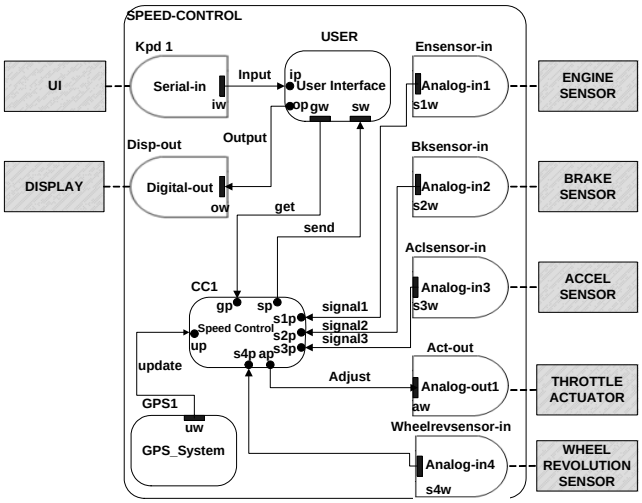

Fig. 2 Cruise System Network Diagram

Server Spsensor-in: Analog-in1(s1w=CC1.s1p); etc.

The subsystem components e.g. speed control, user interface, etc. can also be identified. Component coupling is rigorously enforced. The diagram can be decomposed further.

UML interaction diagrams can describe the communication processes for the cruise control.

D. Parallelism

Parallelism [21],[24] implies detecting computations that can be carried out in parallel. If more than one processor of the same type is available it is possible to carry out some of the tasks in parallel. Petri nets and task graphs expand this possibility.

III. Petri Net Modeling

A. Constructing the Petri Net

Constructing the Petri net is a simple process. The algorithm in section IIB is analyzed. The following steps are used: i) add a dummy source transition (node) at the top ii) add a dummy sink transition at the bottom (end) iii) the tasks in the algorithm are placed in sequence between the source and the sink node. Transitions for tasks that can be carried out in parallel are placed next to each other iv) Places are added to join the transitions. For parallel processing a fork point is used and a join point is used to joint the output of the concurrent tasks.

Practically speaking the Petri net represents the possible task execution sequence and it is similar to a task graph [8],[11],[21]. The Petri net is both a visual and formal executable specification that is easy to understand.

B. Initial Net

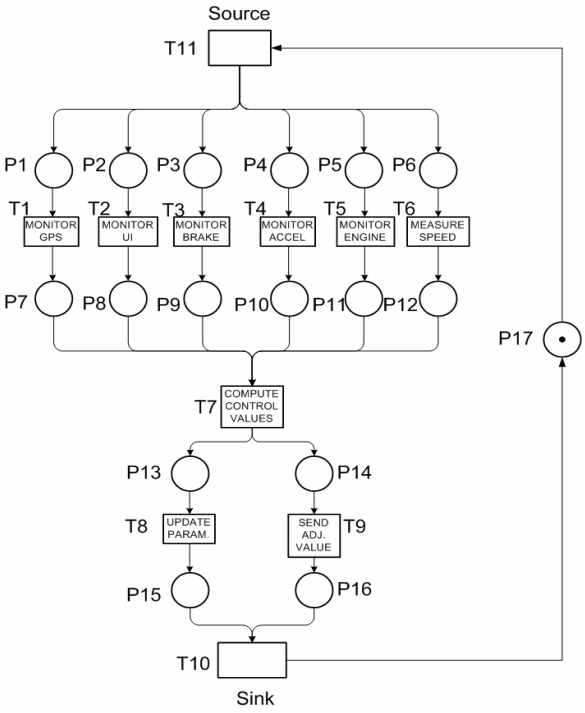

Fig. 3 Cruise Control Petri Net

C. Cruise Control Directed Graph

The cyclic directed graph similar to a task graph for the cruise control system is simply obtained by ignoring all the places in the Petri net and replacing the transitions with nodes. This is possible because each place holds exactly one token i.e. it is a 1-safe PN. This graph can be reduced into a DAG by removing e17.

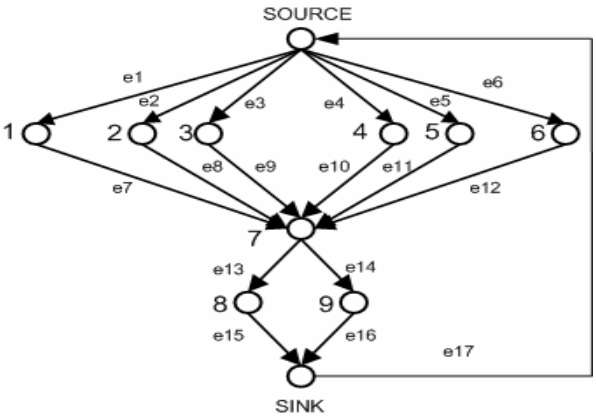

Fig. 4 Cruise Control Directed Graph

D. Redrawing the Graph for Two Processors

The vertices indicate which tasks can be carried out in parallel. It is evident that if all T1, T6 had to be executed completely in parallel six processors are required. Normally we can assume just two processors are available. The directed graph needs to be redesigned to reflect this. The problem is to find an optimum solution to redistribute/schedule concurrent tasks. The following algorithm can be used for this.

identify all critical tasks
identify all parallel tasks
Add tasks in order to a processor

If (critical task) then
If (time (P_a) = time (P_b)) add task to P_a or P_b
If (time (P_a) > time (P_b)) add task to P_a
If (time (P_a) < time (P_b)) add task to P_b
Set time for P_a , P_b = max time

If (parallel task) then
If time (P_a) + newtask < time (P_b) + newtask
Add task to P_a
If time (P_b) + newtask < time (P_a) + newtask
Add task to P_b
If (time (P_a) + newtask = time (P_b) + newtask add
task to P_a or P_b

If the given times for tasks are: T1,T6,T9 = 20ms, T2,T5 = 10ms, T3,T4 =15ms , T7= 40ms and T8= 25ms the result is as shown in Fig. 5 and 6.

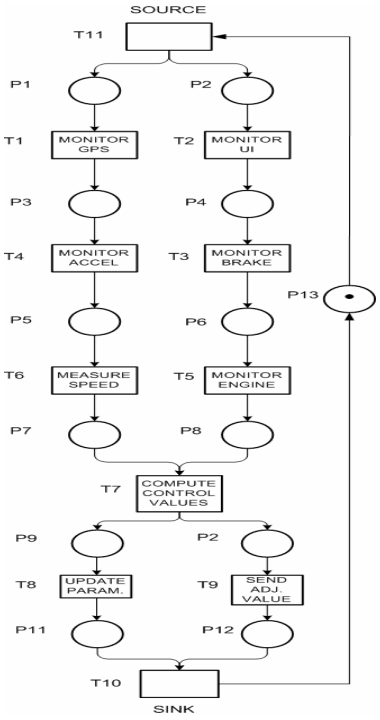

Fig. 5 Cruise Control Petri Net for Two Processors

In Fig. 6 the critical path or longest cycle is shown in bold. This is sequence of events on processor A, T1-T4-T6-T7-T8. The sequence of events on processor B is T2-T3-T5-T9.

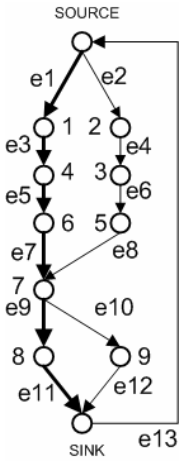

Fig. 6 Cruise Control Directed Graph for Two Processors

IV. Analysis

A. Incidence Matrix

An incidence, flow or change matrix C_{ij} is a special matrix representing the ordered input flows and output flows of the Petri net. For this matrix $1 \le i \le m, 1 \le j \le n$ and C = input flows – output flows. The incidence matrix is important for expressing basic structural properties of the net. If every row has some non zero values and for a row i, $\sum_{k=1}^{n} a_{jk} = 0$ this can indicate that each transition has exactly one input and output flow.

B. Invariants

There are several classes invariants [15]-[18]. Linear invariants are used here. A vector $v \in Z^m$ is by definition a P-invariant **iff** $v^t \cdot C = 0$ for a given Petri net. For the PN $v^t \cdot M = v^t \cdot M' + v^t.C.\vec{s}$, where M = initial marking, M' = next marking, C = incidence matrix and $\vec{s}$ =firing vector. $v^t \cdot M = v^t \cdot M'$ for all reachable markings denoting that the weighted sum of tokens remains constant or unchanged. A vector $y \in Z^m$ is by definition a T-invariant **iff** $Cy = 0$ for a given Petri net denoting a repetitive firing cycle.

Analyzing the Petri nets in Fig. 3 and Fig. 4 the following results are obtained. This data was obtained using the Dnanet Petri net tool.

```
1 1 0 0 0 0 0 0 0 0 0 0    (main.p1)      1 1 0 0    (main.p1)
0 0 1 0 0 0 0 1 0 0 0 0    (main.p2)      0 0 1 1    (main.p2)
0 0 0 1 0 0 0 0 1 0 0 0    (main.p3)      1 1 0 0    (main.p3)
0 0 0 0 1 0 0 0 0 1 0 0    (main.p4)      0 0 1 1    (main.p4)
0 0 0 0 0 1 0 0 0 0 1 0    (main.p5)      1 1 0 0    (main.p5)
0 0 0 0 0 0 1 0 0 0 0 1    (main.p6)      0 0 1 1    (main.p6)
1 1 0 0 0 0 0 0 0 0 0 0    (main.p7)      1 1 0 0    (main.p7)
0 0 1 0 0 0 0 1 0 0 0 0    (main.p8)      0 0 1 1    (main.p8)
0 0 0 1 0 0 0 0 1 0 0 0    (main.p9)      1 0 1 0    (main.p9)
0 0 0 0 1 0 0 0 0 1 0 0    (main.p10)     0 1 0 1    (main.p10)
0 0 0 0 0 1 0 0 0 0 1 0    (main.p11)     0 1 0 1    (main.p11)
0 0 0 0 0 0 1 0 0 0 0 1    (main.p12)     1 1 1 1    (main.p12)
1 0 1 1 1 1 1 0 0 0 0 0    (main.p13)     1 0 1 0    (main.p13)
0 1 0 0 0 0 0 1 1 1 1 1    (main.p14)
1 0 1 1 1 1 1 0 0 0 0 0    (main.p15)
0 1 0 0 0 0 0 1 1 1 1 1    (main.p16)
1 1 1 1 1 1 1 1 1 1 1 1    (main.p17)
```

Fig. 7 Place Invariants for fig.3 & fig. 5

```
1    (main.t1)
1    (main.t2)
1    (main.t3)
1    (main.t4)
1    (main.t5)
1    (main.t6)
1    (main.t7)
1    (main.t8)
1    (main.t9)
1    (main.t10)
1    (main.t11)
```

Fig. 8 Transition Invariants for fig.3 & fig. 5.

C. Other Behavioral and Structural Properties

The reachability tree or marking graph can be used for constructing the reachability tree and testing for deadlock.

Invariants can be used for further analysis. i) Bounded and conservative behavior is denoted by $\exists v > 0, v^T C = 0$, ii) Repetitive behavior by $\exists y > 0, Cy \ge 0$ and iii) Consistent behavior by $\exists y > 0, Cy = 0$. Other issues like home states, cyclic behavior, deadlock , boundedness can be properly interpreted from the invariant results. Petri net test suites and CASE tools like Dnanet, etc. can be used for further checking. The results of the reachability trees and invariants are presented in Table I & II.

TABLE I
Cruise Control Petri Net Reachability & Invariant Comparison

PETRI NET	REACHABILITY MARKING GRAPH	CONNECTED COMPONENTS	T- INVARIANTS	P-INVARIANTS
Fig. 3	69 unique markings	1 strongly	identical	not identical
Fig. 5	21 unique markings	1 strongly	identical	not identical

TABLE II
Cruise Control Petri Net Behavior Comparison

PETRI NET	DEADLOCK POSSIBLE	BOUNDED	CYCLIC BEHAVIOUR	HOME STATES
Fig. 3	NO	YES	YES	YES
Fig. 5	NO	YES	YES	YES

V. Interpretation of Results

The Petri net models for the cruise control system have been successfully validated. They are both conflict free, deadlock free and do not have unwanted states. The most strongly connected component is T7. This task is the most critical and important task.

The transition invariant analysis shows that even though the

algorithm and firing cycle is modified, the basic properties and execution remain unchanged. I.e. both models are formally correct and valid.

The structures have a relatively small reachability tree. The two processors Petri net in Fig. 5 have only 21 unique markings compared with the 69 of Fig. 4. This is indicative that if more processors are introduced the overall system state space is increased and becomes more difficult to handle. More synchronization overhead is necessary to coordinate and control process communication. On the other hand reducing the parallel tasks in the system the complexity is reduced so there is less switching overhead.

The number of parallel places in the Petri net or the concurrent tasks in the directed task graph indicate the total number of processors required to execute those tasks in parallel.

One of difficulty in parallel systems is load balancing. The graphical result for the Petri net in Fig. 5 and task graph in fig. 6 is shown in Fig. 9. This is obtained from the algorithm in section III D. Processor I is fully utilized and processor II has a utilization of approximately 46 % only. Adding another processor can reduce the time of processor I, but the task like *compute control values* has precedence constraints. Hence the other initial tasks must have completed.

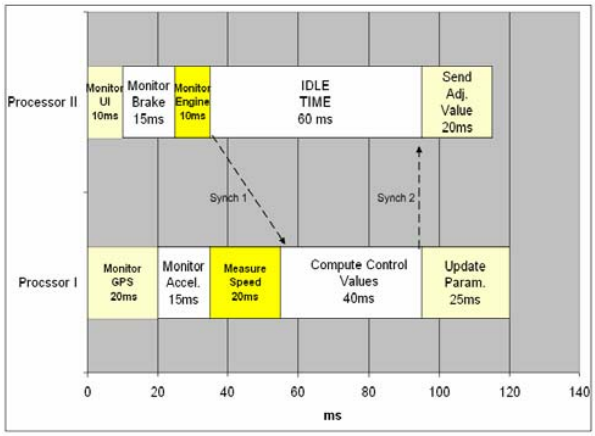

Fig. 9 Two Processor Task Scheduling

VI. Conclusion

A cruise control system has been explained and analyzed using simple Petri nets, directed task graphs and Petri net related theory. Petri net theory can be used for model optimization and proving the correctness of models used for real time and critical systems. Useful properties like invariants ,reachability tree, etc. can be easily obtained for deterministic systems.

More detailed analysis based on graph theory, other Petri net properties and simulation techniques should be considered.

Colored Petri nets and Higher order nets can provide us with other possibilities.

The two processor Petri net model can be translated into LLD ladder logic diagrams which are useful for PLC programming. The diagrams in Fig. 4 and 6 qualify for the automatic generation of methods [10].

References

[1] S. Bennett, J. Skelton, K. Lunn, *UML*. Schaums Outline 2nd ed., New York: McGraw-Hill, 2005, pp. 5–18.
[2] P. Roques, *UML in Practice*. UK: Wiley, 2005, ch. 1. & ch. 2.
[3] J.E. Cooling, *Software Design for Real-Time Systems*, Chapman & Hall, London, 1995 ch. 10.
[4] R. Williams, *Real-Time Systems Development*. UK: ELSEVIER, 2006, ch. 11.
[5] H. Gomaa, *Software Design Methods for Concurrent and Real-Time Systems*, Addison-Wesley, 2001, ch. 1-13,19,23.
[6] H. Gomaa, *Designing Concurrent, Distributed, and Real-Time Applications with UML*, Addison-Wesley, 2001, ch. 2.
[7] G.P. Mullery, "CORE - A Method for Controlled Requirement Specification", *Proceedings of the 4th international conference on Software engineering*, Munich Germany 1979 , pp.126 – 135.
[8] Y. Abdeddaim, A. Kerbaa, O. Maler, " Task Graph Scheduling using Timed Automata", *IEEE Parallel and Distributed Processing Symposium*, Apr 2003.
[9] J. Brusey, D. McFarlane, "Designing Communication Protocols for Holonic Control Devices using Elementary Nets", *LNCS 0302-9743 Volume 3593/2005*, Aug 2005, pp. 76-86.
[10] K. Maruyama, "Automated Method-Extraction Refactoring by Using Block-Based Slicing", *ACM Software Engineering Notes*, Vol 26 no 3, May 2001, pp.31-40.
[11] J.A. Saldhana, S.M. Shatz Z. Hu, "Formalization of Object Behavior and Interactions From UML Models", *International Journal of Software Engineering and Knowledge Engineering IJSEKE*, Vol. 11 No 6., Dec 2001, pp. 643-673.
[12] L.A. Cortes, P. Eles, Z. Peng, "A Petri Net based Model for Heterogeneous Embedded Systems", *NORCHIP Conference*, 1999, pp. 248-255.
[13] T. Gehrke, U. Goltz, H. Wehrheim, "The Dynamic Models of UML: Towards a Semantics and its Application in the Development Process", *Technical Report Informatik-Bericht 11/98*, University of Hildesheim, Germany, 1998.
[14] K. Jensen, G. Rosenberg, High-Level Petri Nets: Theory and Application , Springer – Verlag, Berlin, 1991.
[15] S. Sankaranarayana, H. Simpa, Z. Manna, " Petri Net Analysis using Invariant Generation",*LNCS Vol, 2772 – Springer Verlag* ,ISSN: 0302-9743, 2004, pp. 682-701.
[16] R. Clariso, E. Rodriguez-Carbonell, J. Cortadella, "Derivation of Non-structutal Invariants of Petri Nets using Abstract Interpretation", *ICATPN LNCS*, Vol. 3536- Springer Verlag, 2005, pp. 188-207.
[17] K.M. Van Hee, *Information Systems Engineering A Formal Approach*, University Press, Cambridge, 1994, pp. 237-240.
[18] M. Zhou, K. Venkatesk, *Modeling, Simulation and Control of Flexible Manufacturing Systems- A Petri Net Approach*, World-Scientific Publishing, N.J. ,1999.
[19] J. Desel, E. Kindler, "Petri Nets and Components extending the DAWN approach", *D. Moldt (ed.): Workshop on Modelling of Objects, Components, and Agents*., Aarhus Denmark, Aug 2001.
[20] J. Kramer, J. Magee, "Exposing the Skeleton in the Coordination Closet", *Proceedings of the Second International Conference on Coordination Languages and Models*, 1997,pp. 18-31.
[21] Z. Hanzalek, "Parallel Algorithms for Distributed Control - A Petri Net Based Approach", PhD. thesis, Prague 1997, ch 2-6.
[22] K. Yamalidou, J Moody, M. Lemmon, P. Antsaklis, "Feedback Control of Petri Nets Based on Place Invariants", *Technical Report of the ISIS Group University of Notre Dame IN 46556*, ISIS-94-002, 1994.
[23] Exspect Tool, Technische Universiteit, Eindhoven.
[24] J.W.S. Liu, *Real-Time Systems*, Pretence Hall, NJ, 2000.