7. DESIGN METHODS of REAL-TIME SYSTEMS

The following tasks must be performed in order to describe the functional and temporal behaviour of physical systems and computer programs:

1. functional and structural specification and verification
   - to define functional and structural properties of a real-time system,
   - to prove that the functional and structural specification is correct,
2. temporal specification and verification
   - to define temporal properties of the real-time system,
   - to prove that the temporal specification is correct,

Real-time systems may have "hard" real-time constraints. These systems response to external event. A response consists of series of activities that are described by their start times, execution times and deadlines (i.e. time constraints) in time domain. These activities are performed by internal processes of real-time systems. Activities, i.e. processes that are executing them, must be scheduled according to time requirements of real-time systems to provide responses to external events at the right time.

To design real-time systems, their time constraints must be specified and verified and their system performance must be also checked. Correctness and performance are two major concerns in the design of real-time systems. Real-time system is said to be correct when it satisfies a given functional and temporal specification. This relation must be established using an abstract model of either a physical system or a programming language. Correctness and performance are closely related in real-time systems. It may be incorrect to produce both the wrong results at the right time and/or the right result at the wrong time. First, time constraints of real-time systems are given in the specification phase. Their correctness is examined during the verification phase, while performance is studied in the scheduling phase. The main tasks of these three design phases are as follows:

specification = defines the set of functional and time constraints imposed on the behaviour of the real-time system,

verification = (safety analysis) - proves that the specification describes correctly the behaviour of the real-time system, in other words if any correct implementation satisfies the requirements the specification is safe (nothing bad will happen)

performance analysis = (schedulability analysis) - examines whether the hardware and/or software meets both the functional and time specification.

Many formal methods were developed to describe and analyse real-time systems including control, diagnosis and computer systems. They can be divided into descriptive and operational formalisms. The descriptive formalism specifies system properties, while the operational formalism analyses correctness of specification.

7.1 Specification Methods

The task of specification is to define a class of solution that will satisfy the requirements. Therefore it must define activities to be performed by the real-time systems and time constraints that apply to them. The specification methods describe set of properties, of real-time systems, including functional, structural and temporal ones. Real-time systems can be described at several levels: for example, in terms of their external requirements, their formal specification, their implementation in programming languages and their execution on some particular system. For any kind of specification, there will be a set of properties that must be established during the specification. The first step in the specification is to define the

requirements of the environment. These requirements can be given by a set of relations between the external events and the necessary responses, and the distribution of events and responses in time. Once this information is available the formal specification of the requirements of real-time systems can be developed.

7.3.1 Descriptive Formalism

Denotational semantics method. Two specification languages based on denotational semantics were developed: real-time Communicating Sequential Processes (CSP), [Koy88]; and real-time Communicating Shared Resources (CSR), [Ger91], language. The CSR is a further developed variant of real-time CSP that handles shared resources. Both they describe functional, structural and temporal requirements of real-time systems. Basic elements of the computational model of these languages are events and activities; processes and resources that are used to describe and specify behaviour of real-time systems. Execution of processes in these methods is based on a maximal parallelism model. Their semantic domain consists of state-history pairs. The state is a function of states of activities and events. History is record of a sequence of activities, which lead to the state associated with the history. To handle the time both approaches were extended with real-time constructs as for example time, aperiodic and periodic processes, time-out, deadline, exception handling, etc. They describe time in a global time domain and a discrete time scale.

Real-time logic method [Del93], [Jah86], [Mok91]. It is a formal method that describes behaviour of real-time systems. It is based on an event-action model that has the following elements: action (activity that uses system resources), event (temporal marker of action), state predicate (assertion about the state of the system), causality relations (dependencies amongst actions), time constraints (assertions on time of events). The event-action model captures control, data, and temporal dependencies between actions. It uses a predefined set of events and a set of primitive actions, taking into account the following assumptions: all primitive actions terminate and their execution times are known. Each action has two associated events ; the initiation event and the termination event. Having this model the data and control dependencies, and the temporal ordering of actions can be expressed by state predicates. The state predicates describe values of Boolean system attributes during intervals (represented by pairs of event occurrences). Events are central to RTL because actions and state predicates are transformed into events. The system specification given by the event-action model is transformed into a set of Real Time Logic - RTL - formulas. The transformation coverts assertions about the physical states over time into algebraic relations.

Temporal logic method [All83], [Cla86], [Ion95], [Lam94], [Pnu88]. This method represents each activity with a time interval during that the activity takes place. It describes temporal behaviour of systems as relationship between temporal intervals using constraint propagation. Relation of any two intervals can be expressed by several interval relations. Relationships between intervals are described by a network whose nodes are individual intervals. The network represents the information about the relation of intervals. Reference intervals describe the temporal hierarchy in the network. They group together intervals whose temporal constraints between each pair of intervals are defined. Every interval defines one or more reference intervals that belongs to. If two intervals are not explicitly related to each other, a relationship between them can be described by finding a path between them by searching reference hierarchy until paths between the nodes is not found. The temporal logic method defines system behaviour by an infinite sequence of states. Every state is associated with the time that it entered at. Thus, all states have their time intervals that define their temporal behaviour and properties.

7.2. Verification Methods

The implementation of physical systems and computer programs must satisfy their specification. They must meet both functional, structural and temporal requirements. The verification of temporal properties involves determining the time of completion of activities in real-time systems. This in turn may depend on the availability of systems and their resources. In real-time systems resources are limited, thus the time required for an activity or a computation can depend on the availability of resources. As a result, it requires more complex verification procedure than "non" real-time systems. There are two approaches to verify the specification. In the first one, a system will satisfy its specification if it is derived from the specification by formal transformation. In the second one, it must be proved that properties of the system meet the specification. A property is valid if there is no occurrence function that can satisfy the negation of the property under investigation in conduction with the assertions in the system specification. If the validity of a desired property cannot be established from the specification then some practical constraints must be imposed on the system to achieve the desired result.

7.2.1 Operational Formalism

First order logic method [Jah86], [Mok91], [Seb94], [Zuc95]. This formal method evaluates specification (i.e. it executes safety analysis) and performance of real-time systems. It proves the consistency of system specification that is based on formulas developed by Real-Time Logic descriptive formalism. It analyses the safety properties of the real-time system. The specification contains a set of primitive and composite activities, a set of known events, a set of state predicates and the time constraints of activities. The activities interact in a simple way so that a partial order of the primitive activities of a system is easily derived from the description. Formulas reason about occurrences of activities represented by event constants. They are analysed to assert the occurrence of events in the system because the occurrence function maps the occurrence of the event into the time of that occurrence. Each event is described by an occurrence function. This function represents the time of an occurrence of the event. Time constraints are represented by restrictions imposed on occurrence functions. A given safety property holds if there is no mapping of event occurrences in time domain that is consistent with the negation of the safety property and the specified restrictions on the occurrence function.

7.2.2. Dual-Language Methods

State machine based method [Das85], [Sha89]. This method models the real-time system as a finite state machine whose behaviour is defined by activities and states of the system. The machine moves from being in one state to being in another state as a result of some external event. The response of the system is determined by its current state and the event of the system. While the machine is moving from one state to another, it may carry out a sequence of activities. The state of a machine is often represented graphically by a state transition diagram. Real-time requirements of the system are defined by behavioural and performance constraints, which are time constraints. This method classifies three types of restrictions on time constraints: maximum, minimum and duration constraints. The time constraints express the minimum or maximum time between the occurrence of activities and responses of the system. The activity-state table method was developed to implement finite state virtual machine that specifies and verifies real-time systems. It contains two types of tables: activity and state ones. It produces a small general-purpose program that defines its activities by looking up instructions in data tables. The activity table determines all activities with their next states. To model real-time systems as state machines, it is frequently necessary to have more than one possible next state after the occurrence of an event. To solve this problem state table includes all possible states of the machine in basic and auxiliary tables. Once these tables are created the next step is to construct a program or a system that simulates the activity of the state machine.

Petri net based method [Mer76], [Ram80]. This approach includes both descriptive and operational formalisms. It represents activities in systems by places and transitions. Places represent states or conditions of activities. They would contain tokens to describe dynamics of systems. Transitions describe events between states and conditions. Activities in systems are performed by firing transitions. Firing would remove some tokens from their input places and put some tokens to their output places. States of places determine when particular transitions can fire. In the "basic" Petri net no facilities are provided to model system behaviour in time domain, i.e. when and how long activities take place. To specify and verify real-time systems the "basic" Petri net was augmented by time constraints (maximum, minimum and/or duration times). Time is associated with places or transitions or both. The time would represent the duration of certain activities, e.g. the time interval when a transition starts firing and when tokens are available at the output place or the time interval when a token has appeared at a place and when it is available to the output transition. Both the specification and verification procedure of real-time systems is divided into two steps: behavioural and temporal one. Specification starts with description of functional and structural behaviour of a system using a "basic" Petri net modelling approach. It continues with definition of temporal properties. The "basic" Petri net model is transformed into "timed" Petri net by adding time constraints to its elements. In verification phase the functional property analysis is performed to assert the correctness of the functionality. It can be done by using existing analysis methods of Petri nets. Next, the analysis of temporal properties is performed to evaluate the correctness of behaviour in time domain. The system specification is valid if the final state is reachable from the initial state, i.e. all places are schedulable and transitions are fireable in the firing sequence from the initial marking (state)to the final marking (state).

Process algebra based method [Ace96], [Bus94], [Koy88], [Lee94]. Process algebra consists of a set of operators and syntactic rules for constructing processes, a semantic mapping with assigns meaning or interpretation to processes, a notion of equivalence or partial order between processes, a set of algebraic laws that allows syntactic manipulation of processes. Basic operators in process algebra are prefix operators for sequencing of processes, parallel operator for composing two processes run in parallel, restriction or hiding operator for abstracting the details or names of actions, relabeling or renaming operator, recursion operator for describing processes with infinite behaviour. Its real-time version captures the time at which event occurs. They are associated with time intervals. It supports some operators that deal with time for example: delay execution, time-out, bound the time. The execution of processes is described by a labelled transition system in process algebra. There is also a finite set of transition rules which depicts the stepwise behaviour of the process. There are two types of actions in process algebra: timed action, those which consume time, and instantaneous events those that are instantaneous. They have separate priority orderings. Verification in process algebra proves that one process is equivalent to another process. The algebraic laws of the process algebra allow the transformation of one process into another equivalent one. These laws are based on a notion of equivalence between processes. This equivalence is based on the concept of either bisimulation or strong bisimulation.

Constrained expression method [Avr91], [Dil88]. This formal method translates the system description into formal representation called constrained expression representation. Next, it evaluates constrained expressions defined in the specification phase. This approach handles the system as a sequence of events. It defines occurrence of events and the time between their occurrences. The method associates an event symbol with each event to describe its possible behaviour. The event symbol is a string over the alphabet of event symbols. The behaviour of the whole system is described by the total orders of events. This method system defines expressions that describe the behaviour of the system. These expressions include strings representing all possible sequence of system behaviour. Among them may be strings that do not represent any possible behaviour. These strings are removed by constraint expressions.

After elaborating expressions that specify system behaviour the next task is to search for occurrence of event symbols and sequences of events in a system string that describe possible sequences of system behaviour. Having these expressions, inequalities, which involve numbers of occurrences of event symbols in system string, are elaborated. If the expressions are inconsistent then there are event symbols or sequences of event symbols that do not represent the system behaviour. To analyse temporal properties of the system this method is extended by assigning time values to each event. This extension enables to define the upper bound of time between two events.

Assertional reasoning method [Att91], [Sha87], [Tel88]. It proves temporal and safety properties of the system. The real-time system is described by a timed automation model. This model defines the dependency requirements and time constraints of the system. The timed automation model includes an I/O automation and time assumptions on the system behaviour. Time assumptions define upper and lower time bounds while events must be executed or states must be appeared. The I/O automation and the time assumptions generate a set of execution sequences and a corresponding set of behaviour in time domain. Time constraints of the system, represented by timed automation model, are given by a set of time conditions. If given a timed automation model and a set of time requirements, then a requirement automation model is developed to describe the first and the last time values that events can occur at. These time values are included in the state space of the requirement automation model. The timed automation model satisfies the time requirements if it behaves similarly to requirement automation model. This is proved by an invariant assertion technique that maps timed automation model to requirement automation model. If the timed automation model can be mapped into the corresponding requirement model then the specification is correct.

7.2.3. Evaluation of Specification and Verification Methods

Descriptional formalism. *CSP and CSR* are compositional languages (or syntax-directed) that is very important for specification. They support description of composite processes using specification of its components without referring to their internal details. The information of the specification can be partitioned into two parts: observable and non-observable entities. Both methods are more suitable to specify computer systems than control ones. They have a common drawback. They have no graphical support to represent events. *Real-Time Logic* provides a uniform way to express different types of constraints on a system by means of algebraic relations on event occurrences. The even-action computation model enables specification the temporal behaviour of a system without having to consider the formal specification. Temporal logic is particularly suited to deal with temporal behaviour and evolutionary aspects of real-time systems. It is a common formalism to specify and derive properties of real-time systems. It provides precise and concise means of representing real-time systems before their implementation. *Temporal logic* have many advantages, especially abstraction and flexibility of the specification process. However, there are number of limitations that restrict the use of this method for development of real-time systems. In particular, formal notations cannot easily be used to co-ordinate the activities of human factors. It also fails to directly characterise situations easily modelled by model-based formalisms, such as "local" properties of execution sequences.

Operational formalism. *First order logic* allows designers to make time constraint assertions about the system. To overcome problem of verification of large and complex systems it either filters out the irrelevant assertions from the system specification or uses subclasses of formulas that are not too restrictive in describing the system specification. It is more suitable for analysing specification of real-time systems than computer programs.

Dual-languages approaches. The *state machine* method minimises the complexity of the design and implementation of real-time systems. The control structure of many real-time systems seems to be naturally given by state machines. However, the state transitions are

governed not only by the state of the system and the occurrences of the event, but also by their time of occurrence. The major advantage of this approach is that a particular system can be altered simply by changing the contents of tables. State machines may be hierarchically and modularly composed, with state transitions triggering complicated interactions among the state machine components. It has one main drawback; the exponential state explosion modelling complex systems. The *process algebra* supports the modular specification and verification of real-time systems due to its algebraic laws and proof system. It is possible to verify the whole system by reasoning about its parts. Hiding or restriction operator allows not to take into account unnecessary details. It also supports the substitution of one component with another equal component in the system. Thus, a complex system can be decomposed into simple subsystems. The transition systems incorporate time information, and this enables to express concurrency: merely adding timing to transition systems. Moreover, this method provides a complete axiomatisation of bisimulation for algebra; a result that is often problematic in a timed setting. Another advantage of incorporating temporal information into the calculus is that it allows a particularly simple definition of action refinement. *Constrained expression method* is mainly used for performing automated analysis of logical properties of a range of concurrent systems. It decomposes analysis problems into smaller parts in order to get appropriate results for the full systems. The *assertional method* can handle systems of arbitrary complexity. Mapping techniques used by this method are only capable of proving safety properties, but not liveness. This is a drawback because temporal properties integrate both safety and liveness. The lower bound is regarded as a safety property. In contrast, the upper bound is connected to both liveness (i.e. the designated amounts of time elapse) and safety (i.e. time cannot elapse without the event having occurred).

7.3 The Petri Net

Activities in real-time systems are associated with information, internal/external events, operations, resources and time. In Petri nets they are described by their conditions, functions and sequence of occurrence. Events are either external events, which take place outside systems, or internal events, which occur in systems. Both external and internal events may change the state of real-time systems. Conditions and events define system states. An operation occurs if set of conditions, which are called preconditions, are met. Preconditions change after occurring an activity and another set of conditions, which are called post conditions, become valid. In Petri net activities are modelled by transitions and conditions and execution of activities by tokens held by places. Preconditions are inputs to transitions and post conditions are outputs of transitions. The time in "basic" Petri net is not included explicitly. Instead, it is defined implicitly by sequence of occurrence of activities. Operation in real-time systems corresponds to a sequence of discrete activities in Petri nets. The occurrence of an activity is modelled by firing of transition(s). If any time more than one transition is enabled then any of them may be the next to fire. When a transition fires it removes tokens representing preconditions from its input place(s) and creates new tokens that define post conditions. It puts one token to each output place. The firing of a transition is an instantaneous activity that takes zero time. Activities modelled by Petri nets are primitive ones, that are instantaneous and non-simultaneous.

7.3.1 Definition of Petri Net

According to Peterson [Pet81] the Petri net is given by a four-tuple:

$$PN=(P,T,I,O) \tag{7.3.1}$$

where

$$P = \{p_1, p_2, \ldots, p_m\} \tag{7.3.2}$$

- finite set of places

where i=1..m

$$T = \{t_1, t_2, ..., t_n\} \quad (7.3.3)$$

- finite set of transitions

where j=1.. n

$$I(p_i): T^{\infty} \rightarrow P \quad (7.3.4)$$

- input function of place

$$O(p_i): P \rightarrow T^{\infty} \quad (7.3.5)$$

- output function of place

$$I(t_j): P^{\infty} \rightarrow T \quad (7.3.6)$$

- input function of transition

$$O(t_j): T \rightarrow P^{\infty} \quad (7.3.7)$$

- output function of transition

Places are represented by circles and transitions by bars in Petri nets. There is an example of graphical representation of Petri net in Figure 7.3.1. The input function of transition $I(t_j)$ maps from bags of places p_i to a transition t_j The place p_i is an input place of a transition t_j if $p_i \in I(t_j)$. The output function of transition $O(t_j)$ maps a transition t_j to bags of places p_i. The place p_i is an output place if $p_i \in O(t_j)$. The bags of places are the inputs and outputs of a transition. A bag is a generalisation of a set that allows multiple occurrences of an element in a set.

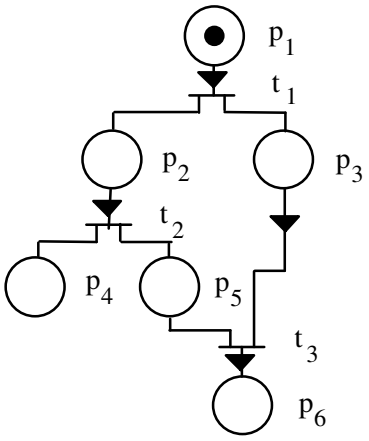

Figure 7.3.1 Petri net

7.3.2 Petri Net Graph

The Petri net graph has arcs that are directed from an element of one set to an element of other set, for example: from a place to a transition. In Petri net graph directed arrows connect places to transitions. An arc directed from a place p_i to a transition t_j defines the place to be an input to the transition. An output place is indicated by an arc from the transition t_j to the place p_i. A Petri net graph G is bipartite directed multigraph and given by:

$$G=(V,A) \quad (7.3.8)$$

where

$V = \{v_1, v_2, \ldots v_p\}$ - set of vertices

$A = \{a_1, a_2, \ldots a_r\}$ - bag of directed arcs,

where $a_j = (v_k, v_j\}$ and $v_k, v_j \in V$

i=1..p - number of vertices

j=1..r - number of arcs

The set V can be partitioned into two disjoint sets of places P and transition T. If a directed arc $a_j \in A$ if $a_j = (v_k, v_l)$ belongs to A then either $v_k \in P$ and $v_l \in T$ or $v_k \in T$ and $v_l \in P$.

7.3.3 Marking of Petri Net

Marking is an assignment of tokens to places in Petri net. A marked Petri net is defined by the following five-tuple:

$$PN=(P,T,I,O,M_0) \tag{7.3.9}$$

where - M_0 is the initial marking.

In the Petri net the tokens are represented by small dots in places. The marking $m_k(p_i)$ defines the number of tokens assigned to a place p_i at a moment of time k. It is a function from this place to the non-negative integers

$$m_k(p_i): p_i \rightarrow n_{ik} \tag{7.3.10}$$

The marking of the Petri net at a time instant k is represented by vector of markings M_k which is column vector whose i-th component is the marking of place p_i at time k.

$$M_k = [m_k(p_1), \ldots m_k(p_i), \ldots m_k(p_m)]^T \tag{7.3.11}$$

The marking of Petri net M_k is a function from the bag of places P to the non-negative integers N that is defined by

$$M_k: P -> N_k \tag{7.3.12}$$

7.3.4. State and Execution of Petri Net

The state of the Petri net is defined by its marking. The state space of the Petri net with m places is the set of all markings N^m. The firing of a transition represents a change in the state of the Petri net and is defined by a next-state function:

$$\delta : N^m * T \rightarrow N^m \tag{7.3.14}$$

The next-state function for a marked Petri net with a marking M_k and a transition $t_j \in T$ is defined if and only if $m_k(p_i) \geq \#(p_i, I(t_j)$ for all $p_i \in P$. If the next-state function of a Petri net is given as $\delta(M_k, t_j) = M_1$, the next-state of place p_i is defined by (7.3.14) for all $p_i \in P$. If given a Petri net with an initial marking M_0, it can be executed by successive firings. Firing an enabled transition t_j from initial marking produces a new marking $M_1 = \delta(M_0, t_j)$. This can continue as long as there is at least one enabled transition in each marking. If we reach a marking that no transition is enabled in, then no transition can fire. As a result, the next-state function will be undefined and the execution will stop. The static behaviour of the Petri net is described algebraically by incidence matrix W that is associated with the structure of the Petri net:

$$W = \begin{bmatrix} a_{11} & . & . & a_{1n} \\ a_{21} & . & . & a_{2n} \\ . & . & . & . \\ a_{m1} & . & . & a_{mn} \end{bmatrix} \tag{7.3.15}$$

where

$a_{ij} = a_{ij}^{+} - a_{ij}^{-}$

where

a_{ij}^{+} - weight of arc from transition t_j to place p_i

a_{ij}^{-} - weight of arc from place p_i to transition t_j

The execution of Petri net is controlled by tokens assigned to places. It is modelled by firing of transitions. Firing of a transition removes one token from each of its input places and deposits one token to each of its output places. Transition is enabled if each of its input places has at least one token. The firing is an instantaneous and indivisible action. There are two firing modes: weak and strong firing modes. In weak firing mode an enabled transition may or may not fire. If the transition is enabled it will not be immediately fired in weak firing mode, this is only a possibility. In strong firing mode a transition is forced to fire as soon as it is enabled. A transition in a marked Petri net with marking m_k is enabled if

$$t_j \in T \quad \text{and} \quad p_i \in P \quad \text{true} \quad m_k(p_i) \geq \#(p_i, I(t_j)) \tag{7.3.16}$$

Firing a transition will change the marking M_k to a new marking M_l. Firing of transition t_j results in a new marking m_l of place p_i defined by:

$$m_l(p_i) = m_k(p_i) - \#(p_i, I(t_j)) + \#(p_i, O(t_j)) \tag{7.3.17}$$

where t_j - input transition
t_{j+1} - output transition

Firing in a Petri net is described by a firing sequence. The firing sequence defines a path of transitions in the Petri net along that the Petri net is "executed". If there is a firing sequence from marking M_k to marking M_l this firing sequence is described by a vector of firing sequence:

$$\sigma = (t_j, t_{j+1,}, t_{j+2}, \ldots t_m) \tag{7.3.18}$$

where $j=1...m-1$

The dynamic behaviour of the Petri net is represented by the firing rule and is described by the next equation if the firing sequence S is applied to it from initial marking M_0::

$$M_k = M_0 + WS \tag{7.3.19}$$

7.3.5 Properties of Petri Nets

Systems modelled by Petri net are described by functional and structural properties. Functional properties depend on the initial marking M_0, or actual marking M_k of Petri net. The topology, or net structure of Petri net defines structural properties.

Functional properties. *Reachability* describes whether a certain state in a system is reachable from an initial state. A marking M_k is reachable from initial marking M_0 if there exists a firing sequence that transforms a marking M_0 to M_k. *Coverability* defines whether there is a reachable marking that contains a particular marking subset from a given initial marking. *Liveness* defines whether a system can execute a particular task. A Petri net is live if from a marking M_i reached from an initial marking M_0, there is a firing sequence that would enable any further transition t_j to fire. If a Petri net is live, then it does not have any transition that can not fire in any firing sequence. If a Petri net is live there is no deadlock in it, i.e. no marking can be reached that disables firing of one or more transitions. *Conservativness* defines whether the number of tokens in the net remains constant. A Petri net is strictly conservative or part of a Petri net is non-strictly conservative if the number of tokens remains constant within it. In other words, if a Petri net is conservative it does not gain or lose tokens. Conservativness is important in Petri net where tokens describe objects that represent resources *Boundedness* describes whether a system has a finite capacity. A Petri net is k-bounded if the number of tokens in any place p_i is always less or equal to k for every reachable marking M_k from the initial marking M_0, where k is a bound of the Petri net. If a Petri net is bounded then for every reachable marking the number of tokens in every place is bounded. *Safeness* is a special case of boundedness where k equal to 1. If a Petri net is safe then there is no possibility to restart process that is currently executed. *Reversibility* describes a cyclic behaviour of a system, i.e. system can perform its tasks repeatedly. Petri net is reversible if from marking M_k the initial

marking M_0 is reachable. In reversible Petri net a system can always return to its initial marking from any actual marking.

Structural properties. They are independent of initial or actual marking. *Liveness.* A Petri net is structurally live if there is a finite set of initial marking which enable transitions to fire. This set is a set of live initial marking for a Petri net. *Boundedness.* A Petri net is structurally bounded if it is bounded for any finite set of initial marking. *Conservativness.* A Petri net is structurally conservative if the weighted sum of tokens for every place is constant for every fixed initial marking M_0 and actual marking M_k. Conservativness is a special case of boundedness.. *Repetitiveness.* A Petri net is repetitive if there exists an initial marking M_0 and a firing sequence σ from M_0 that every transition appears infinitely often in σ. *Consistency.* A Petri net is consistent if there is a marking M_0 and a firing sequence σ on path from M_0 to M_0 that every transition occurs at least once in σ in. Consistency is a special case of repetitiveness. *Controllability.* A Petri net is controllable if any marking is reachable from any other marking.

7.3.6 Time Augmented Petri net

Activities modelled by the "basic" Petri nets are primitive activities that are instantaneous and non-simultaneous. Activities in real-time systems are non-primitive events that are performed during a certain amount of time. There are also simultaneous activities in real-time systems that may overlap in time. The "basic" Petri net cannot properly model these activities. They must be extended by time in order to model real-time systems. The time would represent the duration and occurrence time of activities. The duration time defines time interval during that an activity is performed. For example: it is equal to execution time of an activity. The occurrence time describes a time instant when an activity happens. For example: an activity starts or ends.

The time augmented Petri nets belong to non-autonomous Petri nets that permit quantitative modelling of real-time systems. They describe not only what happens but also when it happens. The time is associated with either the places (P-timed PN) or transitions (T-timed PN) or both (PT-timed PN) in the time augmented Petri nets.

7.3.6.1 P-Timed Petri Net

Non-negative time δ is assigned to each place p_i in P-timed Petri nets. Each place has either a positive time to model execution of an activity or zero time to model a condition. The start and end of these activities are represented by transitions and their execution by places. In P-timed Petri net no time is associated with transitions. Thus, whenever a transition fires, the corresponding input tokens are removed immediately and the corresponding output tokens are also placed at its output places immediately.

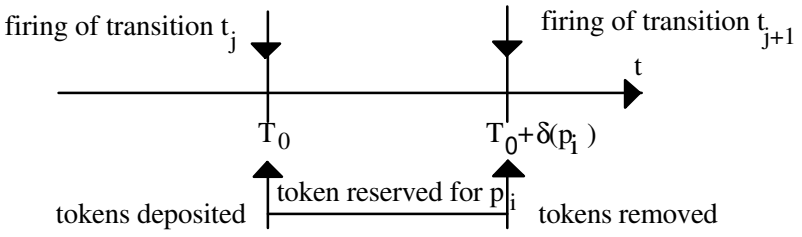

Figure 7.3.2 P-timed Petri net

For example in Fig. 7.3.2. a token is deposited to a place p_i at T_0. after firing transition t_j. This token will be unavailable to output transitions of the place p_i during the time interval $\delta(p_i)$. As a result, it cannot be removed by its output transition(s) during this time interval. The token can be removed by transition t_j only after completing the activity assigned to place p_i. Thus, transition t_{j+1} cannot fire earlier than $T_0+\delta(p_i)$.

7.3.6.2 T-Timed Petri Net

In this approach the firing of a transition has a duration time that is equal to execution time of an activity. Thus, a non-negative time value is assigned to each transition t_j. When a transition

t_j starts firing at a time T_0, tokens in its input place(s) are not available for any other transition during the time interval given by non-negative time value of transition t_j. They cannot be used by other transitions during this time interval. Tokens may be removed from input place(s) of transition t_j only after finishing the firing. In T-timed Petri net no time is associated with a place. Thus, whenever a token is put to a place, it is immediately available to its output transition(s). There are two types of T-timed Petri nets: timed and time Petri nets.

Timed Petri nets In Timed Petri nets [Ram80], [Hol87] duration of firing of the transition is given by a non-negative time delay TC_{del}. A non-negative time delay is assigned to each transition t_j. While a transition t_j is firing tokens remain in its input place(s). They are reserved for transition t_j. In this approach firing is an non-instantaneous event that takes place during time interval TC_{del}. The tokens are immediately removed after elapsing time TC_{del} because timed Petri Net has strong firing mode. For example in Figure 7.3.3 a transition t_j starts firing at T_0. It removes tokens from its input places only after finishing firing, i.e. at time $T_0 + TC_{del}$.

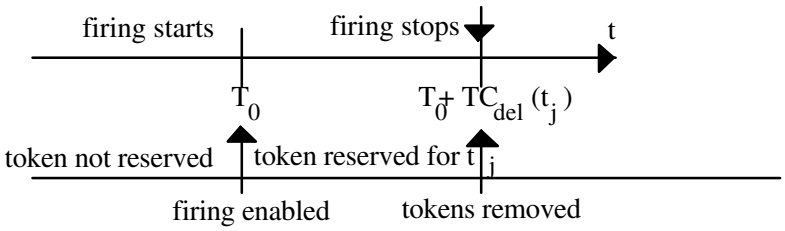

Figure 7.3.3 Timing of Timed Petri net

Time Petri Nets In the Time Petri Net [Mer76], [Lev87] there is a pair of time constraint; minimum time $T_{min}(t_j)$ and maximum time $T_{max}(t_j)$ that are assigned to each transition t_j. This time pair defines a time interval for each transition. $T_{min}(t_j)$ is a minimum delay that the transition cannot fire before and $T_{max}(t_j)$ is a maximum delay that the transition cannot fire after. Tokens are reserved for transition t_j in its input place(s) during time interval $[T_0 + T_{min}]$. This time interval represents the execution time of an activity. As a result, the firing is an "instantaneous" event in the Time Petri Net. For example: in Figure 7.4.2.2 timing of a Time Petri net is shown. If a transition t_j is enabled at time T_0 then it can fire at any time in time interval $[T_0 + T_{min}; T_0 + T_{max}]$. If the transition did not fire during this time interval token(s) must be removed from its input place at time $T_0 + T_{max}$ because the Time Petri Net also has a strong firing mode.

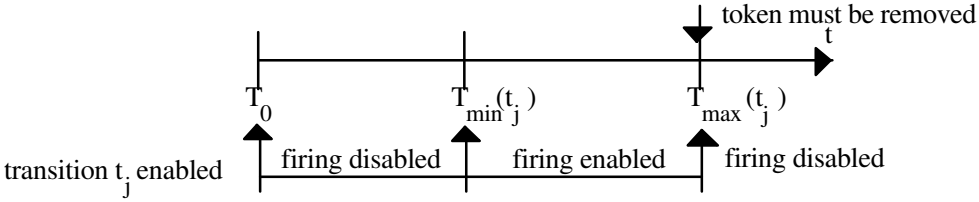

Figure 7.3.4.Timing of Time Petri net

7.3.7. Formal Methods Based on Time Augmented Petri Net

7.3.7.1. P-timed approaches

Coolahan et al. [Coo83], [Coo85], developed a modular dual-language method to specify and verify real-time systems. They describe behaviour of time-driven systems using a driving cycle and modular blocks. The driving cycle consists of a place and a transition that form a loop. They defined and described the following blocks: sequential, asynchronous, synchronous, decisive, alternative, non-shared and shared resources blocks. These blocks have predetermined structures with predetermined properties. They describe temporal properties of real-time systems by using frequencies that transitions fire at. Frequencies of transitions are given relatively to the firing frequency of the transition in the driving cycle. The driving cycle determines the execution time of each process. Authors defined a maximum relative firing frequency of transition (MRFF), which is the number of times the transition fires after each firing of the transition in the driving cycle, and minimum token arrival time of place (MTIAT), which is the shortest possible time between arrivals of any two consecutive tokens at a place.

To specify real-time systems they develop a system model. The modular blocks provide a modular approach to model temporal behaviour. Their initial model contains a place, which represents the whole system. This place is connected to the driving cycle. It is decomposed using the modular blocks to get the final system model. Having the system model they define parallelism and shared resources in systems and remove redundancy to optimise performance. They calculate the MRFF of all input transitions of place to specify time constraints and conditions for safeness. They assign value of MRFF to each transition and elaborate input and output equations of places. According to their definition a real-time system is safe if its input and output equations are consistent. They are consistent if and only if MRFFs assigned to input transitions of each place are equals to MRFFs assigned to output transitions of each place. To verify specification they solve these equations. If they contain contradiction then the system is inconsistent and system specification is not correct.

7.3.7.2. T-Timed Approaches

Berthomieu et al. [Ber91] developed a dual-language method that describes the system behaviour by states and state classes. It analyses the system behaviour by a reachability analysis. The state consists of a usual marking and firing interval set that contains firing times of enabled transitions. Each firing interval consists of minimum and maximum firing times that define a time interval while the transition must fire. The state class is union of states that includes marking and firing domain set. The firing domain set is a set of firing constraints. It is defined by a set of inequalities. A state is reached from an initial state by a firing sequence whose firing times are defined by firing interval set. A state class is a set of all states reached from the initial state by a firing sequence whose firing constraints are determined by firing domain set. A specification of a system, which is described by a set of state classes, is analysed by searching solutions of set of inequalities of firing domains. If these inequalities have a solution then the next state class is reachable. For example: if the initial class is given and the firing a transition t_j is enabled by the initial marking then the next class can be reached if inequalities, which describe the next firing domain, have solutions. If all state classes of the system are reachable then the specification is correct.

7.3.7.4. PT-Timed Approaches

Tsai et al [Tsa95] proposed a dual-language PT-timed approach. They developed a Time Augmented Petri Net whose places and transitions are associated with minimum, maximum and duration time constraints. A system specification is performed by functional and temporal specification steps. During temporal specification minimum, maximum and duration time constraints are added to places and transitions of the "basic" Petri net. Having these time constraints time intervals during that transitions are enabled, starting and ending times of firing of transitions and arrival and removing times of tokens can be calculated. Functional and

temporal properties of the system are asserted by reachability and schedulability analyses after the specification phase. In reachability analysis they verify whether a given marking is reachable from the initial marking in the "basic" Petri net. Unfortunately, reachable marking in the "basic" Petri net is not necessarily reachable under time constraints in time augmented Petri net. As a result, they also verify whether a reachable marking is schedulable under time constraints. In the schedulability analysis each transition is checked whether it is a weakly or strongly fireable or schedulable. Fireability and schedulability analysis are based on earliest fire beginning and latest fire ending times that are calculated from minimum, maximum and duration time constraints by "backward" (for earliest fire beginning time) and "forward" (for latest fire ending time) computations. Firing sequences are checked whether they are fireable and/or schedulable in time augmented Petri net using computed earliest fire starting and latest fire ending time values. A given marking is shedulable if there is at least one firing sequence whose transitions are strongly schedulable from the initial marking taking into account time constraints. If a given marking is not schedulable under the corresponding time constraints then the specification must be modified to meet requirements of strongly schedulability.

7.3.8. Evaluation of formal methods based on time augmented Petri net

According to Alanche et al. [Ala85] P-timed and T-timed Petri nets are equivalent because place-transition pairs are isomorphic when time constraints of transitions are considered. For example, given a model in T-timed Petri net, an equivalent model in P-timed net can be obtained by simple replacing each transition that has a positive time value by a "start" transition, a place with the corresponding time value and a "stop" transition. After replacing procedure, "start" and "stop" transitions are no more associated with time values. Similarly, a model in P-timed Petri net can be converted into T-timed Petri net.

All methods, analysed in previous chapter, specify behavioural and temporal properties of real-time systems using Petri net-based descriptive formalism. Activities and time values are associated with places and/or transitions of P-, T- and PT-timed Petri net to describe temporal behaviour and properties of real-time systems. In these methods places and transitions model components, their functions and relationship between them. As a result, these methods describe casual and temporal dependencies between activities and their constraints in behavioural and temporal domain. These methods represent graphically description of real-time systems. The graphical representation is more suitable to check correctness and performance of real-time systems in both design and application phases than methods that are not supported by graphical tools. Any inconsistency in specification, for example: any component with incorrect description, can be easily located. Petri net-based descriptive formalism developed by Coolahan et al. is a modular approach. It describes system models with modular blocks. It is more efficient to describe and analyse these modular blocks than the whole system.

All these methods verify temporal properties of real-time systems using Petri net-based operational formalism. Coolahan et al. [Coo85] developed a set of equations and inequalities, and a set of conditions for input and output transitions of each place of the modular blocks to check correctness of specification. They analyse modular blocks to define whether they meet the corresponding time constraints. Finding any inconsistency in specification they have to modify the corresponding modular block that contains it. All other approaches are non-modular ones, i.e. a system is not decomposed into subsystems to perform verification procedure. These methods examine the whole system. Tsai et al. [Tsa95] perform analysis of reachability and schedulability by checking firing sequences one by one not taking into consideration their relationships. They calculate firing and scheduling time of each firing sequence and examine them whether they can be performed taking into account their time constraints. If either fireability or schedulability is not obtained, they locate transitions whose time values must be modified to meet time constraints. But transitions may belong to more than one firing sequence. As a result, inconsistency in the specification and a subsequent change of

time values of transitions may affect fireability and schedulability of other firing sequences that include incorrect specification.

REFERENCES

[Ace96] Aceto L, Murphy D: Timing and Causality in Process Algebra
Acta Informatica, 1996, Vol. 33, No.4, pp. 317-350

[Ala85] Alanche P, Benzakour K, Dolle F, Gillet P, Rodriguez P, Valette R: PSI. A Petri Net Based Simulator for Flexible Manufacturing Systems.
Lecture Notes in Computer Science, 1985, Vol. 188, pp. 266-276

[All83] J. AllenJ: Maintaining Knowledge about Temporal Intervals
Communications of the ACM, 1983, Vol. 26, pp. 832-843

[Att91] H. Attiya, N. Lynch: Theory of Real-Time Systems - Project Survey
in "Foundations of Real-Time Computing: Formal Specifications and Metods" edited by Tilborg, Koob, Kluwer Academic, 1991, pp. 111-138

[Avr91] G. Avranunin, J. WiledenJ: Automated Analysis of Concurrent and Real-Time Software
in "Foundations of Real-Time Computing: Formal Specifications and Metods" edited by Tilborg, Koob, Kluwer Academic, 1991, pp. 195-215

[Ber91] B. Berthomieu, M. Diaz: Modeling and Verification of Time Dependent Systems Using Time Petri Nets
IEEE Trans. on Software Eng. 1991, Vol. 17, pp. 259-273

[Bus94] Bustard D. W, Winstanley A. C: Making Changes to Formal Specifications - Requirements and an Example
IEEE Trans. on Software Engineering, 1994, Vol. 20, No.8, pp. 562-568

[Cla86] E. Clarke, E. Emerson, A. Sistla: Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic
ACM Trans. on Programming Languages and Systems, 1986, Vol. 8, No.2, pp. 244-263

[Coo83] J. Coolahan, N. Roussopoulos: Timing Requirements for Timen-Driven Systems Using Augmented Petri Nets
IEEE Tran. on Software Eng. 1983, Vol. 9. No.5. pp. 603-616

[Das85] B. Dasarathy: Timing Constraints of Real-Time Systems: Constructs for Expresssing Them, Methods for Validating Them
IEEE Trans. on Software Eng. 1985, Vol. 11, pp. 80-86

[Del93] Deleon H, Grumberg O: Modular Abstractions for Verifying Real-Time Distributed Systems
Formal Methods in System Design, 1993, Vol. 2, No.1, pp. 7-43

[Dil88] L. Dillon, G. Avrunin, J. Wileden: Constrained Expressions: Toward Broad Applicability of Analysis Methods for Distributed Software Systems.
ACM Trans. on Programming Language Systems, 1988, Vol. 10, pp. 373-402

[Fel94] M. Felder, D. Mandrioli, A. Morzenti: Proving Properties of Real-Time Systems through Logical Specification and Petri Net Models
IEEE Trans. on Software Eng. 1994, Vol. 20, pp. 127-141

[Gen83] H. J. Genrich, K. Lautenbach: Application and Theory of Petri Nets
Springer, Berlin, 1983

[Ger92] R. Gerber, I. Lee: A Layered Approach to Automating the Verification of Real-Time Systems
IEEE Trans. on Software Eng. 1992, Vol. 18, pp. 768-784

[Haa81] V. Haase: Real-Time Behavior of Programs
IEEE Trans. on Software Eng. 1981, Vol. 7, pp. 494-501

[Hol87] M. A. Holliday, M. K. Vernon: A Generalized Timed Petri Nets Model for Performance Analysis
IEEE Trans. on Software Eng., 1987, Vol. 13, pp. 12297-1310

[Int85] International Workshop on Timed Petri Nets, Torino, Italy, July 1-3, 1985
IEEE Computer Society Press, 1985

[Jah86] F. Jahanian, A. Mok.: Safety Analysis of Timing Properties in Real-time Systems
IEEE Trans. on Software Eng. 1986, Vol. 12, Sep. pp. 890-904

[Koy88] R. Koyman, W. Shyamasundar; W. deRoever; R. Gerth; S. Arun-Kumar: Compositional Semantics for Real-Time Distributed Computing
Information and Computation, 1988, Vol. 79, No.3, pp. 210-256

[Lam83] Lamport L: Specifying Concurrent Program Modules
Communications of the ACM, 1983, Vol. 21, No.7, pp. 190-222

[Lam94] Lamport L: The Temporal Logic of Actions
ACM Trans. on Programming Languages and Systems, 1994, Vol. 16, No.3, pp. 872-923

[Lee94] Lee I. S, Bremondgregoire P, Gerber R: A Process Algebraic Approach to the Specification and Analysis of Resource-Bound Real-Time Systems
Proceedings of the IEEE, 1994, Vol. 82, No.1, pp. 158-171

[Lev87] N. G. Leveson, J. L. Stolzy: Safety Analysis Ussing petri Nets
IEEE Trans. on Software Eng., 1987, Vol. 13, pp. 386-397

[Mer76] P. M. Merlin, D. J. Farber: Recoverability of Communication Protocols Implications of Theoretical Study.
IEEE Trans. on Communication, 1976, Vol. 24, pp. 1036-1043

[Mok91] A. Mok:Towards Mechanisation of Real-Time System Design
in "Foundations of Real-Time Computing: Formal Specifications and Metods" edited by Tilborg, Koob, Kluwer Academic, 1991, pp. 1-37

[Pet81] J. Peterson: Petri Net Theory and the Modeling of Systems
Prentice Hall, 1981

[Pnu88] A. Pnueli, E. Harel: Application of Temporal Logic to the Specification of Real-Time Systems
Lecture Notes on Computer Science, Springer-Verlag, 1988 Vol. 331, pp. 84-98

[Ram80] C. V. Ramamoorthy, G. S. Ho: Performance Evaluation of Asynchronous Concurrent Systems Using Petri Nets
IEEE Trans. on Software Eng, 1980, Vol. 6, pp. 440-449

[Reu90] C. Reutenauber: The Mathematics of Petri Nets
Prentice-Hall, London, 1990

[Seb94) Sebastiani F, Straccia U: Incremental Knowledge Acquisition for Non-Monotonic Reasoning
Computers and Artificial Intelligence, 1994, Vol. 13, No.4, pp. 377-396

[Sha87] Shankar A. U, Lam S. L: Time-Dependent Distributed Systems: Provinf safety, Liveness and Real-Time Properties
Distributed Computing, 1987, Vol. 2, pp. 61-79

[Sha89] A. Shaw: Reasoning about Time in Higher-Level Language Software
IEEE Trans. on Software Eng. 1989, Vol. 15, No.7, pp. 875-889

[Tel88] Tel G: Assertional Verification of a Timer-Based Protocol
Lecture Notes of Computer Science 1988, Vol. 317. pp. 600-614

[Tsa95] J. Tsai, S. J. Yang, Y. Chang: Timing constraint Petri Net and Their Application to Scedulability Analysis of Real-Time System Specifications
IEEE Tran. on Software Eng. 1995, Vol. 21. No.1. pp. 32-49