Programmazione Avanzata e Paradigmi
Ingegneria e Scienze Informatiche - UNIBO
a.a 2013/2014
Lecturer: Alessandro Ricci

[module 3.2]
VISUAL FORMALISMS
VISUAL FORMALISMS

• Formalisms for rigorously describing models of concurrent systems by means of some kind of visual diagrams
  – representing system *dynamics*
  – representing task structure and task dependencies
• Useful for both requirement specification, analysis and design
  – formal analysis when formally specified
• Formalisms considered in this module
  – *Petri Nets*
  – *Statecharts*
  – *Activity Diagrams*
PETRI NETS
PETRI NETS

- Abstract, formal model of information flow
  - describing and analyzing the flow of information and control in systems
  - particularly systems that may exhibit asynchronous and concurrent activities
- Introduced by Carl Adam Petri ~ 1965
  - further developed, extended and adopted in many computer science contexts
- Major use
  - modeling of systems of events in which it is possible for some events to occur concurrently but there are constraints on the concurrence, precedence, or frequency of these occurrences
**PETRI NET GRAPH**

- Bi-partite graphs representing a Petri Net
  - two types of *nodes*
    - *places* (the circles) and *transitions* (the bars)
  - connected by directed arcs
    - from node $i$ to node $j$: $i$ is an input to $j$ and $i$ is an output of $i$

- Models the static properties of the system

![Petri Net Graph Diagram](image-url)
TOKENS AND MARKINGS

- In addition to the static properties represented by the graph, a Petri Net has *dynamic properties* that result from its *execution*
  - the execution of a Petri net is controlled by the position and movement of markers called **tokens** in the net.

- tokens are indicated by black dots, residing in places

- A Petri Net with tokens is called **marked Petri Net**
EXECUTION RULES

- Tokens are moved by the firing of transitions of the net
  - a transition must be enabled in order to fire
    - a transition is enabled when all of its input places have a token in them
  - the transition fires by removing the enabling tokens from their input places and generating new tokens which are deposited in the output place of the transition
MARKINGS

- The distribution of tokens in a marked Petri Net defines the state of the net and is called **marking**
  - the marking may change as a result of firing transitions
- Different transitions may fire, with different result markings
  - inherent *non-determinism*
TRANSITIONS ON THE BOUNDARY

- *source* transition
  - without any input place
  - just produce tokens

- *sink* transition
  - without any output place
  - just consume tokens
WEIGHTED ARCS

- A variant consider also weights for arcs:
  - a transition is enabled if each input place \( p \) of \( t \) is marked with at least \( w(p,t) \) tokens, where \( w(p,t) \) is the weight of the arc from \( p \) to \( t \)
  - a firing of an enabled transition \( t \) removes \( w(p,t) \) tokens from each input place \( p \) of \( t \), and adds \( w_1 \) tokens to each output place \( p \) of \( t \), where \( w(t,p) \) is the weight of the arc from \( t \) to \( p \)

- Reaction example

\[
2H_2 + O_2 \rightarrow 2H_2O
\]
MODELING WITH PETRI NETS

- Petri nets can be used to model quite naturally concurrent systems in terms of:
  - *events* and *conditions*
  - the relationships among them
- Interpretation
  - in a system at any given time certain conditions will hold
  - the fact that these conditions hold may cause the occurrence of certain events
  - the occurrence of events may change the state of the system
    - causing some of the previous conditions to cease holding and causing other conditions to begin to hold
  - firing of a transition = occurrence of an event
    - considered instantaneous or better: atomic change of the system
EXAMPLE: RESOURCE ALLOCATION

A NEW JOB ENTERS THE SYSTEM

A JOB IS ON THE INPUT LIST

THE PROCESSOR IS IDLE

JOB PROCESSING IS STARTED

A JOB IS BEING PROCESSED

JOB PROCESSING IS COMPLETED

A JOB IS ON THE OUTPUT LIST

A JOB LEAVES THE SYSTEM

A CARD READER IS NEEDED

ALLOCATE THE CARD READER

A CARD READER IS AVAILABLE

NO CARD READER IS AVAILABLE
EXAMPLE: A VENDING MACHINE
**INTERPRETATIONS**

- Typical interpretations of places and transitions

<table>
<thead>
<tr>
<th>Input places</th>
<th>Transition</th>
<th>Output places</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-conditions</td>
<td>Event</td>
<td>Post-conditions</td>
</tr>
<tr>
<td>Input data</td>
<td>Computational step</td>
<td>output data</td>
</tr>
<tr>
<td>Input signals</td>
<td>Signal processor</td>
<td>Output signals</td>
</tr>
<tr>
<td>Resource needed</td>
<td>Task or Job</td>
<td>Resource Released</td>
</tr>
<tr>
<td>Conditions</td>
<td>Clause in Logic</td>
<td>Conclusion(s)</td>
</tr>
<tr>
<td>Buffers</td>
<td>Processor</td>
<td>Buffers</td>
</tr>
</tbody>
</table>
MODELING CONCURRENCY AND PARALLELISM

- PN are ideal for modeling systems of distributed control with multiple processes occurring concurrently.

Modeling two parallel activities

Modeling two parallel activities with synchronization
DATA-FLOW COMPUTATION

\[ x = \frac{a + b}{a - b} \]

The diagram illustrates a data-flow computation process using Petri nets. The steps are as follows:

1. **Copy** input \( a \) and \( b \).
2. **Add** \( a \) and \( b \) to get \( a + b \).
3. **Divide** \( a + b \) by \( a - b \) to get \( x \).

Additionally, there is a conditional check:

- If \( a - b \neq 0 \), proceed with the division.
- If \( a - b = 0 \), then \( x \) is undefined.
MODELING CONFLICT AND CONCURRENT EVENTS

- Representing conflicting or choice events:

- **Conflict events vs. concurrent events**
  - two events e1 and e2 are *in conflict* if either e1 or e2 can occur but not both
  - two events e1 and e2 are *concurrent* if both events can occur in any order without conflicts

- A situation where conflict and concurrency are mixed is called a *confusion*
ASYNCHRONY AND LOCALITY

• In a PN there is no inherent measure of time or the flow of time
  – the only important property of time, from a logical point of view, is in defining a **partial ordering** of the occurrence of events
  – events which need not be constrained in terms of their relative order of occurrence are not constrained

• **Locality**
  – in a complex systems composed by independent asynchronously operating subparts each part can be modeled by a Petri Net
  – the enabling and firing of transitions are then affected by, and in turn affect only, *local changes* in the marking of the Petri Net
NON-DETERMINISM

• Naturally modeling non deterministic behaviors
  – a PN is viewed as a sequence of discrete events whose order of occurrence is one of the possibly many allowed by the basic structure
  – if at any time more than one transition is enabled, then any of the several enabled transitions may fire
  – the choice as to which transition fires is made in a nondeterministic manner
    • randomly or by forces that are not modeled
ATOMIC VS. NON-ATOMIC EVENTS

- The occurrence of (primitive) events is instantaneous
  - non primitive events (with a duration) must be modeled by multiple events
  - e.g.: activity with a beginning and an ending event
HIERARCHIES

- Natural support to model hierarchies
  - an entire net may be replaced by a single place or transition for modeling at a more abstract level (abstraction)
  - places and transitions may be replaced by subnets to provide more detailed modeling (refinement)
APPLICATION TO CONCURRENT DESIGN AND PROGRAMMING

• PN can be naturally used to model software systems, concurrent software systems in particular
• Representing problems
  – critical sections and mutual exclusion problems
  – synchronization
• Representing mechanisms behavior
  – semaphores
  – synchronizers
• Representing entire problems
  – Producers / Consumers
  – Readers-Writers
  – Dining Philosophers
CRITICAL SECTION AND MUTUAL EXCLUSION PROBLEMS

- p2 and p4 represent critical sections
  - s is the token needed for entering in CS
SYNCHRONIZATION

- Imposing an order between process actions
  - process actions represented by transitions
  - relating actions (transitions) through conditions (places)

- b action of process Q can be executed after the execution of action a of process P

- b action of process Q and a action of process P must be executed synchronously
COMMUNICATION

The diagram illustrates a communication process modeled using Petri nets. It shows the interaction between two processes, Process 1 and Process 2, involving the actions of sending, receiving, and acknowledging messages. Key states include:

- **Send message**
- **Receive message**
- **Message received**
- **Send ack.**
- **Receive ack.**
- **Ack. received**
- **Ack. sent**

The transitions between these states are indicated by directed arcs, demonstrating the flow of messages and acknowledgments.
SEMAPHORES

- Semaphore modeled as a shared resource (place)
  - modeling wait (P) as a transition with the semaphore res. as input place
  - modeling signal (Q) as a transition with the semaphore res. as output place
PRODUCERS AND CONSUMERS

PRODUCERS

BOUNDDED-BUFFER

CONSUMERS

PCD LM - II Facoltà Ingegneria - Cesena

Modeling Processes - Petri nets
READERS-WRITERS

- The n tokens in p1 represent n processes that may want to read or write a shared memory represented by p3
DINING PHILOSOPHERS

- Forks are represented by places $a_i$
- Philosophers thinking by $A_t, B_t, C_t, D_t, E_t$ places and eating by $A_e, B_e, C_e, D_e, E_e$ places
EXTENDED PETRI NETS WITH INHIBITOR ARCS

- Zero-testing extension
  - extending the basic PN with the possibility of firing a transition only if a certain place has zero tokens
    - *inhibitor arc* represented by an arc with a small circle at the end

- PN+inhibitor arc = Turing-equivalent
  - expressiveness and undecidability problems
FORMAL DESCRIPTION OF PETRI NETS

• PN can be formally described so as to enable a rigorous analysis of properties and problems of the system modeled
  – structural properties
    • independent from the initial marking
  – behavioral properties
    • dependent on the marking

• Mapping correctness of the systems on to structural / behavioural properties of the nets
  – safety and liveness properties
PETRI NET STRUCTURE

• The structure of a Petri Net can be formally described as a tuple

\[ C = (P, T, I, O) \]

– \( P \) is a set of places
– \( T \) is a set of transitions
– input function \( I \) defines the set of input places for each transition \( t_j \)
– output function \( O \) defines the set of output places for each transition \( t_j \)
EXAMPLE

\[ P = \{ p_1, p_2, p_3, p_4, p_5 \} \]
\[ T = \{ t_1, t_2, t_3, t_4 \} \]

\[ I(t_1) = \{p_1\} \]
\[ I(t_2) = \{p_2, p_3, p_5\} \]
\[ I(t_3) = \{p_3\} \]
\[ I(t_4) = \{p_4\} \]

\[ O(t_1) = \{p_2, p_3, p_5\} \]
\[ O(t_2) = \{p_5\} \]
\[ O(t_3) = \{p_4\} \]
\[ O(t_4) = \{p_2, p_3\} \]

Corresponding Petri-Net graph:
MARKING

• A **marking** is an assignment of tokens to the places of the net

• Can be formally represented either as
  - a vector of N elements, one for each place, representing the number of tokens for each place
  - a function \( \mu : P \rightarrow N \)

• \( \mu(p_i) \) is the number of tokens in the place \( p_i \)

• A marked Petri Net is represented by 5-tuple: \( M = (P, T, I, O, \mu) \)
EXECUTION RULE SEMANTICS

• The state of a Petri Net is defined by is marking
  – firing of a transition represents a change in the state of the net.

• Next-State partial function $\delta(\mu, t_j)$
  – the function is undefined if the transition is not enabled in the marking
  – if $t_j$ is enabled, $\mu' = \delta(\mu, t_j)$ is the marking that results from removing tokens from the input of $t_j$ and adding tokens to the output of $t_j$.

• Given a PN and an initial marking, we can execute the PN by successive transition firings
  – firing a transition $t_j$ in the initial marking produces a new marking $\mu^1 = \delta(\mu^0, t_j)$
  – in this new marking we can fire any new enabled transition, say $t_k$, resulting in a new marking $\mu^2 = \delta(\mu^1, t_k)$
  – this can continue as long as there is at least one enabled transition in each marking
  – if we reach a marking in which no transition is enabled, then no transition can fire and the PN must stop

• Non determinism
  – multiple sequence of markings $(\mu^0, \mu^1, \mu^2, \ldots)$ and related transitions $(t^0_j, t^1_j, t^2_j, \ldots)$ can result by executing a PN
REACHABILITY SET

• **Immediately reachable** marking
  – a marking $m_i'$ is *immediately reachable* from $m_i$ if we can fire some enabled transition in $m_i$ resulting in $m_i'$

• **Reachable marking**
  – a marking $m_i'$ is *reachable* from $m_i$ if it is immediately reachable from $m_i$ or is reachable from any marking $m_i''$ which is immediately reachable from $m_i$

• **Reachability set** of a PN
  – set of all states into which the PN can enter by any possible execution
MATRIX REPRESENTATION

- The components of a Petri net can be conveniently represented as matrices
  - \textit{input} matrix \( D_- \), \textit{output} matrix \( D_+ \), \textit{incident} matrix \( D \)
    - \((m \times n)\) matrix, where \( m \) (rows) are transitions and \( n \) (columns) are places
    - \textit{Input matrix} \( D_- \)
      - element \( D_-[i,j] \) = weight of arc connecting place \( p_j \) with transition \( t_i \)
    - \textit{Output matrix} \( D_+ \)
      - element \( D_+[i,j] \) = weight of arc connecting transition \( t_i \) with place \( p_j \)
    - \textit{Incident matrix} \( D = D_+ - D_- \)
  - \textit{transition} matrix \( T \)
    - \((1 \times m)\) matrix representing the firing of the Petri net, i.e. what transitions can be fire
  - \textit{current marking} matrix \( M \)
    - \((1 \times n)\) matrix representing the current number of tokens in places
- To compute the evolution of a Petri Net:
  \( M' = T \cdot D + M \)
ANALYSIS OF PN MODELS

• **Safe** nets
  – Petri nets in which no more than one token can ever be in any place of the net at the same time.
  – Justification based on the original definition of events and conditions
    • a condition is represented by a place.
    • the fact that the condition holds is indicated by a token in the place
    • so a token should be either present or not: more than 2 token is pointless

• **Bounded net** or **k-bounded net** (boundness)
  – Nets in which the number of tokens in any place is bounded by k
  – safe nets are 1-bounded net
  – Boundedness is a very important practical property

• **Conservative** net
  – PN is conservative if the number of tokens in the net is conserved.
  – think for instance of using tokens to represent resources..
LIVENESS

• Based on transition analysis
  – dead transition in a marking
    • if there is no sequence of transition firings that can enable it
    • related to deadlock situations
  – potentially fireable
    • if there exists some sequence that enables it
    • related to starvation situation
  – live transition
    • if it is potentially fireable in all reachable markings

• For liveness, it is important not only that a transition be fireable in a given marking, but staying potentially fireable in all markings reachable from that marking
  – if it is not true, then it is possible to reach a state in which the transition is dead => deadlock
PETRI NET EXTENSIONS

• **Timed Nets**
  – introducing time delays associated with transitions and/or places
    • useful for performance evaluation and scheduling problems
  – deterministic timed nets
    • delays are deterministically given
  – **stochastic nets**
    • delays are probabilistically specified

• **High-level Nets**
  – associate some kind of symbolic / numerical information to tokens and some computational rules to transition consuming and producing tokens
    • Coloured Petri Nets, Predicate Transitions Nets
  – **Coloured Petri Nets**
    • assigning typed values (“a color”) to each token
PETRI NET TOOLS

• Many tools available for creating and analyzing Petri Nets
  – check http://www.informatik.uni-hamburg.de/TGI/PetriNets/
• An example: PIPE 2
  – Platform Independent Petri net Editor 2
  – Java-based, open-source: http://pipe2.sourceforge.net/
STATECHARTS
STATECHARTS

• Introduced by David Harel in 1987 for modelling complex reactive systems
  – now part of UML with the name of state diagrams
• Reactive systems
  – systems being, to a large extent, event-driven, continuously having to react to external and internal stimuli
    • examples include automobiles, communication networks, operating systems, avionic systems, man-machine interface of many ordinary software
  – contrast to transformational systems
    • input / output, data-processing systems
• Main objective
  – introducing a way of describing reactive behaviour that is clear and realistic, and at the same time formal and rigorous
    • to be simulated and analyzed
BEYOND BASIC STATE DIAGRAMS

• General agreement that **states** and **events** are a rather natural medium for describing the dynamic behaviour of a complex system:
  – state transition: "when event alfa occurs in state A, if condition C is true at that time, the system transfers to state B"

• But finite state machine and state transition diagrams don’t scale with complexity:
  – unmanageable, exponentially growing multitude of states, all of which have to be arranged in a "flat" unstratified manner
  – lead to unstructured, unrealistic and chaotic state diagrams

• To be useful a state/event approach must be *modular*, *hierarchical* and *well-structured*:
  – it must solve the exponential blow-up problem, by somehow relaxing the requirement that all combinations of states have to be represented explicitly

• Statecharts proposal:
  – extension of conventional state diagrams by mechanisms to enhance the descriptive power
STATECHARTS FORMALISM

• Visual formalism to describe states and transitions in a modular fashion
  – hierarchy
    • clustering
    • refinement
    • promoting 'zoom' capabilities for moving easily back and forth between levels of abstractions
  – orthogonality
    • independence / concurrency of substates
    • synchronisation among substates
STATE AND EVENTS IN STATECHARTS

- States and events
  - **boxes** (rounded rectangle) denote states
  - **arrows** labelled with event
    - optionally with a parenthesised **conditions** and an **action** (described later on)

- Different state levels ( = hierarchy support )
  - encapsulation express hierarchies
  - arrows can originate and terminate at any level
HIERARCHY: CLUSTERING

- XOR Decomposition
  - economizing arrows

  ![Diagram](image)

- since beta takes the system to B from either A or C, we can cluster the latter into a new super-state D and replace the two arrows by one
  - the semantics of D is a XOR of A and C: to be in state D one must be either in A or in C, and not in both.
  - D is an *abstraction* of A and C
    - capturing common properties
- bottom-up approach
REFINEMENT

- We can proceed on the opposite direction, refining states:
  - in this case the incoming alfa and beta arrows are underspecified

- top-down approach

- Zooming in and out support
  - zooming-in
    - by looking inside a state
  - zooming-out
    - abstracting from the inside of a state
DEFAULT STATES

• Special arrow to explicitly represent the default entering state
  – at any level

• Take into the account the history
  – entering the state most recently visited
  – H default-state arrows
HIERARCHY 2: ORTHOGONALITY

• AND decomposition
  – capturing the property that, being in a state, the system must be in all of its AND components
  – the notation used in statecharts is the physical splitting of a box into component using dashed lines

• state Y consists of AND components A and D
  • with the property that being in Y entails being in some combination of B or C with E, F or G.
  • Y is the orthogonal product of A and D

– Independency and / or Concurrency
INDEPENDENCE

- On the other hand, mi occurs at (B,F) it affects only the D component, resulting in (B,E)

- This illustrates a certain kind of independence
  - the transition is the same whether the system is in B or C in its A components
AND-FREE EQUIVALENT

• The AND-Free equivalent diagram has the product of the states

• That is: if we have two components with 1000 states, we have one million of states in the product
  – if we have 3 components: $10^9$ states..
SYNCHRONISATION

- In the example, if an event alfa occurs, it transfers B to C and F to G simultaneously, resulting in a new combined state (C,G)

- This illustrates a certain kind of synchronisation
  - a single event causing simultaneous happenings
NOTATIONS FOR AND-DECOMPOSITION
AN EXAMPLE

AVIONICS SYSTEM

general-mode

navigate

cruise
touchdown
take off

on-ground

radar

off
switch off
standby
end warmup

on

switch on

abc-system

off
lever off

calibrate
end calibration

lever on
ACTIONS (1/2)

- Actions represent the ability of the statecharts to generate events and to change the value of conditions
  - influencing other components of the system
  - influencing the environment of the system
- Expressed by the notation ".../W" that can be attached to the label of the transition
  - W is an action carried out by the system
  - actions have instantaneous occurrences that take ideally 0 time.
ACTIONS (2/2)

- Actions can be produced also when entering or when leaving a state
  - entry action (S, V, U in the example)
  - exit action (T in the example)
REPRESENTING ACTIVITIES

• Activities always take a nonzero amount of time (like beeping, displaying, executing lengthy computations..)
  – activities are durable
• So in statecharts activities are associated with states
  – exploiting entry and exit actions

![Statechart Diagram]
FURTHER FEATURES: UNCLUSTERING

- Laying out parts outside the natural neighbourhood
THE STATEMATE TOOL

- Statemate is a comprehensive graphical modeling and simulation tool for the rapid development of complex embedded systems based on statecharts
  - using a combination of traditional graphical design notations combined with some of the Unified Modeling Language (UML) diagrams
- Statemate provides a direct and formal link between user requirements and software implementation by allowing the user to create a complete, executable specification
  - this specification may be executed, or graphically simulated, so the system engineer can explore what-if scenarios to determine if the behavior and the interactions between system elements are correct
  - these scenarios can be captured and included in Test Plans which are later run on the embedded system to ensure that what gets built meets what was specified.
  - this executable specification is also used to communicate with the customer or end user to confirm that the specification meets their requirements
ACTIVITY DIAGRAMS
ACTIVITY DIAGRAMS

• Activity diagrams are one of the diagrams adopted in UML to represent the business and operational workflows of software system
  – an activity diagram is a dynamic diagram that shows the activity and the event that causes the object to be in the particular state

• Activity diagrams vs. state diagrams
  – a state diagram shows the different states an object is in during the lifecycle of its existence in the system, and the transitions in the states of the objects
    • these transitions depict the activities causing these transitions, shown by arrows
  – an activity diagram talks more about these transitions and activities causing the changes in the object states
ACTIVITY DIAGRAM OVERVIEW (*)

• Showing the flow of activities through the system
  – diagrams are read from top to bottom and have branches and forks to describe conditions and parallel activities. A fork is used when multiple activities are occurring at the same time.

• The diagram below shows a fork after activity1
  – this indicates that both activity2 and activity3 are occurring at the same time. After activity2 there is a branch. The branch describes what activities will take place based on a set of conditions. All branches at some point are followed by a merge to indicate the end of the conditional behavior started by that branch. After the merge all of the parallel activities must be combined by a join before transitioning into the final activity state.

(*) taken from http://atlas.kennesaw.edu/~dbraun/csis4650/A&D/UML_tutorial/activity.htm
PROCESSING ORDER EXAMPLE (*)

- The diagram shows the flow of actions in the system's workflow
  - once the order is received the activities split into two parallel sets of activities
  - one side fills and sends the order while the other handles the billing
  - on the Fill Order side, the method of delivery is decided conditionally.
  - depending on the condition either the Overnight Delivery activity or the Regular Delivery activity is performed.
  - finally the parallel activities combine to close the order.

(*) taken from http://atlas.kennesaw.edu/~dbraun/csis4650/A&D/UML_tutorial/activity.htm
SWIMLANES

- A swimlane is a way to group activities performed by the same actor on an activity diagram or to group activities in a single thread.
SIGNALS

- *Signal Activities*
  - activities that send or receive messages (output / input signals)
- *Triggers*
  - temporal signals

input signal

![Diagram of signal activities and triggers]

output signal

**Join Spec**: The schedule is printed and the date is on or after April 1st.
PASSING OBJECTS

• Specifying objects passed between activities

object passed between activities
BIBLIOGRAPHY