Timed Languages

- A time sequence $\tau=\tau_1\tau_2...$ is an infinite sequence of time values $\tau_i \in R$ with $\tau_i>0$, satisfying the following constraints:
  - Monotonicity: τ increases strictly monotonically so that $\tau_i < \tau_{i+1}$ for all $i \geq 1$.
  - Progress: For every $t \in R$, there is some $i \geq 1$ such that $\tau_i > t$.
- A timed word over an alphabet Σ is a pair (σ,τ) where $\sigma=\sigma_1\sigma_2...$ is an infinite word over Σ and τ is a time sequence. A timed language over Σ is a set of timed words over Σ.
- Viewed as an input to an automaton a timed word (σ,τ) presents the symbol σ at time τ.

Examples of Timed Languages

$$L_1 = \{((a \mid b)^{\omega}, \tau) \mid \forall i.((\tau_i > 5.6) \rightarrow (\sigma_i = a))\}$$

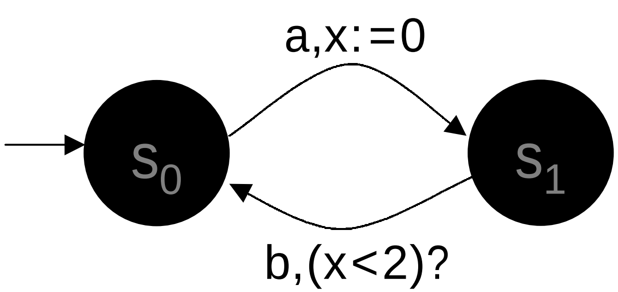
- Language consists of all timed words (σ,τ) such that there is no *b* after time 5.6.

$$L_2 = \{((ab)^{\omega}, \tau) \mid \forall i.((\tau_{2i} - \tau_{2i-1)}) < (\tau_{2i+2} - \tau_{2i+1}))\}$$

- Language consists of all timed words (σ,τ) in which *a* and *b* alternate and for successive pairs of *a* and *b* the time difference between *a* and *b* keeps increasing.

Timed Transition Tables

- The choice of the next state depends on the input symbol read and on the time of the input symbol relative the the times of the previously read symbols.
- Thus, real-valued clocks are associated with the transition table:
  - Clocks can be independently reset to 0 with any transition.
  - Clocks keep track of time elapsed since last reset.
  - Transitions can put constraints on clock values: a transition may be taken only if the current values of the clocks satisfy the associated constraints.
  - All clocks increase at a uniform rate, counting time with respect to a fixed global time frame; they do not correspond to locals clocks of different components in a distributed system.



$$L \approx \{((ab)^\omega, \tau) \mid \forall i.(\tau_{2i} < \tau_{2i-1} + 2)\}$$

Timed Transition Tables

- For a set X of clock variables, the set $\Phi(X)$ of clock constraints δ is defined inductively by
  $\delta := x \leq c \mid c \leq x \mid \neg\delta \mid \delta_1 \wedge \delta_2$
  where x is a clock in X and c is a constant in Q, the set of nonnegative rationals.

- A clock interpretation ν for a set X of clocks assigns a real value to each clock, ie a mapping from X to R.

- A timed transition table A is a tuple (Σ,Q,Q_0,C,E) where Σ is a finite alphabet, Q is a finite set of states, $Q_0 \subseteq Q$ is a set of start states, C is a finite set of clocks, and $E \subseteq Q \times \Sigma \times Q \times 2^C \times \Phi(C)$ gives the set of transitions. An edge (s,a,s',λ,δ) represents a transition from state s to state s' on input symbol a. The set $\lambda \subseteq C$ gives the clocks to be reset with this transition and δ is a clock constraint over C.

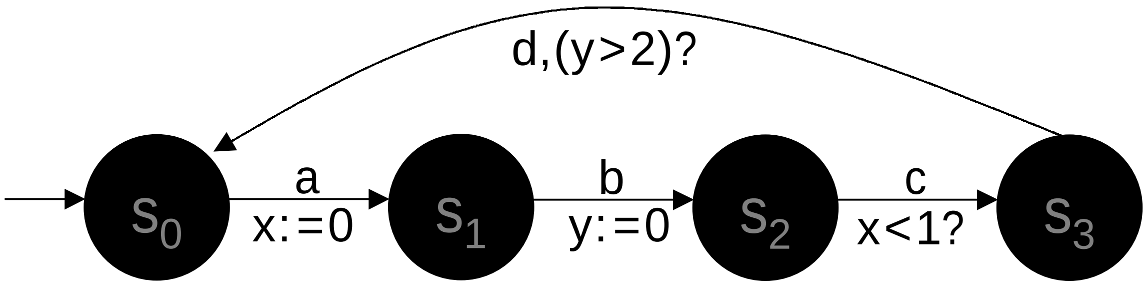
Timed Runs

- A run r, denoted by (s,ν), of a timed transition table (Σ,Q,Q_0,C,E) over a timed word (σ,τ) is an infinite sequence of the form
  $$r : (s_0, \nu_0) \xrightarrow[\tau_1]{\sigma_1} (s_1, \nu_1) \xrightarrow[\tau_2]{\sigma_2} (s_2, \nu_2) \xrightarrow[\tau_3]{\sigma_3} \ldots$$
  with $s_i \in S$ and $\nu_i \in [C \to R]$, for all $i \geq 0$, satisfying the following requirements:
  – Initiation: $s_0 \in Q_0$, and $\nu_0 = 0$ for all $x \in C$.
  – Consecution: for all $i \geq 0$, there is an edge in E of the form $(s_{i-1},\sigma,s_i,\lambda,\delta)$ such that $(\nu_{i-1}+\tau_i-\tau_{i-1})$ satisfies δ_i and ν_i equals $[\lambda_i \to 0]\ (\nu_{i-1}+\tau_i-\tau_{i-1})$

  The set inf(r) consists of those states $s \in Q$ such that $s=s_i$ for infinitely many $i \geq 0$.

Timed Run: Example

- Consider a timed word (a,2) →(b,2.7) →(c,2.8)→ (d,5) given as input to the following timed transition table:



$$r:(s_0,[0,0])\xrightarrow[2]{a}(s_1,[0,2])\xrightarrow[2.7]{b}(s_2,[0.7,0])\xrightarrow[2.8]{c}$$
$$(s_3,[0.8,0.1])\xrightarrow[5]{d}(s_0,[3,2.3])\ldots\ldots$$

Timed Regular Languages

- A timed Büchi automaton TBA is a tuple $(\Sigma, Q, Q_0, C, E, F)$ where (Σ, Q, Q_0, C, E) is a timed transition table and $F \subseteq S$ is a set of accepting states. A run $r=(s,\nu)$ of TBA over a timed word (σ,τ) is called an accepting run iff $\mathrm{inf}(r) \cap F \neq \emptyset$.
- For a TBA $\mathcal{A}$ the language $L(\mathcal{A})$ of timed words it accepts is defined as the set
 $L(\mathcal{A}) = \{(\sigma,\tau) \mid \mathcal{A}$ has an accepting run over $\sigma,\tau)\}$.
- A timed language L is a timed regular language iff $L=L(\mathcal{A})$ for some TBA $\mathcal{A}$.
- The class of timed regular languages is closed under (finite) union and intersection.

Timed Muller Automata

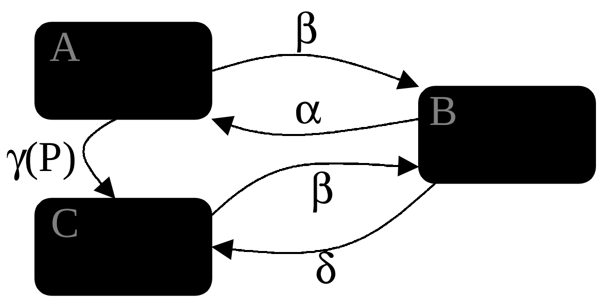
- A timed Muller automaton TMA is a tuple (Σ,Q,Q_0,C,E,F) where (Σ,Q,Q_0,C,E) is a timed transition table and $F \subseteq 2^Q$ specifies an acceptance familiy. A run $r=(s,\nu)$ of TMA over a timed word (σ,τ) is called an accepting run iff $\inf(r) \in F$.

- For a TMA A the language $L(A)$ of timed words it accepts is defined as the set
  $L(A) = \{(\sigma,\tau) \mid A$ has an accepting run over $\sigma,\tau)\}$.

- A timed language is accepted by some timed Büchi automaton iff it is accepted by some timed Muller automaton.

State Transition Diagrams

- State transition: When event γ occurs in state A, if Condition P is true at the time, the system executes action *a* and transfers to state C.
- State diagrams are directed graphs with nodes denoting states, and arrows (labelled with the triggering event, guarding conditions and action to be executed) denoting transitions.
- Example:



- Problem: all combinations of states have to be represented explicitly, leading to exponential blow-up.

State Transition Diagrams

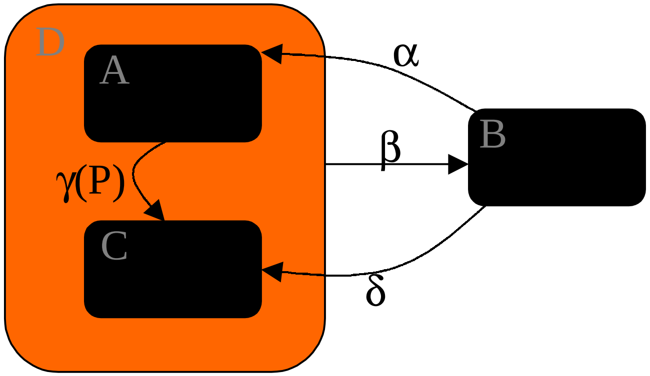
- Disadvantages:
  - No structure (no strategy for bottom-up of top-down development)
  - State-transition diagrams are flat, ie without hierarchy
  - Uneconomical wrt transitions (eg interrupt): exponential blow-up
  - Uneconomical wrt states: exponential blow-up
  - Uneconomical wrt parallel composition: exponential blow-up
  - Inherently sequential; parallelism cannot be expressed in a natural way

Statecharts

- Visual formalism for describing states and transitions of a system in a modular fashion
- Extension of state-transition diagrams:
  - hierarchy
  - concurrency / orthogonality: AND/OR decomposition of states together with inter-level transitions
  - communication, including broadcast for communication between concurrent components
- Can be used as stand-alone behavioral description or as part of a more general design methodology
- statecharts = state diagrams + depth + orthogonality + broadcast communication

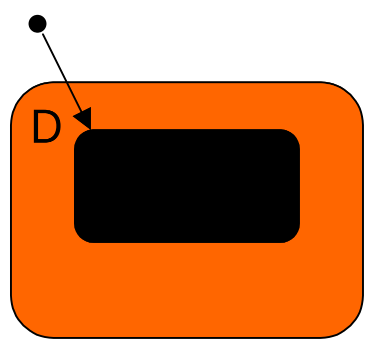
State Levels: Clustering and Refinement

- Rounded rectangles (called boxes): states
- Encapsulation of boxes: hierarchy relation / clustering
- Arrow: transition
- Arrow labels: <event>(<condition>)/<action>
  - All components are optional
  - Syntax for events and conditions is closed under boolean operations *or*, *and* and *not*
- Three types of states: basic states, OR-states, AND-states.



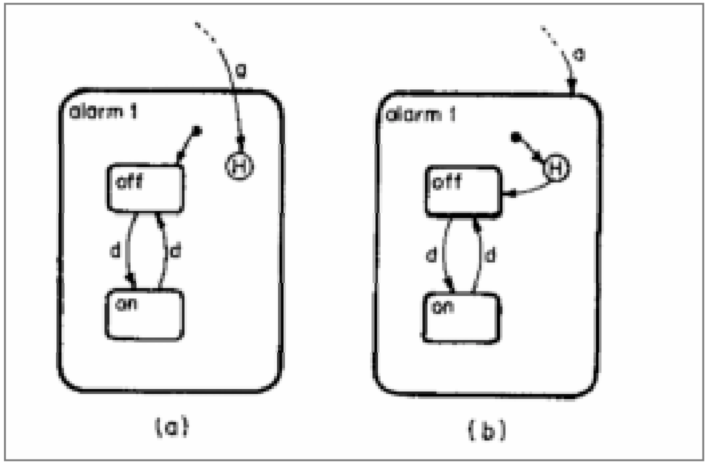
State Levels: Clustering and Refinement

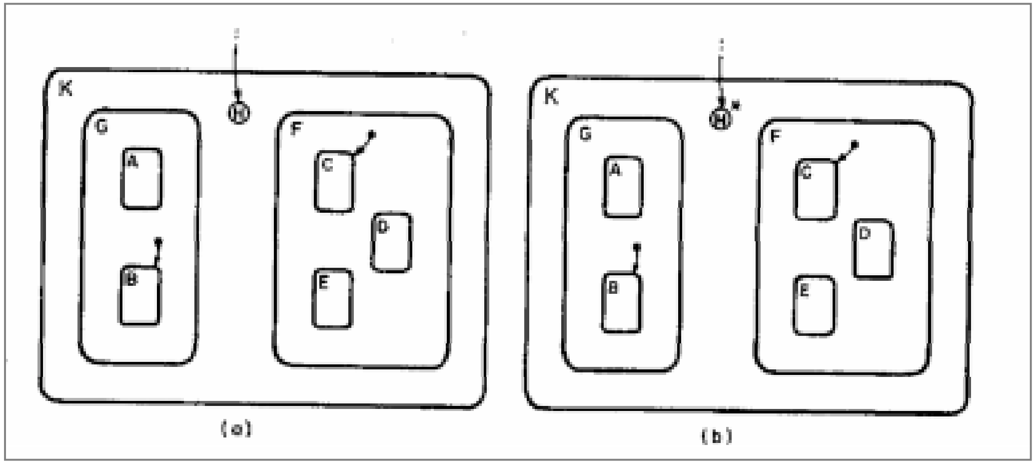
- Semantics of OR-State (D in example): XOR of A and C. To be in state D the system must be either in A or in C, and not in both. D is abstraction of A and C.
- β is a common property of A and C: β leads from them to B.
- Default state: denotes state that is entered when abstraction of set of states is entered.



History

- Enter-by-history: enter the state most recently visited.
  - H: history is applied only on the level in which it appears.
  - H*: history is applied down to the lowest level of states.

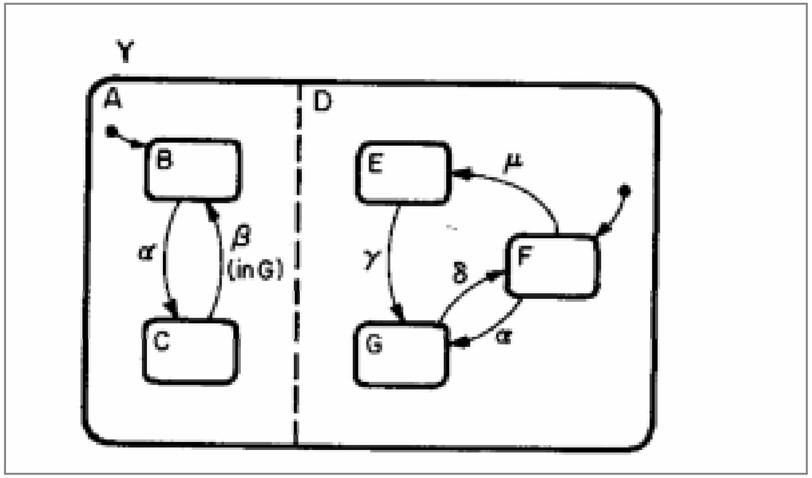




Source [1]

Orthogonality

- AND decomposition: being in a state the system must be in all of its AND components.
- Example: state Y consiting of AND components A and D. Y is orthogonal product of A and D.
- If Y is entered, the system enters the combination (B,F) by the default arrows. If event α occurs, it transfers B to C and F to G simultaneously, resulting in the new combined state (C,G).
  -> Synchronization.

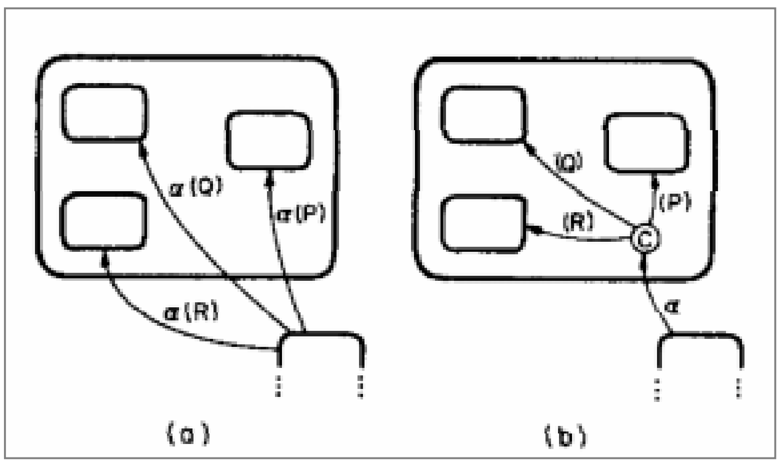


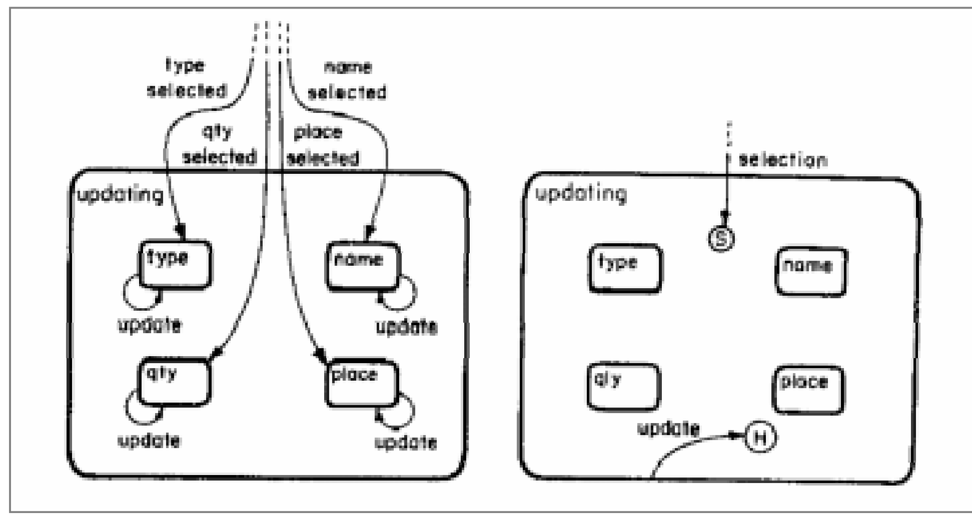
Source [1]

- If μ occurs, it affects only the D components: independence.
- Due to 'in G' conditions the AND decomposition is not necessarily a disjoint product of states since some dependence between components can be introduced.

Conditions and Selection Entrances

- Circled connectives for abbreviating more complex entrances to substates:
  - C: conditional.
  - S: selection. Event is selection of one of a number of clearly defined options chosen to modelled as states.



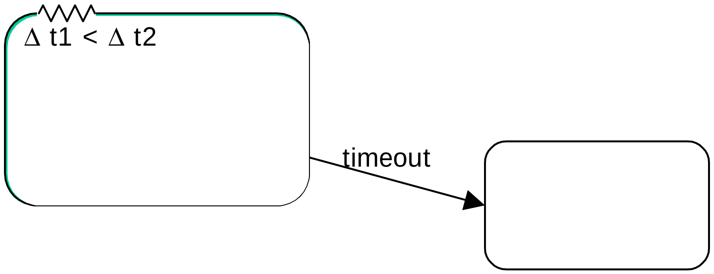


Source [1]

Delays and Timeouts

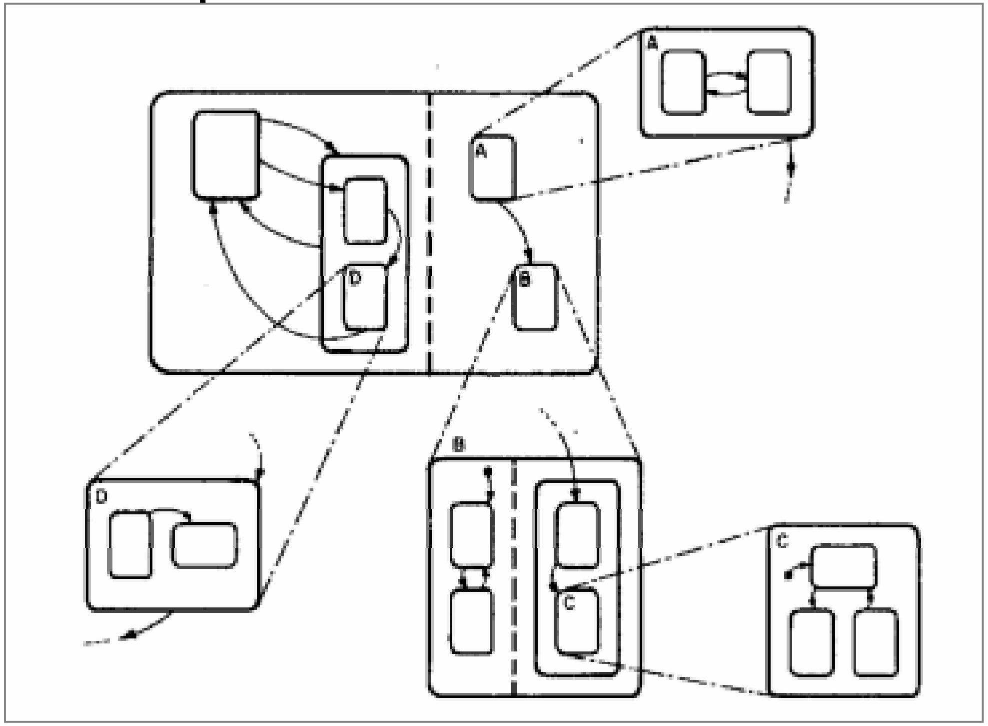
- Event expression *timeout*(*event*, *number*): represents event that occurs precisely when the specified number of time units have elapsed from the occurrence of the specified event *event*.

- Graphical notation for special case *timeout* (*entered state*, *bound*)



Unclustering

- Laying out parts of the startchart not within but outside of their natural naborhood. Useful for manuals and computerized use.
- Example:



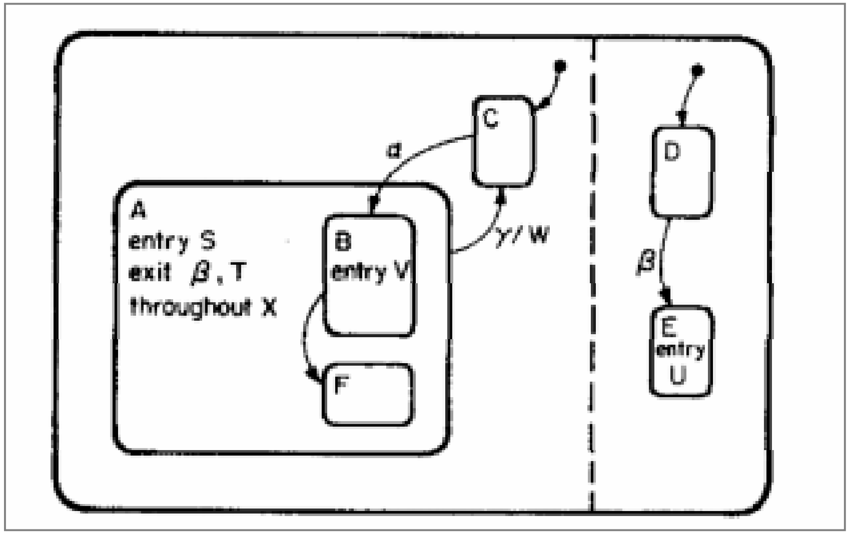
Source [1]

Actions and Activities

- Pure statecharts represents the control part of a system responsible for making time-dependent decisions that influence the system's entire behavior.
- Ability of statecharts to generate events and to change the value of conditions provided by attaching /S to the label of a transition where S is an action carried out by the system.
- Actions: instantaneous occurrences that take ideally zero time. Output in automata-theretic terms.
- Activities: are durable, ie take nonzero amount of time.
- Associated with each activity X:
  - two special kinds of actions to start and stop activities: start(X), stop(X)
  - special condition: active(X)

Actions and Activities

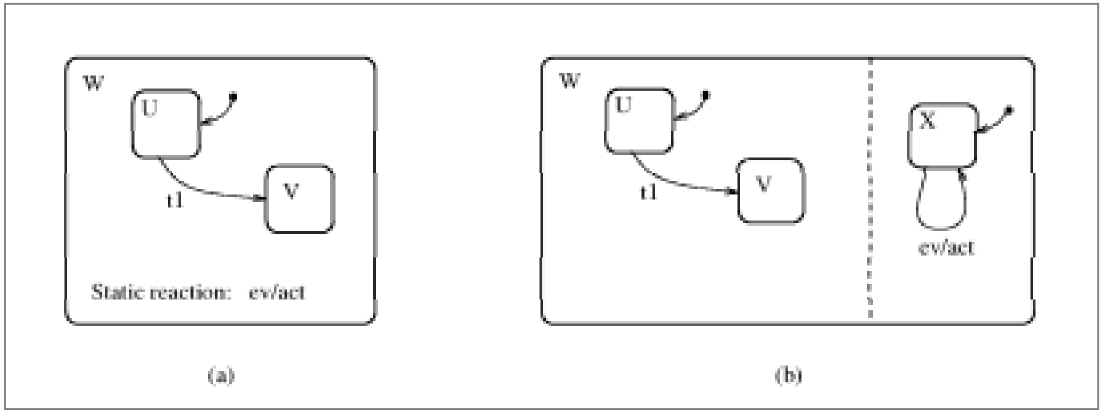
- Actions can be associated with
  - the entrance in and the exit from a state
  - transitions
- Activity X is carried throughout a state A: action start(X) is carried out upon entering A and stop(X) upon leaving A.
- Example:



Source [1]

Static Reactions

- Each state can be associated with static reactions (SRs) to be carried out (whenever) enabled as long as the system is in (and not exiting) the state in question.
- Syntax: e[c]/a
- Semantically: each SR in state S can be regarded as transition in a virtual substate of S that is orthogonal to its ordinary substates and to the other SRs of S.
- Example:



Source [1]

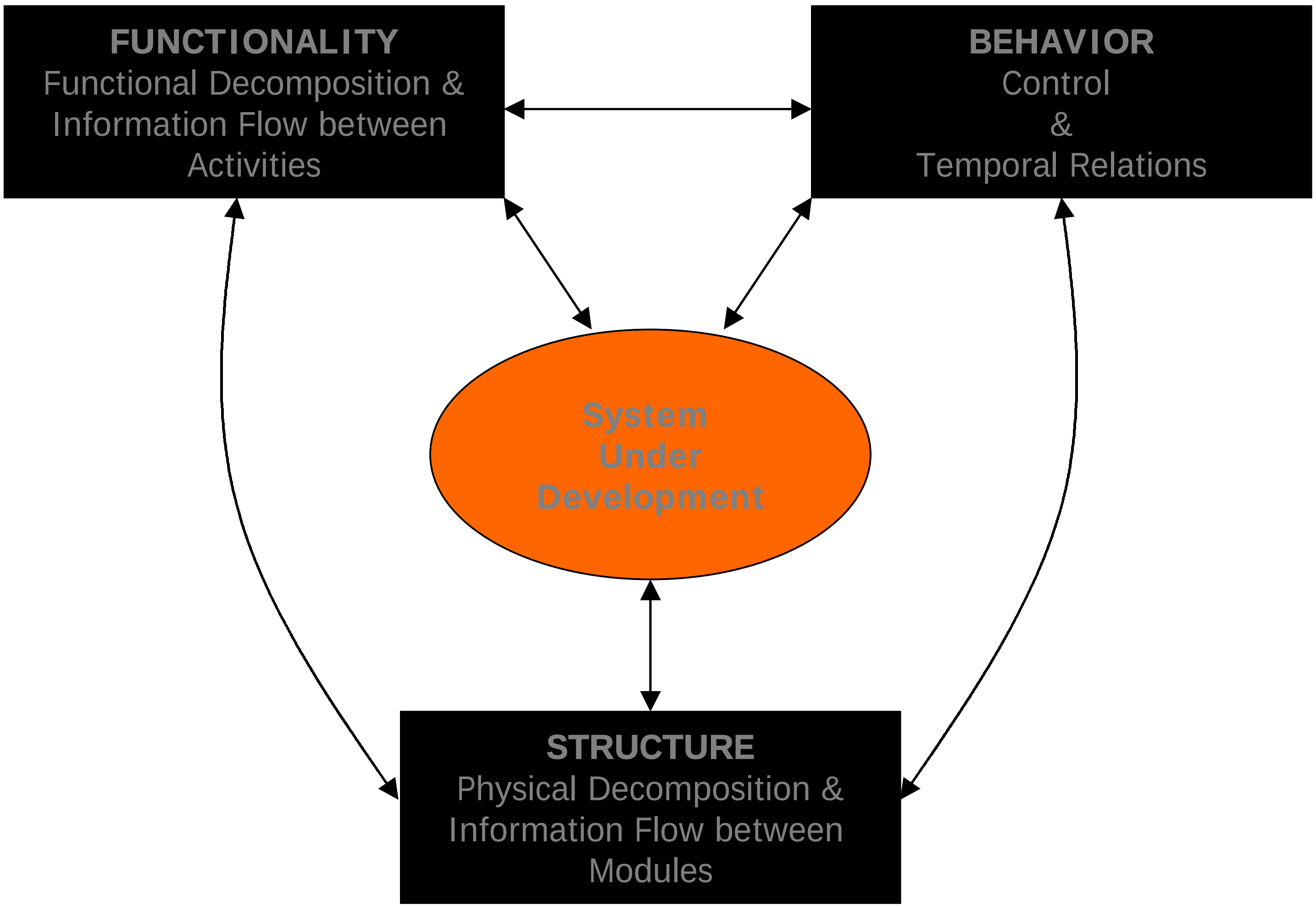
Special Events, Conditions and Actions

	EVENTS	CONDITIONS	ACTIONS
in statechart	entered(S)/en(S) exited(S)/ex(S)	in(S)	clear-history(state) clear-history(state*)
connecting statecharts to activities	started(A) stopped(A)	active(A) hangig(A)	start(A) / st!(A) stop(A) suspend(A) resume(A)
information items	read(D) written(D) true(C) false(C)	D=exp D<exp D>exp ...	D:=exp made_true(C) make_false(C)
time	timeout(E,n)		schedule(Ac,n)/sc!(Ac,d)

The STATEMATE System

- Graphical working environment for the specification, analysis, design and documentation of large and complex reactive systems.
- Three points of views, each covered by own visual formalism:
  - structure: module charts
  - functionality: activity charts
  - behavior: statecharts
- Statecharts used to depict reactive behavior over time.
- Each visual formalism admits a formal semantics that provides each feature, graphical and non-graphical alike with a precise and unambiguous meaning.
- Goal: enabling user to specify a system, and to run, debug and analyze the specifications and designs that result from the graphical languages.

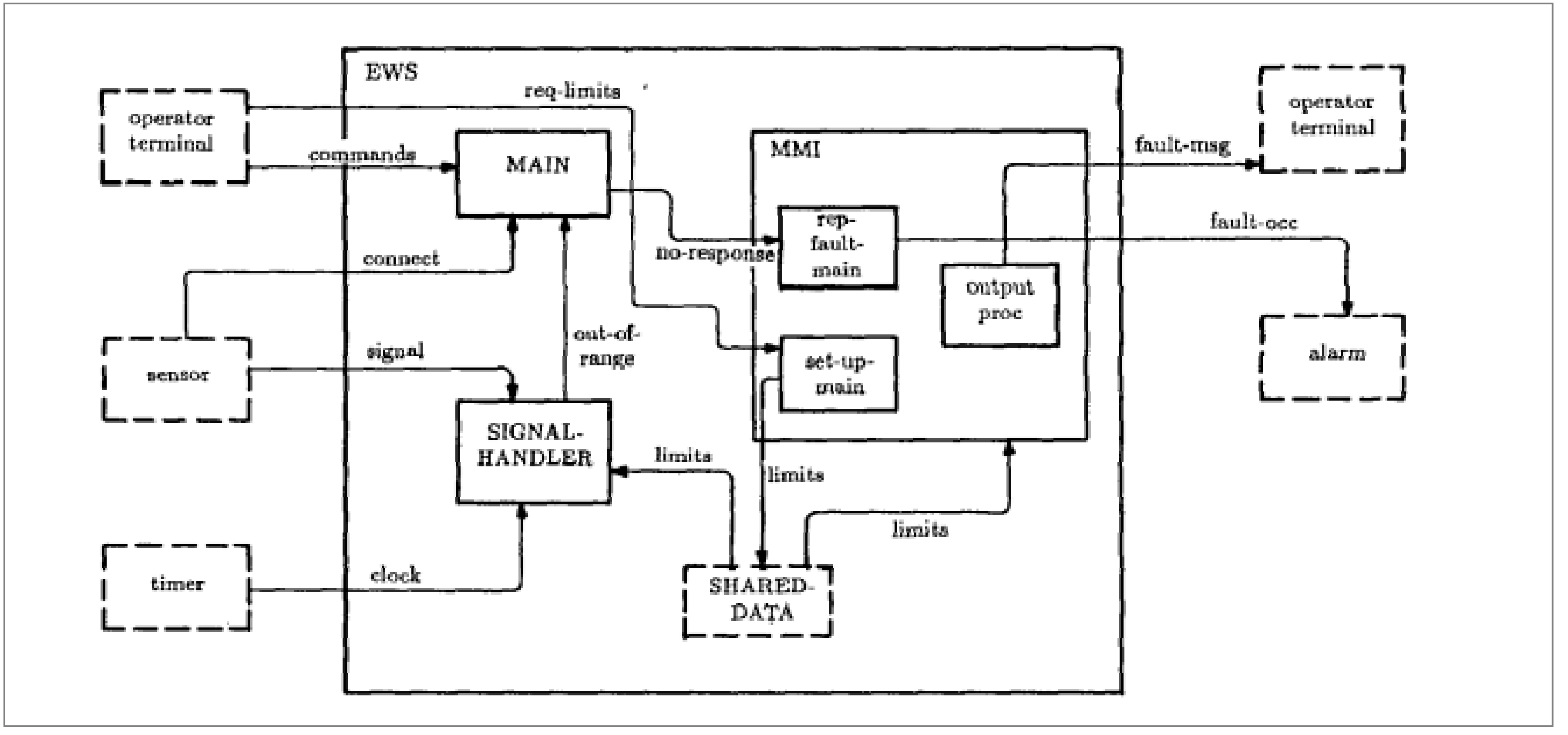
The three Views of a SUD



Structural View

- Provides hierarchical decomposition of SUD into its physical components, called modules (e.g. piece of hardware, or subroutines or blocks of software).
- Identifies information that flows between modules (data and control signals).
- Visual specification language: module-charts
  - Modules: rectilinear shapes
    - Storage modules: dashed sides
    - Envionment modules: dashed-line rectangles external to that of the SUD itself
  - Sub-module relationship: encapsulation
  - Information flow: arrows / hyperarrows

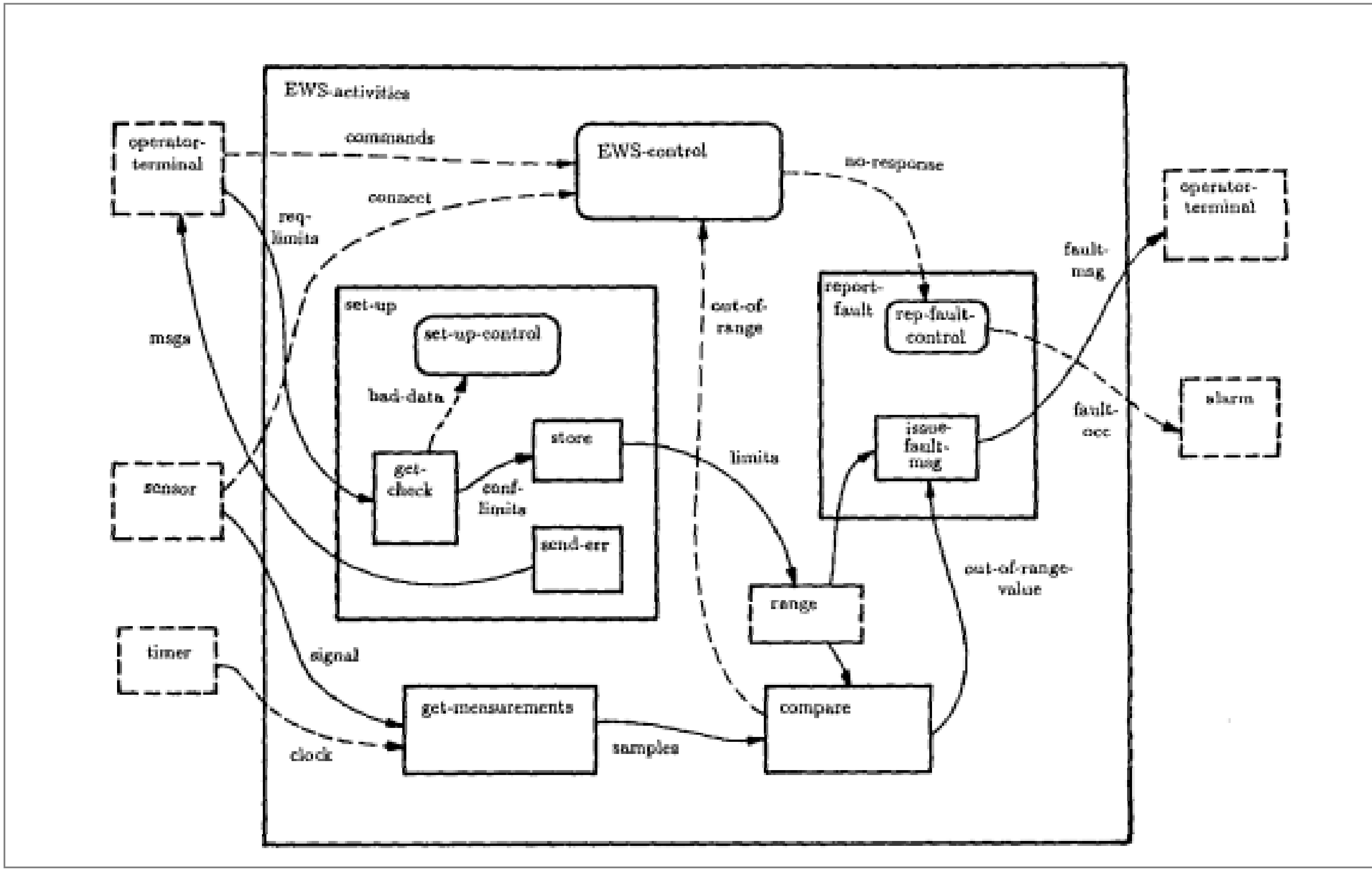
Module-Charts Example (EWS)



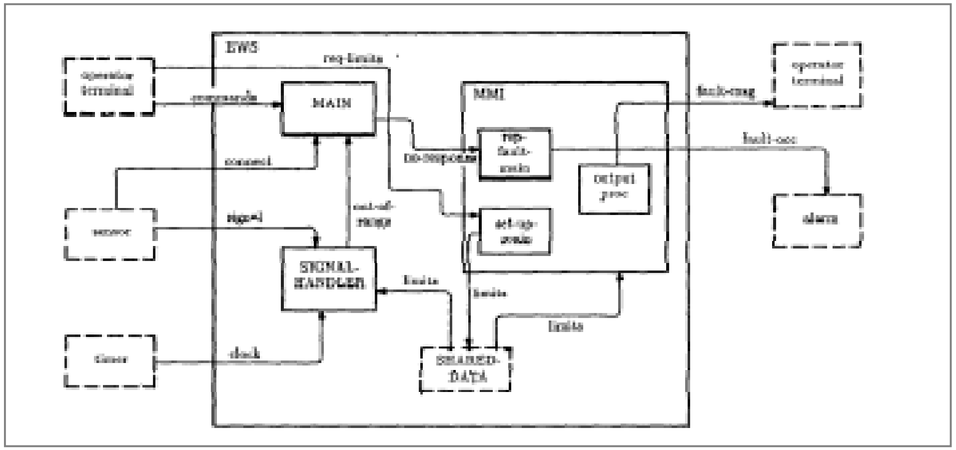
Functional View

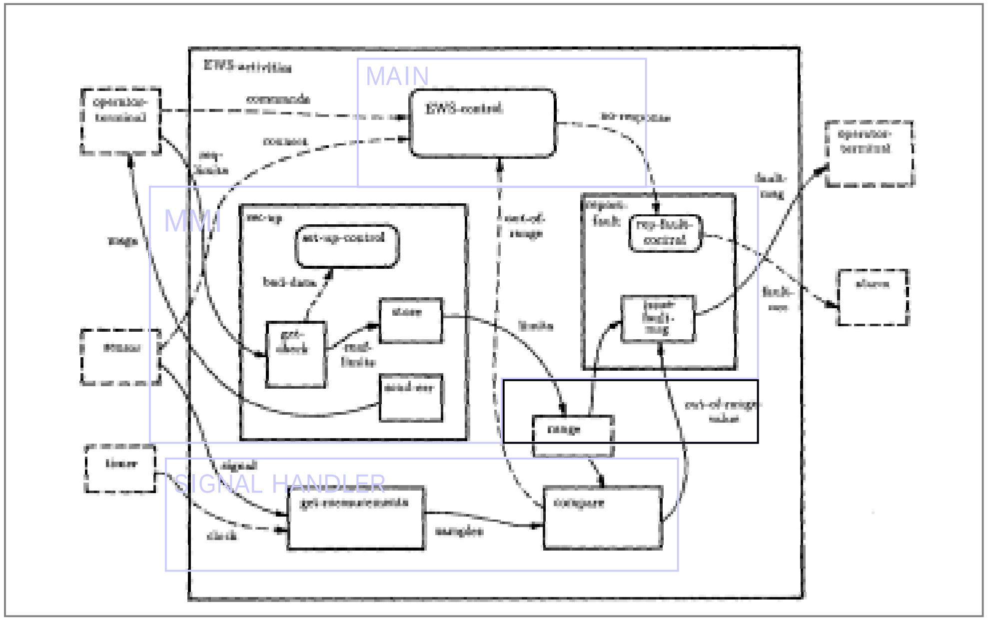
- Identifies a hierarchy of activities with the details of the data items and control signals that flow between them: functional decomposition of the SUD.
- Non-committing semantics, ie. it only asserts that something can happen. No specification of dynamics, eg when activities will be activated, whether/when they terminate, etc.
- Visual specification language: activity-charts:
  - Activities: rectilinear shapes
  - flow of data items: solid lines
  - flow of control items: dashed lines
  - Data-stores: represent databases, buffers, etc.
  - Control activities: behavioral view of the system; appear as empty boxes in the activity-chart (with rounded edges). Their contents are specified by statecharts.

Activity-Charts Example (EWS)



Module and Activity Charts

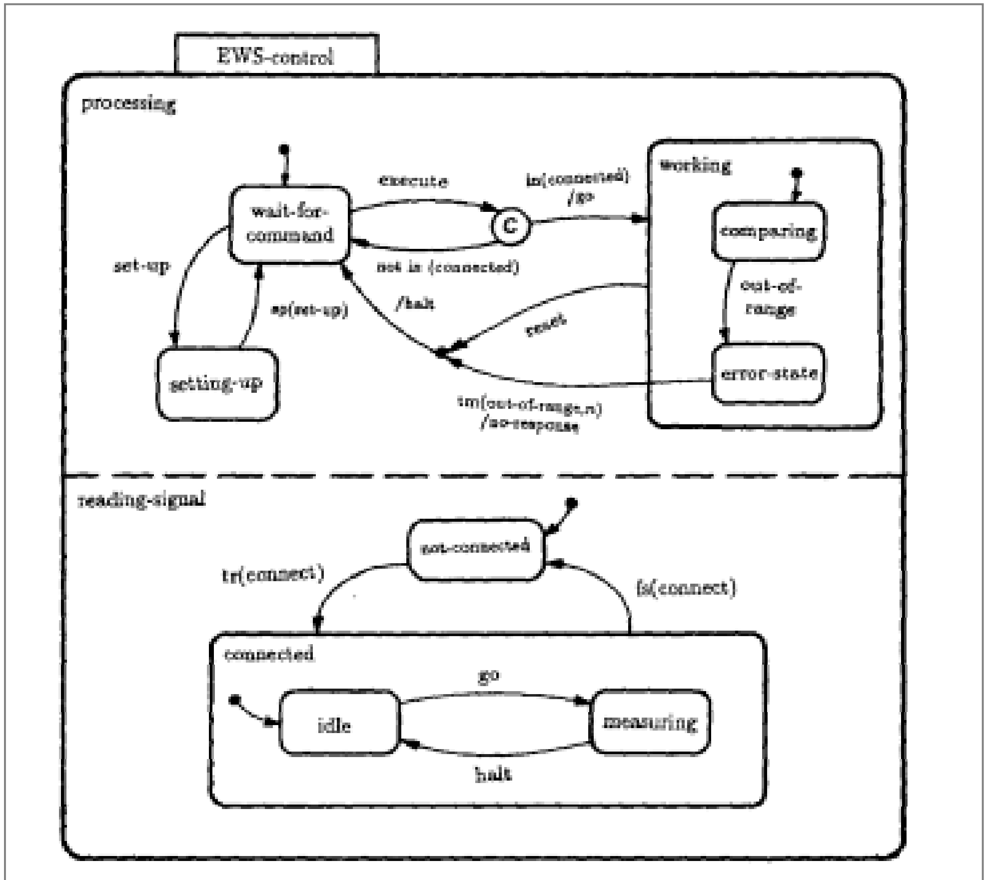




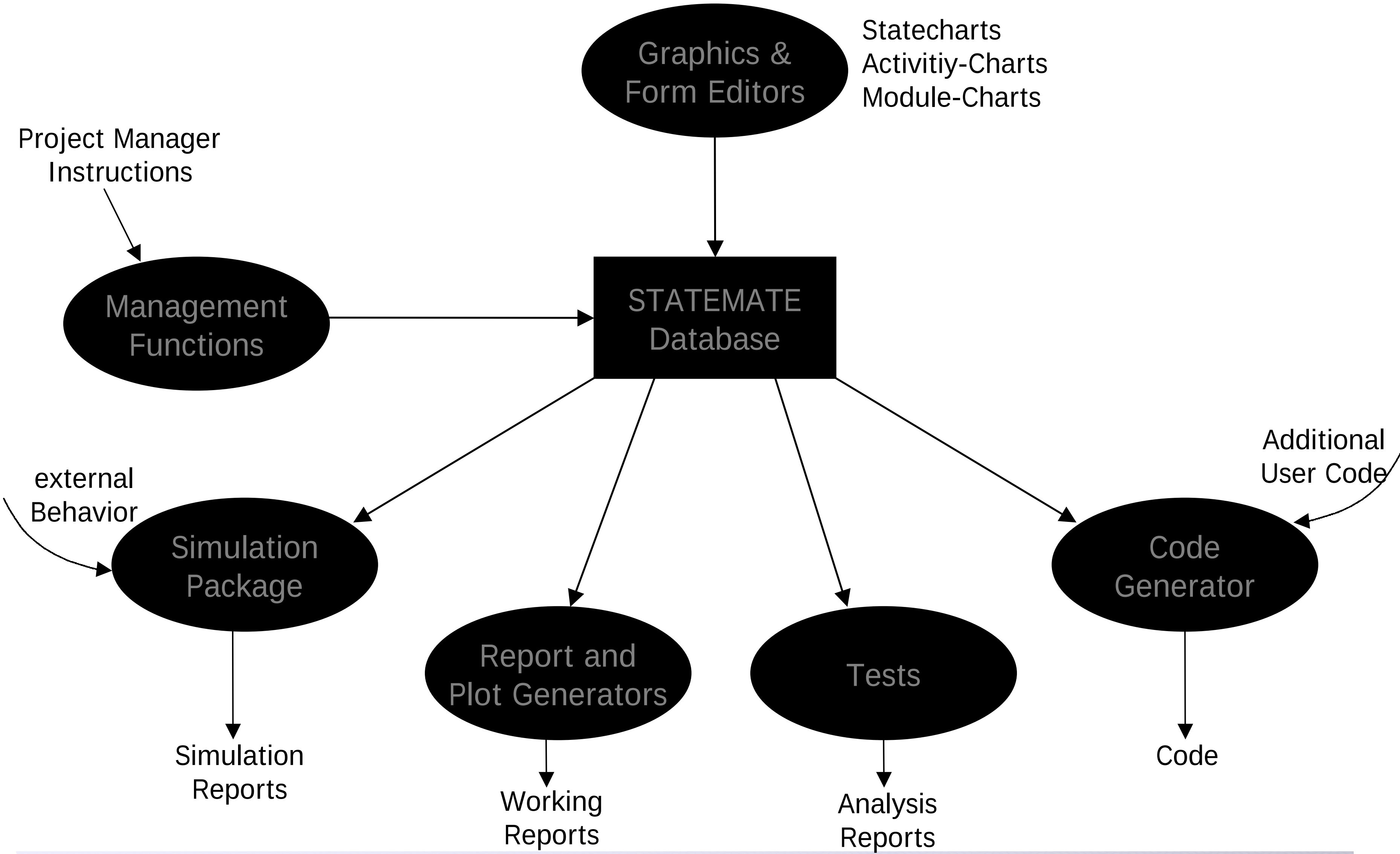
Behavioral View

- Specifies control, ie when, how and why things happen as the SUD reacts over time.
- Visual specification language: statecharts

Statecharts Example (EWS)



Structure of STATEMATE



Advanced Features (1)

- Consistency and completeness tests, static logic tests. Examples:
  - module hierarchy consistent with activity hierarchy?
  - cyclic definitions?
- Object List Generator (OLG): querying the model of the SUD as described by the modelling languages.
- Single step execution:
  - Step: one unit of dynamic behavior
  - At the beginning and end of a step, the SUD is in a legal status.
  - Status: currently active states and activities, current values of variables and conditions, etc.
  - During a step, the environment activities can generate external events, change the truth values of conditions, and update variables and other data items.

Advanced Features (2)

- Batch simulation: carry out many steps in order, controlled by a simulation control program (SCP) written in SCL (Simulation Control Language).
- Breakpoints, simulation reports.
- Dynamic tests, mostly by carrying out exhaustive sets of execution:
  - reachability
  - non-determinism
  - deadlock
  - usage of transitions
- Code generation: specification (parts) can be automatically translated into C, Ada, VHDL, Verilog.