6.2.3 The Basic Causality Model

(OMG, 2011b, 11)

"Causality model" is a specification of how things happen at run time [...]. The causality model is quite straightforward:

• Objects respond to messages that are generated by objects executing communication actions.
• When these messages arrive, the receiving objects eventually respond by executing the behavior that is matched to that message.
• The dispatching method by which a particular behavior is associated with a given message depends on the higher-level formalism used and is not defined in the UML specification (i.e., it is a semantic variation point).

The causality model also subsumes behaviors invoking each other and passing information to each other through arguments to parameters of the invoked behavior, [...]. This purely 'procedural' or 'process' model can be used by itself or in conjunction with the object-oriented model of the previous example.

• Event occurrences are detected, dispatched, and then processed by the state-machine, one at a time.
• The semantics of event occurrence processing is based on the run-to-completion assumption, interpreted as run-to-completion processing.
• Run-to-completion processing means that an event [...], can only be taken from the pool and dispatched if the processing of the previous [...], is fully completed.
• The processing of a single event occurrence by a state machine is known as a run-to-completion step.
• Before commencing on a run-to-completion step, a state machine is in a stable state configuration with all entry/exit/internal-activities (but not necessarily do-activities) completed.
• The same conditions apply after the run-to-completion step is completed.
• Thus, an event occurrence will never be processed [...], in some intermediate and inconsistent situation.
• [IOW,] The run-to-completion step is the passage between two state configurations of the state machine.
• The run-to-completion assumption simplifies the transition function of the StM, since concurrency conflicts are avoided during the processing of event, allowing the StM to safely complete its run-to-completion step.
• The order of dequeuing is not defined, leaving open the possibility of modeling different priority-based schemes.
• Run-to-completion may be implemented in various ways [...].
Our standard distinguishes (among others) an event, i.e., an operation which yields a set of events (i.e., signal instances) ready to be dequeued.

Let \( \text{Ether} \) denote the event, \( \text{D} \) the type of requesting action, the target, the properties of the communication transition.

The order of dequeuing is not defined. Let \( stb_1 \), \( stb_2 \), \( atr_1 \), \( atr_2 \) be the request signals, and \( s_0 = \epsilon, \ldots, s_n = \epsilon \) be the set of states.

The standard distinguishes (among others) two priority-based schemes.

Example 1.12 Schedule the causal chain by.

\[ \text{strong} \]
A (single, global, shared, reliable) FIFO queue is an ether:

\[ \text{Eth} = \big( D(C) \times D(E) \big) \ast \text{the set of finite sequences of pairs } (u, e) \in D(C) \times D(E) \]

- **ready**: \( \text{Eth} \times D(C) \rightarrow 2^{D(E)} \)
  \[ (u_1, e, u_2) \mapsto \begin{cases} \{ (u_1, e) \}, & \text{if } u_1 = u_2 \\ \emptyset, & \text{otherwise} \end{cases} \]

- **⊕**: \( \text{Eth} \times D(C) \times D(E) \rightarrow \text{Eth} \)
  \[ (\varepsilon, u, e) \mapsto \varepsilon \]

- **⊖**: \( \text{Eth} \times D(E) \rightarrow \text{Eth} \)
  \[ (\varepsilon, (u, e_1), e_2) \mapsto \begin{cases} \varepsilon, & \text{if } e_2 = e_1 \\ \varepsilon.(u, e_1) & \text{otherwise} \end{cases} \]

- **·**: \( \text{Eth} \times D(C) \rightarrow \text{Eth} \)
  \[ \text{remove all } (u, e) \text{ from } \varepsilon \]

Other Examples

- One FIFO queue per active object is an ether.
- One-place buffer.
- Priority queue.
- Multi-queues (one per sender).
- Trivial example: sink, "black hole".
- Lossy queue (⊕ needs to become a relation then).

System Configuration

**Definition.** Let \( S_0 = (T_0, C_0, V_0, \text{atr}_0, E_0) \) be a signature with signals, \( D_0 \) a structure of \( S_0 \), \( (\text{Eth}, \text{ready}, \oplus, \ominus, \cdot) \) an ether over \( S_0 \) and \( D_0 \).

Furthermore assume there is one core state machine \( M_C \) per class \( C \in C \).

A system configuration over \( S_0 \), \( D_0 \), and \( \text{Eth} \) is a pair \( (\sigma, \varepsilon) \in \Sigma \times \text{Eth} \) where

- \( S = (T_0 \cup \{ S_M_C \mid C \in C \}, C_0, \dot{V_0} \cup \{ \langle \text{stable}: \text{Bool}, \neg, \text{true}, \emptyset \rangle \} \cup \{ \langle \text{st}_C: S_M_C, +, s_0, \emptyset \rangle \mid C \in C \} \cup \{ \langle \text{params} E: E_0, 1, +, \emptyset \rangle \mid E \in E_0 \}, \dot{E_0}) \)

- \( D = D_0 \cup \{ S_M_C \mapsto S(M_C) \mid C \in C \}\)

- \( \sigma(u)(r) \cap D(E_0) = \emptyset \) for each \( u \in \text{dom}(\sigma) \) and \( r \in V_0 \).

References