Chapter 14: Formal Specification and Enactment

Service-Oriented Computing: Semantics, Processes, Agents
Munindar P. Singh and Michael N. Huhns, Wiley, 2005
Formal Specification and Enactment

Declarative representations based on logic

- Contrast with procedural flow specifications
  - Branch and join primitives
  - Central execution engine

- Capture the essence of what is required
  - Minimally constrain service enactment
  - Accommodate greater efficiencies
  - Facilitate better handling of exceptions and opportunities
  - Support naturally distributed enactment
Temporal Logic

Logic of time

- Based on *significant* events: relevant to interaction
  - Start: \( s \)
  - Commit: \( c \)
  - Abort: \( a \) or rather \( \overline{c} \)

- Declaratively specify *dependencies*, i.e., constraints

- Maximize flexibility: any events that satisfy the stated constraints

- Would support a high-level reasoner
Example Dependencies

- If $T_1$ starts then $T_2$ starts: $\overline{s_1} \lor s_2$
- If air ticket transaction starts then hotel booking transaction starts: $\overline{s_A} \lor s_H$
- If order (O) is canceled and payment (P) is made then refund (R) is initiated:
  $$c_O \lor \overline{s_P} \lor \overline{c_P} \lor s_R$$
- If refund is initiated then payment must previously have been made: $\overline{s_R} \lor c_P \cdot s_R$

Events are the atoms; $\overline{e}$ is the complement of $e$; and the dot operator $\cdot$ indicates temporal order.
Specification Syntax

- The center dot (·) orders events
- Complementation means hard opposite: commit versus abort
  - Used in specifications
- Negation means soft opposite: commit versus not commit
  - Not used in specifications

$L_1$. $l \rightarrow dep | dep \land l \llbracket \text{interleaving} \rrbracket$
$L_2$. $dep \rightarrow seq | seq \lor dep \llbracket \text{choice} \rrbracket$
$L_3$. $seq \rightarrow bool | \text{event} | \text{event} \cdot \text{event} \llbracket \text{ordering} \rrbracket$
$L_4$. $bool \rightarrow 0 | \top$
Specification Semantics

Identify the desirable “runs” or computations

- Universe consists of *legal* runs:
  - Event instances and their complements are mutually exclusive
  - Event instances don’t repeat (transaction identifiers can ensure uniqueness)

\[ M_1. \quad \tau \models e \iff (\exists i : \tau_i = e) \]
\[ M_2. \quad \tau \models I_1 \lor I_2 \iff \tau \models I_1 \text{ or } \tau \models I_2 \]
\[ M_3. \quad \tau \models I_1 \land I_2 \iff \tau \models I_1 \text{ and } \tau \models I_2 \]
\[ M_4. \quad \tau \models I_1 \cdot I_2 \iff (\exists i : \tau[0,i] \models I_1 \text{ and } \tau[i+1,|\tau|] \models I_2) \]
Example Coordination Relationships

- \( D_\prec = \overline{e} \lor \overline{f} \lor e \cdot f \)
  - If both \( e \) and \( f \) occur, then \( e \) precedes \( f \)
  - If \( e \) and \( f \) occur on \( \tau \), neither \( \overline{e} \) nor \( \overline{f} \) can occur on \( \tau \). So \( \tau \) must satisfy \( e \cdot f \), which means that an initial part of \( \tau \) satisfies \( e \) and the remainder satisfies \( f \)

- \( (\overline{e} \lor f \lor g) \land (\overline{g} \lor e) \land (\overline{g} \lor \overline{f}) \)
  - If \( e \) happens and \( f \) does not, then and only then do \( g \)
  - Typical with data updates, where \( g \) restores consistency (potentially) violated by the success of \( e \) and the failure of \( f \)
Enactment

Control execution of tasks to meet satisfy specified dependencies

- **Allow, delay, deny, or trigger events**
  - Any realized run is in the denotation of each dependency

- System state = the runs allowed
  - Initially, given by the stated dependencies
  - Narrows down as events occur

- Key requirements
  - Maximal set of allowed runs (flexibility)
  - Compute symbolically and modularly
Residuation

E₁. \( 0/e \doteq 0 \)
E₂. \( \top/e \doteq \top \)
E₃. \( (D \land F)/e \doteq (D/e \land F/e) \)
E₄. \( (D \lor F)/e \doteq (D/e \lor F/e) \)
E₅. \( e/e \doteq \top \)
E₆. \( \overline{e}/e \doteq 0 \)
E₇. \( (e \cdot f)/e \doteq f \)
E₈. \( (\overline{e} \cdot f)/e \doteq 0 \)
E₉. \( (d \cdot e)/e \doteq 0 \)
E₁₀. \( (d \cdot \overline{e})/e \doteq 0 \)
E₁₁. \( (d \cdot f)/e \doteq d \cdot f \)
E₁₂. \( d/e \doteq d \)

The above rules apply if we swap \( e \) and \( \overline{e} \)
Example of Residuation

\[ D_\prec = \overline{e} \lor \overline{f} \lor e \cdot f \]

Figure 1: Scheduler states and transitions for \( D_\prec \)
Distributed Enactment

- Constrain autonomy based only on dependencies
  - Local decisions
- Place a *guard* on each event
  - When true, the event can safely happen
  - Modified as relevant events occur (messages arrive)

- Challenges
  - Representing them
  - Reasoning with them in a distributed manner
Guard Syntax

Enables stating whether an event can occur *now*

\[ L_5. \quad T \rightarrow \text{conj} \mid \text{conj} \land T \]
\[ L_6. \quad \text{conj} \rightarrow \text{disj} \mid \text{disj} \lor \text{conj} \]
\[ L_7. \quad \text{disj} \rightarrow \text{bool} \mid \Box \text{seq} \mid \Diamond \text{seq} \mid \neg \text{event} \]

- Events are *stable* or durable
- \( \Box e \) means \( e \) has occurred
- \( \Diamond e \) means \( e \) has occurred or will occur eventually
- \( \neg e \) means \( e \) has *not yet* occurred
Guard Semantics

- Universe consists of maximal runs (either an event or its complement occurs)

\[ M_5. \ u \models_k E \text{ iff } u \models_{0,k} E \]
\[ M_6. \ u \models_{i,k} f \text{ iff } (\exists j : i \leq j \leq k \text{ and } u_j = f) \]
\[ M_7. \ u \models_{i,k} E \lor F \text{ iff } u \models_{i,k} E \text{ or } u \models_{i,k} F \]
\[ M_8. \ u \models_{i,k} E \land F \text{ iff } u \models_{i,k} E \text{ and } u \models_{i,k} F \]
\[ M_9. \ u \models_{i,k} E \cdot F \text{ iff } (\exists j : i \leq j \leq k \text{ and } u \models_{i,j} E \text{ and } u \models_{j+1,k} F) \]
\[ M_{10}. \ u \models_{i,k} \top \]
\[ M_{11}. \ u \models_{i,k} \neg E \text{ iff } u \not\models_{i,k} E \]
\[ M_{12}. \ u \models_{i,k} \square E \text{ iff } (\forall j : k \leq j \Rightarrow u \models_{i,j} E) \]
\[ M_{13}. \ u \models_{i,k} \Diamond E \text{ iff } (\exists j : k \leq j \text{ and } u \models_{i,j} E) \]
Guards for $D_\prec = \overline{e} \lor \overline{f} \lor e \cdot f$

- $\neg e \land \neg \overline{e} \land \neg f \land \neg \overline{f}$

- $G_b(D_\prec, e) = (\neg f \land \neg \overline{f} \land \Diamond(\overline{f} \lor f)) \lor (\Box \overline{f} \land \top) = \neg f \lor \Box \overline{f} = \neg f$

- $G_b(D_\prec, \overline{e}) = \top$

- $G_b(D_\prec, \overline{f}) = \top$

- $G_b(D_\prec, f) = (\neg e \land \neg \overline{e} \land \Diamond \overline{e}) \lor \Box e \lor \Box \overline{e} \cong \Diamond \overline{e} \lor \Box e$
Scheduling with Guards: Example

- If \( e \) is attempted first
  - \( G(e) = \top \): \( e \) executes and notifies
  - Notification \( \Box e \) changes
    \( G(f) = \Diamond \overline{e} \lor \Box e = \top \), enabling \( f \)

- If \( f \) is attempted first
  - \( G(f) = (\Diamond \overline{e} \lor \Box e) \neq \top \), so it waits
  - Notification of \( \Box \overline{e} \) or \( \Box e \) changes \( G(f) \) to \( \top \), thus enabling \( f \)

- \( G(\overline{e}) = \top \) and \( G(\overline{f}) = \top \), so they can happen any time
Motivations for Formalization

- Proving correctness when
  - Guards are created by compiling the dependencies
  - Guards are preprocessed
  - Events are executed and guards updated

- Justifying improvements in efficiency
  - Simplifying guards prior to execution
  - Updating guards incrementally
  - Skipping some steps
Formalization Sketch: 1

- Evaluation strategy: a function that captures
  - Evolution of guards
  - Execution of events
- An evaluation strategy *generates* a run $u$ if
  - For each event $e$ that occurs on $u$,
  - $u$ satisfies $e$’s current guard due to the strategy
  - At the index *preceding* $e$’s occurrence
- Generation is more abstract than execution:
  - A true guard may involve ◇ expressions
Formalization Sketch: 2

- Begin with trivial strategy
  - Easily correct, but useless
- Replace with better strategies
  - Symbolically calculate guards from dependencies
  - Safely discard certain terms
  - Process messages symbolically
Symbolically Calculating Guards

- $G(0, e) \triangleq 0$
- $G(\top, e) \triangleq \top$
- $G(D \lor F, e) \triangleq G(D, e) \lor G(F, e)$
- $G(D \land F, e) \triangleq G(D, e) \land G(F, e)$
- $G(e, e) \triangleq \top$
- $G(\overline{e}, e) \triangleq 0$
- $G(d \cdot e, e) \triangleq \Box d$
- $G(d \cdot \overline{e}, e) \triangleq 0$
- $G(e \cdot f, e) \triangleq \neg f \land \Diamond f$
- $G(\overline{e} \cdot f, e) \triangleq 0$
- $G(d, e) \triangleq \Diamond d$
- $G(d \cdot f, e) \triangleq \Diamond (d \cdot f)$

The above rules apply if we swap $e$ and $\overline{e}$
Calculating Guards: Example

For $D_\prec = \overline{e} \lor \overline{f} \lor e \cdot f$:

- $G(D_\prec, e) = (\Diamond \overline{f} \lor (\neg f \land \Diamond f)) \cong \neg f$
- $G(D_\prec, \overline{e}) = \top$
- $G(D_\prec, f) = \Diamond \overline{e} \lor \Box e$
- $G(D_\prec, \overline{f}) = \top$
Assimilating Messages

Old: $G$

<table>
<thead>
<tr>
<th>Old</th>
<th>Message: $M$</th>
<th>New: $G \div M$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$G_1 \lor G_2$</td>
<td>$M$</td>
<td>$G_1 \div M \lor G_2 \div M$</td>
</tr>
<tr>
<td>$G_1 \land G_2$</td>
<td>$M$</td>
<td>$G_1 \div M \land G_2 \div M$</td>
</tr>
<tr>
<td>$\Box e$</td>
<td>$\Box e$</td>
<td>$\top$</td>
</tr>
<tr>
<td>$\Diamond e$</td>
<td>$\Box e \lor \Diamond e$</td>
<td>$\top$</td>
</tr>
<tr>
<td>$\Box \neg e \lor \Diamond \neg e$</td>
<td>$\Box e \lor \Diamond e$</td>
<td>$0$</td>
</tr>
<tr>
<td>$\Diamond (e \cdot f)$</td>
<td>$\Box e$</td>
<td>$\Diamond f$</td>
</tr>
<tr>
<td>$\Diamond (e \cdot f)$</td>
<td>$\Diamond (e \cdot f)$</td>
<td>$\top$</td>
</tr>
<tr>
<td>$\Diamond (e \cdot f)$</td>
<td>$\Box (f \cdot e) \lor \Diamond (f \cdot e) \lor \Box \neg e_i \lor \Diamond \neg e_i$</td>
<td>$0$</td>
</tr>
<tr>
<td>$\neg e$</td>
<td>$\Box e$</td>
<td>$0$</td>
</tr>
<tr>
<td>$\neg \neg e$</td>
<td>$\Box e \lor \Diamond e$</td>
<td>$\top$</td>
</tr>
<tr>
<td>$G$</td>
<td>$M$</td>
<td>$G$, otherwise</td>
</tr>
</tbody>
</table>
Event Classes

- **Flexible**, agent can delay or omit
- **Inevitable**, agent can delay but not omit
- **Immediate**, agent will neither delay nor omit

\[
D'_< = \overline{e} \cdot \overline{f} \lor \overline{f} \lor \overline{e} \cdot f
\]

\[
D = \overline{e} \lor \overline{f} \cdot e
\]

- \(e\) is inevitable
- \(D'_< = 0\)

- \(e\) is immediate
- \(D' = 0\)
Summary

- Generic approach to describe processes and extended transactions
  - Hides low-level details
  - Combines declarative specifications and operational decision procedures

- Directions
  - Refining methodologies, based on assessment of scenarios
  - Accommodating richer heuristics for distributed evaluations