Coping with Mismatched Semantics of Dependencies in Workflow Applications

Jian Tang

Department of Computer Science
Memorial University of Newfoundland
St. John's, Newfoundland, A1B 3X5 Canada
jian@cs.mun.ca

San-Yih Hwang

Department of Information Management
National Sun Yat-Sen University
Kaohsiung, 80424, Taiwan
syhwang@mis.nsysu.edu.tw

1 Outline of the Presentation

1. Basic Concepts: workflow, tasks and dependencies$\cdots$
2. A Workflow Model
3. More on Dependencies
4. Semantic Mismatch in HAD Environment
5. An Approach
6. Examples
7. Conclusion

2 Basic Concepts

- Workflow - a process involving possibly a large number of individual tasks
- Individual Task - a black box which consists of externally observable states (*not-executing, executing, prepare, commit, abort*$\cdots$).
- State Dependencies among tasks - constraining the task's entering into a particular state (e.g, t_1 AOC t_2, t_1 BFC t_2).
- Other Components - other dependencies, real time requirements, acceptable states, consistency, (forward) recovery$\cdots$ (not the focus of the paper).

3 A Workflow Model - three level hierarchical view

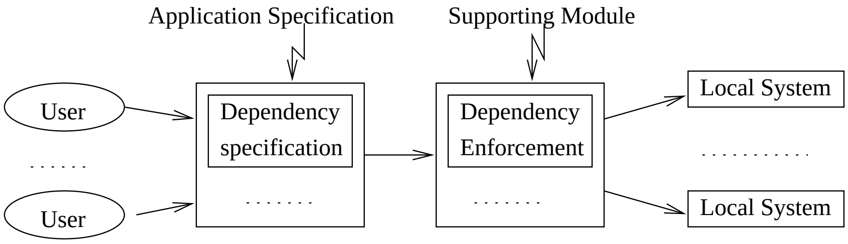

Figure 1. A conceptual view of system architecture.

1. Specification: tasks, states, control flow (dependencies), data flow $\cdots$

2. Supporting Module: coordinates the task execution to meet the specified constraints (e.g., dependencies).

3. local systems: where the tasks are actually executed. Local systems may be heterogeneous and autonomous.

- Significant events: its occurrence signifies the entering of a particular state [Klein91]

  Example:$st(t)$ signifies t enters *executing*, $done(t)$ signifies t enters *prepare*.

- Expressing state dependencies in terms of events

  – Two primitive dependencies on events:

    1. existential dependency: $e_1 \rightarrow e_2$
    2. order dependency: $e_1 < e_2$

  – State dependencies can be expressed by the primitive dependencies. Example:

    1. t_1 AOC $t_2 \equiv cm(t_1) \rightarrow ab(t_2)$;
    2. if both t_1 and t_2 abort, then t_3 must start: $ab(t_1) \rightarrow (ab(t_2) \rightarrow st(t_3))$

- *Attributes* of an event: forcibility, rejectability
- The impact of attributes of events on the *enforcibility* of dependencies

  Example: in general $cm(t_1) \rightarrow ab(t_2)$ enforcible but $ab(t_1) \rightarrow cm(t_2)$ not.
- A dependency not being enforcible implies the workflow semantics cannot always be implemented
- Implication: all dependencies in the specification must be enforcible

4 Semantic Mismatch in HAD environment

- Different views by specification and supporting module: a dependency may be enforcible when viewed by the specification but not when viewed by supporting module

  Example: $cm(t_1) \rightarrow ab(t_2)$

  Specification: $cm(t_1)$ rejectable, $ab(t_2)$ forcible (by common sense)

  Supporting module: $cm(t_1)$ not rejectable, $ab(t_2)$ not forcible (by local autonomy)

- Cause

  1. Specification is done independently of implementation
  2. Existence of the autonomy of local system
  3. As a result, the event attributes may be determined differently at the two levels

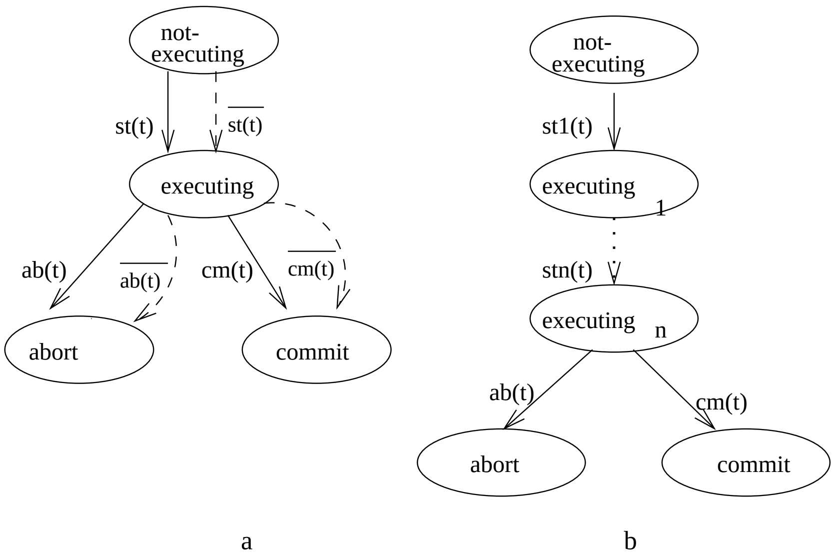

Figure 2. State transitions viewed by a). application and b). supporting module.
The labels attached to edges denote events.

(Semantic mismatch - continued)

4. Two levels may also observe different states of the same task (Figure 2).

5 An approach - dependency conversion

- idea: if D_1 is not enforcible (from the viewpoint of supporting module), we try to substitute D_2 for D_1 if

  1. D_2 is more likely to be enforcible
  2. Enforcing D_2 implies enforcing D_1 as seen by the application

- How to find D_2? (D_2 is called *dominating* dependency of D_1): by event substitution

(dependency conversion - continued)

- *semantically equivalent* events (SE): e_1 SE e_2 if their occurrences have the same effects from the application point of view (e.g, $ab(t)$ SE $\overline{st(t)}$).
- Two types of events in any particular dependency D
  - E-event: forcibility contributes to the enforcibility of D
  - V-event: rejectability contributes to the enforcibility of D
- Example 1: for $e_1 \rightarrow e_2$, e_1 is a V-event and e_2 is an E-event
- Example 2: for $e_1 < e_2$, both e_1 and e_2 are V-events
- E-events and V-events in a general dependency (refer to the paper)

(dependency conversion - continued)

- Event substitution rules: Let e be an event in (supposedly not enforcible) dependency D. Substitute e' for e if

  1. e' is semantically equivalent to e and e is an E-event in D or
  2. $\overline{e'}$ is semantically equivalent to $\overline{e}$ and e is a V-event

- observation: after substitution, the new dependency can be considered to imply the old one

  – Example: consider $D = cm(t_1) \rightarrow ab(t_2)$
    we have $st(t_1) \rightarrow ab(t_2) \Rightarrow D$ if $\overline{st(t_1)}$ SE $\overline{cm(t_1)}$
    and
    $cm(t_1) \rightarrow \overline{st(t_2)} \Rightarrow D$ if $\overline{st(t_2)}$ SE $ab(t_2)$

Algorithm (outline)

1. Given a dependency D;
2. If Enforcible(D) then done, otherwise
3. if D can be converted by applying the above rules, then goto step 2, otherwise signify conversion failure; (i.e., workflow must be remodeled.)

Example: if $cm(t_1) \rightarrow ab(t_2)$ is not enforcible, convert it to $st(t_1) \rightarrow ab(t_2)$. If it is still not enforcible, convert it to $st(t_1) \rightarrow \overline{st(t_2)}$

Function Enforcible(D) is based on:

Theorem 2: Let D_1 and D_2 be two dependencies.

1. if either D_1 is violatable or D_2 is enforcible

t hen $D_1 \rightarrow D_2$ is enforcible;

2. if either D_1 or D_2 is violatable, then

$D_1 < D_2$ is enforcible;

3. if D_1 is violatable, then $\overline{D_1}$ is enfor cible;

Theorem 3: Let D_1 and D_2 be two dependencies.

1. if D_1 is enforcible and D_2 is violatable,

then $D_1 \rightarrow D_2$ is violatable;

2. if D_1 is enforcible, then $\overline{D_1}$ is viola table;

6 Conclusion

1. Semantic mismatch occurs in workflow system
   - Independent modeling and implementation processes
   - They may envision different attributes of the same events
   - The inconsistency may occur in the enforcibility of dependencies
2. Resolving semantic mismatch by dependency conversion
   (a) event equivalence
   (b) event substitution
3. Algorithm
4. Further topics
   (a) If a dependency has more than one dominating dependency, how to choose a better one
   (b) Dominating dependencies usually imply worse performance, how to cope with this problem?